الجمهوريـــة الجزائريـــة الديمقراطيـــة الشعبيـــة

People's Democratic Republic of Algeria

والبـحـث الـعـلـمــي وزارة الـتـعـلـيـم الـعـالي

Ministry of Higher Education and Scientific Research

MUSTAPHA Stambouli University

Mascara

جامعة مصطفى اسطمبولي

معسكر

Faculty of Exact Sciences

Computer Science department

كلية العلوم الدقيقة

قسم الإعلام الألي

3rd cycle DOCTORATE THESIS

Sector: Computer Science

Specialty: Information and Communication Technologies

Title

Deep learning for emotion recognition

Presented by: Mohamed Amine Mahmoudi

On 07/07/2022 at 10:30 a.m.

In front of the jury:

President	Boudjelal Meftah	Professor	Univesity of Mascara
Examiner	Debakla Mohammed	MCA	Univesity of Mascara
Examiner	Mansouri Dou El Kefl	MCA	Univesity of Tiaret
Supervisor	Boufera Fatma	MCA	Univesity of Mascara
Co-Supervisor	Chetouani Aladine	Associate Professor	Univesity of Orléans
Guest	Tabia Hedi	Professor	Univesity of Paris Saclay

College year: 2021/2022

الجمهوريــة الجزائريــة الديمقراطيــة الشعبيــة

République Algérienne Démocratique et Populaire

وزارة الـتعلـيـم الـعـالي والبـحـث الـعلـمـي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université MUSTAPHA Stambouli

Mascara

جامعة مصطفى اسطمبولي

معسكر

Faculté des Sciences Exactes

Département d'informatique

كلية العلوم الدقيقة

قسم الإعلام الألي

THESE de DOCTORAT de 3ème cycle

Filière : Informatique

Spécialité : Technologies des informations et des communications

Intitulée

Apprentissage profond pour la reconnaissance d'émotions

Présentée par : **Mohamed Amine Mahmoudi**

Le 07/07/2022 à 10 H 30

Devant le jury :

Président	Boudjelal Meftah	Professeur	Université de Mascara
Examinateur	Debakla Mohammed	MCA	Université de Mascara
Examinateur	Mansouri Dou El Kefl	MCA	Université de Tiaret
Encadreur	Boufera Fatma	MCA	Université de Mascara
Co-Encadreur	Chetouani Aladine	Maitre de Conférences HDR	Université d'Orléans
Invité	Tabia Hedi	Professeur	Université de Paris Saclay

Année Universitaire : 2021/2022

To all who believe in success through hard work and sacrifice.

Acknowledgements

I thank Allah the Almighty for giving me strength and courage to complete this work. I wish to express my sincerest gratitude to my advisors:

- Ms Fatma Boufera professor at Mustapha Stambouli University of Mascara, Algeria,
- Mr Aladine Chetouani Associate Professor at Univesity of Orléans, France,
- Mr Hedi Tabia Professor at Univesity of Paris Saclay, France,

who have provided much encouragement and freedom for my research. I also thank them for their unconditional help in supervising me. They were always kindly sharing deep knowledge in the field which inspired my research. Their attitudes towards research no doubt set a good example of what a true researcher should be.

I would like to thank the members of the jury composed of:

- Mr Boudjelal Meftah professor at Mustapha Stambouli University of Mascara, Algeria,
- Mr Debakla Mohammed MCA at Mustapha Stambouli University of Mascara, Algeria,
- Mr Mansouri Dou El Kefl lecturer at the Ibn Khladoun University of Tiaret, Algeria,

for taking the time to read and correct my thesis. Their remarks will undoubtedly improve the quality of the manuscript and will constitute a rich experience for my future research.

Last but not least, I must thank my beloved parents, as well as other family members, without whose unconditional love, support and understanding, this work would not have been possible.

Résumé

La reconnaissance de l'expression faciale (FER) est un domaine de recherche qui consiste à classer les émotions humaines à travers les expressions de leur visage comme l'une des sept émotions de base : bonheur, tristesse, peur, dégoût, colère, surprise et neutre. FER trouve des applications dans différents domaines, notamment la sécurité, l'interaction homme-machine intelligente, la robotique et la médecine clinique pour l'autisme, la dépression, la douleur et les problèmes de santé mentale. FER est un problème très difficile en raison des différences subtiles qui existent entre ses catégories. En effet, la différence dans les catégories d'expressions faciales repose sur de petites zones subtiles dans les images faciales comme la bouche, les sourcils. Ce type de problème est connu, au sein de la communauté de la vision par ordinateur, sous le nom de reconnaissance à grain fin. Il consiste en des catégories discriminantes qui étaient auparavant considérées comme une seule catégorie et qui ne présentent que de petites différences visuelles subtiles (ex. espèces d'oiseaux, modèles de voitures... etc.).

Avec la résurgence des techniques d'apprentissage en profondeur, la communauté de la vision par ordinateur a connu une ère de résultats florissants. L'une des techniques d'apprentissage en profondeur les plus utilisées sont les réseaux de neurones convolutifs (CNN) qui ont connu un énorme succès dans ce domaine. Cependant, dans la reconnaissance à grain fin, les CNN ne fonctionnent pas aussi bien que la classification d'image habituelle. Nous pensons que cela est dû à la fonction de noyau linéaire sur laquelle les CNN sont construits. Les fonctions de noyau linéaires sont moins discriminantes et ne correspondent pas aux données d'entrée. Surtout lorsque les données ne sont pas linéairement séparables. Pour surmonter ce problème, nous avons incorporé des fonctions plus complexes dans CNN, au lieu de simples fonctions linéaires, à différents niveaux. Ces fonctions de noyau non linéaires sont capables d'ajuster des données d'entrée plus complexes que la fonction de noyau linéaire et donc d'être plus discriminantes. Ces méthodes ont également l'avantage d'être moins consommatrices de mémoire, même si elles sont plus difficiles à apprendre.

Au niveau de la mise en commun, nous avons d'abord proposé d'utiliser la mise en commun bilinéaire et bilinéaire améliorée avec les CNN pour le FER. Ce cadre a été évalué pour les ensembles de données FER et a montré que l'utilisation de la mise en commun bilinéaire et améliorée avec les CNN peut améliorer la précision globale à près de 3% pour le FER et obtenir des résultats de pointe. Nous avons également introduit une couche de regroupement plus sensible à la distorsion du filtre basée sur les fonctions du noyau. La mise en commun proposée réduit les dimensions de la carte d'entités tout en gardant une trace de la majorité des informations transmises à la couche suivante au lieu d'en ignorer une partie. Les expériences sur les bases de données FER démontrent les avantages d'une telle couche et montrent que notre modèle atteint des résultats compétitifs par rapport aux approches de l'état de l'art. Au niveau des couches entièrement connectées, nous avons proposé une couche dense noyautée (KDL) qui capture les interactions d'entités d'ordre supérieur au lieu des relations linéaires conventionnelles. Les résultats expérimentaux démontrent les avantages d'une telle couche et montrent que notre modèle atteint des résultats compétitifs par rapport aux approches de l'état de l'art sur les jeux de données FER.

Pour améliorer encore les performances des CNN, nous avons étudié l'utilisation des fonctions du noyau aux différentes couches de ce dernier. Nous avons mené des études approfondies sur leur impact sur les couches convolutionnelles, de mise en commun et entièrement connectées. Nous remarquons que le noyau linéaire peut ne pas être suffisamment efficace pour s'adapter aux distributions de données d'entrée, alors que les noyaux d'ordre élevé sont sujets à un surajustement. Cela conduit à conclure qu'un compromis entre complexité et performance doit être atteint. Nous avons utilisé des combinaisons de nos méthodes précédemment proposées sur plusieurs ensembles de données. Les expériences sur des ensembles de données de classification conventionnels, c'est-à-dire MNIST, FASHION-MNIST et CIFAR-10, montrent que les techniques proposées améliorent les performances du réseau par rapport à la convolution classique, au regroupement et aux couches entièrement connectées. De plus, des expériences sur la classification fine, c'est-à-dire les bases de données FER, ont démontré que le pouvoir discriminatif du réseau est renforcé puisque les techniques proposées améliorent la prise de conscience des légers détails visuels et permettent au réseau d'atteindre des résultats de pointe.

L'étude approfondie décrite ci-dessus nous a amenés à conclure que ni les noyaux linéaires ni non linéaires ne sont suffisants pour atteindre les meilleures performances sans sur-ajustement. Ainsi, une combinaison de ces méthodes doit être utilisée pour atteindre les meilleurs résultats. Par conséquent, nous avons proposé une méthode de combinaison, basée sur le modèle CNN amélioré par noyau. Notre méthode améliore les performances d'un CNN sans augmenter ni sa profondeur ni sa largeur. Il consiste à développer la fonction noyau linéaire, utilisée à différents niveaux d'un CNN. L'expansion est effectuée en combinant plusieurs noyaux polynomiaux avec des degrés différents. Ce faisant, nous permettons au réseau d'apprendre automatiquement le noyau approprié pour la tâche cible spécifique. Le réseau peut soit utiliser un noyau spécifique, soit une combinaison de plusieurs noyaux. Dans ce dernier cas, nous aurons un noyau sous la forme d'un noyau en série de Taylor. Cette fonction noyau est plus sensible aux détails subtils que la fonction linéaire et est capable de mieux s'adapter aux données d'entrée. La sensibilité aux détails visuels subtils est un facteur clé pour une meilleure reconnaissance des expressions faciales. De plus, cette méthode utilise le même nombre de paramètres qu'une couche de convolution ou qu'une couche dense. Les expériences menées sur les jeux de données FER montrent que l'utilisation de notre méthode permet au réseau de surpasser les CNN ordinaires.

mots-clés: reconnaissance des expressions faciales ; Reconnaissance fine; Fonction noyau ; L'apprentissage en profondeur.

Abstract

Facial expression recognition (FER) is a research area that consists of classifying human emotions through the expressions on their faces as one of seven basic emotions: happiness, sadness, fear, disgust, anger, surprise, and neutral. FER finds applications in different fields including security, intelligent human-computer interaction, robotics, and clinical medicine for autism, depression, pain, and mental health problems. FER is a very challenging problem due to the subtle differences that exist between its categories. Indeed, the difference in facial expression categories relies on small subtle areas in the facial images like the mouth, eyebrows. This type of problem is known, within the computer vision community as Fine-Grained recognition. It consists of discriminating categories that were considered previously as a single category and have only small subtle visual differences (e.g. bird species, car models…etc.).

With the resurgence of deep learning techniques, the computer vision community has witnessed an era of blossoming results. One of the most used deep learning techniques are Convolutional Neural Networks (CNNs) which have been extremely successful in that field. However, in fine-grained recognition CNNs do not perform as well as the usual image classification. We believe this is due to the linear kernel function that CNNs are built on. Linear kernel functions are less discriminative and fails to fit the input data. Especially when the data is not linearly separable. To overcome this issue, we have incorporated more complex functions in CNN, instead of simple linear functions, at different levels. These non-linear kernel functions are able to fit more complex input data than the linear kernel function and thus be more discriminative. These methods also have the benefits of being less memory-consuming, even though they are harder to train.

At the pooling level, we first proposed to use bilinear and improved bilinear pooling with CNNs for FER. This framework has been evaluated FER datasets and has shown that the use of bilinear and improved bilinear pooling with CNNs can enhance the overall accuracy to nearly 3% for FER and achieve state-of-the-art results. We have also introduced a more filter distortion-aware pooling layer based on kernel functions. The proposed pooling reduces the feature map dimensions while keeping track of the majority of the information fed to the next layer instead of ignoring part of them. The experiments on FER databases demonstrate the benefits of such a layer and show that our model achieves competitive results with respect to state-of-the-art approaches. At the fully connected layers level, we proposed a Kernelized Dense Layer (KDL) which captures higher-order feature interactions instead of conventional linear relations. The experimental results demonstrate the benefits of such a layer and show that our model achieves competitive results with respect to the state-of-the-art approaches on FER datasets.

To further improve CNNs performance, we investigated the usage of kernel functions at the different layers of the latter. We carried out extensive studies of their impact on convolutional, pooling, and fully-connected layers. We notice that the linear kernel may not be sufficiently effective to fit the input data distributions, whereas high order kernels are prone to over-fitting. This leads to conclude that a trade-off between complexity and performance should be reached. We have used combinations of our previously proposed methods on several datasets. The experiments on conventional classification datasets i.e. MNIST, FASHION-MNIST, and CIFAR-10, show that the proposed techniques improve the performance of the network compared to classical convolution, pooling, and fully connected layers. Moreover, experiments on fine-grained classification i.e. FER databases demonstrated that the discriminative power of the network is boosted since the proposed techniques improve the awareness to slight visual details and allow the network to reach state-of-the-art results.

The extensive study described above led us to conclude that neither linear nor non-linear kernels are sufficient enough to reach the best performance without over-fitting. Thus a combination of these methods must be used to reach the best results. Therefore, we proposed a combination method, based on kernel enhanced CNN model. Our method improves the performance of a CNN without increasing neither its depth nor its width. It consists of expanding the linear kernel function, used at different levels of a CNN. The expansion is performed by combining multiple polynomial kernels with different degrees. By doing so, we allow the network to automatically learn the suitable kernel for the specific target task. The network can either use one specific kernel or a combination of multiple kernels. In the latter case, we will have a kernel in the form of a Taylor series kernel. This kernel function is more sensitive to subtle details than the linear one and is able to better fit the input data. The sensitivity to subtle visual details is a key factor for better facial expression recognition. Furthermore, this method uses the same number of parameters as a convolution layer or a dense layer. The experiments conducted on FER datasets show that the use of our method allows the network to outperform ordinary CNNs.

keywords: Facial expression recognition; Fine-grained recognition; Kernel function; Deep learning.

Contents

List of Figures

List of Tables

Introduction

0.1 Scope and Overview

In recent years, machine learning has become more and more popular in research and has been incorporated in a large number of applications, including multimedia concept retrieval, image classification, video recommendation, social network analysis, text mining, and so forth. Among various machine-learning algorithms, "deep learning," also known as representation learning [Deng, 2014], is widely used in these applications. The explosive growth and availability of data and the remarkable advancement in hardware technologies have led to the emergence of newstudies in distributed and deep learning. Deep learning, which has its roots from conventional neural networks, significantly outperforms its predecessors. It utilizes graph technologies with transformations among neurons to develop many-layered learning models. Many of the latest deep learning techniques have been presented and have demonstrated promising results across different kinds of applications such as Natural Language Processing (NLP), visual data processing, speech and audio processing, and many other well-known applications [Yan u. a., 2017; 2015]

One of the most used deep learning techniques are Convolutional Neural Networks (CNNs) which have been extremely successful in computer vision applications. They showed to perform very competitive results while linear operations are used at different layers of the network. Linear functions are efficient, particularly, when the original data is linearly separable, which should have, in general, a high dimensional representation. In such a case, the decision boundary can be representable as a linear combination of the original features. It is worth noting that not every high dimensional problems are linearly separable [Robert, 2014]. For instance, images may have a high dimensional representation, but individual pixels are not very informative. Moreover, taking in consideration only small regions of the image, dramatically reduces their dimension, which makes linear functions less sensitive to subtle changes in input data. Researchers are trying to overcome this problem by either increasing the network size or by employing more complex functions. In the first case, researchers are continuously trying to enhance CNNs by increasing their depth (number of layers) or width (size of the output of each layer). Even though by doing so the performance of the network is effectively enhanced, it can not be a longstanding solution. Indeed, these methods drastically increase the number of weights and the network complexity. Therefore, the resulting models can only be used on powerful devices. In the second case, the focus is more on computation. Many researchers incorporated more complex kernel functions in CNN, instead of simple linear functions, at different levels. These methods have the benefits of being less memory consuming, even though they are harder to train.

Kernel methods are a class of algorithms for pattern analysis or recognition, whose best known element is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (such as clusters, rankings, principal components, correlations, classifications) in general types of data (such as sequences, text documents, sets of points, vectors, images, graphs, etc). The intuition behind is to make the underlying linear kernel operates on higher dimensional feature map so that it becomes more discriminative. In other words, instead of running a linear classifier directly on feature, they are first mapped to a higher-dimensional Reproducing Kernel Hilbert Space (RKHS) using a positive definite kernel function. For certain kernel functions, the RKHS can even be infinite dimensional. A linear classifier is then run on this high-dimensional RKHS. Since the dimensionality of the feature vectors is dramatically increased via this mapping, a linear classifier in the RKHS corresponds to a powerful nonlinear classifier in the original feature vector space. Such a classifier is capable of learning more complex patterns than a linear classifier directly operating on the feature vectors.

This great success encouraged the computer vision community to tackle more challenging tasks in this field. One of these challenging tasks is the fine-grained recognition. It consists of discriminating categories that were considered previously as a single category and have only small subtle visual differences(e.g. bird species). Facial expression recognition is considered one of most challenging fine-grained recognition problems. Indeed, the difference in facial expression categories relies on small subtle areas in the facial images like the mouth, eyebrows and the noise. To overcome this issue, facial expression recognition systems must be able to recognise this subtle differences efficiently. We believe that incorporating non-linear kernel functions at different level of a CNN can enhance the discriminative power of the later. Hence, it will be more accurate on image classification task. Moreover, it can perform well on fine-grained visual classification tasks like facial expression recognition.

0.2 Outline of the Thesis

Our thesis start with an introduction in which the scope of our study is briefly defined. It also sheds light on the intuition behind the research path we have chosen, the different techniques involved and purpose of choosing a specific case of study. After that, the outline of the remaining chapters of the thesis is given.

Chapter 1 sheds lights on the important concept related to our field of study. First of all, We detail this new emergent domain in artificial intelligence called Deep Learning, review the most important models, framework and its different application fields. Second, we give a brief overview about kernel methods. This overview covers the mathematical basics that the kernel methods rely on. It also explains the construction steps of kernels, the important concept of the kernel trick, and enumerate some well known kernel functions. Since our work is centred around the incorporation of kernel methods in deep learning networks. Among the deep learning models, our research focused specifically on convolutional neural networks (CNN). Therefore, the later is more detailed than other models. Finally, our case of study: facial expression recognition is defined and some important study in this field are also illustrated.

Chapter 2 presents some techniques that were used to improve the CNN performance on fine-grained visual tasks. This accuracy enhancement brought in multiple visual tasks, shows that their is still room for improvement for CNNs on FER. In this chapter, we propose to use bilinear and improved bilinear pooling with CNNs for FER. This framework has been evaluated on three well known datasets, namely ExpW, FER2013 and RAF-DB. It has shown that the use of bilinear and improved bilinear pooling with CNNs can enhance the overall accuracy to nearly 3% for FER and achieve state-of-the-art results.

Chapter 3 introduce one of our major contribution. It consists of a more filter distortion aware pooling layer based on kernel functions. The proposed pooling reduces the feature map dimensions while keeping track of the majority of the information fed to the next layer instead of ignoring part of them. The experiments on RAF, FER2013 and ExpW databases demonstrate the benefits of such layer and show that our model achieves competitive results with respect to the state-of-the-art approaches.

Chapter 4 also introduce an innovative work that is similar to the precedent chapter, yet focuses on the fully connected layer of Convolutional Neural Networks. This method is called Kernelized Dense Layer (KDL) which captures higher order feature interactions instead of conventional linear relations. We apply this method to Facial Expression Recognition (FER) and evaluate its performance on RAF, FER2013 and ExpW datasets. The experimental results demonstrate the benefits of such layer and show that our model achieves competitive results with respect to the state-of-the-art approaches.

Chapter 5 investigates the usage of kernel functions at the different layers in a convolutional neural network. We carry out extensive studies of their impact on convolutional, pooling and fully-connected layers. We notice that the linear kernel may not be sufficiently effective to fit the input data distributions, whereas high order kernels prone to over-fitting. This leads to conclude that a trade-off between complexity and performance should be reached. We show how one can effectively leverage kernel functions, by using our proposed pooling layers (chapter 3) and the proposed Kernelized Dense Layers (chapter 4). The experiments on conventional classification datasets i.e. MNIST, FASHION-MNIST and CIFAR-10, show that the proposed techniques improve the performance of the network compared to classical convolution, pooling and fully connected layers. Moreover, experiments on fine-grained classification i.e. facial expression databases, namely RAF-DB, FER2013 and ExpW demonstrate that the discriminative power of the network is boosted, since the proposed techniques improve the awareness to slight visual details and allows the network reaching state-of-the-art results.

Chapter 6 introduces a Facial Expression Recognition (FER) method, based on kernel enhanced CNN model. Our method improves the performance of a CNN without increasing its depth nor its width. It consists of expanding the linear kernel function, used at different levels of a CNN. The expansion is performed by combining multiple polynomial kernels with different degrees. By doing so, we allow the network to automatically learn the suitable kernel for the specific target task. The network can either uses one specific kernel or a combination of multiple kernels. In the latter case we will have a kernel in the form of a Taylor series kernel. This kernel function is more sensitive to subtle details than the linear one and is able to better fit the input data. The sensitivity to subtle visual details is a key factor for a better facial expression recognition. Furthermore, this method uses the same number of parameters as a convolution layer or a dense layer. The experiments conducted on FER datasets show that the use of our method allows the network to outperform ordinary CNNs.

Finally, a general conclusion concludes our thesis. It summarizes all the important findings we reached in our different contributions. It also point out the different challenges we discovered trough our research and some of which we could not overcome. Nevertheless, these non fixed challenges, consist the basics of our future work. The later are illustrated at the end of the general conclusion.

Literature Review 1

Contents

This chapter sheds lights on the important concept related to our field of study. First of all, We detail this new emergent domain in artificial intelligence called deep learning, review the most important models, framework and its different application fields. Second, we give a brief overview about kernel methods. This overview covers the mathematical basics that the kernel methods rely on. It also explains the construction steps of kernels, the important concept of the kernel trick, and enumerate some well known kernel functions. Since our work is centred around the incorporation of kernel methods in deep learning networks. Among the deep learning models, our research focused specifically on convolutional neural networks (CNN). Therefore, the later is more detailed than other models. Finally, our case of study: facial expression recognition is defined and some important study in this field are also illustrated.

1.1 Deep learning

In recent years, machine learning has become more and more popular in research and has been incorporated in a large number of applications, including multimedia concept retrieval, image classification, video recommendation, social network analysis, text mining, and so forth. Among various machine-learning algorithms, “deep learning,” also known as representation learning [Deng, 2014], is widely used in these applications. The explosive growth and availability of data and the remarkable advancement in hardware technologies have led to the emergence of new studies in distributed and deep learning. Deep learning, which has its roots from conventional neural networks, significantly outperforms its predecessors. It utilizes graph technologies with transformations among neurons to develop many-layered learning models. Many of the latest deep learning techniques have been presented and have demonstrated promising results across different kinds of applications such as Natural Language Processing (NLP), visual data processing, speech and audio processing, and many other well-known applications [Yan u. a., 2017; 2015]

Traditionally, the efficiency of machine-learning algorithms highly relied on the goodness of the representation of the input data. A bad data representation often leads to lower performance compared to a good data representation. Therefore, feature engineering has been an important research direction in machine learning for a long time, which focuses on building features from raw data and has led to lots of research studies. Furthermore, feature engineering is often very domain specific and requires significant human effort. For example, in computer vision, different kinds of features have been proposed and compared, including Histogram of Oriented Gradients (HOG) [Dalal und Triggs, 2005], Scale Invariant Feature Transform (SIFT) [Lowe, 1999], and Bag of Words (BoW). Once a new feature is proposed and performs well, it becomes a trend for years. Similar situations have happened in other domains including speech recognition and NLP.

Comparatively, deep learning algorithms perform feature extraction in an automated way, which allows researchers to extract discriminative features with minimal domain knowledge and human effort [Najafabadi u. a., 2015]. These algorithms include a layered architecture of data representation, where the high-level features can be extracted from the last layers of the networks while the low-level features are extracted from the lower layers. These kinds of architectures were originally inspired by Artificial Intelligence (AI) simulating its process of the key sensorial areas in the human brain. Our brains can automatically extract data representation from different scenes. The input is the scene information received from eyes, while the output is the classified objects. This highlights the major advantage of deep learning—i.e., it mimics how the human brain works.

1.1.1 Deep learning networks

In this section, several popular deep learning networks such as recurrent neural network(RNN), convolutional neural network (CNN), and deep generative models are discussed. However, since deep learning has been growing very fast, many new networks and new architectures appear every few months, which is out of the scope of this Thesis.

Convolutional Neural Network (CNN)

CNN is also a popular and widely used algorithm in deep learning [LeCun u. a., 1995]. It has been extensively applied in different applications such as NLP [Wang u. a., 2016], speech processing [Dahl u. a., 2011], and computer vision [Krizhevsky u. a., 2012], to name a few. Similar to the traditional neural networks, its structure is inspired by the neurons in animal and human brains. Specifically, it simulates the visual cortex in a cat's brain containing a complex sequence of cells [Hubel und Wiesel, 1962]. As described in [Goodfellow u. a., 2016], CNN has three main advantages, namely, parameter sharing, sparse interactions, and equivalent representations. To fully utilize the twodimensional structure of an input data (e.g., image signal), local connections and shared weights in the network are utilized, instead of traditional fully connected networks. This process results in very fewer parameters, which makes the network faster and easier to train. This operation is similar to the one in the visual cortex cells. These cells are sensitive to small sections of a scene rather than the whole scene. In other words, the cells operate as local filters over the input and extract spatially local correlation existing in the data.

In typical CNNs, there are a number of convolutional layers followed by pooling (subsampling) layers, and in the final stage layers, fully connected layers (identical to Multilayer Perceptron (MLP)) are usually used. The layers in CNNs have the inputs x arranged in three dimensions, $W \times H \times C$, wherem refers to the height and width of the input, and C refers to the depth or the channel numbers (e.g., $C = 3$ for an RGB image).

1. **Convolution layer:** In each convolutional layer, there are several filters (kernels) k of size $n \times n \times q$. Here, n should be smaller than the input image, but q can be either smaller or the same size as C . As mentioned earlier, the filters are the base of local connections that are convolved with the input and share the same parameters (weight W^k and bias b^k) to generate k feature maps (h^k). Similar to MLP, the convolutional layer computes a dot product between the weights and its inputs (as illustrated in Equation 1.1), but the inputs are small regions of the original input volume (Fig 1.1). Then, an activation function f or a nonlinearity is applied to the output of the convolutional layers:

$$h^k = f(W^k * x + b^k) \tag{1.1}$$

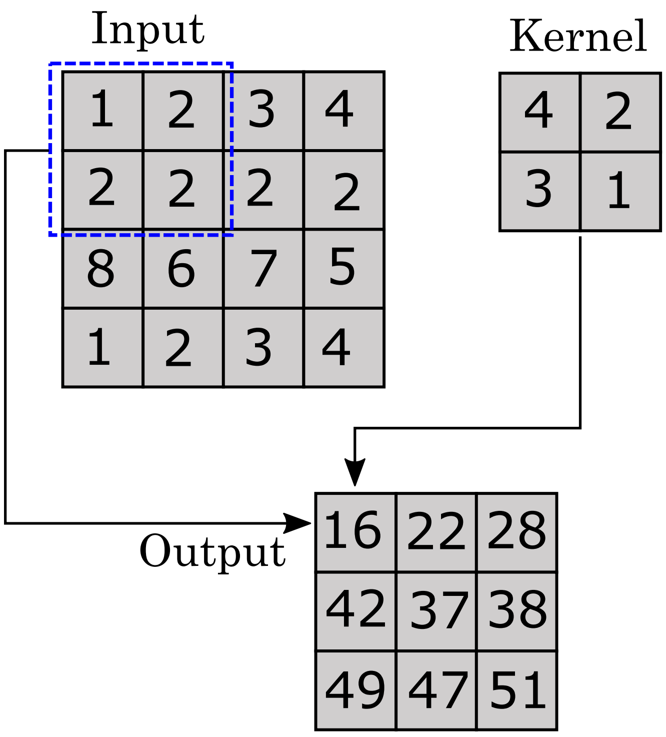


Figure 1.1 – *Convolution layer with a kernel of size* 2×2.

2. **Pooling layer:** Thereafter, in the subsampling layers, each feature map is down-sampled to decrease the parameters in the network, speeds up the training process, and hence controls overfitting. The pooling operation (e.g., average or max) is done over a $p \times p$ (where p is the filter size) contiguous region for all feature maps (Fig 1.2).

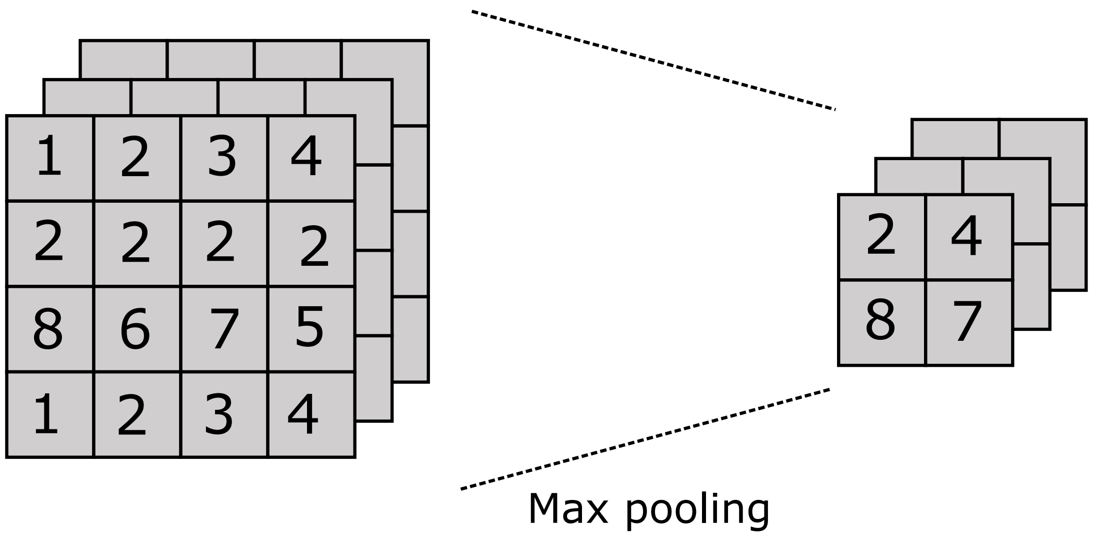


Figure 1.2 – *The Max pooling method keeps only the maximum values over the channel axis. Average pooling work in a similar manner, yet instead of keeping the maximum value it computes the average.*

3. **Fully connected layers:** Finally, the final stage layers are usually fully connected

as seen in the regular neural networks. These layers take previous low-level and midlevel features and generate the high-level abstraction from the data. The last layer can be used to generate the classification scores, where each score is the probability of a certain class for a given instance.

In recent years, we have witnessed the birth of numerous CNNs. These networks have gotten so deep that it has become extremely difficult to visualise the entire model. We stop keeping track of them and treat them as black-box models. In the following we will present some of the most popular CNN architecture.

1. **LeNet:** is the most popular CNN architecture it is also the first CNN model which came in the year 1998 [LeCun u. a., 2015a]. LeNet was originally developed to categorise handwritten digits from 0–9 of the MNIST Dataset. It is made up of seven layers, each with its own set of trainable parameters. It accepts a 32 × 32 pixel picture, which is rather huge in comparison to the images in the data sets used to train the network. RELU is the activation function that has been used.

2. **AlexNet:** Starting with an 11x11 kernel, Alexnet [Krizhevsky u. a., 2012] is built up of 5 conv layers. For the three massive linear layers, it was the first design to use max-pooling layers, ReLu activation functions, and dropout. The network was used to classify images into 1000 different categories. The network is similar to the LeNet Architecture, but it includes a lot more filters than the original LeNet, allowing it to categorise a lot more objects. Furthermore, it deals with overfitting by using "dropout" rather than regularisation. Two GPUs were used to train the initial network. It contains eight layers, each with its own set of settings that may be learned. RGB photos are used as input to the Model. Relu is the activation function utilised in all levels. Two Dropout layers were employed. Softmax is the activation function utilised in the output layer.

3. **GoogleNet / Inception:** The ILSVRC 2014 competition was won by the GoogleNet [Szegedy u. a., 2015] or Inception Network, which had a top-5 error rate of 6.67 percent, which was virtually human level performance. Google created the model, which incorporates an improved implementation of the original LeNet design. This is based on the inception module concept. GoogLeNet is a variation of the Inception Network, which is a 22-layer deep convolutional neural network. GoogLeNet is now utilised for a variety of computer vision applications, including face detection and identification, adversarial training, and so on. The InceptionNet/GoogleLeNet design is made up of nine inception modules stacked on top of each other, with max-pooling layers between them (to halve the spatial dimensions). It is made up of 22 layers (27 with the pooling layers). After the last inception module, it employs global average pooling.

4. **ResNet:** is a well-known deep learning model that was first introduced in [He u. a., 2016] . ResNet is one of the most widely used and effective deep learning

models to date. ResNets are made up of what's known as a residual block. This is built on the concept of "skip-connections" and uses a lot of batch-normalization to let it train hundreds of layers successfully without sacrificing speed over time.

5. **VGG:** VGG [Simonyan und Zisserman, 2014] is a convolutional neural network design that has been around for a long time. It was based on a study on how to make such networks more dense. Small 3 x 3 filters are used in the network. The network is otherwise defined by its simplicity, with simply pooling layers and a fully linked layer as additional components. In comparison to AlexNet, VGG was created with 19 layers deep to replicate the relationship between depth and network representational capability. Small size filters can increase the performance of CNNs. Based on these observations, VGG replaced the 11x11 and 5x5 filters with a stack of 3x3 filters, demonstrating that the simultaneous placement of small size (3x3) filters may provide the effect of a big size filter (5x5 and 7x7). By lowering the number of parameters, the usage of tiny size filters gives an additional benefit of low computing complexity. These discoveries ushered in a new research trend at CNN, which is to work with lower size filters.

Recurrent Neural Network (RNN)

Another widely used and popular algorithm in deep learning, especially in NLP and speech processing, is RNN [Cho u. a., 2014]. Unlike traditional neural networks, RNN utilizes the sequential information in the network. This property is essential in many applications where the embedded structure in the data sequence conveys useful knowledge. For example, to understand a word in a sentence, it is necessary to know the context. Therefore, an RNN can be seen as short-term memory units that include the input layer x, hidden (state) layer s, and the output layer y.

One main issue of an RNN is its sensitivity to the vanishing and exploding gradients [Glorot und Bengio, 2010]. In other words, the gradients might decay or explode exponentially due to the multiplications of lots of small or big derivatives during the training. This sensitivity reduces over time, which means that the network forgets the initial input with the entrance of the new ones. Therefore, Long Short-Term Memory (LSTM) [Li und Wu, 2015] is utilized to handle this issue by providing memory blocks on its recurrent connections. Each memory block includes memory cells that store the temporal states of the network. Moreover, it includes gated units to control the information flow. Furthermore, residual connections in very deep networks [He u. a., 2016] can alleviate the vanishing gradient issue significantly.

Generative Adversarial Networks (GAN)

These are the class of generative models based on game theory [Goodfellow u. a., 2014]. Which do not explicitly model the data distribution but rather models the sample from it. Sampling is performed using a deep neural network. The neural network takes as input random noise and transforms it into model distribution. Generative Adversarial Network consists of two neural networks. One is called Generator and another one is called Discriminator. This model is called adversarial because the generator is constantly trying to fool the discriminator into believing that the input is from training data(real data). While the discriminator always distinguishes between the two.

1. **Generator:** A neural network that takes as input, a random noise vector and transform it into a model distribution.
2. **Discriminator:** It is a neural network that distinguishes between output data point (Fake) and training data samples (Real). It acts like a classifier as if the input is real or fake.

These two neural networks are always trying to work against each other. In these setting the weights of generator learns to converts a random noise vector into a model distribution. The generator takes a random noise vector from the latent space and outputs some samples. Now the discriminator takes input from training data (real) and checks against the generated fake sample from generator. The training data should have images from the similar kinds of tasks say paintings or faces etc. Upon taking both inputs and errors the function outputs probability that particular sample is real or fake. This output is used to train the weights of the generator as well as the discriminator. The other important part is formulation of error function or cost function in GANs. This problem is formulated as MiniMax zero sum game.

1.1.2 Deep learning techniques and frameworks

Different deep learning algorithms help improve the learning performance, broaden the scopes of applications, and simplify the calculation process. However, the extremely long training time of the deep learning models remains a major problem for the researchers. Furthermore, the classification accuracy can be drastically enhanced by increasing the size of training data and model parameters. In order to accelerate the deep learning processing, several advanced techniques are proposed in the literature. Deep learning frameworks combine the implementation of modularized deep learning algorithms, optimization techniques, distribution techniques, and support to infrastructures. They are developed to simplify the implementation process and boost the system-level development and research. In this section, some of these representative techniques and frameworks are introduced.

Unsupervised and Transfer Learning

Contrary to the vast amount of work done in supervised deep learning, very few studies have addressed the unsupervised learning problem in deep learning. However, in recent years, the benefit of learning reusable features using unsupervised techniques has shown promising results in different applications. In the last decade, the idea of having a self-taught learning framework has been widely discussed in the literature [Le, 2013, Radford u. a., 2015, Sermanet u. a., 2013].

In practice, very few people have the luxury of accessing very high-speed GPUs and powerful hardware to train a very deep network from scratch in a reasonable time. Therefore, pretraining a deep network (e.g., CNN) on large-scale datasets (e.g., ImageNet) is very common. This technique is also known as transfer learning [Learning, 2017], which can be done by using the pretrained networks as fixed feature extractors (especially for small new datasets) or fine-tuning the weights of the pretrained model (especially for large new datasets that are similar to the original one). In the latter, the model should continue the learning to fine-tune the weights of all or some of the high-level parts of the deep network. This approach can be considered as a semisupervised learning, in which the labeled data is insufficient to train a whole deep network.

Online Learning

Usually, the network topologies and architectures in deep learning are time static (i.e., they are predefined before the learning starts) and are also time invariant [LeCun u. a., 2015b]. This restriction on time complexity poses a serious challenge when the data is streamed online. Online learning previously came into mainstream research [Choy u. a., 2006], but only a modest advancement has been observed in online deep learning. Conventionally, deep neural networks (DNN) are built upon the stochastic gradient descent (SGD) approach in which the training samples are used individually to update themodel parameters with a known label. The need is that rather than the sequential processing of each sample, the updates should be applied as batch processing. One approach was presented in [Scherer u. a., 2010] where the samples in each batch are treated as Independent and Identically Distributed (IID). The batch processing approach proportionally balances the computing resources and execution time.

Another challenge that stacks up on the issue of online learning is high-velocity data with timevarying distributions. This challenge represents the retail and banking data pipelines that hold tremendous business values. The current premise is that the data is largely close in time to safely assume piecewise stationarity, thus having a similar distribution. This assumption characterizes data with a certain degree of correlation and develops the models accordingly, as discussed in [Chien und Hsieh, 2013]. Unfortunately, these nonstationary data streams are not IID and are often longitudinal data streams. Moreover, online learning is often memory delimited, is harder to parallelize, and requires a linear learning rate on each input sample. Developing methods that are capable of online learning from non-IID data would be a big leap forward for big data deep learning.

Optimization Techniques in Deep Learning

Training a DNN is an optimization process, i.e., finding the parameters in the network that minimize the loss function. In practice, the SGD method [Sutskever u. a., 2013] is a fundamental algorithm applied to deep learning, which iteratively adjusts the parameters based on the gradient for each training sample. The computational complexity of SGD is lower than that of the original gradient descent method, in which the whole dataset is considered every time the parameters are updated.

In the learning process, the updating speed is controlled by the hyperparameter learning rate. Lower learning rates will eventually lead to an optimal state after a long time, while higher learning rates decay the loss faster but may cause fluctuations during the training [Pouyanfar und Chen, 2017]. In order to control the oscillation of SGD, the idea of using momentum is introduced.

On the other hand, several techniques are proposed to determine the proper learning rate. Primitively, weight decay and learning rate decay are introduced to adjust the learning rate and accelerate the convergence [Loshchilov und Hutter, 2017, Zhang u. a., 2018a, Xie u. a., 2020]. A weight decay works as a penalty coefficient in the cost function to avoid overfitting, and a learning rate decay can reduce the learning rate dynamically to improve the performance. Moreover, adapting the learning rate with respect to the gradient of the previous stages is found helpful to avoid the fluctuation

Deep Learning in Distributed Systems

The efficiency of model training is limited to a single-machine system, and the distributed deep learning techniques have been developed to further accelerate the training process. There are two main approaches to train the model in a distributed system, namely, data parallelism and model parallelism [Pouyanfar u. a., 2018]. For data parallelism, the model is replicated to all the computational nodes and each model is trained with the assigned subset of data. After a certain period of time, the weight update needs to be synchronized among the nodes. Comparatively, for model parallelism, all the data is processed with one model where each node is responsible for the partial estimation of the parameters in the model.

Both data-parallel and model-parallel strategies have their own limitations. On one hand, if data parallelism has too many training modules, it has to decrease the learning rate to make the training procedure smooth. On the other hand, if model parallelism has too many segmentations, the output from the nodes will increase sharply and reduce the efficiency accordingly [Yadan u. a., 2013]. Generally speaking, the larger the dataset is, the more beneficial it is to have data parallelism. The larger the deep learning model is, the more suitable it is to have model parallelism. Besides, compared to data parallelism, it is hard to hide the communication needed for synchronization in model parallelism because only partial information is included in each node for the whole batch. Thus, it is necessary to wait till the synchronization step finishes before moving forward to the next layer since the activities are unable to be processed with only partial information. The two kinds of strategies can also be fused to a hybrid model as discussed in [Yadan u. a., 2013].

Deep Learning Frameworks

Table 1.1 – *The Comparison of Different Deep Learning Frameworks.*

Framework	License	Core Language	Interface support	CNN & RNN support	DBN support
Caffe [Jia u. a., 2014]	BSD	C++	Python & MATLAB	Yes	No
DeepLearning4j (DL4j)	Apache 2.0	Java	Java? Scala & Python	Yes	Yes
Torch [Collobert u. a., 2002]	BSD	C & Lua	C/C++, Lua & Python	Yes	Yes
Neon	Apache 2.0	Python	Python	Yes	Yes
Theano [Team u. a., 2016]	BSD	Python	Python	Yes	Yes
MXNet [Chen u. a., 2015]	Apache 2.0	C++	C++, Python, R, Scala, Perl,Julia, etc	Yes	Yes
TensorFlow [Abadi u. a., 2016]	Apache 2.0	C++ & Python	Python, C/C++, Java & Go	Yes	Yes
CNTK [Yu u. a., 2014b]	MIT	C++	Python, C++ & BrainScript	Yes	Yes

Table 1.1 lists a smattering of popular deep learning frameworks for architecture designs, such as Caffe [Jia u. a., 2014], DeepLearning4j (DL4j), Torch [Collobert u. a., 2002], Neon, Theano [Team u. a., 2016], MXNet [Chen u. a., 2015], TensorFlow [Abadi u. a., 2016], and Microsoft Cognitive Toolkit (CNTK) [Yu u. a., 2014b]. In Table 1.1, the license, core language, supported interface language, and framework support of CNN, RNN, and DBN are also listed.

It can be observed from Table 1.1 that C++ is usually used for implementation of deep learning frameworks because it accelerates the speed of training. Since GPU is significantly helpful to speed up the matrix computation, most of the aforementioned frameworks also support GPU via the interface provided by CuDNN [Chetlur u. a., 2014]. Meanwhile, Python has become the most common language for deep learning architecture design since it can make the programming more efficient and easier by simplifying the programming process. Also, distributed calculation becomes common in some recently released frameworks such as TensorFlow,MXNet, and CNTK. The goal is to further improve the calculation efficiency for deep learning. Moreover, TensorFlow also includes support for the customized deep learning Application-Specific Integrated Circuit (ASIC), called Tensor Processing Unit (TPU), to help increase the efficiency and decrease the power consumption.

Caffe, implemented by Berkeley Vision and Learning Center, is one of the most widely used frameworks [Jia u. a., 2014]. It supports the most commonly used layers for both CNN and RNN but does not directly enable the use of DBN. Users of Caffe design their architecture by declaring the structure of a computation graph, such as convolutional layers. There are pretrained models available for a wide range of neural networks such as AlexNet [Krizhevsky u. a., 2012], GoogleNet [Szegedy u. a., 2015], and ResNet [He u. a., 2016]. Furthermore, Caffe is a single-machine framework. In other words, it does not support multinode execution while the multi-GPU calculation is supported when there are external offerings like CaffeOnSpark by Yahoo that integrate Caffe with a big data engine like Spark.

DL4j is the most popular framework implemented in Java, developed and maintained by Skymind since 2014. Cooperating with Hadoop and Spark, DL4j is capable of distributed computation as well. However, this framework is reported to have a longer training time for similar architectures benchmarked with other frameworks [Kovalev u. a., 2016].

Torch was first released in 2002 and extended its deep learning feature in 2011 [Collobert u. a., 2002]. Combined with Facebook's deep learning CUDA library (fbcunn) [Vasilache u. a., 2014], Torch can operate model and data level parallel computation. Unlike other frameworks, Torch is built based on a dynamic graph representation instead of a static graph. The dynamic graph allows the user to update the computational graph (i.e., to change the model structure) during runtime, while the static graph uses certain functions to define the graphs in advance. Torch released its Python interface, PyTorch, and the usage of this framework has greatly increased due to its flexibility.

Neon and Theano [Team u. a., 2016] are two frameworks developed in Python by Intel and the University of Montreal, respectively. Both of them perform code optimizations in the system and kernel level. Therefore, their training speeds usually outperform other frameworks. However, although only parallelism and multi-GPU are supported, the multinode calculation is not designed in these frameworks.

MXNet [Chen u. a., 2015] supports several interfaces, including C++, Python, R, Scala, Perl, MATLAB, Javascript, Go, and Julia. It supports both computation graph declarations and imperative computation declarations for architecture design. MXNet not only supports data and model parallelism but also follows parameter server schemes to support distributed calculation as well. MXNet has the most comprehensive functionality, but the performance is not optimized as much as other state-of-theart frameworks.

TensorFlow [Abadi u. a., 2016] is implemented by Google and provides a series of internal functions to help implement any deep neural network based on the static computational graph. Recently, Keras started to support Tensorflow via a high-level interface and allowed users to design the architecture without worrying about the internal design. The framework provides different levels of parallel and distributed operations and well-designed fatal tolerance. The robustness of its design attracts a lot of users and it has become one of the most popular deep learning frameworks since its release. All our experiments are implemented in tensorflow 2.0. However, they can be run in earlier versions of tensorflow, except for the improved bilinear models introduced in chapter 2. This technique can be used with tensorflow 1.4 and later version. Since the matrix logarithm function and the matrix square-root function are only implemented in these version of tensorflow, unless they are built from scratch in version 1.3 or earlier.

CNTK [Yu u. a., 2014b], designed by Microsoft, has a specific high-level script language, BrainScript, for neural network implementation. CNTK models the neural network as a directed graph. Each node in the graph represents an operation or a filter and each edge refers to the data flow. Instead of the parameter server model, the Message Passing Interface is applied for distributed calculation support.

1.1.3 Applications of deep learning

Nowadays, applications of deep learning include but are not limited to NLP (e.g., sentence classification, translation, etc.), visual data processing (e.g., computer vision, multimedia data analysis, etc.), speech and audio processing (e.g., enhancement, recognition, etc.), social network analysis, and healthcare. This section provides details for the different techniques used for each application.

Visual Data Processing

Deep learning techniques have become the main parts of various state-of-the-art multimedia systems and computer vision [Ha u. a., 2015]. More specifically, CNNs have shown significant results in different real-world tasks, including image processing, object detection, and video processing.

1. **Image Classification:** With the advent of deep learning, in combination with robust AI hardware and GPUs, outstanding performance can be achieved on image classification tasks. Hence, deep learning brought great successes in the entire field of image recognition, face recognition, and image classification algorithms achieve above human-level performance and real-time object detection.

   In comparison to the conventional computer vision approach in early image processing around two decades ago, deep learning requires only the knowledge of engineering of a machine learning tool. It doesn't need expertise in particular machine vision areas to create handcrafted features.

2. **Object Detection and Semantic Segmentation:** Deep learning techniques play a major role in the advancement of object detection in recent years. Before that, the best object detection performance came from complex systems with several low-level features (e.g., SIFT, HOG, etc.) and high-level contexts. However, with the advent of new deep learning techniques, object detection has also reached a new stage of advancement. These advances are driven by successful methods such as region proposal and Region-based CNN (R-CNN) [Girshick u. a., 2014].

   Semantic segmentation is the process of understanding an image in pixel level that is necessary for real-world applications such as autonomous driving, robot vision, and medical systems. Now the question is how to convert image classification to semantic segmentation. In recent years, many research studies apply deep learning techniques to classify an image pixel-wise. A deconvolutional network [Noh u. a., 2015], for instance, includes deconvolution and unpooling modules to detect and classify the segmentation regions.

3. **Video Processing:** Video analytics has attracted considerable attention in the computer vision community and is considered as a challenging task since it includes both spatial and temporal information. In an early work, large-scale YouTube videos containing 487 sport classes are used to train a CNN model [Karpathy u. a., 2014]. The model includes a multiresolution architecture that utilizes the local motion information in videos and includes context stream (for low-resolution image modeling) and fovea stream (for high-resolution image processing) modules to classify videos. An event detection from sport videos using deep learning is presented in [Tsagkatakis u. a., 2017]. In that work, both spatial and temporal information are encoded using CNNs and feature fusion via regularized Autoencoders.

Natural language processing (NLP)

NLP is a series of algorithms and techniques that mainly focus on teaching computers to understand the human language. Some NLP tasks include document classification, translation, paraphrase identification, text similarity, summarization, and question answering. NLP development is challenging due to the complexity and ambiguous structure of the human language. Moreover, natural language is highly context specific, where literal meanings change based on the form of words, sarcasm, and domain specificity. Deep learning methods have recently been able to demonstrate several successful attempts in achieving high accuracy in NLP tasks.

1. **Sentiment Analysis:** This branch of NLP deals with examining a text and classifying the feeling or opinion of the writer. Most datasets for sentiment analysis are labeled as either positive or negative, and neutral phrases are removed by subjectivity classification methods.

2. **Machine Translation:** Deep learning has played an important role in the improvements of traditional automatic translation methods.

3. **Paraphrase Identification:** Paraphrase identification is the process of analyzing two sentences and projecting how similar they are based on their underlying hidden semantics. It is a key feature that is beneficial for several NLP jobs such as plagiarism detection, answers to questions, context detection, summarization, and domain identification.

4. **Summarization:** Automatic summarization can extract the most significant and relevant information from large text documents. A well-represented summary effectively reduces the size of text without losing the most important information. This can considerably decrease the time and computations required to analyze large text-based datasets.

5. **Question Answering:** An automatic question-and-answering system should be able to interpret a natural language question and use reasoning to return an appropriate reply.

Speech and Audio Processing

Audio processing is the process that operates directly on electrical or analog audio signals. It is necessary for speech recognition (or speech transcription), speech enhancement, phone classification, and music classification. Speech processing is an active research area because of its importance in perfect human-computer interaction. Besides speech recognition tasks, many research studies focus on Speech Emotion Recognition (SER) [El Ayadi u. a., 2011], Speech Enhancement (SE), and Seaker Separation (SS),as follow:

1. **Speech Emotion Recognition (SER):** Emotions influence both the voice characteristics and linguistic content of speech. SER relies heavily on the effectiveness of the speech features used for classification.

2. **Speech Enhancement (SE):** Recently, speech enhancement has aimed to improve the speech quality by using the deep learning algorithm.

3. **Speech Separation (SS):** can be viewed as a subtask of speech enhancement, which aims to separate reverberant target speech from spatially diffuse background interference [Zhang und Wang, 2017]. Different from a single-speaker environment, speaker separation focuses on reconstructing the speech of each speaker from a mixed speech with more than one speaker talking simultaneously

Other than all the aforementioned applications, deep learning algorithms are also applied to information retrieval, robotics, transportation prediction, autonomous driving, biomedicine, disaster management, and so forth.

1.2 Kernel methods: an overview

Kernel methods are a class of algorithms for pattern analysis or recognition, whose best known element is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (such as clusters, rankings, principal components, correlations, classifications) in general types of data (such as sequences, text documents, sets of points, vectors, images, graphs, etc).

The main characteristic of Kernel Methods, however, is their distinct approach to this problem. Kernel methods map the data into higher dimensional spaces in the hope that in this higher-dimensional space the data could become more easily separated or better structured. There are also no constraints on the form of this mapping, which could even lead to infinite-dimensional spaces. This mapping function, however, hardly needs to be computed because of a tool called the kernel trick.

Kernel functions must be continuous, symmetric, and most preferably should have a positive (semi-) definite Gram matrix. Kernels which are said to satisfy the Mercer's theorem are positive semi-definite, meaning their kernel matrices have only non-negative Eigen values. The use of a positive definite kernel insures that the optimization problem will be convex and solution will be unique.

However, many kernel functions which aren't strictly positive definite also have been shown to perform very well in practice. An example is the Sigmoid kernel, which, despite its wide use, it is not positive semi-definite for certain values of its parameters.

1.2.1 Kernel construction

Every linearization function ϕ defines a kernel function via

$$K(x,y) = \langle \phi(x), \phi(y) \rangle \tag{1.2}$$

It is always possible to define a kernel by choosing a linearization function ϕ and an inner product. The function $k(.,.)$ can be evaluated by explicitly mapping patterns to the linearization space and calculating the inner product in the linearization space. However, sometimes it is not necessary to actually compute ϕ. It is natural to ask under what circumstances does a function $k(.,.)$ implement an inner product in a linearization space and what does the corresponding linearization space and linearization function look like. As it turns out there is a well-developed branch of mathematics that deals with these questions: Functional analysis. In short the answer is that if $k(.,.)$ is a symmetric and positive definite kernel then k implements an inner product in a linearization space. Constructing a linearization space and an inner product for a positive definite kernel is the purpose of this section.

First, the introduction of some notation is required. For a set of patterns x_1 to x_N and a function $k(.,.)$ of two arguments the kernel matrix is the matrix that collects all pairwise applications of k to the patterns. Let us denote this $N \times N$ matrix with K and denote the entry in the i^{th} row and j^{th} column with k_{ij} then K with $k_{ij} = (x_i, x_j)$ is called the kernel matrix or Gram matrix for the patterns $x_1, \ldots, x_N$. A real and symmetric function $k(.,.)$, that is a function with the property $k(x,y) = k(y,x)$, is called a positive definite kernel if for all choices of N points the corresponding kernel matrix K is positive semi-definite, that is for all N-dimensional vectors w:

$$w^T K w \geq 0 \tag{1.3}$$

Note that for a matrix to be positive semi-definite we do not require that equality only holds for $w = 0$ (as opposed to the definition of a positive definite matrix). As K is only positive semi-definite it can have eigenvalues that are zero and does not have to be full rank. This definition of a positive definite kernel seems confusing because for a kernel to be positive definite we require the corresponding kernel matrices to be positive semi-definite. However, the definition we give is the usual definition used in machine learning and therefore we will use it, too [Schölkopf u. a., 2002].

With the definition of a positive definite kernel in mind, it is possible to construct a vector space, an inner product, and a linearization function such that the kernel condition (Eq 1.2) is fulfilled. In the following, these three steps are demonstrated in a purely formal way.

Constructing a vector space

The vector space will be a space of functions constructed from the kernel. Let $K(., x)$ denote a function that is taken to be a function of its first argument with a fixed second argument. The vector space is then defined as all functions of the form:

$$f(x) = \sum_{i=1}^{N} w_i K(x, x_i) \tag{1.4}$$

Each function in the space is a linear combination of kernel functions $K(., x_i)$ and can be expressed by some set of N patterns $x_1; \ldots; x_N$ with real coefficients $w_1; \ldots; w_N$. It is important to realize that these N patterns could be different for different functions. All functions are linear combinations of kernel functions given by k and because they are linear combinations they define a vector space—functions can be added and multiplied with scalars. When functions are added potentially all the kernel functions of the two added functions need to be included in the expansion of the summed function but the sum will still be in the vector space.

The expansion of f given in Eq. 1.4 might not be unique. There is no requirement in the definition of f that the kernel functions need to be linearly independent. If they are not independent then the same function can be expressed in different ways. The function space is the span of the generating system of functions. If there is an infinite number of potential independent kernel functions then the vector space is infinite dimensional, even though each function f can be expressed by a finite sum.

Constructing an inner product

Next we will equip this vector space with an inner product. A possibly infinite dimensional vector space with an inner product is called a pre-Hilbert space. If the limit points of all Cauchy sequences are included in the space, the space is completed and turned into Hilbert space proper. Completeness is, for example, important for defining unique projections. We will ignore these technicalities here (but see [Schölkopf u. a., 2002]) and simply note that Hilbert spaces can be thought of as the possibly infinite dimensional generalization of Euclidean spaces. Take a function f with an expansion given by Eq. 1.4

and let $g(x) = \sum_{i=1}^{M} v_i k(x, y_i)$ be another function from this space then we can define the inner product between the two functions f and g as:

$$\langle f, g \rangle_{\mathscr{H}} = \sum_{i=1}^{N} \sum_{j=1}^{M} w_i v_j k(x_i, y_j) \tag{1.5}$$

In order to distinguish the inner product in Hilbert space from the normal inner product in Euclidean space we have added the little index $\mathscr{H}$. We have to show that this definition is indeed an inner product. First we have to show that it is well-defined. The particular expansions of f and g that are used in the definition might not be unique, as mentioned above. Fortunately, the definition (Eq. 1.5) does not depend on the particular expansions of f and g that are used to calculate the inner product. To see this, let $f(x) = \sum_{i=1}^{N'} w_i' k(x, x_i')$ and $g(x) = \sum_{i=1}^{M'} v_i' k(x, y_i')$ be two new expansions of f and g that are different from the ones used in the definition of the inner product (Eq. 1.5). They will, however, result in the same inner product because

$$\begin{aligned} \sum_{i=1}^{N} \sum_{j=1}^{M} w_i v_j k(x_i, y_j) &= \sum_{i=1}^{N} w_i g(x_i) \\ &= \sum_{i=1}^{N} \sum_{j=1}^{M'} w_i v_j' k(x, y_j') \\ &= \sum_{j=1}^{M'} v_j' f(y_j') \\ &= \sum_{i=1}^{N'} \sum_{j=1}^{M'} w_i' v_j' k(x_i', y_j') \end{aligned} \tag{1.6}$$

Therefore, equation 1.5 is indeed well-defined. To show that it is an inner product it also has to be symmetric, linear in its arguments and positive definite. As k is symmetric in both arguments the above definition is also symmetric. It is obviously linear because of the linearity of the sum. Positive definiteness means that $\langle f, f \rangle_{\mathscr{H}} \geq 0$ where equality only holds for $f = 0$. Note that $\langle f, f \rangle_{\mathscr{H}} = w^T K w$ by definition. As the defining property of a positive definite kernel is that the kernel matrix K is always positive semi-definite (Eq. 1.3), it is immediately clear that $\langle f, f \rangle_{\mathscr{H}} \geq 0$. Definiteness is a bit more tricky but it can be proved that for all positive definite kernels definiteness of Eq. 1.5 holds [Schölkopf u. a., 2002]. Hence, all positive definite kernels can define an inner product in the above way. This may also justify calling these kernels positive definite.

Constructing a linearization function

Each kernel function $K(., x)$ with a fixed x is trivially contained in the vector space. It is simply an expansion with only one kernel function and a weight of one. Therefore, the inner product (Eq. 1.5) of this function with a function f that has N coefficients w_i and kernel functions $K(., x_i)$ is

$$\langle K(., x), f \rangle_{\mathscr{H}} = \sum_{i=1}^{N} w_i K(x, x_i) = f(x), \tag{1.7}$$

by the definition of the function space (Eq. 1.4). This is a remarkable fact: The inner product with the function $K(., x)$ evaluates the function f at point x. Therefore $K(., x)$ is also called the representer of evaluation. Another remarkable property directly follows from the definition of the inner product (Eq. 1.5)

$$\langle K(., x), K(., y) \rangle_{\mathscr{H}} = K(x, y), \tag{1.8}$$

because each of the two kernel functions has a simple expansion with just one summand and a coefficient of one. Due to these two properties the linear space of functions as given in Eq. 1.4 with the above dot product $\langle K(., x), K(., y) \rangle_{\mathscr{H}}$ is called a reproducing kernel Hilbert space (RKHS) in functional analysis (if it is completed).

Now, a linearization function can be defined in the following way $\phi(x) := K(., x)$. Because of the reproducing property the kernel condition $K(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathscr{H}}$ holds for this linearization function. The linearization space is a space of functions over the x. The linearization function that was constructed maps each point x in the input space to a function $K(., x)$ in the linearization space.

1.2.2 The Kernel trick

The Kernel trick is a very interesting and powerful tool. It is powerful because it provides a bridge from linearity to non-linearity to any algorithm that can expressed solely on terms of dot products between two vectors. It comes from the fact that, if we first map our input data into a higher-dimensional space, a linear algorithm operating in this space will behave non-linearly in the original input space.

The Kernel trick is really interesting because that mapping does not need to be ever computed. If our algorithm can be expressed only in terms of a inner product between two vectors, all we need is replace this inner product with the inner product from some other suitable space. That is where resides the "trick": wherever a dot product is used, it is replaced with a Kernel function. The kernel function denotes an inner product in feature space and is usually denoted as:

$$K(x, y) = \langle \phi(x), \phi(y) \rangle \tag{1.9}$$

Using the Kernel function, the algorithm can then be carried into a higher-dimension space without explicitly mapping the input points into this space. This is highly desirable, as sometimes our higher-dimensional feature space could even be infinite-dimensional and thus unfeasible to compute. For two points x and y in $\mathbb{R}^2$.

$$\langle \phi(x), \phi(y) \rangle = x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2 = \langle x, y \rangle^2 \tag{1.10}$$

A more general result can be proved. For an n dimensional input space a class of popular and flexible linearization functions is given by all monomials of degree d. A monomial of degree d takes the product of d components of an input vector x. For instance, for $n = 5$ the following are monomials of degree $d = 3 : x_1^3, x_1 x_2 x_5$: and $x_2^2 x_4$. The possible number of monomials is given by choosing d out of n with replacement. The order does not matter because of the commutativity of the product. However, for simplicity let us consider a linearization function that takes all n^d possible ordered monomials. Thus, $x_1 x_2 x_3$ is a dimension in the new space but $x_2 x_3 x_1$ would be another dimension. For the linearization function $\phi' : \mathbb{R}^n \mapsto \mathbb{R}^{n^d}$ that computes all ordered monomials it holds that:

$$\begin{aligned}
\langle \phi'(x), \phi'(y) \rangle &= \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \cdots \sum_{i_d=1}^{n} x_{i_1} x_{i_2} \ldots x_{i_d} y_{i_1} y_{i_2} \ldots y_{i_d} \\
&= \sum_{i_1=1}^{n} x_{i_1} y_{i_1} \sum_{i_2=1}^{n} x_{i_2} y_{i_2} \cdots \sum_{i_d=1}^{n} x_{i_d} y_{i_d} \\
&= \left(\sum_{i=1}^{n} x_i y_i \right)^d \\
&= \langle x, y \rangle^d
\end{aligned} \tag{1.11}$$

Calculating the inner product in the linearization space is the same as taking the inner product in the original space and taking it to the power of d. Computationally, this is an extremely attractive result. Remember that a high number of dimensions is needed to make the linearization space sufficiently flexible to be useful. If calculated naively the computational effort of the inner product in the linearization space scales with its dimensions. However, this result shows that, in the case of a monomial linearization function, it is not necessary to explicitly map the vectors x and y to the n^d dimensional linearization space to calculate the dot product of the two vectors in this space. It is enough to calculate the standard inner product in input space and take it to the power of d.

Intuitively, kernels can provide a way to efficiently calculate inner products in higher dimensional linearization spaces. They also provide a convenient non-linear generalization of inner products. With the help of a kernel, it is easy to build non-linear variants of simple linear algorithms that are based on inner products. This is called the kernel trick in the machine learning literature.

1.2.3 Kernel types

There are several types of kernel functions that were proposed in the literature. Each of these kernel functions is proposed for a specific purpose, for instance, text classification, pattern recognition...etc. Listing all kernel functions is out of the scope of this thesis. Below is a list of some kernel functions that we have used in our studies. Appendix C lists some of the most common kernel function used in the literatue.

Linear Kernel

Linear Kernel is used when the data is Linearly separable, that is, it can be separated using a single Line. It is one of the most common kernels to be used. It is mostly used when there are a large number of features in a particular dataset. One of the examples where there are a lot of features, is Text Classification, as each alphabet is a new feature.

The Linear kernel is the simplest kernel function. It is given by the inner product <x,y> plus an optional constant c.

$$K(x,y) = x^T y + c \tag{1.12}$$

The advantages of using linear kernel are:

- Training with a linear kernel is faster than with any other Kernel.
- When training with a Linear Kernel, only the optimisation of the c regularisation parameter is required. On the other hand, when training with other kernels, there is a need to optimise other parameter as well, which means that performing a grid search will usually take more time.

Polynomial Kernel

In machine learning, the polynomial kernel is a kernel function commonly used with the kernelized models (such as SVMs), that represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models. Intuitively, the polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these. Such combinations are known as interaction features. The (implicit) feature space of a polynomial kernel is equivalent to that of higher feature space, but without the combinatorial blowup in the number of parameters to be learned.

The Polynomial kernel is a non-stationary kernel. Polynomial kernels are well suited for problems where all the training data is normalized. Adjustable parameters are the slope alpha, the constant term c and the polynomial degree d.

$$K(x,y) = (\alpha x^T y + c)^d \tag{1.13}$$

Gaussian Kernel

The Gaussian kernel is an example of radial basis function kernel.

$$K(x,y) = exp\left(-\frac{\|x-w\|^2}{2\sigma^2}\right) \tag{1.14}$$

The adjustable parameter sigma plays a major role in the performance of the kernel, and should be carefully tuned to the problem at hand. If overestimated, the exponential will behave almost linearly and the higher-dimensional projection will start to lose its non-linear power. In the other hand, if underestimated, the function will lack regularization and the decision boundary will be highly sensitive to noise in training data.

1.3 Facial expression recognition

Communication between human beings can be of two kinds: verbal (audible) through the voice, or non-verbal through movements, body posture, gestures and facial expressions. According to [Dubey und Singh, 2016], Mehrabian [Mehrabian, 2008] states that the first form of communication can only contribute with 7% in the transmission of the message, while 38% of communication is done through vocal parts in the form of intonation. This means that the remaining 55% is done through non-verbal ways of communication, mainly through facial expressions. Therefore, the ability of recognizing facial expressions is key for a better communication. Facial expressions are either viewed as dimensions (continuous representation) or viewed as categorical groups with well-defined boundaries (discrete representation). Continuous representation uses dimensional space such as arousal and valence to describe emotion. While the discrete representation categorizes facial expressions through seven basic emotions, namely happiness, sadness, surprise, fear, disgust, anger, and neutral. These expressions are identified according to the movement of the facial muscles, mainly forming the parts of the eyebrows, mouth, nose and eyes. Ekman [Ekman und Friesen, 1971] proposed a purely objective system for coding facial expressions by quantifying muscle movement. This system, called FACS (facial action coding system). is made up of 44 action units. Each unit action (AU) corresponds to a movement of the facial muscles. The intensity of the muscle contraction is also coded on five levels. The basic emotions defined by Ekman were described by prototypes specific to each expression of emotion. However, a big problem with this system is finding professionals who judge AU and its expression, which can be difficult and expensive. The recognition of facial expressions is useful in various fields such as human machine interaction, games, alert systems and monitoring of patients, especially those who find it difficult to speak like disabled persons and autistic, to detect feelings of pain for example. The automatic recognition of facial expressions is not a recent field of research, in fact, the first research in this field dates back to a little over twenty years. However, new advances have been made in the last five years in the field of object recognition in general, have further revived research in the field of facial expressions recognition.

In the early days of facial expression recognition research, researcher would develop a small set of images with a limited number of people, which they used as a database for learning and testing. This method has a strong chance of leading to over-learning. At the end of the 1990s, some researchers began to develop fairly large and varied databases. The facial expression databases are characterized by the number of subjects photographed, the environment implemented for this purpose, the recording media as well as the annotation. Subjects can be of different sex, age and origin. The environment implemented varies according to the equipment used for taking and recording images and videos: the number of cameras (in color or in grayscale) and flash used, their arrangement (front or profile), the resolution used, the number of images per second for the videos and the fact that the expressions are spontaneous or posed. The annotation is either automatic, semi-automatic or completely manual. We have distinguished two types of databases. The first type of databases is implemented in laboratories with specific subjects and controlled conditions, for instance [Kanade u. a., 2000, Zhao u. a., 2011, Bacivarov, 2009, Yin u. a., 2006, Sim u. a., 2002, Gross u. a., 2010, Pantic u. a., 2005, Lyons u. a., 1998, McKeown u. a., 2011]. The second type of databases is made up of images collected from the Internet, for example [Mollahosseini u. a., 2017, Dhall u. a., 2012; 2011, Fabian Benitez-Quiroz u. a., 2016, Li u. a., 2017, Zhang u. a., 2018b, Goodfellow u. a., 2013]. The former allow to produce images of good quality and with a better annotation but are of small size. Whereas the latter allow to have a gigantic number of data but on the contrary are of poorer quality.

The facial expression recognition system, like almost any conventional image recognition system, consists of three major steps: pre-processing, feature extraction, and classification. The pre-processing step is a very important step in the process of recognizing facial expressions. In fact, this step consists, first of all, in acquiring the images that it is desired to process, either in static form or in the form of image sequences. These images are often grayscale, although color images can convey more information about emotions such as blushing. Then, it is necessary to eliminate the zones of the image which are not of interest while keeping the parts which will be used in a subsequent processing and to normalize the resulting image which makes it possible to obtain a uniform size and an alignment. correct image. The feature extraction step extracts the significant shapes from the image, which constitute the parts of the face which are considered to be determining in the expression recognition step. Compared to the original image, feature extraction dramatically reduces the image information, which provides an advantage in terms of storage and processing. The classification of expressions and the method of extracting the characteristics of the expression are closely related. The categorization of expressions is performed by a classifier. The entry into the classifier is a set of features extracted from the facial region in the previous step. The set of characteristics is formed to describe the expression of the face. Classification requires supervised learning from labeled data. After training is complete, the classifier can recognize the input images by assigning them a particular class label or point in the continuous representation space.

The classic facial expression recognition approach has been widely used over the past two decades with different methods at all levels giving varying results. However, in the last five years a new approach, Deep Learning, has been used with impressive results. Deep Learning is a set of end-to-end methods, enabling a machine to be supplied with raw data and automatically discovering the representations necessary for detection or classification, with several levels of representation, obtained by composing simple modules. One of the most used deep learning techniques are Convolutional Neural Networks (CNNs) which have been extremely successful in computer vision applications. This great success encouraged the computer vision community to tackle more challenging tasks in this field. One of these challenging tasks is the fine-grained recognition. It consists of discriminating categories that were considered previously as a single category and have only small subtle visual differences(e.g. bird species). Facial expression recognition is considered one of most challenging fine-grained recognition problems. Indeed, the difference in facial expression categories relies on small subtle areas in the facial images like the mouth, eyebrows and the noise. To overcome this issue, facial expression recognition systems must be able to recognise this subtle differences efficiently. Researchers are trying to overcome this problem by either increasing the network size or by employing more complex functions. In the first case, researchers are continuously trying to enhance CNNs by increasing their depth (number of layers) or width (size of the output of each layer). Even though by doing so the performance of the network is effectively enhanced, it can not be a longstanding solution. Indeed, these methods drastically increase the number of weights and the network complexity. Therefore, the resulting models can only be used on powerful devices. In the second case, the focus is more on computation. Many researchers incorporated more complex functions in CNN, instead of simple linear functions, at different levels. These methods have the benefits of being less memory consuming, even though they are harder to train.

Facial expression recognition methods are divided into two categories: Macro-expression recognition methods and Micro-expression recognition methods.

1.3.1 Macro-expression recognition methods

These methods are macroscopic methods that measure the displacements of certain parts of the face such as the eyebrows or the corners of the mouth. In other words, they consist in calculating the geometric distance between the facial action units extracted by searching and tracking crucial points in the facial region. Examples of such algorithms are Active Shape Model (ASM) [Cootes u. a., 1995] and Active Appearance Model (AAM) [Cootes u. a., 2001, Pantic und Rothkrantz, 2000] and Descriptor Scale Invariant Transformation. According to [Gharsalli, 2016] there are two kinds of macro-expression recognition methods: macro-expression recognition methods based on landmarks and macro-expression recognition methods based on models.

1. **Macro-expression recognition methods based on landmarks:** landmarks describe the expressions either by their displacements, or by the variation of the distances between them. The displacement of the characteristic points requires two steps, namely the extraction of their positions and their monitoring (Figure 1.3).

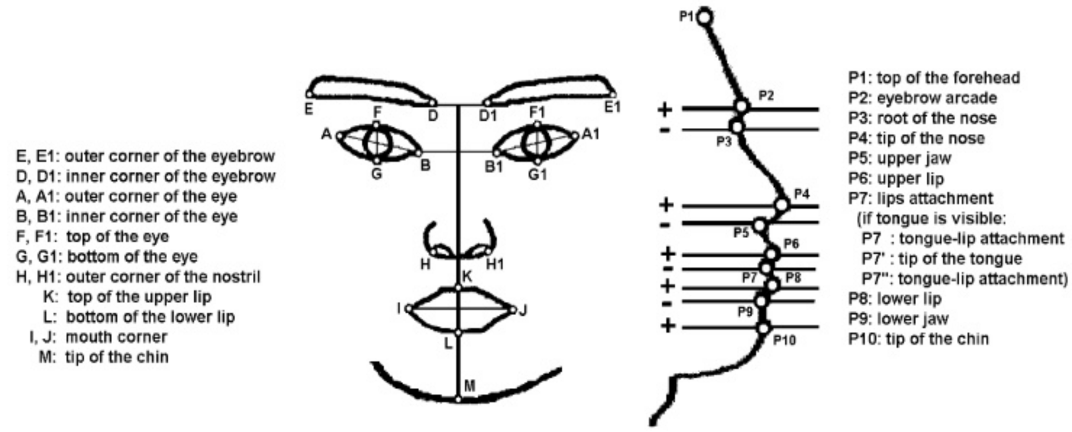


Figure 1.3 – *Macro-expression recognition methods based on landmarks.*

2. **Macro-expression recognition methods based on the models:** patterns for detecting muscle deformities and changes in feature states are used for facial expression recognition. The definition and initialization of these models is usually done on a neutral face. They often encode the movement of muscles either by moving nodes placed on facial features, or by transforming it into 3D information, which provides information on the intensity of the movement (Figure 1.4). The effectiveness of these methods, however, becomes critical when it comes to weak muscle movement.

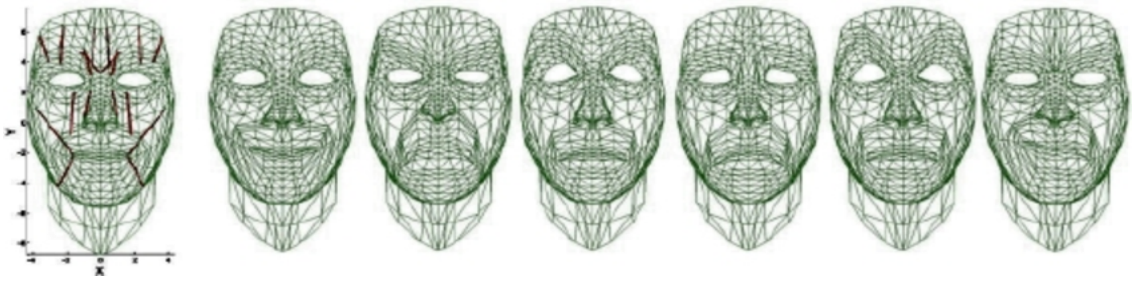


Figure 1.4 – *Macro-expression recognition methods based on the models.*

1.3.2 Micro-expression recognition methods

These microscopic methods describe the change in face texture according to pixel properties, when a particular action is performed, such as bumps, contour, wrinkles, the region around the mouth and eyes. With the advance made in computing with GPUs and the availability of the later to a larger number of researchers, the computer vision community started focusing on these microscopic methods rather than the macroscopic methods. These microscopic methods allow the use of number of features that were neglected in the macroscopic methods like bumps, contour, wrinkles and blush, especialy with RGB images. Micro-expression recognition methods may be expressed differently in different papers. Sometimes it is refered to as fine-grained expression recognition. It consists of discriminating categories that were considered previously as a single category and have only small subtle visual differences(e.g. bird species). Facial expression recognition is considered one of most challenging fine-grained recognition problems. Indeed, the difference in facial expression categories relies on small subtle areas in the facial images like the mouth, eyebrows and the noise. We prefer to use this term as it is more general than FER, since the techniques used in this field can be used in a wide range of computer vision tasks.

In [Bai u. a., 2021], the author investigates the effects of using video motion magnification methods based on amplitude and phase, respectively, to amplify small facial movements. They hypothesise that this approach will assist in the micro-expression recognition task. To this end, they apply the pre-trained VGGFace2 model with its excellent facial feature capturing ability to transfer learn the magnified micro-expression movement, then encode the spatial information and decode the spatial and temporal information by Bi-LSTM model. Moreover, Grad-CAM is utilised to map the model and visually explain the operating mechanism of the spatio-temporal network.

In [Saeed, 2021], the authors investigate the utility of micro-expressions as a soft biometric for person recognition. The proposed system is based on the fusion of traditional facial features that model the facial appearance with soft biometric features that model the micro-expressions in an image sequence. They tested a texture-based traditional feature extraction technique, two motion-based soft biometric techniques, and several fusion methods at feature, rank, and decision level.

In [Li u. a., 2020] paper firstly proposes a new method to detect the apex frame by estimating pixel-level change rates in the frequency domain. With frequency information, it performs more effectively on apex frame spotting than the currently existing apex frame spotting methods based on the spatio-temporal change information. Secondly, with the apex frame, this paper proposes a joint feature learning architecture coupling local and global information to recognize micro-expressions, because not all regions make the same contribution to micro-expressions recognition and some regions do not even contain any emotional information. More specifically, the proposed model involves the local information learned from the facial regions contributing major emotion information, and the global information learned from the whole face. Leveraging the local and global information enables the model to learn discriminative micro-expressions representations and suppress the negative influence of unrelated regions to micro-expressions.

1.3.3 Hybrid methods

Micro-expression recognition methods are often criticized for a lack of movement representation of facial features. Macro-expression recognition methods, representing only the shape and geometric movements neglect information such as transient wrinkles which can be essential characteristics for the differentiation between emotions. There is, however, a combination of the two methods. The latter can be performed either directly with models presenting the two pieces of information such as the Active Appearance Model (AAM).

A second alternative is applied to describe both the macro and micro-expression recognition. It consists in extracting each piece of information individually by a dedicated method for this purpose and then applying a fusion. In general two merging schemes are used, namely an upstream scheme and a downstream scheme [Gharsalli, 2016].

1. **Upstream schema:** it combines the descriptors of different types of information before going to the classification stage. In this case a pre-processing is applied to the two pieces of information in order to be able to merge them into the same vector. The latter is then used as input data by the classification method (Figure 1.5)

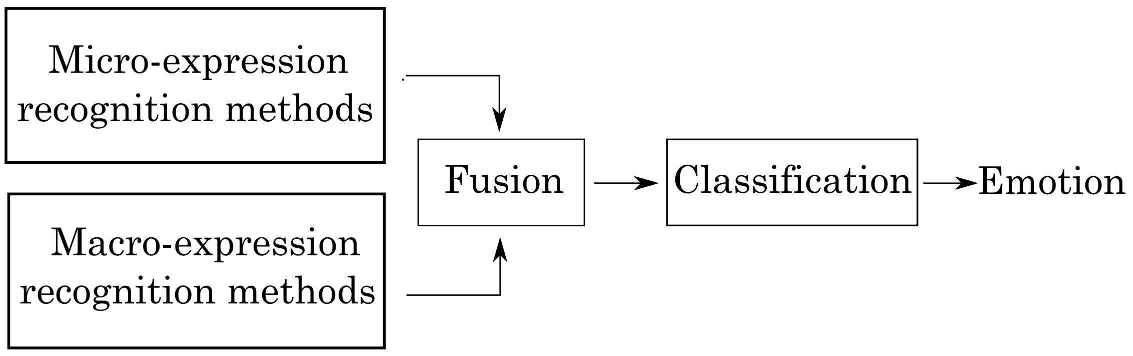

Micro-expression recognition methods
Macro-expression recognition methods
Fusion
Classification
Emotion

Figure 1.5 – *Upstream schema fusion method..*

2. **Downstream schema:** it combines the descriptors after the classification step. For each type of descriptors extracted a classification is applied. The decisions from the classification step are then combined into a merging entity (Figure 1.6)

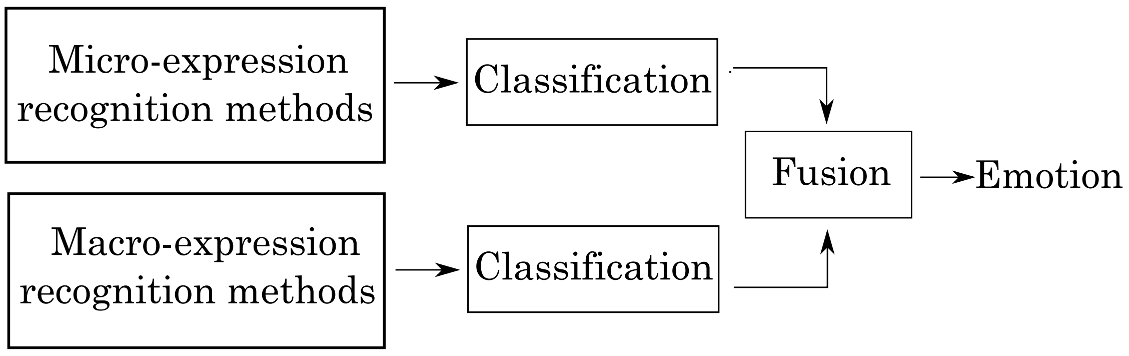

Micro-expression recognition methods
Classification
Macro-expression recognition methods
Classification
Fusion
Emotion

Figure 1.6 – *Downstream schema fusion method.*

1.3.4 Facial expression recognition datasets

In the early days of facial expression recognition research, each researcher developed a small set of images of a limited number of people as a database for training and testing. This method has a strong chance of leading to over-fitting, in addition to the impossibility of comparing the work carried out because they are tested on different proprietary bases. At the end of the 90s, some researchers began to develop fairly large and varied databases, accessible to researchers and the general public, which can be used as a "benchmark". This allowed the researchers to compare the different techniques used.

The databases of facial expressions are characterized by the number of subjects photographed, the environment implemented for this purpose, the recording medium and the annotation. The subjects can be of different sex, age and origins. The environment implemented varies according to the equipment used for taking and recording images and videos: the number of cameras (in color or grayscale) and cameras used, their arrangement (frontal or profile), the resolution used, the number of frames per second for the videos and whether the images they contain are spontaneous or posed. Annotation is either automatic, semi-automatic or completely manual.

We have distinguished two types of databases: databases implemented in laboratories with specific subjects and conditions and in-the wild databases. The first allow to produce images of good quality and with better annotation but are of restricted size. While the latest allow to have a gigantic amount of data but on the contrary are of lower quality.

For our experiments, we have used three well-known in the wild FER datasets, namely RAF-DB, ExpW and FER2013. Below is a brief description of these datasets and an illustration (Fig. 1.7) of their content. Some other datasets are described in table 1.2.

- The RAF-DB [Li u. a., 2017] stands for the Real-world Affective Face DataBase. It is a real-world dataset that contains 29,672 highly diverse facial images, down-

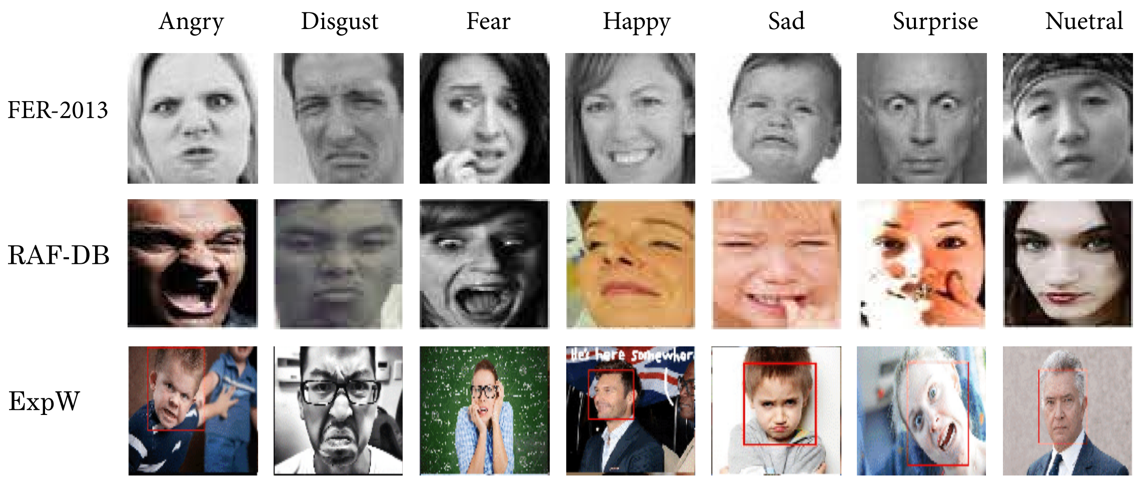


Figure 1.7 – *An overview of the content of the datasets used. All these datasets categorize emotions into seven classes, namely Anger, Disgust, Fear, Happy, Sad, Surprise and Neutral.*

loaded from the Internet. With manually crowd-sourced annotation and reliable estimation, seven basic and eleven compound emotion labels are provided for the samples. This dataset is divided in training and validation subsets.

- The ExpW [Zhang u. a., 2018b] stands for the EXPression in-the-Wild dataset. It contains 91,793 facial images downloaded using Google image search. Each of the face images, was manually annotated as one of the seven basic expression categories.
- The FER2013 database was first introduced during the ICML 2013 Challenges in Representation Learning [Goodfellow u. a., 2013]. This database contains 28,709 training images, 3,589 validation images and 3,589 test images with seven expression labels: fear, happiness, anger, disgust, surprise, sadness and neutral.

1.4 Experimental setting

In this section we will describe the experimental setting that we have used in all our experiments. We will first describe the hardware configuration, then the software setting and finally the metrics we have used to evaluate our models.

1. **Hardware configuration:** For our experiments, we have used the following hardware configuration:
   - **CPU:** Intel Core i7-8700K Desktop Processor 6 Cores up to 4.7GHz Turbo Unlocked LGA1151 300 Series 95W.
   - **GPU:** MSI GAMING GeForce GTX 1060 6GB GDRR5 192-bit HDCP Support DirectX 12 Dual TORX 2.0 Fan VR Ready Graphics Card (GTX 1060 GAMING X 6G).

- **RAM:** 16 Gb of memory.

2. **Software configuration:** All our experiments are implemented in tensorflow 2.0. However, they can be run in earlier versions of tensorflow, except for the improved bilinear models introduced in chapter 2. This technique can be used with tensorflow 1.4 and later version. Since the matrix logarithm function and the matrix square-root function are only implemented in these version of tensorflow, unless they are built from scratch in version 1.3 or earlier.

3. **Metric:** For all our experiments, we used the accuracy rate. Accuracy is one of the most popular metrics in multi-class classification and it is directly computed from the confusion matrix [Grandini u. a., 2020].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.15}$$

   The formula of the Accuracy considers the sum of True Positive and True Negative elements at the numerator and the sum of all the entries of the confusion matrix at the denominator. True Positives and True Negatives are the elements correctly classified by the model and they are on the main diagonal of the confusion matrix, while the denominator also considers all the elements out of the main diagonal that have been incorrectly classified by the model.

Table 1.2 – *An overview of the facial expression datasets.*

Database	Samples	Subject	Collection condition	Elicitation method	Expression distribution
CK+ [32]	593 image sequences	123	Lab	P & S	7 BE + contempt
MMI [35]	740 images and 2,900 videos	25	Lab	P	7 BE
JAFFE [36]	213 images	10	Lab	P	7 BE
TFD [37]	112,234 images	N/A	Lab	P	7 BE
FER-2013 [21]	35,887 images	N/A	Web	P & S	7 BE
AFEW 7.0 [23]	1,809 videos	N/A	Movie	P & S	7 BE
SFEW 2.0 [22]	1,766 images	N/A	Movie	P & S	7 BE
Multi-PIE [38]	755,370 images	337	Lab	P	Smile, surprised, squint, disgust, scream and neutral
BU-3DFE [39]	2,500 3D images	100	Lab	P	7 BE
BU-4DFE [40]	606 3D sequences	101	Lab	P	7 BE
Oulu-CASIA [33]	2,880 image sequences	80	Lab	P	six BE without neutral
RaFD [41]	1,608 images	67	Lab	P	7 BE + contempt
KDEF [42]	4,900 images	70	Lab	P	7 BE
EmotioNet [43]	1,000,000 images	N/A	Internet	P & S	23 BE or CE
RAF-DB [34], [44]	29672 images	N/A	Internet	P & S	7 BE and 12 CE
AffectNet [45]	450,000 images (labeled)	N/A	Internet	P & S	7 BE
ExpW [46]	91,793 images	N/A	Internet	P & S	7 BE
4DFAB [47]	1.8 million 3D faces	180	Lab	P & S	7 BE

P = posed; S = spontaneous; BE = basic expressions; CE = compound expressions

2 Improved Bilinear Model For Facial Expression Recognition

Contents

Even though convolutional neural networks (CNN) achieved impressive results in several computer vision tasks, they still do not perform as well in FER. Many techniques, like bilinear pooling and improved bilinear pooling, have been proposed to improve the CNN performance on similar problems. The accuracy enhancement they brought in multiple visual tasks, shows that their is still room for improvement for CNNs on FER. In this chapter, we propose to use bilinear and improved bilinear pooling with CNNs for FER. This framework has been evaluated on three well known datasets, namely ExpW, FER2013 and RAF-DB. It has shown that the use of bilinear and improved bilinear pooling with CNNs can enhance the overall accuracy to nearly 3% for FER and achieve state-of-the-art results.

2.1 Introduction

A CNN is mainly a stack of three different types of layers: convolution layers, pooling layers and fully-connected layers. Each of these types of layers a perform specific task. Convolution layers are the core building block of a CNN leveraging the fact that an input image is composed of small details, or features, and create a mechanism for analyzing each feature in isolation, which makes a decision about the image as a whole. Pooling layers, on the other hand, are used for the gradual spatial down-sampling of the feature map. This results in reducing the number of parameters and thus decreases both the consumption of the memory and the complexity of computing. In addition, pooling layers widen the receptive field size of the intermediate neurons which allow the latter to receive data from a larger area of the image. These two layers are usually used in alternation until getting the most size-effective representative feature which is finally fed into a fully connected neural network in order to take a final classification decision. However, the fully connected layers are prone to overfitting, thus hampering the generalization ability of the overall network. Many techniques were proposed in the literature to overcome this problem. For instance, Dropout is proposed by [Hinton u. a., 2012] as a regularizer which randomly sets half of the activations to the fully connected layers to zero during training. It has improved the generalization ability and largely prevents overfitting. Other methods consist of applying pooling before the fully connected layers.

There are two groups of pooling generally used in CNNs. The first one is local pooling, where the pooling is performed from small local regions (e.g., 2×2) to downsample the feature maps. The second one is global pooling, which is performed from each of the entire feature map to get a scalar value of a feature vector for image representation. This representation is then passed to the fully connected layers for classification. In this chapter, the focus is on later form of pooling, while the former form of pooling is discussed in the next chapter.

The most used global pooling methods in CNNs are global average pooling and global max pooling. Both of these methods take as input a tensor of size $(H \times W \times C)$, where H is the height of the tensor, W its width and C its channel depth. This input tensor is downsized into an output vector of size C (i.e. it keeps a single value for each feature map). The difference between global max and global average pooling is that the former keeps only the maximum values over the channel axis while the later computes the average value. Figure 2.1 shows the process of global Max pooling method. Global average pooling work in a similar manner, yet instead of keeping the maximum value it computes the average. Both of these methods present some weakness. For example, global Max pooling only keeps the largest input values assuming that the rest of values are not representative and do not bring relevant information. This assumption however is not always true, especially in the last layers of the network where even the small values represent a very relevant information. Therefore global max pooling dramati-

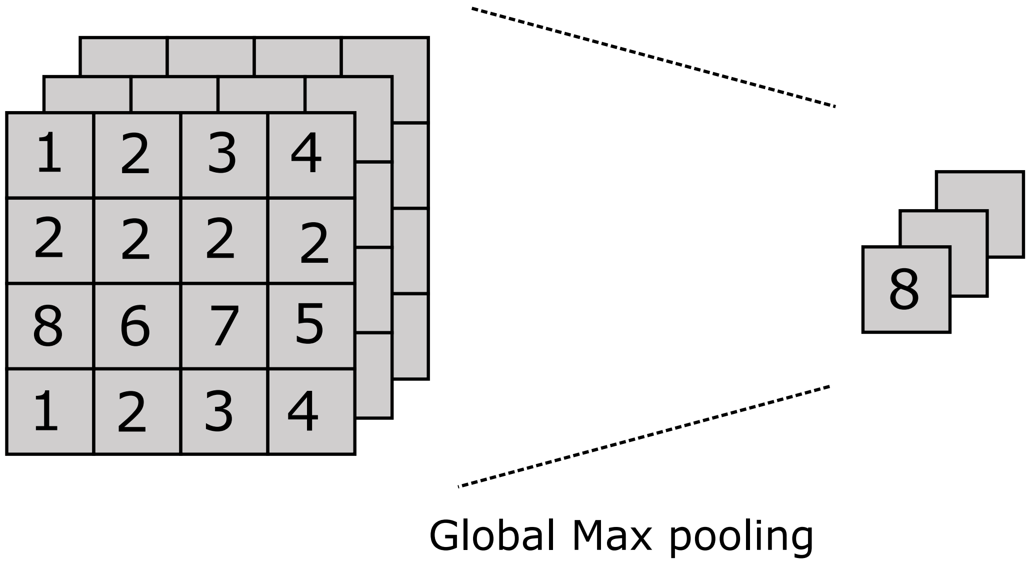


Figure 2.1 – *The global Max pooling method keeps only the maximum values over the channel axis. Global average pooling work in a similar manner, yet instead of keeping the maximum value it computes the average.*

cally reduces the amount of useful information in the forward pass. Moreover, global max pooling wrongly affects the learning of the network in the backward pass, since only one branch is activated in each input neighborhood. In a global average pooling layer, all the inputs equally contribute to the output computation. This causes a constant and gradual attenuation of the contribution of individual neurons in the backward and forward passes [Saeedan u. a., 2018]. To overcome the loss of information limitations, several pooling methods where proposed. Among these recent techniques, we are especially interested in bilinear CNN models and their ameliorations such as compact bilinear pooling [Gao u. a., 2016] and the improved bilinear pooling [Lin und Maji, 2017].

Bilinear CNN model is a combination of two CNNs A and B that takes as input the same image and output two feature maps. These feature maps are then multiplied at each location using tensor product. The result is pooled to obtain a global image descriptor of the image. The latter is passed to a classifier throughout make a prediction. Compared to single CNNs, bilinear CNN models have shown to achieve very good results on various visual tasks. For instance, semantic segmentation, visual questions answering and fine-grained recognition. In this chapter, in addition of using bilinear CNN models, we propose to use an improved bilinear pooling with CNNs models for FER. In this framework, various ways of normalization are used to improve the accuracy, including the matrix square root, element-wise square root and L2 normalization.

The remainder of this chapter is organized as follow: Section 2.2 reviews similar works that have been done on FER and bilinear CNN models. Section 2.3 gives more details about this approach. Section 2.4 presents our experiments, datasets and results; and Section 2.5 concludes the chapter.

2.2 Related work

Several methods have been proposed to improve the performance of CNNs. In [Lin u. a., 2015] a bilinear pooling method for fine-grained recognition was proposed. Inspired from the second order pooling model introduced by [Tenenbaum und Freeman, 2000], this model can capture higher interaction between image locations, which makes the model more discriminant than a simple model. This method have been used for FER by Zhou et al. [Zhou u. a., 2018] and noticed that they significantly outperformed their respective baselines. However, these models are high dimensional and could be impractical for a multitude of image analysis. In [Zhang u. a., 2019b] the authors introduced factorized bilinear pooling (FBP) to deeply integrate the features of audio and video. The features are selected through the embedded attention mechanism from respective modalities to obtain the emotion-related regions. Hierarchical Bilinear Pooling was proposed by [Yu u. a., 2018] in which a cross-layer bilinear pooling approach is proposed to capture the inter-layer part feature relations, which results in superior performance compared with other bilinear pooling based approaches. Moreover, they proposed a novel hierarchical bilinear pooling framework to integrate multiple cross-layer bilinear features to enhance their representation capability. Yu et al [Yu u. a., 2017] developed a Multi-modal Factorized Bilinear (MFB) pooling approach to efficiently and effectively combine multi-modal features, which results in superior performance for visual question answering compared with other bilinear pooling approaches. For fine-grained image and question representation, they developed a 'co-attention' mechanism using an end-to-end deep network architecture to jointly learn both the image and question attentions. Combining the proposed MFB approach with co-attention learning in a new network architecture provides a unified model for visual question answering. Zhang et al [Zhang u. a., 2019a] introduced a Local Temporal Bilinear Pooling which is used in intermediate layers of a temporal convolutional encoder-decoder net. In contrast to previous work, the proposed bilinear pooling is learnable and hence can capture more complex local statistics than the conventional counterpart. In addition, they introduced exact lower dimension representations of their bilinear forms, so that the dimensionality is reduced without suffering from information loss nor requiring extra computation.

In [Gao u. a., 2016] two compact bilinear representations of these models have been proposed. They reached results as the full bilinear representation, yet with only a few thousand dimensions. This compact representations have also been used, by [Nguyen u. a., 2018], in a multi-modal emotion recognition, combining facial expressions and voice sound. It was also used by [Chetouani u. a., 2020] to classify patterns of ceramic sherds by combining deep learning-based features extracted from some pre-trained Convolutional Neural Network (CNN) models. Fukuri et al [Fukui u. a., 2016] proposed utilizing Multimodal Compact Bilinear pooling (MCB) to efficiently and expressively combine multimodal features on the visual question answering and visual grounding tasks. A similar method to efficiently reduce the dimension of bilinear pooling descriptors was proposed by [López-Sánchez u. a., 2020] by performing a Random Projection. Conveniently, this is achieved without ever computing the high-dimensional descriptor explicitly. The latter was further generalized in the form of Taylor series kernel in [Cui u. a., 2017]. The proposed method captures high order and non-linear feature interactions via compact explicit feature mapping. The approximated representation is fully differentiable, and the kernel composition can be learned together with a CNN in an end-to-end manner. Another compact form of bilinear pooling was proposed by [Liao u. a., 2019] called Squeezed Bilinear Pooling. It is a supervised selection based method to decrease both the computation and the feature dimension of the original bilinear pooling. Different from currently existing compressed second-order pooling methods, the proposed selection method is matrix normalization applicable. Moreover, by extracting the selected highly semantic feature channels, they proposed the Fisher-Recurrent-Attention structure and achieved state-of-the-art fine-grained classification results among the VGG-16 based models. In [Wei u. a., 2018] the authors proposed an alternative pooling method which transforms the CNN feature matrix to an orthonormal matrix consists of its principal singular vectors. Geometrically, such orthonormal matrix lies on the Grassmann manifold, a Riemannian manifold whose points represent subspaces of the Euclidean space. Similarity measurement of images reduces to comparing the principal angles between these "homogeneous" subspaces and thus is independent of the magnitudes and correlations of local CNN activations. In particular, they demonstrate that the projection distance on the Grassmann manifold deduces a bilinear feature mapping without explicitly computing the bilinear feature matrix, which enables a very compact feature and classifier representation.

Lin et al. have furthered their bilinear CNN model, by applying matrix normalization functions. Two matrix functions have been used, namely matrix logarithm and matrix square-root. All these methods are plugged at the end of the network, right between the convolution layers and the fully connected layers. They act as a basis expansion layers, increasing thereby the discrimination power of the fully connected layers, This discrimination power is back-propagated through the convolution layers. These methods have attracted increasing attentions, achieving better performance than classical first-order networks in a variety of tasks. Even-thought these methods increase the CNN performance, they are unable to learn by themselves and rely entirely on the CNN architecture. Furthermore, effectively introducing higher-order representation in earlier pooling layers, for improving non-linear capability of CNNs, is still an open problem.

To the best of our knowledge, improved bilinear CNN models have never been used for FER. We believe that these models can enhance the CNN performance also for FER, given that FER is very similar to fine grained recognition. In the following sections, we will give more details about the bilinear and the improved bilinear CNN models. We will also explore the effect of using them on FER.

2.3 Approach

In this section we will describe the approach we used for our FER task. This technique, called bilinear CNN model, was inspired by Lin et al. [Lin und Maji, 2017]. It performed very well on fine-grained visual recognition tasks, and was later improved in [Lin und Maji, 2017]. We will describe bellow in more details bilinear CNN models and the improved version.

2.3.1 Bilinear CNN models

Bilinear pooling models were first introduced by Tenenbaum and Freeman [Tenenbaum und Freeman, 2000]. Also called second order pooling models, they were used to separate style and content. These models have been later used for fine grained recognition and semantic segmentation using both hand-tuned and learned features.

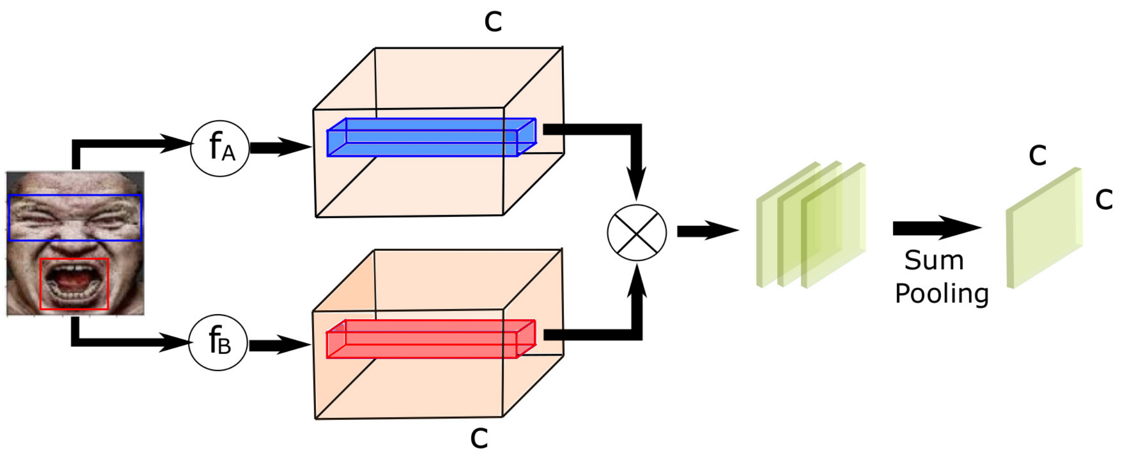


Figure 2.2 – *A bilinear model*

For image classification, we can generally formulate a bilinear model B as a quadruple $B(f_A, f_B, P, C)$ (Figure 2.2). Where f_A, and f_B, are feature functions, P a pooling function and C a classification function. A feature function takes an image Img and a location $l \in Loc$ as inputs and produces a feature vectors, for each location in Loc, as follows:

$$f(l, Img) \mapsto \mathcal{R}^c \tag{2.1}$$

We then combine these feature functions outputs vectors using the tensor product (equation 2.2) at each location. Here, A and B are feature vectors produced by the feature functions f_A, and f_B respectively.

$$A \otimes B = \begin{bmatrix} a_1 \\ a_2 \\ . \\ . \\ a_c \end{bmatrix} \otimes \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ y_b \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \dots & a_1 b_c \\ a_2 b_1 & a_2 b_2 & \dots & a_2 b_c \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ a_n b_1 & a_n b_2 & \dots & a_c b_c \end{bmatrix} \tag{2.2}$$

Formally, the bilinear feature combination of f_A and f_B at a location $l \in Loc$ is given by:

$$Bilinear(l, Img, f_A, f_B) = f_A(l, Img) \otimes f_B(l, Img) \tag{2.3}$$

The pooling function P combines the bilinear features throughout the different locations in the image (equation 2.4), which will produce a global image descriptor. One of the most used pooling functions are the sum and the max-pooling functions of all the bilinear features. Both functions ignore the location of the features and are hence orderless [Lin u. a., 2015].

$$P(Loc, Img, f_A, f_B) = \sum_{l \in Loc} f_A(l, Img) \otimes f_B(l, Img) \tag{2.4}$$

A natural candidate for the feature function f is a CNN consisting of a succession of convolutional and pooling layers. According to [Lin u. a., 2015], the use of CNNs is beneficial at many levels. It allows to use pre-trained CNNs in which we take only the convolutional layers including non-linearities as feature extractors. This can be beneficial specially when domain specific data is scarce. Another benefit of using only the convolutional layers is that the resulting CNN can process images of an arbitrary size in a single forward-propagation step. It produces outputs indexed by the location in the image and feature channel, in addition of reducing considerably the network's parameters number. Finally, the use of CNNs for a bilinear model allows this model to be trained in an end-to-end fashion. This technique has been used in a number of recognition tasks. For instance object detection, texture recognition and fine-grained classification and shown to give very good results.

Lin et al.[Lin u. a., 2015] proposed bilinear CNN Models for fine-grained visual recognition (Figure 2.3). The model consists of two CNNs, each trained to recognize special features. The resulting feature maps are sum-pooled to aggregate the bilinear features across the image. The resulting bilinear vector is then passed through signed square-root step, followed by L2 normalization, which improves performance in practice. Finally, the result will be fed to a classifier.

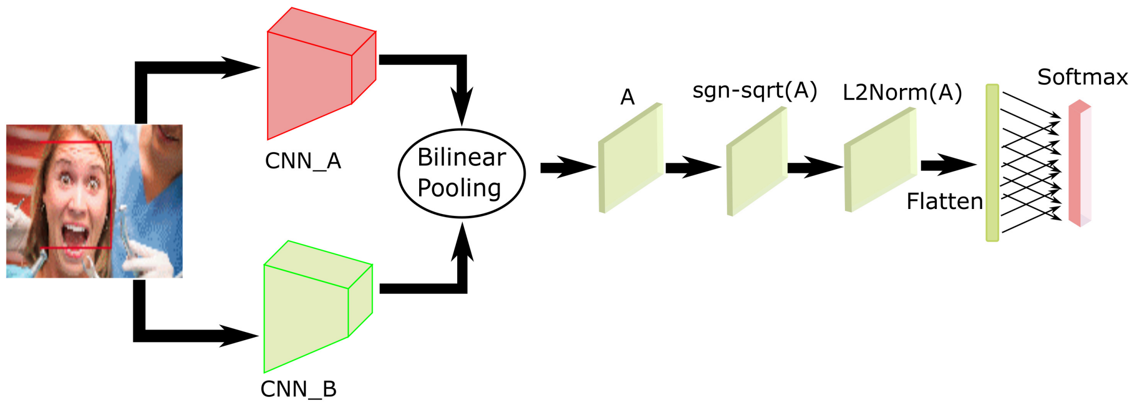


Figure 2.3 – *Bilinear CNN architecture.*

2.3.2 Improved bilinear pooling

Lin et al.[Lin und Maji, 2017] have also investigated various ways of normalization to improve the representation power of their bilinear model. In particular, a class of matrix functions were used to scale the spectrum (eigenvalues) of the co-variance matrix resulting of the bilinear pooling. One example of such normalization is the matrix-logarithm function defined for Symmetric Positive Definite (SPD) matrices. It maps the Riemannian manifold of SPD matrices to an Euclidean space that preserves the geodesic distance between elements in the underlying manifold (Figure 2.4). An other normalization is the matrix square-root normalization which offers significant improvements and outperforms the matrix logarithm normalization when combined with element-wise square-root and L2 normalization. This improved the accuracy by 2-3% on a range of fine-grained recognition datasets leading to a new state-of-the-art.

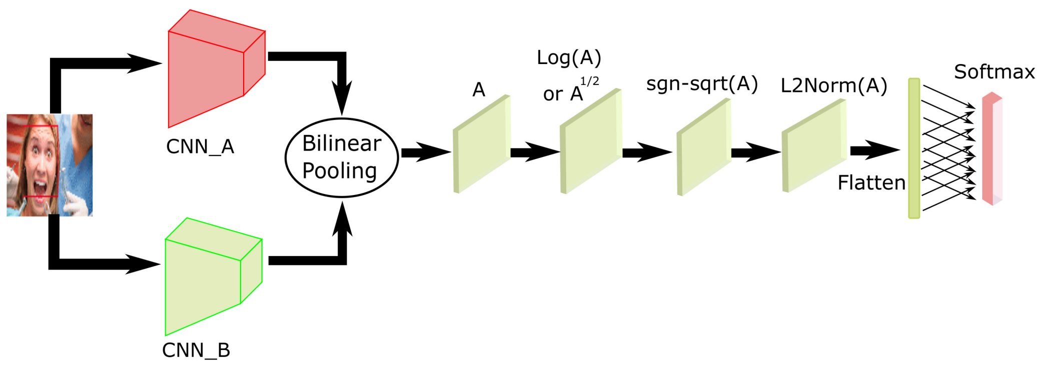


Figure 2.4 – *Improved Bilinear CNN architecture.*

The strength of bilinear models relies in the fact that they capture higher interaction between image locations, which makes the model more discriminant than a simple model. This allowed them to achieve impressive results in various image recognition tasks including FER. For instance, bilinear pooling has recently been used for FER in fine-grained manner [Zhou u. a., 2018]. Compact bilinear pooling has also been used in a multi-modal emotion recognition combining facial expressions and voice sound [Nguyen u. a., 2018]. But as far as we know, the improved bilinear pooling has never been used for FER. In the following section, we will explore the effect of using an improved bilinear CNN for FER. We will also implement a bilinear CNN to further appreciate the enhancement that can the improved bilinear CNN provide.

2.4 Experiments

In this section we will give more details about the experiments we performed in order to evaluate the approach described above. First, we give a brief description of the datasets we have used. After that, we describe architecture of the used models and training process. Finally, we discuss the obtained results.

2.4.1 Datasets

Our experiments have been conducted on three well-known facial expression datasets, namely the RAF-DB [Li u. a., 2017], ExpW [Zhang u. a., 2018b] and FER2013 [Goodfellow u. a., 2013]. Facial expression datasets contain few classes that are nearly identical, which makes the recognition process more challenging.

In order to have the same dataset structure for all datasets, we divided the validation subset in RAF-DB [Li u. a., 2017] into validation and test subsets by a ratio of 0.5 each. We have also divided ExpW dataset with a ratio of 0.7 for training, 0.15 for validation and 0.15 for test.

2.4.2 Model architecture and Training process

For our experiment, we have used both a VGG-16 pre-trained on ImageNet database and a model built from scratch. For the VGG-16 we only took the convolution layers without the top fully connected ones. We added a batch normalization layer after each convolution layer (this enhances the model's accuracy by nearly 1%). We added only one fully connected layer of size 512 and a final Softmax layer of seven output classes.

On the other hand, our model architecture, as shown in Figure 2.5 is quite simple and can effectively run on cost-effective GPUs. It is composed of five convolutional blocks. Each block consists of a convolution, batch normalization and rectified linear unit activation layers. The use of batch normalization [Zou u. a., 2019] before the activation brings more stability to parameter initialization and achieves higher learning rate. Each of the five convolutional blocks is followed by a dropout layer. In the following we refer to this network architecture as (Model-1).

The only pre-processing which we have employed on all experiments is cropping the face region and resizing the resulting images to 100×100 pixels. We have used Adam optimiser with a learning rate varying from 0.001 to 5e-5. This learning rate is decreased by a factor of 0.63 if the validation accuracy does not increase over ten epochs.

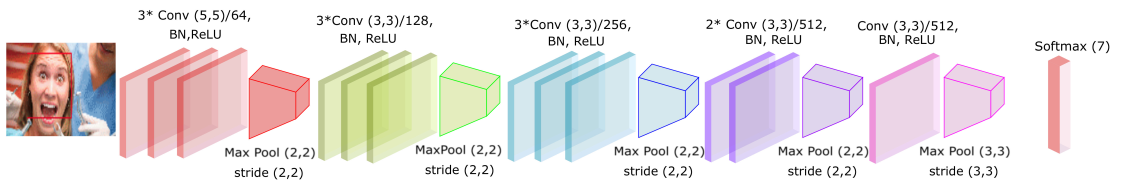

3* Conv (5,5)/64, BN,ReLU
Max Pool (2,2) stride (2,2)
3*Conv (3,3)/128, BN, ReLU
MaxPool (2,2) stride (2,2)
3*Conv (3,3)/256, BN, ReLU
Max Pool (2,2) stride (2,2)
2* Conv (3,3)/512, BN, ReLU
Max Pool (2,2) stride (2,2)
Conv (3,3)/512, BN, ReLU
Max Pool (3,3) stride (3,3)
Softmax (7)

Figure 2.5 – *Base model architecture (Model-1).*

To avoid over-fitting we have first augmented the data using a range degree for random rotations of 20, a shear intensity of 0.2, a range for random zoom of 0.2 and randomly flip inputs horizontally. We have also employed early stopping if validation accuracy does not improve by a factor of 0.01 over 20 epochs.

2.4.3 Ablation Study

This section explores the impact of using bilinear pooling and improved version on the overall accuracy of the two base models (VGG-16 and Model-1). All the following experiments follow the same training process described above.

First we fine tuned the VGG-16 model on the three datasets and trained our model from scratch. Secondly, we took only the convolution part of the two trained models and add bilinear pooling (as shown in Fig 2.3) with the following configurations: a) bilinear pooling on top of VGG-16, b) bilinear pooling on top of Model-1 and c) bilinear pooling on top of both VGG-16 and Model-1. We begin with fine tuning the bilinear pooling part only by freezing the underlying models. After that we train model in an end-to-end fashion. Finally, we repeated the same process of bilinear pooling with the improved version. That is to take the convolution part only of the fine-tuned VGG-16 and Model-1 and add the improved bilinear pooling (as shown in Fig2.4). We followed the same three configurations used for bilinear pooling.

Table 2.1 present the result of the two base models with comparison to these models with bilinear pooling and improved bilinear pooling. The VGG-16 model attains an accuracy rate of 65.23%, 67.61% and 85.23% on FER2013, ExpW and RAF-DB respectively. Whereas Model-1 attains an accuracy of 70.13%, 75.91% and 87.05% respectvely on FER2013, ExpW and RAF-DB.

Table 2.1 – *Accuracy Rates of the proposed approach*

Models	Dataset		
	FER2013	***ExpW***	***RAF-DB***
VGG-16	65.23%	67.61%	85.23%
VGG-16-BP[a]	68.15%	68.82%	85.77%
VGG-16-IBP[b]	68.71%	69.1%	86.34%
Model-1	70.13%	75.91%	87.05%
Model-1-BP[a]	71.63%	76.59%	88.48%
Model-1-IBP[b]	**72.65%**	**77.81%**	**89.02%**
(VGG-16/Model-1)-BP[a]	70.37%	73.57%	86.47%
(VGG-16/Model-1)-IBP[b]	71.22%	74.41%	87.13%

[a]BP: Bilinear Pooling.
[b]IBP: Improved Bilinear Pooling.

On the other hand one can notice that the use of bilinear pooling on top of a model increases considerably its accuracy. As reported in table 2.1, the use of bilinear pooling on to of VGG-16 increases the accuracy for nearly 3% for FER2013 and more than 1% for both ExpW and RAF-DB. Similarly, the use of bilinear pooling on top of Model-1 increases the accuracy for about 1% on all datasets. However using bilinear pooling on top of both models gives an average accuracy rate between the underlying models accuracies. The resulting accuracy rates are 70.37%, 73.57% and 86.47% for FER2013, EwpW and RAF-DB respectively. This is due to the difference in accuracy between the two underlying models in the first place.

Finally, the use improved bilinear pooling increases further the accuracy rate for about 1% for all models with all datasets, compared to bilinear pooling. For instance, the accuracy rate of improved bilinear pooling on top of VGG-16 is 68.71%, 69.1% and 86.34% for FER2013, ExpW and RAF-DB respectively. Similarly, improved bilinear pooling on top of Model-1 gives 72.65%, 77.81% and 89.02% accuracy rates respectively for FER2013, ExpW and RAF-DB. The accuracy rate also increases when using improved bilinear pooling on top of both models. The later gives 71.22%, 74.41% and 87.13% on FER2013, ExpW and RAF-DB respectively.

These results demonstrate that the use of bilinear pooling and specially improved bilinear pooling, in the case of FER problem, are beneficial for the overall accuracy of the model. These techniques enhance the discriminative power of the model, compared to a CNN with only fully connected layers.

2.4.4 Comparison with the State-of-the-Art

In this section, we compare the performance of the bilinear and improved bilinear CNN with respect to several state-of-the-art FER methods. The obtained results are reported in Table 2.2.

Table 2.2 – *Accuracy rate of the proposed approach and state of the art approach*

Models	**Dataset**		
	FER2013	***ExpW***	***RAF-DB***
VGG-16-BP[a]	68.15%	68.82%	85.77%
VGG-16-IBP[b]	68.71%	69.1%	86.34%
Model-1-BP[a]	**71.63%**	**76.59%**	**88.48%**
Model-1-IBP[b]	**72.65%**	**77.81%**	**89.02%**
(VGG-16/Model-1)-BP[a]	70.37%	**73.57%**	86.47%
(VGG-16/Model-1)-IBP[b]	71.22%	**74.41%**	**87.13%**
Tang et al. [Tang, 2013]	71.16%	–	–
Guo et al. [Guo u. a., 2016]	71.33%	–	–
Kim et al. [Kim u. a., 2016]	**73.73%**	–	–
Bishay et al. [Bishay u. a., 2019]	–	**73.1%**	–
Lian et al. [Lian u. a., 2020]	–	71.9 %	–
Acharya et al. [Acharya u. a., 2018a]	–	–	**87%**
S Li et al. [Li und Deng, 2018b]	–	–	74.2%
Z.Liu et al. [Liu u. a., 2017b]	–	–	73.19%

[a]BP: Bilinear Pooling.
[b]IBP: Improved Bilinear Pooling.

According to Table 2.2, the bilinear and improved bilinear CNN outperforms the state-of-the-art methods on the ExpW dataset. The best accuracy rate is 77.81% and has been reached using the improved bilinear pooling on top of Model-1. Bilinear pooling on top of Model-1 gives, for his turn, 76.59%. Moreover bilinear and improved bilinear on top of both VGG-16 and Model-1 gives respectively 73.57% and 74.41%. Whereas bilinear pooling and the improved version on top of VGG-16 give lower rates than state-of-the-art methods [Bishay u. a., 2019] (**73.1%**).

On RAF-DB dataset, the accuracy of our models is also superior to state of the art methods. The best accuracy rate is 89.02% and has been reached using the improved bilinear pooling on top of Model-1. Bilinear pooling on top of Model-1 gives, for his turn, 88.48%. Moreover improved bilinear on top of both VGG-16 and Model-1 gives 87.13%. Whereas bilinear pooling and the improved version on top of VGG-16, as well as bilinear pooling on top of both VGG-16 and Model-1 give lower rates than state-of-the-art methods [Acharya u. a., 2018a] (**87%**).

For FER2013, even thought using the bilinear and improved bilinear pooling improves considerably the models accuracy, the obtained results are still under the state of the art results. The best accuracy rate for this dataset, namely 72.65%, was reached using improved bilinear pooling on top of Model-1. Which 1% less than the state-of-the-art method [Kim u. a., 2016] (**73.73%**).

2.5 Conclusions and perspectives

This study proposes a FER method based on the improved bilinear CNN model. In this framework, various ways of normalization are used to improve the accuracy, including the matrix square root, element-wise square root and L2 normalization. To validate our method, we have used three large, well known, facial expression databases which are FER2013, RAF-DB and ExpW. In order to evaluate the improvement of our method, we have first implemented a CNN from scratch and fine-tuned pre-trained VGG-16 on our facial expressions datasets. After that we have implemented a bilinear model on top of the above models individually and on top of both of them. Finally, we repeated the same procedure with the improved bilinear model. The experiments show that this framework improves the overall accuracy for about 3%.

Bilinear models have been shown to achieve very good accuracy results on different visual recognition domains, like fine grained recognition, semantic segmentation and face recognition. Nevertheless, the dimensions of bilinear features are very high, usually on the order of hundreds of thousands to a few million. The reason why they are not practical for many visual recognition fields. Moreover, matrix square root function and bilinear pooling function are very memory and CPU consuming, which decrease the performance of the model. Therefore, many improvements have been applied to CNN, for instance compact bilinear pooling [Gao u. a., 2016], reaching the same discriminative power as the full bilinear representation but with a representations having only a few thousand dimensions. An other improvement is the kernel pooling for CNNs [Cui u. a., 2017] which is a general pooling framework that captures higher order interactions of features in the form of kernels.

Our Future work will focus on using more compact alternatives of the methods used in this work. Moreover, our perspective is to use multiple input data types (text, image and sound) in parallel, thus forming a multilinear FER model.

3 Learnable pooling weights for facial expression recognition

Contents

Pooling layers are spatial down-sampling layers used in CNNs to gradually downscale the feature map, increase the receptive field size and reduce the number of the parameters in the model. The use of pooling layers leads to less computing complexity and memory consumption reduction but also introduces invariance to certain filter distortions which may induce subtle detail loss. This behaviour is undesired for some fine-grained recognition tasks such as FER which highly relies on specific regional distortion detection. In this chapter, we introduce a more filter distortion aware pooling layer based on kernel functions. The proposed pooling reduces the feature map dimensions while keeping track of the majority of the information fed to the next layer instead of ignoring part of them. The experiments on RAF, FER2013 and ExpW databases demonstrate the benefits of such layer and show that our model achieves competitive results with respect to the state-of-the-art approaches.

3.1 Introduction

Pooling layers generally enhance the network performance and are mainly used for the gradual spatial down-sampling of the feature map. This will reduce the parameters number and thus reduces the memory consumption and computing complexity. In addition, pooling layers increase the receptive field size of the intermediate neurons allowing the latter to receive information from a larger area of the image. However, pooling layers introduce invariance to slight distortion which decreases the network performance on FER tasks as this distortion may cause the loss of some discriminative details [Gao u. a., 2019b].

In the literature, three conventional pooling methods have usually been employed with CNNs, namely: (1) max pooling, (2) average pooling and (3) strided convolution, having each their advantages and drawbacks. Max pooling, for instance, only keeps the largest input values assuming that the rest of values are not representative and do not bring relevant information. This assumption however is not always true, especially in the last layers of the network where even the small values represent a very relevant information. Therefore max pooling dramatically reduces the amount of useful information in the forward pass. Moreover, max pooling wrongly affects the learning of the network in the backward pass, since only one branch is activated in each input neighborhood. In an average pooling layer, all the inputs equally contribute to the output computation. This causes a constant and gradual attenuation of the contribution of individual neurons in the backward and forward passes [Saeedan u. a., 2018]. Strided convolution is simply a convolution layer with a stride bigger than one. This kind of layer is used in some very deep networks like ResNet [Cohen und Welling, 2016], whereas max and average pooling are used in mid-size networks like VGG [Simonyan und Zisserman, 2014] and GoogleNet [Szegedy u. a., 2015].

Although, these pooling methods are easy to compute from an input neighborhood, they omit important discriminant details which are crucial to many fine-grained classification problems. Particularly for FER, in which, we are more interested in detecting specific distortions of facial regions rather than simply identifying it in a given location (which is the case in a max-pooling operation [Acharya u. a., 2018b]). To handle this problem, we introduce in this chapter, a more filter-distortion aware pooling layer based on a kernel function which reduces the feature map dimensions while keeping track of the most discriminant information for FER instead of ignoring part of it. We show that the proposed layer improves the model performance in FER task, without many additional parameters.

Our contributions

We propose a FER method based on a CNN model to which we specifically designed a novel pooling layer that retains the down-sampling advantage of the ordinary pooling function and brings several new features. The proposed pooling layer has learnable weights which generalize standard pooling functions (i.e. max and average pooling). In other words, it can learn a suitable pooling from a continuum of methods that ranges from average to max (or extremum) pooling. It additionally encodes patch-wise non-linearity which in turn improves the discrimination power of the full network. The novel pooling is completely differentiable and can be used at any level of the network, allowing an end-to-end learning.

The remainder of this chapter is organized as follow: Section 3.2 reviews similar works that have been proposed for pooling based networks. Section 3.3 introduces the proposed pooling layer for FER. Section 3.4 presents the different conducted experiments and their related results. Section 3.5 concludes the chapter.

3.2 Pooling method: an overview

All presented CNN based methods [Li und Deng, 2018a] use one or many conventional pooling layers (max/average pooling, or strided convolution). As stated in Section 3.1 these pooling layers introduce invariance to slight distortion which may decrease the network performance on fine-grained classification tasks. Several methods have been proposed to overcome this limitation and thereby improves the performance of CNNs. As discussed in chapter 2, several pooling techniques were proposed in the literature. However, all these methods are always plugged at the end of the network, right between the convolution layers and the fully connected layers. They act as a basis expansion layers, increasing thereby the discrimination power of the fully connected layers. This discrimination power is back-propagated through the convolution layers allowing the network to learn in an end-to-end fashion. These methods have attracted increasing attentions, achieving better performance than classical, first-order networks in a variety of computer vision tasks. Even-thought these methods increase the CNN performance, they are enable to learn by themselves and rely entirely on CNN architecture. Furthermore, how to effectively introduce higher-order representation in earlier layers for improving non-linear capability of CNNs is still an open problem. In the following we describe some of the proposed pooling methods used in CNNs.

1. **Mixed Pooling**

   Max pooling extracts only the maximum activation whereas average pooling down-weighs the activation by combining the non-maximal activations. To overcome this problem, Yu et al. [Yu u. a., 2014a] proposed a hybrid approach by combining the average pooling and max pooling. This approach is highly inspired by dropout [Hinton u. a., 2012] and Drop connect [Wan u. a., 2013]. Mixed pooling can be represented as:

$$S_j = \lambda max_{i \in R_j} a_i + (1-\lambda)\frac{1}{|R|}\sum_{i \in R_j} a_i \tag{3.1}$$

where λ decides the choice of either using max pooling or average pooling. The value of λ is selected randomly either 0 or 1. When $\lambda = 0$ it behaves like average pooling, and when $\lambda = 1$ it works like max pooling. The value of $\lambda$is recorded for forward-propagation order and it is used during the backpropagation process. Yu et al. showed its superiority over max and average pooling by performing image classification on three different datasets.

2. **L_p poling**

   Sermanet et al. [Sermanet u. a., 2012] proposed the concept of L_p pooling and claimed that its generalization ability is better than max pooling. In this pooling, a weighted average of inputs is taken in pooling region. It is represented as:

$$S_j = \left(\frac{1}{|R|} \sum_{i \in R_j} a_i^p \right)^{\frac{1}{p}} \tag{3.2}$$

   where S_j represents the output of the pooling operator at location j, a_j is the feature value at location i within the pooling region R_j. The value of p varies between 1 and ∞. When $p = 1$, L_p operator behaves as average pooling and at $p = \infty$ it leads to max-pooling. For L_p pooling, $p > 1$ is examined as a trade-off between average and max pooling.

3. **Stochastic Pooling**

   Inspired by the dropout [Hinton u. a., 2012], Zeiler and Fergus [Zeiler und Fergus, 2013] proposed the idea of stochastic pooling. In max pooling, the maximum activation is selected from each pooling region. Whereas the areas of high activation are down-weighted by areas of low-activation in average pooling, because all elements in the pooling region are examined, and their average is taken. It is a major problem with average pooling. The issues of max and average pooling are addressed using stochastic pooling. Stochastic pooling applies multinomial distribution to pick the value randomly. It includes the non-maximal activations of the feature map. In stochastic pooling, first, the probabilities P_i is computed for each region j by normalizing the activations within the regions, as given in:

$$P_i = \frac{a_i}{\sum_{k \in R_j} a_k} \tag{3.3}$$

   These probabilities create a multinomial distribution that is used to select location l and corresponding pooled activation a_i based on p. Multinomial distribution selects a location l within the region. In simple words, the activations are selected based on the probabilities calculated by multinomial distribution. In this, all activations get the chances according to their probability proportionate. Stochastic pooling prohibits overfitting because of the stochastic component. Some advantages of max-pooling are also available in the stochastic pooling, and it also utilizes non-maximal activations. It is to be noted that stochastic pooling represents the multinomial distribution of activations within the region; hence the selected element may or may not be the largest element. It gives high chances to stronger activations and suppresses the weaker activations.

4. **Spatial Pyramid Pooling**

   Among the new methods used for the pooling layer, is the spatial pyramid pooling. Spatial pyramid pooling [Grauman und Darrell, 2005, Lazebnik u. a., 2006] (popularly known as spatial pyramid matching or SPM [Lazebnik u. a., 2006]), as an extension of the Bag-of-Words (BoW) model [Sivic und Zisserman, 2003], is one of the most successful methods in computer vision. It partitions the image into divisions from finer to coarser levels and aggregates local features in them. In [He u. a., 2015b], He et.al introduced a spatial pyramid pooling (SPP) [Grauman und Darrell, 2005, Lazebnik u. a., 2006] layer to remove the fixed-size constraint of the network. Specifically, they added an SPP layer on top of the last convolutional layer. The SPP layer pools the features and generates fixed-length outputs, which are then fed into the fully-connected layers. In other words, Huang et.al in [Huang u. a., 2020] performed some information aggregation at a deeper stage of the network hierarchy, between convolutional layers and fully-connected layers, to avoid the need for cropping or warping at the beginning and build the YOLO detection method.

5. **Region of Interest Pooling**

   The Region of Interest (RoI) Pooling layer is an important component of convolutional neural networks which is mostly used for object detection [Girshick, 2015] and segmentation [Liu u. a., 2017a]. The ROI pooling layer worked by shifting the processing specific to individual bounding-boxes later in the network architecture. An input image is processed through the deep network and intermediate CNN feature maps (with reduced spatial dimensions compared to the input image) are obtained. The ROI pooling layer takes the input feature map of the complete image and the coordinates of each ROI as its input. The ROI co-ordinates can be used to roughly locate the features corresponding to a specific object. However, the features thus obtained have different spatial sizes because each ROI can be of a different dimension.

   Since CNN layers can only operate on fixed dimensional inputs, an ROI pooling layer converts these variable sized feature maps (corresponding to different object proposals) to a fixed-sized output feature map for each object proposal. The fixed-size output dimensions are a hyper-parameter which is fixed during the training process. Specifically, this same-sized output is achieved by dividing each ROI into a set of cells with equal dimensions. The number of these cells is the same as the required output dimensions. Afterward, the maximum value in each cell is calculated (max-pooling) and it is assigned to the corresponding output feature map location.

Using a single set of input feature maps to generate a feature representation for each region proposal, the ROI pooling layer greatly improves the efficiency of a deep network.

6. **Universal pooling**

   Hyuan et al. [Hyun u. a., 2019] proposed a new pooling method called universal pooling. The method intends to generate pooling function which better fits any problem given a dataset. Universal pooling has been inspired by attention methods and can be considered as a channel-wise form of local spatial attention. The strength of these methods relies on the fact that they capture additional discriminant information compared to conventional pooling techniques. This makes them more suitable for fine-grained classification problem.

In this chapter, we build upon these works and introduce a novel pooling layer that not only uses all input information but also extracts linear and non-linear relations between features. To do so, we leverage kernel functions which allow to generalise linear pooling while capturing higher order information.

3.3 Proposed method

We propose an end-to-end model to perform the FER task. Our network architecture is simple. It is designed to capture discriminant facial features through successive layers of multiple non-linear transformations and representations. It follows standard CNN as it alternates convolutional and pooling layers and ends with a fully connected softmax activation layer (see Figure 3.1). The convolutional layers learn several filter weights which are convolved with the input facial image and produce a set of feature maps. The filter weights are learned such that the final classification score is high (categorical cross entropy loss is employed, see Section 3.3.2).

The novelty in this work is a specific pooling layers which are more sensitive to subtle details in feature maps than standard pooling techniques (i.e. max-pooling, average-pooling, etc). This is ensured by adding learnable weights to the pooling layers, similarly as in convolutional layers, while reducing the feature map size. By doing so, the standard pooling techniques can be seen as a particular case of our new pooling with fixed (non-learnable) weights. In Section 3.3.1, we present our proposed pooling layer used for boosting the FER task.

3.3.1 Learnable pooling

The proposed learnable pooling layer is similar to an ordinary pooling one in the way that it applies a pooling function on a specific location and a specific stride. The difference from previous poolings corresponds to the capability to dynamically extract more relevant features from the input map. This is particularly performed by learning different pooling weights for each feature map. These weights are learned in a similar fashion as convolutional weights but with a single depth output (see Figure 3.1).

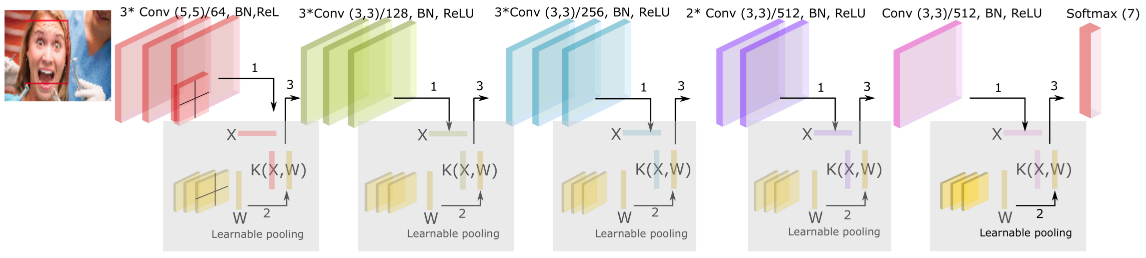


Figure 3.1 – *Our proposed network architecture for FER task. The CNN alternates convolutional layers and specifically designed layers. It ends by a fully softmax activation layer. Each convolutional layer is followed by batch normalization and rectified linear unit activation.*

Finally, a combination of the original feature map and the resulting weights is computed using a specific function. This function is carefully chosen to capture linear and non linear relations between both the weights and the original feature map. The output of our pooling layer is a new feature map with reduced high and width.

Formally and as shown in Figure 3.2, we consider a flattened feature map vector $x = \{x_1, x_2, \dots, x_d\}$ and a vector of pooling weights $w = \{w_1, w_2, \dots, w_d\}$. In Figure 3.2, d is equal to 4 which corresponds to a 2×2 pixels. The vector w can be seen as a second feature map which is dynamically learned. In order to capture linear and non-linear relations between x and w, we employ a Symmetric Positive Definite (SPD) function $\mathcal{K} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. The choice of the SPD function is motivated by the fact that standard pooling (e.g. average pooling) can be considered as a linear combination of fixed filter weights and the feature map values in a particular location. For instance given the feature map vector x and a set of non-learnable weights $w = \{1/d, 1/d, \dots, 1/d\}$, the average pooling can be computed as: $\sum_{i=1}^{d} w_i * x_i = x^T w$ which is the dot product of both vectors x and w, and corresponds to the inner product in $\mathbb{R}^d$. By employing a SPD function, we emulate an inner product in a higher dimensional space after a non linear mapping of both x and w vectors. Thus, the pooling operation turns out to be

$$\langle \varphi(x), \varphi(w) \rangle \approx \mathcal{K}(x, w). \tag{3.4}$$

In this work, we employed three different functions defined on the feature map space which has an Euclidean structure $\mathbb{R}^d$.

- Linear kernel:

$$\mathcal{K}(x, w) = x^T w, \quad x, w \in \mathbb{R}^d. \tag{3.5}$$

- Polynomial kernel:

$$\mathcal{K}(x, w) = (x^T w + r)^n, \quad x, w \in \mathbb{R}^d, r \geq 0. \tag{3.6}$$

- Gaussian kernel (RBF Kernel):

$$\mathcal{K}(x, w) = e^{-\frac{\|x-w\|^2}{2\sigma^2}}, \quad x, w \in \mathbb{R}^d, \sigma > 0. \tag{3.7}$$

The linear kernel (Equation 3.5) looks at the similarity between the feature map vector x and the filter weight vector w. Starting form $n > 1$ in Equation 3.6, the polynomial kernel encodes not only the linear relation between both x and w vectors, but also non-linear relations between them. Thanks to the exponential term in Equation 3.7, the Gaussian kernel expands the pooling non-linearity to the infinity. This expansion can also be reached by other functions, such as the Laplacian kernel defined by $\mathcal{K}(x, w) = e^{-\alpha\|x-w\|}$, or the Abel kernel defined as $\mathcal{K}(x, w) = e^{-\alpha|x-w|}$, where, $\alpha > 0$ in both kernels.

The proposed learnable pooling preserves the main purpose of a standard pooling layer which corresponds to the down-sampling of the input feature map. But it not only summarizes the presence of specific features in patches, it also captures the non-linear relations between these features.

3.3.2 Learning

Our network ends with a fully connected layer to make sure that all activations in the previous layer are connected to the last layer and to allow the pooled 2D feature maps to be converted into a vector of probabilities for FER. In this work, we chose to use the traditional softmax layer, with a cross entropy loss, to simply force features of different expressions to remain apart. Many authors have proposed advanced losses for FER such as the center loss [Wen u. a., 2016], the island loss [Cai u. a., 2018], and the locality-preserving loss [Fabian Benitez-Quiroz u. a., 2016]. However here we opt for a simple softmax layer and a cross entropy loss to demonstrate the efficiency of the proposed learnable pooling layer.

Given a facial image I with a label vector y of y_i elements ($y_i = 1$ if I belongs to C_i otherwise $y_i = 0$ where C_i indicates the ground truth expression of the face in I), the objective of our learning problem is to minimize the cross entropy loss over the set of C classes :

$$CE = \sum_{i}^{C} y_i \log(f(I)_i), \tag{3.8}$$

where $f(I)_i$ stands for the softmax activation of the $i - th$ class.

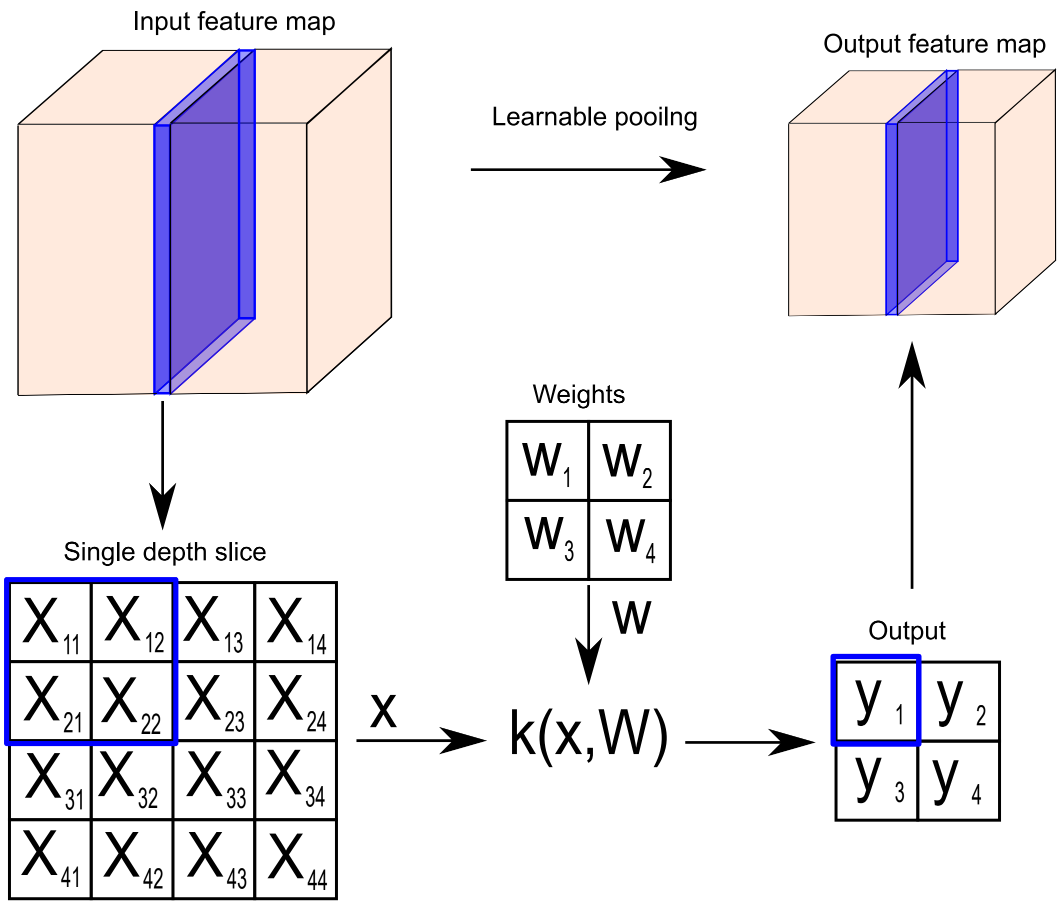

Input feature map
Learnable pooilng
Output feature map
Weights
W_1 W_2 W_3 W_4
W
Single depth slice
X_{11} X_{12} X_{13} X_{14}
X_{21} X_{22} X_{23} X_{24}
X_{31} X_{32} X_{33} X_{34}
X_{41} X_{42} X_{43} X_{44}
x
k(x,W)
Output
y_1 y_2 y_3 y_4

Figure 3.2 – *The processing of the learnable weights pooling layer is similar to usual pooling layer in the manner that it down-scales the spatial dimensions of the input. Learnable weights pooling rely on learnable weights to encode important relations between features through kernel function.*

The learnable pooling weights are initialized using *He* normal function. It draws samples from a truncated normal distribution centered on 0 (zero) with a standard deviation given by:

$$stddev = \sqrt{\frac{2}{N}} \tag{3.9}$$

Where N is the number of input units in the weight tensor.

3.4 Experiments

In order to evaluate our method, several experiments have been conducted on three well-known datasets, namely the RAF-DB [Li u. a., 2017], ExpW [Zhang u. a., 2018b] and FER2013 [Goodfellow u. a., 2013]. The only preprocessing which we have employed on all experiments is cropping the face region and resizing the resulting images to 100×100 pixels.

3.4.1 Training process

In order to demonstrate the efficiency of the proposed learnable pooling, we build a simple CNN from scratch, in addition of using two pre-trained models, namely: ResNet-50 and VGG-16. We have used Adam optimiser with a learning rate varying from 0.001 to 5e-5. This learning rate is decreased by a factor of 0.63 if the validation accuracy does not increase over ten epochs. To avoid over-fitting we have also augmented the data using a range degree for random rotations of 20, a shear intensity of 0.2, a range for random zoom of 0.2 and randomly flip inputs horizontally.

As shown in Figure 3.1 our model architecture is quite simple and can effectively run on cost-effective GPUs. It is composed of five convolutional blocks. Each block consists of a convolution, batch normalization and rectified linear unit activation layers. The use of batch normalization [Zou u. a., 2019] before the activation brings more stability to parameter initialization and achieves higher learning rate. Each of the five convolutional blocks is followed by a pooling layer. In the following we refer to this network architecture as (Model-1).

3.4.2 Ablation Study

This section explores the impact of the use of the proposed learnable pooling layer on the overall accuracy of Model-1, VGG-16 and ResNet-50. We evaluated the performance of these network architectures with different pooling techniques. First, we evaluate these three models using the standard pooling techniques, namely: max pooling, average pooling and strided convolution. After that, we replaced these poolings by our learnable pooling layers. We studied the behaviour of four different kernel functions, namely; (1) the linear kernel, (2) the second-order polynomial kernel, (3) the third-order polynomial kernel and (4) the Gaussian RBF kernel. We also evaluated the performance of these methods with different window sizes. The experiments show that the increase of the pooling window affect negatively the performance of the model. Yet our pooling method still performs better than the usual pooling methods. As consequence, the smallest pooling window (2×2) is the best suited for better results.

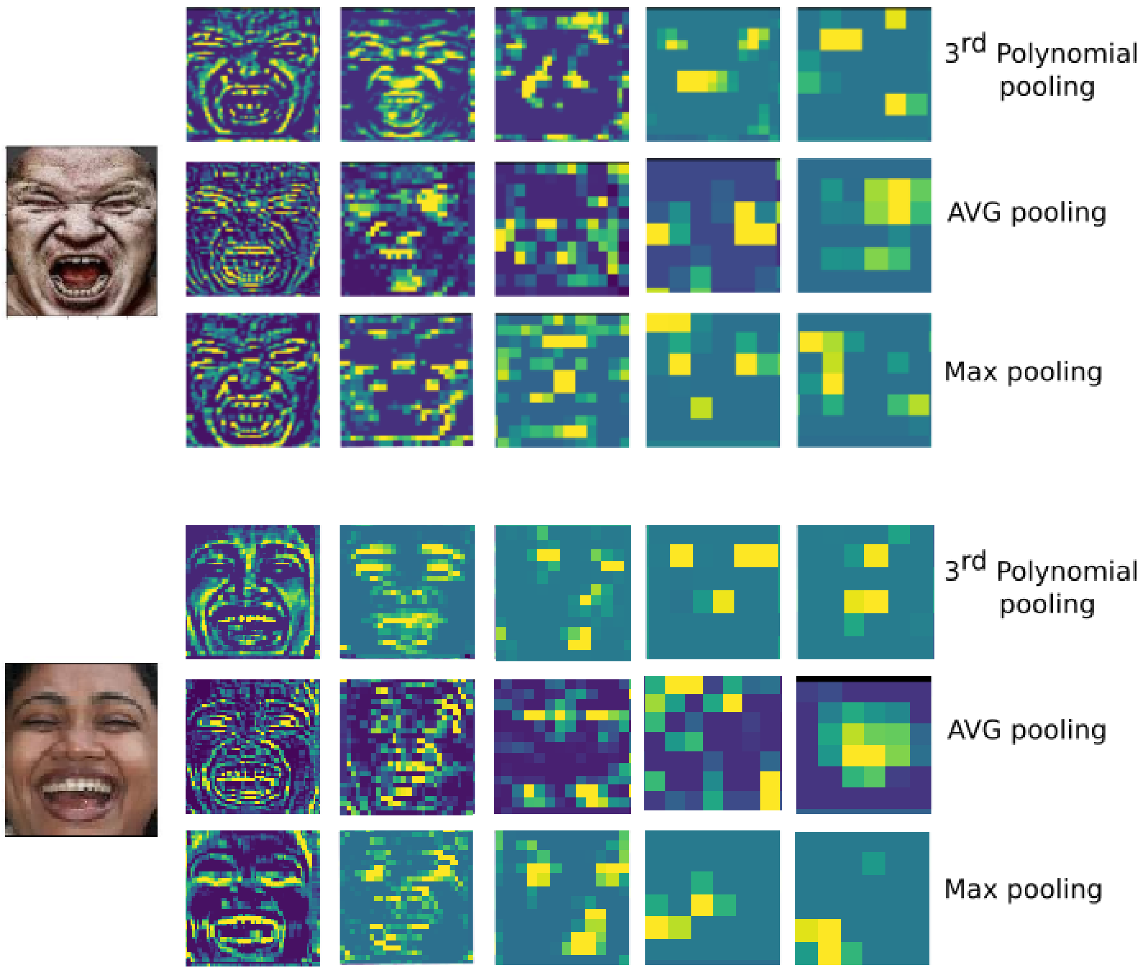
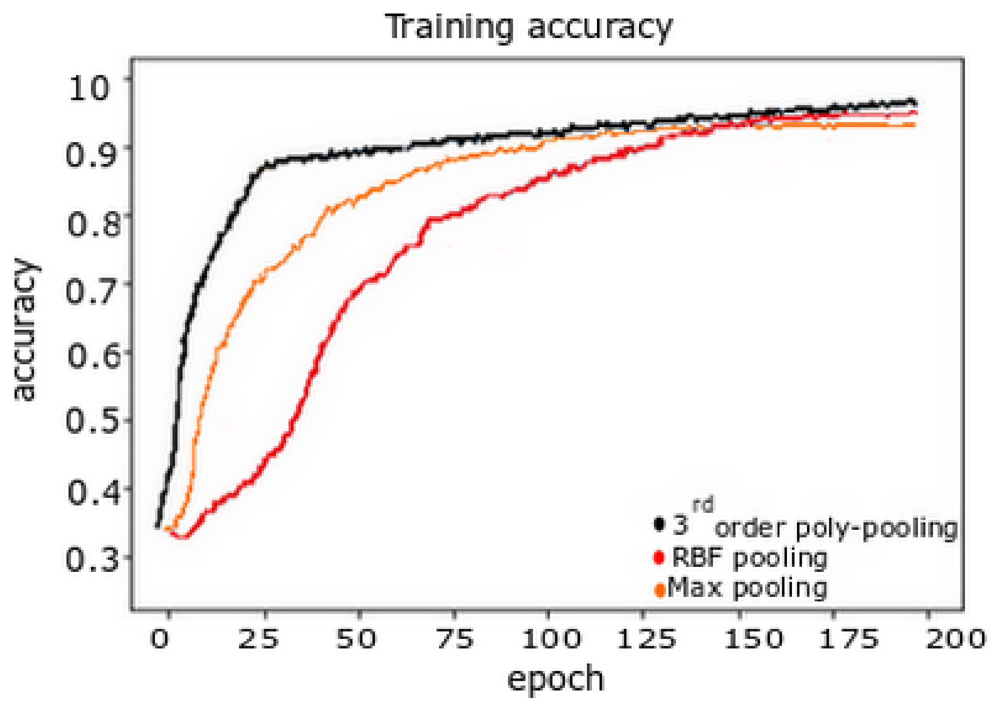


Figure 3.3 – *Visualisation of the outputs from the pooling layers. These visualisations are generated from two facial expressions (the face in the left, and the face in the middle). Given an input image, we show the feature maps after each of the five pooling layers used in our CNN. The first row shows the feature maps after third polynomial kernel based pooling. The second and the third rows present feature maps after the standard average and max pooling respectively.*



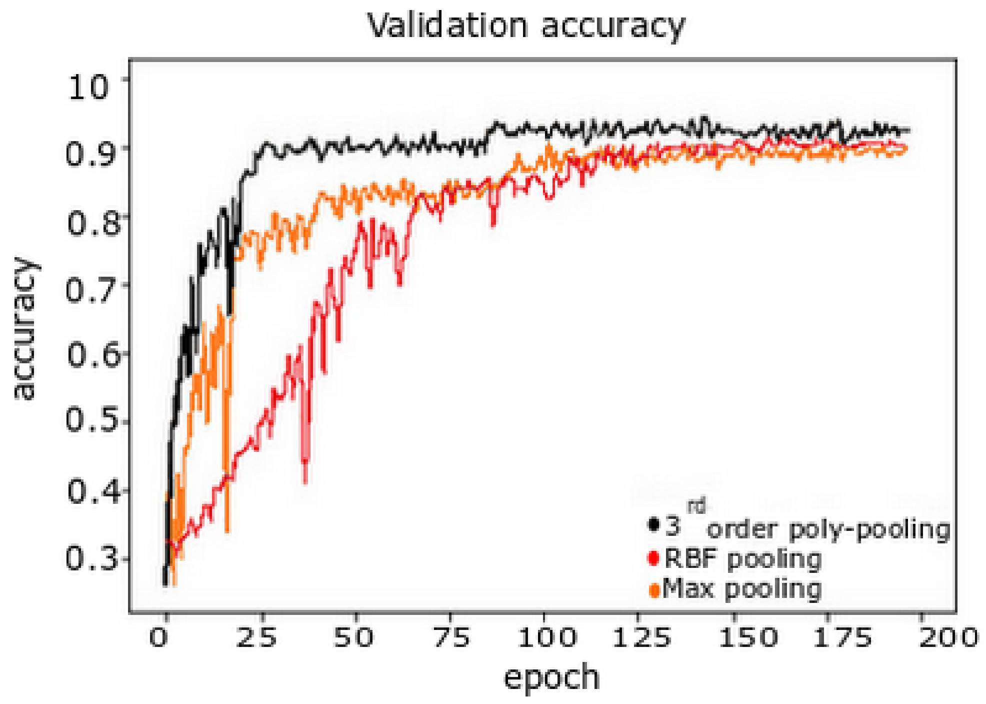


Figure 3.4 – *Visualizations of accuracy versus epoch plots. This figure reports the impact of the learnable pooling on the convergence speed of the used CNN. A comparison between the performance of the CNN using max pooling, the third order polynomial, and the RBF pooling methods on the RAF-DB dataset.*

Table 3.1 – *Accuracy rate of our proposed approach for different pooling strategies. In this table, Model-1 architecture is used with the indicated pooling method*

Pooling	ExpW	RAF-DB	FER2013
Max	75.91%	87.05%	70.49%
AVG	75.74%	86.89%	70.13%
Strided Conv	73.81%	85.23%	68.74%
Linear kernel	76.28%	90.81%	70.69%
2^{nd}-order Poly	76.64%	92.87%	70.88%
3^{rd}-order Poly	76.81%	93.21%	71.35%
Gaussian RBF	76.42%	92.74%	70.74%

The experiments are conducted with the same training parameters as described above. Tables 3.1, 3.2 and 3.3 present the results of Model-1, ResNet-50 and VGG-16 using standard pooling methods with comparison to the proposed learnable pooling. From table 3.1, one can notice that considering Model-1 architecture, the use of max or average pooling gives approximately the same results with a slight improvement when max pooling is employed. On the other hand, strided convolution reaches lower accuracy rates than both max and Average pooling. In the case of max pooling, Model-1 attains 75.91%, 87.05% and 70.49% of accuracy rate on respectively ExpW, RAF-DB and FER2013 datasets. When using average pooling layers instead of max pooling, Model-1 reaches an accuracy rate of 75.74%, 86.89%, and 70.13% for respectively ExpW, RAF-DB and FER2013 datasets. Whereas, using strided convolution, model-1 reaches an accuracy rate of 73.81%, 85.23% and 68.74% respectively on ExpW, RAF-DB and FER2013 datasets. However in contrast to these three cases, the use of learnable pooling layers in Model-1 considerably increases their accuracy. As reported in Table 3.1, the accuracy of Model-1 in which we use a learnable pooling with a linear kernel increases the accuracy up to 0.4% for ExpW, 3.75% for RAF-DB and 0.2% for FER2013.

Following the same principle, we further studies the impact of the usage of three additional kernels; (1) the second-order polynomial kernel with $r = 1$, (2) the third-order polynomial kernel with $r = 1$ and (3) the Gaussian kernel with $\gamma = 0.9$. Although the computational complexity of these kernels is higher compared to the max and the average pooling, they strongly improve the model accuracy when used. As shown in Table 3.1, 3.2 and 3.3, the same model architecture but using the third-order polynomial kernel outperforms the other methods. The use of this kernel improves the accuracy rate close to 71.35% for Model-1 on FER2013, 76.81% on ExpW, and enhances the accuracy rate up to 93.21% on RAF-DB. Less efficient than the third-order polynomial kernel but also computationally expensive is the Gaussian kernel. It increases the accuracy, for Model-1, up to 76.42% for ExpW and 70.74% FER2013 comparing to max and average poolings and 92.74% for RAF-DB. Moreover, the Gaussian kernel takes a considerable time to converge. Finally, according to Table 3.1, the second-order polynomial kernel is less efficient than the third-order one but remains better than the Gaussian kernel. It merely outperforms the linear kernel with slightly higher complexity. However, it is the fastest kernel to converge compared to the third-order polynomial and the Gaussian kernels.

Table 3.2 – *Accuracy rate of our proposed approach for different pooling strategies. In this table, ResNet-50 architecture is used with the indicated pooling method*

Pooling	ExpW	RAF-DB	FER2013
Max	72.32%	85.48%	68.12%
AVG	73.18%	84.74%	68.09%
Strided Conv	73.95%	85.11%	68.18%
Linear kernel	74.35%	87.72%	68.42%
2^{nd}-order Poly	74.41%	88.21%	68.94%
3^{rd}-order Poly	75.13%	89.49%	69.68%
Gaussian RBF	74.19%	88.16%	68.53%

Similarly to Model-1, ResNet-50 with max and average pooling gives approximately the same results, as shown in table 3.2. In the case of max pooling, ResNet-50 attains 72.32%, 85.48% and 68.12% of accuracy rate on respectively ExpW, RAF-DB and FER2013. Whereas, when using average pooling ResNet-50 reaches 73.18%, 84.74% and 68.09% of accuracy rate on respectively ExpW, RAF-DB and FER2013. On the other hand, ResNet-50 with strided convolution reaches better results than max and average pooling. Strided convolution is the built-in pooling method for pre-trained ResNet-50 model. It gives 73.95% for ExpW, 85.11% for RAF-DB and 68.18% for FER2013. As for Model-1 the linear kernel on ResNet-50 performs at least as good as the other pooling methods. It reached an accuracy rate of 74.35% on ExpW, 87.72% on RAF-DB and 68.42% on FER2013. The use of higher order kernels also increases the accuracy rate of ResNet-50. Second order polynomial kernel nearly outperforms the linear kernel. It attains 74.41%, 88.21% and 68.94% on ExpW, RAF-DB and FER2013 respectively. Third order polynomial kernel remains the most efficient kernel for our learnable pooling method. It reached with ResNet-50 an accuracy rate of 75.13% on ExpW, 89.49% on RAF-DB and 69.68% FER2013. Finally, Gaussian RBF kernel remains the less efficient non-linear kernel with an accuracy rate of 74.19%, 88.16% and 68.53% on respectively ExpW, RAF-DB and FER2013.

Table 3.3 – *Accuracy rate of our proposed approach for different pooling strategies. In this table, VGG-16 architecture is used with the indicated pooling method*

Pooling	ExpW	RAF-DB	FER2013
Max	67.61%	85.23%	65.23%
AVG	67.42%	84.92%	65.11%
Strided Conv	66.87%	84.51%	64.74%
Linear kernel	68.17%	85.86%	65.67%
2^{nd}-order Poly	68.56%	86.31%	66.19%
3^{rd}-order Poly	70.24%	87.04%	67.13%
Gaussian RBF	68.43%	86.16%	66.04%

Finally, VGG-16 with standard pooling methods gives approximately the same results, similarly to Model-1 and ResNet-50. For instance, max pooling reached 67.61% on ExpW, 85.23% on RAF-DB and 65.23% on FER2013. Average pooling, on the other hand, attained 67.42%, 84.92% and 65.11% on ExpW, RAF-DB and FER2013 respectively. In the case of strided convolution, the accuracy rates reached were 66.87%, 84.51% and 64.74% for ExpW, RAF-DB and FER2013 respectively. Again, linear kernel is at least as efficient as the standard pooling methods. It reached 68.17%, 85.86% and 65.67% of accuracy on ExpW, RAF-DB and FER2013 respectively. For the non-linear kernels, third order polynomial kernel remains the most efficient kernel with 70.24% on ExpW, 87.04% on RAF-DB and 67.13% on FER2013. Less efficient but faster is the second order polynomial kernel with an accuracy rate of 68.56%, 86.31% and 66.19% on ExpW, RAF-DB and FER2013 respectively. Finally, Gaussian RBF kernel is the less accurate and slowest non-linear kernel with 68.43%, 86.16% and 66.04% on ExpW, RAF-DB and FER2013 respectively.

As demonstrated above, using learnable pooling layers with linear kernels performs at least as good as the use of layers with max pooling, average pooling or strided convolution. This behaviour can be explained by the fact that the linear kernel can automatically learn the suitable pooling method from a continuum of methods which include strided convolution, average and max pooling as particular cases. Although our proposed pooling layer increases the number of learnable parameters, compared to the max and the average pooling, the use of the simple linear kernel gives more flexibility to the pooling since it acts as a decision maker of which weights to fix or use for each particular filter. Eventhough, using learnable pooling with non-linear kernels increases the complexity, it allows the model to reach better results. Second order polynomial kernel outperforms linear kernel with slightly higher complexity. Third order polynomial kernel add more complexity but reaches the highest results. The less efficient and also computationally expensive is the Gaussian RBF kernel.

To further compare the proposed pooling technique with standard ones, we display in Figure 3.3 the output image after each pooling layer. We compared max and average pooling with the third order polynomial pooling. As depicted in Figure 3.3, our pooling method is able to capture more details than the standard pooling techniques. The visualizations show that the third order pooling captures relevant features which likely correspond to the expression action units e.g. the outlines of the mouth and the eyes. Moreover, the third order pooling outperforms the other techniques in discarding non-informative regions. One can clearly notice particularly in the two last layers, even when the results are abstract and difficult to interpret, that the learnable pooling keep activation of well localized features (nose, mouth and eyes). On the contrary, one can also notice some common activated regions particularly in the earlier layers, this can be explained by the fact that our pooling encompasses standard poolings thanks to the linear term in the polynomial kernel.

In Figure 3.4, we report the training accuracy and validation accuracy versus epoch plots of our CNN when using the max pooling, the RBF, and the third order kernel pooling layers. These plots give an idea about the influence of the different pooling techniques on the convergence rate of the network. The sub-Figures demonstrate that the third order based pooling has an important impact on the convergence speed of the network compared to the max and the RBF based poolings. Although the computational complexity of the third order kernel is higher than the max pooling, it is still able to converge to a higher validation accuracy in less epochs. Yet, the prediction speed increases accordingly with the kernel complexity. On an average machine, with 6 Gb GPU, The procesing of an image of size 100×100 pixels takes 14 milliseconds with Max pooling, 14.7 with linear kernel, 16 milliseconds with third degree polynomial kernel and 18 milliseconds with Gaussian RBF kernel.

Note that in the literature, few researchers claim that the standard max pooling performs a noise removal arguing that it gets rid of noisy features and also brings denoising along with dimensionality reduction [Gao u. a., 2019b]. On the contrary, the average pooling only carries out dimensionality reduction. Thus, the max pooling is generally considered to achieve better performance than average pooling. However, by using our proposed pooling, we demonstrate that the majority of the features in the input feature map are relevant. By leveraging kernel functions, non-linear relation between features in a given patch are captured and this allows to performs better than standard max and average pooling. These results demonstrate that non-linear relations between features in the feature map produced after a convolutional layer are beneficial for the overall accuracy of the FER problem. The use of the third order pooling kernel allows to achieve the best performance compared to standard pooling techniques as well as the different studied kernels.

Table 3.4 – *Accuracy rate of our proposed approach and state of the art approach*

Methods	ExpW	RAF-DB	FER2013
Linear kernel	76.28 %	90.81%	70.69%
2^{nd}-order Poly	76.64%	92.87 %	70.88 %
3^{rd}-order Poly	**76.81%**	**93.21 %**	**71.35%**
Gaussian RBF	76.42 %	92.74 %	70.74%
LIP [Gao u. a., 2019b]	76.52%	87.63%	70.79%
Tang et al. [Tang, 2013]	–	–	71.16 %
Guo et al. [Guo u. a., 2016]	–	–	71.33 %
Kim et al. [Kim u. a., 2016]	–	–	**73.73 %**
Bishay et al. [Bishay u. a., 2019]	73.1 %	–	–
Lian et al. [Lian u. a., 2020]	71.9 %	–	–
Acharya et al. [Acharya u. a., 2018a]	–	87%	–
Kuo et al. [Kuo u. a., 2018]	–	65.52%	–
Deng et al. [Deng u. a., 2015]	–	68.2%	–
S Li et al. [Li und Deng, 2018b]	–	74.2%	–
Z.Liu et al. [Liu u. a., 2017b]	–	73.19%	–

3.4.3 Comparison with the State-of-the-Art

In this section, we compare the performance of Model-1 which uses the proposed learnable pooling with respect to several state-of-the-art methods. Moreover, we have also used state-of-the-art pooling method LIP [Gao u. a., 2019b] with Model-1 for further comparison. The obtained results are reported in Table 3.4. According to Table 3.4, our proposed model outperforms the state-of-the-art methods on the ExpW dataset. The best accuracy rate is 76.81% and has been reached using the third order polynomial kernel. The second order polynomial kernel gives 76.64% while the linear kernel achieves 76.28% as accuracy rate. Finally, using the Gaussian kernel our model achieves 76.42% of accuracy.

On RAF-DB dataset, the accuracy of our model is also superior to state-of-the-art methods. Using the third order polynomial pooling, our model accuracy exceeds the other methods by more than 1% and it obtains 93.21%. One can also notice that all kernels outperform state-of-the-art methods. Using a linear kernel, our model achieves 90.81% of accuracy while 92.87% is reached using the second order polynomial pooling. The use of the Gaussian kernel allows to reach 92.74%, whereas the best state-of-the-art method only reports 87% of accuracy [Acharya u. a., 2018a] and 87.63% with state-of-the-art LIP pooling method [Gao u. a., 2019b]. Similarly, with the ExpW dataset and using the proposed method, we outperform state-of-the-art methods with all kernels. We obtained a 76.81% of accuracy using the third order polynomial pooling which exceeds the other methods by more than 3%. Finally, even though we did not outperform state-of-the-art methods on FER2013, we confirmed the superiority of our method compared to standard pooling methods. We reached an accuracy rate of 71.35% with the third order polynomial pooling which is 2% less than state-of-the-art method [Kim u. a., 2016]. LIP pooling method slightly outperforms linear kernel on the ExpW and the FRE2013 datasets. It is however, outperformed by the third order polynomial pooling on the three datasets.

3.4.4 Cross-dataset evaluation

We have also evaluated the generalizability of our network on data from different distributions. We conducted an experiment on a cross-dataset. We compared the performance of our network using the proposed learnable pooling weights with the same network using standard pooling layers. The considered intra-dataset protocol is a training over the whole ExpW dataset and a testing on RAF-DB dataset. The obtained results also confirm the efficiency of the proposed pooling layer in the FER task. Our method using a linear kernel gives 80.27% as accuracy rate. The use of the second-order polynomial kernel allows to achieve 80.56%. The third-order polynomial kernel and the Gaussian RBF kernel give respectively 81.43% and 81.03%. On the contrary, the use of max-pooling layers instead of our learnable weights only allows to reach 80.12%.

3.5 Conclusion

In this chapter, we proposed a FER method based on a CNN model to which we specifically designed a novel pooling layer which retains the down-sampling advantage of an ordinary pooling function and brings several new features. The proposed pooling layer, which has learnable weights, generalizes standard pooling functions and, additionally encodes non-linear relation between features. It is differentiable and can be plugged at any level of the network, allowing, in turns, an end-to-end learning. The experiments on ExpW, RAF-DB and FER2013 datasets demonstrate the efficiency of the proposed pooling method compared to standard pooling. The experiments also showed that the proposed FER method outperforms state-of-the-art methods. The performance of our model is essentially due to its capability of capturing high order information that are crucial for fine-grained classification tasks such as the FER.

4 Kernelized dense layers for facial expression recognition

Contents

Fully connected layer is an essential component of Convolutional Neural Networks (CNNs), which demonstrates its efficiency in computer vision tasks. The CNN process usually starts with convolution and pooling layers that first break down the input images into features, and then analyze them independently. The result of this process feeds into a fully connected neural network structure which drives the final classification decision. In this chapter, we propose a Kernelized Dense Layer (KDL) which captures higher order feature interactions instead of conventional linear relations. We apply this method to Facial Expression Recognition (FER) and evaluate its performance on RAF, FER2013 and ExpW datasets. The experimental results demonstrate the benefits of such layer and show that our model achieves competitive results with respect to the state-of-the-art approaches.

4.1 Introduction

Deep neural network architecture was found to be inefficient for computer vision tasks, since images represent a large input for a neural network (they can have hundreds or thousands of pixels and up to 3 color channels) with a huge number of connections and network parameters. CNNs leverage the fact that an image is composed of smaller details, or features, and creates a mechanism for analyzing each feature in isolation, which informs a decision about the image as a whole. As part of the convolutional network, FC layer uses the output from the the convolution/pooling process and learns a classification decision. It is an essential component of CNNs, which demonstrates its utility in several computer vision tasks. The input values flow into the first FC layer and they are multiplied by weights. The latter usually go through an activation function (typically ReLu), just like in a classic artificial neural network. They then pass forward to the output layer, in which every neuron represents a classification label. The FC part of the CNN goes through its own back-propagation process to determine the most accurate weights where each neuron receives weights that prioritize the most appropriate label. In fully connected layers, the neuron applies a linear transformation to the input vector through a weights matrix. It applies a dot product between a vector of weights $W = \{w_1, w_2, \ldots, w_n\}$ and an input vector $x = \{x_1, x_2, \ldots, x_n\}$, add a bias vector ($b \geq 0$) and eventually applies an activation function f. Let $Y_i \in \mathbb{R}$ be the $i^t h$ output from the fully connected layer for j input values. Then $Y_i \in \mathbb{R}$ is computed as follows:

$$Y_i = f\left(\sum_j X_j W_{i,j} + B_i\right) \tag{4.1}$$

One way of thinking about fully connected layers is that each fully connected layer effects a transformation of the feature space in which the problem resides. The ability to perform problem-specific transformations can be immensely powerful [Bosagh und Ramsundar, 2018]. Standard transformation techniques couldn't solve problems of image or speech analysis, while deep networks are capable of solving these problems with relative ease due to the inherent flexibility of the learned representations.

In a typical deep neural network, the FC layers comprise most of the parameters of the network. AlexNet has 60 million parameters, out of which 58 million parameters correspond to the FC layers [Krizhevsky u. a., 2012]. Similarly, VGGNet has a total of 138 million parameters, out of which 123 million parameters belong to FC layers [Simonyan und Zisserman, 2014]. This huge number of trainable parameters in FC layers are required to fit complex nonlinear discriminant functions in the feature space into which the input data elements are mapped. This fact However, this large number of parameters may result in over-fitting the classifier.

To improve the performance of CNNs, several methods using higher order kernel function than the ordinary linear kernel have been proposed in the literature. In [Cui u. a., 2017], a novel pooling method in the form of Taylor series kernel has been proposed. This method captures high order and non-linear feature interactions via compact explicit feature mapping. The approximated representation is fully differentiable, thus the kernel composition can be learned together with a CNN in an end-to-end manner. It acts as a basis expansion layers, increasing thereby the discrimination power of the FC layers. These methods have attracted increasing attentions, achieving better performance than classical first-order networks in a variety of computer vision tasks. Wang et al. [Wang u. a., 2019] focused more one the convolution part and they proposed to replace the convolution layers in a CNN by kernel-based layers, called kervolution layers. The use of these layers increases the model capacity to capture higher order features at the convolutional phase.

The remainder of this chapter is organized as follow: Section 4.2 reviews similar works which was proposed for incorporating kernel function for classification. Section 4.3 introduces the proposed KDL for FER. Section 4.4 presents the different conducted experiments and their related results. Section 4.5 concludes the chapter.

4.2 Kernelized classification

When working with non-linear problems, it's useful to transform the original vectors by projecting them into a higher dimensional space where they can be linearly separated. One of the most popular kernel based techniques in machine learning is the kernel Support Vector Machine (KSVM). It is a supervised learning algorithm mostly used for classification. The main idea is that based on the labeled data (training data) the algorithm tries to find the optimal hyperplane which can be used to classify new data points. The kernel SVMs return the inner product between two points in a suitable feature space. Thus by defining a notion of similarity, with little computational cost even in very high-dimensional spaces, according to the following equation:

$$K(X_i, X_j) = (X_i X_j + C)^n \tag{4.2}$$

where C ($C \in \mathbb{R}^+$) is constant and n ($n \in \mathbb{Z}^+$) is the polynomial order.

Elisseeffet al. [Elisseeff und Weston, 2001] presented a Support Vector Machine (SVM) like learning system to handle multi-label problems. Such problems are usually decomposed into many two-class problems but the expressive power of such a system can be weak [McCallum, 1999, Schapire und Singer, 2000]. It is based on a large margin ranking system that shares a lot of common properties with SVMs.

One of the earliest attempts to connect neural networks and kernel methods was the sigmoid kernel [Vapnik Vladimir, 1995], which became popular in SVMs due to the early success of the neural networks. This kernel was inspired by the sigmoid activation used in the early generations of neural networks. More recently, the authors of [Cho und Saul, 2009] proposed a family of kernel functions that mimic the computation in multilayer neural nets, and showed their usage in multilayer kernel networks. Chen et al. [Chen

u. a., 2018] proposed a simple yet effective framework for point set feature learning by leveraging a nonlinear activation layer encoded by Radial Basis Function (RBF) kernels. It models the spatial distribution of point clouds by aggregating features from sparsely distributed RBF kernels. A typical RBF kernel, e.g. Gaussian function, naturally penalizes long-distance response and is only activated by neighboring points. Such localized response generates highly discriminative features given different point distributions. In addition, it allows joint optimization of kernel distribution and its receptive field, automatically evolving kernel configurations in an end-to-end manner.

In this chapter, we build upon these works and introduce a novel FC layer. We leverage kernel functions to build a neuron unit that applies a higher order function on its inputs instead of calculating their weighted sum. The proposed Kernelized Dense Layers (KDL) permits to improve the discrimination power of the full network and it is completely differentiable, allowing an end-to-end learning. The experimental results demonstrate the benefits of such layer in FER task and show that our model achieves competitive results with respect to the state-of-the-art approaches.

4.3 kernelized Dense Layer

The proposed kernelized Dense Layer is similar to a classical neuron layer in the way that it applies a dot product between a vector of weights and an input vector, add a bias vector ($b \geq 0$) and eventually applies an activation function. The difference from standard FC layers is that our proposed method applies higher degree kernel function instead of a simple linear dot product, which allows the model to map the input data to a higher space and thus be more discriminative than a classical linear layer.

Figure 4.1 shows the processing of an elementary unit (kernel neuron) of our proposed KDL. Formally, the output Y is computed by applying a kernel function K on an input vector $x = \{x_1, x_2, \ldots, x_n\}$ and the corresponding vector of weights $W = \{w_1, w_2, \ldots, w_n\}$ and, adding the bias vector ($b \geq 0$).

In this work, we employed two different kernel functions which have an Euclidean structure $\mathbb{R}^d$.

- **Linear kernel:**

$$\mathcal{K}(x, w) = x^T w + b, \quad x, w \in \mathbb{R}^d, b \geq 0. \tag{4.3}$$

The linear kernel (Equation 4.3) looks at the similarity between the input vector x and the filter weight vector w.

- **Polynomial kernel:**

$$\mathcal{K}(x, w) = (x^T w + b)^n, \quad x, w \in \mathbb{R}^d, b \geq 0. \tag{4.4}$$

Starting form $n > 1$ in Equation 4.4, the polynomial kernel K encodes not only the

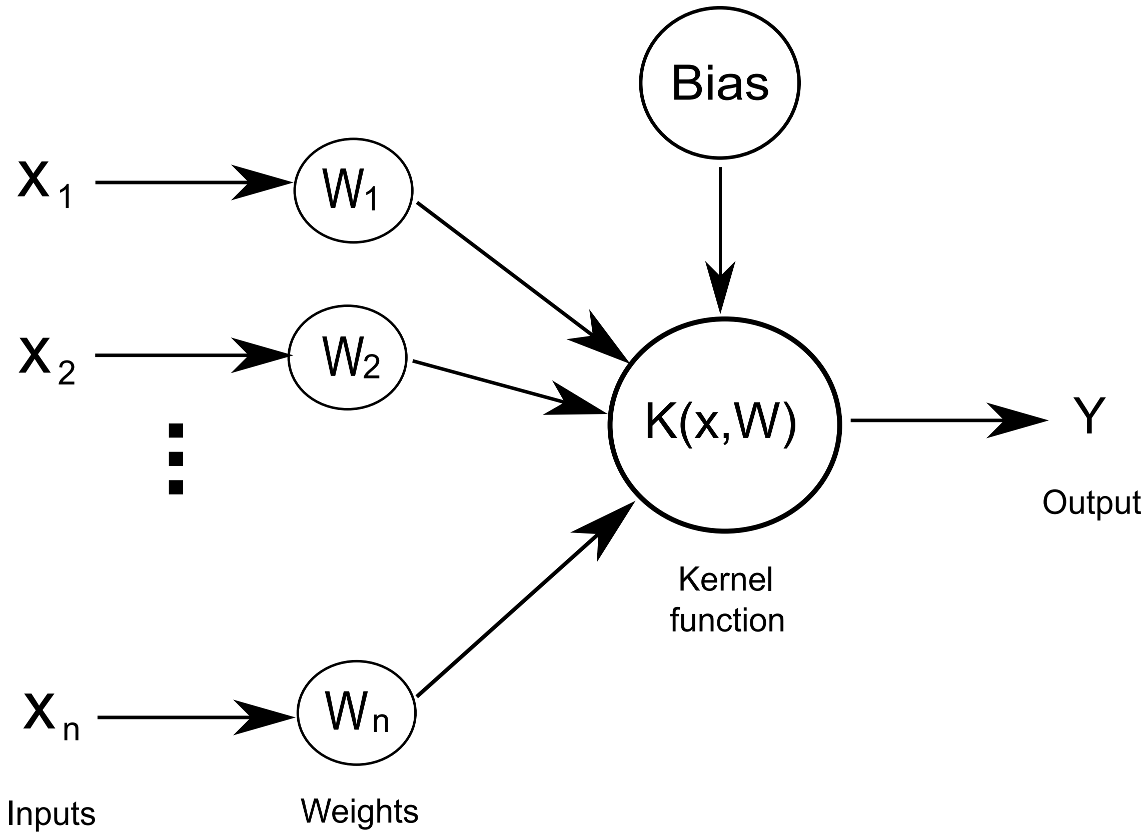


Figure 4.1 – *The basic unit of our proposed KDL is a kernel neuron. It applies a kernel function on an input vector* $x = \{x_1, x_2, \ldots, x_n\}$ *and a vector of weights* $w = \{w_1, w_2, \ldots, w_n\}$*, adds a bias term and eventually applies an activation function.*

linear relation between both x and w vectors, but also non-linear relations between them. It corresponds to an inner product in a feature space based on some mapping φ:

$$\langle \varphi(x), \varphi(w) \rangle \approx \mathcal{K}(x, w). \tag{4.5}$$

Note that in the case of polynomial kernel with degree $n > 1$, we do not apply an activation function on the neuron output, since non linearity is already added by the high polynomial kernel degree.

Other kernels that expand the non-linearity to the infinity can also be used. For instance, the Gaussian kernel defined by $\mathcal{K}(x, w) = e^{-\frac{\|x-w\|^2}{2\sigma^2}}, \quad x, w \in \mathbb{R}^n, \sigma > 0$, Laplacian kernel defined by $\mathcal{K}(x, w) = e^{-\alpha\|x-w\|}$, or the Abel kernel defined as $\mathcal{K}(x, w) = e^{-\alpha|x-w|}$, where, $\alpha > 0$ in both kernels.

4.3.1 Datasets

Our experiments have been conducted on three well-known facial expression datasets: RAF-DB [Li u. a., 2017], ExpW [Zhang u. a., 2018b] and FER2013 [Goodfellow u. a., 2013].

Facial expression datasets contain few classes that are nearly identical, which makes the recognition process more challenging. The only pre-processing which we have employed on all experiments is cropping the face region and resizing the resulting images to 100×100 pixels.

4.3.2 Training process

We have used Adam optimiser with a learning rate varying from 0.001 to 5e-5. This learning rate is decreased by a factor of 0.63 if the validation accuracy does not increase over ten epochs. To avoid over-fitting we have first augmented the data using a range degree for random rotations of 20, a shear intensity of 0.2, a range for random zoom of 0.2 and randomly flip inputs horizontally. We have also employed earl stopping if validation accuracy does not improve by a factor of 0.01 over 20 epochs. Each KDL of our model is initialized with *He* normal distribution and a weight decay of 0.0001.

4.4 Experiments

In order to demonstrate the efficiency of the proposed KDL, we built a simple CNN from scratch rather than using a pre-trained one. As shown in Figure 4.2, our model architecture is composed of five convolutional blocks. Each block consists of a convolution, batch normalization and rectified linear unit activation layers. The use of batch normalization [Zou u. a., 2019] before the activation brings more stability to parameter initialization and achieves higher learning rate. Each of the five convolutional blocks is followed by a max pooling layer and a dropout layer. Finally, two KDL are added on top of these convolution blocks with respectively 128 and 7 units. On the former, we apply a ReLU activation function in the case of linear kernel only, since non-linearity is already added by the polynomial kernel. Softmax activation function is finally applied on the last KDL.

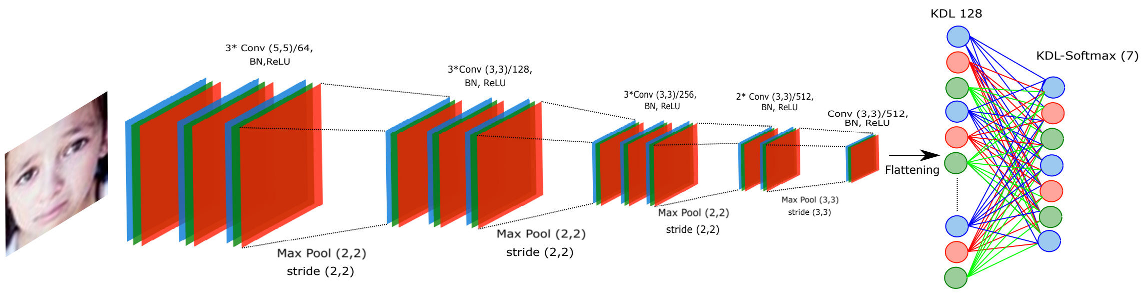

KDL 128
3* Conv (5,5)/64, BN,ReLU
3*Conv (3,3)/128, BN, ReLU
3*Conv (3,3)/256, BN, ReLU
2* Conv (3,3)/512, BN, ReLU
Conv (3,3)/512, BN, ReLU
KDL-Softmax (7)
Flattening
Max Pool (2,2) stride (2,2)
Max Pool (2,2) stride (2,2)
Max Pool (2,2) stride (2,2)
Max Pool (3,3) stride (3,3)

Figure 4.2 – *Base model architecture: it is composed of five convolutional blocks. Each block consists of a convolution, batch normalization and rectified linear unit activation layers. Each of the five convolutional blocks is followed by a dropout layer. Finally, two KDL are added on top of these convolution blocks with respectively 128 units and ReLU activation and 7 units with softmax activation.*

4.4.1 Ablation Study

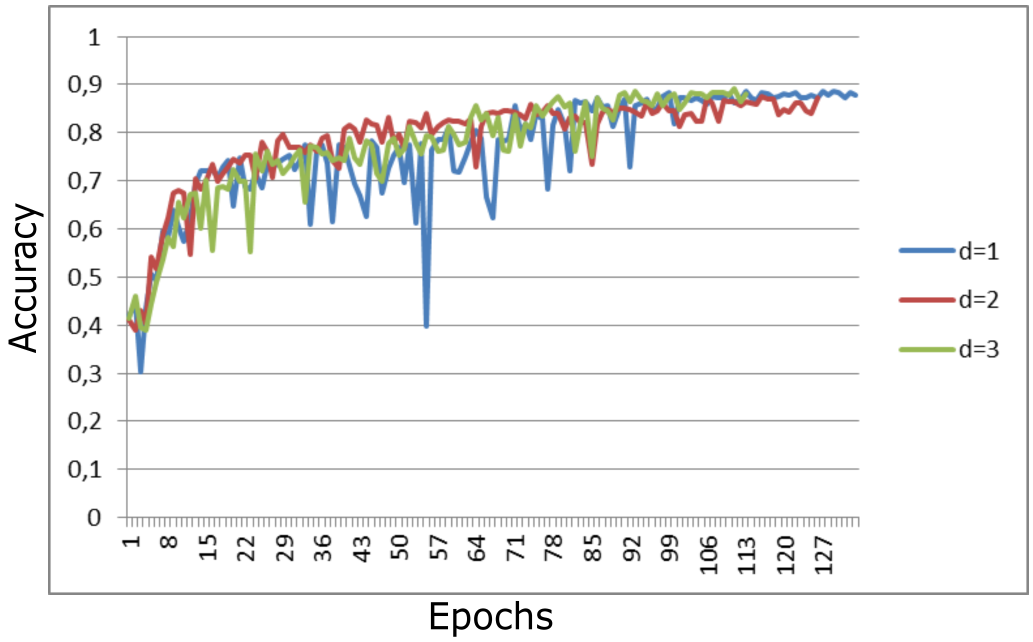
This section explores the impact of the use of the proposed KDL on the overall accuracy of a CNN model. We evaluated the performance using the same network architecture with different FC techniques. First, we used our model with standard FC layers which gives the same results as kernel neuron layers with a polynomial function with degree (n=1). After that, we replaced these FC layers by our KDL. We studied the behaviour of two different kernel functions, namely; the second-order polynomial kernel and the third-order polynomial kernel. The experiments are conducted with the same training parameters as described above.

Table 4.1 presents the results of our model using standard FC layers with comparison to the proposed KDL. In the case of the standard FC layers (Base-Model-FC), our base model attains 70.13%, 75.91% and 87.05% of accuracy rate on respectively FER2013, ExpW and RAF datasets, while the use of KDL considerably increases its accuracy. Indeed, one can notice that the accuracy for the second-order polynomial kernel increases for about 0.7% for FER2013, 0.25% for ExpW and 0.6% for RAF-DB. In the same way, using the third-order polynomial kernel increases further the overall accuracy. Compared to standard FC layers, the third-order polynomial KLD enhances the model accuracy for about 1.15% for FER2013, 0.75% for ExpW and 1% for RAF-DB. These results are consistent with previous work [Wang u. a., 2019], where the third-order applied on convolution layers gave the best performance. Although the computational complexity of these kernels is higher compared to the standard layers, they allow to strongly improve the model accuracy. These results demonstrate that the use of KDL, in the case of FER problem, are beneficial for the overall accuracy of the model. These techniques enhance the discriminative power of the model, compared to a standard FC layer.

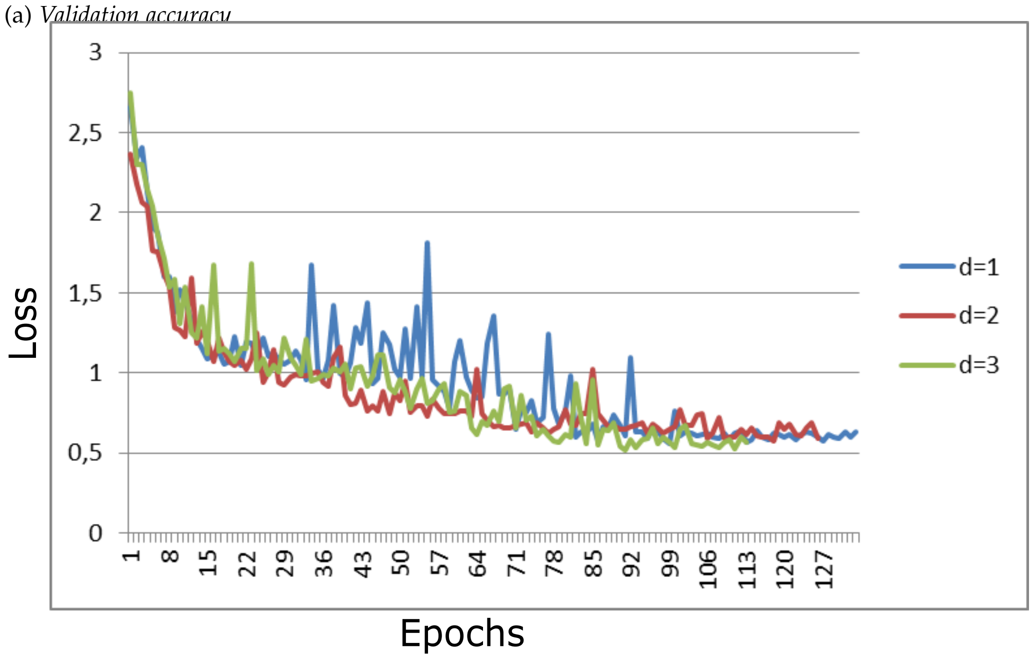
Table 4.1 – *Accuracy Rates of the proposed approach*

Models	**Dataset**		
	FER2013	***ExpW***	***RAF-DB***
Base-Model-FC	70.13%	75.91%	87.05%
Base-Model-KDL[a] (n=1)	70.09%	76.87%	87.03%
Base-Model-KDL[a] (n=2)	70.85%	76.13%	87.64%
Base-Model-KDL (n=3)	**71.28%**	**76.64%**	**88.02%**

[a]KDL: Kernelized Dense Layer.

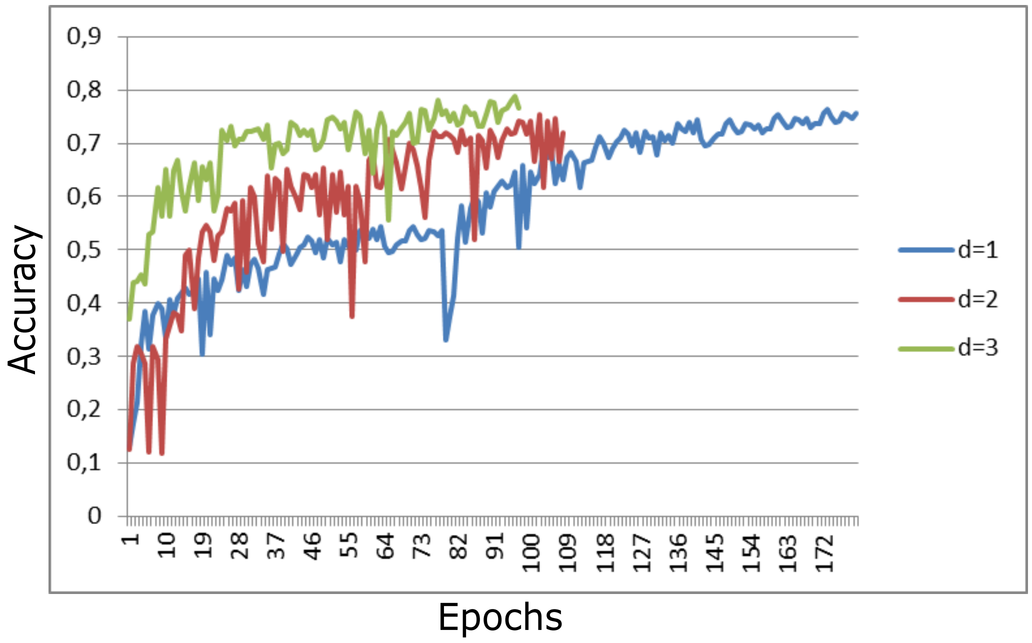


(a) *Validation accuracy*

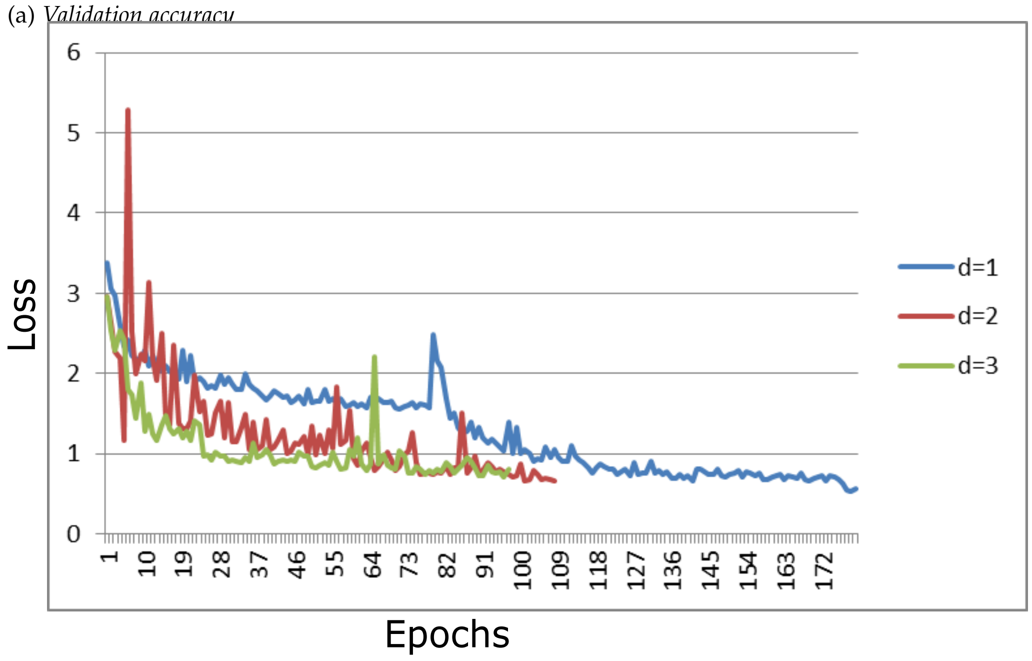


(b) *Validation loss*

Figure 4.3 – *Validation accuracy and validation loss on RAF-DB with the three kernel configurations.*



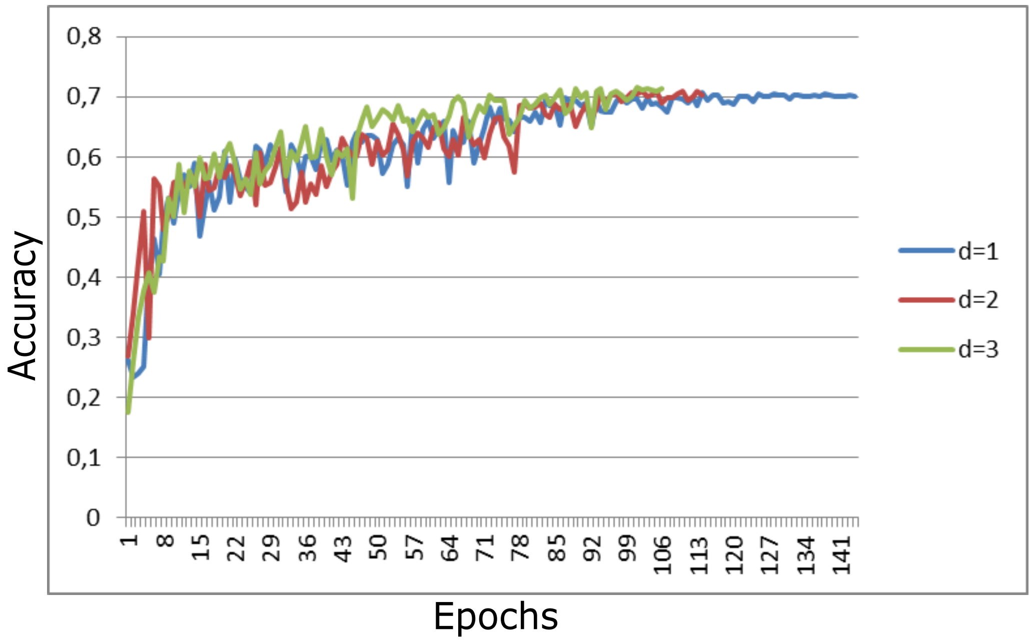
(a) *Validation accuracy*



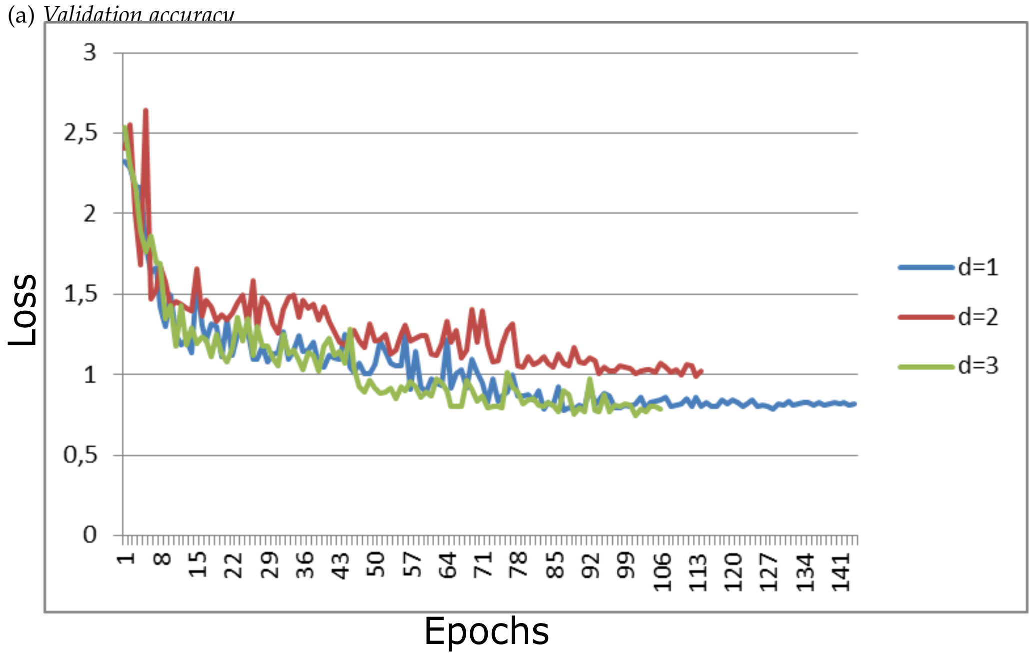
(b) *Validation loss*

Figure 4.4 – *Validation accuracy and validation loss on ExpW with the three kernel configurations.*

Another beneficial aspect of using KDL is the speed of convergence. As shown in Figures 4.3, 4.4 and 4.5, the higher degree is the kernel function, the fast it converge. Due to the use of early stopping in our training process, the learning process is interrupted as soon as the model begins to overfit. As can be seen, the higher is the kernel function degree, the sooner it stops training (the blue, red and green curves correspond, respectively, to n=1, n=2 and n=3). High degree kernel functions are known to be prone to overfitting, which in our case limits the number of units and layers of our proposed KDL.



(a) *Validation accuracy*



(b) *Validation loss*

Figure 4.5 – *Validation accuracy and validation loss on FER2013 with the three kernel configurations.*

Table 4.2 – *Accuracy rate of the proposed approach and state-of-the-art approach*

Models	**Dataset**		
	FER2013	***ExpW***	***RAF-DB***
Base-Model-FC	70.13%	75.91%	87.05%
Base-Model-KDL[a] (n=1)	70.09%	76.87%	87.03%
Base-Model-KDL[a] (n=2)	70.85%	76.13%	87.64%
Base-Model-KDL (n=3)	71.28%	**76.64**%	**88.02**%
Tang et al. [Tang, 2013]	71.16%	–	–
Guo et al. [Guo u. a., 2016]	71.33%	–	–
Kim et al. [Kim u. a., 2016]	**73.73%**	–	–
Bishay et al. [Bishay u. a., 2019]	–	**73.1%**	–
Lian et al. [Lian u. a., 2020]	–	71.9 %	–
Acharya et al. [Acharya u. a., 2018a]	–	–	**87%**
S Li et al. [Li und Deng, 2018b]	–	–	74.2%
Z.Liu et al. [Liu u. a., 2017b]	–	–	73.19%

[a]KDL: Fully connected kernel.

4.4.2 Comparison with the State-of-the-Art

In this section, we compare the performance of the KDL with CNN to several state-of-the-art FER methods. The obtained results are reported in Table 4.2. As can be seen, KDL with CNN outperforms the state-of-the-art methods on the ExpW dataset. The best accuracy rate is 76.64% and has been reached using the third-order polynomial KDL. Second-order polynomial KDL gives, for his turn, 76.13%. Whereas the state-of-the-art methods [Bishay u. a., 2019] reached **73.1%**.

On RAF-DB dataset, the accuracy of our models is also superior to state-of-the-art methods. The best accuracy rate is 88.02% and has been reached using the third-order polynomial KDL. Second-order polynomial KDL gives, for his turn, 87.64%. Whereas the state-of-the-art methods [Acharya u. a., 2018a] gives **87%**.

For FER2013, even thought using the KDL improves considerably the models accuracy, the obtained results are still under the state-of-the-art results. The best accuracy rate for this dataset, namely 71.28%, was reached using third-order KDL. Despite the improvement, the result obtained is 2.5% less than the state-of-the-art method [Kim u. a., 2016] (**73.73**%) but remains competitive.

4.5 Conclusion

In this chapter, we designed Kernelized Dense Layer for CNN model that aims to enhance the discriminative power of the overall model. It consists of applying higher order kernel method than the standard FC layer. Experimental results on ExpW, RAF-DB and FER2013 datasets demonstrate the efficiency of the proposed KDL compared to standard FC layer in terms of convergence, speed and overall accuracy. The proposed FER method outperforms most of the state-of-the-art methods and remains competitive. The performance of our model is essentially due to its capability of capturing high order information that are crucial for fine-grained classification tasks such as the FER.

5 Kernel function impact on convolutional neural networks

Contents

This chapter investigates the usage of kernel functions at the different layers in a convolutional neural network. We carry out extensive studies of their impact on CNN layers. We show how one can effectively leverage kernel functions, by using previously introduced Learnable Pooling Wheights (LPW) layers as well as Kernelized Dense Layers (KDL). The experiments on conventional classification datasets i.e. MNIST, FASHION-MNIST and CIFAR-10, show that the proposed techniques improve the performance of the network compared to classical convolution, pooling and fully connected layers. Moreover, experiments on fine-grained classification i.e. facial expression databases, namely RAF-DB, FER2013 and ExpW demonstrate that the discriminative power of the network is boosted, since the proposed techniques improve the awareness to slight visual details and allows the network reaching state-of-the-art results.

5.1 Introduction

Image classification has always been one of the most studied operation of computer vision. Several methods have been proposed in the literature addressing this problem, which consist in efficiently assigning the correct label to an image. Recently, with the emergence of Convolutional Neural Network (CNN), the computer vision community has witnessed an era of blossoming result thanks to the use of very large training databases. These databases contain a very large number of different images (i.e. objects, animals...etc). This advance encouraged the computer vision community to go beyond classical image classification that recognizes basic-level categories. The new challenge consists of discriminating categories that were considered previously as a single category and have only small subtle visual differences. This new sub-topic of image classification, called fine-grained image classification, is receiving a special attention from the computer vision community [Liu u. a., 2020, Tang u. a., 2020, Chen u. a., 2020, Ji u. a., 2020, Wang u. a., 2020, Huang und Li, 2020, Gao u. a., 2020, Zhuang u. a., 2020]. Such methods aims at discriminating between classes in a sub-category of objects like birds species [Welinder u. a., 2010], models of cars [Krause u. a., 2013a], and facial expressions, which makes the classification more difficult due to the high intra-class and low inter-class variations. State-of-the-art approaches typically rely on convolutional neural network as classification backbone and propose a method to improve its awareness to subtle visual details.

A CNN is mainly a stack of three different types of layers: convolution layers, pooling layers and fully-connected layers. Each of these types of layers perform specific task. Convolution layers are the core building block of a CNN by leveraging the fact that an input image is composed of small details, or features, and create a mechanism for analyzing each feature in isolation, which makes a decision about the image as a whole. Pooling layers, on the other hand, are used for the gradual spatial down-sampling of the feature map by reducing the number of parameters and thus decreases both the consumption of the memory and the complexity of computing . In addition, pooling layers widen the receptive field size of the intermediate neurons which allow the latter to receive data from a larger area of the image. These two layers are usually used in alternation until getting the most size-effective representative feature which is finally fed into a fully connected neural network in order to take a final classification decision.

CNNs have been used for a multitude of visual tasks. They showed to perform very competitive results while linear operations are used at different layers of the network. Linear functions are efficient, particularly, when the original data is linearly separable, which should have, in general, a high dimensional representation. In such a case, the decision boundary can be representable as a linear combination of the original features. It is worth noting that not every high dimensional problems are linearly separable [Robert, 2014]. For instance, images may have a high dimensional representation, but individual pixels are not very informative. Moreover, taking in consideration only small regions of the image, dramatically reduces their dimension, which makes linear functions less sensitive to subtle changes in input data. The ability of detecting such differences is crucial essentially for fine-grained recognition.

To overcome these limitations, some researches investigated different ways to include non-linear functions in CNNs. Starting from non linear activation functions like ReLU, eLU [Clevert u. a., 2015], SeLU [Klambauer u. a., 2017] and more recently [Sitzmann u. a., 2020]. Moreover, some recent work intended to replace the underlying linear function of a CNN by non linear kernel function without resorting to activation functions. For instance, some of them replaced convolution layers ([Zoumpourlis u. a., 2017], [Wang u. a., 2019]), while others replaced the pooling layers ([Lin u. a., 2015], [Tenenbaum und Freeman, 2000], [Lin und Maji, 2017], [Gao u. a., 2016], [Cui u. a., 2017], [Gao u. a., 2019a], [Hyun u. a., 2019]). The higher order kernel function are, the more susceptible they are to fit slight changes in data. We leverage this kernel function property to find the best use of these functions in different level of the CNN.

In this chapter, we investigate the usage of kernel functions at the different layers in a CNN. We carry out extensive studies of their impact on convolutional, pooling and fully-connected layers. For this purpose, we first replace the convolution operation in CNNs by a non-linear kernel function similarly to Kervolution [Wang u. a., 2019]. We further used the learnable pooling weights proposed in chapter 3, and kernelized dense layers proposed in chapter 4. We explore different dispositions and configurations of these layers to find which configuration gives the best accuracy without over-fitting

5.2 Study design

In this section, we present the different proposed kernel based techniques which can be plugged into a CNN. These techniques consist in new feature extraction, pooling and classification layers that can be used in the same manner as the usual CNN layers. Furthermore, these novel layers can be used solely or jointly with the usual CNN layers. This flexibility makes them usable in any architecture or even plugged at any level of a pre-trained CNN model. The novelty in this work is the use of higher order kernel function to replace the underlying function of each layer. These kernel functions allow them to perform the same task and brings additional features.

In the following, we give details on how kernel functions can be employed at each level of a CNN. As shown in Figure 5.1, we replace each layer type of an ordinary CNN (Fig 5.1-(a)) with a higher order kernel layer (Fig 5.1-(b)). More precisely, we replace convolution layers with Kervolution layers, Max/AVG pooling layers with Learnable weights pooling layers, and fully connected layers with Kernelized dense layers KDL.

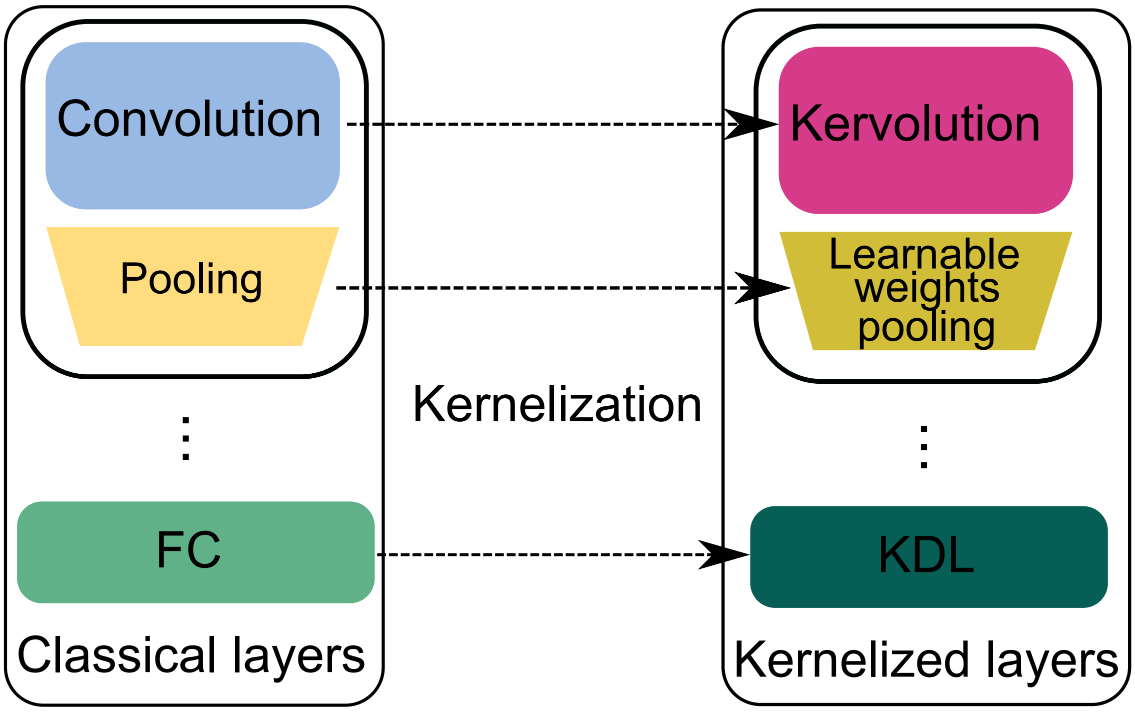


Figure 5.1 – *The different configurations to replace each layer type of an ordinary CNN with a higher order kernel layer, notably Kervolution, Learnable weights pooling and kernelized dense layers KDL.*

5.2.1 Kervolution

Kervolution has been proposed by Wang et al [Wang u. a., 2019]. It extends the convolution operation which computes the dot product between an input vector X and a weight vector W, and adds eventually a bias term, according to Equation 5.1:

$$C_{o,i,j} = (X \times W)_{i,j} = \sum_g \sum_h X_{g+i,h+j} W_{o,g,h} + B_o, \tag{5.1}$$

Where o corresponds to the output size, i and j are specific locations in the input, g and h are respectively the width and height of the convolution filter and B is a bias term.

Convolution is a linear operation that usually requires adding an activation function to introduce non-linearity. Without these activation functions the CNN performance drops dramatically. Kervolution leverages this fact and proposes to replace the convolution operation in CNNs by a non-linear function that performs the same task as convolution without resorting to activation functions. In this work, we use Kervolution with three kernel functions:

1. **Linear kervolution** which corresponds to the convolution function (Equation 5.1);

2. **Polynomial kervolution:**

$$K_{o,i,j} = \langle X, W \rangle_{i,j} = \sum_g \sum_h (X_{g+i,h+j} W_{o,g,h} + C)^n + B_o \tag{5.2}$$

where C ($C \in \mathbb{R}^+$) is a learnable constant and n ($n \in \mathbb{Z}^+$) is the polynomial order, it extends the feature space to n dimensions;

3. **Gaussian RBF kervolution:**

$$K_{o,i,j} = e^{-\frac{\|X_{g+i,h+j} - W_{o,g,h}\|^2}{2\sigma^2}} \tag{5.3}$$

where σ ($\sigma \in \mathbb{R}^+$) is a hyperparameter to control the smoothness of decision boundary. It extends kervolution to infinite dimensions.

The linear kernel measures the similarity between the filter weight vector W and the feature map vector X. However, when $n > 1$ in Equation 5.2, the polynomial kernel encodes the non-linear relations between both X and W vectors, in addition to the linear relation between them. In the case of the Gaussian kernel (Equation 5.3), the non-linearity is expanded to the infinity. For this purpose, we can also use other kernels, such as the the Abel kernel defined as $\mathcal{K}(X, W) = e^{-\alpha|X-W|}$, or the Laplacian kernel defined by $\mathcal{K}(X, W) = e^{-\alpha\|X-W\|}$, where, $\alpha > 0$ in both kernels.

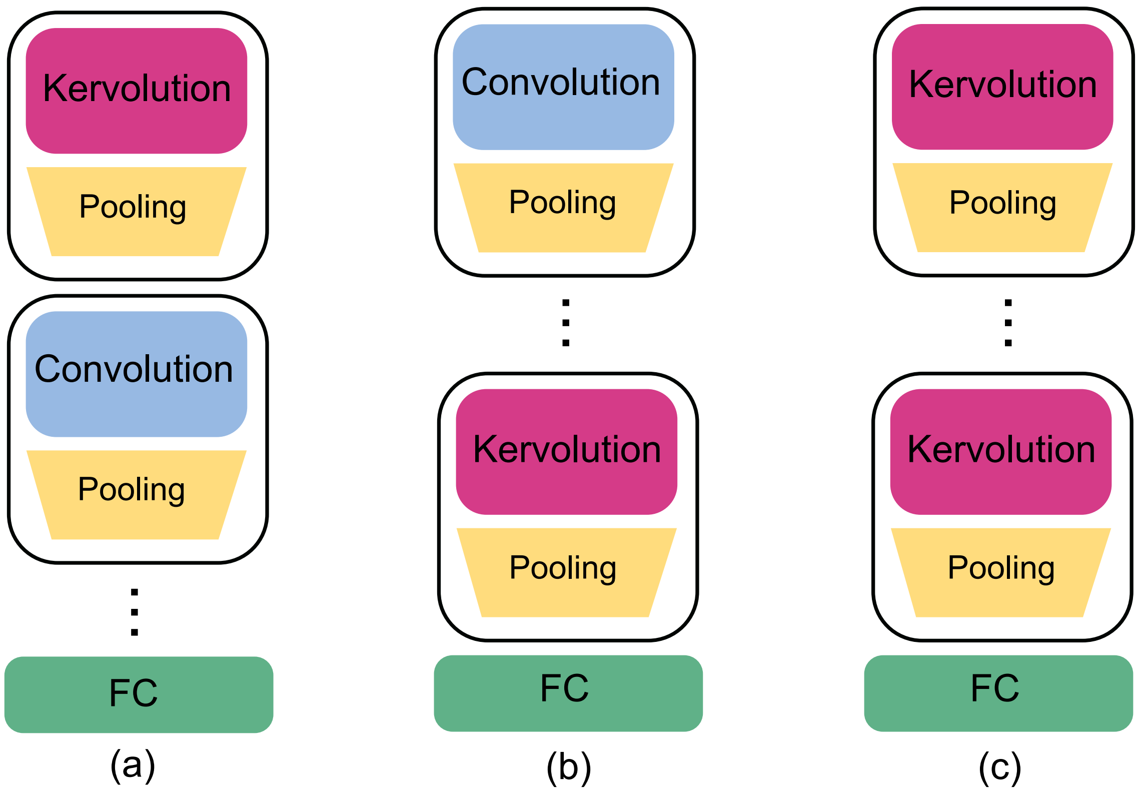


Figure 5.2 – *The three study configurations of kervolution layers, namely: (a) one kervolution layer at the beginning of the network, (b) one kervolution layer at the end of the network and (c) an end-to-end kervolution network.*

Kervolution layers are as flexible as convolution layers and can be plugged at any level of a CNN. In this chapter, we use three configurations to study the kervolution layer impact on CNNs. As shown in Figure 5.2, we use one kervolution layer at the beginning of the network (Fig 5.2-(a)), one kervolution layer at the end of the network (Fig 5.2-(b)), and an end-to-end kervolution network (Fig 5.2-(c)).

5.2.2 Learnable Weights Pooling

In order to efficiently study the impact of our proposed pooling layer on a CNN, we test it following the three configurations. As shown in Figure 5.3, we study the impact of one Learnable weights pooling layer after the first convolution layer (Fig 5.3-(a)), one Learnable weights pooling layer after the last convolution layer (Fig 5.3-(b)), and an end-to-end Learnable weights pooling network (Fig 5.3-(c)).

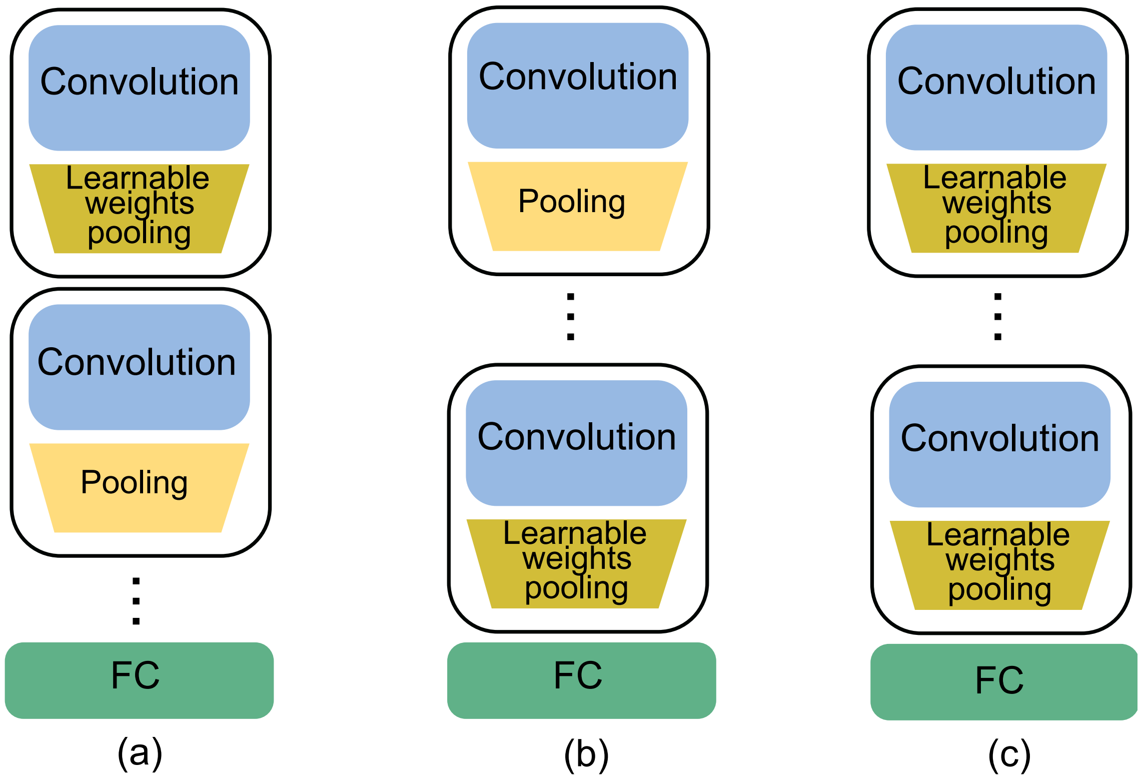

Convolution
Learnable weights pooling
Convolution
Pooling
FC
(a)
Convolution
Pooling
Convolution
Learnable weights pooling
FC
(b)
Convolution
Learnable weights pooling
Convolution
Learnable weights pooling
FC
(c)

Figure 5.3 – *The three study configurations of Learnable weights pooling layers, namely: (a) one Learnable weights pooling layer after the first convolution layer, (b) one Learnable weights pooling layer after the last convolution layer, and (c) an end-to-end Learnable weights pooling network.*

5.2.3 Kernelized dense layer

The inputs of the fully-connected layers consist of the result of the subsequent alternation of convolution and pooling layers. These input values goes through the first fully-connected layer where we multiply them by weights. After that, we apply an activation function (i.e ReLU). Finally, they goes forward through the output layer, in which each neuron represents a classification category. The fully-connected layers back-propagates the most accurate weights where every neuron gets weights that prioritize the most relevant category.

In this chapter, we use a novel dense layer composed of neuron that uses a kernel function instead of the usual dot product, Kernelized Dense Layer (KDL), proposed in [Mahmoudi u. a., 2020]. In contrary to a classical neuron layer where it computes a dot product between an input vector and a vector of weights, add a bias vector ($b \geq 0$), KDL applies higher degree kernel function, which permits to the latter to map the input data to a higher space and therefore be more discriminative than a usual linear layer.

In the case of KDL there is only one possible configuration to study the impact of the later on the overall accuracy and convergence of a CNN, which is a fully KDL after the succession of convolution and pooling.

5.3 Experiments

In this section, we evaluate the performance of the kernel-based layers described above, in terms of accuracy rate and convergence speed. Several experiments have been conducted on six well-known datasets, following the configuration shown in Figure 5.2 and 5.3. Note that for KDL, only one configuration can be tested, since they can be plugged only at the end of the network. In the following, we detail our experiments process. First we describe the datasets used to evaluate our approach (sec.5.3.1). After that we define the training process of our networks (sec.5.3.2). Then we discuss the obtained results (sec.5.3.3). Finally, we compare our results to state-of-the-art results.

5.3.1 Datasets

Our experiments have been conducted on six well-known datasets. In order to study different characteristics of kernel functions in CNN, these datasets have been grouped in two categories. The first category is used to study the accuracy enhancement and convergence speed brought by the kernel function. It is composed of the following three datasets: **MNIST**, **Fashion-MNIST** [Xiao u. a., 2017] and **CIFAR-10** [Krizhevsky u. a., 2009]. The second category is specially used to study the awareness of kernel function to subtle visual details. It is composed of three well-known fine-grained facial expression datasets: RAF-DB [Li u. a., 2017], ExpW [Zhang u. a., 2018b] and FER2013 [Goodfellow u. a., 2013]. Facial expression datasets contain few classes that are nearly identical, which makes the recognition process more challenging.

5.3.2 Training process

The datasets we have used for our experiments are not similar in terms image complexity. MNIST and Fashion-MNIST contain small and very simple images, while Cifar-10 and FER datasets contain larger complex images. We have thus decided to use two model architectures, as shown in Figures 5.4 and 5.5. These architectures are quite simple and can effectively run on cost-effective GPUs. Model-1 (Figure 5.4) is used for MNIST and Fashion-MNIST. It is composed of two blocks. Each one of these blocks is

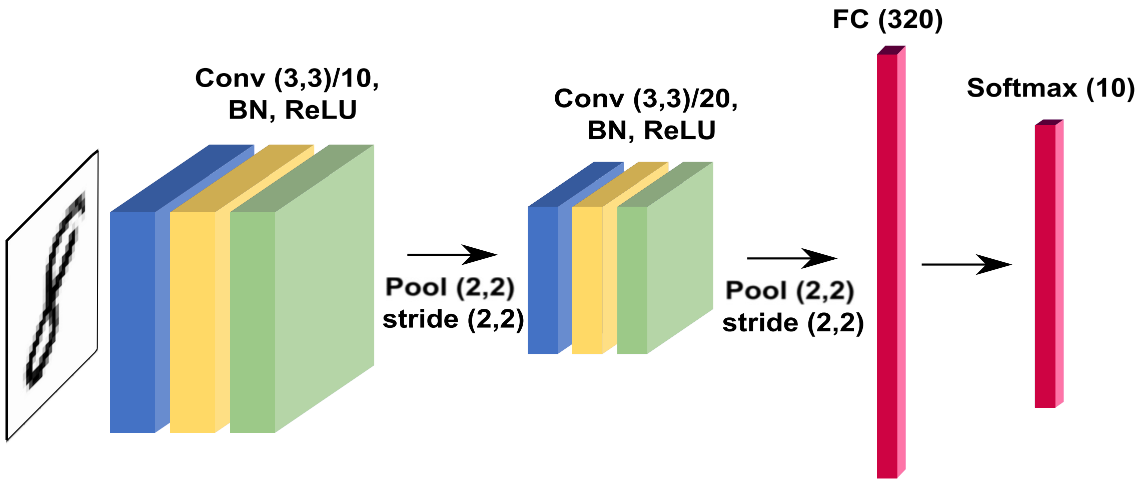


Figure 5.4 – *Base model-1 architecture: it is composed of two blocks. Each one of these blocks is composed of a convolution layer, a batch normalization layer, a dropout layer and ReLU activation. At the end, two fully-connected layers are added with respectively 320 units and ReLU activation and 10 units with softmax activation.*

composed of a convolution layer, a batch normalization layer, a dropout layer and ReLU activation. At the end, two fully-connected layers are added with respectively 320 units and ReLU activation and 10 units with softmax activation. On the other hand, model-2 (Figure 5.5) is used for CIFAR-10 and FER datasets. It is composed of five blocks. Each one of these block is composed of a convolution layer, a batch normalization layer, a dropout layer and ReLU activation. At the end, two fully-connected layers are added with respectively 128 units and ReLU activation and 10 units, for CIFAR-10, or 7 units for FER datasets with softmax activation.

We have used Adam optimiser with a learning rate starting from 0.001 decreased by a factor of 0.5 if the validation accuracy does not increase over two epochs for Model-1 and five epochs for Model-2. We trained Model-1 for 50 epochs and Model-2 for 100 epochs. To avoid over-fitting, for FER datasets, we used data augmentation with a shear intensity of 0.2, a range degree for random rotations of 20, randomly flip inputs horizontally and a range for random zoom of 0.2. We have also cropped the face region on FER datasets and resize the resulting images to 100×100 pixels.

5.3.3 Ablation Study

In this section, we explore the impact of the three kernel-based layers in terms of accuracy rate and convergence speed. We study the impact of each layer solely, in a full non-linear configuration network. After that, we test them jointly with the usual CNN layer, plugging them either at the beginning or at the end of the network. The kernel functions we have used are: (1) the linear kernel functions, (2) polynomial kernel functions from second degree to degree five and (3) the Gaussian RBF kernel with $\sigma = 0.9$. The accuracy rate results are illustrated in tables 5.1- 5.7 and convergence speed results are illustrated in figures 5.6- 5.12. In order to distinguish clearly the difference in convergence among

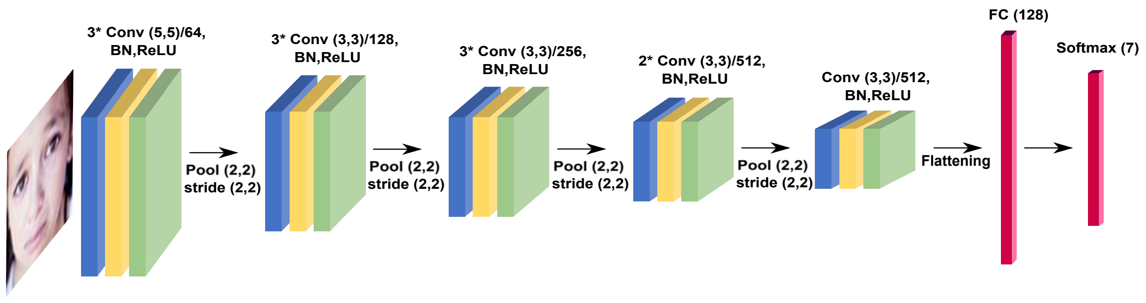


Figure 5.5 – *Base model-2 architecture: it is composed of five blocks. Each one of these block is composed of a convolution layer, a batch normalization layer, a dropout layer and ReLU activation. At the end, two fully-connected layers are added with respectively 128 units and ReLU activation and 10 or 7 units with softmax activation.*

kernel, we chose to plot only the first 20 epochs for MNIST, Fashion-MNIST and Cifar-10 datasets.

Kervolution

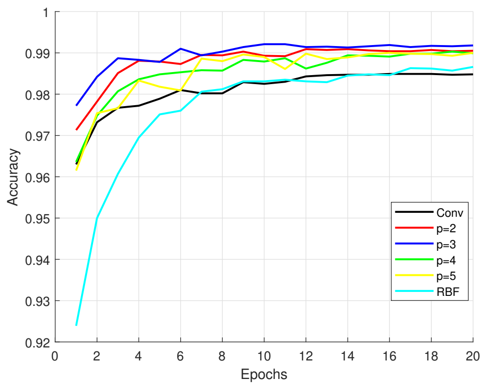
In this section we present the obtained results obtained with our two base models with a full Kervolution configuration (Table 5.1, Figure 5.6), a single Kervolution layer at the beginning of the network (Table 5.2, Figure 5.7), and a single Kervolution layer at the end of the network (Table 5.3, Figure 5.8).

Table 5.1 – *Accuracy rates of full kervolution networks*

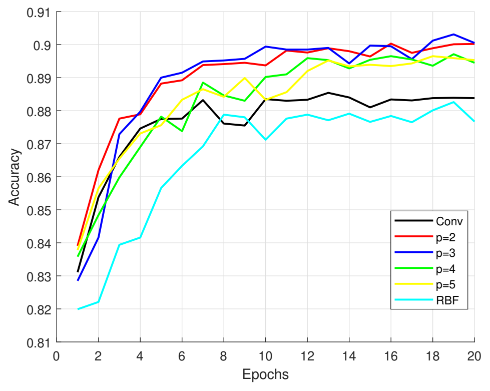
	Model-1		Model-2			
Layers configuration	MNIST	Fashion-MNIST	Cifar10	RAF-DB	FER2013	ExpW
Convolution (Linear kernel)	98.50%	88.29%	86.87%	87.05%	70.49%	75.91%
2^{nd}-order Poly	**99.05%**	**90.06** %	87.92%	**87.77%**	**70.68%**	**76.25%**
3^{rd}-order Poly	**99.16%**	**90.04** %	**88.64%**	**87.93%**	**70.95%**	**76.32%**
4^{nd}-order Poly	**99.02%**	89.57%	**88.41%**	**87.31%**	**70.82%**	**76.16%**
5^{rd}-order Poly	98.99%	89.71%	87.13%	86.04%	69.65%	75.80%
Gaussian RBF $\sigma = 0.9$	98.61%	88.78%	87.51%	**87.33%**	**70.78%**	**76.23%**

Table 5.1 illustrates the accuracy rates obtained using our two base models with a full kervolution configuration. In table 5.1 one can clearly notice that the use of kervolution layers enhance the overall accuracy of the network, especially with second, third and fourth order polynomial kernels. In comparison to full convolution network, the full kervolution network enhances the accuracy of Model-1 on MNIST dataset by 0.65% and 1.77% on Fashion-MNIST. On the other hand, the accuracy of Model-2 increases by 1.73% on CIFAR-10, 0.88% on RAF-DB, 0.46% on FER2013 and 0.41% on ExpW. The best accuracy rates are, in most cases, reached with third order polynomial kernel and decreases with higher order polynomial kernels. We have also noticed that Gaussian RBF kernel is more beneficial on fine-grained FER dataset than other datasets. Therefore, we can deduce that Gaussian RBF kernels are more sensitive to subtle details.

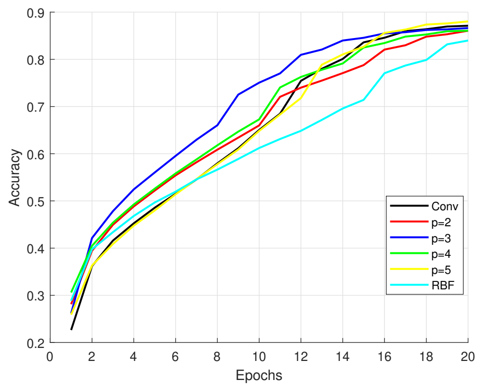
Figure 5.6 shows the validation accuracy rates evolution of our base models with full convolution and kervolution configurations. We can clearly notice that kervolution networks, especially second and third order polynomial kernels, converge in fewer epochs than convolution network. On the other hand, Gaussian RBF kernel takes more time to converge than the other kernels. We have also noticed that, for fine-grained FER datasets, the accuracy curve fluctuates accordingly to the kernel degree. This can be explained by the fact that kernels are more sensitive to subtle changes in input data. Therefore, any small change in the weights during the training phase can cause a big change in the final classification decision.



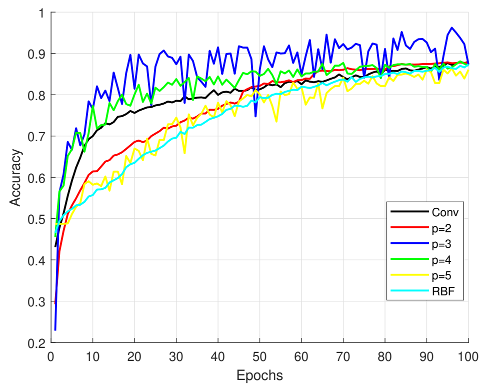
(a) *MNIST*



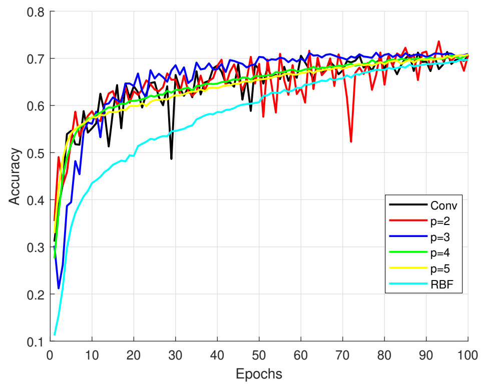
(b) *Fashion-MNIST*



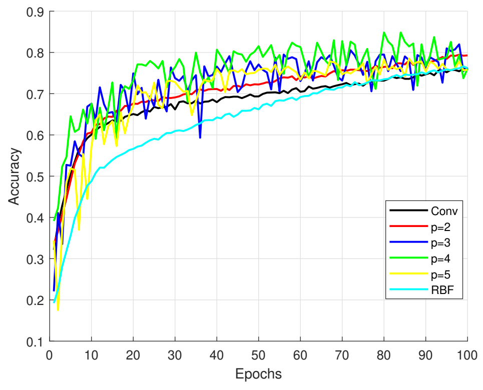
(c) *Cifar-10*



(d) *RAF-DB*



(e) *Fer2013*



(f) *ExpW*

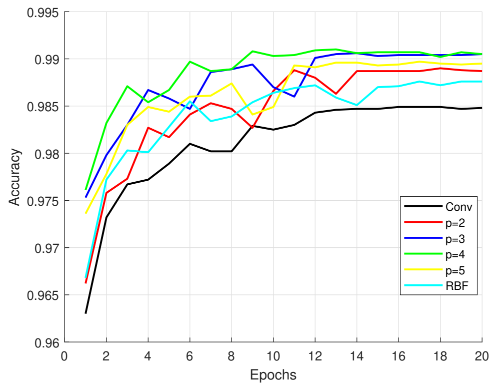
Figure 5.6 – *Convergence of full kervolution networks*

Table 5.2 – *Accuracy rates with a single kervolution layer at the begining*

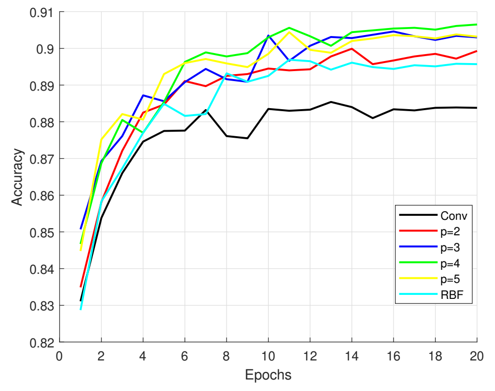
	Model-1		Model-2			
Layers configuration	MNIST	Fashion-MNIST	Cifar10	RAF-DB	FER2013	ExpW
Convolution (Linear kernel)	98.50%	88.29%	86.87%	87.05%	70.49%	75.91%
2^{nd}-order Poly	98.87%	89.74%	87.81%	87.62%	70.82%	76.64%
3^{rd}-order Poly	**99.04**%	**90.26**%	**88.73**%	**88.06**%	**71.06**%	**76.85**%
4^{nd}-order Poly	**99.04**%	**90.61**%	**88.52**%	**87.88**%	70.88%	**76.53**%
5^{rd}-order Poly	98.99%	**90.35**%	86.48%	86.71%	69.89%	75.56%
Gaussian RBF $\sigma = 0.9$	98.74%	89.53%	**88.48**%	**87.89**%	**70.98**%	**76.75**%

Table 5.2 presents the accuracy rate results of our two base models, in which we replaced the first convolution layer by a kervolution layer. In table 5.2 we can also notice that the use of one kervolution layer at the beginning of the network increases the accuracy of the network comparing to full convolution network. Furthermore, the use of only one kervolution layer in the beginning of the network allows to reach better results than full-kervolution network. Compared to full convolution network, the accuracy rate increases up to 1.97% on Fashion-MNIST, 1.86% on Cifar-10, 2% on RAF-DB, 0.57% on FER2013 and 0.94 on ExpW. Gaussian RBF kernels are slightly less accurate than third and fourth polynomial kernels, yet they also increase the accuracy rate compared to full-convolution network. They enhance the accuracy on Cifar-10 by 1.61%, 0.84% on REF-DB, 0.49% on FER2013 and 0.84% on ExpW. Another remark is that higher order kernels are more accurate than lower order kernel when used only at the beginning of the network. This may be explained by the fact that higher order kernels can fit to subtle details more efficiently than lower order kernels. Therefore, the network learns to detect more useful information than full-convolution network and will be less prone to over-fitting than full-kervolution network.

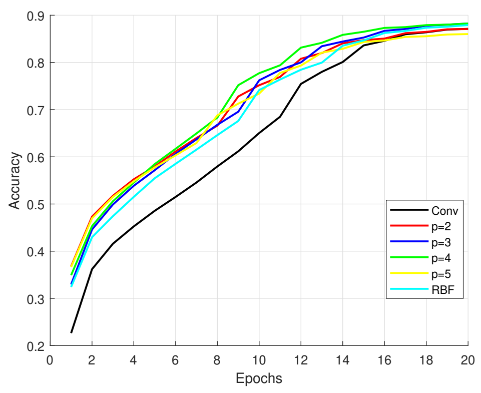
Figure 5.7 shows the accuracy convergence of our two base models, in which we replaced the first convolution layer by a kervolution layer. As illustrated in figure 5.7, using one kervolution layer at the beginning of the network allows the latter to converge in less time than the full convolution network. Similarly to full-kervolution configuration, second and third degree polynomial are the fastest to converge in all cases. Higher degree polynomial layers are also faster to converge than full convolution configuration except for FER datasets. On the other hand, Gaussian RBF layers are the slowest to converge, though they surpass convolution on MNIST, Fashion MNIST and Cifar-10.



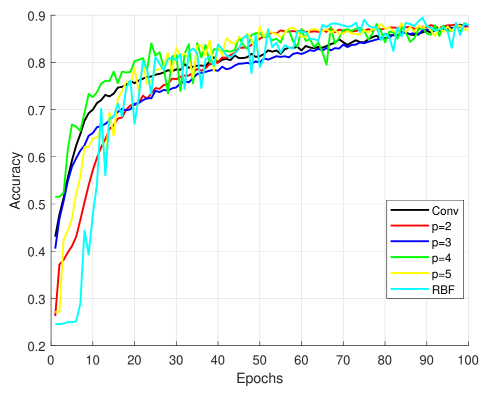
(a) *MNIST*



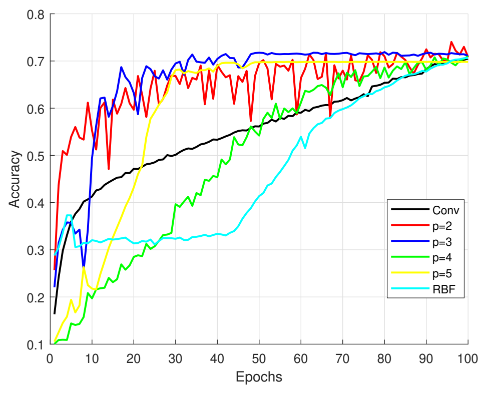
(b) *Fashion MNIST*



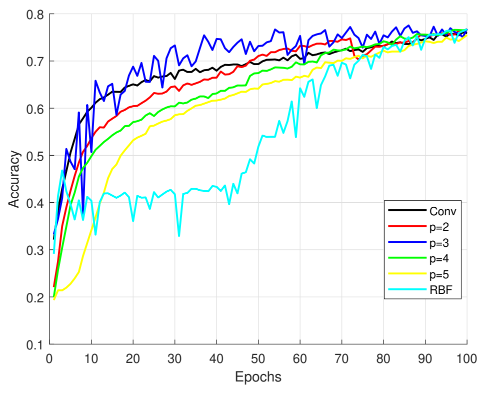
(c) *Cifar-10*



(d) *RAF-DB*



(e) *FER2013*



(f) *A ExpW*

Figure 5.7 – *Convergence of networks with one kervolution layer at the begining*

Table 5.3 – *Accuracy rates with a single kervolution layer at the end*

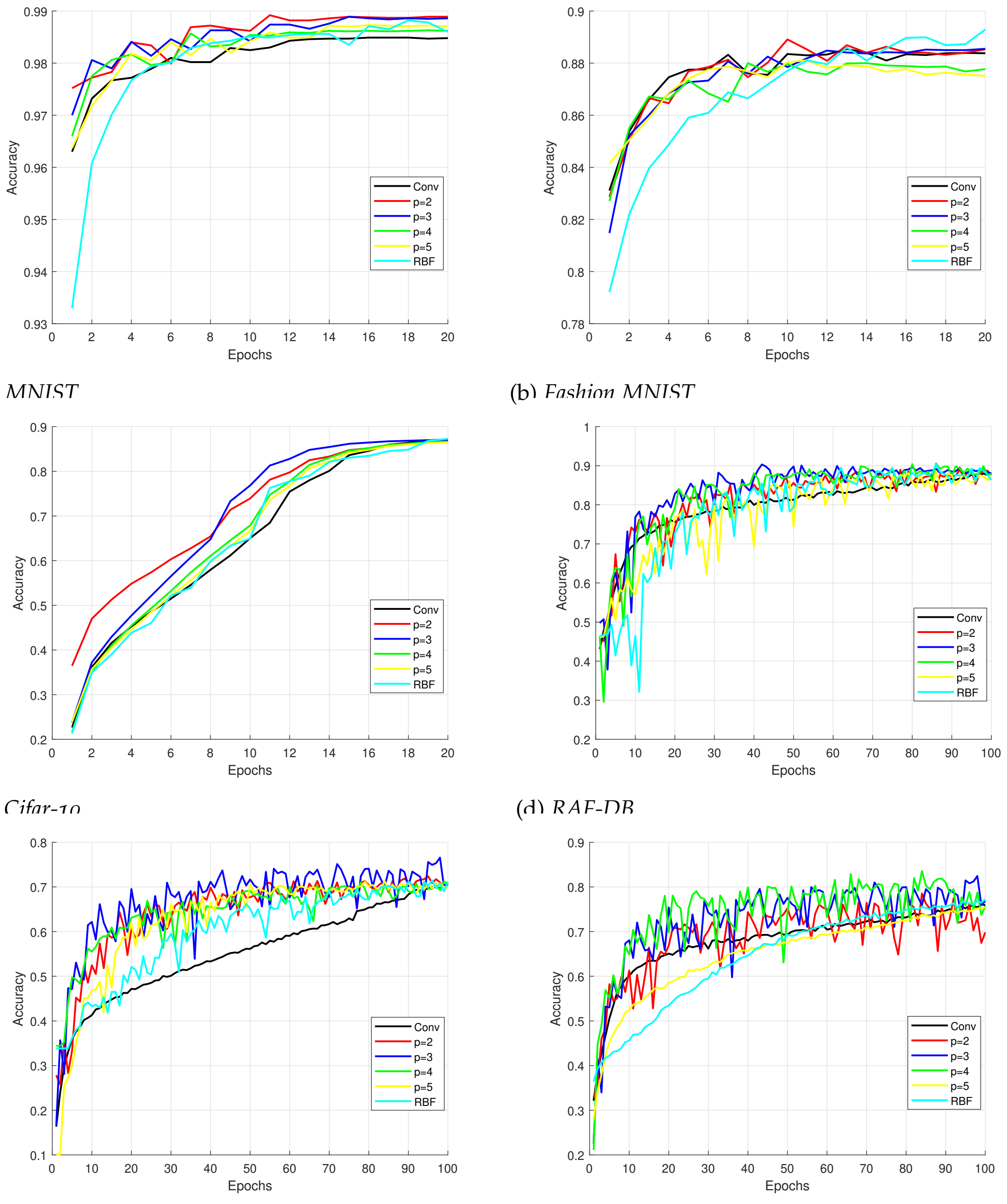
	Model-1		Model-2			
Layers configuration	MNIST	Fashion-MNIST	Cifar10	RAF-DB	FER2013	ExpW
Convolution (Linear kernel)	98.50%	88.29%	86.87%	87.05%	70.49%	75.91%
2nd-order Poly	98.89%	88.37%	87.11%	87.63%	70.81%	76.24%
3rd-order Poly	98.84%	88.45%	87.23%	88.03%	70.95%	76.82%
4nd-order Poly	98.62%	87.76%	87.09%	87.85%	70.75%	76.66%
5rd-order Poly	98.72%	87.45%	86.97%	87.15%	69.76%	75.45%
Gaussian RBF $\sigma = 0.9$	**98.75%**	**89.33%**	**90.13%**	**89.36%**	**71.15%**	**77.21%**

Table 5.3 presents the accuracy rate results of our two base models, in which we

replaced the last convolution layer by a kervolution layer. In table 5.3 one can notice that the use of polynomial kervolution layers at the end of the network decreases its accuracy compared to full-kervolution configuration and kervolution at the beginning. Yet, it performs better than full-convolution configuration. Indeed, we could surpass full-convolution configuration by 0.34% on MNIST, 0.16% on Fashion MNIST, 0.36% on Cifar-10, .0.98% on RAF-DB, 0.46% on FER2013 and 0.91% on ExpW. The best accuracy rates where reached with third degree polynomial kernel. Other polynomial kernels also perform slightly better than full-convolution configuration, but are less accurate than the other kervolution configurations. On the other hand, Gaussian RBF kernels increase remarkably the overall accuracy of the network compared to full-convolution configuration and other Gaussian RBF kervolution configurations. Indeed, it increases the overall accuracy rates by 0.25% on MNIST, 1.04% on Fashion MNIST, 3.26% on Cifar-10, 2.31% on RAF-DB, 0.66% on FER2013 and 1.30% on ExpW. With this has been said, we deduce that polynomial kervolution layers are more suited for feature extraction which explains why they are more accurate when plugged at the beginning of the network. On the other hand, Gaussian RBF kervolution layers are more beneficial when plugged at the end of the network than the beginning. This can be explained by the fact that Gaussian RBF kernels are more accurate for classification than for feature extraction.

Figure 5.8 shows the accuracy convergence of our two base models, in which we replaced the last convolution layer by a kervolution layer. We can clearly notice that the use of one kervolution layer at the end of the network quicken remarkably its convergence. This impact on convergence speed is valid with all kernels. On the other hand, even-though Gaussian RBF kernel reaches the highest accuracy rates among all kernels, it is still the slowest kernel to converge. Polynomial kernels, on the other hand, are still the fastest in convergence.

According to these results, we can say that the use of kervolution layers have clearly a beneficial impact on the accuracy rate and convergence speed of the network. As we have seen above, this positive impact is noticeable wherever kervolution layers are used. In terms of accuracy, using one kervolution layer at the beginning of the network shown to be the most efficient configuration. Whereas, in terms of convergence speed, using one kervolution layer at the end of the network shown to be the fastest to converge. Full-kervolution configuration is, for its part, average in both accuracy and convergence speed compared with the above configurations. In addition, full-kervolution configuration is more prone to overfitting than the other kervolution configurations.



(a) *MNIST* (b) *Fashion MNIST*
(c) *Cifar-10* (d) *RAF-DB*
(e) *FER2013* (f) *A ExpW*

Figure 5.8 – *Convergence of networks with one kervolution layer at the end*

Learnable weights pooling

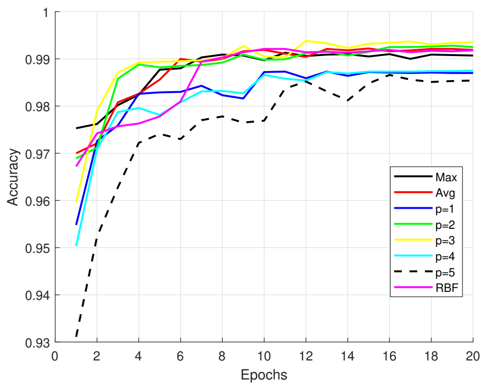
Similarly to kervolution, we explore the impact of using learnable pooling with three configurations, namely: full learnable pooling network (Table 5.4, Figure 5.9), learnable pooling after the first convolution block (Table 5.5, Figure 5.10) and learnable pooling at the end of the network (Table 5.6, Figure 5.11). This time, we compare the performance of the network with the usual pooling methods, namely: Max and average pooling.

Table 5.4 – *Accuracy rates of full learnable pooling network*

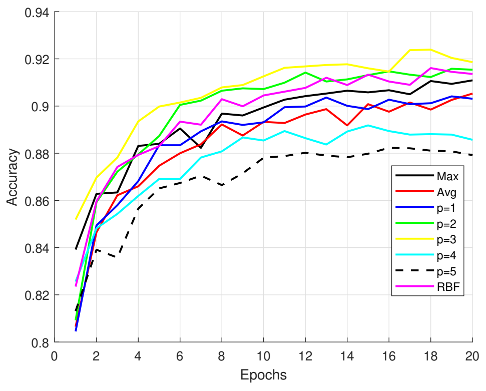
	Model-1		Model-2			
Layers configuration	MNIST	Fashion-MNIST	Cifar10	RAF-DB	FER2013	ExpW
Max	**99.05**%	**91.34**%	88.42%	87.05%	70.49%	75.91%
AVG	**99.17**%	**91.37**%	88.12%	86.89%	70.13%	75.74%
Linear kernel	98.85%	**90.05**%	**90.25**%	**90.81**%	70.96%	76.28%
2^{nd}-order Poly	**99.26**%	**91.42**%	**90.62**%	**92.87**%	70.88%	**76.64**%
3^{rd}-order Poly	**99.35**%	**91.79**%	**90.97**%	**93.21**%	**71.35**%	**76.81**%
4^{nd}-order Poly	98.73%	88.56%	89.88%	**93.03**%	**71.13**%	**76.73**%
5^{rd}-order Poly	98.52%	87.91%	89.61%	**92.64**%	69.91%	75.56%
Gaussian RBF $\sigma = 0.9$	**99.11**%	**91.32**%	**90.45**%	**92.74**%	70.74%	76.42%

Table 5.4 presents the results of our base models with learnable pooling only. One can clearly notice that learnable pooling enhance the accuracy of the network with all kernel functions. Second and third order polynomial kernels are the most accurate kernels for pooling with all datasets. With third order polynomial kernel Model-1 surpassed usual pooling methods by 0.18% on MNIST and 0.42% on Fashion-MNIST. On the other hand, Model-2 outperformed usual pooling method by 2.55% on Cifar-10, 6.15% on RAF-DB, 0.86% on FER2013 and 0.90% on ExpW. We can also notice that for fine-grained FER datasets, higher order polynomial kernels are also beneficial for the model for the same reasons cited for kervolution at the beginning of the network. Indeed, learnable pooling uses less weights than convolution or kervolution which allows it to fit subtle details without over-fitting. Finally, Gaussian RBF learnable pooling performs better than usual pooling methods even-though it does not outperform polynomial kernels. Although it did not surpass usual pooling method on model-1, it outperformed ordinary pooling methods with Model-2 by 2.03% on Cifar-10, 5.69% on RAF-DB, 0.25% on FER2013 and 0.51 on ExpW.

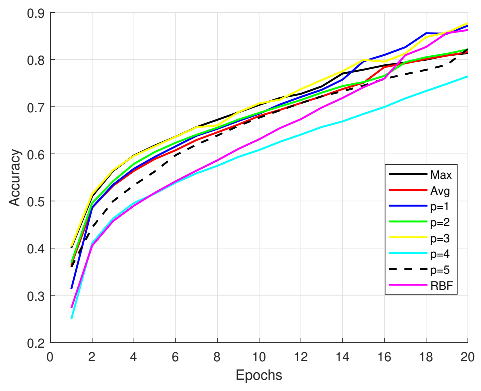
The convergence progression of full learnable pooling networks is shown in Figure 5.9. We can notice that second and third order polynomial kernels are the fastest to converge. On the other hand, higher order polynomial kernels convergence is the slowest among all kernels, even if they reach higher accuracy rates than Max and average pooling for FER datasets. Finally, Gaussian RBF kernel has similar accuracy progression to Max and average pooling.



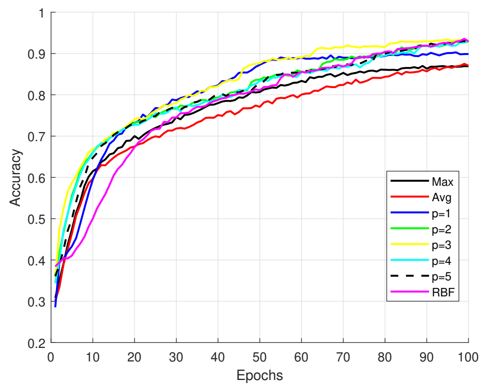
(a) *MNIST*



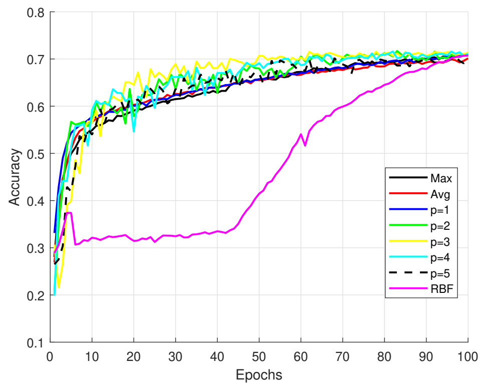
(b) *Fashion MNIST*



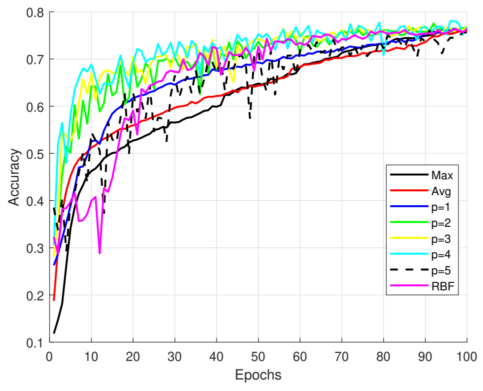
(c) *FER2013*



(d) *RAF-DB*



(e) *FER2013*



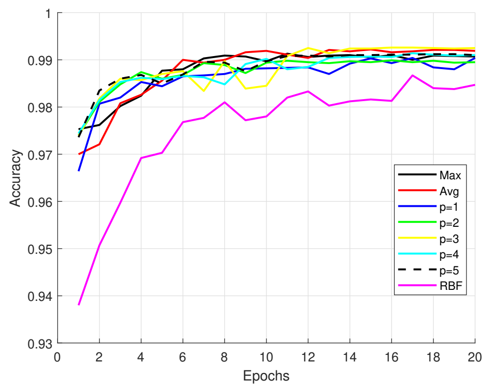
(f) *A ExpW*

Figure 5.9 – *Convergence of networks with learnable pooling*

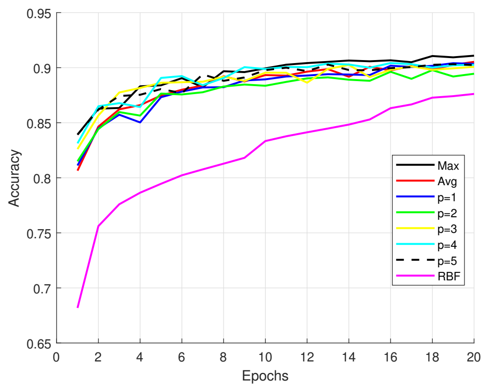
In table 5.5 we notice that using one learnable pooling layer after the first convolution block does not perform as well as the full learnable pooling network. Even-though linear kernel outperforms the usual pooling methods. As stated before, linear kernel pooling can learn a suitable pooling from a continuum of methods that ranges from average to max pooling. This is the reason why linear kernel pooling performs well wherever it is plugged.

Table 5.5 – *Accuracy rates networks with a single learnable pooling layer at the begining*

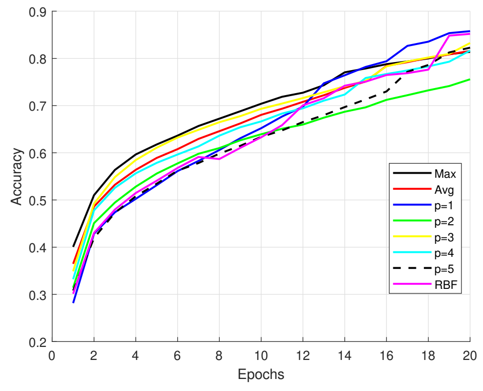
	Model-1		Model-2			
Layers configuration	MNIST	Fashion-MNIST	Cifar10	RAF-DB	FER2013	ExpW
Max	**99.05**%	**91.34**%	88.42%	87.05%	70.49%	75.91%
AVG	**99.17**%	**91.37**%	88.12%	86.89%	70.13%	75.74%
Linear kernel	**99.11**%	**90.58**%	**88.15**%	**87.19**%	**70.26**%	**75.95**%
2nd-order Poly	98.97%	89.80%	87.71%	86.92%	70.08%	75.82%
3rd-order Poly	**99.24**%	89.88%	87.54%	86.79%	69.92%	75.23%
4th-order Poly	**99.12**%	**90.32**%	87%	86.57%	69.79%	74.84%
5th-order Poly	**99.10**%	**90.24**%	86.89%	86.38%	69.54%	74.88%
Gaussian RBF $\sigma = 0.9$	98.43%	88.93%	85.71%	84.12%	68.63%	73.21%



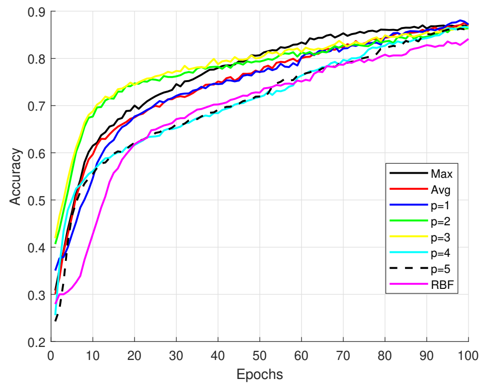
(a) *MNIST*



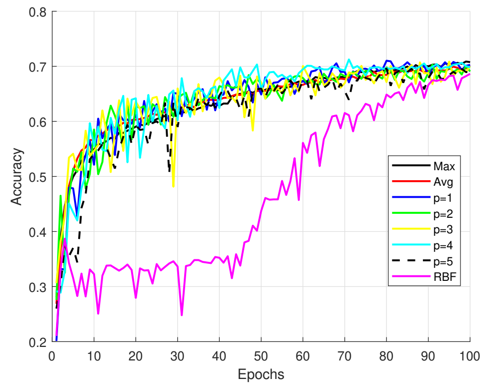
(b) *Fashion MNIST*



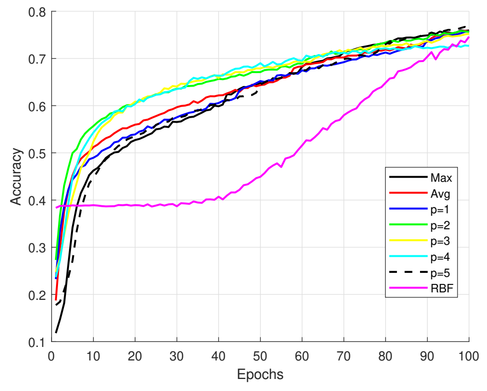
(c) *Cifar-10*



(d) *RAF-DB*



(e) *FER2013*



(f) *A ExpW*

Figure 5.10 – *Networks convergence with learnable pooling at the beginning*

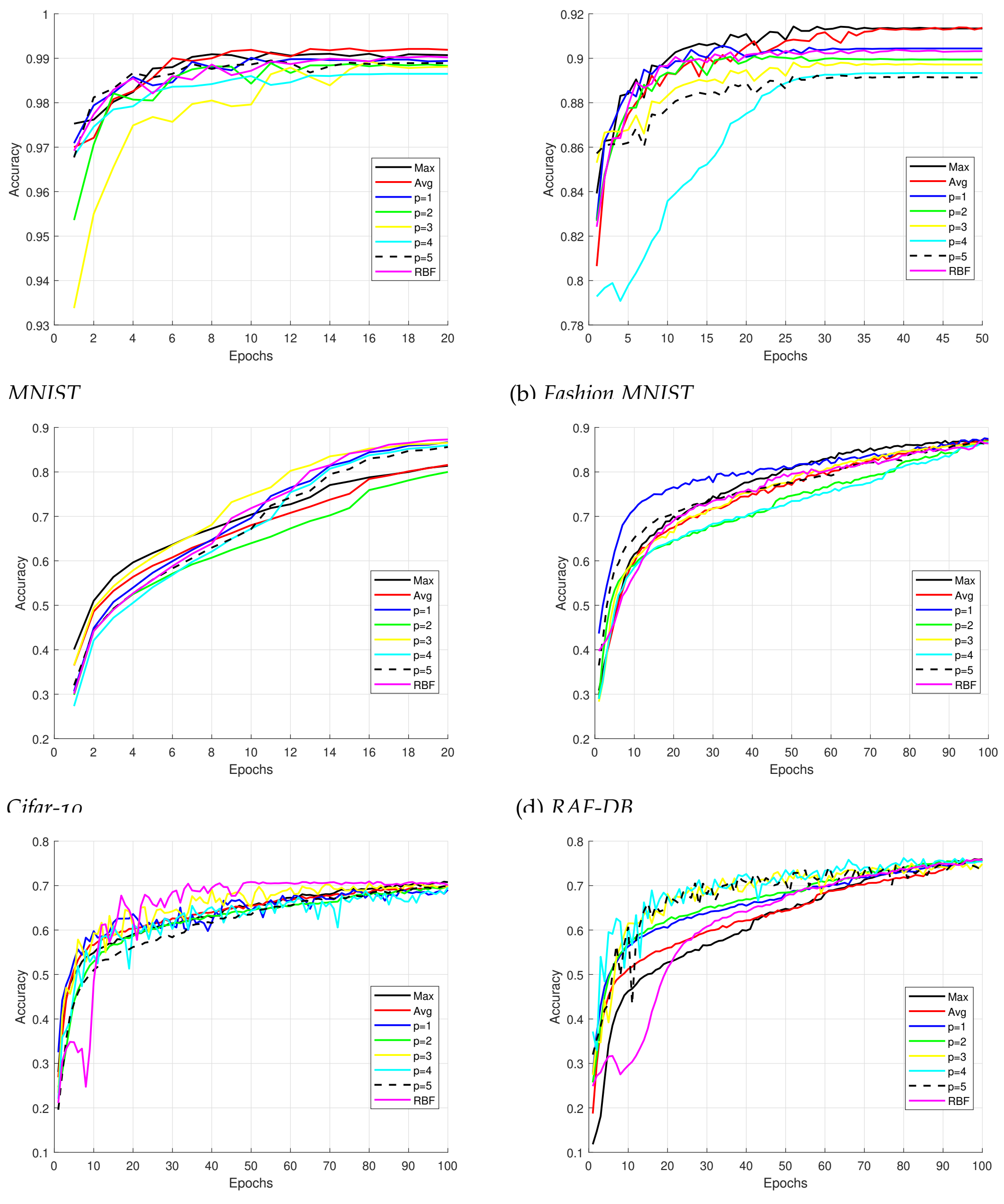
The convergence progression of learnable pooling at the beginning of the networks is shown in Figure 5.10. We can notice that polynomial kernels are, in general, the fastest to converge. Second and third order polynomial kernels are still the most efficient among these kernels. On the other hand, Gaussian RBF kernel is the slowest kernel to converge, specially for the biggest FER datasets (i.e. FER2013 and ExpW). This kernel is more sensitive to the learning rate used. As we can see in Figure 5.10, Gaussian RBF kernel does not start to converge until reaching a specific learning rate.

Table 5.6 – *Accuracy rates networks with a single learnable pooling layer at the end*

	Model-1		Model-2			
Layers configuration	MNIST	Fashion-MNIST	Cifar10	RAF-DB	FER2013	ExpW
Max	**99.05**%	**91.34**%	88.42%	87.05%	70.49%	75.91%
AVG	**99.17**%	**91.37**%	88.12%	86.89%	70.13%	75.74%
Linear kernel	98.93%	90.42%	87.52%	**87.12**%	**70.06**%	**75.87**%
2^{nd}-order Poly	98.85%	89.94%	87.28%	86.89%	69.87%	75.66%
3^{rd}-order Poly	98.78%	89.72%	87.04%	86.75%	69.71%	75.48%
4^{nd}-order Poly	98.62%	89.37%	86.79%	86.69%	69.58%	75.36%
5^{rd}-order Poly	98.90%	89.10%	86.63%	86.42%	69.22%	74.81%
Gaussian RBF $\sigma = 0.9$	**99.03**%	**90.36**%	**88.09**%	**86.94**%	**70.11**%	**75.79**%

Similarly to learnable pooling at the beginning of the network (Table 5.5), using one learnable pooling layer after the last convolution layer and before fully connected layers (Table 5.6) does not perform as well as the full learnable pooling network. One exception is the Gaussian RBF kernel which slightly performs better at the end than at the beginning. This last remark strengthens the deduction made on kervolution at the end of the network which states that Gaussian RBF kernel is best suited for classification than feature extraction or even pooling.

The convergence speed of learnable pooling at the end of the network is illustrated in Figure 5.11. One can clearly notice that kernel based pooling does not perform as well as when plugged at the beginning or in a full configuration. It performs similarly to the usual pooling methods. On the other hand, Gaussian RBF kernels have a faster convergence when used at the end of the network. Moreover, it outperforms other pooling method, specially with FER dataset. This confirms the assumption made above that Gaussian RBF kernel is more efficient when used just before the fully connected layers.



(a) *MNIST* (b) *Fashion MNIST*
(c) *Cifar-10* (d) *RAF-DB*
(e) *FER2013* (f) *ExpW*

Figure 5.11 – *Networks convergence with a single learnable pooling layer at the end*

kernelized Dense Layers

kernelized Dense Layers are only plugged at the end of the network since they replace the fully connected layers. Table 5.7 shows the result of using kernelized Dense Layers instead of the usual fully-connected layers. One can clearly notice that using kernel function for classification improves the accuracy of the network. For instance, polynomial kernel enhances the accuracy of the network for about 1% with fine-grained FER datasets. Third and Fourth order polynomial kernels are those who gave the best result with all datasets. On the other hand, Gaussian RBF kernel results confirm the deduction made trough this experimental section that it is best suited for classification than feature extraction. Whenever it is used at the end of the network it performs better than when used in the beginning.

Table 5.7 – *Accuracy rates of networks with KDL*

	Model-1		Model-2			
Layers configuration	MNIST	Fashion-MNIST	Cifar10	RAF-DB	FER2013	ExpW
Linear kernel	**99.07**%	**90.07**%	86.92%	**87.05**%	70.49%	75.91%
2nd-order Poly	**99.12**%	**90.73**%	87.61%	**87.64**%	70.85%	76.13%
3rd-order Poly	**99.11**%	**90.63**%	**89.37**%	**88.12**%	**71.28**%	**76.64**%
4nd-order Poly	**99.09**%	**90.30**%	**89.09**%	**87.83**%	**71.13**%	**76.42**%
5rd-order Poly	**99.01**%	**91.07**%	88.95%	86.93%	70.62%	75.86%
Gaussian RBF $\sigma = 0.9$	**99.23**%	**90.79**%	**89.11**%	**88.03**%	**71.06**%	**76.51**%

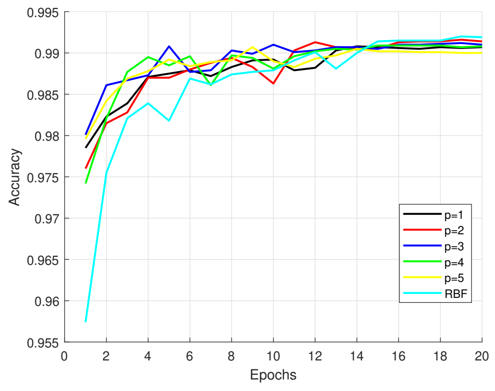
Figure 5.12 shows the convergence of networks with KDL. We can say that polynomial kernels converge faster than other kernels. Even-though second and third order polynomial reach better accuracy rates than other polynomial kernels, they have not better convergence. On the other hand, Gaussian RBF kernel shows the slowest convergence even if it gives the best accuracy rates.

We have also tried to combine these kernel-based methods together in the same network. The number of all possible configurations is very big, therefore we only show the configuration which gave the best results. These results are shown in Table 5.8. All the combinations tested shown a dramatically decreasing in performance. The only reasonable performances we could attain were using one kervolution layer at the beginning of the network and KDL at the end. Yet we have noticed that this configuration shows a clear over fitting.

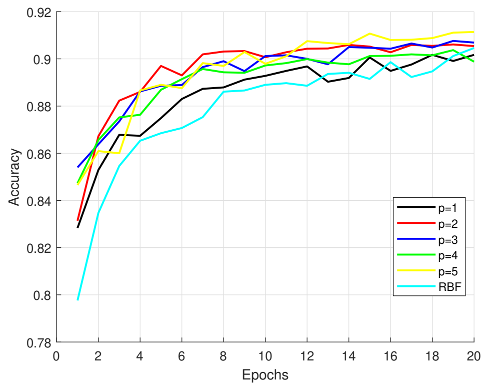
Table 5.8 – *Accuracy rates of networks with best combinations*

	Model-1		Model-2			
Layers configuration	MNIST	Fashion-MNIST	Cifar10	RAF-DB	FER2013	ExpW
Single Kervolution layer at the beginning and KDL						
2rd-order Poly (1kerv-KDL)	**99.16**%	**91.19**%	65.86%	72.81%	62.41%	58.72%
3rd-order Poly (1kerv-KDL)	98.96%	90.23%	65.75%	72.64%	61.73%	57.96%
4nd-order Poly (1kerv-KDL)	98.88%	89.64%	61.87%	69.58%	59.86%	57.41%
Gaussian RBF $\sigma = 0.9$ (1kerv-KDL)	98.57%	88.92%	61.20%	68.12%	59.12%	56.87%

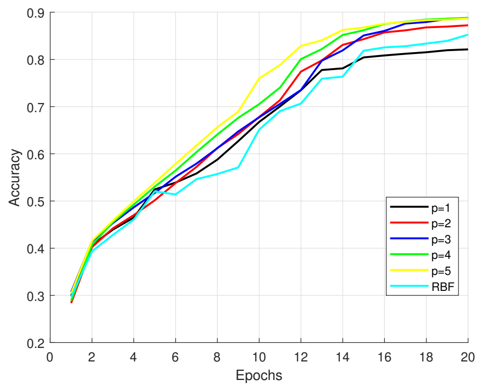
A more elaborated study on the use of kernel function on all over the network is presented in appendix A. This shows the performance of a deep kernelized network using kervolution, KDL and SVM on fine-grained and FER datasets.



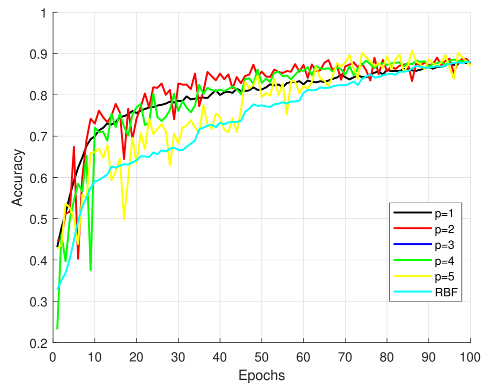
(a) *MNIST*



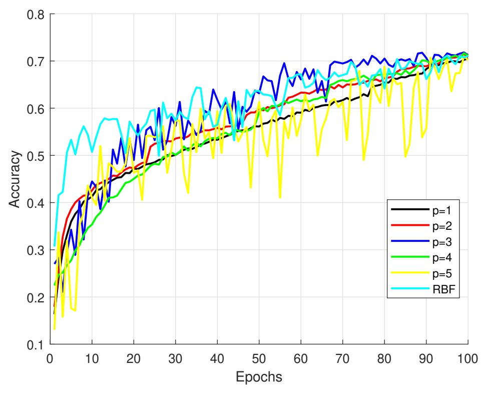
(b) *Fashion MNIST*



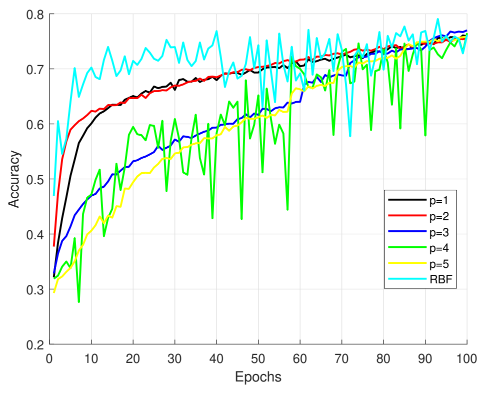
(c) *Cifar-10*



(d) *RAF-DB*



(e) *FER2013*



(f) *A ExpW*

Figure 5.12 – *Networks convergence with KDL*

5.3.4 Comparison with the State-of-the-Art

In this section, we compare the performance of the proposed methods, namely: kervolution, learnable pooling and KDL, with respect to several state-of-the-art methods. According to table 5.9, we obtained state-of-the-art results on two FER datasets (RAF-DB and ExpW) with all the proposed methods. For the remaining datasets namely MNIST, Fashion MNIST, Cifar-10 and FER2013, we could not surpass state-of-the-art results, yet we obtained very close results. In the following, we give the best results of each configuration.

With full kervolution network (Table 5.1), the highest accuracy rate we reached, was using third order polynomial kernel. It gave **99.16%** on MNIST, **87.93%** on RAF-DB and **76.32%** on ExpW. Kervolution attains better results when used at the beginning of the network (Table 5.2). With this configuration, we got **99.04** on MNIST, **88.73%** on RAF-DB and **76.85%** on ExpW using third order polynomial kernel. On the other hand, when using kervolution at the end of the network (Table 5.3), the Gaussian RBF reaches the best results. It attains **89.36%** on RAF-DB and **77.21%** on ExpW.

Learnable pooling is the method that allowed to reach the best results. With the full learnable pooling configuration (Table 5.4), we could reach the best results on all datasets using third order polynomial kernel. We got very close results to state-of-the-art with **99.35%** on MNIST, **91.79%** on Fashion MNIST, **90.97%** on Cifar-10 and **71.35%** on FER2013. We have also reached state-of-the-art result on RAF-DB and ExpW with respectively **93.21%** and **76.81%**. Using learnable pooling after the first convolution block (Table 5.5), we could not surpass full learnable pooling network. The best results were obtained with linear kernel with **87.19%** on RAF-DB and **75.95%** on Expw. The same results were obtained when using learnable pooling at the end of the network, with a slight improvement for Gaussian RBF kernel. Yet linear kernel remains the more accurate with **87.12%** for RAF-DB and **75.87%** for ExpW.

Finally, KDL (Table 5.7) gives very good results with all the used kernels. On MNIST dataset, all the results are above **99%** and reached **99.23%** with Gaussian RBF kernel. It also reached **88.03%** on RAF-DB and **76.51** on ExpW. The best KDL results were obtained with third order polynomial kernel. It reached **88.12%** on RAF-DB and **76.64%** on ExpW.

Table 5.9 – *Accuracy rates of networks with best combinations*

	Model-1		Model-2			
Layers configuration	MNIST	Fashion-MNIST	Cifar10	RAF-DB	FER2013	ExpW
Full kervolution networks						
3^{rd}-order Poly	**99.16%**	**90.04 %**	**88.64%**	**87.93%**	**70.95%**	**76.32%**
Gaussian RBF $\sigma = 0.9$	98.61%	88.78%	87.51%	**87.33%**	**70.78%**	**76.23%**
Single kervolution layer at the beginning						
3^{rd}-order Poly	**99.04%**	**90.26%**	**88.73%**	**88.06%**	**71.06%**	**76.85%**
Gaussian RBF $\sigma = 0.9$	98.74%	89.53%	**88.48%**	**87.89%**	**70.98%**	76.75%
Single kervolution layer at the end						
3^{rd}-order Poly	98.84%	88.45%	87.23%	88.03%	70.95%	76.82%
Gaussian RBF $\sigma = 0.9$	**98.75%**	**89.33%**	**90.13%**	**89.36%**	**71.15%**	**77.21%**
Full learnable pooling						
3^{rd}-order Poly	**99.35%**	**91.79%**	**90.97%**	**93.21%**	**71.35%**	**76.81%**
Gaussian RBF $\sigma = 0.9$	**99.11%**	**91.32%**	**90.45%**	**92.74%**	70.74%	76.42%
Learnable pooling at the beginning						
3^{rd}-order Poly	**99.24%**	89.88%	87.54%	86.79%	69.92%	75.23%
Gaussian RBF $\sigma = 0.9$	98.74%	89.53%	**88.48%**	**87.89%**	**70.98%**	**76.75%**
Learnable pooling at the end						
Linear kernel	98.93%	90.42%	87.52%	**87.12%**	**70.06%**	**75.87%**
Gaussian RBF $\sigma = 0.9$	**99.03%**	**90.36%**	**88.09%**	**86.94%**	**70.11%**	**75.79%**
Kernelized Dense Layer						
3^{rd}-order Poly	**99.11%**	**90.63%**	**89.37%**	**88.12%**	**71.28%**	**76.64%**
Gaussian RBF $\sigma = 0.9$	**99.23%**	**90.79%**	**89.11%**	**88.03%**	**71.06%**	**76.51%**
State-of-the-art results						
Byerly et al [Byerly u. a., 2020]	**99.84%**	–	–	–	–	–
Assiri [Assiri, 2020]	**99.83%**	–	–	–	–	–
Jayasundara et al [Jayasundara u. a., 2019]	**99.71%**	**96.36%**	–	–	–	–
Kolesnikov et al [Kolesnikov u. a., 2019]	–	–	**99.30%**	–	–	–
Huang et al [Huang u. a., 2019]	–	–	**99%**	–	–	–
Ridnik et al [Ridnik u. a., 2020]	–	–	**99%**	–	–	–
Tang et al. [Tang, 2013]	–	–	–	–	–	71.16%
Guo et al. [Guo u. a., 2016]	–	–	–	–	–	71.33%
Kim et al. [Kim u. a., 2016]	–	–	–	–	–	**73.73%**
Bishay et al. [Bishay u. a., 2019]	–	–	–	–	**73.10%**	–
Lian et al. [Lian u. a., 2020]	–	–	–	–	71.90 %	–
Acharya et al. [Acharya u. a., 2018a]	–	–	–	**87%**	–	–
S Li et al. [Li und Deng, 2018b]	–	–	–	74.20%	–	–
Z.Liu et al. [Liu u. a., 2017b]	–	–	–	73.19%	–	–

5.4 Discussion

In this chapter, we extensively studied the impact of using kernel functions on different levels of a CNN, in particular, convolution, pooling and fully-connected layers. We replaced convolution layer by a non-linear layer as introduced by Wang et al. [Wang u. a., 2019]. We tested the performance of this layer following three network configurations. We first tested them solely, in a full configuration network. After that, we tested them jointly with the usual convolution layers, plugging them either at the beginning of the network or at the end.

We have also used a novel pooling layer, based on kernel functions, that keeps the down-scaling aspect of the standard pooling function and brings various new features. This pooling layer relay on learnable weights that generalize ordinary pooling operations (i.e. average pooling and max pooling). Furthermore, it encodes patch-wise non-linearity. In this manner, the discrimination power of the full network is enhanced. The novel pooling, called learnable weights pooling, can be used at any level of the network and is fully differentiable, which allows the network to be trained in an end-to-end training. We have also explored their impact following the same network configuration as stated above.

We also use a novel fully-connected layer in which we use kernel functions to create a neuron unit that uses a higher degree kernel function on its inputs rather than computing the weighted sum. The used layer, called Kernelized Dense Layers (KDL), is also differentiable and demonstrates its usefulness in the improvement of the discrimination power of the full network.

For the three proposed layers, we explored their impact on the overall accuracy and convergence speed of the network. The results illustrated in Tables 5.1- 5.7 and Figures 5.6- 5.12 allow us to deduce the following. Kervolution layer is best suited for feature extraction and gives the best of its results when used at beginning of the network conjointly with convolution layers. It is more accurate with fine-grained FER datasets which means that it is more sensitive to subtle details. Adding more kervolution layers only increases over-fitting. While using kervolution at the end of the network decreases its performance. On the other hand, learnable pooling works best when used in full network configuration. It has the same advantages of kervolution in term of being more sensitive to subtle details with fewer parameters, which prevents it from over-fitting. The performance of learnable pooling decreases when used only at the beginning of the network or at the end, compared to the full network configuration. Finally, KDL is the most stable proposed layer. It shows good performance with all kernels and datasets.

In terms of kernels, we have also noticed that they may perform differently when used at different levels. For instance, polynomial kernels work best as feature extractors. They work better when used at the beginning of the network only; as stated in [Wang u. a., 2019]. On the other hand, Gaussian RBF kernels work best when used at the end of the network. Either as feature extraction or classification layer.

5.5 Conclusion

In light of the results obtained in the experiments, we have conducted in this work, we deduce that kernel functions impact positively the CNN performance. Indeed, the experimental results prove that the use of kernel functions, instead of the linear functions used in CNN layers, improves the accuracy rate of the whole model and its convergence speed. Furthermore, we concluded that the impact of these kernel functions varies according to the level in which they are plugged into and the specific kernel in use. We have also concluded that high order kernels prone to overfitting, whereas low order kernels might not be sufficiently effective to fit the input data distributions. Therefore, a trade-off between complexity and performance should be reached.

Expanding Convolutional Neural Network Kernel For Facial Expression Recognition 6

Contents

In this chapter, we propose a Facial Expression Recognition (FER) method, based on kernel enhanced Convolutional Neural Network (CNN) model. Our method improves the performance of a CNN without increasing its depth nor its width. It consists of expanding the linear kernel function, used at different levels of a CNN. The expansion is performed by combining multiple polynomial kernels with different degrees. By doing so, we allow the network to automatically learn the suitable kernel for the specific target task. The network can either uses one specific kernel or a combination of multiple kernels. In the latter case we will have a kernel in the form of a Taylor series kernel, which is more sensitive to subtle details than the linear one and is able to better fit the input data. Furthermore, this method uses the same number of parameters as a convolution layer or a dense layer.

6.1 Introduction

A CNN is mainly a stack of three different types of layers; namely convolution, pooling and fully connected layers. These layers use linear kernel functions in order to extract, down-sample and classify features, respectively. Let $x \in \mathbf{R}^n$ be an input vector fed into a layer L and $W \in \mathbf{R}^n$ its related weight vector. A linear kernel is represented as in equation 6.1.

$$K_{linear}(x, W) = \langle x, W \rangle, \tag{6.1}$$

where $\langle ., . \rangle$ is the inner product.

Linear functions are efficient, particularly when the original data is linearly separable. These data should have, in general, a high dimensional representation. In such a case, the decision boundary is likely to be representable as a linear combination of the original features. It is worth noting that not all high dimensional problems are linearly separable [Robert, 2014]. For example, images may have a high dimensional representation, but individual pixels are not very informative. Moreover, taking in consideration only small regions of the image, dramatically reduces their dimension. Thus making linear functions less sensitive to subtle changes in input data. However, the ability of detecting such details is crucial in fine-grained recognition, and particularly for FER. Such recognition application requires a method that can detect the finest features in input data.

To overcome this issue, facial expression recognition systems must be able to recognise this subtle differences efficiently. Researchers are trying to overcome this problem by either increasing the network size or by employing more complex functions. In the first case, researchers are continuously trying to enhance CNNs by increasing their depth (number of layers) or width (size of the output of each layer). Even though by doing so the performance of the network is effectively enhanced, it can not be a longstanding solution. Indeed, these methods drastically increase the number of weights and the network complexity. Therefore, the resulting models can only be used on powerful devices. In the second case, the focus is more on computation. Many researchers incorporated more complex functions in CNN, instead of simple linear functions, at different levels. These methods have the benefits of being less memory consuming, even though they are harder to train.

In this chapter, we propose to enhance CNN performance without increasing the number of parameters. Our method consists of expanding the linear kernel function, used at different level of a CNN. The expansion is performed by combining multiple polynomial kernels with different degrees. This method allows the network to automatically learn the suitable kernel for a specific task. The network can either uses one specific kernel or a combination of multiple kernels. In the latter case we will have a kernel in the form of a Taylor series kernel. This kernel function is more sensitive to subtle details than the linear kernel and is able to better fit the input data. The sensitivity to subtle visual details is a key factor for a better facial expression recognition. Furthermore, this method uses the same number of parameters as a convolution layer or a dense layer.

The remainder of this chapter is organized as follows: Section 6.2 introduces the proposed expansion method for convolution and fully connected layers. Section 6.3 presents our experiments setting, the datasets we used and their related results. Finally, Section 6.4 concludes the chapter.

6.2 Method

Many researches attempted to enhance the underlying linear kernel in CNNs. Among these techniques, we focus more on those in which higher order kernels are used instead of the linear one. The intuition behind is to make the underlying linear kernel operates on higher dimensional feature map so that it becomes more discriminative. In other words, instead of running a linear classifier directly on feature, they are first mapped to a higher-dimensional Reproducing Kernel Hilbert Space (RKHS) using a positive definite kernel function. For certain kernel functions, the RKHS can even be infinite dimensional. A linear classifier is then run on this high-dimensional RKHS. Since the dimensionality of the feature vectors is dramatically increased via this mapping, a linear classifier in the RKHS corresponds to a powerful nonlinear classifier in the original feature vector space. Such a classifier is capable of learning more complex patterns than a linear classifier directly operating on the feature vectors. According to [Cui u. a., 2017], there are two ways in general to map features to a higher order RKHS. The first and most commonly used one is to implicitly map the feature via the kernel trick, like in the case of kernel SVM [Schölkopf u. a., 2002, Cortes und Vapnik, 1995]. Thanks to the kernel trick, we never have to explicitly calculate the high-dimensional vectors in the RKHS, which will be computationally expensive (or even impossible in the case of an infinite-dimensional space) to compute and store. Let $\varphi(.) : \mathbf{R}^n \longmapsto \mathbb{H}$ represent this RKHS embedding.

$$K(x, W)_{\mathbb{H}} = \langle \varphi(x), \varphi(W) \rangle_{\mathbb{H}}, \tag{6.2}$$

where $\langle ., . \rangle_{\mathbb{H}}$ denotes the inner product in the Hilbert space $\mathbb{H}$.

The key difference between equation 6.1 and equation 6.2 is that the dot products between x and W have been replaced with the inner products between $\varphi(x)$ and $\varphi(W)$. The more general notion of inner product is used instead of the dot product because the Hilbert space $\mathbb{H}$ can be infinite dimensional.

according to [Cui u. a., 2017, Jayasumana u. a., 2020], the disadvantages are twofold. The storage needed and the evaluation time are both proportional to the number of training data, which makes it inefficient on large datasets. In addition, the construction of the kernel makes it hard to use stochastic learning methods, including Stochastic Gradient Descent (SGD) in the training of CNNs. The second way is to explicitly map the feature vector into high dimensional space with products of features (monomials). The drawback of this method is the dimension of the explicit feature map, which makes it impractical to use in real world applications. Despite of these disadvantages, higher order kernel functions proved that they are more susceptible to fit slight details in data.

We leverage this kernel function properties, and propose to improve the linear kernel function of both convolution and fully connected layers. This enhancement is done through mapping feature with both ways, implicitly and explicitly. In the first step, we use higher order kernel methods instead of the linear subsequent functions in these levels of the CNN. Here we implicitly compute the feature mapping using the kernel trick, as shown in Eq 6.2. After that, we can achieve the explicit mapping of the input features by concatenating the resulting feature maps from the previous step, in addition to the linear feature map. The combination is also considered as feature mapping, since the addition of these kernels is also a valid kernel function. In the following sections, we describe our proposed method for expanding convolution layers and fully connected layers into higher degree kernel layers. These proposed layers can be used in the same manner as the usual CNN layers. Furthermore, these novel layers can be used solely or jointly with the usual CNN layers. This flexibility makes them usable in any architecture or even plugged at any level of a pre-trained CNN model.

6.2.1 Convolution layer expansion

Convolution layers are the core building block of a CNN. They leverage the fact that an input image is composed of small details, or features, and create a mechanism for analyzing each feature in isolation, which makes a decision about the image as a whole. This mechanism allowed CNNs to achieve very good results in various fields. Convolution layer is ruled by a linear kernel function as shown in Eq. 6.1. This makes convolution implementation relatively simple and computationally inexpensive. Yet, in some cases convolution fails to learn fully linearly separable features [Jayasumana u. a., 2020, Mahmoudi u. a., 2021a]. As described above, we leverage kernel function to expand convolution operation to a higher degree kernel function. This will be achieved in both explicit and implicit ways.

First of all, to implicitly expand the convolution function we use multiple polynomial kernels (Eq. 6.3) that will map the input features to a higher dimensional RKHS as follows:

$$K_{Polynomial}(x, W) = \langle \varphi(x), \varphi(W) \rangle = \sum_{i=2}^{p} (x^T W)^p, \tag{6.3}$$

where $\varphi(.) : \mathbf{R}^n \longmapsto \mathbf{R}^d (d \gg n)$ is a non-linear mapping function.

This enables us to extract features in a high dimensional space, while its computational complexity is also much higher than Eq. 6.1. Fortunately, we are able to bypass the explicit calculation of the high dimensional features $\varphi(x)$ via the kernel trick. We use multiple polynomial kernels with different degrees $p \geq 2$. Each of these kernels will output a feature map. These feature maps must have the same height and width, but not necessarily the same depth (number of output channels), in order to be used in the next step.

The second step consists of mapping explicitly the input feature maps. For this purpose, we use special kernel function called Taylor series kernel. Eq. 6.4 describes a Taylor series kernel of order p as follows:

$$K_{Taylor}(x, W) = \langle \varphi(x), \varphi(W) \rangle = \sum_{i=0}^{p} \langle x, W \rangle^{p}, \tag{6.4}$$

where for $p = 1$, K_{Taylor} is equivalent to a convolution kernel.

Whereas, starting from $p \geq 2$, K_{Taylor} is equivalent to a polynomial kernel (Eq. 6.3). To obtain a kernel in the form of Taylor series kernel, we compute simultaneously the convolution operation in addition to higher degree polynomial kernels (from the previous step). The result of all kernel functions is concatenated over the channel axis. This is similarly to an inception module [Szegedy u. a., 2015], except that instead of using multiple convolution of different kernel size, we use multiple kernel functions. Fig. 6.1 shows the processing of our expansion method.

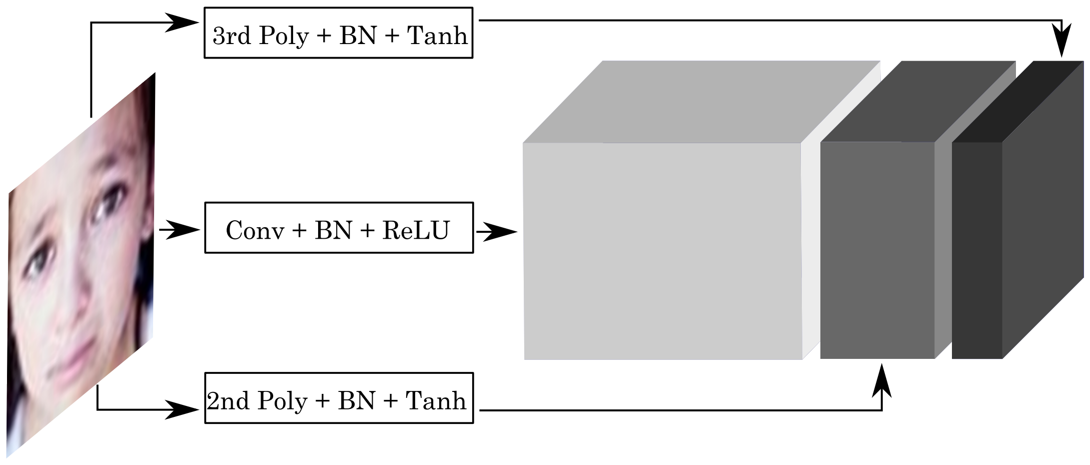


Figure 6.1 – *The proposed expansion method consists of applying higher degree (≥ 2) polynomial kernels to the input, in addition to convolution. The result of these kernels is concatenated over the channel axis.*

6.2.2 Dense layer expansion

Fully connected layers are an essential component of CNNs. These layers take as input the feature maps resulting from the successive convolution and pooling layers (commonly referred to as the feature vector f_v of the input image) in order to drive the final classification decision. Similarly to convolution layers, fully connected layers are driven by a linear kernel function (Eq 6.1). As described above, this function, despite of being simple and computationally inexpensive, fails to learn fully linearly separable features [Jayasumana u. a., 2020]. Therefore, we propose to expand its linear kernel to a higher degree kernel function as well. This also will be achieved in both explicit and implicit ways.

First of all, to expand the dense layers, we use the method of Kernelized Dense Layer (KDL), proposed in [Mahmoudi u. a., 2020]. KDL is similar to a classical neuron layer in the way that it applies a dot product between a vector of weights and an input vector, adds a bias vector ($b \geq 0$) and eventually applies an activation function. The difference from standard fully connected layers is that this method applies a higher degree kernel function instead of a simple linear dot product, which allows the model to map the input data to a higher space and thus be more discriminative than a classical linear layer.

Formally, the output Y is computed by applying a kernel function K on an input vector $x \in \mathbf{R}^n$ and the corresponding vector of weights $W \in \mathbf{R}^n$ and, adding the bias vector ($b \geq 0$). Here, we also use multiple polynomial kernels (Eq. 6.3) that will map the input features to a higher dimensional RKHS. Yet, in contrary to convolution expansion, the output has no size constraints in order to be used in the next step.

The second step consists of mapping explicitly the input feature maps. As for convolution expansion we compute simultaneously the linear dot product, which corresponds to the ordinary fully connected layers, in addition to layers of higher degree polynomial kernels. The result of these computations is concatenated to form a single vector. This vector is then in the form of Taylor series kernel. This resulted vector will constitute the input for the next fully connected layer. An illustration of this process is shown in Fig. 6.3.

As illustrated in Fig. 6.3, we also propose to combine the two expansion methods described above. This combination will result in a fully expanded model in the form of a Taylor series kernel.

6.3 Experiments

In this section, we shed light on the experimental settings, that have been used to evaluate our expansion method. First, we briefly describe the FER datasets that have been used for the experiments. Then we detail the architecture of the the models we have used and their training process. Recall from previous section that our expansion method operates mainly on two type of layers which are convolution and dense layers. Therefore, we study the impact of each expansion method, solely and jointly as shown in figure 6.2, on the network accuracy. This results are compared to the ordinary convolution and dense layers. Finally, we discuss the obtained results with regard to state-of-the-art results.

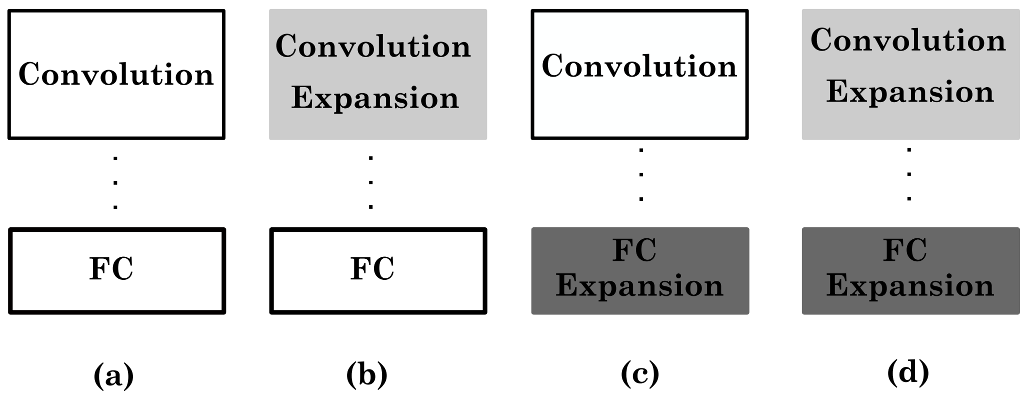


Figure 6.2 – *In our experiments, we compare the results to an ordinary CNN (a) with our proposed method over three different configurations. In the first case (b), we replace convolution by the convolution expansion allover the network followed by fully connected layers. In the second case (c), we replace only the fully connected layers with our expansion method for these layers. Finally, we test our expansion methods allover the network (d).*

6.3.1 Models architecture,training process and datasets

The assumption made in this article is that linear kernel, used at different levels on a CNN, can achieve better performance if they are expanded using our proposed method. Moreover, this improvement is reached without the need of additional weight parameters. To verify this assumption we compare the performance of an ordinary CNN to a network with the same architecture, yet using our proposed expansion method. For this purpose, we used two pre-trained networks namely, VGG-16 and VGG-19 in addition to a CNN built from scratch. For the pre-trained CNNs we took only the convolution part and added two dense layers of 256 unit each and a final softmax layer with 7 output units. This models will be referred to as VGG-16-bese and VGG-19-bese. Whereas, we will refer to the model built from scratch as Model-1. This model architecture is composed of five convolutional blocks. Each block consists of a convolution, batch normalization and rectified linear unit activation layers. The use of batch normalization [Zou u. a., 2019] before the activation brings more stability to parameter initialization and achieves higher learning rate. Each of the five convolutional blocks is followed by a max pooling layer and a dropout layer. Finally, three fully connected layers are added on top of the last convolution block with respectively 128, 128 and 7 units. Furthermore, we build expanded models, with the same architecture as the three models described above, following the configuration shown in figure 6.2.

For training, We used Adam optimiser with a learning rate varying from 0.001 to 5e-5. This learning rate is decreased by a factor of 0.5 if the validation accuracy does not increase over five epochs. To avoid overfitting, we first augmented the data using a range degree for random rotations of 20, a shear intensity of 0.2, a range for random zoom of 0.2, and randomly flip the inputs horizontally. We also employed early stopping if the validation accuracy does not improve by a factor of 0.01 over ten epochs. Each layer of

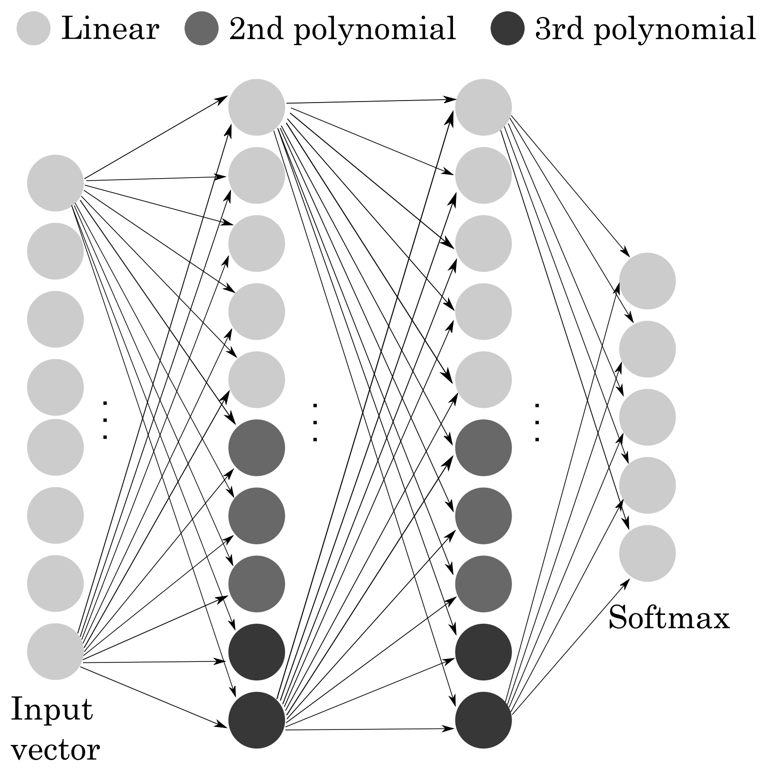


Figure 6.3 – *Expansion of dense layer proceeds in two steps. First, it computes multiple polynomial KDL in addition to the linear layer. Then, it concatenates the resulting vectors in a single expanded dense layer vector. The latter will be finally fed to the next expanded dense layer or dense layer. Best viewed in color.*

our model is initialized with *He* normal distribution [He u. a., 2015a] and a weight decay of 0.0001. For our experiments, we have used three well-known in the wild FER datasets, namely RAF-DB, ExpW and FER2013. The only preprocessing which we employed on all experiments is cropping the face region and resizing the resulting images to 100×100 pixels.

6.3.2 Ablation Study

This section explores the impact of the use of the proposed expansion method on the overall accuracy of VGG-16, VGG-19 and Model-1. The obtained results using these models are reported in Tables 6.1, 6.2 and 6.4, as base models. As shown in Tables 6.1, 6.2, and 6.4 the accuracy rates obtained with VGG-16-base is 69.38%, 85.42%, 77.75% on FE2013, RAF-DB and ExpW, respectively. Whereas for VGG-19-base, the obtained accuracy rates are 69.52% for FER2013, 85.99% for RAF-DB, and 77.92% for ExpW. Lastly, the obtained accuracy rates for Model-1 are 70.13%, 87.05%, and 75.91% for FER2013, RAF-DB and ExpW, respectively.

After that, we evaluated the performance of these network architectures, with our proposed method, on different levels and with different kernel function degrees (Fig. 6.2). These expanded models of VGG-16, VGG-19 and Model-1, were, as mentioned above, built and trained from scratch. As shown in figure 6.2, we performed CNN layers expansion following three main configurations. First, we used the expansion on the convolution level. After that, we expanded the fully connected layers. Finally, we tested full expansion by combining the two previous expansion methods. For all of these expansions, we use three different kernels, namely: i) a Taylor series kernel up to the second degree; ii) a Taylor series kernel up to the third degree; and iii) the linear kernel which corresponds to a Taylor series kernel up to degree one. We have expanded the convolution layer by first adding two blocks of feature maps resulting from two polynomial kernels of second and third degree. The proportion of the kernels are approximately 70% for convolution, 20% for polynomial kernel of second degree, and 10% for polynomial kernel of third degree. For instance, if the output channel size is 128, the proportion of the three kernels output is 90 for convolution, 26 for polynomial kernel of second degree and 12 for polynomial kernel of third degree. In the case of expansion up to second degree only, the proportion are approximately 80% for convolution and 20% for polynomial kernel of second degree. Following the previous example, the proportion of the two kernels output is 102 for convolution and 26 for polynomial kernel of second degree. We also used the same proportion in the dense layers expansion and the full network expansion. The sections bellow discuss the obtained results at each level solely, then the results of their combinations.

Convolution layer expansion

As shown in Table 6.1, the accuracy rates obtained with VGG-16 like model are 70.07% for FER2013, 86.13% for RAF-DB, and 78.17% for ExpW, with the Taylor kernel up to second degree. This represents an improvement up to 0.7% over the full linear model. Whereas, with the Taylor kernel up to third degree, the obtained accuracy rates are 70.31% for FER2013, 86.24% for RAF-DB, and 78.61% for ExpW. This kernel improves further the accuracy of the model up to 1% more than its full linear counterpart. On the other hand, the accuracy rates obtained with VGG-19 like model are 70.82% for FER2013, 86.44% for RAF-DB, and 78.33% for ExpW, with the Taylor kernel up to second degree. Similarly to VGG-16, the use of this kernel also enhances the accuracy for VGG-19 up to 0.8% more than the linear kernel model. Whereas, with the Taylor kernel up to third degree, the obtained accuracy rates are 71.08% for FER2013, 87.04% for RAF-DB, and 79.16% for ExpW. Again, this kernel improves the accuracy up to 1.5% compared to its linear counterpart. Finally, the accuracy rates obtained with Model-1 are 70.84% for FER2013, 87.69% for RAF-DB, and 76.52% for ExpW, with the Taylor kernel up to second degree. Similarly to the previous models, the use of this kernel also enhances the accuracy for Model-1 up to 0.6% more than the linear kernel model. Whereas, with the Taylor kernel up to third degree, the obtained accuracy rates are 71% for FER2013, 87.84% for RAF-DB, and 79.71% for ExpW. Therefore, this kernel improves the accuracy up to 0.8% compared to its linear counterpart.

Table 6.1 – *Results of convolution layer expansion method.*

Level	Model	Kernel	FER2013	RAF	ExpW
VGG-16-base			69.38%	85.42%	77.75%
VGG-19-base			69.52%	85.99%	77.92%
Model-1			70.13%	87.05%	75.91%
Conv-Expansion	VGG-16	Conv-2nd	70.07%	86.13%	78.17%
		Conv-3rd	**70.31**%	**86.24**%	**78.61**%
	VGG-19	Conv-2nd	70.82%	86.44%	78.33%
		Conv-3rd	**71.08**%	**86.57**%	**78.71**%
	Model-1	Conv-2nd	70.84%	87.69%	76.52 %
		Conv-3rd	71%	87.84%	76.71 %

Dense layer expansion

First of all, as shown in Table 6.2, the accuracy rates obtained with VGG-16 like model are 70.98% for FER2013, 86.89% for RAF-DB, and 78.85% for ExpW, with the Taylor kernel up to second degree. This kernel improves further the accuracy of the model up to 1.3% more than its full linear counterpart. Whereas, with the Taylor kernel up to third degree, the obtained accuracy rates are 71.58% for FER2013, 87.04% for RAF-DB, and 79.16% for ExpW. This represents an improvement up to 2% over the full linear model. On the other hand, the accuracy rates obtained with VGG-19 like model are 71.46% for FER2013, 87.18% for RAF-DB, and 79.21% for ExpW, with the Taylor kernel up to second degree. Similarly to VGG-16, the use of this kernel also enhances the accuracy for VGG-19 up to 1.9% more than the linear kernel model. Whereas, with the Taylor kernel up to third degree, the obtained accuracy rates are 71.95% for FER2013, 87.29% for RAF-DB, and 79.39% for ExpW. This represents an improvement up to 2.5% over its linear counterpart. Finally, the accuracy rates obtained with Model-1 are 71.41% for FER2013, 88.26% for RAF-DB, and 76.11% for ExpW, with the Taylor kernel up to second degree. This represents an improvement up to 1.3% over its linear counterpart. With the Taylor kernel up to third degree, Model-1 reached 71.86%, 88.59% and 76.63% on FE2013, RAF-DB and ExpW, respectively. This represents an improvement up to 1.8% over its linear counterpart.

Table 6.2 – *Results of dense layer method.*

Level	Model	Kernel	FER2013	RAF	ExpW
VGG-16-base			69.38%	85.42%	77.75%
VGG-19-base			69.52%	85.99%	77.92%
Model-1			70.13%	87.05%	75.91%
Dense-Expansion	VGG-16	Dense-2nd	70.98%	86.89%	78.85%
		Dense-3rd	**71.58%**	**87.04%**	**79.16%**
	VGG-19	Dense-2nd	71.46%	87.18%	79.21%
		Dense-3rd	**71.95%**	**87.29%**	**79.39%**
	Model-1	Conv-2nd	71.41%	88.26%	76.11 %
		Conv-3rd	71.86%	88.59%	76.63%

To further evaluate the efficiency of dense layer expansion, we tested these layers solely in an MLP fashion. The goal, here, is not achieve state-of-the-art or competitive results. It is rather to demonstrate the improvement that an expanded dense layer can bring when used, solely, in an MLP fashion. We built an MLP with two hidden layers of 256 units each and a softmax output layer of 7 units. We followed the same configuration as the previous dense layer expansion, that is: i) a full linear MLP; ii) an MLP with a Taylor series kernel up to the second degree; and ii) an MLP with a Taylor series kernel up to the third degree. Given the small size of these MLP networks, We resized the FER dateset images to 48×48 pixels. The results of these configuration are reported in 6.3. As one can see in table 6.3, the linear MLP achieved 43.18%, 51.29%, 39.65% on FE2013, RAF-DB and ExpW, respectively. Whereas, the MLP with Taylor series expanded dense layers up to the second degree reached 45.32% on FER2013, 51.84% on RAF and 40.11% on ExpW. This is represents an improvement up to 2.12% in accuracy rate compared to its linear counterpart. Finally, the MLP with Taylor series expanded dense layers up to the third degree reached 45.80%, 52.26%, 40.81% on FE2013, RAF-DB and ExpW, respectively. Once again, the MLP with Taylor series expanded dense layers up to the third degree outperformed the other expansion kernels and enhanced the accuracy for about 2.62%. This short experiment show the inherent descriminative power of neuron with a kernel of high degree. In other words, it show the enhancement that such a neuron can bring at any level in any type of neural network.

Table 6.3 – *Results of dense layer as MLP.*

Kernel	FER2013	RAF	ExpW
linear-MLP	43.18%	51.29%	39.65%
Dense-2nd-MLP	45.32%	51.84%	40.11%
Dense-3nd-MLP	45.80%	52.26%	40.81%

Full expansion

After testing expansion on the two main levels of the CNN, namely: convolution layer and fully connected layer, we have also tested the efficiency of this expansion method jointly on these layer types. As shown in Table 6.4, the accuracy rates obtained with VGG-16 like model are 70.22% for FER2013, 86.17% for RAF-DB, and 78.32% for ExpW, with the Taylor kernel up to second degree. This kernel improves further the accuracy of the model up to 0.8% more than its full linear counterpart. Whereas, with the Taylor kernel up to third degree, the obtained accuracy rates are 70.28% for FER2013, 86.20% for RAF-DB, and 78.41% for ExpW. This represents an improvement up to 0.9% over the full linear model. On the other hand, the accuracy rates obtained with VGG-19 like model are 70.91% for FER2013, 86.48% for RAF-DB, and 78.52% for ExpW, with the Taylor kernel up to second degree. Similarly to VGG-16, the use of this kernel also enhances the accuracy for VGG-19 up to 1.5% more than the linear kernel model. Whereas, with the Taylor kernel up to third degree, the obtained accuracy rates are 70.97% for FER2013, 86.53% for RAF-DB, and 78.63% for ExpW. This represents an improvement up to 1.6% over its linear counterpart. Unfortunately, the expansion to the third degree Taylor series kernel does not enhance much the overall accuracy of the network compared with the second degree expansion.

Table 6.4 – *Results of full expansion method.*

Level	Model	Kernel	FER2013	RAF	ExpW
VGG-16-base			69.38%	85.42%	77.75%
VGG-19-base			69.52%	85.99%	77.92%
Model-1			70.13%	87.05%	75.91%
Full-Expansion	VGG-16	Full-2nd	70.22%	86.17%	78.32 %
		Full-3rd	70.28%	86.20%	78.41%
	VGG-19	Full-2nd	70.91%	86.48%	78.52%
		Full-3rd	70.97%	86.53%	78.63%
	Model-1	Conv-2nd	70.95%	87.78%	76.68%
		Conv-3rd	71.12%	87.93%	76.84%

The obtained results with our expansion method show that the use of higher order kernel along with the linear kernel is beneficial to the overall accuracy of the network. However, the impact of our method changes according to the level where it is applied. For instance, the use of our expansion method at the convolution level increases the accuracy according to the kernel degree. The higher is the kernel degree, the better is the accuracy. Similarly to the convolution layers, the fully connected layers also increase the accuracy rates according to the kernel degree. However, the use of our expansion method on the fully connected layers performs better than the convolution expansion. Finally, even though the full network expansion increases the overall accuracy over the linear model, it slightly outperforms the convolution expansion method. Furthermore, the increase of the accuracy with respect to the kernel degree is not truly perceptible.

Also, the use of the full expansion seems to be prone to over-fitting. Taking into consideration the computation cost of a full expansion, the latter seems to be less useful than the precedent uses of our expansion method.

6.3.3 Comparison with state-of-the-art

In this section, we compare the performance of our two models which uses the proposed expansion method with respect to several state-of-the-art methods. The obtained results are reported in Table 6.5. According to Table 6.5, our proposed expansion method outperforms all the state-of-the-art methods on the ExpW dataset. The best accuracy rate is 79.39% and has been reached using the fully connected layers expansion up to the third degree kernel with VGG-19 model. The same kernel with VGG-16 model gives 79.16%. On the other hand, convolution expansion reached an accuracy rate of 78.71% with the third degree kernel on VGG-19, and 78.61% with the same expansion on VGG-16. As stated in the previous section, the use of a full expansion overall the network does not enhance the accuracy as it might be expected.

On RAF dataset, the performance of our method outperforms most of the state-of-the-art methods. The best performance on this dataset were reached with the fully connected layer expansion on both VGG-19 and VGG-16 models. The other expansion techniques performs less than the latter, and reached lower results compared to state-of-the-art methods. The methods we could not outperform are Mahmoudi et al which implemented kernel function as pooling layer and FC layer.

Finally, even though we did not outperform state-of-the-art methods on FER2013, we confirmed the superiority of our method compared to standard CNNs. We reached an accuracy rate of 71.95% with the third order kernel expansion on fully connected layers which is 1.78% less than state-of-the-art method [Kim u. a., 2016].

Table 6.5 – *Accuracy rates of the proposed approach and state-of-the-art approach*

Models	FER2013	RAF	ExpW
VGG-16-Conv-3rd	**70.31%**	**86.24%**	**78.61%**
VGG-19-Conv-3rd	**71.08%**	**86.57%**	**78.71%**
VGG-16-Dense-3rd	**71.58%**	**87.04%**	**79.16%**
VGG-19-Dense-3rd	**71.95%**	**87.29%**	**79.39%**
VGG-16-Full-3rd	70.28%	86.20%	78.41%
VGG-19-Full-3rd	70.97%	86.53%	78.63%
Model-1-Conv-3rd	71%	87.84%	76.71%
Mahmoudi et al. [Mahmoudi u. a., 2020]	**71.35%**	**93.21%**	**76.81%**
Mahmoudi et al [Mahmoudi u. a., 2020]	71.28%	**88.02%**	**76.64%**
Tang et al. [Tang, 2013]	71.16%	–	–
Guo et al. [Guo u. a., 2016]	71.33%	–	–
Kim et al. [Kim u. a., 2016]	**73.73%**	–	–
Bishay et al. [Bishay u. a., 2019]	–	–	**73.1%**
Lian et al. [Lian u. a., 2020]	–	–	71.9 %
Acharya et al. [Acharya u. a., 2018a]	–	**87%**	–
S Li et al. [Li und Deng, 2018b]	–	74.2%	–
Z.Liu et al. [Liu u. a., 2017b]	–	73.19%	–

6.4 Conclusion

In this chapter, we proposed to improve CNN performance without increasing the number of learnable parameters. Our method consists of expanding the linear kernel function, used at different levels of a CNN. The expansion is performed by combining multiple polynomial kernels with different degrees. By doing so, the network automatically learns the suitable kernel that optimizes the target objective. In our settings, a network can either use a single kernel or a combination of multiple ones which make a Taylor series kernel. We demonstrated that the used kernel function is more sensitive to subtle details than the linear one. This is important for fine-grained classification in particular it increases both the representation and the classification power of the CNN for facial expression recognition. The experiments conducted on FER datasets showed that the use of our method allows the network to outperform conventional CNNs. The obtained results showed that the use of higher order kernel along with the linear one is beneficial to the overall accuracy of the network. However, we noticed that the position of the plugged kernel impacts on the accuracy of the full network. For instance, the kernel expansion at the convolutional level increases the accuracy according to the kernel degree. We also observed that, the fully connected layers react similarly as the convolutional layers, to the kernel degree. However, the kernel expansion used on the fully connected layers performs better than any convolutional layer in the network. We finally observed that, even though the full network expansion increases the overall accuracy over the linear model, it slightly outperforms the convolution. Furthermore, in the full network expansion, the increase of the accuracy with respect to the kernel degree is not truly perceptible. Also, the use of the full expansion seems to be prone to over-fitting. Taking into consideration the computation cost of a full expansion, one may prefer to use either the convolutional layer expansion or the fully connected layer expansion taken separately.

As perspectives, we aim at developing an algorithm that allows to learn the appropriate kernel for a specific task. The algorithm favors a kernel, among a variety of kernels, that optimizes the objective function. Moreover, the use of this algorithm may results in a combination of different kernels at different levels of a network.

Conclusion and Future Work

Our studies focused on the incorporation of non-linear kernel functions at different levels of a CNN. The assumption that we have made is that linear functions are efficient only when the original data is linearly separable, which should have, in general, a high dimensional representation. In such a case, the decision boundary can be representable as a linear combination of the original features. On the other case, linear kernel functions fail to fit the input data. Therefore, a more discriminative kernel functions must be used. The intuition behind is to make the underlying linear kernel operates on higher dimensional feature map so that it becomes more discriminative. In other words, instead of running a linear classifier directly on feature, they are first mapped to a higher-dimensional Reproducing Kernel Hilbert Space (RKHS) using a positive definite kernel function. For certain kernel functions, the RKHS can even be infinite dimensional. A linear classifier is then run on this high-dimensional RKHS. Since the dimensionality of the feature vectors is dramatically increased via this mapping, a linear classifier in the RKHS corresponds to a powerful nonlinear classifier in the original feature vector space. Such a classifier is capable of learning more complex patterns than a linear classifier directly operating on the feature vectors. To evaluate the performance of the resulting methods, we chose to apply the later on more challenging fine-grained recognition. Facial expression recognition is considered one of most challenging fine-grained recognition problems. Indeed, the difference in facial expression categories relies on small subtle areas in the facial images like the mouth, eyebrows and the noise. To overcome this issue, facial expression recognition systems must be able to recognise this subtle differences efficiently.

Chapter 1 sheds lights on the important concept related to our field of study. First of all, We detail this new emergent domain in artificial intelligence called Deep Learning, review the most important models, framework and its different application fields. Second, we give a brief overview about kernel methods. This overview covers the mathematical basics that the kernel methods rely on. It also explains the construction steps of kernels, the important concept of the kernel trick, and enumerate some well known kernel functions. Since our work is centred around the incorporation of kernel methods in deep learning networks. Among the deep learning models, our research focused specifically on convolutional neural networks (CNN). Therefore, the later is more detailed than other models. Finally, our case of study: facial expression recognition is defined and some important study in this field were also illustrated.

Chapter 2 proposed a FER method based on the improved bilinear CNN model.

In this framework, various ways of normalization were used to improve the accuracy, including the matrix square root, element-wise square root and L2 normalization. To validate our method, we have used three large, well known, facial expression databases which are FER2013, RAF-DB and ExpW. In order to evaluate the improvement of our method, we have first implemented a CNN from scratch and fine-tuned pre-trained VGG-16 on our facial expressions datasets. After that we have implemented a bilinear model on top of the above models individually and on top of both of them. Finally, we repeated the same procedure with the improved bilinear model. The experiments show that this framework improves the overall accuracy for about 3%.

Bilinear models have been shown to achieve very good accuracy results on different visual recognition domains, like fine grained recognition, semantic segmentation and face recognition. Nevertheless, the dimensions of bilinear features are very high, usually on the order of hundreds of thousands to a few million. The reason why they are not practical for many visual recognition fields. Moreover, matrix square root function and bilinear pooling function are very memory and CPU consuming, which decrease the performance of the model. Therefore, many improvements have been applied to CNN, for instance compact bilinear pooling [Gao u. a., 2016], reaching the same discriminative power as the full bilinear representation but with a representations having only a few thousand dimensions. An other improvement is the kernel pooling for CNNs [Cui u. a., 2017] which is a general pooling framework that captures higher order interactions of features in the form of kernels.

Chapter 3 introduce one of our major contribution. we proposed a FER method based on a CNN model to which we specifically designed a novel pooling layer which retains the down-sampling advantage of an ordinary pooling function and brings several new features. The proposed pooling layer, which has learnable weights, generalizes standard pooling functions and, additionally encodes non-linear relation between features. It is differentiable and can be plugged at any level of the network, allowing, in turns, an end-to-end learning. The experiments on ExpW, RAF-DB and FER2013 datasets demonstrate the efficiency of the proposed pooling method compared to standard pooling. The experiments also showed that the proposed FER method outperforms state-of-the-art methods. The performance of our model is essentially due to its capability of capturing high order information that are crucial for fine-grained classification tasks such as the FER.

Chapter 4 also introduce an innovative work that is similar to the precedent chapter, yet focuses on the fully connected layer of Convolutional Neural Networks. we designed Kernelized Dense Layer for CNN model that aims to enhance the discriminative power of the overall model. It consists of applying higher order kernel method than the standard FC layer. Experimental results on ExpW, RAF-DB and FER2013 datasets demonstrate the efficiency of the proposed KDL compared to standard FC layer in terms of convergence, speed and overall accuracy. The proposed FER method outperforms most of the state-of-the-art methods and remains competitive. The performance of our model is essentially due to its capability of capturing high order information that are crucial for fine-grained classification tasks such as the FER.

Chapter 5 investigates the usage of kernel functions at the different layers in a convolutional neural network. We carry out extensive studies of their impact on convolutional, pooling and fully-connected layers. We notice that the linear kernel may not be sufficiently effective to fit the input data distributions, whereas high order kernels prone to over-fitting. This leads to conclude that a trade-off between complexity and performance should be reached. We show how one can effectively leverage kernel functions, by using our proposed pooling layers (chapter 3) and the proposed Kernelized Dense Layers (chapter 4). The experiments on conventional classification datasets i.e. MNIST, FASHION-MNIST and CIFAR-10, show that the proposed techniques improve the performance of the network compared to classical convolution, pooling and fully connected layers. Moreover, experiments on fine-grained classification i.e. facial expression databases, namely RAF-DB, FER2013 and ExpW demonstrate that the discriminative power of the network is boosted, since the proposed techniques improve the awareness to slight visual details and allows the network reaching state-of-the-art results.

In light of the results obtained in the experiments, we have conducted in this work, we deduce that kernel functions impact positively the CNN performance. Indeed, the experimental results prove that the use of kernel functions, instead of the linear functions used in CNN layers, improves the accuracy rate of the whole model and its convergence speed. Furthermore, we concluded that the impact of these kernel functions varies according to the level in which they are plugged into and the specific kernel in use. We have also concluded that high order kernels prone to overfitting, whereas low order kernels might not be sufficiently effective to fit the input data distributions. Therefore, a trade-off between complexity and performance should be reached.

Chapter 6 we proposed to improve CNN performance without increasing the number of learnable parameters. Our method consists of expanding the linear kernel function, used at different levels of a CNN. The expansion is performed by combining multiple polynomial kernels with different degrees. By doing so, the network automatically learns the suitable kernel that optimizes the target objective. In our settings, a network can either use a single kernel or a combination of multiple ones which make a Taylor series kernel. We demonstrated that the used kernel function is more sensitive to subtle details than the linear one. This is important for fine-grained classification in particular it increases both the representation and the classification power of the CNN for facial expression recognition. The experiments conducted on FER datasets showed that the use of our method allows the network to outperform conventional CNNs. The obtained results showed that the use of higher order kernel along with the linear one is beneficial to the overall accuracy of the network. However, we noticed that the position of the plugged kernel impacts on the accuracy of the full network. For instance, the kernel expansion at the convolutional level increases the accuracy according to the kernel degree. We also observed that, the fully connected layers react similarly as the convolutional layers, to the kernel degree. However, the kernel expansion used on the fully connected layers performs better than any convolutional layer in the network. We finally observed that, even though the full network expansion increases the overall accuracy over the linear model, it slightly outperforms the convolution. Furthermore, in the full network expansion, the increase of the accuracy with respect to the kernel degree is not truly perceptible. Also, the use of the full expansion seems to be prone to over-fitting. Taking into consideration the computation cost of a full expansion, one may prefer to use either the convolutional layer expansion or the fully connected layer expansion taken separately.

As perspectives, we aim at developing an algorithm that allows to learn the appropriate kernel for a specific task. The algorithm favors a kernel, among a variety of kernels, that optimizes the objective function. Moreover, the use of this algorithm may results in a combination of different kernels at different levels of a network.

Our contributions

- M Amine Mahmoudi, Aladine Chetouani, Fatma Boufera, and Hedi Tabia. " Learnable pooling weights for facial expression recognition". Pattern Recognition Letters, 138, 2020. [Mahmoudi u. a., 2020]
- M. A. Mahmoudi, A. Chetouani, F. Boufera, and H. Tabia. "Kernelized dense layers for facial expression recognition". 2020 IEEE International Conference on Image Processing (ICIP), pages 2226–2230, 2020. [Mahmoudi u. a., 2020]
- M Amine Mahmoudi, Aladine Chetouani, Fatma Boufera, and Hedi Tabia. "Improved bilinear model for facial expression recognition". Pattern Recognition and Artificial Intelligence. MedPRAI2020. Communications in Computer and Information Science, volume 1322, pages 47–59. Springer, 2021a. [Mahmoudi u. a., 2021b]
- M Amine Mahmoudi, Aladine Chetouani, Fatma Boufera, and Hedi Tabia. "Taylor series kernelized layer for fine-grained recognition". 2021 IEEE International Conference on Image Processing (ICIP), pages 1914–1918. IEEE, 2021b. [Mahmoudi u. a., 2021c]
- M Mahmoudi, Aladine Chetouani, Fatma Boufera, and Hedi Tabia. "Deep kernelized network for fine-grained recognition". International Conference on Neural Information Processing, pages100–111. Springer, 2021 [Mahmoudi u. a., 2021a]
- M Amine Mahmoudi, Aladine Chetouani, Fatma Boufera, and Hedi Tabia. "Kernel-based convolution expansion for facial expression recognition". Pattern Recognition Letters, Volume 160, August 2022, Pages 128-134. [Mahmoudi u. a., 2022]

Contributions - Under Review

- M Amine Mahmoudi, Aladine Chetouani, Fatma Boufera, and Hedi Tabia. "Kernel Function Impact on Convolutional Neural Networks". In Multimedia Tools and Applications – Springer, since August 2021.
- M Amine Mahmoudi, Aladine Chetouani, Fatma Boufera, and Hedi Tabia. "Expanding Convolutional Neural Network Kernel For Facial Expression Recognition". In IEEE Transactions on Cognitive and Developmental Systems, since September 2021.

A Deep kernelized network for fine-grained recognition

In this chapter, we investigate the usage of different kernel methods in a CNN fashion. The goal here is to use the structure of a CNN, but instead of using simple linear kernel function which are usually in CNN layers, we use Higher degree kernel function. These kernel function are more discriminative than linear function, therefore they can bring more discrimination power to the network. However, the impact of these function can vary from a layer to another. For this purpose, we first replace the convolution operation in CNNs by a non-linear kernel function similarly to Kervolution [Wang u. a., 2019]. We also replace fully connected layer alternatively with Kernelized Dense Layers (KDL) [Mahmoudi u. a., 2020] and kernel SVM [Burges u. a., 1999]. The remainder of this chapter is organized as follows: Section A.1 introduces the study design. Section A.2 presents our experiments setting, the datasets we used and their related results. Finally, Section A.3 concludes the paper.

A.1 Study design

In this section, we describe the study design that we have followed in order to investigate the impact of kernel function methods when used in a deep network fashion. Figure A.1 shows the different deep kernel networks configuration used for this study. In the first configuration (a), we used a Kervolution based network followed by fully connected layers. This first configuration is used only for training the Kervolution based network that will be used in later configurations. In the second configuration (b), we used the same Kervolution based network followed by Kernelized dense layers (KDL). Finally, for the third configuration, we took also the Kervolution based network and plugged a kernel SVM at its end. The two last configurations represent two fully kernelized deep networks that our study will be based on.

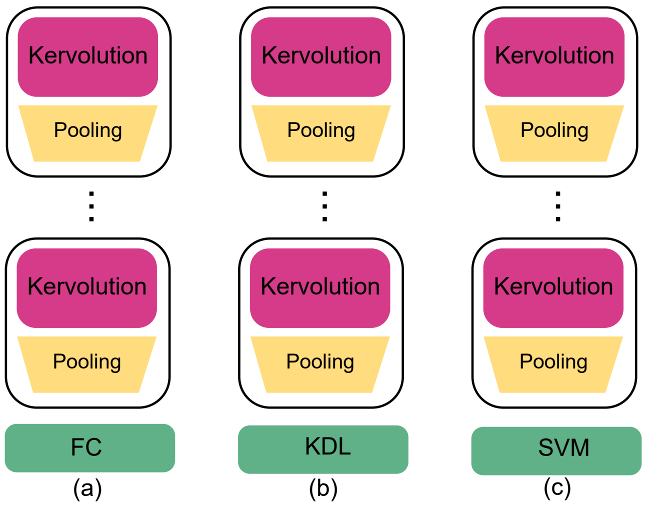


Figure A.1 – *The deep kernel networks configuration used for this study.*

A.2 Experiments

In this section, we explain in more details the experiments we have performed in order to evaluate our methods described above. For this purpose, we have used three well-known fine-grained as well as three well-known FER datasets. Details of these datasets are given below. After that, implementation details are given including the models architectures and the training process. Finally, we discuss the obtained results are discuss the improvement given by each technique.

A.2.1 Datasets

We evaluated our models on two categories of datasets. First of all, we used three well-known fine-grained datasets, namely, FGVC-Aircraft [Maji u. a., 2013], Stanford-Cars [Krause u. a., 2013b] and CVPRIndoor [Quattoni und Torralba, 2009].

- The **FGVC-Aircraft [Maji u. a., 2013]** dataset is composed of 10,200 images of aircraft images. These images are categorized in 102 different aircraft model variants, most of which are airplanes.

- The **StanfordCars [Krause u. a., 2013b]** dataset is composed of 16,185 images of 196 categories of cars. These images are divided into 8,144 for training and 8,041 for testing.

- The **CVPRIndoor [Quattoni und Torralba, 2009]** dataset is composed of 15620 images categorized in 67 Indoor classes with at least 100 images per category.

In addition, we also tested our approach on three FER datasets, namely, RAF-DB, ExpW and FER2013.

- The **RAF-DB** [Li u. a., 2017] or Real-world Affective Face DataBase is composed of 29,672 facial in the wild images. This images are categorized in either seven basic classes or eleven compound classes.
- The **ExpW** [Zhang u. a., 2018b] or EXPression in-the-Wild dataset is composed of 91,793 facial in the wild images. The annotation was done manually on individual images.
- The **FER2013** database was used for the ICML 2013 Challenges in Representation Learning [Goodfellow u. a., 2013]. It is composed of 28,709 training images and 3,589 images for both validation and test.

The intuition behind that was to prove that kernel based models are efficient for recognizing object with subtle visual details. It is worth noting that both categories of the datasets used in this chapter meet these criteria.

A.2.2 Models architecture and training process

To demonstrate the efficiency of the proposed method, we used VGG-16 [Simonyan und Zisserman, 2014] like models with two fully connected layers of 256 units each and a final softmax layer. In addition, we built a second model from scratch (Fig A.2). This model architecture is composed of five Kervolutional blocks, each followed by a max pooling layer and a dropout layer. A block consists of a Kervolutional layer, batch normalization layers. We will refer to these models as VGG-16-base and Model-1, respectively. For training, we have used Adam optimiser using a learning rate starting from 0.001 to 5e-5. This learning rate is lowered by a factor of 0.5 if the validation accuracy does not improve for five epochs. To avoid overfitting, we first augmented the data using a shear intensity of 0.2, a range degree for random rotations of 20, and randomly flip the inputs horizontally, a range for random zoom of 0.2,. We also employed early stopping if the validation accuracy does not increase by a factor of 0.01 over ten epochs. Each layer of our model is initialized with *He* normal distribution [He u. a., 2015a] and a weight decay of 0.0001. The only preprocessing we employed on all our experiments is cropping the face region and resizing the resulting images to 100×100 pixels for FER dataset. For both FGVC-Aircraft and StanfordCars, we cropped the object region. We resize the resulting images so that its longer side is 100 while keeping its aspect ratio. We have also zero padded these images to obtain 100×100 pixels.

We first trained our two base kervolutional models, VGG-16-base and Model-1 with fully connected layers. These models use both second and third order polynomial kernels. After that, we took the Kervolution backbones and replaced the fully connected layers with KDL and SVM with the same kernel used in the later (Fig A.1). These resulted in six different configurations for each model.

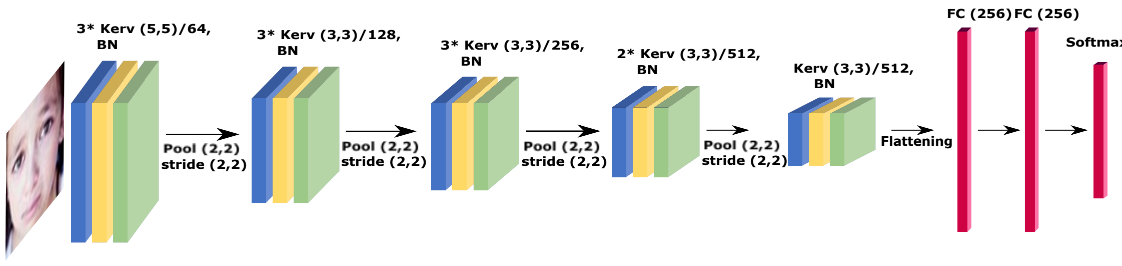


Figure A.2 – *Model-1 architecture: it is composed of five Kervolutional blocks. Each block consists of a Kervolutional layer, batch normalization layers. Each block is followed by a max pooling layer and a dropout layer. Finally, two fully-connected layers are added on top of these convolution blocks with respectively 256 units and ReLU activation and an output softmax layer.*

A.2.3 Ablation Study

This section explores the impact of kernel-based methods on the overall network accuracy, following the network configurations explained above. The results obtained with these models are reported as VGG-16-base and Model-1 in Table A.1 and Table A.2 for fine-grained and FER datasets, respectively.

Table A.1 – *Results of the different configurations on fine-grained datasets.*

Model	Kernel		FGVC-Aircraft	StanfordCars	IndoorCVPR
VGG-16	FC	2nd order poly	65.42%	62.15%	64.23%
		3rd order poly	65.87%	62.64%	64.92%
	KDL	2nd order poly	66.11%	62.95%	65.06%
		3rd order poly	66.72%	63.36%	65.7%
	SVM	2nd order poly	66.4%	63.31%	65.52%
		3rd order poly	67.08%	63.96%	66.01%
Model-1	FC	2nd order poly	65.88%	62.74%	64.79%
		3rd order poly	66.45%	63.09%	65.32%
	KDL	2nd order poly	65.27%	62.19%	64.08%
		3rd order poly	66.8%	63.68%	65.88%
	SVM	2nd order poly	65.59%	62.79%	64.6%
		3rd order poly	67.03%	63.36%	64.24%

As shown in Table A.1, the accuracy rates obtained with VGG-16-base using second order Kervolution and fully connected layers are 65.42%, 62.15%, 64.23 % on FGVC-Aircraft, StanfordCars and IndoorCVPR, respectively. Whereas for third order Kervolution VGG-16-base and fully connected layers, the obtained accuracy rates on FGVC-Aircraft, StanfordCars and IndoorCVPR, respectively, are 65.87%, 62.64%, 64.92%. On the other hand, the accuracy rates obtained with VGG-16 like model are 66.11% on FGVC-Aircraft, 62.95% on StanfordCars and 65.06% on IndoorCVPR, with second order Kervolution and KDL. This represents an improvement up to 0.8% over the fully connected layers. Whereas, with Kervolution and KDL third degree polynomial, the obtained accuracy rates are 66.72% on FGVC-Aircraft, 63.36% on StanfordCars, 65.7% on IndoorCVPR. This kernel improves further the accuracy of the model up to 0.9% more than its FC counterpart. Lastly, the accuracy rates obtained with VGG-16 like model are 66.4% on FGVC-Aircraft, 63.31% on StanfordCars, 65.52% on IndoorCVPR, with second order Kervolution and SVM. This represents an improvement up to 1.3% over the fully connected layers. Whereas, with Kervolution and SVM third degree polynomial, the obtained accuracy rates are 67.08% on FGVC-Aircraft, 63.96% on StanfordCars, 66.01% on IndoorCVPR. This kernel improves further the accuracy of the model up to 1.8% more than its FC counterpart.

On the other hand, the accuracy rates obtained with Model-1 using second order Kervolution and fully connected layers are 65.88%, 62.74%, 64.79% on FGVC-Aircraft, StanfordCars and IndoorCVPR, respectively. Whereas for third order Kervolution Model-1 and fully connected layers, the obtained accuracy rates on FGVC-Aircraft, StanfordCars and IndoorCVPR, respectively, are 66.45%, 63.09%, 65.32%. On the other hand, the accuracy rates obtained with Model-1 are 65.27% on FGVC-Aircraft, 62.19% on StanfordCars and 64.08% on IndoorCVPR, with second order Kervolution and KDL. This represents an improvement up to 0.5% over the fully connected layers. Whereas, with Kervolution and KDL third degree polynomial, the obtained accuracy rates are 66.8% on FGVC-Aircraft, 63.68% on StanfordCars, 65.88% on IndoorCVPR. This kernel improves further the accuracy of the model up to 0.6% more than its FC counterpart. Lastly, the accuracy rates obtained with Model-1 are 65.59% on FGVC-Aircraft, 62.79% on StanfordCars, 64.6% on IndoorCVPR, with second order Kervolution and SVM. This represents an improvement up to 0.9% over the fully connected layers. Whereas, with Kervolution and SVM third degree polynomial, the obtained accuracy rates are 67.03% on FGVC-Aircraft, 63.36% on StanfordCars, 64.24% on IndoorCVPR. This kernel improves further the accuracy of the model up to 1.5% more than its FC counterpart.

Table A.2 – *Results of the different configurations on FER datasets.*

Model	Kernel		RAF - DB	FER2013 - DB	ExpW - DB
VGG-16	FC	2nd order poly	86.42%	69.57%	74.13%
		3rd order poly	87.08%	69.85%	74.46%
	KDL	2nd order poly	86.72%	69.97%	74.84%
		3rd order poly	87.46%	70.29%	75.08%
	SVM	2nd order poly	87.11%	70.32%	75.27%
		3rd order poly	87.82 %	70.62%	76%
Model-1	FC	2nd order poly	87.77%	70.68%	76.25%
		3rd order poly	87.93%	70.95%	76.32%
	KDL	2nd order poly	87.91%	71.01%	76.72%
		3rd order poly	88.14%	71.13%	76.96%
	SVM	2nd order poly	88.52%	71.28%	77.09%
		3rd order poly	88.71 %	71.41%	77.87%

As shown in Table A.2, the accuracy rates obtained with VGG-16-base using second order Kervolution and fully connected layers are 86.42%, 69.57%, 74.13% on RAF-DB, FE2013 and ExpW, respectively. Whereas for third order Kervolution VGG-16-base and fully connected layers, the obtained accuracy rates on RAF-DB, FE2013 and ExpW, respectively, are 87.08%, 69.85%, 74.46%. On the other hand, the accuracy rates obtained with VGG-16 like model are 86.72% for RAF-DB, 69.97% for FER2013, 74.84% for ExpW, with second order Kervolution and KDL. This represents an improvement up to 0.7% over the fully connected layers. Whereas, with Kervolution and KDL third degree polynomial, the obtained accuracy rates are 87.46% for RAF-DB, 70.29% for FER2013, 75.08% for ExpW. This kernel improves further the accuracy of the model up to 0.6% more than its FC counterpart. Lastly, the accuracy rates obtained with VGG-16 like model are 87.11% for RAF-DB, 70.32% for FER2013, 75.27% for ExpW, with second order Kervolution and SVM. This represents an improvement up to 1.2% over the fully connected layers. Whereas, with Kervolution and SVM third degree polynomial, the obtained accuracy rates are 87.82 % for RAF-DB, 70.62% for FER2013, 76% for ExpW. This kernel improves further the accuracy of the model up to 1.6% more than its FC counterpart.

The accuracy rates obtained with Model-1 using second order Kervolution and fully connected layers are 87.77%, 70.68%, 76.25% on RAF-DB, FE2013 and ExpW, respectively. Whereas for third order Kervolution Model-1 and fully connected layers, the obtained accuracy rates on RAF-DB, FE2013 and ExpW, respectively, are 87.93%for RAF-DB, 70.95%, for FER 76.32 for ExpW. On the other hand, the accuracy rates obtained with Model-1 are 87.91% for RAF-DB, 71.01% for FER2013, 76.72% for ExpW, with second order Kervolution and KDL. This represents an improvement up to 0.5% over the fully connected layers. Whereas, with Kervolution and KDL third degree polynomial, the obtained accuracy rates are 88.14% for RAF-DB, 71.13% for FER2013, 76.96% for ExpW. This kernel improves further the accuracy of the model up to 0.6% more than its FC counterpart. Lastly, the accuracy rates obtained with Model-1 are 88.52% for RAF-DB, 71.28% for FER2013, 77.09% for ExpW, with second order Kervolution and SVM. This represents an improvement up to 0.8% over the fully connected layers. Whereas, with Kervolution and SVM third degree polynomial, the obtained accuracy rates are 88.71 % for RAF-DB, 71.41% for FER2013, 77.87% for ExpW. This kernel improves further the accuracy of the model up to 1.5% more than its FC counterpart.

As discussed before, one can clearly conclude that kernel based methods improve considerably the network performance in terms of accuracy. Indeed, using all configurations and with all the datasets used, the network performance was enhanced. Moreover, the kernel methods used have different impact on the network, as well as, the kernel function itself. For instance, kernel SVM can improve the accuracy from 0.8% to 1.5% using second and third order polynomial kernel respectively. Whereas, KDL can improve the accuracy from 0.5% using second polynomial kernel to 0.7% using third order polynomial kernel respectively.

A.3 Conclusion

In this chapter, we investigated the impact of using higher order kernels at different levels of the network. For this purpose, we replaced convolution layers with Kervolution layers proposed in [Wang u. a., 2019]. Similarly, we replaced fully connected layers alternatively with Kernelized Dense Layers (KDL) proposed in [Mahmoudi u. a., 2020] and Kernel Support vector Machines (SVM) [Burges u. a., 1999]. These kernel-based methods are more discriminative in the way that they can learn more complex patterns compared to the linear one. Those methods first maps input data to a higher space. After that, they learn a linear classifier in that space which is similar to a powerful non-linear classifier in the first space. The experimental results performed on challenging Fine-Grained datasets namely, FGVC-Aircraft, StanfordCars and CVPRIndoor as well Facial Expression Recognition (FER) datasets namely, RAF-DB, ExpW and FER2013. demonstrate that these methods outperform the ordinary linear layers when used in a deep network fashion. Finally, all kernel based methods considered in this work allow to improve the network performance. The best result was achieved by kernel SVMs followed by KDL, then Kervolution with FC layers.

B Taylor series kernelized layer for fine-grained recognition

To further demonstrate the efficiency of our Taylor Series Kernelized Layer (TSKL) method, described in chapter 6, we applied the later on fine-grained datasets. This further proves the improvement that this layer can bring to a CNN. In this section, we give more details about the experiments we performed to evaluate the approach described above. First, we give a brief description of the datasets we have used. After that, we describe architecture of the used models and training process. Finally, we discuss the obtained results and compare them to those obtained using the ordinary dense layers.

B.0.1 Datasets and experimental settings

We demonstrated the efficiency of the proposed TSKL on both MLPs and CNNs. In the first case, we built a small network architecture which will be used for an ordinary MLP and an MLP built with TSKL. We will refer to the latter architecture as Multi-TSKL Network (MTSKLN). These networks are composed of two hidden layers with 256 units both and an output layer with a number of units corresponding to the number of categories of each dataset. In the case of MTSKLN, the hidden layers are composed of 128 linear units, 96 second degree polynomial units, and 32 third degree polynomial units. For this experiment, we used datasets with relatively small images namely, MNIST, Fashion-MNIST [Xiao u. a., 2017], CIFAR-10, CIFAR-100 [Krizhevsky u. a., 2009] and UCI-Iris. These datasets can be handled by a small neural network.

In the second case, we used some pre-trained CNNs, namely VGG-16, VGG-19 [Simonyan und Zisserman, 2014], ResNet50 [He u. a., 2016], MobileNet [Howard u. a., 2017] and DenseNet121 [Huang u. a., 2017]. First of all, we start by fine tuning these CNNs with the same process described in [Cui u. a., 2017]. After that, we took only the convolution part of these networks and add two fully connected layers similar to the MLP and MTSKLN used previously. In addition, we built a CNN from scratch to further demonstrate the efficiency of our method. This model architecture is composed of five convolutional blocks. Each block consists of a set of convolution layers followed by a batch normalization layer and a ReLU activation function. The first three blocks are composed of three convolution layers, while the fourth and fifth blocks are composed

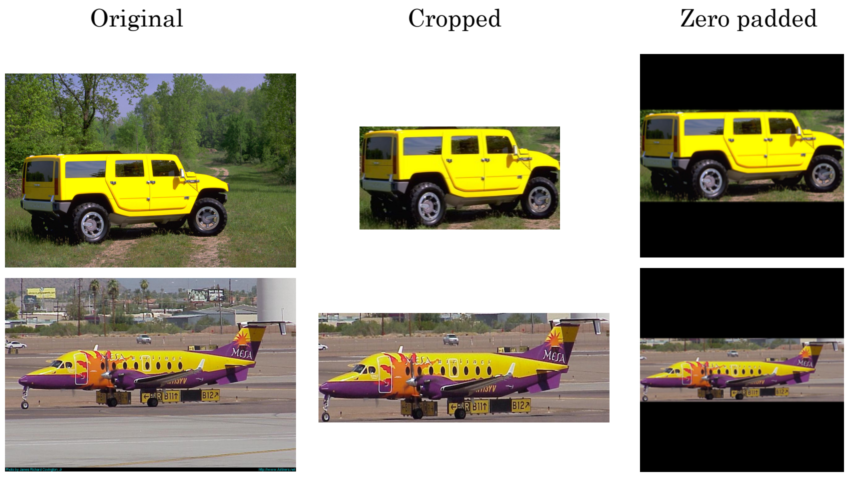


Figure B.1 – *Pre-processing steps for fine-grained datasets.*

of two and one convolution layers, respectively. The convolution kernel size is 5×5 for the first block and 3×3 for the remaining blocks. Each block outputs successively 64, 128, 256, 512 and 512 feature maps. Each block is followed by a dropout layer of 0.3 and a 2×2 Max pooling layer. Two fully connected layers of size 256, 256 are finally added after the last block.

We evaluated these CNN models on three well-known fine-grained datasets, namely: FGVC-Aircraft [Maji u. a., 2013], StanfordCars [Krause u. a., 2013b] and CVPRIndoor [Quattoni und Torralba, 2009]. In all cases, we applied ReLU activation function on linear kernels. For higher order kernels, we added batch normalization followed by Tanh activation function. We have used Adam optimiser with a learning rate starting from 0.001. This learning rate is decreased by a factor of 0.5 if the validation accuracy does not increase over two epochs in the case of MLP and five epochs in the case of CNN. To avoid overfitting, we have first augmented the data using a range degree for random rotations of 20, a shear intensity of 0.2, a range for random zoom of 0.2, randomly flip the inputs horizontally, and subtract it with the pixel-wise image mean. The only preprocessing we have employed for fine-grained datasets is cropping the object region for both FGVC-Aircraft and StanfordCars. We resize the resulting images so that its longer side is 224 while keeping its aspect ratio. We have also zero padded these images to obtain 224×224 pixels. This preprocessing, shown in Fig. B.1, has a considerable impact on learning process. For CVPRIndoor, we only resized the images to 224×224 pixels.

B.0.2 Performance Analysis

In this section, we study the impact of our TSKL on both MLPs and CNNs compared to the ordinary dense layer. The results achieved by TSKL on MLPs and CNNs are reported respectively in Tables B.1 and B.2.

TSKL impact on MLP

In this section, we study the impact of our proposed TSKL layer on an MLP. The goal, here, is not achieve state-of-the-art or competitive results. It is rather to demonstrate the improvement that a TSKL layer can bring when used, solely, in an MLP fashion. The obtained accuracy rate results of MTSKLN compared to MLP are reported in Table B.1.

Table B.1 – *Accuracy rates of MTSKLN network compared to MLP.*

–	MNIST	Fashion MNIST	Cifar-10	Cifar-100	UCI-Iris
MLP	98.34%	**90.14**%	52.46%	26.46%	96%
MTSKLN	**98.6**%	90.09%	**57.52**%	**28.51**%	**98%**

As shown in Table B.1, our proposed MTSKLN outperforms the MLP with all datasets, except for Fashion-MNIST where they achieved quite similar results. Similarly to Fashion-MNIST, MTSKLN slightly outperforms MLP with 0.26% on MNIST. Whereas for both UCI-Iris and Cifar-100, MTSKLN surpasses MLP with more than 2%. Finally, MTSKLN achieves 5.06% more than MLP for Cifar-10. These results demonstrate the superiority of the proposed TSKL layer compared to an ordinary dense layer.

TSKL impact on CNN

In this section, we study the impact of our proposed TSKL layer on CNNs. The obtained accuracy rate results of TSKL compared to fully connected layers are reported in Table B.2.

Table B.2 – *Accuracy rates of CNNs with SKLN compared to CNNs with fully connected layers.*

Model		FGVC-Aircraft	StanfordCars	IndoorCVPR
VGG-16	FC	70.59%	68.76%	71.12%
	TSKL	**71.63%**	**69.42%**	**72.36%**
VGG-19	FC	70.98%	68.92%	71.65%
	TSKL	**72.04%**	**70.09%**	**72.82%**
ResNet50	FC	75.96%	72.41%	74.72%
	TSKL	**77.23%**	**74.06%**	**76.18%**
MobileNet	FC	71.38%	70.02%	70.69%
	TSKL	**72.21%**	**71.61%**	**71.38%**
DenseNet121	FC	72.03%	71.15%	71.86%
	TSKL	**72.87%**	**72.07%**	**72.49%**
Model-1	FC	58.19%	56.78%	58.31%
	TSKL	**60.13%**	**57.22%**	**59.84%**

As reported in Table B.2, CNNs with TSKL outperforms all CNNs with fully connected layers with all models and datasets. The worst results we could achieve with pre-trained CNNs are obtained with both VGG-16 and VGG-19. The reason for such results is that these two CNNs have huge fully connected layers with two 4096 layers and 1000 output layer. Omitting completely these layers impact negatively their performance. Despite this, our TSKL could outperform the fully connected layers, with these CNNs on all datasets. It reaches **71.63%**, **69.42%**, **72.36%** with VGG-16 on respectively FGVC-Aircraft, StanfordCars and CVPRIndoor. This represents an average accuracy rate improvement of 1%. For VGG-19, it reaches **72.04%**, **70.09%** and **72.82%** on respectively FGVC-Aircraft, StanfordCars and CVPRIndoor which represents an average accuracy rate improvement of 1.13%.

The best results we could achieve are obtained with ResNet50; This model does not rely greatly on fully connected layers and therefore, its convolution part carries more knowledge than the convolution part of both VGG-16 and VGG-19. In addition to that, using TSKL allows to improve further the performance of this CNN compared to fully connected layers reaching an accuracy rate of **77.23%**, **74.06%** and **76.18%** on respectively FGVC-Aircraft, StanfordCars and CVPRIndoor, enhacing thus the accuracy with an average of 1.46% on all datasets. TSKL is also beneficial to MobileNet since it reaches an accuracy rate of **72.21%**, **71.61%** and **71.38%** on respectively FGVC-Aircraft, StanfordCars and CVPRIndoor, which represents an average improvement of 1% over fully connected layers. TSKL used on DenseNet121 reaches **72.87%**, **72.07%** and **72.49%** on respectively FGVC-Aircraft, StanfordCars and CVPRIndoor, improving by so the accuracy by an average of 0.76% over fully connected layers.

Finally, in the case of Model-1, even though it reaches clearly lower results compared to pre-trained CNNs, it demonstrates the superiority of our proposed TSKL layer. It reaches **60.13%**, **57.22%** and **59.84%** on respectively FGVC-Aircraft, StanfordCars and CVPRIndoor with an average improvement of 1.3% over the fully connected layers on all datasets.

B.1 Conclusion

In this section, we proposed a new dense layer powered by a Taylor Series Kernel instead of a linear one. The intuition behind, was to map input data into a higher feature space. This mapping was done implicitly using several polynomial kernels of different degrees, leveraging the kernel trick. Then, explicitly by concatenating the output of the first step in the form of a Taylor series kernel. This Taylor Series Kernelized Layer is able to learn more complex patterns than an ordinary dense layer and thus be more discriminative. The experimental results demonstrate that this layer outperforms the ordinary dense layer when used as MLPs, or on top of CNNs on fine-grained datasets.

List of kernels C

C.1 List of kernels

1. **Linear Kernel:** Linear Kernel is used when the data is linearly separable, that is, it can be separated using a single line. It is one of the most common kernels to be used. It is mostly used when there are a large number of features in a particular dataset. One of the examples where there are a lot of features, is Text Classification, as each alphabet is a new feature.

   The linear kernel is the simplest kernel function. It is given by the inner product <x,y> plus an optional constant c.

   $$K(x,y) = x^T y + c \tag{C.1}$$

   The advantages of using linear kernel are:

   - Training with a linear kernel is faster than with any other Kernel.
   - When training with a linear kernel, only the optimisation of the c regularisation parameter is required. On the other hand, when training with other kernels, there is a need to optimise other parameter as well, which means that performing a grid search will usually take more time.

2. **Polynomial Kernel:** In machine learning, the polynomial kernel is a kernel function commonly used with the kernelized models (such as SVMs), that represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models. Intuitively, the polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these. Such combinations are known as interaction features. The (implicit) feature space of a polynomial kernel is equivalent to that of higher feature space, but without the combinatorial blowup in the number of parameters to be learned.

   The Polynomial kernel is a non-stationary kernel. Polynomial kernels are well suited for problems where all the training data is normalized. Adjustable parameters are the slope alpha, the constant term c and the polynomial degree d.

$$K(x,y) = (\alpha x^T y + c)^d \tag{C.2}$$

3. **Gaussian Kernel:**

   The Gaussian kernel is an example of radial basis function kernel.

   $$K(x,y) = exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \tag{C.3}$$

   The adjustable parameter sigma plays a major role in the performance of the kernel, and should be carefully tuned to the problem at hand. If overestimated, the exponential will behave almost linearly and the higher-dimensional projection will start to lose its non-linear power. In the other hand, if underestimated, the function will lack regularization and the decision boundary will be highly sensitive to noise in training data.

4. **Exponential Kernel:** The exponential kernel is closely related to the Gaussian kernel, with only the square of the norm left out. It is also a radial basis function kernel.

   $$K(x,y) = exp\left(-\frac{\|x-y\|}{2\sigma^2}\right) \tag{C.4}$$

5. **Laplacian Kernel:** The Laplace Kernel is completely equivalent to the exponential kernel, except for being less sensitive for changes in the sigma parameter. Being equivalent, it is also a radial basis function kernel.

   $$K(x,y) = exp\left(-\frac{\|x-y\|}{\sigma}\right) \tag{C.5}$$

   It is important to note that the observations made about the sigma parameter for the Gaussian kernel also apply to the Exponential and Laplacian kernels.

6. **ANOVA Kernel:** The ANOVA kernel is also a radial basis function kernel, just as the Gaussian and Laplacian kernels. It is said to perform well in multidimensional regression problems.

   $$K(x,y) = \sum_{k=1}^{n} exp\left(-\sigma(x^k - y^k)^2\right)^d \tag{C.6}$$

7. **Hyperbolic Tangent (Sigmoid) Kernel:** The Hyperbolic Tangent Kernel is also known as the Sigmoid Kernel and as the Multilayer Perceptron (MLP) kernel. The Sigmoid Kernel comes from the Neural Networks field, where the bipolar sigmoid function is often used as an activation function for artificial neurons.

$$K(x,y) = tanh(\alpha x^T y + c)^d \tag{C.7}$$

It is interesting to note that a SVM model using a sigmoid kernel function is equivalent to a two-layer, perceptron neural network. This kernel was quite popular for support vector machines due to its origin from neural network theory. Also, despite being only conditionally positive definite, it has been found to perform well in practice. There are two adjustable parameters in the sigmoid kernel, the slope alpha and the intercept constant c. A common value for alpha is 1/N, where N is the data dimension.

8. **Rational Quadratic Kernel:** The Rational Quadratic kernel is less computationally intensive than the Gaussian kernel and can be used as an alternative when using the Gaussian becomes too expensive.

$$K(x,y) = 1 - \frac{\|x-y\|^2}{\|x-y\|^2 + c} \tag{C.8}$$

9. **Multiquadric Kernel:** The Multiquadric kernel can be used in the same situations as the Rational Quadratic kernel. As is the case with the Sigmoid kernel, it is also an example of an non-positive definite kernel.

$$K(x,y) = \sqrt{\|x-y\|^2 + c} \tag{C.9}$$

10. **Inverse Multiquadric Kernel:** The Inverse Multi Quadric kernel. As with the Gaussian kernel, it results in a kernel matrix with full rank and thus forms a infinite dimension feature space.

$$K(x,y) = \frac{1}{\sqrt{\|x-y\|^2 + c}} \tag{C.10}$$

11. **Circular Kernel:** The circular kernel is used in geostatic applications. It is an example of an isotropic stationary kernel and is positive definite in $\mathbb{R}^2$.

$$K(x,y) = \frac{2}{\pi} arccos\left(-\frac{\|x-y\|}{\sigma}\right) - \frac{2}{\pi} - \frac{\|x-y\|}{\sigma}\sqrt{1-\left(\frac{\|x-y\|}{\sigma}\right)^2} \tag{C.11}$$

$$if \|x-y\| < \sigma, zerootherwise.$$

12. **Spherical Kernel:** The spherical kernel is similar to the circular kernel, but is positive definite in $\mathbb{R}^3$.

$$K(x,y) = 1 - \frac{3}{2}\frac{\|x-y\|}{\sigma} + \frac{1}{2}\left(\frac{\|x-y\|}{\sigma}\right)^3 \quad \text{(C.12)}$$
$$if\|x-y\| < \sigma, zerootherwise.$$

13. **Power Kernel:** The Power kernel is also known as the (unrectified) triangular kernel. It is an example of scale-invariant kernel and is also only conditionally positive definite.

$$K(x,y) = -\|x-y\|^d \quad \text{(C.13)}$$

14. **Log Kernel:** The Log kernel seems to be particularly interesting for images, but is only conditionally positive definite.

$$K(x,y) = -\log\left(\|x-y\|^d + 1\right) \quad \text{(C.14)}$$

15. **Chi-Square Kernel:** The Chi-Square kernel comes from the Chi-Square distribution:

$$K(x,y) = 1 - \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{\frac{1}{2}(x_i + y_i)} \quad \text{(C.15)}$$

However, as noted by commenter Alexis Mignon, this version of the kernel is only conditionally positive-definite (CPD). A positive-definite version of this kernel is given as

$$K(x,y) = \sum_{i=1}^{n} \frac{2x_i y_i}{(x_i + y_i)} \quad \text{(C.16)}$$

and is suitable to be used by methods other than support vector machines.

16. **Histogram Intersection Kernel:** The Histogram Intersection Kernel is also known as the Min Kernel and has been proven useful in image classification.

$$K(x,y) = \sum_{i=1}^{n} min(x_i, y_i) \quad \text{(C.17)}$$

17. **Generalized Histogram Intersection:** The Generalized Histogram Intersection kernel is built based on the Histogram Intersection Kernel for image classification but applies in a much larger variety of contexts. It is given by:

$$K(x,y) = \sum_{i=1}^{n} min(|x_i|^{\alpha}, |y_i|^{\beta}) \quad \text{(C.18)}$$

18. **Generalized T-Student Kernel:** The Generalized T-Student Kernel has been proven to be a Mercel Kernel, thus having a positive semi-definite Kernel matrix. It is given by:

$$K(x,y) = \frac{1}{1 + \|x-y\|^d} \tag{C.19}$$

19. **Bayesian Kernel:** The Bayesian kernel could be given as:

$$K(x,y) = \prod_{i=1}^{n} k_i(x_i, y_i) \tag{C.20}$$

where:

$$k_i(a,b) = \sum_{c \in \{0;1\}} P(Y = c | X_i = a) P(Y = c | X_i = b) \tag{C.21}$$

Bibliography

[Abadi u. a. 2016] Abadi, Martín ; Agarwal, Ashish ; Barham, Paul ; Brevdo, Eugene ; Chen, Zhifeng ; Citro, Craig ; Corrado, Greg S. ; Davis, Andy ; Dean, Jeffrey ; Devin, Matthieu u. a.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. In: *arXiv preprint arXiv:1603.04467* (2016)

[Acharya u. a. 2018a] Acharya, Dinesh ; Huang, Zhiwu ; Pani Paudel, Danda ; Van Gool, Luc: Covariance pooling for facial expression recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, S. 367–374

[Acharya u. a. 2018b] Acharya, Dinesh ; Huang, Zhiwu ; Pani Paudel, Danda ; Van Gool, Luc: Covariance Pooling for Facial Expression Recognition. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018

[Assiri 2020] Assiri, Yahia: Stochastic Optimization of Plain Convolutional Neural Networks with Simple methods. In: *arXiv preprint arXiv:2001.08856* (2020)

[Bacivarov 2009] Bacivarov, Ioana: *Advances in the modelling of facial sub-regions and facial expressions using active appearance techniques*, Dissertation, 2009

[Bai u. a. 2021] Bai, Mengjiong ; Goecke, Roland ; Herath, Damith: Micro-Expression Recognition Based On Video Motion Magnification And Pre-Trained Neural Network. In: *2021 IEEE International Conference on Image Processing (ICIP)* IEEE (Veranst.), 2021, S. 549–553

[Bishay u. a. 2019] Bishay, Mina ; Palasek, Petar ; Priebe, Stefan ; Patras, Ioannis: SchiNet: Automatic Estimation of Symptoms of Schizophrenia from Facial Behaviour Analysis. In: *IEEE Transactions on Affective Computing* (2019)

[Bosagh und Ramsundar 2018] Bosagh, Reza Z. ; Ramsundar, Bharath: *TensorFlow for deep learning*. O'Reilly Media, Incorporated, 2018

[Burges u. a. 1999] Burges, Christopher J. ; Scholkopf, Bernhard ; Smola, Alexander J.: *Advances in kernel methods: support vector learning*. MIT press Cambridge, MA, USA:, 1999

[Byerly u. a. 2020] Byerly, Adam ; Kalganova, Tatiana ; Dear, Ian: A Branching and Merging Convolutional Network with Homogeneous Filter Capsules. In: *arXiv preprint arXiv:2001.09136* (2020)

[Cai u. a. 2018] Cai, Jie ; Meng, Zibo ; Khan, Ahmed S. ; Li, Zhiyuan ; O'Reilly, James ; Tong, Yan: Island loss for learning discriminative features in facial expression recognition. In: *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)* IEEE (Veranst.), 2018, S. 302–309

[Chen u. a. 2020] Chen, Shizhe ; Zhao, Yida ; Jin, Qin ; Wu, Qi: Fine-grained Video-Text Retrieval with Hierarchical Graph Reasoning. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, S. 10638–10647

[Chen u. a. 2015] Chen, Tianqi ; Li, Mu ; Li, Yutian ; Lin, Min ; Wang, Naiyan ; Wang, Minjie ; Xiao, Tianjun ; Xu, Bing ; Zhang, Chiyuan ; Zhang, Zheng: Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. In: *arXiv preprint arXiv:1512.01274* (2015)

[Chen u. a. 2018] Chen, Weikai ; Han, Xiaoguang ; Li, Guanbin ; Chen, Chao ; Xing, Jun ; Zhao, Yajie ; Li, Hao: Deep rbfnet: Point cloud feature learning using radial basis functions. In: *arXiv preprint arXiv:1812.04302* (2018)

[Chetlur u. a. 2014] Chetlur, Sharan ; Woolley, Cliff ; Vandermersch, Philippe ; Cohen, Jonathan ; Tran, John ; Catanzaro, Bryan ; Shelhamer, Evan: cudnn: Efficient primitives for deep learning. In: *arXiv preprint arXiv:1410.0759* (2014)

[Chetouani u. a. 2020] Chetouani, Aladine ; Treuillet, Sylvie ; Exbrayat, Matthieu ; Jesset, Sébastien: Classification of engraved pottery sherds mixing deep-learning features by compact bilinear pooling. In: *Pattern Recognition Letters* 131 (2020), S. 1–7

[Chien und Hsieh 2013] Chien, Jen-Tzung ; Hsieh, Hsin-Lung: Nonstationary source separation using sequential and variational Bayesian learning. In: *IEEE Transactions on Neural Networks and Learning Systems* 24 (2013), Nr. 5, S. 681–694

[Cho u. a. 2014] Cho, Kyunghyun ; Van Merriënboer, Bart ; Gulcehre, Caglar ; Bahdanau, Dzmitry ; Bougares, Fethi ; Schwenk, Holger ; Bengio, Yoshua: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: *arXiv preprint arXiv:1406.1078* (2014)

[Cho und Saul 2009] Cho, Youngmin ; Saul, Lawrence: Kernel methods for deep learning. In: *Advances in neural information processing systems* 22 (2009)

[Choy u. a. 2006] Choy, Min C. ; Srinivasan, Dipti ; Cheu, Ruey L.: Neural networks for continuous online learning and control. In: *IEEE Transactions on Neural Networks* 17 (2006), Nr. 6, S. 1511–1531

[Clevert u. a. 2015] Clevert, Djork-Arné ; Unterthiner, Thomas ; Hochreiter, Sepp: Fast and accurate deep network learning by exponential linear units (elus). In: *arXiv preprint arXiv:1511.07289* (2015)

[Cohen und Welling 2016] Cohen, Taco ; Welling, Max: Group equivariant convolutional networks. In: *International conference on machine learning*, 2016, S. 2990–2999

[Collobert u. a. 2002] Collobert, Ronan ; Bengio, Samy ; Mariéthoz, Johnny: Torch: a modular machine learning software library / Idiap. 2002. – Forschungsbericht

[Cootes u. a. 2001] Cootes, Timothy F. ; Edwards, Gareth J. ; Taylor, Christopher J.: Active appearance models. In: *IEEE Transactions on pattern analysis and machine intelligence* 23 (2001), Nr. 6, S. 681–685

[Cootes u. a. 1995] Cootes, Timothy F. ; Taylor, Christopher J. ; Cooper, David H. ; Graham, Jim: Active shape models-their training and application. In: *Computer vision and image understanding* 61 (1995), Nr. 1, S. 38–59

[Cortes und Vapnik 1995] Cortes, Corinna ; Vapnik, Vladimir: Support-vector networks. In: *Machine learning* 20 (1995), Nr. 3, S. 273–297

[Cui u. a. 2017] Cui, Yin ; Zhou, Feng ; Wang, Jiang ; Liu, Xiao ; Lin, Yuanqing ; Belongie, Serge: Kernel pooling for convolutional neural networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, S. 2921–2930

[Dahl u. a. 2011] Dahl, George E. ; Yu, Dong ; Deng, Li ; Acero, Alex: Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. In: *IEEE Transactions on audio, speech, and language processing* 20 (2011), Nr. 1, S. 30–42

[Dalal und Triggs 2005] Dalal, Navneet ; Triggs, Bill: Histograms of oriented gradients for human detection. In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)* Bd. 1 Ieee (Veranst.), 2005, S. 886–893

[Deng 2014] Deng, Li: A tutorial survey of architectures, algorithms, and applications for deep learning. In: *APSIPA transactions on Signal and Information Processing* 3 (2014)

[Deng u. a. 2015] Deng, Weihong ; Hu, Jiani ; Zhang, Shuo ; Guo, Jun: DeepEmo: real-world facial expression analysis via deep learning. In: *2015 Visual Communications and Image Processing (VCIP)* IEEE (Veranst.), 2015, S. 1–4

[Dhall u. a. 2011] Dhall, Abhinav ; Goecke, Roland ; Lucey, Simon ; Gedeon, Tom: Static facial expression analysis in tough conditions: Data, evaluation protocol and benchmark. In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)* IEEE (Veranst.), 2011, S. 2106–2112

[Dhall u. a. 2012] Dhall, Abhinav ; Goecke, Roland ; Lucey, Simon ; Gedeon, Tom: Collecting large, richly annotated facial-expression databases from movies. In: *IEEE Annals of the History of Computing* 19 (2012), Nr. 03, S. 34–41

[Dubey und Singh 2016] Dubey, Monika ; Singh, Lokesh: Automatic Emotion Recognition Using Facial Expression: A Review. In: *International Research Journal of Engineering and Technology (IRJET)* (2016)

[Ekman und Friesen 1971] Ekman, Paul ; Friesen, Wallace V.: Constants across cultures in the face and emotion. In: *Journal of personality and social psychology* 17 (1971), Nr. 2, S. 124

[El Ayadi u. a. 2011] El Ayadi, Moataz ; Kamel, Mohamed S. ; Karray, Fakhri: Survey on speech emotion recognition: Features, classification schemes, and databases. In: *Pattern recognition* 44 (2011), Nr. 3, S. 572–587

[Elisseeff und Weston 2001] Elisseeff, André ; Weston, Jason: A kernel method for multi-labelled classification. In: *Advances in neural information processing systems* 14 (2001)

[Fabian Benitez-Quiroz u. a. 2016] Fabian Benitez-Quiroz, C ; Srinivasan, Ramprakash ; Martinez, Aleix M.: Emotionet: An accurate, real-time algorithm for the automatic annotation of a million facial expressions in the wild. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, S. 5562–5570

[Fukui u. a. 2016] Fukui, Akira ; Park, Dong H. ; Yang, Daylen ; Rohrbach, Anna ; Darrell, Trevor ; Rohrbach, Marcus: Multimodal compact bilinear pooling for visual question answering and visual grounding. In: *arXiv preprint arXiv:1606.01847* (2016)

[Gao u. a. 2016] Gao, Yang ; Beijbom, Oscar ; Zhang, Ning ; Darrell, Trevor: Compact bilinear pooling. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, S. 317–326

[Gao u. a. 2020] Gao, Yu ; Han, Xintong ; Wang, Xun ; Huang, Weilin ; Scott, Matthew: Channel Interaction Networks for Fine-Grained Image Categorization. In: *AAAI*, 2020, S. 10818–10825

[Gao u. a. 2019a] Gao, Zilin ; Xie, Jiangtao ; Wang, Qilong ; Li, Peihua: Global Second-order Pooling Convolutional Networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, S. 3024–3033

[Gao u. a. 2019b] Gao, Ziteng ; Wang, Limin ; Wu, Gangshan: LIP: Local Importance-based Pooling. In: *arXiv preprint arXiv:1908.04156* (2019)

[Gharsalli 2016] Gharsalli, Sonia: *Reconnaissance des émotions par traitement d'images*, Université d'Orléans, Dissertation, 2016

[Girshick 2015] Girshick, Ross: Fast r-cnn. In: *Proceedings of the IEEE international conference on computer vision*, 2015, S. 1440–1448

[Girshick u. a. 2014] Girshick, Ross ; Donahue, Jeff ; Darrell, Trevor ; Malik, Jitendra: Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, S. 580–587

[Glorot und Bengio 2010] Glorot, Xavier ; Bengio, Yoshua: Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics* JMLR Workshop and Conference Proceedings (Veranst.), 2010, S. 249–256

[Goodfellow u. a. 2016] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep learning*. MIT press, 2016

[Goodfellow u. a. 2014] Goodfellow, Ian ; Pouget-Abadie, Jean ; Mirza, Mehdi ; Xu, Bing ; Warde-Farley, David ; Ozair, Sherjil ; Courville, Aaron ; Bengio, Yoshua: Generative adversarial nets. In: *Advances in neural information processing systems* 27 (2014)

[Goodfellow u. a. 2013] Goodfellow, Ian J. ; Erhan, Dumitru ; Carrier, Pierre L. ; Courville, Aaron ; Mirza, Mehdi ; Hamner, Ben ; Cukierski, Will ; Tang, Yichuan ; Thaler, David ; Lee, Dong-Hyun u. a.: Challenges in representation learning: A report on three machine learning contests. In: *International Conference on Neural Information Processing, Springer* (2013), S. 117–124

[Grandini u. a. 2020] Grandini, Margherita ; Bagli, Enrico ; Visani, Giorgio: Metrics for multi-class classification: an overview. In: *arXiv preprint arXiv:2008.05756* (2020)

[Grauman und Darrell 2005] Grauman, Kristen ; Darrell, Trevor: The pyramid match kernel: Discriminative classification with sets of image features. In: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1* Bd. 2 IEEE (Veranst.), 2005, S. 1458–1465

[Gross u. a. 2010] Gross, Ralph ; Matthews, Iain ; Cohn, Jeffrey ; Kanade, Takeo ; Baker, Simon: Multi-pie. In: *Image and vision computing* 28 (2010), Nr. 5, S. 807–813

[Guo u. a. 2016] Guo, Yanan ; Tao, Dapeng ; Yu, Jun ; Xiong, Hao ; Li, Yaotang ; Tao, Dacheng: Deep neural networks with relativity learning for facial expression recognition. In: *2016 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)* IEEE (Veranst.), 2016, S. 1–6

[Ha u. a. 2015] Ha, Hsin-Yu ; Yang, Yimin ; Pouyanfar, Samira ; Tian, Haiman ; Chen, Shu-Ching: Correlation-based deep learning for multimedia semantic concept detection. In: *International Conference on Web Information Systems Engineering* Springer (Veranst.), 2015, S. 473–487

[He u. a. 2015a] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015

[He u. a. 2015b] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Spatial pyramid pooling in deep convolutional networks for visual recognition. In: *IEEE transactions on pattern analysis and machine intelligence* 37 (2015), Nr. 9, S. 1904–1916

[He u. a. 2016] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, S. 770–778

[Hinton u. a. 2012] Hinton, Geoffrey E. ; Srivastava, Nitish ; Krizhevsky, Alex ; Sutskever, Ilya ; Salakhutdinov, Ruslan R.: Improving neural networks by preventing co-adaptation of feature detectors. In: *arXiv preprint arXiv:1207.0580* (2012)

[Howard u. a. 2017] Howard, Andrew G. ; Zhu, Menglong ; Chen, Bo ; Kalenichenko, Dmitry ; Wang, Weijun ; Weyand, Tobias ; Andreetto, Marco ; Adam, Hartwig: Mobilenets: Efficient convolutional neural networks for mobile vision applications. In: *arXiv preprint arXiv:1704.04861* (2017)

[Huang u. a. 2017] Huang, Gao ; Liu, Zhuang ; Van Der Maaten, Laurens ; Weinberger, Kilian Q.: Densely connected convolutional networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, S. 4700–4708

[Huang u. a. 2019] Huang, Yanping ; Cheng, Youlong ; Bapna, Ankur ; Firat, Orhan ; Chen, Dehao ; Chen, Mia ; Lee, HyoukJoong ; Ngiam, Jiquan ; Le, Quoc V. ; Wu, Yonghui u. a.: Gpipe: Efficient training of giant neural networks using pipeline parallelism. In: *Advances in Neural Information Processing Systems*, 2019, S. 103–112

[Huang u. a. 2020] Huang, Zhanchao ; Wang, Jianlin ; Fu, Xuesong ; Yu, Tao ; Guo, Yongqi ; Wang, Rutong: DC-SPP-YOLO: Dense connection and spatial pyramid pooling based YOLO for object detection. In: *Information Sciences* 522 (2020), S. 241–258

[Huang und Li 2020] Huang, Zixuan ; Li, Yin: Interpretable and Accurate Fine-grained Recognition via Region Grouping. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, S. 8662–8672

[Hubel und Wiesel 1962] Hubel, David H. ; Wiesel, Torsten N.: Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. In: *The Journal of physiology* 160 (1962), Nr. 1, S. 106

[Hyun u. a. 2019] Hyun, Junhyuk ; Seong, Hongje ; Kim, Euntai: Universal Pooling–A New Pooling Method for Convolutional Neural Networks. In: *arXiv preprint arXiv:1907.11440* (2019)

[Jayasumana u. a. 2020] Jayasumana, Sadeep ; Ramalingam, Srikumar ; Kumar, Sanjiv: Kernelized Classification in Deep Networks. In: *arXiv preprint arXiv:2012.09607* (2020)

[Jayasundara u. a. 2019] Jayasundara, Vinoj ; Jayasekara, Sandaru ; Jayasekara, Hirunima ; Rajasegaran, Jathushan ; Seneviratne, Suranga ; Rodrigo, Ranga: Textcaps: Handwritten character recognition with very small datasets. In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)* IEEE (Veranst.), 2019, S. 254–262

[Ji u. a. 2020] Ji, Ruyi ; Wen, Longyin ; Zhang, Libo ; Du, Dawei ; Wu, Yanjun ; Zhao, Chen ; Liu, Xianglong ; Huang, Feiyue: Attention Convolutional Binary Neural Tree for Fine-Grained Visual Categorization. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, S. 10468–10477

[Jia u. a. 2014] Jia, Yangqing ; Shelhamer, Evan ; Donahue, Jeff ; Karayev, Sergey ; Long, Jonathan ; Girshick, Ross ; Guadarrama, Sergio ; Darrell, Trevor: Caffe: Convolutional architecture for fast feature embedding. In: *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, S. 675–678

[Kanade u. a. 2000] Kanade, Takeo ; Cohn, Jeffrey F. ; Tian, Yingli: Comprehensive database for facial expression analysis. In: *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)* IEEE (Veranst.), 2000, S. 46–53

[Karpathy u. a. 2014] Karpathy, Andrej ; Toderici, George ; Shetty, Sanketh ; Leung, Thomas ; Sukthankar, Rahul ; Fei-Fei, Li: Large-scale video classification with convolutional neural networks. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, S. 1725–1732

[Kim u. a. 2016] Kim, Bo-Kyeong ; Dong, Suh-Yeon ; Roh, Jihyeon ; Kim, Geonmin ; Lee, Soo-Young: Fusing aligned and non-aligned face information for automatic affect recognition in the wild: a deep learning approach. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2016), S. 48–57

[Klambauer u. a. 2017] Klambauer, Günter ; Unterthiner, Thomas ; Mayr, Andreas ; Hochreiter, Sepp: Self-normalizing neural networks. In: *Advances in neural information processing systems*, 2017, S. 971–980

[Kolesnikov u. a. 2019] Kolesnikov, Alexander ; Beyer, Lucas ; Zhai, Xiaohua ; Puigcerver, Joan ; Yung, Jessica ; Gelly, Sylvain ; Houlsby, Neil: Large Scale Learning of General Visual Representations for Transfer. In: *arXiv preprint arXiv:1912.11370* (2019)

[Kovalev u. a. 2016] Kovalev, Vassili ; Kalinovsky, Alexander ; Kovalev, Sergey: Deep learning with theano, torch, caffe, tensorflow, and deeplearning4j: Which one is the best in speed and accuracy? (2016)

[Krause u. a. 2013a] Krause, Jonathan ; Stark, Michael ; Deng, Jia ; Fei-Fei, Li: 3D Object Representations for Fine-Grained Categorization. In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013

[Krause u. a. 2013b] Krause, Jonathan ; Stark, Michael ; Deng, Jia ; Fei-Fei, Li: 3D Object Representations for Fine-Grained Categorization. In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013

[Krizhevsky u. a. 2009] Krizhevsky, Alex ; Hinton, Geoffrey u. a.: Learning multiple layers of features from tiny images. (2009)

[Krizhevsky u. a. 2012] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems* (2012), S. 1097–1105

[Kuo u. a. 2018] Kuo, Chieh-Ming ; Lai, Shang-Hong ; Sarkis, Michel: A compact deep learning model for robust facial expression recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, S. 2121–2129

[Lazebnik u. a. 2006] Lazebnik, Svetlana ; Schmid, Cordelia ; Ponce, Jean: Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In: *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)* Bd. 2 IEEE (Veranst.), 2006, S. 2169–2178

[Le 2013] Le, Quoc V.: Building high-level features using large scale unsupervised learning. In: *2013 IEEE international conference on acoustics, speech and signal processing* IEEE (Veranst.), 2013, S. 8595–8598

[Learning 2017] Learning, Transfer: *Convolutional Neural Network for Visual Recognition*. 2017

[LeCun u. a. 2015a] LeCun, Yann u. a.: LeNet-5, convolutional neural networks. In: *URL: http://yann. lecun. com/exdb/lenet* 20 (2015), Nr. 5, S. 14

[LeCun u. a. 1995] LeCun, Yann ; Bengio, Yoshua u. a.: Convolutional networks for images, speech, and time series. In: *The handbook of brain theory and neural networks* 3361 (1995), Nr. 10, S. 1995

[LeCun u. a. 2015b] LeCun, Yann ; Bengio, Yoshua ; Hinton, Geoffrey: Deep learning. In: *nature* 521 (2015), Nr. 7553, S. 436–444

[Li und Deng 2018a] Li, Shan ; Deng, Weihong: Deep facial expression recognition: A survey. In: *arXiv preprint arXiv:1804.08348* (2018)

[Li und Deng 2018b] Li, Shan ; Deng, Weihong: Reliable crowdsourcing and deep locality-preserving learning for unconstrained facial expression recognition. In: *IEEE Transactions on Image Processing* 28 (2018), Nr. 1, S. 356–370

[Li u. a. 2017] Li, Shan ; Deng, Weihong ; Du, JunPing: Reliable crowdsourcing and deep locality-preserving learning for expression recognition in the wild. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on* IEEE (Veranst.), 2017, S. 2584–2593

[Li und Wu 2015] Li, Xiangang ; Wu, Xihong: Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In: *2015 ieee international conference on acoustics, speech and signal processing (icassp)* IEEE (Veranst.), 2015, S. 4520–4524

[Li u. a. 2020] Li, Yante ; Huang, Xiaohua ; Zhao, Guoying: Joint Local and Global Information Learning With Single Apex Frame Detection for Micro-Expression Recognition. In: *IEEE Transactions on Image Processing* 30 (2020), S. 249–263

[Lian u. a. 2020] Lian, Zheng ; Li, Ya ; Tao, Jian-Hua ; Huang, Jian ; Niu, Ming-Yue: Expression analysis based on face regions in real-world conditions. In: *International Journal of Automation and Computing* 17 (2020), Nr. 1, S. 96–107

[Liao u. a. 2019] Liao, Qiyu ; Wang, Dadong ; Holewa, Hamish ; Xu, Min: Squeezed bilinear pooling for fine-grained visual categorization. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, S. 0–0

[Lin und Maji 2017] Lin, Tsung-Yu ; Maji, Subhransu: Improved bilinear pooling with cnns. In: *arXiv preprint arXiv:1707.06772* (2017)

[Lin u. a. 2015] Lin, Tsung-Yu ; RoyChowdhury, Aruni ; Maji, Subhransu: Bilinear cnn models for fine-grained visual recognition. In: *Proceedings of the IEEE international conference on computer vision*, 2015, S. 1449–1457

[Liu u. a. 2017a] Liu, Wenhan ; Zhang, Mengxin ; Zhang, Yidan ; Liao, Yuan ; Huang, Qijun ; Chang, Sheng ; Wang, Hao ; He, Jin: Real-time multilead convolutional neural network for myocardial infarction detection. In: *IEEE journal of biomedical and health informatics* 22 (2017), Nr. 5, S. 1434–1444

[Liu u. a. 2017b] Liu, Zhiwen ; Li, Shan ; Deng, Weihong: Boosting-POOF: boosting part based one vs one feature for facial expression recognition in the wild. In: *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)* IEEE (Veranst.), 2017, S. 967–972

[Liu u. a. 2020] Liu, Zongdai ; Lu, Feixiang ; Wang, Peng ; Miao, Hui ; Zhang, Liangjun ; Yang, Ruigang ; Zhou, Bin: 3D Part Guided Image Editing for Fine-Grained Object Understanding. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, S. 11336–11345

[López-Sánchez u. a. 2020] López-Sánchez, Daniel ; Arrieta, Angélica G. ; Corchado, Juan M.: Compact bilinear pooling via kernelized random projection for fine-grained image categorization on low computational power devices. In: *Neurocomputing* 398 (2020), S. 411–421

[Loshchilov und Hutter 2017] Loshchilov, Ilya ; Hutter, Frank: Decoupled weight decay regularization. In: *arXiv preprint arXiv:1711.05101* (2017)

[Lowe 1999] Lowe, David G.: Object recognition from local scale-invariant features. In: *Proceedings of the seventh IEEE international conference on computer vision* Bd. 2 Ieee (Veranst.), 1999, S. 1150–1157

[Lyons u. a. 1998] Lyons, Michael ; Akamatsu, Shigeru ; Kamachi, Miyuki ; Gyoba, Jiro: Coding facial expressions with gabor wavelets. In: *Proceedings Third IEEE international conference on automatic face and gesture recognition* IEEE (Veranst.), 1998, S. 200–205

[Mahmoudi u. a. 2021a] Mahmoudi, M ; Chetouani, Aladine ; Boufera, Fatma ; Tabia, Hedi: Deep Kernelized Network for Fine-Grained Recognition. In: *International Conference on Neural Information Processing* Springer (Veranst.), 2021, S. 100–111

[Mahmoudi u. a. 2020] Mahmoudi, M. A. ; Chetouani, A. ; Boufera, F. ; Tabia, H.: Kernelized Dense Layers For Facial Expression Recognition. In: *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, S. 2226–2230

[Mahmoudi u. a. 2020] Mahmoudi, M A. ; Chetouani, Aladine ; Boufera, Fatma ; Tabia, Hedi: Learnable pooling weights for facial expression recognition. In: *Pattern Recognition Letters* 138 (2020)

[Mahmoudi u. a. 2021b] Mahmoudi, M A. ; Chetouani, Aladine ; Boufera, Fatma ; Tabia, Hedi: Improved Bilinear Model for Facial Expression Recognition. In: *Pattern Recognition and Artificial Intelligence. MedPRAI 2020. Communications in Computer and Information Science* Bd. 1322 Springer (Veranst.), 2021, S. 47–59

[Mahmoudi u. a. 2021c] Mahmoudi, M A. ; Chetouani, Aladine ; Boufera, Fatma ; Tabia, Hedi: Taylor Series Kernelized Layer for Fine-Grained Recognition. In: *2021 IEEE International Conference on Image Processing (ICIP)* IEEE (Veranst.), 2021, S. 1914–1918

[Mahmoudi u. a. 2022] Mahmoudi, M A. ; Chetouani, Aladine ; Boufera, Fatma ; Tabia, Hedi: Kernel-based convolution expansion for facial expression recognition. In: *Pattern Recognition Letters* (2022)

[Maji u. a. 2013] Maji, Subhransu ; Rahtu, Esa ; Kannala, Juho ; Blaschko, Matthew ; Vedaldi, Andrea: Fine-grained visual classification of aircraft. In: *arXiv preprint arXiv:1306.5151* (2013)

[McCallum 1999] McCallum, Andrew K.: Multi-label text classification with a mixture model trained by EM. In: *AAAI 99 workshop on text learning* Citeseer (Veranst.), 1999

[McKeown u. a. 2011] McKeown, Gary ; Valstar, Michel ; Cowie, Roddy ; Pantic, Maja ; Schroder, Marc: The semaine database: Annotated multimodal records of emotionally colored conversations between a person and a limited agent. In: *IEEE transactions on affective computing* 3 (2011), Nr. 1, S. 5–17

[Mehrabian 2008] Mehrabian, Albert: Communication without words. In: *Communication theory* (2008), S. 193–200

[Mollahosseini u. a. 2017] Mollahosseini, Ali ; Hasani, Behzad ; Mahoor, Mohammad H.: Affectnet: A database for facial expression, valence, and arousal computing in the wild. In: *IEEE Transactions on Affective Computing* (2017)

[Najafabadi u. a. 2015] Najafabadi, Maryam M. ; Villanustre, Flavio ; Khoshgoftaar, Taghi M. ; Seliya, Naeem ; Wald, Randall ; Muharemagic, Edin: Deep learning applications and challenges in big data analytics. In: *Journal of big data* 2 (2015), Nr. 1, S. 1–21

[Nguyen u. a. 2018] Nguyen, Dung ; Nguyen, Kien ; Sridharan, Sridha ; Dean, David ; Fookes, Clinton: Deep spatio-temporal feature fusion with compact bilinear pooling for multimodal emotion recognition. In: *Computer Vision and Image Understanding* 174 (2018), S. 33–42

[Noh u. a. 2015] Noh, Hyeonwoo ; Hong, Seunghoon ; Han, Bohyung: Learning deconvolution network for semantic segmentation. In: *Proceedings of the IEEE international conference on computer vision*, 2015, S. 1520–1528

[Pantic und Rothkrantz 2000] Pantic, Maja ; Rothkrantz, Leon J. M.: Automatic analysis of facial expressions: The state of the art. In: *IEEE Transactions on pattern analysis and machine intelligence* 22 (2000), Nr. 12, S. 1424–1445

[Pantic u. a. 2005] Pantic, Maja ; Valstar, Michel ; Rademaker, Ron ; Maat, Ludo: Web-based database for facial expression analysis. In: *2005 IEEE international conference on multimedia and Expo* IEEE (Veranst.), 2005, S. 5–pp

[Pouyanfar und Chen 2017] Pouyanfar, Samira ; Chen, Shu-Ching: T-LRA: Trend-based learning rate annealing for deep neural networks. In: *2017 IEEE Third International Conference on Multimedia Big Data (BigMM)* IEEE (Veranst.), 2017, S. 50–57

[Pouyanfar u. a. 2018] Pouyanfar, Samira ; Sadiq, Saad ; Yan, Yilin ; Tian, Haiman ; Tao, Yudong ; Reyes, Maria P. ; Shyu, Mei-Ling ; Chen, Shu-Ching ; Iyengar, Sundaraja S.: A survey on deep learning: Algorithms, techniques, and applications. In: *ACM Computing Surveys (CSUR)* 51 (2018), Nr. 5, S. 1–36

[Quattoni und Torralba 2009] Quattoni, Ariadna ; Torralba, Antonio: Recognizing indoor scenes. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* IEEE (Veranst.), 2009, S. 413–420

[Radford u. a. 2015] Radford, Alec ; Metz, Luke ; Chintala, Soumith: Unsupervised representation learning with deep convolutional generative adversarial networks. In: *arXiv preprint arXiv:1511.06434* (2015)

[Ridnik u. a. 2020] Ridnik, Tal ; Lawen, Hussam ; Noy, Asaf ; Friedman, Itamar: TResNet: High Performance GPU-Dedicated Architecture. In: *arXiv preprint arXiv:2003.13630* (2020)

[Robert 2014] Robert, Christian: *Machine learning, a probabilistic perspective*. 2014

[Saeed 2021] Saeed, Usman: Facial micro-expressions as a soft biometric for person recognition. In: *Pattern Recognition Letters* 143 (2021), S. 95–103

[Saeedan u. a. 2018] Saeedan, Faraz ; Weber, Nicolas ; Goesele, Michael ; Roth, Stefan: Detail-preserving pooling in deep networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, S. 9108–9116

[Schapire und Singer 2000] Schapire, Robert E. ; Singer, Yoram: BoosTexter: A boosting-based system for text categorization. In: *Machine learning* 39 (2000), Nr. 2, S. 135–168

[Scherer u. a. 2010] Scherer, Dominik ; Müller, Andreas ; Behnke, Sven: Evaluation of pooling operations in convolutional architectures for object recognition. In: *International conference on artificial neural networks* Springer (Veranst.), 2010, S. 92–101

[Schölkopf u. a. 2002] Schölkopf, Bernhard ; Smola, Alexander J. ; Bach, Francis u. a.: *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002

[Sermanet u. a. 2012] Sermanet, Pierre ; Chintala, Soumith ; LeCun, Yann: Convolutional neural networks applied to house numbers digit classification. In: *Proceedings of the 21st international conference on pattern recognition (ICPR2012)* IEEE (Veranst.), 2012, S. 3288–3291

[Sermanet u. a. 2013] Sermanet, Pierre ; Kavukcuoglu, Koray ; Chintala, Soumith ; LeCun, Yann: Pedestrian detection with unsupervised multi-stage feature learning. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, S. 3626–3633

[Sim u. a. 2002] Sim, Terence ; Baker, Simon ; Bsat, Maan: The CMU pose, illumination, and expression (PIE) database. In: *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition* IEEE (Veranst.), 2002, S. 53–58

[Simonyan und Zisserman 2014] Simonyan, Karen ; Zisserman, Andrew: Very deep convolutional networks for large-scale image recognition. In: *arXiv preprint arXiv:1409.1556* (2014)

[Sitzmann u. a. 2020] Sitzmann, Vincent ; Martel, Julien N. ; Bergman, Alexander W. ; Lindell, David B. ; Wetzstein, Gordon: Implicit Neural Representations with Periodic Activation Functions. In: *arXiv preprint arXiv:2006.09661* (2020)

[Sivic und Zisserman 2003] Sivic, Josef ; Zisserman, Andrew: Video Google: A text retrieval approach to object matching in videos. In: *Computer Vision, IEEE International Conference on* Bd. 3 IEEE Computer Society (Veranst.), 2003, S. 1470–1470

[Sutskever u. a. 2013] Sutskever, Ilya ; Martens, James ; Dahl, George ; Hinton, Geoffrey: On the importance of initialization and momentum in deep learning. In: *International conference on machine learning* PMLR (Veranst.), 2013, S. 1139–1147

[Szegedy u. a. 2015] Szegedy, Christian ; Liu, Wei ; Jia, Yangqing ; Sermanet, Pierre ; Reed, Scott ; Anguelov, Dragomir ; Erhan, Dumitru ; Vanhoucke, Vincent ; Rabinovich, Andrew: Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), S. 1–9

[Tang u. a. 2020] Tang, Luming ; Wertheimer, Davis ; Hariharan, Bharath: Revisiting Pose-Normalization for Fine-Grained Few-Shot Recognition. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, S. 14352–14361

[Tang 2013] Tang, Yichuan: Deep learning using linear support vector machines. In: *arXiv preprint arXiv:1306.0239* (2013)

[Team u. a. 2016] Team, The Theano D. ; Al-Rfou, Rami ; Alain, Guillaume ; Almahairi, Amjad ; Angermueller, Christof ; Bahdanau, Dzmitry ; Ballas, Nicolas ; Bastien, Frédéric ; Bayer, Justin ; Belikov, Anatoly u. a.: Theano: A Python framework for fast computation of mathematical expressions. In: *arXiv preprint arXiv:1605.02688* (2016)

[Tenenbaum und Freeman 2000] Tenenbaum, Joshua B. ; Freeman, William T.: Separating style and content with bilinear models. In: *Neural computation* 12 (2000), Nr. 6, S. 1247–1283

[Tsagkatakis u. a. 2017] Tsagkatakis, Grigorios ; Jaber, Mustafa ; Tsakalides, Panagiotis: Goal!! event detection in sports video. In: *Electronic Imaging* 2017 (2017), Nr. 16, S. 15–20

[Vapnik Vladimir 1995] Vapnik Vladimir, N_: *The nature of statistical learning theory.* 1995

[Vasilache u. a. 2014] Vasilache, Nicolas ; Johnson, Jeff ; Mathieu, Michael ; Chintala, Soumith ; Piantino, Serkan ; LeCun, Yann: Fast convolutional nets with fbfft: A GPU performance evaluation. In: *arXiv preprint arXiv:1412.7580* (2014)

[Wan u. a. 2013] Wan, Li ; Zeiler, Matthew ; Zhang, Sixin ; Le Cun, Yann ; Fergus, Rob: Regularization of neural networks using dropconnect. In: *International conference on machine learning* PMLR (Veranst.), 2013, S. 1058–1066

[Wang u. a. 2019] Wang, Chen ; Yang, Jianfei ; Xie, Lihua ; Yuan, Junsong: Kervolutional Neural Networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, S. 31–40

[Wang u. a. 2016] Wang, Yequan ; Huang, Minlie ; Zhu, Xiaoyan ; Zhao, Li: Attention-based LSTM for aspect-level sentiment classification. In: *Proceedings of the 2016 conference on empirical methods in natural language processing*, 2016, S. 606–615

[Wang u. a. 2020] Wang, Zhihui ; Wang, Shijie ; Yang, Shuhui ; Li, Haojie ; Li, Jianjun ; Li, Zezhou: Weakly Supervised Fine-Grained Image Classification via Guassian Mixture Model Oriented Discriminative Learning. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, S. 9749–9758

[Wei u. a. 2018] Wei, Xing ; Zhang, Yue ; Gong, Yihong ; Zhang, Jiawei ; Zheng, Nanning: Grassmann pooling as compact homogeneous bilinear pooling for fine-grained visual classification. In: *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, S. 355–370

[Welinder u. a. 2010] Welinder, P. ; Branson, S. ; Mita, T. ; Wah, C. ; Schroff, F. ; Belongie, S. ; Perona, P.: Caltech-UCSD Birds 200 / California Institute of Technology. 2010 (CNS-TR-2010-001). – Forschungsbericht

[Wen u. a. 2016] Wen, Yandong ; Zhang, Kaipeng ; Li, Zhifeng ; Qiao, Yu: A discriminative feature learning approach for deep face recognition. In: *European conference on computer vision* Springer (Veranst.), 2016, S. 499–515

[Xiao u. a. 2017] Xiao, Han ; Rasul, Kashif ; Vollgraf, Roland: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. In: *arXiv preprint arXiv:1708.07747* (2017)

[Xie u. a. 2020] Xie, Zeke ; Sato, Issei ; Sugiyama, Masashi: Stable weight decay regularization. (2020)

[Yadan u. a. 2013] Yadan, Omry ; Adams, Keith ; Taigman, Yaniv ; Ranzato, Marc'Aurelio: Multi-gpu training of convnets. In: *arXiv preprint arXiv:1312.5853* (2013)

[Yan u. a. 2017] Yan, Yilin ; Chen, Min ; Sadiq, Saad ; Shyu, Mei-Ling: Efficient imbalanced multimedia concept retrieval by deep learning on spark clusters. In: *International Journal of Multimedia Data Engineering and Management (IJMDEM)* 8 (2017), Nr. 1, S. 1–20

[Yan u. a. 2015] Yan, Yilin ; Chen, Min ; Shyu, Mei-Ling ; Chen, Shu-Ching: Deep learning for imbalanced multimedia data classification. In: *2015 IEEE international symposium on multimedia (ISM)* IEEE (Veranst.), 2015, S. 483–488

[Yin u. a. 2006] Yin, Lijun ; Wei, Xiaozhou ; Sun, Yi ; Wang, Jun ; Rosato, Matthew J.: A 3D facial expression database for facial behavior research. In: *7th international conference on automatic face and gesture recognition (FGR06)* IEEE (Veranst.), 2006, S. 211–216

[Yu u. a. 2018] Yu, Chaojian ; Zhao, Xinyi ; Zheng, Qi ; Zhang, Peng ; You, Xinge: Hierarchical bilinear pooling for fine-grained visual recognition. In: *Proceedings of the European conference on computer vision (ECCV)*, 2018, S. 574–589

[Yu u. a. 2014a] Yu, Dingjun ; Wang, Hanli ; Chen, Peiqiu ; Wei, Zhihua: Mixed pooling for convolutional neural networks. In: *International conference on rough sets and knowledge technology* Springer (Veranst.), 2014, S. 364–375

[Yu u. a. 2014b] Yu, Dong ; Eversole, Adam ; Seltzer, Michael L. ; Yao, Kaisheng ; Guenter, Brian ; Kuchaiev, Oleksii ; Seide, Frank ; Wang, Huaming ; Droppo, Jasha ; Huang, Zhiheng u. a.: An introduction to computational networks and the computational network toolkit (invited talk). In: *INTERSPEECH*, 2014

[Yu u. a. 2017] Yu, Zhou ; Yu, Jun ; Fan, Jianping ; Tao, Dacheng: Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In: *Proceedings of the IEEE international conference on computer vision*, 2017, S. 1821–1830

[Zeiler und Fergus 2013] Zeiler, Matthew D. ; Fergus, Rob: Stochastic pooling for regularization of deep convolutional neural networks. In: *arXiv preprint arXiv:1301.3557* (2013)

[Zhang u. a. 2018a] Zhang, Guodong ; Wang, Chaoqi ; Xu, Bowen ; Grosse, Roger: Three mechanisms of weight decay regularization. In: *arXiv preprint arXiv:1810.12281* (2018)

[Zhang und Wang 2017] Zhang, Xueliang ; Wang, DeLiang: Deep learning based binaural speech separation in reverberant environments. In: *IEEE/ACM transactions on audio, speech, and language processing* 25 (2017), Nr. 5, S. 1075–1084

[Zhang u. a. 2019a] Zhang, Yan ; Tang, Siyu ; Muandet, Krikamol ; Jarvers, Christian ; Neumann, Heiko: Local temporal bilinear pooling for fine-grained action parsing. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, S. 12005–12015

[Zhang u. a. 2019b] Zhang, Yuanyuan ; Wang, Zi-Rui ; Du, Jun: Deep fusion: An attention guided factorized bilinear pooling for audio-video emotion recognition. In: *2019 International Joint Conference on Neural Networks (IJCNN)* IEEE (Veranst.), 2019, S. 1–8

[Zhang u. a. 2018b] Zhang, Zhanpeng ; Luo, Ping ; Loy, Chen C. ; Tang, Xiaoou: From facial expression recognition to interpersonal relation prediction. In: *International Journal of Computer Vision, Springer* 126 (2018), Nr. 5, S. 550–569

[Zhao u. a. 2011] Zhao, Guoying ; Huang, Xiaohua ; Taini, Matti ; Li, Stan Z. ; PietikäInen, Matti: Facial expression recognition from near-infrared videos. In: *Image and Vision Computing* 29 (2011), Nr. 9, S. 607–619

[Zhou u. a. 2018] Zhou, Feng ; Kong, Shu ; Fowlkes, Charless ; Chen, Tao ; Lei, Baiying: Fine-grained facial expression analysis using dimensional emotion model. In: *arXiv preprint arXiv:1805.01024* (2018)

[Zhuang u. a. 2020] Zhuang, Peiqin ; Wang, Yali ; Qiao, Yu: Learning Attentive Pairwise Interaction for Fine-Grained Classification. In: *AAAI*, 2020, S. 13130–13137

[Zou u. a. 2019] Zou, Xiaowu ; Wang, Zidong ; Li, Qi ; Sheng, Weiguo: Integration of residual network and convolutional neural network along with various activation functions and global pooling for time series classification. In: *Neurocomputing* (2019)

[Zoumpourlis u. a. 2017] Zoumpourlis, Georgios ; Doumanoglou, Alexandros ; Vretos, Nicholas ; Daras, Petros: Non-linear convolution filters for CNN-based learning. In: *Proceedings of the IEEE International Conference on Computer Vision*, 2017, S. 4761–4769