OBJECT RECOGNITION WITH COMPETITIVE CONVOLUTIONAL NEURAL NETWORKS

TUĞBA ERKOÇ

IŞIK UNIVERSITY
JUNE, 2023

OBJECT RECOGNITION WITH COMPETITIVE CONVOLUTIONAL NEURAL NETWORKS

TUĞBA ERKOÇ
Işık University, Graduate School of Higher Education
Computer Engineering Doctoral Program, 2023

This thesis is submitted to the Graduate School of Higher Education at Işık University for the degree of Doctor of Philosophy (PhD) in Computer Engineering

IŞIK UNIVERSITY
JUNE, 2023

IŞIK UNIVERSITY
GRADUATE SCHOOL OF HIGHER EDUCATION
COMPUTER ENGINEERING DOCTORAL PROGRAM

OBJECT RECOGNITION WITH COMPETITIVE CONVOLUTIONAL NEURAL NETWORKS

TUĞBA ERKOÇ

APPROVED BY:

Prof. Dr. M. Taner Eskil (Thesis Supervisor)	Işık University
Assoc. Prof. Erkin Dinçmen	Işık University
Assist. Prof. Emine Ekin	Işık University
Prof. Dr. Bahadır K. Güntürk	İstanbul Medipol University
Assoc. Prof. Yusuf Yaslan	İstanbul Technical University

APPROVAL DATE: 12/06/2023

* "6689 Sayılı Kişisel Verilerin Korunması Kanunu Hükümlerine Göre Çevrimiçi Yayın Dosyasında Bulunan Kişisel Veriler Ve Islak İmzalar Silinmiştir."

OBJECT RECOGNITION WITH COMPETITIVE CONVOLUTIONAL NEURAL NETWORKS

ABSTRACT

In recent years, Artificial Intelligence (AI) has achieved impressive results, often surpassing human capabilities in tasks involving language comprehension and visual recognition. Among these, computer vision has experienced remarkable progress, largely due to the introduction of Convolutional Neural Networks (CNNs). CNNs are inspired by the hierarchical structure of the visual cortex and are designed to detect patterns, objects, and complex relationships within visual data. One key advantage is their ability to learn directly from pixel values without the need for domain expertise, which has contributed to their popularity. These networks are trained using supervised backpropagation, a process that calculates gradients of the network's parameters (weights and biases) with respect to the loss function. While backpropagation enables impressive performance with CNNs, it also presents certain drawbacks. One such drawback is the requirement for large amounts of labeled data. When the available data samples are limited, the gradients estimated from this limited information may not accurately capture the overall data behavior, leading to suboptimal parameter updates. However, obtaining a sufficient quantity of labeled data poses a challenge. Another drawback is the requirement of careful configuration of hyperparameters, including the number of neurons, learning rate, and network architecture. Finding optimal values for these hyperparameters can be a time-consuming process. Furthermore, as the complexity of the task increases, the network architecture becomes deeper and more complex. To effectively train the shallow layers of the network, one must increase the number of epochs and experiment with solutions to prevent vanishing gradients. Complex problems often require a greater number of epochs to learn the intricate patterns and features present in the data. It's important to note that while CNNs aim to mimic the structure of the visual cortex, the brain's learning mechanism does not necessarily involve back-propagation. Although CNNs incorporate the layered architecture of the visual cortex, the reliance on backpropagation introduces an artificial learning procedure that may not align with the brain's actual learning process.

Therefore, it is crucial to explore alternative learning paradigms that do not rely on backpropagation.

In this dissertation study, a unique approach to unsupervised training for CNNs is explored, setting it apart from previous research. Unlike other unsupervised methods, the proposed approach eliminates the reliance on backpropagation for training the filters. Instead, we introduce a filter extraction algorithm capable of extracting dataset features by processing images only once, without requiring data labels or backward error updates. This approach operates on individual convolutional layers, gradually constructing them by discovering filters. To evaluate the effectiveness of this backpropagation-free algorithm, we design four distinct CNN architectures and conduct experiments. The results demonstrate the promising performance of training without backpropagation, achieving impressive classification accuracies on different datasets. Notably, these outcomes are attained using a single network setup without any data augmentation. Additionally, our study reveals that the proposed algorithm eliminates the need to predefine the number of filters per convolutional layer, as the algorithm automatically determines this value. Furthermore, we demonstrate that filter initialization from a random distribution is unnecessary when backpropagation is not employed during training.

Keywords: Convolutional Neural Networks, Unsupervised Learning, Feature Extraction

REKABETÇİ EVRİŞİMLİ SİNİR AĞLARI İLE NESNE TANIMA

ÖZET

Son yıllarda Yapay Zeka (YZ) dili anlama ve görsel tanımayı içeren görevlerde genellikle insan yeteneklerini geride bırakarak etkileyici sonuçlar elde etti. Bunların arasında, bilgisayarla görme, büyük ölçüde Evrişimli Sinir Ağlarının (ESA) ortaya çıkması ile dikkate değer bir ilerleme kaydetti. ESAlar, görsel korteksin hiyerarşik yapısından ilham alarak görsel verilerdeki kalıpları, nesneleri ve karmaşık ilişkileri tespit etmek icin tasarlanmıştır. En önemli avantajlarından biri, popülerliklerine katkıda bulunan, bir uzmana ihtiya. Duymadan doğrudan piksel değerlerinden öğrenme yetenekleridir. Bu ağlar, kayıp fonksiyonuna göre ağ parametrelerinin (ağrılıklar ve eğilimler) gradyanlarını hesaplayan denetimli geri yayılım ile eğitilir. Geri yayılım, ESAlarda etkileyici bir performans sağlarken, bazı dezavantajlar da getirir. Bu dezavantajlardan biri büyük miktarlarda etiketlenmiş veri gereksinimidir. Mevcut veri örnekleri sınırlı olduğunda, bu sınırlı bilgiden hesaplanan gradyanlar , genel veri davranışını doğru bir şekilde yakalayamayabilir ve bu da yetersiz parameter güncellemelerine yol açar. Bununla birlikte, yeterli miktarda etiketlenmiş veri elde etmek bir zorluk teşkil etmektedir. Diğer nir dezavantaj nöron sayısı, öğrenme hızı ve ağ mimarisi dahil olmak üzere hiperparametrelerin dikkatli bir şekilde yapılandırılması gerekliliğidir. Bu hiperparametreler için en uygun değerleri bulmak zaman alıcı bir süreç olabilir. Ayrıca, görevin karmaşıklığı arttıkça ağ mimarisi daha derin ve karmaşık bir hale gelir. Ağın sığ katmanlarını etkili bir şekilde eğitmek için, epok sayısı artırılmalı ve kaybolan gradyanları önlemek için çözümler üretilmelidir. Karmaşık problemler, verilerde bulunan karmaşık kalıpları ve özellikleri öğrenmek için genellikle daha fazla sayıda epok gerektirir. ESAlar görsel korteksin yapısını taklit etmeyi amaçlasa da, beynin öğrenme mekanizmasının mutlaka geri yayılımı içermediğini not etmek önemlidir. ESAlar görsel korteksin katmanlı mimarisini içermelerine rağmen, geri yayılıma dayanan öğrenme, beynin gerçek öğrenme süreciyle uyumlu olmayabilen yapay bir öğrenme prosedürü sunar. Bu nedenle, geri yayılıma dayanmayan alternatif öğrenme paradigmalarını keşfetmek önem teşkil etmektedir.

Bu tez çalışmasında, önceki araştırmalardan farklı olarak ESAlar için denetimsiz eğitime yönelik benzersiz bir yaklaşım araştırılmaktadır. Önerilen yaklaşım diğer denetimsiz yöntemlerin aksine, filtrelerin eğitimi için geri yayılmaya olan bağlılığı kaldırır. Geri yayılım ile öğrenme yerine, veri etiketleri veya geriye dönük hata güncellemeleri gerektirmeden görüntüleri yalnızca bir kez işleyerek veri kümesi özelliklerini çıkarabilen bir filtre çıkarma algoritması sunuyoruz. Bu yaklaşım bireysel Evrişimli katmanlar üzerinde çalışır ve filtreleri eğitim örnekleri üzerinden keşfederek evrişim katmanının filtrelerini kademeli olarak oluşturur. Bu geri yayılımsız algoritmanın etkinliğini değerlendirmek için dört farklı ESA mimarisi tasarladık ve deneyler yaptık. Sonuçlar, farklı veri kümelerinde etkileyici sınıflandırma doğrulukları elde ederek, geri yayılım olmadan eğitimin mümkün olabileceğini göstermektedir. Özellikle, bu sonuçlara herhangi bir veri arttırımı olmadan vet ek bir ağ kullanılarak ulaşılmıştır. Ek olarak, çalışmamızda önerilen algoritma, evrişim katmanı başına filtre sayısını önceden belirleme ihtiyacını ortadan kaldırmaktadır çünkü algoritmamız bu değeri otomatik olarak belirlemektedir. Ayrıca, eğitim sırasında geri yayılım kullanılmadığından rastgele bir dağılımdan filtrelere ilkdeğer verilmesinin gereksiz olduğunu da bu çalışma ile gösterdik.

Anahtar Kelimeler: Evrişimli Sinir Ağları, Denetimsiz Öğrenme, Özellik Çıkarma

ACKNOWLEDGEMENTS

I am deeply grateful to the individuals who have made significant contributions to the completion of this dissertation: First and foremost, I would like to express my immense gratitude to Prof. Dr. M. Taner Eskil, my supervisor, for their guidance and support throughout this research journey. Their invaluable insights and constructive feedback have played a pivotal role in shaping this dissertation. I extend my heartfelt appreciation to Assoc. Prof. Erkin Dinçmen, Assist. Prof. Emine Ekin, Prof. Dr. Hazım K. Ekenel and Prof. Dr. Ercan Solak for generously dedicating their time and offering thoughtful suggestions. Their scholarly guidance has significantly enhanced the quality of this research. Additionally, I am sincerely thankful to my thesis jury members Prof. Dr. Bahadır K. Güntürk and Assoc. Prof. Yusuf Yaslan for their interest and support. I am grateful to my colleagues and friends who have provided moral support and offered encouragement throughout this dissertation. Their friendship has made this challenging journey more enjoyable. I would like to express special thanks to my family for their support, love, and understanding. Their encouragement and belief in my abilities have served as the driving force behind my accomplishments. In conclusion, I would like to acknowledge the contributions of all those who have played a part in shaping this dissertation. Your support and encouragement have been indispensable, and I am truly grateful for your involvement.

Tuğba ERKOÇ

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant 118E293.

This study is dedicated to my family.

TABLE OF CONTENTS

LIST OF TABLES

TABLE OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

AE:	Auto Encoder
AI:	Artificial Intelligence
AiS:	Add if Silent
ANN:	Artificial Neural Network
CAE:	Convolutional Auto Encoder
CNN:	Convolutional Neural Network
CCNN:	Competitive Convolutional Neural Network
CoG:	Center of Gravity
DEC:	Deep Embedded Clustering
FN:	False Negative
FP:	False Positive
GAN:	Generative Adversarial Network
GMM:	Gaussian Mixture Model
GPU:	Graphics Processing Unit
GS:	Grid Search
KL:	Kullback-Leibler
MLP:	Multi-Layer Perceptron
MNIST:	Modified NIST
NIST:	National Institute of Standards and Technology
PReLU:	Parametrized Rectified Linear Unit
ReLU:	Rectified Linear Unit
RS:	Random Search
SAE:	Stacked Auto Encoder
SGD:	Stochastic Gradient Descent

SMBO:	Sequential Model Based Optimization
TN:	True Negative
TP:	True Positive
WTA:	Winner Take All

CHAPTER 1

1. INTRODUCTION

In recent years, CNNs have emerged as the predominant approach for image classification tasks. Even though it gained popularity in recent years, the history of CNNs dates back to 1959. Hubel and Weisel (Hubel, Wiesel, 1959) discovered the presence of alternating set of neurons in the visual cortex of a cat that fire when an oriented edge stimulus is presented. Fukushima (Fukushima, 1980) proposed the first CNN architecture that was based on the findings of Hubel and Weisel (Hubel, Wiesel, 1959). Unlike the majority of modern CNN implementations, Neocognitron deviated from using gradient-based algorithms for training. Later in 1998 (LeCun, Bottou, Bengio, Haffner, 1998), LeCun et. al proposed a CNN architecture that is similar to Neocognitron but trained with the backpropagation algorithm, which has become the standard for CNNs today. Many different CNN architectures (Krizhevsky, Sutskever, Geoffrey E., 2012; Simonyan Zisserman, 2014; Zeiler Fergus, 2014) were proposed since LeNet. Most of the architectures proposed in the literature are similar; they consist of several convolutional, pooling and fully connected layers and they are trained through gradient-based backpropagation.

A CNN can be considered as a network that is composed of a feature extractor and a classifier. The feature extractor implements convolutional and pooling layers while the classifier part consists of fully connected and softmax layers. Each of these layers should be carefully preconfigured to achieve reasonable results with training. The hyperparameters that have to be initialized only for the convolutional layer include the number of filters and filter size in each layer, stride, activation function and the learning rate. There is unfortunately no rule of thumb for selecting the right value for the hyperparameters of a network. Each application domain requires specific settings to attain the optimal architecture, which are generally obtained through empirical hyperparameter optimization methods.

In traditional CNN implementations, specification of the network architecture is followed by initialization of the weights of the entire network from a specific distribution, often without any regard to the input domain (Glorot, Bengio, 2010; He, Zhang, Ren, Sun, 2015). The expectation of a researcher in this stage is to coincidentally initialize a set of neurons at such points in the search space that a gradient based walk will progress their weights to some optima. Since it is coincidental, the researcher has no other choice than generating a superfluous number of neurons at each layer to be able to span the search space. Moreover, the experiments will produce varying results, some of which are subpar due to circumstantial starting points in the search space. Thus, decision on the number of neurons/features and the initialization of them in the convolutional layers of CNN are problems that needs a less tedious process than hyperparameter optimization methods.

The convolutional layers of CNN architectures serve for one purpose only; extraction of good features to be fed to a classifier. In visual analysis of images, these features are cues that involve edges, corners and patterns, all visually observable and meaningful. These visual cues are building blocks of all objects which are to be assembled in incremental complexity from shallow to deep layers of CNN. In the past, these features were handcrafted to reach better classification results. However, in current CNN architecture, the features are initialized with a random distribution as mentioned in the previous paragraph. Then, they are trained through gradient based backpropagation algorithm to some convergence point which is generally reached after a couple of hundreds of epochs. Moreover, gradient based approaches lose a leverage of great significance by random initialization of weights and stochastic search during training, since visual cues in the training set are overlooked. If the random initialization of the features in convolutional layers could be eliminated, training the features with a gradient based training algorithm could be discarded. This would allow less time spent on training the overall CNN model while obtaining domain specific visual cues.

The utilization of backpropagation for training of the CNN introduces the credit assignment problem. In neural networks, we do not know which neuron made the correct/incorrect decision to reach the current classification result. Hence, we do not know which neuron’s weight should be updated with the error signal. Our approach to creation of a deep convolutional network and its training for feature extraction is based purely on observations, hence it is not vulnerable to the credit assignment problem like traditional CNNs that are trained by backpropagation. As aforementioned, in traditional CNN implementations the network is built with a number of neurons per layer, its weights are initialized from a random distribution and the entire network is trained through backpropagation. The backpropagation algorithm updates the weights of the network incrementally and depending on the partial derivative of the error on the weights. Since backpropagation is a gradient based approach, it takes multiple epochs for the neurons to converge to meaningful filters. This is inherent of the approach since all that is observable is the gradient of the hyperplane on the search space towards the direction that minimizes the error. We are forced to take small steps since a large step might lead to divergence, making training impossible.

Backpropagation based training is also vulnerable to the problem of exploding or vanishing gradients (Hochreiter, 1991) which impacts the learning process in deep networks. It is shown that when the gradient of the error is small in the last layer, it diminishes to infinitesimal values until the backpropagation reaches to the first layer. Since the value of the gradient gets smaller as backpropagation approaches to the first layer, training the shallow layers of deep networks through pure backpropagation is difficult in deep networks. Combined with the weakened error signal, deep architectures challenge us with an exemplary version of the credit assignment problem. We cannot correctly distribute and backpropagate the error to shallow layers, hence producing subpar features at early and simple stages of image processing. Thus, the quality of the patterns learned in the deeper layers is hampered and the classification performance deteriorates. Skip connections in Residual networks (He et al., 2015) were proposed as a work around for the gradual vanishing of the gradients which adds more complexity to the CNN architectures.

In this work, we introduce Competitive Convolutional Neural Network (CCNN); an unsupervised deep learning architecture and training algorithm that extracts a sufficient number of features that span the input domain. We neither predefine the number of neurons nor initialize them with random values from a distribution as in conventional CNNs. The filters in our model are discovered within a single epoch using an unsupervised approach that utilizes competitive learning and a filter discovery rule. This new approach does not add more complexity to the traditional CNN models since we do not propose new layer or connection types.

1.1. Contributions

Our contributions to the literature with this work are 3-fold. First, we propose an architecture and an unsupervised training algorithm that eliminate the need for setting the number of filters in layers before the training commences. The proposed method starts with empty layers and builds them gradually to discover the sufficient number of features to represent the complexity of the input domain, in increasing levels of abstraction from shallow to deep layers.

Our second contribution is to eliminate pre-training initialization of network weights. This follows from the fact that in our approach training starts with an empty network. We create and initialize new neurons as needed; a decision made through the similarity of the observation (input at a specific layer) to the features represented by the existing neurons. The new neuron that is to represent a new feature is initialized to resemble the input sample. Thus, creation and initialization of a new neuron is not random anymore, but rather dependent on the observations, i.e., the input domain.

Our third contribution in this work is an unsupervised training algorithm that is a robust, effective and fast feature extractor. This algorithm searches for clusters in input, finding simple to complex features layer by layer. It is not prone to credit assignment problem as it is purely unsupervised. It is very fast, often converging in one epoch per layer. This new training algorithm allows us to completely eliminate the backpropagation training for the convolutional layers of the model.

1.2. Organization of This Thesis

The motivation and the aim of this thesis is given in the previous sections. The remainder of this dissertation is structured as follows:

- ***Chapter 2 - Convolutional Neural Networks:*** This chapter will introduce the reader to the concepts of Convolutional Neural Networks. It will provide necessary information on the topic, so the reader can easily follow the approach in this work.

- ***Chapter 3 - Literature Survey:*** In this chapter, a brief literature survey on the topics covered in this thesis will be presented.
- ***Chapter 4 - Approach:*** The proposed CNN architecture and the unsupervised training algorithm for the discovery of the features to be used in the convolutional layers will be discussed in detail in this chapter.
- ***Chapter 5 - Experiments:*** In this chapter, experiment setup, CCNN models, datasets and the performance metrics used in the experiments are described. Then, how experiments are conducted will be discussed.
- ***Chapter 6 - Results:*** The results of the experiments detailed in the previous chapter is discussed in this chapter.
- ***Chapter 7 - Discussion:*** In this chapter, the results of the new approach are compared with the previous studies.
- ***Chapter 8 - Conclusion:*** In the last chapter of the thesis, a conclusion will be shared with possible new directions this research may lead to.

CHAPTER 2

2. CONVOLUTIONAL NEURAL NETWORKS

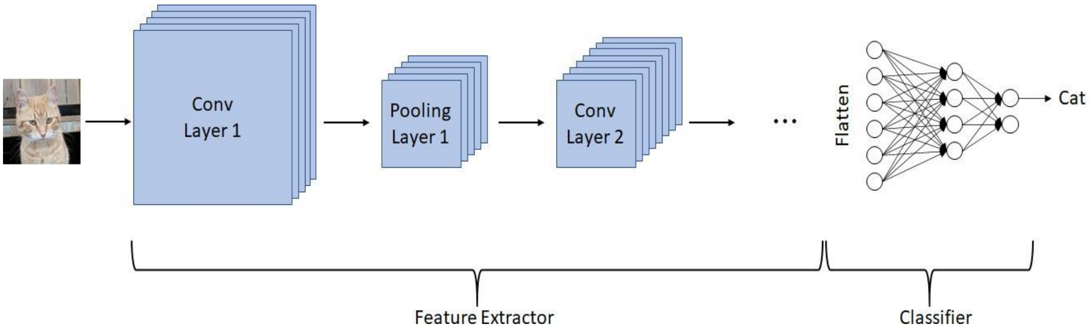


Figure 2.1 A typical Convolutional Neural Network

CNN is a type of multilayer feed-forward Artificial Neural Network (ANN) which is based on the structure of animal visual cortex (Hubel, Wiesel, 1959). Due to its hierarchical layered architecture inspired by the visual cortex of animals, CNNs are well-suited for visual classification tasks. In basic terms, a CNN consists of several different kinds of layers that learn features from training images hierarchically and predicts the class of a given test image based on the learned features in the images. Convolution and Pooling layers are the key layers in learning the features from training images while Fully Connected layers act as classifiers (Figure2.1).

2.1 Convolutional Neural Network Architecture

2.1.1 Convolutional Layer

The purpose of convolutional layer is detecting high level features from given visual data so that the classifier can classify that data into specific classes according to the particular features detected in the given data. Prior to the usage of CNNs, the features had to be hand crafted by the field experts who has domain expertise on that particular task. The hand crafting of features is a daunting task since every variation in illumination, position, scale and variations in same class of objects must be considered during the feature creation process. The solution to this is creating a feature detection pipeline which resembles the animal visual cortex that could start by learning basic features like lines and edges and hierarchically building more complex features (Figure 2.2) to classify images. To be able to work with this kind of pipeline, we need to be able to extract local features from the given data. Convolutional layers perform this task with filters. However, for each possible feature in the given data, we need to define a filter of specific size. This specific size of the filter ensures that the feature that we are trying to extract is within some local area which is called receptive field in the given data. Conventionally, the filters are defined in sizes of 3 × 3, 5 × 5, 7 × 7 or 11 × 11 depending on the visual task at hand.

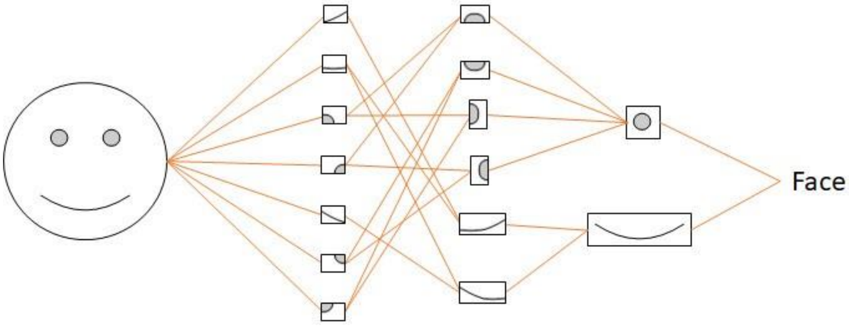


Figure 2.2 CNNs can recognize high level concepts like face by hierarchically building feature detectors starting from basic edge like shapes to complex features like eyes.

Since we need one filter per feature, the number of filters - which we do not know beforehand - in a convolutional layer must also be defined during the construction of the convolutional layer. The number of filters on a convolutional layer is a hyperparameter. This hyperparameter’s value needs to be selected with hyperparameter optimization techniques to achieve good classification performance. One of the easiest such hyperparameter optimization method is Grid Search (GS), where a subset of hyperparameter space is searched (Bengio, 2012) for the optimal performance. However, it is not a suitable optimization technique since the number of hyperparameters in the CNNs are too large which renders GS a computationally expensive optimization method (Kaneko, Funatsu, 2015), and GS is often stuck at some local optima (Keerthi, Lin, 2003). For the optimization of hyperparameters, a combination of GS and Random Search (RS) is proposed by in works (Larochelle, Erhan, Courville, Bergstra, Bengio, 2007;Yann, Bottou, Orr, Müller, 1998; Hinton, 2010) whereas it is argued (Hutter, 2009) that Sequential Model Based Optimization(SMBO) more effectively finds the best solution than RS. It is shown that CNN hyperparameters can be effectively tuned (Snoek, Larochelle, Adams, 2012) with a Bayesian approach. Regardless of the method of optimization, hyperparameter values need to be optimized for achieving the optimal performance from CNN.

The last step in defining feature extracting filters is setting the weights of said filters. However, in CNNs we do not handcraft the features, so the filters should be initialized with some values. Initializing the filters in convolutional layers is an important task, since it affects the ability to learn features from the training data. If the values of the weights are initialized too small or too big, the learning from the training data will be either too slow or will not happen at all. This situation is called vanishing/exploding gradients since learning depends on the backpropagation of errors which is calculated based on these weight values. Thus, just randomly initializing the filter weights is not enough. Initializing the weights of the filters requires some initialization technique that allows CNN to learn from the data in an acceptable time. The commonly used weight initialization techniques are Gaussian initialization where the weights are initialized from a zero-mean Gaussian distribution with a standard deviation of 0.01 (Krizhevsky et al., 2012), Glorot (Glorot, Bengio, 2010) initialization technique in which the weights are initialized based on the incoming and outgoing connection counts and He(He et al., 2015) initialization method which is a variant of Glorot that allows very deep networks to be built. Choosing one of those initialization methods depends on which activation function is used in the convolutional layer.

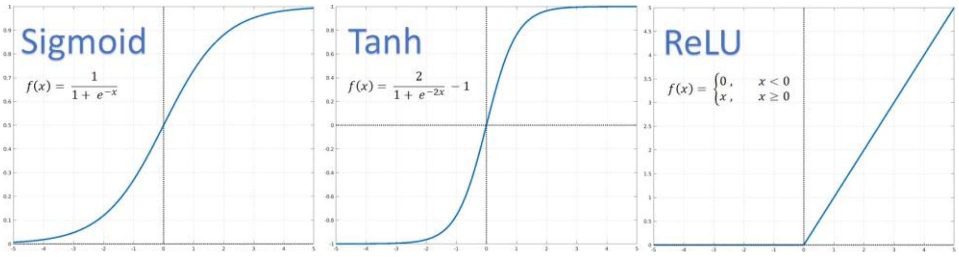


Figure 2.3 Sigmoid, hyperbolic tangent and ReLU activation function curves shown. ReLU is most popular activation function in CNNs.

2.1.1.1 Activation Function

Activation functions are used to introduce non-linearity to the neural network models so that the model can generalize well by deciding whether its related neuron will fire or not. It would not be possible to correctly classify objects that belong to the same class that have intra-class variations with a linear model. Thus, non-linearity is introduced to the ANNs through activation functions like sigmoid, hyperbolic tangent (tanh) and Rectified Linear Units (ReLU) (Nair, Hinton, 2010) which add generalization capability to ANNs. Instead of sigmoid or hyperbolic tangent, ReLU and its variants like PReLU or Leaky ReLU is utilized in deep architectures. The reason behind this selection is based on the vanishing gradients problem. With the increasing depth of the network in CNNs, very small gradients are obtained from both sigmoid and tanh functions in backward pass of the training which slows down the learning until a point where learning becomes impossible. They also get saturated on the both positive and negative sides and the gradients become very close to zero for very small or big weight values which again affects the weight updates in the backward pass of the training phase. ReLU can be described as a piece-wise function which acts as a linear function for positive inputs whereas it acts non-linearly for negative inputs by setting the negative values to zero. Computation cost of the ReLU is small compared to sigmoid or tanh since these functions involve exponentiation operations. The lack of exponentiation in ReLU reduces the cost of complex derivation operation which accelerates the training. This allows the addition of more layers to the network since the computation cost freed up from the derivation equations can be invested in increasing the capacity of the network. Compared to sigmoid and tanh, ReLU do not saturate except for the negative values. This leads to the downside of using ReLU which is called dying ReLU problem. When a large gradient flow updates the weight of a neuron in a way that it starts generating negative responses to stimuli, we encounter dying ReLU problem. There is no way to recover from this situation when ReLU is the chosen activation function and some neurons might end up dead (i.e., no contribution to learning). This problem is addressed with the ReLU variants PReLU (He et al., 2015) and Leaky ReLU(Maas, Hannun, Ng, 2013) where instead of zeroing out the negative response to the stimuli, both of these activation functions have a small slope to the curve in the negative side of the original ReLU function as can be seen in Figure 2.4. This small slope allows for a small gradient to flow for negative responses of the neurons instead of completely shutting them down with the cost of introducing another parameter into the calculations.

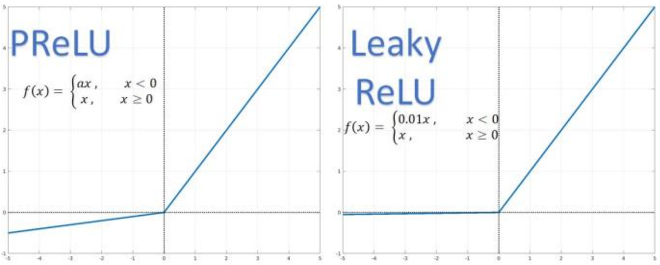


Figure 2.4 PReLU and Leaky ReLU activation function curves. PReLU and Leaky ReLU allows a small gradient for negative values whereas original ReLU strictly sets the negative values to zero.

2.1.1.2 Convolution Operation

The aim of convolution operation is filtering out specific patterns from the image which are crucial for identifying the object that is present in the image. Thus, the convolution operation takes an input image and a filter/kernel. The kernel is used to search for the pattern locally in the image by applying the convolution operation shown in Equation 2.1. The convolution operation is repeated for each kernel K defined in the convolutional layer for the same input image I.

$$K * I = \sum_{m} \sum_{n} I(m,n)K(i - mij - n) \qquad (2.1)$$

Starting from the top left of the image, the convolution operation is performed on the image I. The result is a single value which denotes the likelihood of the presence of that pattern at that specific location which the current kernel is seeking in the image. Then, the kernel is moved in the horizontal direction. The amount of movement is called stride and, it is a parameter that should be predetermined. If the stride is 1 pixel, the kernel is moved in the horizontal direction 1 step. When the horizontal locations are exhausted, the kernel is moved in the vertical direction according to the stride parameter. This is repeated until all possible locations on the given image is spanned by the kernel as seen in Figure 2.5.

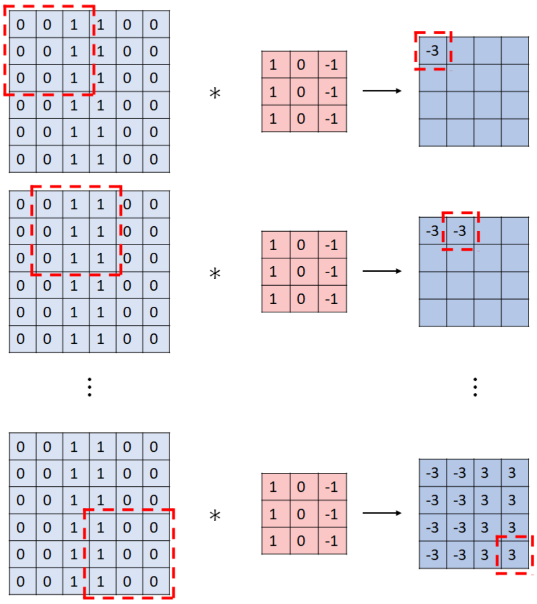
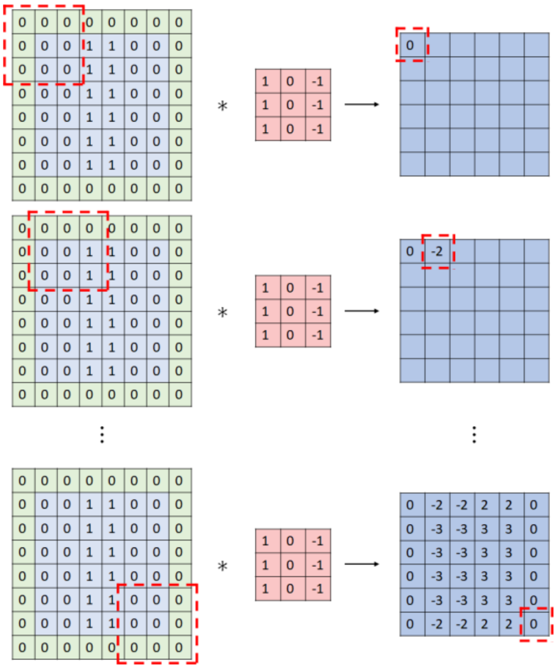


Figure 2.5 Visualization of convolution operation on with a 5 × 5 image and a 3 × 3 filter with a stride of 1 pixel.



$$\begin{bmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&0&0&0&0&0 \end{bmatrix} * \begin{bmatrix} 1&0&-1\\ 1&0&-1\\ 1&0&-1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0&&&&&\\ &&&&&\\ &&&&&\\ &&&&&\\ &&&&&\\ &&&&& \end{bmatrix}$$

$$\begin{bmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&0&0&0&0&0 \end{bmatrix} * \begin{bmatrix} 1&0&-1\\ 1&0&-1\\ 1&0&-1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0&-2&&&&\\ &&&&&\\ &&&&&\\ &&&&&\\ &&&&&\\ &&&&& \end{bmatrix}$$

$$\vdots \qquad\qquad\qquad\qquad \vdots$$

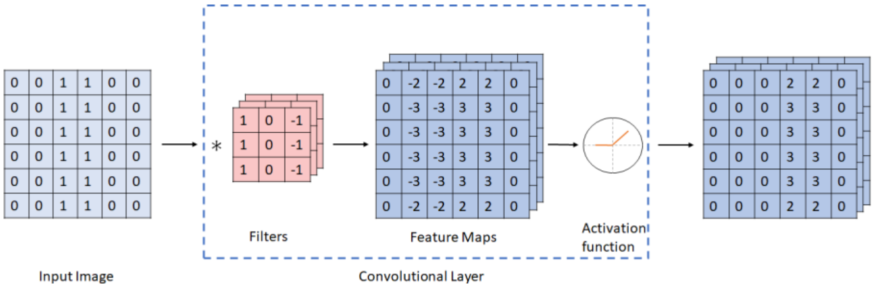
$$\begin{bmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&0&0&0&0&0 \end{bmatrix} * \begin{bmatrix} 1&0&-1\\ 1&0&-1\\ 1&0&-1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0&-2&-2&2&2&0\\ 0&-3&-3&3&3&0\\ 0&-3&-3&3&3&0\\ 0&-3&-3&3&3&0\\ 0&-3&-3&3&3&0\\ 0&-2&-2&2&2&0 \end{bmatrix}$$

Figure 2.6 Visualization of convolution operation on with a 5 × 5 image and a 3 × 3 filter with a stride of 1 pixel and zero padding of 1 pixel as per Equation 2.2. Green background is padding while the image data is shown with light blue background.

As it can be observed from Figure 2.5, applying convolution to the image changes its dimensions. If we apply another layer of convolution, the image will further shrink to 2 × 2 dimensions. The information at the borders of images is rapidly lost in this fashion. Instead of losing data in a rapid fashion, we would like to preserve data even after convolution operation. To be able to preserve as much as we can, padding is applied to the images before convolution operation. In CNNs, padding is generally just adding zeros around the image as if we are framing a portrait. The size of the padding is calculated with Equation 2.2 for stride of 1 pixel and filter size k. The dimensions of the output of the convolution operation can be calculated with Equation 2.3. Figure 2.6 shows convolution operation with the same image padded with 1 pixel of zeros on each side of the image.

$$P = \frac{k-1}{2} \tag{2.2}$$

$$O = (I_h - k + 1 + 2P, I_w - k + 1 + 2P) \tag{2.3}$$



0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0

1	0	-1
1	0	-1
1	0	-1

0	-2	-2	2	2	0
0	-3	-3	3	3	0
0	-3	-3	3	3	0
0	-3	-3	3	3	0
0	-3	-3	3	3	0
0	-2	-2	2	2	0

0	0	0	2	2	0
0	0	0	3	3	0
0	0	0	3	3	0
0	0	0	3	3	0
0	0	0	3	3	0
0	0	0	2	2	0

Figure 2.7 Convolution layer applies convolution operation to the input images. The feature maps are then introduced to non-linearity with activation function.

The output of convolution operation is another matrix that is called feature map or activation map since moving the filter over the image and calculating convolution maps all possible locations that this particular feature might be present (i.e., map of filter activation). One feature map per filter is generated for the same image I which means that if there are K filters in the convolutional layer, the number of feature maps generated for image I would be K. However, the actual output of the convolutional layer is not the raw feature maps. The chosen non-linearity/activation function is applied to these feature maps as seen in Figure 2.7. If the activation funciton is ReLU, the result is called rectified feature maps (Zeiler, Fergus, 2014).

2.1.2 Pooling Layer

Pooling operation is employed in CNNs to downsample the images so that the number of equations can be restricted to a manageable number. This dimension reduction also introduces spatial invariance to the CNNs. Another advantage of using pooling layer is that it allows the next layer to focus on a larger receptive field while maintaining the same filter size as the previous convolutional layer. This leads to detecting more complex features compared to the earlier layers.

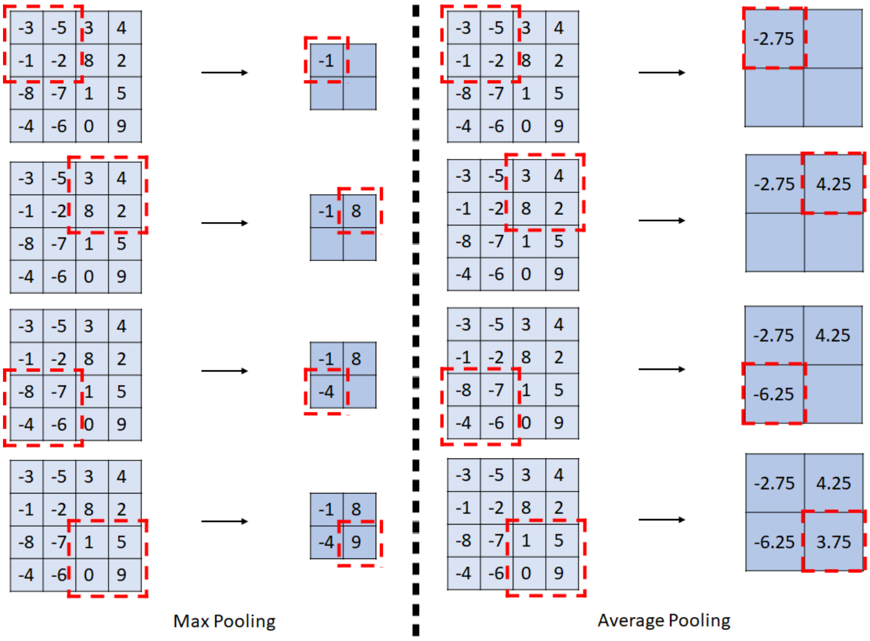


Figure 2.8 Visualization of max pooling operation on with a 4 × 4 image with 2 × 2 window size and strides of 2.

Similar to defining a filter in the convolutional layer, a window size is defined for pooling. This window size is the local data that will be pooled down to a single value. Pooling window starts from the top left of the image as in the convolution operation and pooling is applied to the pixels in that region of the image. After the pooling is applied, the pooling window is shifted in the horizontal direction according to the stride value. When all the horizontal positions are exhausted, the pooling window is moved in the vertical direction according to the stride value. The most commonly used pooling types are max and average pooling. In max pooling the maximum value inside the pooling window is the output of the pooling operation whereas in average pooling the average of all values inside the pooling window is the output of the pooling operation. Max pooling preserves the feature that has the most activation and removes the surrounding features in the pooling window by ruling them out as noise while average pooling takes all of the information into account during the pooling process including the noise. This is why max pooling performs better than average pooling in classification tasks with CNNs. Figure 2.8 shows how both max and average pooling operations are applied on the same input with stride of

1 and windows size of 2 × 2. The dimensions of the image are halved because of the pooling window size.

2.1.3 Fully Connected Layer

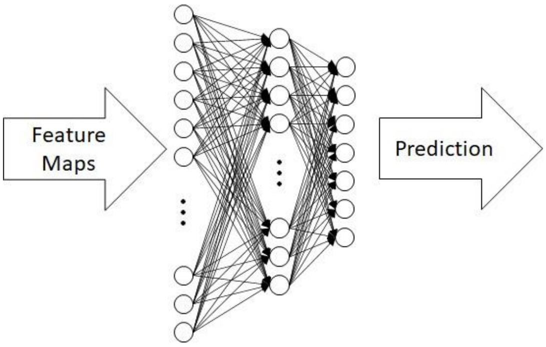


Figure 2.9 Fully connected layers learn the relations between the high-level patterns.

The convolutional and pooling layers extracts the patterns that could be useful in classifying the image while fully connected layers(Figure 2.9) learn how to combine these features to define a specific class. Thus, we can call the fully connected layers as the classifier part of the CNNs while the other half of the network works as a feature extractor. The feature maps are three-dimensional data which need to be connected to fully connected layer which only accepts single dimensional data. Flatten operation is applied on the feature maps and the flattened feature maps are connected to fully connected layers. The number of fully connected layers and how many neurons these layers will include are all hyperparameters that needs to be tuned with optimization techniques. The neurons (Figure 2.10) in the fully connected layers are the same as ANN neurons. Each neuron has weights connected to them and an activation function is applied to the output of the neuron.

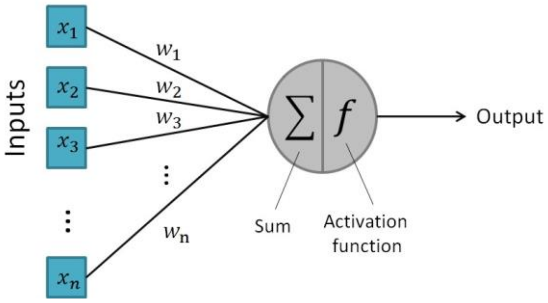


Figure 2.10 Single neuron in Fully Connected Layer.

Fully connected layers are prone to overfitting during the training due to the immense number of connections between the neurons. A regularization technique called Dropout (N. Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov, 2014) is applied to avoid overfitting during the training of the fully connected layers. Dropout randomly selects some neurons in the fully connected layers and disables that neuron with a probability of p to receive or transmit signal temporarily for the current iteration of the training as seen in Figure 2.11. The neuron that is disabled in one training iteration might become active in the next one since the neuron to be dropped out is re-selected on each iteration. The probability p is a hyperparameter that needs to be configured. If the value is set as 0.3, it means that 30% of the neurons will be dropped in each iteration of the training.

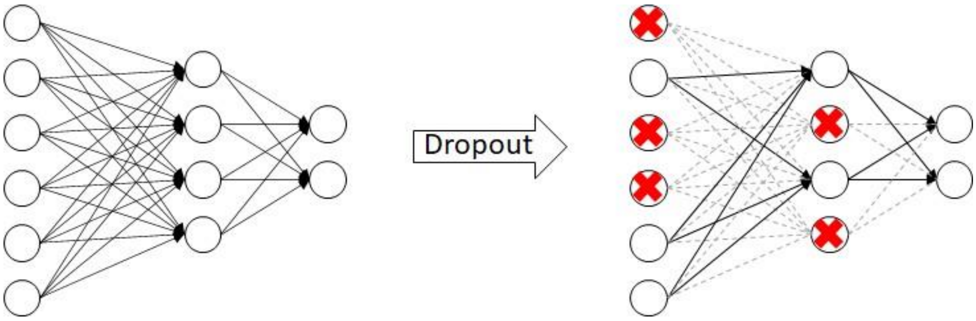


Figure 2.11 Neurons dropped out with dropout regularization technique do not receive or transmit signal. Dropout with probability p = 0.5 is applied to the neurons.

2.1.4 Output Layer

Output layer is the last layer of a CNN. This layer is actually another fully connected layer but a special layer since this is where the predictions are made by CNN. The number of neurons is determined by the number of classes in the dataset. Thus, each neuron represents a class. The activation function in this layer is Softmax. Softmax makes sure that the activation value of each one of these neurons is in (0, 1) interval and the total of the activations of the neurons in the output layer does not exceed 1. The Softmax value is calculated with Equation 2.4.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.4}$$

The activation amount for each neuron in this layer represents the belief that how much the given sample image resembles the specific class represented by the neurons. The neuron with the maximum activation determines the class of the image. As an example, assume that there is a CNN model which is trained to recognize cats and dogs as in Figure 2.12. The Softmax function turns the amount of the stimuli these two neurons receive into a probability distribution. The network believes that this input image belongs to cat class with 77% confidence while the confidence of the network is 23% for this image belonging to the dog class. Since the highest confidence value belongs to the cat neuron, this image is labeled as cat. At this point the predicted label and the actual label of the image is compared and if the labels do

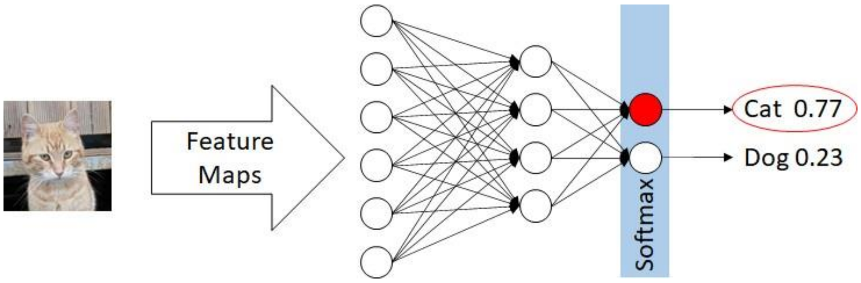


Figure 2.12 Output Layer in a CNN.

not match, the error is calculated to update the weights of the network. The training is explained in detail in the next section.

2.1.5 Training of CNN

As mentioned in the previous sections, CNNs consist of convolutional and fully connected layers which are initialized with a pre-defined initialization method. After initialization, the weights of the filters in convolutional layers and the neurons in the fully connected layers need to be trained to be able to identify the class of the presented object. The training of the CNNs is performed by a gradient descent (GD) algorithm which minimizes the loss function, and the gradients are calculated with backpropagation algorithm.

In basic terms, training has two phases; forward pass and backward pass. In the forward pass, we send an image as the input to the CNN, and we obtain a result from the output layer. This output is then checked for correctness. This is done by comparing the real label of the input image and the class label predicted by the CNN. If both labels are the same, no weight update is required. However, if the labels are found to be different, then the weights in all layers must be updated one by one so that the amount of error between the output and the correct label is minimized (Rumelhart, Hinton, Williams, 1986).

$$CE(y,\hat{y}) = -\sum_{i=1}^{N}(y_i) \cdot \log \hat{y_i} \tag{2.5}$$

The backpropagation starts with the prediction of the label of a particular input data. After this prediction is made by the network, the error between the actual and predicted labels is calculated by a loss function. This loss function is typically categorical cross entropy (Equation 2.5) for scenarios where CNN is labeling more than two classes of objects.

$$W^{t+1} = W^{t} - \eta * \nabla_{g_t}(W) \tag{2.6}$$

We do not know which connections in the network is responsible for this error which is called credit assignment problem. Thus, the error is distributed to all of the units in the network by adjusting the weights by calculating partial derivatives of the errors with respect to the weights according to chain rule until we reach the input layer. The backpropagation algorithm ends when the stopping criteria is met. This stopping criterion could be reaching a specific loss value, a specific number of training epochs or monitoring the validation error rate. Since the weight updates are done in the opposite direction of the gradient (Equation 2.6) and the value of gradients depend on the weights, the performance of the backpropagation is strongly related to the weight initialization method (Sutskever, Martens, Dahl, Hinton, 2013). The learning rate is a hyperparameter whose value should be carefully determined. If the value of η is too small, the weight updates will be small and the convergence would take a long time. On the other hand, if the value of η is large, weight updates might occur in a manner that misses the convergence point by fluctuating around it or in extreme cases instead of converging system might diverge.

Stochastic Gradient Descent (SGD) is the most commonly used training algorithm in CNNs which updates the weights after each training input compared to gradient descent which only applies one weight update by calculating gradients for whole dataset. Because of a single update which requires calculation of all gradients, the gradient descent can be very slow for big datasets. Compared to gradient descent, SGD is much faster because of the gradient calculations per training sample that prevent re-calculation of same gradients over and over as opposed to gradient descent. However, this means that SGD weight updates does not converge to the local/global minima as smoothly as gradient descent. To introduce the smoothness of GD, learning rate annealing is applied to SGD. Both the smoothness of GD and the speed of SGD can be achieved by using mini-batched SGD for training of the CNN where weight updates are calculated for mini-batches of n training samples. The size of the mini-batches depends on the application of CNN. However, there are still problems that need to be addressed in training the network. Selection of initial learning rate value, the annealing schedule or SGD getting stuck at some sub optimal local minima or saddle points (Dauphin et al., 2014) where the gradients are close to zero and it is not possible to escape such a point.

$$\triangle W_t = \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_\tau^2}} g_t \tag{2.7}$$

SGD optimization techniques are implemented to fix the problems of SGD. One of those optimization schemes is introducing momentum(Qian, 1999) which tries to lessen the oscillations to speed up the training. Another method is applying weight updates in a way (Equation 2.7) that allows usage of higher learning rates for less frequent patterns and smaller learning rates for frequent patterns. This is achieved by automatically adjusting the learning rate based on the past gradients computed for the weights. This method is called Adagrad (Duchi, Hazan, Singer, 2011) and removes the manual learning annealing process. However, the learning rate might get very small during training epochs since this method takes all past gradients into account while calculating the new learning rate. This would lead to not learning anything at all.

$$\triangle W_t = -\frac{RMS[\triangle W]_{t-1}}{RMS[g]_t} g_t \tag{2.8}$$

The solution for this problem is proposed in another SGD optimization method called Adadelta (Zeiler, 2012) which is actually a variant of Adagrad. As opposed to Adagrad, Adadelta only takes the past gradients in a small window of fixed size for the calculations instead of all past gradients. Another advantage of Adadelta is that the weight updates does not require a global learning rate value to be set since it is not used in weight update rule as can be observed in Equation 2.8.

CHAPTER 3

3. LITERATURE SURVEY

3.1 Initial Steps

Neural networks have a long history starting from the first description of an artificial neuron by McCulloch and Pitts (McCulloch, Pitts, 1943). Although its evolution was first disrupted by Minsky and Papert's work (Minsky, Papert, 1969) , then by technical limitations on training them, it has become the most thriving research topic due to the technological advances in the recent couple of years.

The concept of artificial neuron was first described by McCulloch and Pitts (Minsky, Papert, 1969) in 1943. The aim of this work was mathematically explaining how the cells in brain works together. This artificial neuron model takes and aggregates one or more binary inputs and applies a linear threshold gate to these inputs to form a binary output. The artificial neuron described here could be used to build networks and solve simple logical expressions containing logical AND, OR or NOT operators. However, this model could only apply some logical operations on the given input and could not learn from experience as in human brain. Later in 1949, a supervised learning algorithm, known as Hebb's rule (Hebb, 1949) today, was proposed. Hebb proposed theories on the learning and memorization mechanisms of the brain. He theorized that if a neuron is responsible of activation of another neuron, then that neuron's efficiency should be increased with some mechanism. At that time there was not any evidence about the neuronal activity happening between the neurons which is known as synaptic plasticity today. The synaptic plasticity is the biological process of strengthening (long-termpotentiation (Lomo,1966)) or weakening (long-term depression (Albus, 1971) (Ito, Sakurai, Tongroach, 1982)) of synapses due to the neuronal activity volume between two neurons. If the volume of synaptic communication between two neurons increases, the synapses are strengthened. In 1958, Rosenblatt (Rosenblatt, 1958) combined the McCulloch-Pitts neuron model with Hebb's ideas and formed Perceptron. In essence, Perceptron changed the way how the inputs are handled to achieve learning with McCulloch-Pitts neurons. Originally McCulloch-Pitts neurons can only accept binary inputs. However, the inputs are associated with adjustable weights in Perceptron. The adjustment on the weights is applied in supervised fashion based on the ideas of Hebb. However, as in McCulloch-Pitts, Perceptron can only work with linearly separable functions and cannot solve XOR as stated in Papert and Minsky's book (Minsky, Papert, 1969) Perceptrons: An Introduction to Computational Geometry in 1969. They demonstrated that it is not possible to classify patterns of nonlinearly separable classes with single layer neural network Perceptron. Actually, this was an oversight of the capability of Perceptrons. Today we know that it is possible to solve nonlinear problems with multilayer Perceptrons. The research on neural networks was slowed down with the limitations mentioned by Papert and Minsky until the proposal of backpropagation algorithm (Rumelhart et al., 1986).

3.2 Backpropagation Era

In modern artificial neural networks, the training is performed with error back-propagation. Even though error backpropagation was first suggested in 1974 (P. Werbosö J. Paul John, 1974) and later applied to an ANN by Werbos (P. J. Werbos, 1982), it became widely known through Rumelhard and Zipser's work (Rumelhardö Zipser, 1985). They showed that when error backpropagation was applied to multilayer neural networks, good internal representations could be discovered. Before the error backpropagation was adopted, domain experts handcrafted features that were specially crafted for the specific task at hand for use in ANNs. The slow process of handcrafting the required features was no longer the issue with the introduction of error backpropagation. Applying backpropagation to ANNs enabled hidden layers to automatically learn these handcrafted features.

Denker et al. (Denker et al., 1989) proposed a neural network which applied convolution operation in 1989. However, the filters used in the convolution operation were handcrafted based on Hubel and Wiesel's work (Hubel, Wiesel, 1959). The handcrafted filters were designed specifically for detecting zip code digits. A similar work by LeCun et al. (Lecun et al., 1989) was also proposed a neural network which included convolutional layers for zip code recognition in the same year as Denker et al.. The difference between Denker et al. and Lecun et al. was how the convolutional filters were obtained. Lecun et al. obtained the convolutional filters with backpropagation training as opposed to handcrafting. Lecun et al.'s network had three hidden layers and to lower the computational cost of the training, a subset of connections within the convolutional layers were discarded. The performance of Lecun et al.'s network was greater than the state-of-the-art at its time of publication. Eventually in 1998, Lecun et al. (LeCun et al., 1998) proposed an updated convolutional neural network architecture named LeNet-5. This network was deeper than its predecessor with a depth of seven layers. This new network's architecture was an alternating set of convolutional and subsampling layers which was connected to fully connected layers. To test the performance of this new architecture, a new dataset called MNIST (LeCun, Cortes, 2010) was created. MNIST only included handwritten digits which were picked from different NIST datasets. Today, this dataset is one of the most popular datasets for benchmarking CNNs.

3.2.1 Fundamental Deep Learning Problem

Introduction of backpropagation training made it possible to train multilayered neural networks. However, it was also the reason that the neural networks research hindered. When backpropagation training made it possible to train the hidden layers, it was seen that the more layers the neural networks had, the training of network became harder or impossible. Thus, the expectation that adding more and more layers to the network would provide better performance was not met. The reason for this training behavior, fundamental deep learning problem, was explained by Hochreiter (Hochreiter, 1991) in his 1991 PhD dissertation. The backpropagation of the errors was not effectively training the first layers on a multilayered neural network due to the vanishing or exploding gradients. In case of vanishing gradients, it was shown that the error signal got smaller and smaller until it made the weight updates insignificantly small on shallow layers of the network. This behavior slowed down the updates and eventually made it impossible to train the most important filters that were on the shallow layers.

3.2.2 Revival of the Neural Networks Research

A greedy unsupervised training scheme was proposed by Hinton et al. (Hinton, Osindero, Teh, 2006) in 2006 which was not affected by the vanishing gradient problem. This new algorithm was a combination of wake-sleep algorithm (Hinton, Dayan, J Frey, M Neal, 1995) and contrastive divergence learning (Hinton, 2002) and applied to Restricted Boltzmann Machines (Smolensky, 1986). It was shown that greedy learning of the initial weights with pre-training overcame the vanishing gradient problem and made it possible to train deep networks with backpropagation. This led to the revival of neural networks research. Even though the shortcomings, the neural networks research kept building on backpropagation training.

3.2.3 GPU Era

With the utilization of Graphical Processing Unit (GPU) instead of Central Processing Unit (CPU) for training, it was seen that it is possible to add more layers to the neural networks and train them relatively faster due to the architectural design of GPUs. The possibility of adding more layers sparked interest in the neural networks research once again in the 2010s. However, adding more layers comes with the fundamental deep learning problem. Thus, the research has been focused on creating mechanisms to avoid this problem since then.

In 2010, it was shown that plain multilayered neural networks can be trained with backpropagation on GPUs (Cireşan, Meier, Gambardella, Schmidhuber, 2010) with a better performance than the state-of-the-art of that time. This was managed with adding more neurons and more layers to plain ANN. To be able to train the network data augmentation techniques were applied to MNIST dataset. No new techniques were applied other than usage of GPU. After this demonstration, Cireşan et al. (Cireşan, Meier, Masci, Gambardella, Schmidhuber, 2011) proposed that CNNs could be trained on GPUs without introducing any new training techniques. The GPU implementation of backpropagation training improved the performance on CIFAR10 (Krizhevsky, 2009) and MNIST datasets.

A new milestone was set in neural networks in 2012. An eight layered CNN model called Alexnet (Krizhevsky et al., 2012) which was trained on GPUs was shown to outperform all of the state-of-the-art machine learning approaches with its remarkable recognition performance on the ImageNet (Deng et al., 2009) dataset.

Alexnet was built on LeNet concepts with more layers. Since the architecture was deeper than its predecessor and still uses backpropagation, it has to deal with the fundamental deep learning problem. The solution was the introduction of avoidance mechanisms that were built around the backpropagation's flaw. Thus, a new activation function which could avoid vanishing gradient problem associated with sigmoid or hyperbolic tangent functions were implemented. The non-linearity was provided with ReLU (Nair, Hinton, 2010) since its gradient was non-saturating which allowed faster convergence. ReLU made it possible to have sparsity of activations which helped with training accuracy and time. However, with such a large network (62 million parameters), overfitting was inevitable. Dropout (N. Srivastava et al., 2014) applied to the hidden neurons and data augmentation techniques were applied to training images to avoid overfitting.

Replacement of sigmoid and hyperbolic tangent with ReLU helped with the vanishing gradient problem. However, ReLU was not the perfect solution. ReLU units tend to die if their gradients become zero. The function f (x) = max(0, x) clearly sets the activation values below zero to zero, where the gradient also becomes zero. If this happens on a neuron, that neuron stops responding to stimuli, and its training permanently stops. This behavior is known as the dying ReLU problem. A predefined slope for negative values is proposed in order to avoid dead neurons in the network. This variation of ReLU is called Leaky ReLU (Maas et al., 2013). Another variation of ReLU was proposed by He et al. (He, Zhang, Ren, Sun, 2016) where the slope is learned as a network parameter. This variation is known as parametric ReLU (PReLU) (He et al., 2015).

Various initialization schemes have been proposed since Alexnet to avoid vanishing/exploding gradient issue. This issue is the fundamental roadblock on achieving convergence on deeper architectures. Glorot et al. (Glorot, Bengio, 2010) proposed a normalized initialization method that took into account the number of input and output connections to each neuron. This initialization technique allowed addition of more layers where the activation function was a sigmoid like function. However, Glorot et al.'s initialization technique was not adequate for networks with ReLU activation functions. He at al. (He et al., 2015) proposed an initialization scheme that allowed very deep networks, which use parametric ReLU as nonlinearity, to converge.

Another side effect of using backpropagation training was pointed by Ioffe et al. as internal covariate shift. It was shown that the weight updates during the back-propagation training changes the weights in a way that would move the inputs of the activation function to saturated regions. As explained in Section 2.1.1.1, when the input moves the sigmoid-like activation functions to their saturated regions, the learning slows down or stops. Ioffe et al. proposed a batch normalization scheme to further augment the weights at a normalization step. Inputs to the activation function were updated with batch normalization method to fix variance and the mean of the layer to avoid the saturated regions of the nonlinearity.

As the CNNs got deeper, the various mechanisms to avoid the weakening of the signal across the layers became insufficient and the training error kept getting bigger with addition of more layers. He et al.(He et al., 2016) proposed skip connections to avoid this problem. Skip connection allowed feeding the output of a layer to some deeper layer rather than just feeding the next layer. The result of adding skip connection was strengthening of the output signal in deeper layer. Since the signal was not as weak as it got with networks without skip connections, He et al. managed to add more layers before network performance drops. A similar network was proposed by Srivastava et al. (Srivastava, Greff, Schmidhuber, 2015a, 2015b) called Highway networks. The difference was that the highway networks used data-dependent parametric gating functions in skip connections.

Often overlooked problem in deep architectures is that going deeper means more and more hyperparameters required to configure the models. In current research, CNNs are defined with hyperparameters like number of filters, number of layers, learning rate, momentum, number epochs, batch size, etc. These hyperparameters need to be tuned carefully to achieve the optimal performance on the task. The tuning of these hyperparameters is another research topic. Grid Search(GS) is the easiest to implement hyperparameter optimization technique where a subset of hyperparameter space is searched (Bengio, 2012) for the optimal performance. However, it is a computationally expensive optimization method (Kaneko, Funatsu, 2015) and the sheer number of hyperparameters used in CNNs make it unsuitable. Even if the number of hyperparameters was not high, GS is found to be stuck at local optima (Keerthi, Lin, 2003). Another method is employing random searches in the hyperparameter space with RS. A combination of GS and RS is proposed in different works (Hinton, 2010; Larochelle et al., 2007; Yann et al., 1998). Hutter (Hutter, 2009) showed that the computationally less expensive SMBO more effectively finds the best solution than RS. Bayesian optimization is shown to be appropriate to optimize(Snoek et al., 2012) CNN hyperparameters since Bayesian approach is suitable for black box type functions. Another approach on hyperparameter optimization on ANNs (Akı, Erkoç, Eskil, 2017) is using a reduced set that will speed up the search for the best parameter values.

As mentioned in LeCun et al. (LeCun, Bengio, Hinton, 2015), the interest in deep networks was revived with Hinton et al.'s unsupervised approach (Hinton et al., 2006). However, the remarkable results of supervised approaches led the research in a purely supervised manner by inventing new ways to circumvent the vanishing gradient problem. LeCun et al. (LeCun et al., 2015) anticipated that unsupervised learning will become more important in the future. As they mention, an infant's brain processes information and extracts concepts purely by observation and not by being taught by a supervisor.

3.2.4 Unsupervised Learning with Backpropagation

While supervised learning remains the dominant focus of research on deep networks, there has also been significant exploration of unsupervised learning within this field. It is worth mentioning that even though the following studies that are discussed under this title are all categorized under unsupervised learning topic, they still employ backpropagation. We can categorize unsupervised methods into self-supervised learning, cluster-based learning, and generative models.

In self-supervised learning, the use of pretext tasks allows for the replacement of data labels with pseudo-labels. Dosovitskiy et al. (Dosovitskiy, Springenberg, Riedmiller, Brox, 2014) generate surrogate classes by first selecting random im- age patches and then applying transformations to the randomly sampled patches. The transformations can be one of rotation, translation, contrast manipulation or scaling operations. The patches may contain whole object or object parts. After the transformations, these surrogate classes are labeled with pseudo-labels. The pseudo-labels are used in backpropagation training instead of the real labels. Thus, the CNN learns to classify the surrogate classes. Another pretext task is usage of relative positions of image patches. This method involves cutting the images into pieces to create a jigsaw. Both (Doersch, Gupta, Efros, 2015) and (Noroozi, Favaro, 2016) utilizes this pretext task. They train their networks to master puzzle-solving. Image colorization is utilized by (Larsson, Maire, Shakhnarovich, 2016, 2017; Zhang, Isola, Efros, 2016). In contrast, the approach proposed by (Pathak, Krahenbuhl, Donahue, Darrell, Efros, 2016) utilizes image in-painting, where the prediction of pixels is based on the information of neighboring pixels. Tracking video frames is also utilized for motion cues in (Misra, Zitnick, Hebert, 2016; Wang, He, Gupta, 2017). Jaehoon et al. (Jaehoon et al., 2018) procure drivable space and surface normals from stereo images. These data then used to produce pseudo ground truth. Finally, to determine the quality of an image, ranking is used as a pretext task in (Liu, Weijer, Bagdanov, 2019).

Gaussian Mixture Models (GMM) and k-means (Macqueen, 1967) is commonly used in cluster-based unsupervised learning methods. The objective of the cluster-based algorithms is generating clusters that can be used for pseudo-labeling the training samples. Training of the network is performed through backpropagation using the assigned pseudo-labels. As a result, the performance of the trained network depends on the clustering performance. Yang et al. (Yang, Parikh, Batra, 2016) performs agglomerative clustering on the output of a CNN. Based on the cluster labels, they update both the clusters and the CNN weights on each backward pass. This is repeated until they reach a stopping criterion. A very similar approach is proposed by (Liao, Schwing, Zemel, Urtasun, 2016). Xie et al. (Xie, Girshick, Farhadi, 2016) proposes deep embedded clustering (DEC) method where input images mapped to feature space by using stacked auto encoders (SAE). To initialize the cluster centroids, k-means clustering is applied on the outputs of SAE. They further refine the clusters by applying Kullback-Leibler (KL) divergence. In another work (Xu, McCord, 2021), spatial vector outputs from a randomly initialized CNN are used to generate clusters by applying GMM. CNN weights are updated by using the cluster assignments. Finally, the features are obtained from the trained CNN. Mahon et al. (Mahon, Lukasiewicz, 2021) trains a couple of auto encoders(AE) in parallel. During the training process, they selectively choose the mutually agreed cluster pseudo-labels. ClusterFit (Yan, Misra, Gupta, Ghadiyaram, Mahajan, 2020) employs self-supervised learning method from (Noroozi, Favaro, 2016) and (Gidaris, Singh, Komodakis, 2018) to train ResNet-50 (He et al., 2016) with the ImageNet (Deng et al., 2009) dataset. They create clusters from another dataset by using k-means along with the pretrained network and assign pseudo-labels to these clusters. They end up with a new dataset with the pseudo-labels generated by clustering process. This dataset is utilized to train a new network which has the same architecture with Resnet-50 from scratch with backpropagation with the objective of minimizing cross-entropy.

Generative Adversarial Networks (GAN) and AEs are unsupervised methods that aim to train models on input data to generate outputs close to inputs. AEs objective is to minimize the reconstruction error between the input training data and their respective reconstructed output. To minimize the error, parameters of the AEs are updated iteratively using gradient descent. In (Masci, Meier, Cireşan, Schmidhuber, 2011), Convolutional Auto Encoder (CAE) is proposed as a weight initialization method for CNNs. CAE is used to obtain localized features from the training data. Later, these feature representations are used as the initial values of a CNN. Another application of CAE (Hou, Yan, 2018) is fingerprint verification. Vincent et al. (Vincent, Larochelle, Lajoie, Bengio, Manzagol, 2010) uses stacked denoising AEs which is able to learn edges resembling to Gabor filters. Using this method is shown to perform better on MNIST dataset compared to ordinary stacked AEs. In (Makhzani, Frey, 2014), k-sparse AE is proposed. In this method, reconstruction is performed by only using the top-k units instead of using all hidden units which allows better accuracy. While AEs aim to learn the latent representations of the input data to better reconstruction in a single network, GANs (Goodfellow et al., 2014) utilizes two networks. The two networks are the generative network and adversary network. While the adversary model aims to differentiate between real and generated data, the generative model’s objective is to deceive the discriminative model. In (Chen et al., 2016), to learn meaningful representations without any label information, mutual information is maximized between the noise variables of the GAN and the observations. Synthetically generated images can be also used to extract features by using GAN (Ren, Lee, 2018). DCGAN (Radford, Metz, Chintala, 2016) is a GAN architecture that uses transposed convolutional network for unsupervised feature extraction.

To summarize, we can divide the unsupervised research into three primary categories: self-supervised learning, cluster-based learning and generative networks. Although we mention the works in this section as unsupervised, all of them still utilize backpropagation for training, whereas our proposal suggests training without backpropagation. By utilizing pretext tasks or clustering methods, the self-supervised and cluster-based learning approaches mentioned earlier designate pseudo-labels to the training data. The models are trained using backpropagation, leveraging the pseudo-labels derived from the aforementioned methods. Generative models consist of two networks. The objective of the generative model is to produce new images that closely resemble the original training images to deceive the discriminative model. On the other hand, the discriminative model’s goal is to differentiate between generated and real data. During the training process, both the generative and discriminative models employ backpropagation for optimization.

3.3 Neocognitron

The roots of CNN architecture dates back to 1980. Fukushima (Fukushima, 1980) proposed a network called Neocognitron which can be trained without back-propagation. While building the Neocognitron architecture, Fukushima implemented the simple and complex cells discovered by Hubel and Wiesel (Hubel, Wiesel, 1959) as alternating layers. With implementing this hierarchical structure, Fukushima managed to extract features through simple cells, while he achieved translation invariance with complex cells. Fukushima proposed supervised and unsupervised learning approaches (Fukushima, 2013, 2016; Fukushima, Hayashi, Léveillé, 2014; Fukushima, Wake, 1991) for training Neocognitron architecture throughout the years following its first introduction.

One of those learning schemes that Fukushima proposed for training of Neocognitron was an unsupervised competitive learning scheme known as Winner-Take-All (WTA) (Fukushima, 2003). Based on Hubel and Weisel’s work, Fukushima implemented simple and complex cells as cell planes in the layers. Upon presenting a visual stimulus to the system, the simple cells engage in competitive interactions to encode the input. Among the simple cells, the one with the strongest response to the input stimulus emerges as the representative within its corresponding cell plane. Through the self-organizing mechanism facilitated by the Winner-Takes-All (WTA) algorithm, the simple cell planes exhibit selective sensitivity towards specific features. This self-organization process relies on a similarity threshold, which regulates the creation of new filters within the WTA algorithm.

Another unsupervised learning method proposed by Fukushima is known as Add-if-Silent(AiS) (Fukushima, 2013). According to the AiS rule, if all post-synaptic simple cells are silent (not stimulated at a predefined rate), a new cell is generated and added to the layer. The input stimulus vector that triggered the generation of a new cell is assigned as the weights of the new cell. The connections to this new cell cannot be changed after the initial values were set. However, since weight updates never occur for the cells, as the training progresses, the number of cell planes steadily grows until the entire feature space is effectively spanned by the reference vectors.

Fukushima’s work is intriguing, as it seeks for visually observable and meaningful cues in the training set as opposed to random initialization of both the number of neurons and their weights and stochastic search towards error minimization, which is the common practice of supervised approaches. As in very early neural network research, we are looking for spatial features that make up simple components of complex objects in the image domain. For this reason, it is both reasonable and intuitive to look into the training images to generate and train new features. Our approach follows this school of thought, which has been strangely ignored in neural network research.

CHAPTER 4

4. APPROACH

4.1 Introduction

In this work, we propose an unsupervised backpropagationless learning algorithm that was inspired by Fukushima’s Neocognitron to train the convolutional layers of CNNs. Our approach (Erkoç, Eskil, 2023) leverages competition of neurons in a convolutional neural network to represent the training samples, hence Competitive Convolutional Neural Network (CCNN). A CCNN is initially empty at the beginning of the training, i.e., there are no neurons/filters defined. Training proceeds layer-wise, the first input stimulus becoming the first filter for the layer. From this point on, we calculate the similarity of the next input stimulus with the filters that are extracted and assigned to the layer. If there is a representative feature of this input in the layer, i.e., the highest similarity result is greater than a similarity threshold, we carry out a weighted update on the weights of the winner filter in contrast to the AiS rule of Fukushima. Otherwise, we conclude that the layer does not possess a representative feature, hence we generate a new filter for the layer and use the particular stimulus to initialize the filter’s weights. Since all of the filters in CCNN are discovered and weight updates are carried with this competitive self-organizing scheme, convolutional layers neither require backpropagation of errors nor a predefined number of filters hyperparameter defined per layer. The process of filter creation and weight adjustments is done in an unsupervised fashion because the decision is based on the similarity of input sample with the previously extracted filters of the current layer instead of backpropagation of error based on a label information.

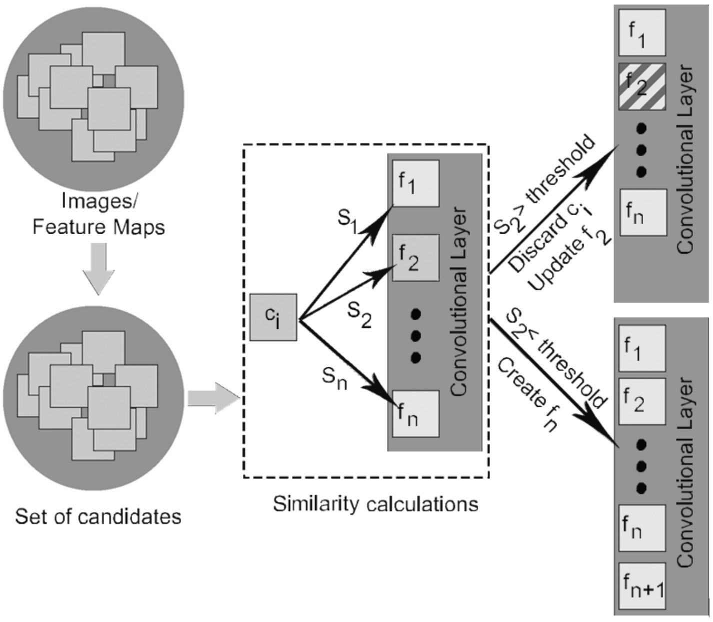


Figure 4.1 The proposed unsupervised backpropagationless filter extraction method. Images/feature maps are converted to candidates from which the filters are discovered without label information. Any filter candidate c_i can become a new filter for the current layer if the maximum similarity value is less than a preset threshold. If not, filter with the highest similarity's weights is updated.

Contrary to the conventional CNN approach, the approach that will be presented here does not require selecting a suitable value for the number of filters hyperparameter of convolutional layers. The filters are discovered in a self-organizing way from the training set images in a single epoch. Proposed training method makes sure that the filters are initialized and trained with a completely unsupervised self-organized scheme. This approach enables us to entirely disregard the filter initialization techniques mentioned in previous sections, as well as the need for training the filters through back-propagation. This results in much fewer epochs of training compared to general CNN approach.

The proposed algorithm is a two-stage filter extraction method. The initial step involves extracting filter candidates from the input data using the center of gravity as a criterion. The next step is to select the filters among the among them using a predefined similarity threshold. The process is depicted in Figure 4.1. The following sections will discuss the method in detail. In Section 4.2, we will discuss convolutional filter discovery scheme of CCNN and in Section 4.2.2, the training of the convolutional layers of the CCNN model is discussed.

4.2 Convolutional Filter Discovery

Typically, the number of filters per convolutional layer is predetermined as a hyperparameter during the model construction process. Since the training of the CNN is the next step after the model building, the filters are actually filled with random numbers from a distribution according to one of the suitable initialization techniques mentioned in Section 2.1.1. The randomly initialized filters then need to be trained to be able to extract meaningful patterns from the images for correct predictions. However, the number of filters is a hyperparameter that needs to be tuned to give the best results since it is not known how many features are needed in each convolutional layer for optimal performance. Another problem here stems from the random initialization of the filters. Because of the randomness of initialization, some number of epochs (that is not known know beforehand) of backpropagation is performed so that the filters can become good feature extractors.

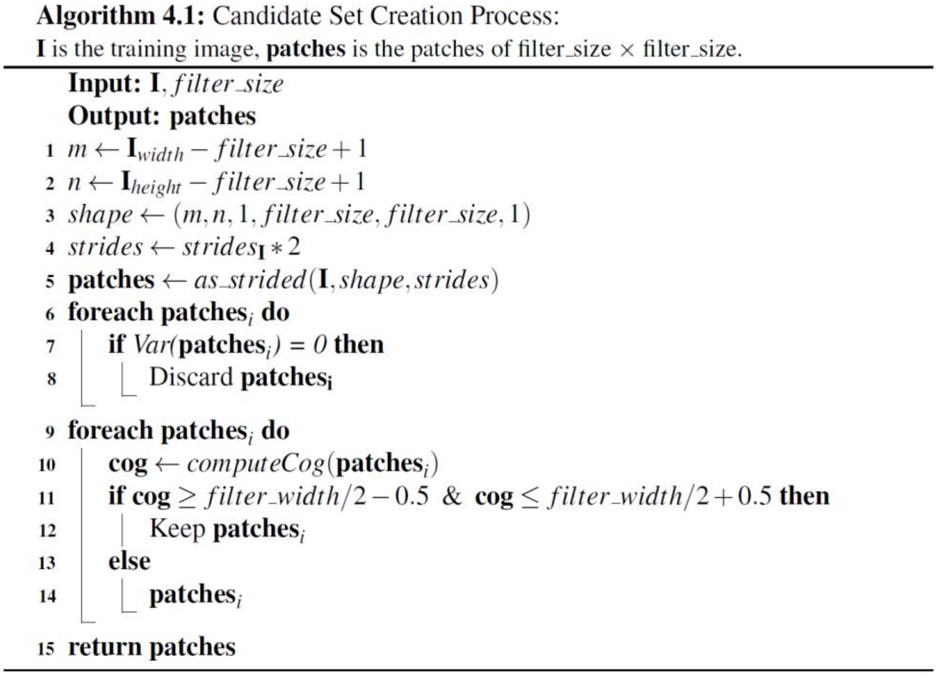
The approach proposed here is a filter extraction scheme where filters are discovered in the training space without the knowledge of the contents and labels of the given data. The input space is analyzed in an unsupervised fashion to discover filters for the current convolutional layer. The process involves competition and relies on a similarity threshold, which is determined through a grid search in increments of 0.1 within the range of [0, 1], where 0 indicates no similarity and 1 represents an exact match. Based on the similarity threshold value, filter discovery scheme either generates a new filter from this candidate filter and adds it to the convolutional layer or updates an already extracted filter that belongs to this convolutional layer. The input filter candidates for the discovery of the filters are prepared from the training images of the given dataset with a process explained in Section 4.2.1. After the inputs are prepared, the filter discovery algorithm is run for only a single epoch. When this epoch is completed, the current convolution layer training is complete and there will be no updates on this layer anymore. This process is repeated for each convolutional layer in CCNN model. The implementation of this approach is a hybrid of CPU and GPU tasks. The filter discovery scheme is implemented entirely on the CPU, while the convolution and maxpooling operations are performed on the GPU for efficient processing. After the filter discovery for the whole system is completed, the CCNN is built and only the fully connected layers undergo training on GPU.

4.2.1 Center of Gravity Based Candidate Filter Extraction

The purpose of convolutional layers is to apply convolution operation to the images. In convolution operation, filters are slid on the input image with a specific stride. At each step, convolution operation is applied on a receptive field sized windows on the input image. The aim of the approach that is presented here is discovering the filters from the input images. Thus, the training images are cut into receptive field sized patches with strides of 1 as discussed in (Erkoç, Eskil, 2022). The stride value is selected as the same value that will be used with the CCNN model for convolutions. In this study, CCNN models are all defined with stride value of 1 in convolutional layers. Since the proposed CCNN model is based on the discovery of the filters, the key to the high performance is based on how the images are transformed into filter candidates. The remaining paragraphs of this section explains the algorithm of choosing the appropriate filter candidates for CCNN model in the perspective of MNIST dataset. Although, the explanation is based on the MNIST dataset, all steps of the proposed algorithm are applicable as it is on other datasets.

Proposed algorithm processes each image in the training set one by one. When an image from the training set is selected, it is first cut into small filter size × filter size patches with strides of 1 which forms a set of filter candidates. However, a very large number of filter candidates are obtained even with a small dataset like MNIST. Moreover, the candidates obtained with this simple process often include no useful information (e.g., background). The proposed algorithm's running time is affected by the sheer number of candidate filters that are obtained with this simple process. The higher the number of the candidates, the longer it takes to calculate similarities. For example,

Each MNIST (LeCun, Cortes, 2010) image's dimensions is 28 × 28. If the filter size is selected as 5 × 5 pixels, then the number of candidate filters that is added to candidate filter set from just one image can be up to 567. Since there are 50000 training images processed by the proposed algorithm, the number of candidate filters obtained from MNIST training set images can reach up to nearly 17 million. The



Algorithm 4.1: Candidate Set Creation Process:
I is the training image, **patches** is the patches of filter_size × filter_size.

```
Input: $\mathbf{I}, filter\_size$
Output: patches
$m \leftarrow \mathbf{I}_{width} - filter\_size + 1$
$n \leftarrow \mathbf{I}_{height} - filter\_size + 1$
$shape \leftarrow (m, n, 1, filter\_size, filter\_size, 1)$
$strides \leftarrow strides_{\mathbf{I}} * 2$
patches $\leftarrow as\_strided(\mathbf{I}, shape, strides)$
foreach patches$_i$ do
    if $Var($patches$_i) = 0$ then
        Discard patches$_{\mathbf{i}}$
foreach patches$_i$ do
    cog $\leftarrow computeCog($patches$_i)$
    if cog $\geq filter\_width/2 - 0.5$ & cog $\leq filter\_width/2 + 0.5$ then
        Keep patches$_i$
    else
        patches$_i$
return patches
```

number of candidates depends on the size of the filters and the number of training images in the dataset. The number of image patches can be larger than 17 million if the size of the filters is reduced from 5 × 5 pixel filters to 3 × 3 pixel filters or just by using another dataset which has more training images than MNIST. The filter extraction process cannot be fast if all of the possible image patches are used as filter candidates in the filter discovery scheme proposed here. Therefore, an elimination procedure is employed on the image patches to guarantee that the size of the candidate set remains within an acceptable range, while retaining essential information.

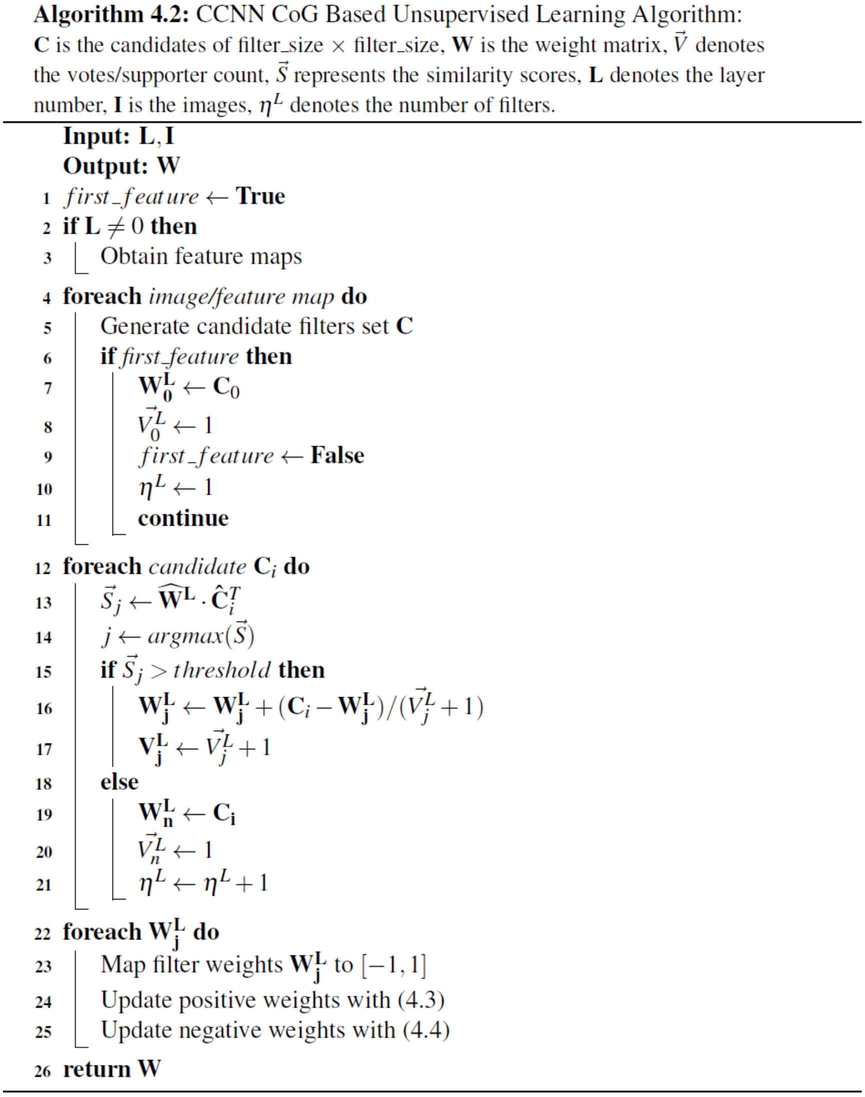
Whenever an image is picked from the training set, the process shown in Algorithm 4.1 is used to extract the candidate filters. The image patches are cut out with specific stride and shape by using Python library NumPy's *as_strided* function. After the execution of *as_strided* function, we obtain a set of patches which contains $(n - filter_{size} + 1)^2$ patches assuming that the image dimensions are n × n. This set contains all possible image patches that can be extracted from that image. However, not all of these patches enclose valuable information. Thus, the patches which have variance value of 0 are discarded because these patches do not contain useful data. Even with this initial elimination, most of the remaining patches in the set do not contain meaningful patterns if closely inspected. Some of the features are on the sides or corners of the filter while the middle part of the filter is all black. This is caused by including the patches that are cut out from the images in strides of 1 throughout the image. While the window is slid through the image, several pieces that contains the same feature in different positions are cut out from the image and added to the candidate filters set. Sometimes the candidate filter window can just capture a couple of pixels from that feature in its corner.

The second elimination removes those kind of image patches from the candidate filters set by just including the features that are positioned in the middle of the patch window. This is accomplished by computing the Center of Gravity (CoG) per image patch. If the determined CoG value is situated at a distance of up to ±0.5 pixels from the CoG of the image patch in both the horizontal and vertical axes, the image patch is considered as a viable filter candidate. Otherwise, the current image patch is discarded from the candidate filter set. CoG based elimination scheme significantly reduces the number of candidate filters. These two elimination methods applied on the image patches can be seen as an attention mechanism rather than a preprocessing step. They ensure that focus is directed towards the features that are relevant for extracting the crucial elements from the training images. After the extraction of filters is completed, the filter weights are stored in a file to be later used in CCNN model.

4.2.2 Unsupervised Learning Algorithm for Convolutional Layers of CCNN Architecture

By executing the first step of the proposed method, we obtain the filter candidates set. The next phase involves the identification of filters from the pool of candidates. The aim is to start from a blank slate and dynamically discovering filters from the training images for each convolutional layer.

The training algorithm described in Algorithm 4.2 operates on a layer-by-layer basis, ensuring that the next convolutional layer is trained only once all possible filters have been discovered for the previous layer. The process starts from the input layer. The input layer L directly fed with the raw training images $\mathbf{I}$ from the selected dataset. The training images $\mathbf{I}$ is processed by Algorithm 4.1 and the candidate patches are stored in $\mathbf{C}$. The discovery of the first filter is a special case since the convolutional layer start with no filters. The algorithm relies on the similarity between a candidate filter patch and convolutional layer filters. Thus, the first candidate filter patch in $\boldsymbol{C_0}$ directly becomes the first discovered filter for the empty



Algorithm 4.2: CCNN CoG Based Unsupervised Learning Algorithm:
$\mathbf{C}$ is the candidates of filter_size × filter_size, $\mathbf{W}$ is the weight matrix, $\vec{V}$ denotes the votes/supporter count, $\vec{S}$ represents the similarity scores, $\mathbf{L}$ denotes the layer number, $\mathbf{I}$ is the images, η^L denotes the number of filters.

```
Input: $\mathbf{L}, \mathbf{I}$
Output: $\mathbf{W}$
$first\_feature \leftarrow$ True
if $\mathbf{L} \neq 0$ then
    Obtain feature maps
foreach image/feature map do
    Generate candidate filters set $\mathbf{C}$
    if first_feature then
        $\mathbf{W^L_0} \leftarrow \mathbf{C}_0$
        $\vec{V}^L_0 \leftarrow 1$
        $first\_feature \leftarrow$ False
        $\eta^L \leftarrow 1$
        continue
foreach candidate $\mathbf{C}_i$ do
    $\vec{S}_j \leftarrow \widehat{\mathbf{W}}^{\mathbf{L}} \cdot \hat{\mathbf{C}}_i^T$
    $j \leftarrow argmax(\vec{S})$
    if $\vec{S}_j > threshold$ then
        $\mathbf{W^L_j} \leftarrow \mathbf{W^L_j} + (\mathbf{C}_i - \mathbf{W^L_j})/(\vec{V}^L_j + 1)$
        $\mathbf{V^L_j} \leftarrow \vec{V}^L_j + 1$
    else
        $\mathbf{W^L_n} \leftarrow \mathbf{C_i}$
        $\vec{V}^L_n \leftarrow 1$
        $\eta^L \leftarrow \eta^L + 1$
foreach $\mathbf{W^L_j}$ do
    Map filter weights $\mathbf{W^L_j}$ to $[-1, 1]$
    Update positive weights with (4.3)
    Update negative weights with (4.4)
return $\mathbf{W}$
```

convolutional layer and added as a filter to this layer. Supporter count (vote) variable $\vec{V}$, which is crucial for the weight update rule of the proposed approach, for this filter is also set as 1. The number of supporters represents the frequency with which the related filter is selected as the representative (winner) for another candidate filter patch. The number of filters η for this layer is incremented by 1.

After the special case of discovering the first filter of the current convolutional layer, the algorithm incorporates a competitive method to discover the remaining filters. All of the previously discovered convolutional layer filters competes to be the representative of the remaining input candidate filters. The competition is based on a similarity threshold and a similarity score. For each candidate filter patch, a similarity score between the candidate and the filters of the convolutional layer is calculated. If the filter and candidate are both vectors in the training space, then the similarity between them can be calculated with dot product. The filter vectors and the candidate vector normalized and dot product calculated between the unit vectors. The calculated dot products are then stored in vector $\vec{S}$. The values inside the vector $\vec{S}$ is the similarity scores of each filter to the current candidate. Since the filters are competing, the winner filter is the one which holds the highest similarity score. However, just winning is not enough to be the representative of this candidate filter patch. The similarity score also must be higher than the user specified similarity threshold. If the score $\overrightarrow{S_j}$ is greater than the similarity threshold value, that means the candidate filter is similar enough to the winner filter. The pattern in the candidate filter $\boldsymbol{C_i}$ is observed in the past. Thus, a new feature is not encountered but a supporter of the winner filter is found among the candidates. Since a new supporter is found for the winner filter, the winner filter adjusts its weights according to weight update rule in Equation 4.1 and the algorithm increases the supporter count associated with the filter by one (Equation 4.2).

$$\mathbf{W_j^L} = \mathbf{W_j^L} + \frac{\mathbf{C_i} - \mathbf{W_j^L}}{\vec{\mathrm{V}}_\mathrm{j}^\mathrm{L} + 1} \tag{4.1}$$

$$\vec{V}_j^L = \vec{V}_j^L + 1 \tag{4.2}$$

If the highest score in $\overrightarrow{S_j}$ is below the similarity threshold value, it indicates that the candidate filter $\boldsymbol{C_i}$ contains a previously unobserved pattern according to the algorithm. Consequently, $\boldsymbol{C_i}$ is acknowledged as a novel filter for the current convolutional layer, and its supporter count is set to 1. This procedure is iterated for each candidate filter until the entire set is evaluated, signifying the completion of the filter search for the current layer **L**. Once all the candidates have been processed, the discovered filters undergo a normalization routine as the final step in the proposed algorithm. As part of the normalization procedure, the filter weights are stretched to fit within the range of [-1, 1]. Subsequently, the positive and negative filter weights are individually updated using Equations (4.3) and (4.4) respectively. Finally, the weights of the discovered filters are stored in a file upon the completion of Algorithm 4.2.

$$\mathbf{W_j^{L(+)}} \leftarrow \mathbf{W_j^{L(+)}} / \left|\sum \boldsymbol{W_j^{L(+)}}\right| \tag{4.1}$$

$$\mathbf{W_j^{L(-)}} \leftarrow \mathbf{W_j^{L(-)}} / \left|\sum \boldsymbol{W_j^{L(-)}}\right| \tag{4.2}$$

The Algorithm 4.2 is also applied to the subsequent convolutional layers in the CCNN model. After the input layer, the process of extracting the filters from the data slightly changes. The input data must now be the output of the previous convolutional and pooling layers. Thus, except for the input layer, proposed algorithm must obtain the feature maps from the previous layer’s output. However, to do this, we first start by creating a Keras Sequential model. The convolutional layers added with Conv2D layer of Keras. The normal operation for Conv2D constructor is to get various input parameters the number of filters, weight initialization method, activation function, bias value, convolution type, stride value, etc. and create the convolutional layer. The weights of the filters are initialized randomly according to the selected initialization scheme. However, in CCNN approach, the filters are discovered before the CCNN model is built in Keras. Thus, the count of discovered filters is set as the number of filters. The discovered filters are also at their trained form, so they should not neither be initialized nor trained. Consequently, the filter weights of the convolutional layer are determined by utilizing the output of the Algorithm 4.2 and setting them using the *set_weights* method in Keras. After the weights of the filters are set, the trainable parameter of the filters is set the False to prevent the training of convolutional layer. If the CCNN model includes a max pooling layer after the convolutional layer, it is created and appended to the
Sequential model.

CHAPTER 5

5. EXPERIMENTS

The proposed method is evaluated on different CCNN model architectures. Handwritten digit datasets MNIST (LeCun, Cortes, 2010) and EMNIST-Digits (Cohen, Afshar, Tapson, van Schaik, 2017), handwritten Japanese character dataset Kuzushiji-MNIST (Clanuwat et al., 2018), and fashion items dataset Fashion-MNIST (Xiao, Rasul, Vollgraf, 2017) are utilized in the experiments. The following sections discuss the model architecture, the experiment settings, the datasets, performance metrics, experiment results and misclassified test samples.

5.1 Model Types

In the experiments, four different model types are utilized. The models are called type A, B, C and D. Table 5.1 shows the general structure of the layers in the CCNN models. We do not know the number of filters hyperparameter value in advance so the number of filters will be different for each parameter setup per dataset. Thus, the final models will be unique to the parameter settings. The max pooling layers are configured to halve the input. Since our algorithm only touches the feature extractor part of the CNN architecture, the classifier part of the models is all set as the same. We configured two fully connected layers separated by a dropout layer.

5.2 Experiment Setup

All models types (Table 5.1) in our experiments are implemented with Keras. The deep learning backend is configured as Theano (Theano Development Team, 2016). We use Sequential model of Keras to build the CCNN models in Table 5.1. After applying our algorithm, we obtain the filter weights for the convolutional layers. These weights are used for initializing the convolutional layers in the Sequential model. However, we freeze the weights so that they become untrainable by Keras.

Table 5.1 CCNN networks that are used in the experiments with various datasets. Convolutional layers either use 5 × 5 or 3 × 3 filters. Maxpooling is applied on the feature maps on 2 × 2 windows with strides of 2. The size of the convolutional filters is denoted with n while the maxpooling window size is shown with *m*.

Model	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Classifier
A	Conv nxn	MaxPool mxm	Conv nxn	-	-	Dense (1000) Dropout (50%) Dense (500) Output (10)
B	Conv nxn	Conv nxn	MaxPool mxm	-	-	
C	Conv nxn	MaxPool mxm	Conv nxn	MaxPool mxm	Conv nxn	
D	Conv nxn	Conv nxn	MaxPool mxm	Conv nxn	Conv nxn	

Convolutional layer convolution mode is set as same to use zero-padding in during the convolution operations. This ensures the preservation of input data dimensions. Pooling layers are configured to apply max pooling operation on 2 × 2 windows with strides of 2. This allows dimensions of the inputs to be halved. Keras trains only the fully connected layers for 50 epochs. Two fully connected layers of 1000 and 500 neurons separated by a Dropout layer with a 50% drop rate in between is configured. The activation function used in the experiments is ReLU. The only exception is the output layer where we used Softmax. The output layer is configured with 10 units. Loss is calculated with categorical cross-entropy and the weights of the fully connected layers are updated with Adadelta (Zeiler, 2012). We used an entry level desktop computer for the experiments which carries a 3.6 GHz Intel Core i7 7700 CPU and a single GTX1050 GPU with 2GBs of VRAM.

5.3 Datasets

The following sections will introduce the datasets that are used in the experiments. Note that the datasets are used as it is. We do not apply preprocessing to the dataset or increase the training set image count by using data augmentation.

5.3.1 MNIST

MNIST is a collection of labeled handwritten digits images. The dataset comprises a training set containing 60000 images and a separate test set containing 10000 images. In our experiments, we partitioned the training set into 50,000 training images and 10,000 validation images. The validation set was randomly chosen and extracted from the original training set. It is important to note that both the training and validation sets exhibit an imbalanced class distribution, resulting in varying sample counts across different classes due to random sampling performed during the separation of the validation sets.

5.3.2 EMNIST-Digits

EMNIST-Digits is a collection of handwritten digit images, similar to the original MNIST dataset, but with an extended range of characters. EMNIST-Digits consists of 10 classes representing the digits 0-9 as in MNIST. It provides a larger and more diverse set of handwritten digit samples in contrast to the original MNIST dataset. The EMNIST-Digits consists of 240000 training and 40000 test images. In the training set, the last 40000 images have been specifically designated as a validation set (Cohen et al., 2017). This validation set has been organized in a way that ensures a balanced distribution of classes.

5.3.3 Kuzushiji-MNIST

The Kuzushiji-MNIST dataset is tailored to capture the distinct features of old cursive Japanese handwriting. It consists of ten distinct hiragana classes. Each hiragana character in cursive Japanese can have multiple variations since they are derived from different kanji characters. As a result, each class in the dataset is represented by several characters that exhibit entirely different writing styles. Due to the substantial intraclass variations, this dataset poses a significant challenge in contrast to the original MNIST dataset. The image counts for the training, validation, and test sets in Kuzushiji-MNIST is identical to the original MNIST dataset. A validation set is generated by randomly selecting and separating a portion from the original training set. Consequently, both the training and validation sets exhibit an imbalanced distribution of classes, with varying numbers of samples across different classes.

5.3.4 Fashion-MNIST

Fashion-MNIST is a dataset designed as a substitute for MNIST, but with a focus on fashion-related images. It comprises a training set containing 50000 images and a distinct test set consisting of 10000 images. To ensure consistency with the MNIST dataset, we follow the same procedure to partition the Fashion-MNIST dataset into training, validation, and test sets.

5.4 Performance Metrics

Performance metrics accuracy, precision, recall, specificity and F1-score are calculated with Equations 5.1-5.5. For calculating these metrics for a class c_i, we use the following definitions:

- ***True Positive (TP):*** the count of images belonging to class c_i that are accurately recognized as class c_i;
- ***True Negative (TN):*** the count of images belonging to other classes and are correctly identified as other classes;
- ***False Positive (FP):*** the count of images belonging to other classes but are incorrectly identified as class c_i;

- ***False Negative (FN):*** the count of images belonging to classes ci but incorrectly identified as other classes.

In addition, we provide a comprehensive assessment by reporting the overall accuracy of all the models.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100 \tag{5.1}$$

$$precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$recall = \frac{TP}{TP + FN} \tag{5.3}$$

$$specificity = \frac{TN}{TN + FP} \tag{5.4}$$

$$F1 - score = 2 \times \frac{recall}{precision + recall} \tag{5.5}$$

5.5 Experiment Details

To evaluate the performance of our proposed method, we run experiments by using different values of similarity threshold. The similarity threshold value is chosen within the range of [0, 1], representing the spectrum from dissimilar to exact match. The threshold values are determined through a grid search process with steps of 0.1. This approach restricts the search space to the specific values listed as: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9. Performance is adversely affected when the similarity threshold is set below 0.5, as setting the similarity threshold too low leads to a considerable reduction in the number of filters obtained from the datasets. We deliberately refrain from employing any preprocessing or data augmentation techniques to evaluate our proposed algorithm in isolation. Each model type is used for each dataset. A total of five training runs are conducted for each individual model, and only the performance metrics of the top-performing CCNN model are reported in the following sections.

CHAPTER 6

6. RESULTS

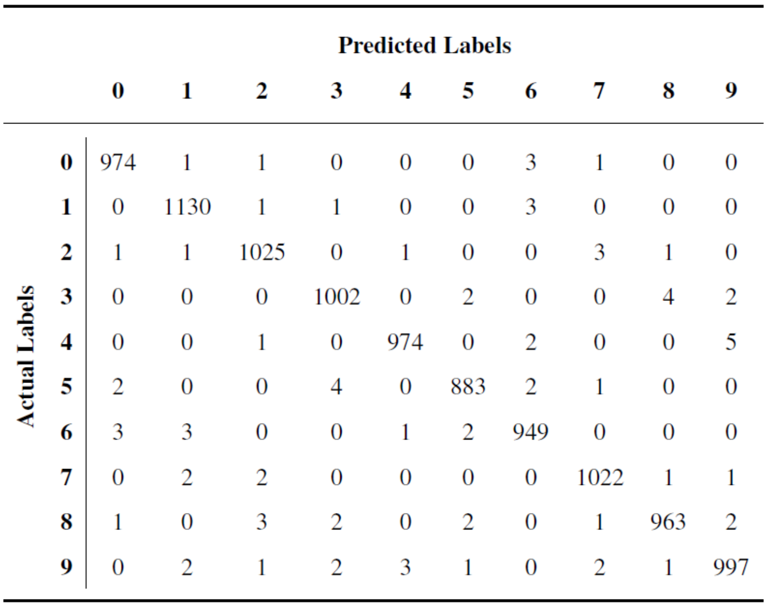
6.1 MNIST Experiment Results

Each model listed in Table 5.1 are evaluated using different combinations of similarity thresholds. The experiments reveal that 5 × 5 filters yield higher accuracy compared to smaller 3 × 3 filters. Among these models, Model A exhibits the highest classification performance achieved on the MNIST dataset, achieving an accuracy of 99.19%, as presented in Table 6.1. The process of extracting filters for the convolutional layers and the top-performing model is trained in a time frame of 30 minutes.

Table 6.1 Extracted filter counts and the test accuracy of individual models on MNIST dataset.

Model	**Similarity Threshold (T)**				**Filter Count (FC)**				**Accuracy(%)**
Type	**T1**	**T2**	**T3**	**T4**	**FC1**	**FC2**	**FC3**	**FC4**	
A	0.6	0.5	-	-	97	54	-	-	**99.19**
B	0.6	0.5	-	-	97	117	-	-	99.18
C	0.6	0.6	0.7	-	97	120	67	-	98.34
D	0.6	0.5	0.6	0.7	97	117	101	145	97.45

Table 6.2 The confusion matrix represents the performance of Model A on the MNIST dataset.



		Predicted Labels									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Actual Labels	**0**	974	1	1	0	0	0	3	1	0	0
	1	0	1130	1	1	0	0	3	0	0	0
	2	1	1	1025	0	1	0	0	3	1	0
	3	0	0	0	1002	0	2	0	0	4	2
	4	0	0	1	0	974	0	2	0	0	5
	5	2	0	0	4	0	883	2	1	0	0
	6	3	3	0	0	1	2	949	0	0	0
	7	0	2	2	0	0	0	0	1022	1	1
	8	1	0	3	2	0	2	0	1	963	2
	9	0	2	1	2	3	1	0	2	1	997

Out of the 10000 test images, the model makes 81 incorrect predictions. The confusion matrix, shown in Table 6.2, provides an overview of the model's performance. It is noteworthy that the digit class that is most accurately predicted by the model is 1, while the most commonly confused class is 9.

Table 6.3 Performance metrics of Model type A for individual classes of MNIST dataset.

Classes	Accuracy(%)	Precision	Recall	Specificity	F1-score
0	99.87	0.9929	0.9939	0.9992	0.9934
1	99.86	0.9921	0.9956	0.9990	0.9938
2	99.84	0.9913	0.9932	0.9990	0.9923
3	99.83	0.9911	0.9921	0.9990	0.9916
4	99.87	0.9949	0.9919	0.9995	0.9934
5	99.84	0.9921	0.9899	0.9992	0.9910
6	99.81	0.9896	0.9906	0.9989	0.9901
7	99.86	0.9922	0.9942	0.9991	0.9932
8	99.82	0.9928	0.9887	0.9992	0.9907
9	99.78	0.9901	0.9881	0.9989	0.9891

6.2 EMNIST-Digits Experiment Results

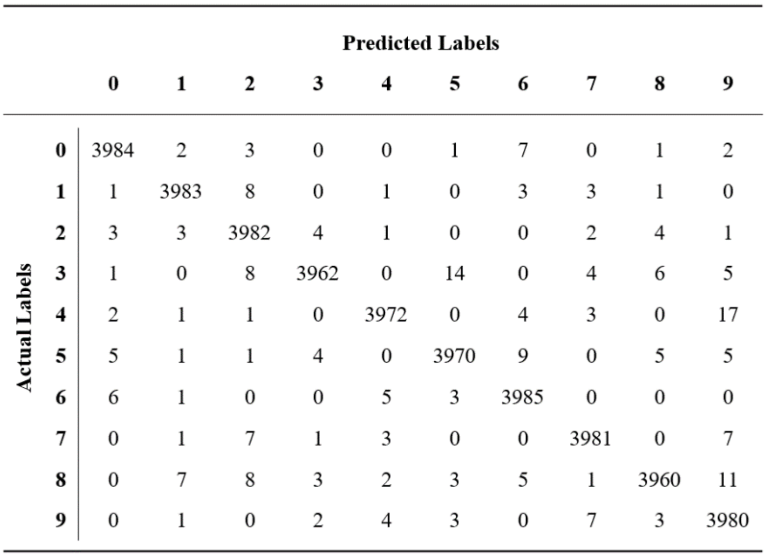
Similar to the results obtained in the MNIST experiments, employing a filter size of 5 × 5 leads to improved accuracy for the EMNIST-Digit experiments. Once again, Model type A remains the best performing model, achieving an accuracy of 99.39% as presented in Table 6.4.

Table 6.4 Extracted filter counts and the test accuracy of individual models on EMNIST-Digits dataset.

Model	**Similarity Threshold (T)**				**Filter Count (FC)**				**Accuracy(%)**
Type	**T1**	**T2**	**T3**	**T4**	**FC1**	**FC2**	**FC3**	**FC4**	
A	0.6	0.5	-	-	145	116	-	-	**99.39**
B	0.5	0.5	-	-	78	161	-	-	99.38
C	0.6	0.5	0.7	-	145	116	101	-	99.11
D	0.5	0.5	0.6	0.5	78	161	148	114	98.94

Out of the 40000 test images, our model makes 244 incorrect predictions. The corresponding confusion matrix can be found in Table 6.5. It is noteworthy that the digit 6 is the class with the highest accuracy in predictions, while digit 8 poses the most significant challenge for the model, as indicated in Table 6.6.

Table 6.5 The confusion matrix represents the performance of Model A on the EMNIST-Digits dataset.



		Predicted Labels									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Actual Labels	**0**	3984	2	3	0	0	1	7	0	1	2
	1	1	3983	8	0	1	0	3	3	1	0
	2	3	3	3982	4	1	0	0	2	4	1
	3	1	0	8	3962	0	14	0	4	6	5
	4	2	1	1	0	3972	0	4	3	0	17
	5	5	1	1	4	0	3970	9	0	5	5
	6	6	1	0	0	5	3	3985	0	0	0
	7	0	1	7	1	3	0	0	3981	0	7
	8	0	7	8	3	2	3	5	1	3960	11
	9	0	1	0	2	4	3	0	7	3	3980

Table 6.6 Performance metrics of Model type A for individual classes of EMNIST-Digits dataset.

Classes	Accuracy(%)	Precision	Recall	Specificity	F1-score
0	99.92	0.9955	0.9960	0.9995	0.9958
1	99.92	0.9958	0.9958	0.9995	0.9958
2	99.87	0.9910	0.9955	0.9990	0.9933
3	99.87	0.9965	0.9905	0.9996	0.9935
4	99.89	0.9960	0.9930	0.9996	0.9945
5	99.87	0.9940	0.9925	0.9993	0.9933
6	99.89	0.9930	0.9963	0.9992	0.9946
7	99.90	0.9950	0.9953	0.9994	0.9951
8	99.85	0.9950	0.9900	0.9994	0.9925
9	99.83	0.9881	0.9950	0.9987	0.9915

6.3 Kuzushiji-MNIST Experiment Results

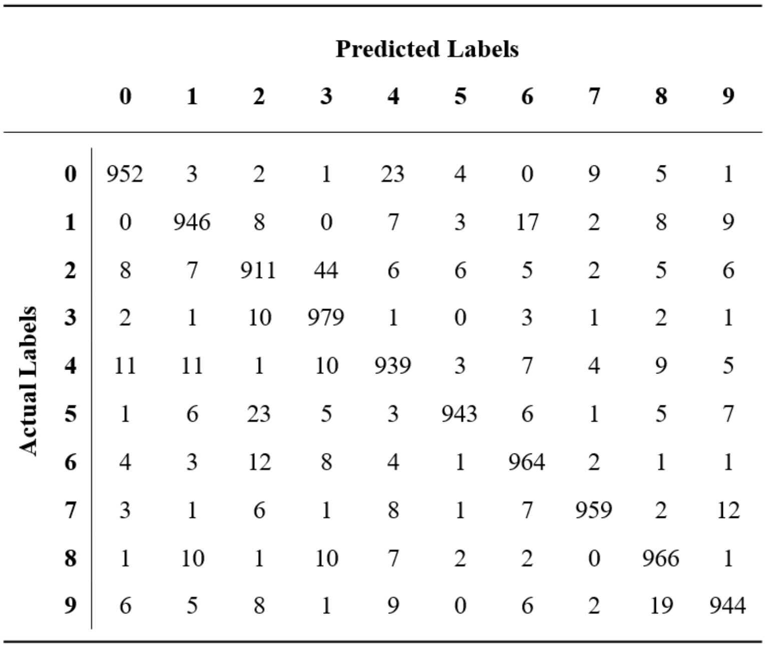
In the case of the Kuzushiji-MNIST dataset, a filter size of 3 × 3 proves to be more effective compared to the use of a 5 × 5 filter size employed with the MNIST and EMNIST-Digit datasets. The highest level of test accuracy is observed in model type B with 95.03%, as presented in Table 6.7.

Table 6.7 Extracted filter counts and the test accuracy of individual models on Kuzushiji-MNIST dataset.

Model	Similarity Threshold (T)				Filter Count (FC)				Accuracy(%)
Type	T1	T2	T3	T4	FC1	FC2	FC3	FC4	
A	0.6	0.5	-	-	51	48	-	-	94.62
B	0.6	0.5	-	-	51	67	-	-	**95.03**
C	0.6	0.6	0.5	-	51	133	43	-	94.90
D	0.6	0.5	0.5	0.4	51	67	140	193	93.55

In the test set predictions, a total of 497 errors are observed. The corresponding confusion matrix for the best model can be found in Table 6.8.

Table 6.8 The confusion matrix represents the performance of Model B on the Kuzushiji-MNIST dataset.



		Predicted Labels									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Actual Labels	**0**	952	3	2	1	23	4	0	9	5	1
	1	0	946	8	0	7	3	17	2	8	9
	2	8	7	911	44	6	6	5	2	5	6
	3	2	1	10	979	1	0	3	1	2	1
	4	11	11	1	10	939	3	7	4	9	5
	5	1	6	23	5	3	943	6	1	5	7
	6	4	3	12	8	4	1	964	2	1	1
	7	3	1	6	1	8	1	7	959	2	12
	8	1	10	1	10	7	2	2	0	966	1
	9	6	5	8	1	9	0	6	2	19	944

Class 3 is the class with the highest accuracy in predictions, while class 2 poses the most significant challenge for the model, as evidenced by a recall of 0.91, as indicated in Table 6.9.

Table 6.9 Performance metrics of Model type B for individual classes of Kuzushiji-MNIST dataset.

Classes	**Accuracy(%)**	**Precision**	**Recall**	**Specificity**	**F1-score**
0	99.16	0.9636	0.9520	0.9960	0.9578
1	98.99	0.9527	0.9460	0.9948	0.9493
2	98.40	0.9277	0.9110	0.9921	0.9193
3	98.99	0.9245	0.9790	0.9911	0.9510
4	98.71	0.9325	0.9390	0.9924	0.9357
5	99.23	0.9792	0.9430	0.9978	0.9608
6	99.11	0.9479	0.9640	0.9941	0.9559
7	99.36	0.9766	0.9590	0.9974	0.9677
8	99.10	0.9452	0.9660	0.9938	0.9555
9	99.01	0.9564	0.9440	0.9952	0.9502

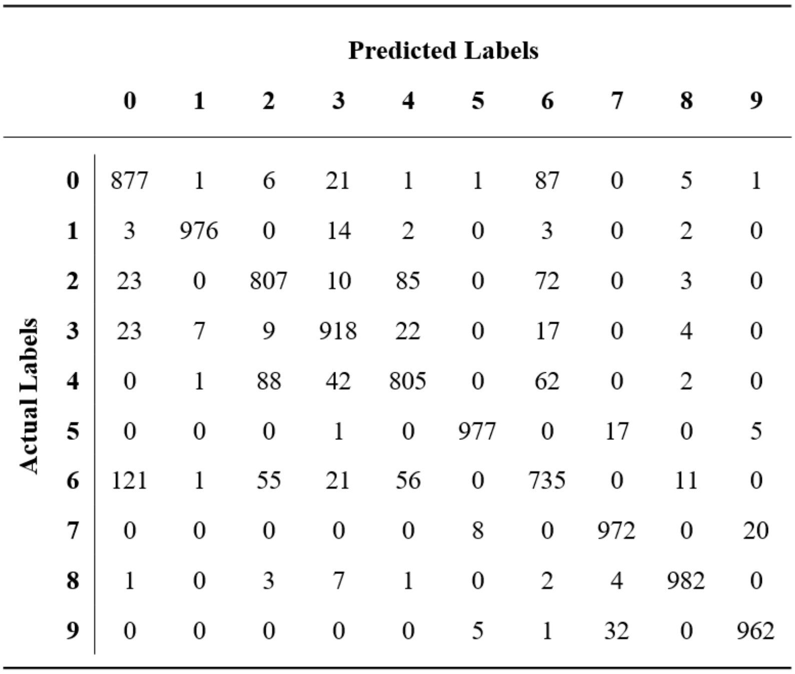
6.4 Fashion-MNIST Experiment Results

In the Fashion-MNIST experiments, it is observed that utilizing a filter size of 3× 3 yields improved results compared to the use of 5 × 5 filters. Model type B emerges as the top-performing model, achieving an accuracy of 90.11%, as depicted in Table 6.10.

Table 6.10 Extracted filter counts and the test accuracy of individual models on Fashion-MNIST dataset.

Model	Similarity Threshold (T)				Filter Count (FC)				Accuracy(%)
Type	T1	T2	T3	T4	FC1	FC2	FC3	FC4	
A	0.7	0.7	-	-	92	48	-	-	88.80
B	0.7	0.6	-	-	92	40	-	-	**90.11**
C	0.7	0.7	0.6	-	92	48	44	-	85.55
D	0.7	0.7	0.5	0.5	92	62	47	18	86.92

Table 6.11 The confusion matrix represents the performance of Model B on the Fashion-MNIST dataset. The classes are assigned to numbers ranging from 0 to 9. In order, the class labels correspond to Tshirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.



		Predicted Labels									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Actual Labels	**0**	877	1	6	21	1	1	87	0	5	1
	1	3	976	0	14	2	0	3	0	2	0
	2	23	0	807	10	85	0	72	0	3	0
	3	23	7	9	918	22	0	17	0	4	0
	4	0	1	88	42	805	0	62	0	2	0
	5	0	0	0	1	0	977	0	17	0	5
	6	121	1	55	21	56	0	735	0	11	0
	7	0	0	0	0	0	8	0	972	0	20
	8	1	0	3	7	1	0	2	4	982	0
	9	0	0	0	0	0	5	1	32	0	962

Table 6.12 Performance metrics of Model type B for individual classes of Fashion-MNIST dataset.

Classes	Accuracy(%)	Precision	Recall	Specificity	F1-score
Tshirt/top	97.06	0.8368	0.8770	0.9810	0.8565
Trouser	99.66	0.9899	0.9760	0.9989	0.9829
Pullover	96.46	0.8337	0.8070	0.9821	0.8201
Dress	98.02	0.8878	0.9180	0.9871	0.9027
Coat	96.38	0.8282	0.8050	0.9814	0.8164
Sandal	99.63	0.9859	0.9770	0.9984	0.9814
Shirt	94.91	0.7508	0.7350	0.9729	0.7428
Sneaker	99.19	0.9483	0.9720	0.9941	0.9600
Bag	99.55	0.9732	0.9820	0.9970	0.9776
Ankle boot	99.36	0.9737	0.9620	0.9971	0.9678

6.5 Filters Discovered via Proposed Unsupervised Process

6.5.1 MNIST Dataset

Figure 6.1 illustrates the filters obtained from the first layer of Model A trained with MNIST dataset. The extracted filters demonstrate noticeable directed edges and curves. These filters extracted through proposed algorithm effectively represent the visual characteristics inherent in the dataset, as depicted in Figure 6.1.One intriguing finding is that a number of our features exhibited convergence towards Gabor-like filters, which have been extensively studied and employed in various research studies.

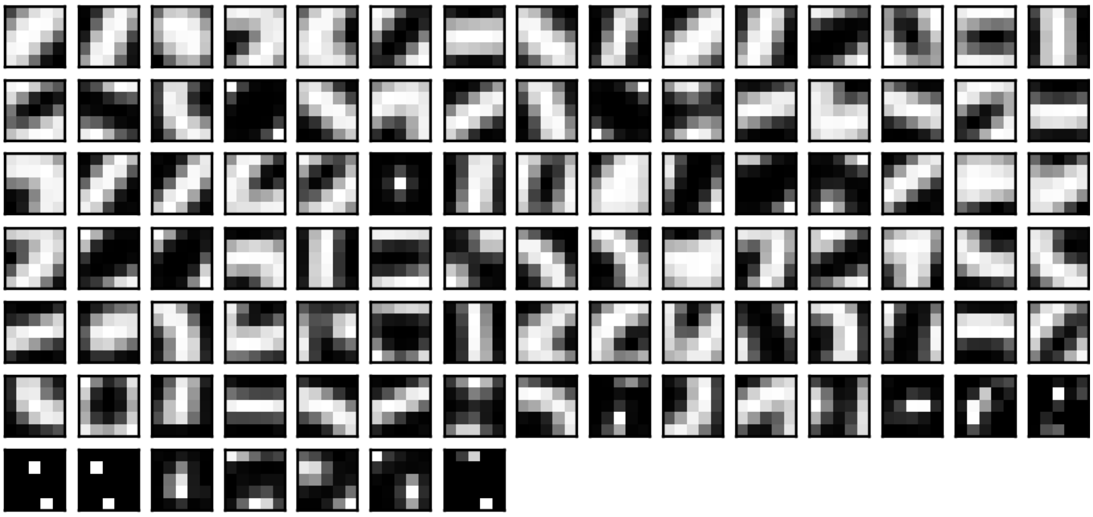


Figure 6.1 First layer filters of Model A trained with MNIST dataset.

6.5.2 EMNIST-Digits Dataset

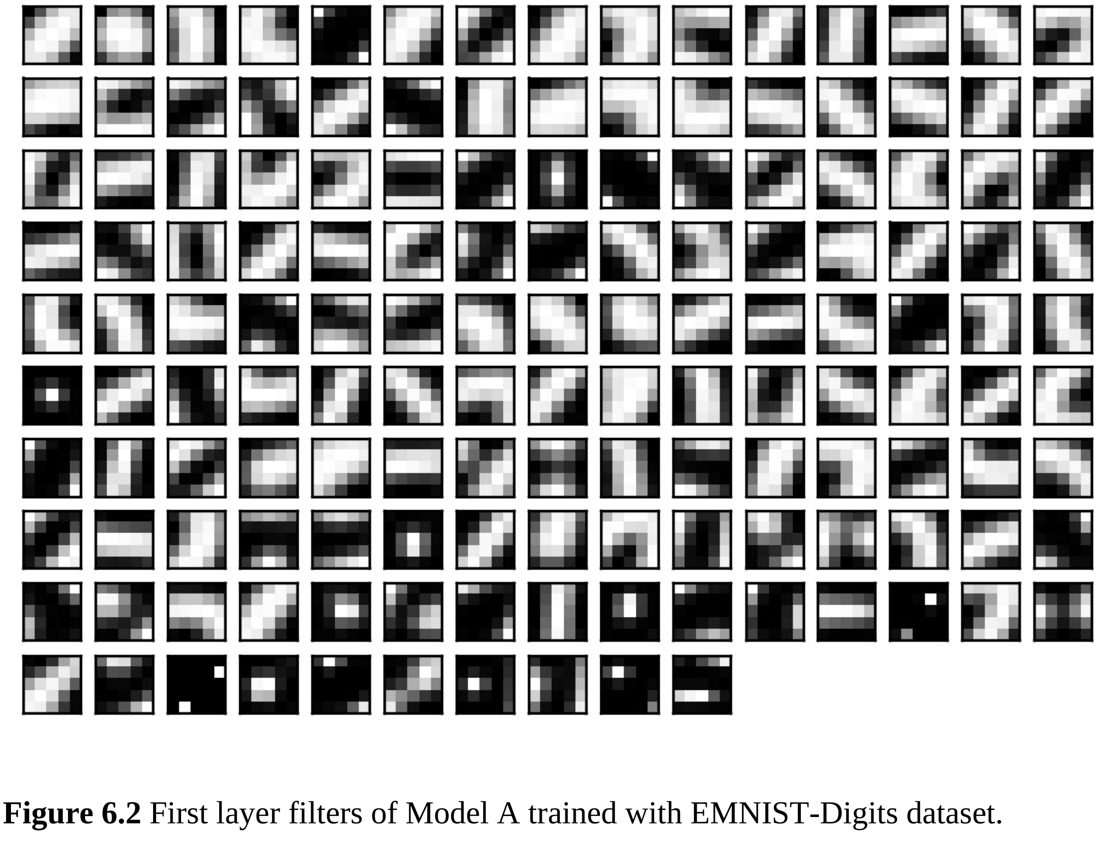


Figure 6.2 First layer filters of Model A trained with EMNIST-Digits dataset.

The EMNIST dataset serves as an extended version of the MNIST dataset, which is why certain filters extracted from the training set of EMNIST-Digits (refer to Figure 6.2) are either identical or highly similar to those depicted in Figure 6.1.

6.5.3 Kuzushiji-MNIST Dataset

In the experiments conducted on the Kuzushiji-MNIST dataset, utilizing 3 × 3 filters for filter extraction yields higher classification accuracy when compared to 5 × 5 filters. Figure 6.3 displays the filters obtained from the training set for the initial convolutional layer. From the extracted filters, one can observe the presence of directed edges and fragments of curved strokes.

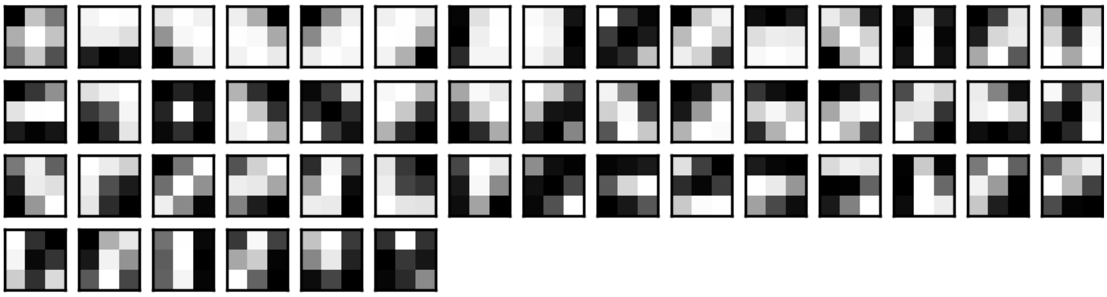


Figure 6.3 First layer filters of Model B trained with Kuzushiji-MNIST dataset.

6.5.4 Fashion-MNIST Dataset

The filters obtained from the Fashion-MNIST dataset capture notable features such as directed edges and corners. In comparison, the MNIST and EMNIST-Digits datasets have a higher number of filters that effectively capture smooth curves, reflecting the prevalence of curved characteristics in the digits as opposed to the fashion items present in the Fashion-MNIST dataset. The extracted filters are visually represented in Figure 6.4.

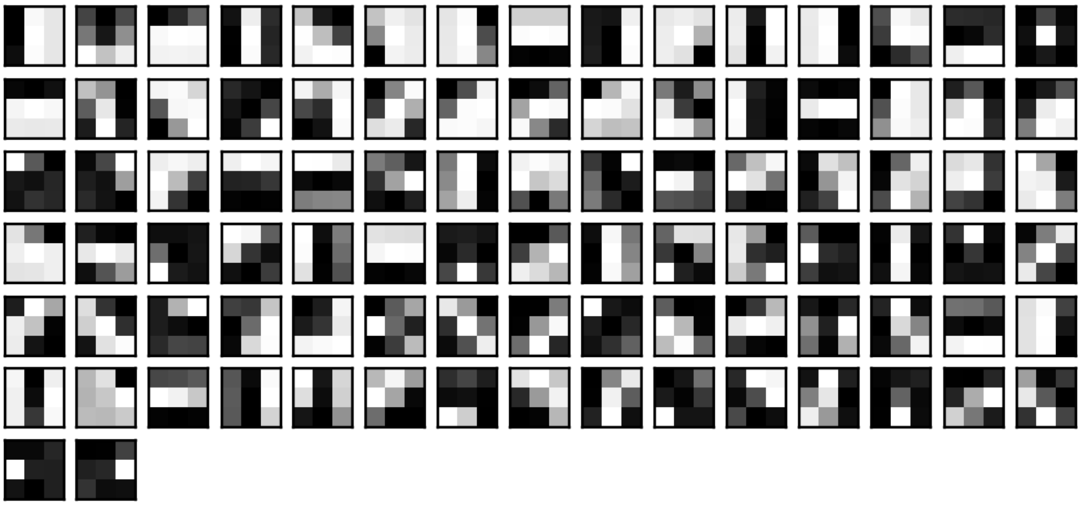


Figure 6.4 First layer filters of Model B trained with Fashion-MNIST dataset.

6.5.5 Extracted Filters in Subsequent Layers

Visualizing the filters in the initial convolutional layer is straightforward since their input weights correspond to specific features and can be easily reshaped to reconstruct images. However, as we delve deeper into subsequent layers, the weights no longer directly map to the input pixels. Therefore, a method is employed to visualize the features in these deep layers, allowing for a more comprehensive understanding of the features extracted by our algorithm.

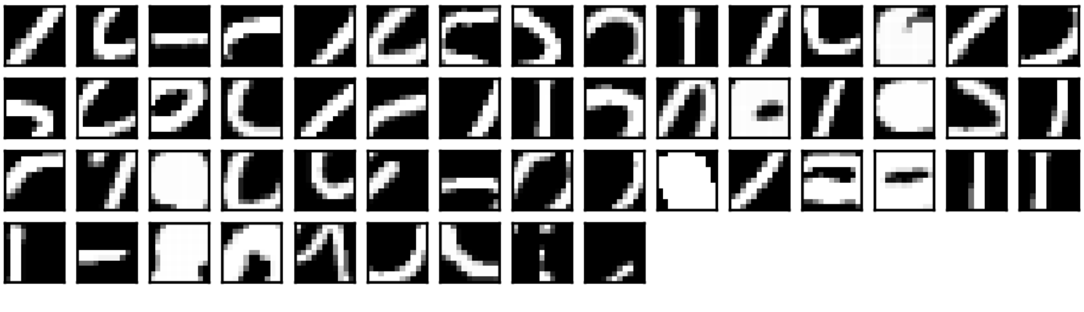


Figure 6.5 The visualization illustrates the collection of 54 features obtained from the MNIST training images using Model type A in the second layer.

Once the training process is completed, the trained model is employed to generate feature maps for each image in the training set at a designated layer. For every training image, the pixel with the highest value across all feature maps is identified and marked. The coordinates of this pixel are then traced back to the original training image, and the corresponding region containing the feature that most strongly activates the specific filter is highlighted. This process is exemplified in Figure 6.5 using the MNIST dataset. The visualization in Figure 6.5 implies that the filters are specialized in detecting features that progressively evolve to represent more intricate characteristics. These intricate features correspond to various parts of the digits, such as closed loops and curves, which are prevalent in digit representations.

6.6 Samples with Incorrect Classification

6.6.1 Incorrectly Classified MNIST Samples

Model type A demonstrates exceptional performance in correctly classifying digit 1 samples from the MNIST dataset, with only 5 misclassifications out of 1135 digit 1 samples in the test set.

The misclassified digit 1 samples are displayed in Figure 6.6. The second, fourth, and fifth images are erroneously labeled as digit 6. We can attribute this to the slight angle and curvature of the digit strokes, as well as the presence of artifacts in the samples, which may have caused confusion in the prediction. The first and third misclassified images are relatively straightforward for human observers to identify correctly; however, the trained model assigns the labels 2 and 3 to them, correspondingly. It is worth noting that these misclassifications may stem from the presence of certain fundamental features shared with other digit 2 samples, leading to an incorrect classification. Upon examining the top-2 predictions for each of these test samples, it is observed that the second most probable prediction is digit 1, with a confidence level very close to the top-1 prediction.

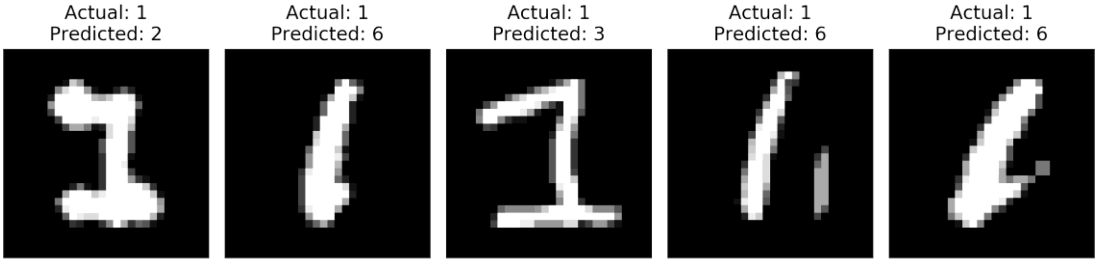


Figure 6.6 The test images belonging to digit class 1 from the MNIST dataset are inaccurately classified by Model A. Among these images, the second, fourth, and fifth samples are mistakenly labeled as 6, potentially due to the presence of artifacts and curved elements within the images

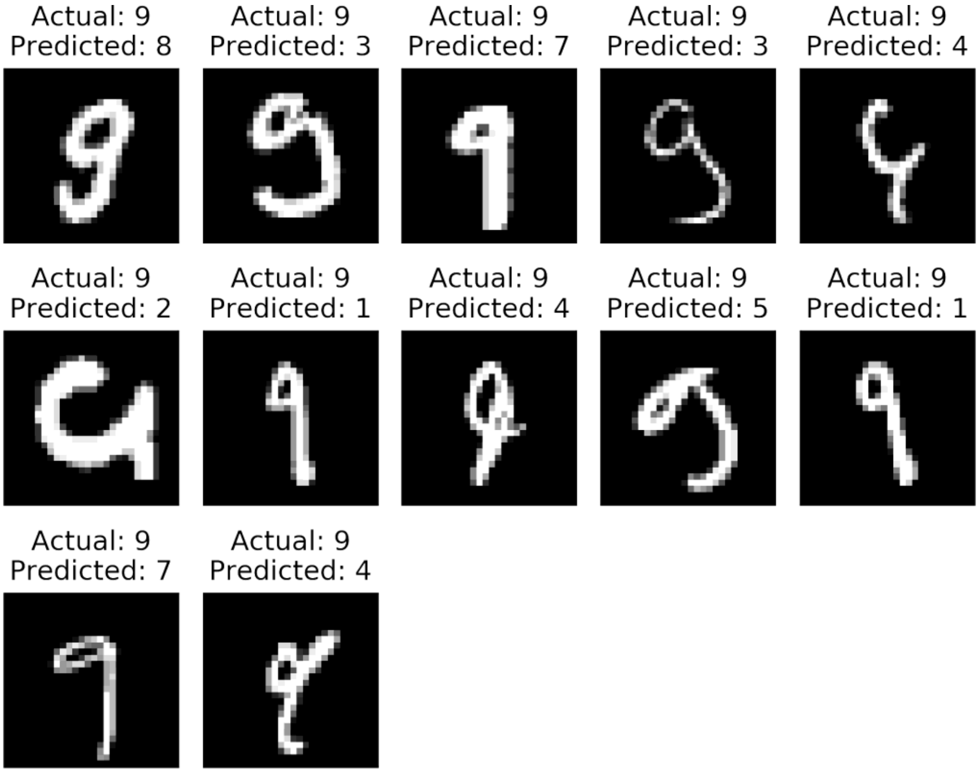


Figure 6.7 The test images belonging to digit class 9 from the MNIST dataset are inaccurately labeled by Model A.

The model's weakest performance is observed in the classification of digit 9 in the MNIST test set. The model demonstrates a tendency to assign varying labels in accordance with distinct writing styles. For images with a small loop diameter, the model tends to assign labels of either digit 1 or 7, determined by the length of the loop, as the loop feature becomes less distinguishable or entirely obscured during the convolution and pooling operations. The first image in the second row presents an intriguing case where the lower half of the digit is clipped, leading to an image that is unidentifiable. All predictions for digit 4 are assigned to unconventional digit 9 samples. Among these, only one is correctly identified as digit 9 by a human observer.

6.6.2 Incorrectly Classified EMNIST-Digits Samples

Our model achieves the highest prediction accuracy when classifying samples from the EMNIST-Digits test set that belong to the digit 6. However, there are instances where our model incorrectly predicts the labels, as depicted in Figure 6.8. Out of the 4000 test samples consisting of the digit 6, our model makes 15 incorrect

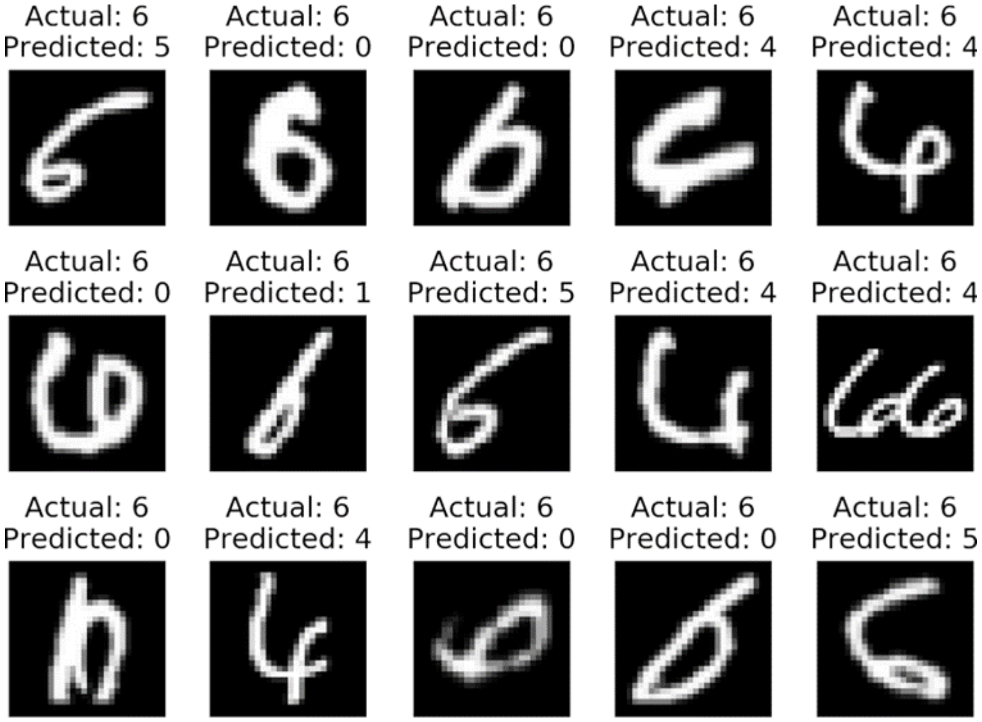


Figure 6.8 Misclassified images from the digit class 6 in the EMNIST-Digits dataset, as predicted by Model A.

predictions. Interestingly, some of these mispredicted samples bear no resemblance to the digit 6 at all. In fact, one of the samples even contains a two-digit number 66 instead of a single digit 6. The presence of rotation and missing parts, caused by cropping, influence the model to favor predicting digit 4. Additionally, mispredictions of digit 0 are also common. Upon analyzing the top-2 predictions, we observe that digit 6 is the subsequent prediction in 12 out of 15 cases.

The digit class 8 exhibits the poorest prediction performance, as Model A incorrectly labels 40 out of 4000 digit 8 images from the EMNIST-Digits test set. In Figure 6.9, we can observe several mispredicted images that lack crucial parts of the digit, making correct classification challenging. Notably, digit 8 is frequently misclassified as digit 9. Upon closer inspection, it becomes evident that 4 of these mispredicted images lack a loop in the lower half of the digit 8. This absence of a prominent curve in the expected location is a common characteristic of these inaccurate predictions. Moreover, some of the misclassified samples do not even resemble digit 8 in any discernible way. Interestingly, when considering the top 2 predictions, digit 8 emerges as the second most likely prediction for 22 out of the mispredicted images.

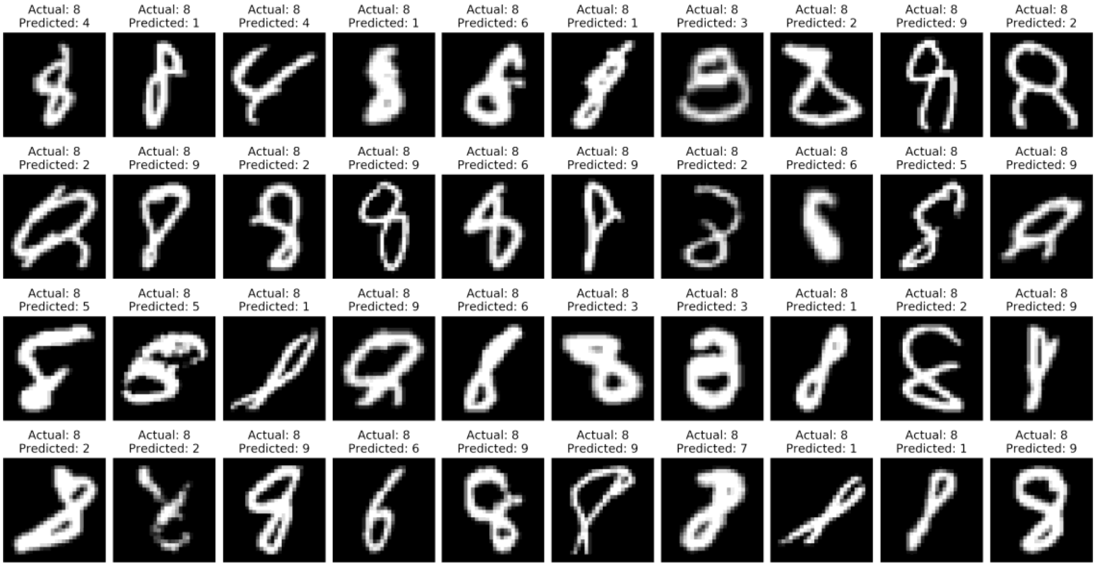


Figure 6.9 The test images belonging to digit class 8 from the EMNIST-Digits dataset are inaccurately labeled by Model A.

6.6.3 Incorrectly Classified Kuzushiji-MNIST Samples

Among all the classes, our best model attains the highest classification performance on class 3 with only 21 prediction errors. However, there is a frequent confusion between class 3 and class 2, leading to mislabeling during testing. The images depicted in Figure 6.10 exhibit features that bear resemblance to other classes, which further complicates the prediction process. Class 3 emerges as the runner-up prediction for 13 of the misclassified images.

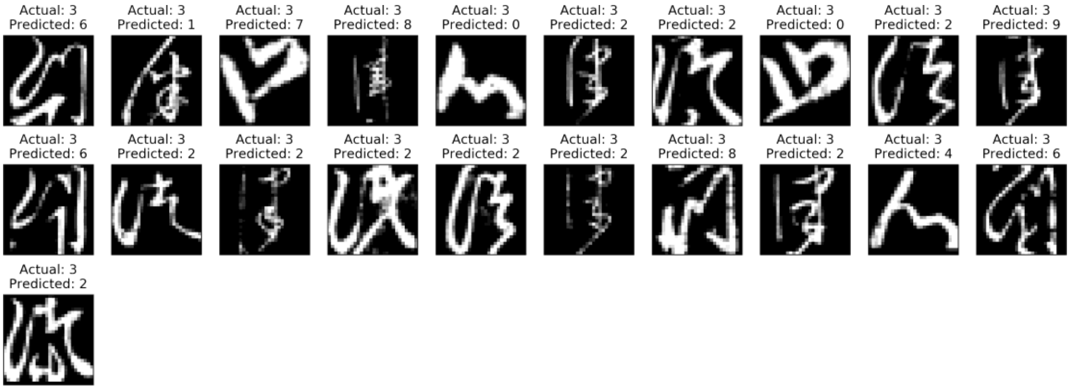


Figure 6.10 Misclassified images from the class 3 in the Kuzushiji-MNIST dataset, as predicted by Model B.

Class 2 poses the greatest confusion for the model, as it frequently misclassifies class 2 images as class 3. Upon closer examination of the mislabeled images, it becomes apparent that many of them exhibit features reminiscent of class 2 samples (Figure 6.11). It is worth noting that for 55 of these samples, the second most accurate prediction corresponds to the correct class.

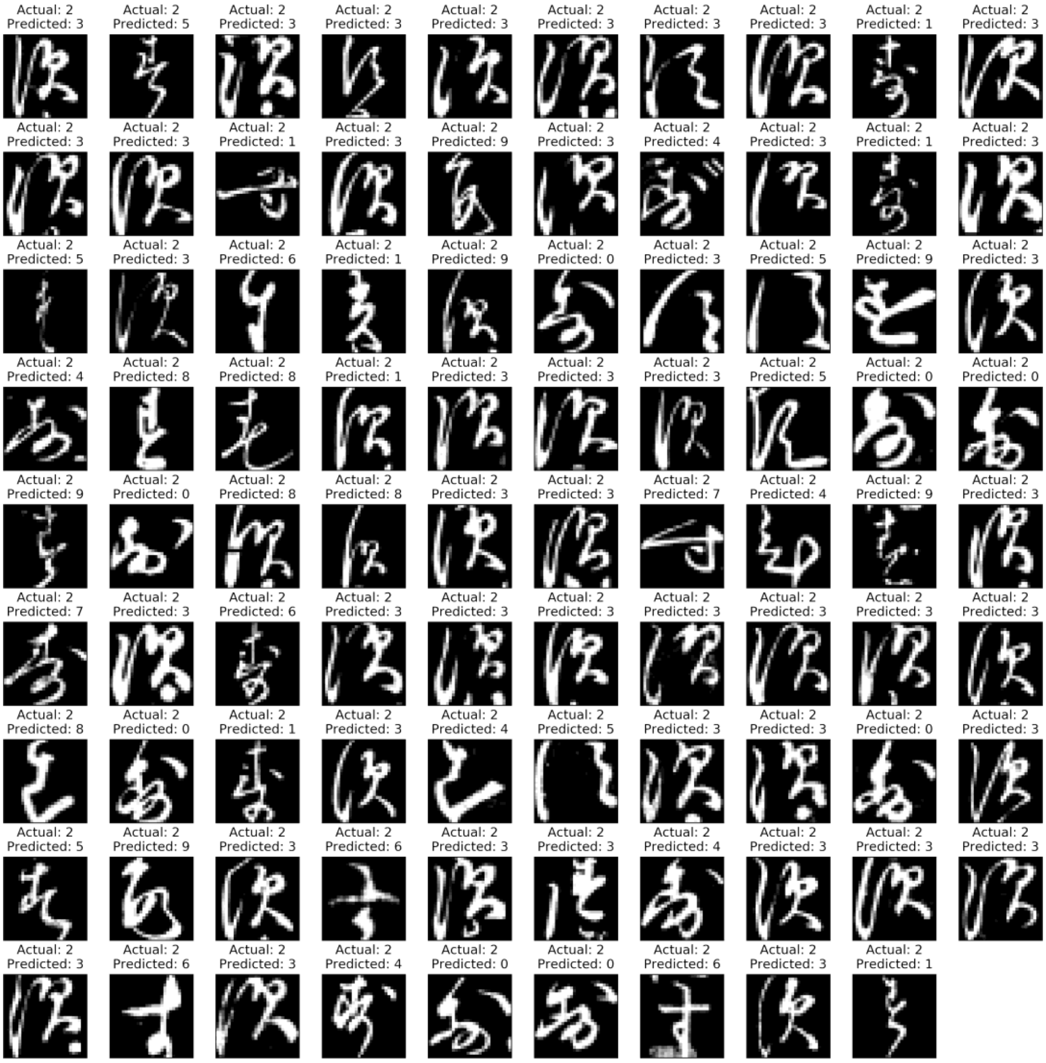


Figure 6.11 The test images belonging to digit class 2 from the Kuzushiji-MNIST dataset are inaccurately labeled by Model B.

6.6.4 Incorrectly Classified Fashion-MNIST Samples

The model achieves its best performance when encountering samples from the Bag class. Out of the 1000 test images of bags, 18 are misclassified. Figure 6.12 displays some of these mislabeled test samples belonging to the Bag class. For instance, in the first row, the second image is incorrectly predicted as a Pullover. The bag image contains two elements that resemble long sleeves, which could have led to the misleading prediction in this particular case. The model frequently confuses Bag class images with Dress class images. Upon examining the second-best predictions for these images, only 2 are correctly identified. Since the Fashion-MNIST images are derived by downsampling colored fashion articles into the MNIST format, many details and features of the objects are lost. The utilization of higher-resolution images could potentially alleviate some of the errors observed in the tests.

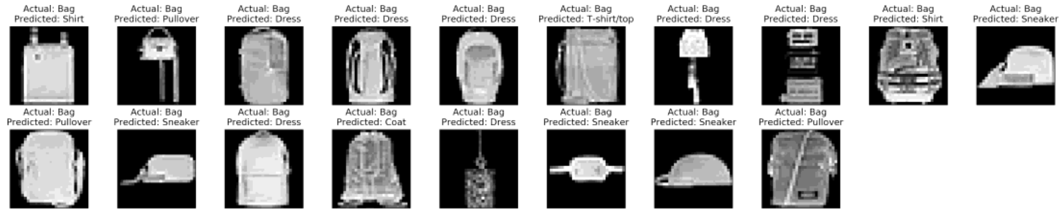

Actual: Bag Predicted: Shirt
Actual: Bag Predicted: Pullover
Actual: Bag Predicted: Dress
Actual: Bag Predicted: Dress
Actual: Bag Predicted: Dress
Actual: Bag Predicted: T-shirt/top
Actual: Bag Predicted: Dress
Actual: Bag Predicted: Dress
Actual: Bag Predicted: Shirt
Actual: Bag Predicted: Sneaker
Actual: Bag Predicted: Pullover
Actual: Bag Predicted: Sneaker
Actual: Bag Predicted: Dress
Actual: Bag Predicted: Coat
Actual: Bag Predicted: Dress
Actual: Bag Predicted: Sneaker
Actual: Bag Predicted: Sneaker
Actual: Bag Predicted: Pullover

Figure 6.12 The test images belonging to the Bag class that were misclassified.

The Shirt class poses the greatest challenge for the model, with 265 test samples misclassified. Figure 6.13 displays some of the incorrectly predicted Shirt images. The model often confuses Shirt samples with those from the T-shirt/Top class. Upon closer inspection, it becomes apparent that the model has learned to associate fashion articles lacking sleeves or with shorter sleeves with the T-shirt/Top category. Analyzing the top-2 predictions reveals that 196 out of the 265 misclassified samples are correctly identified as Shirts.

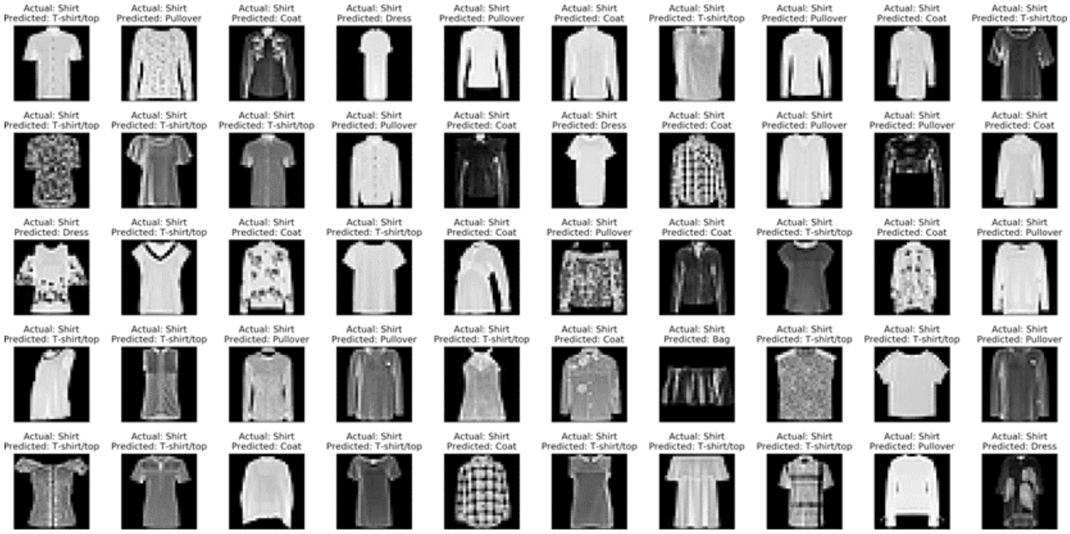

Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: Dress
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: Dress
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: Dress
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: Bag
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Coat
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: T-shirt/top
Actual: Shirt Predicted: Pullover
Actual: Shirt Predicted: Dress

Figure 6.13 The mislabeled test images from the Shirt class, which were incorrectly classified as similar classes by Model B.

CHAPTER 7

7. DISCUSSION

Our method employs a unique training approach for the convolutional layers, utilizing unsupervised learning without the use of backpropagation. In contrast, the fully connected layers are trained using a supervised approach. Unlike previous studies that either trained the network entirely in a supervised manner, or relied on unsupervised learning with pseudo-label backpropagation, or a combination of unsupervised feature learning for initialization with supervised backpropagation, our method offers distinct advantages. In comparison to supervised methods, our approach does not require any labels for training the convolutional layers since we do not employ backpropagation in the training process. Moreover, our method has the advantage of extracting filters without the need for prior domain knowledge. This sets it apart from self-supervised learning methods that rely on the crafting of pretext tasks, which necessitates domain knowledge for satisfactory performance. While our method may resemble unsupervised pre-training, which is typically used for weight initialization, we do not utilize the extracted filters for initialization. This is because we steer clear of supervised training in the convolutional layers.

Table 7.1 Comparison between previous works and our method for the number of epochs of training needed for convolutional filters, whether data augmentation and ensemble of networks are used. The legend of the table: ✓: applied, × : not applied, NA: no information available.

Method	Data Augmentation	Ensemble	Backpropagation	Epochs
HVC (Byerly, Kalganova, Dear, 2021)	✓	✓	✓	300
DropConnect (Wan, Zeiler, Zhang, LeCun,, Fergus, 2013)	✓	✓	✓	1000
MCDNN (Ciresan, Meier, Schmidhuber, 2012)	✓	✓	✓	800
OptConv+Log+Perc (Pad et al., 2020)	✓	×	✓	1000
CAMNet3 (Tissera, Kahatapitiya, Wijesinghe, Fernando, Rodrigo, 2019)	✓	✓	✓	NA
SAM (Foret, Kleiner, Mobahi, Neyshabur, 2021)	✓	×	✓	NA
CAE (Masci, Meier, Cires¸an, Schmidhuber, 2011)	×	×	✓	NA
Deep k-Sparse AE + F.T. (Makhzani, Frey, 2014)	NA	×	✓	200
SPC-best ensemble(Mahon, Lukasiewicz, 2021)	×	✓	✓	NA
SPC-best single (Mahon, Lukasiewicz, 2021)	×	×	✓	NA
k-Sparse AE (Makhzani, Frey, 2014)	NA	×	✓	5000
Disentangled (Agarap, Azcarraga, 2020)	×	×	✓	50
Ours	×	×	×	1
Ours ensemble	×	✓	×	1
Ours init. + train	×	×	×(init.) + ✓(train)	1 (init.) + 50 (train)

7.1 Comparison of Performance Against Other Studies

We evaluate the performance of our proposed method and compare it with unsupervised (Table 7.2), mixed (Table 7.3), and supervised (Table 7.4) approaches. In contrast to our method, other approaches utilize data augmentation, ensembles, and substantial number of training epochs combined with backpropagation to improve their results. A summary of these methods can be found in Table 7.1.

7.1.1 Comparison of Performance Against Unsupervised Studies

The highest reported classification accuracy achieved by unsupervised methods for the MNIST dataset is 99.21% (Mahon, Lukasiewicz, 2021), as indicated in Table 7.2. This accuracy is obtained by leveraging an ensemble of 15 AEs which form clusters. These clusters are associated with k-sets of pseudo-labels, and a consensus function picks the points that are assigned the same pseudo-label in all k-sets for training a Multilayer Perceptron (MLP) using pseudo-labels. The potency of this technique resides in the combined force of the AEs in the ensemble. However, when a single AE is used instead of an ensemble, the accuracy drops to 98.02%, which is inferior to our proposed method. To ensure a fair comparison, we construct an ensemble comprising the top-3 performing Model type A networks obtained from our proposed method. This ensemble achieves a higher accuracy of 99.28% on the test set compared to (Mahon, Lukasiewicz, 2021).

A different unsupervised approach, known as the k-sparse AE (Makhzani, Frey, 2014), explores a training method where the extracted features are held constant, and a logistic regression classifier is trained based on these features. Nonetheless, this method achieves a comparatively lower accuracy of only 98.65% on the MNIST dataset.

Table 7.2 Comparison of the proposed method with other unsupervised studies.

Method	MNIST	F-MNIST
SPC-best ensemble (Mahon, Lukasiewicz, 2021)	99.21	67.94
SPC-best single (Mahon, Lukasiewicz, 2021)	98.02	59.23
k-sparse AE (Makhzani, Frey, 2014)	98.65	-
Ours	99.19	**90.11**
Ours ensemble	**99.28**	**90.43**

SPC-best achieves the highest unsupervised classification accuracy of 67.94% on the Fashion-MNIST dataset. Other unsupervised methods, which yield lower accuracy, are not included in Table 7.2.

As far as our knowledge extends, there are no existing unsupervised studies conducted on the EMNIST-Digits or Kuzushiji-MNIST datasets in the literature.

7.1.2 Comparison of Performance Against Mixed Studies

Masci et al. (Masci et al., 2011) employ a CAE to extract features in an unsupervised manner, which are then utilized to initialize a CNN. Subsequently, the CNN undergoes end-to-end training in a supervised manner, achieving a classification accuracy of 99.29%. Although there are similarities between this method and our proposed approach, we differ in the utilization of extracted features. Unlike Masci et al., we neither use the features for initialization nor subject them to further training. Another related method (Makhzani, Frey, 2014) achieves an accuracy of 99.03% by extracting features using a sparsity constraint on the AE, followed by fine-tuning in a supervised manner.

Table 7.3 Comparison of the proposed method with other mixed studies.

Method	**MNIST**	**F-MNIST**
CAE (Masci, Meier, Cireşan, Schmidhuber, 2011)	99.29	-
Deep k-Sparse AE + F.T. (Makhzani, Frey, 2014)	99.03	-
Disentangled (Agarap, Azcarraga, 2020)	96.20	85.60
Ours	99.19	**90.11**
Ours init. + train	**99.43**	**91.93**

To ensure a fair comparison between our algorithm and the mixed studies (Makhzani, Frey, 2014; Masci et al., 2011), we employ the filters obtained from our algorithm to initialize the convolutional filters. Subsequently, we fine-tune the CNN model A (as shown in Table 5.1) using backpropagation. The performance of this model, referred to as "Ours init. + train," is reported in Table 7.3. We achieve a higher accuracy than the mixed studies, attaining an accuracy of 99.43% on the MNIST test set.

The top performance among the mixed methods on the Fashion-MNIST dataset is documented in (Agarap, Azcarraga, 2020). They achieve an accuracy of 85.60% by training an AE and employing k-means clustering with a soft nearest neighbor loss, which relies on data labels. In comparison, our "Ours init. + train" model achieves a performance that surpasses (Agarap, Azcarraga, 2020) by 6.33% without utilizing data augmentation.

7.1.3 Comparison of Performance Against Supervised Studies

The current highest accuracy achieved for the MNIST dataset is 99.83%, accomplished through the supervised training of capsule networks (Byerly, Kalganova, Dear, 2021). We mention this result to highlight the highest classification accuracy attained among all methods for MNIST. However, our architecture differs from (Byerly et al., 2021) and does not involve capsules, making a direct comparison inappropriate. Similar in architecture, DropConnect (Wan, Zeiler, Zhang, LeCun, Fergus, 2013) and MCDNN (Ciresan, Meier, Schmidhuber, 2012) methods both present outcomes achieved by utilizing ensembles of networks with the aid of data augmentation. In the absence of data augmentation, the performance of DropConnect's 5-network ensemble decreases to 99.43% after training for 1000 epochs. In contrast, our model offers a compelling alternative with a simpler architecture, omitting the need for an ensemble. It trains much faster, requiring just a single epoch to train the convolutional layers, while still achieving an accuracy of 99.19%.

Table 7.4 Comparison of the proposed method with other supervised studies.

Method	**MNIST**	**EMNIST-Digits**	**K-MNIST**	**F-MNIST**
HVC (Byerly, Kalganova, Dear, 2021)	99.83	-	-	93.89
DropConnect (Wan, Zeiler, Zhang, LeCun, Fergus, 2013)	99.79	-	-	-
DropConnect no aug. (Wan, Zeiler, Zhang, LeCun, Fergus, 2013)	99.43	-	-	-
MCDNN 35-net (Ciresan, Meier, Schmidhuber, 2012)	99.77	-	-	-
MCDNN 1-net (Ciresan, Meier, Schmidhuber, 2012)	99.53	-	-	-
OptConv+Log+Perc (Pad et al., 2020)	-	99.43	-	-
CAMNet3 (Tissera, Kahatapitiya, Wijesinghe, Fernando, Rodrigo, 2019)	99.78	-	99.05	94.34
CAMNet3 no aug. (Tissera, Kahatapitiya, Wijesinghe, Fernando, Rodrigo, 2019)	99.47	-	97.48	93.00
SAM (Foret, Kleiner, Mobahi, Neyshabur, 2021)	-	-	-	96.41
Ours	99.19	99.39	95.03	90.11
Ours init. + train	99.43	**99.63**	96.48	91.93

The current highest performance achieved on the EMNIST-Digits dataset is reported by the supervised OptConv+Log+Perc method (Pad et al., 2020), achieving an accuracy of 99.43%. This method applies a large optical convolution with logarithmic activation followed by perceptron training on the images. The study presented in (Pad et al., 2020) relies on a specialized camera setup to attain its optimal performance, in contrast to our study which utilizes the raw dataset images. Unlike our approach, data augmentation is applied during training in (Pad et al., 2020). Our highest-performing model attains an accuracy of 99.39% on the EMNIST-Digits test set, as demonstrated in Table 6.4. This accuracy value is on par with the current leading performance. To ensure a fair comparison, when we continue training the extracted filters, we notice that our model achieves a higher accuracy of 99.63%, surpassing the accuracy of (Pad et al., 2020).

Model B achieves the highest performance on the Kuzushiji-MNIST dataset, with an accuracy of 95.03% (refer to Table 6.7). This accuracy is comparable to the performance of a simple CNN, which achieves 95.12% accuracy as reported in the original Kuzushiji-MNIST paper (Clanuwat et al., 2018). Unlike the MNIST dataset, Kuzushiji-MNIST exhibits significant intraclass variations, where samples belonging to the same class may not resemble each other. This inherent variation poses a challenge for classification, especially without data augmentation. The current state-of-the-art accuracy on the Kuzushiji-MNIST dataset is 99.05%, achieved by CAMNet3 (Tissera, Kahatapitiya, Wijesinghe, Fernando, Rodrigo, 2019). However, CAM-Net3 differs significantly from our architecture. It is a multipath CNN that dynamically routes data flow to different parallel networks based on image content. This unique design of CAMNet3 allows it to better capture the intraclass variation present in Kuzushiji-MNIST compared to the conventional CNN architecture used in our experiments. By utilizing the filters extracted from the unsupervised training phase to initialize the convolutional layers and then applying backpropagation, we achieve a performance boost in the model, resulting in an accuracy of 96.48% without the need for data augmentation.

In contrast to MNIST, the Fashion-MNIST dataset presents more intricate features, greater intraclass variations, and similarities between classes. Consequently, we anticipate a decline in classification performance compared to that of MNIST. The current state-of-the-art accuracy achieved on the Fashion-MNIST dataset is attained by a supervised network, reaching an accuracy of 96.41% (Foret, Kleiner, Mobahi, Neyshabur, 2021). The approach in (Foret et al., 2021) incorporates Wide-Res-Net-28-10 (Zagoruyko, Komodakis, 2016) and Shake-Shake (Gastaldi, 2017) regularization techniques, along with data augmentation methods.

7.2 Proof of Linear Independence of the Extracted Filters

The proposed algorithm claims to extract enough number of filters that are necessary to cover the feature space spanned by the given dataset. To prove this claim, it is important to analyze the extracted filters. It can be proved that the set of the extracted filters span the dataset’s feature space by proving that the filters are linearly independent.

Gram-Schmidt orthogonalization process is defined over linearly independent set of vectors to form an orthogonal basis. Whenever there exist an unnecessary (i.e., linearly dependent) vector in a given set, Gram-Schmidt orthogonalization process outputs zero vector. The goal of the Gram-Schmidt process is to construct an orthogonal set. Thus, a linearly dependent vector does not contribute to the orthogonal basis because it lies in the subspace spanned by the previous vectors. In practical terms, if there exist a set of linearly independent vectors $\{v_1, v_2, \dots, v_n\}$, and one of the vectors v_k is linearly dependent on the previous vectors, the Gram-Schmidt process would produce an orthogonal set $\{u_1, u_2, \dots, u_{k-1}, u_{k+1}, \dots, u_n\}$, where u_{k+1} corresponds to the vector that was originally after v_k.

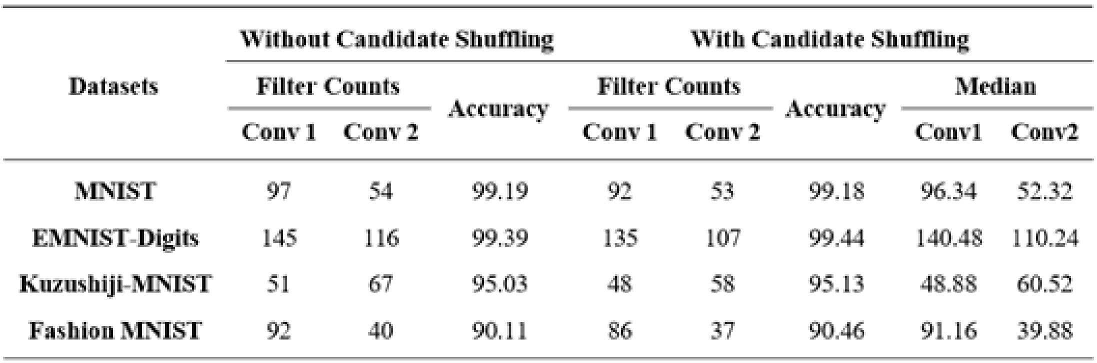
$$u_i = v_i - \sum_{k=1}^{i-1} \frac{u_k \cdot v_i}{\|u_k\|^2} u_k \tag{7.1}$$

After obtaining the filters through Algorithm 4.2, we apply Gram-Schmidt orthogonalization process by using Equation 7.1 for each convolutional layer. The orthogonal basis formed by applying the process does not omit any of the original filters. Thus, we conclude that the set of filters obtained by running Algorithm 4.2 is linearly independent. The absence of a zero-vector output from the process proves that our claims are correct.

7.3 Proof of Independence over the Order of Candidate Processing for Filter Extraction

Algorithm 4.2 takes each candidate from the candidates set and compares its similarity against the filters of the current layer. One might ask whether the order of processing the candidates can impact the filter extraction and overall performance of the proposed algorithm. To alleviate this concern, we designed further experiments where the candidates set is shuffled just after the set is obtained from the input images. To do this, we integrated a shuffle mechanism into the Algorithm 4.2 between step 5 and 6. For each image, we obtain the candidates set and then shuffle the contents of the set. Thus, the order of the candidates that are processed at each run of the algorithm is now random.

Table 7.5 Comparison of best performing model filter counts and test accuracy before and after addition of candidate shuffling. The median of the 50 runs of the experiments is also presented.



Datasets	Without Candidate Shuffling			With Candidate Shuffling				
	Filter Counts		Accuracy	Filter Counts		Accuracy	Median	
	Conv 1	Conv 2		Conv 1	Conv 2		Conv1	Conv2
MNIST	97	54	99.19	92	53	99.18	96.34	52.32
EMNIST-Digits	145	116	99.39	135	107	99.44	140.48	110.24
Kuzushiji-MNIST	51	67	95.03	48	58	95.13	48.88	60.52
Fashion MNIST	92	40	90.11	86	37	90.46	91.16	39.88

After adding shuffling of the candidates to the Algorithm 4.2, we repeat the best performing experiment 50 times for each dataset mentioned in Section 5.3 to observe whether there is a significant impact on the outcome due to the order of

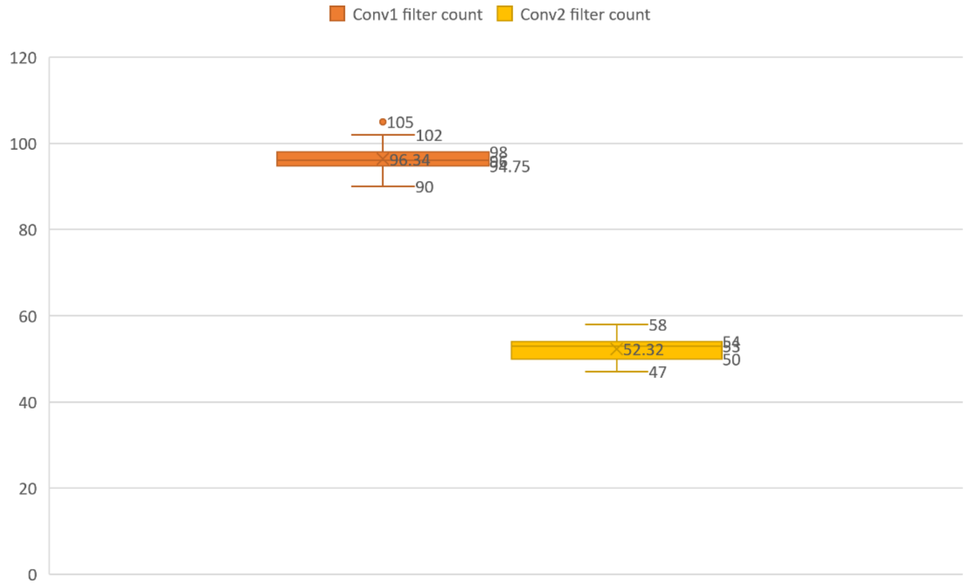


Figure 7.1 Boxplot of the number of filters extracted from MNIST dataset for both layers of Model A with candidate shuffling.

processing the candidates. We represent the distribution of the number of filters extracted for each layer as a boxplot in Figure 7.1, 7.2, 7.3 and 7.4 for MNIST, EMNIST-Digits, Kuzushiji MNIST and Fashion MNIST datasets respectively. The number of filters for each layer in the best performing models per dataset is very close to median of the shuffled candidate set experiments as shown in Table 7.5. We also observed that the test accuracy of the models does not fluctuate much as shown in Figure 7.5, 7.6, 7.7, 7.8. The variation in test accuracy across 50 runs with candidate shuffling is insignificant. Thus, we conclude that the impact of order of processing the candidates for filter extraction is negligible due to similar results obtained in the experiments.

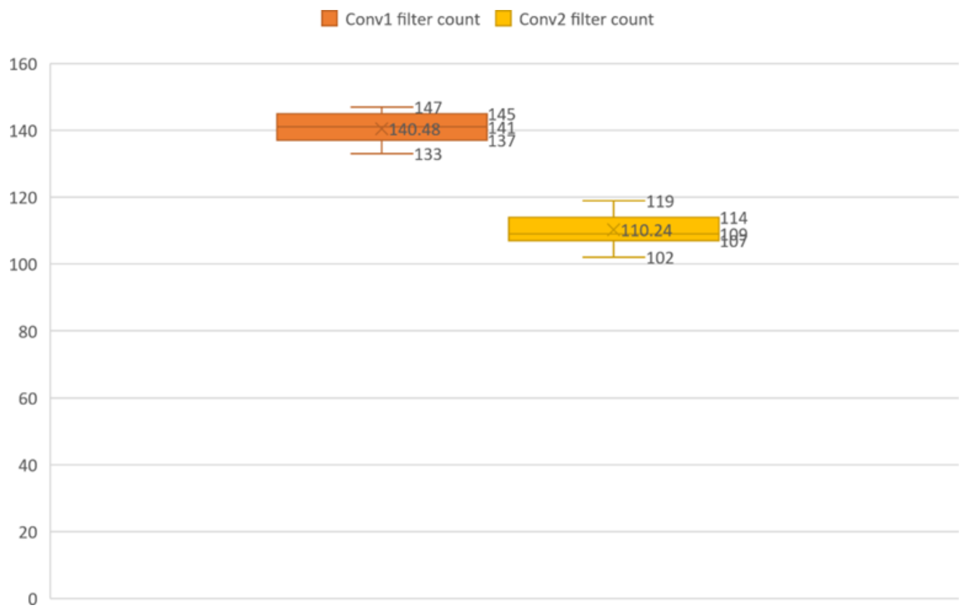


Figure 7.2 Boxplot of the number of filters extracted from EMNIST-Digits dataset for both layers of Model A with candidate shuffling.

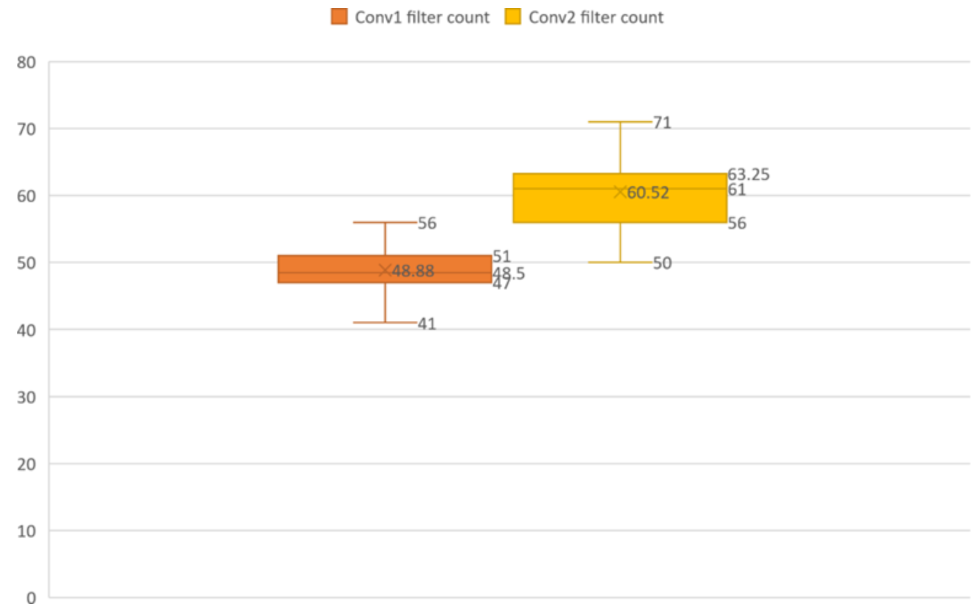


Figure 7.3 Boxplot of the number of filters extracted from Kuzushiji-MNIST dataset for both layers of Model B with candidate shuffling.

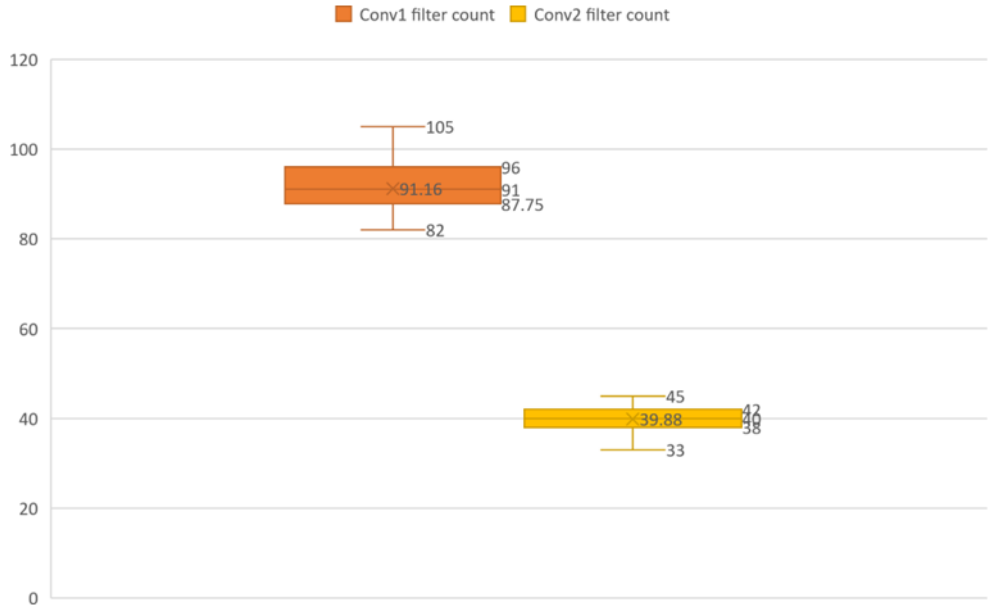


Figure 7.4 Boxplot of the number of filters extracted from Fashion MNIST dataset for both layers of Model B with candidate shuffling.

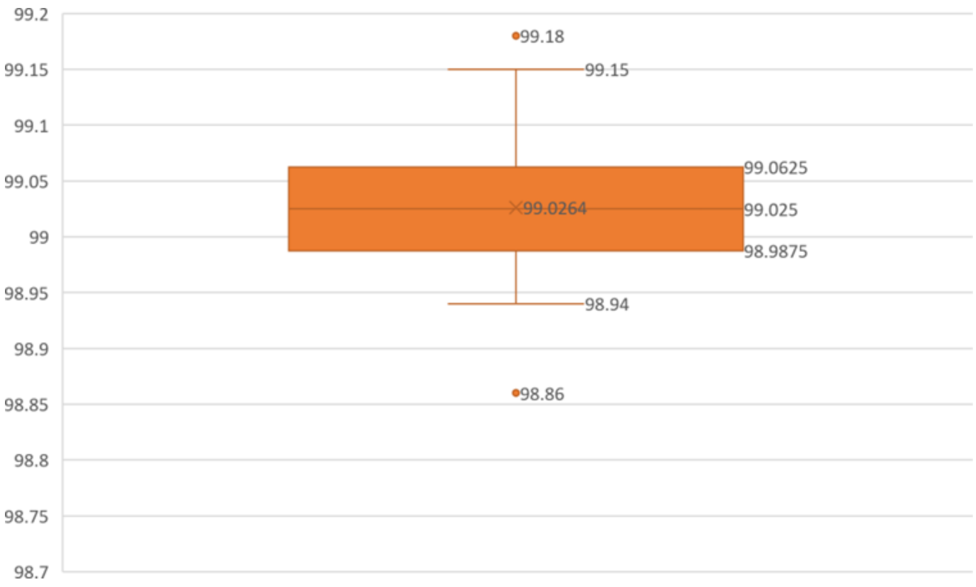


Figure 7.5 Boxplot of the test accuracy distribution of Model A over 50 runs on MNIST dataset with candidate shuffling.

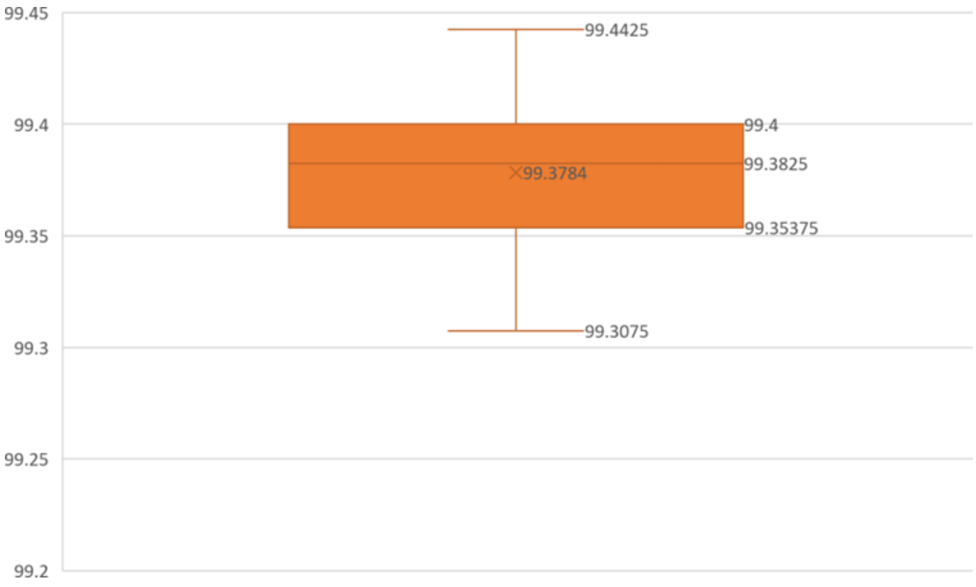


Figure 7.6 Boxplot of the test accuracy distribution of Model A over 50 runs on EMNIST-Digits dataset with candidate shuffling.

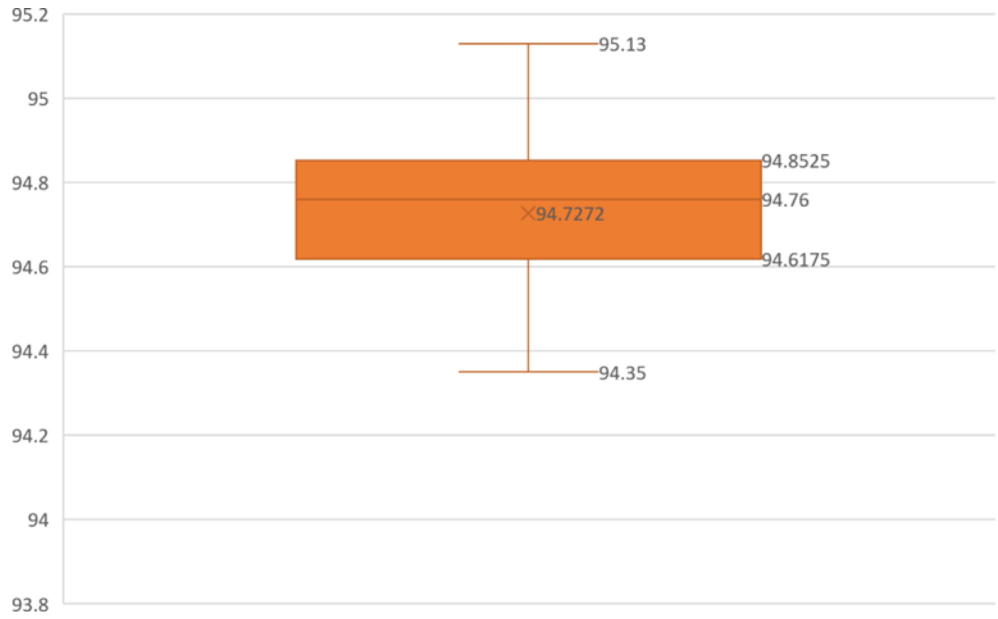


Figure 7.7 Boxplot of the test accuracy distribution of Model B over 50 runs on Kuzushiji-MNIST dataset with candidate shuffling.

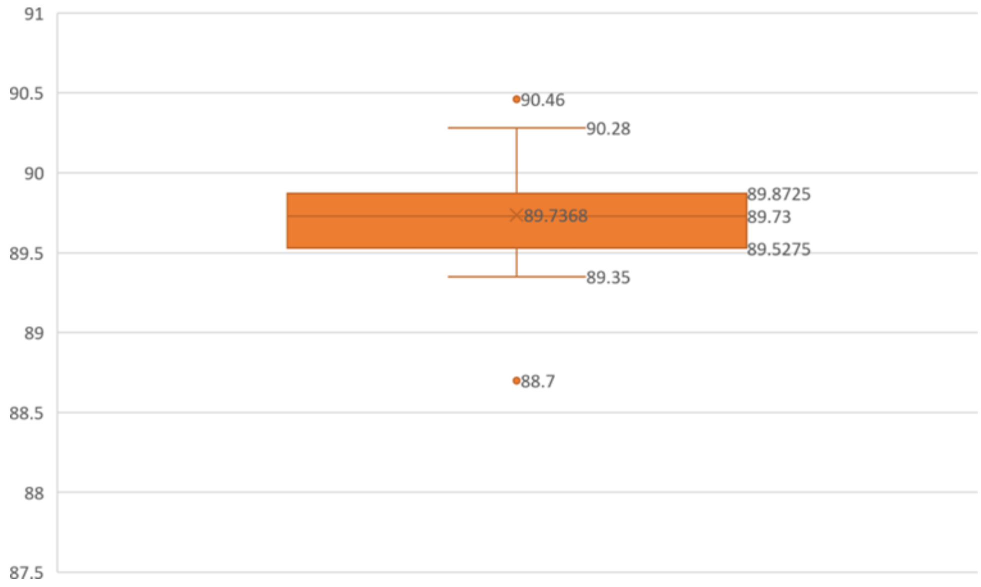


Figure 7.8 Boxplot of the test accuracy distribution of Model B over 50 runs on Fashion MNIST dataset with candidate shuffling.

7.4 Comparison to Low-Capacity CNN

To gauge the performance of a low-capacity CNN against our proposed models, we used lower similarity thresholds of 0.3 and 0.4. With these lower threshold values, the number of filters extracted from the datasets dramatically drops. For MNIST dataset, with 0.4 similarity threshold on Model A produces 23 and 21 filters for the convolutional layers respectively. The test accuracy of the model drops to 98.95%. A CNN model which has the same architecture and parameter count is constructed as the low-capacity CNN and trained in supervised manner with MNIST dataset for 50 epochs. The low-capacity CNN achieved 99.21% test accuracy. When we repeat the same comparison procedure on the other datasets, we again observed that low-capacity CNN to perform 1 - 1.3% better than our method. The self-organization of the filters in the proposed method relies on higher similarity thresholds to capture more distinct features as we have already discussed them in Chapter 6. Thus, it was expected to observe a drop in the classification performance in our models. Low-capacity CNN models we trained seem to benefit from the fully supervised training scheme but still there is not a huge performance margin between the two approaches.

CHAPTER 8

8. CONCLUSION

Convolutional Neural Networks have become an indispensable tool for image classification, computer vision and deep learning tasks in the last decade. This can be attributed to how they are designed. CNN architecture is built around imitating the mammal visual system. This architecture is made up from convolutional layers, pooling layers, and fully connected layers which are developed based on the findings on the visual cortex of mammals. The combination of these layers enabled automatic extraction of the features from inputs to detect patterns in the data which eliminated the need for an expert to extract the features. However, training a CNN relies on the availability of a carefully constructed large labeled dataset because the training process involves the exposure of labeled images to the network, followed by a comparison of its predictions with the ground truth labels which is then used to update the weights of the network via propagating the error signal backward in the network. This process is iteratively repeated for a substantial number of epochs to continuously update the weights until the desired performance is achieved. Thus, the CNN gradually learns from the labeled data via backpropagation of errors to recognize the patterns and to make correct predictions. However, it has not been proved that the brain learns through backpropagation. Even though we build layers based on the visual cortex structures, we rely on an unnatural learning procedure. Backpropagation implies that the individual neurons in the visual cortex must store the input data of the forward pass and then wait for the input data to move through all neurons in the brain so that the connections to other neurons can be updated based on backward pass of the derivatives of the errors.

Even though the supervised training of CNNs is allowing state-of-the-art results, its success depends on the availability of a large labeled dataset. There are a couple of risks related to the usage of labeled datasets. The presence of biased or mislabeled data in labeled datasets is a huge concern in the successful training of the network. If there are mislabeled data in the dataset, the propagated error would be incorrect and this would lead to improper updates on the network parameters, negatively impacting the network performance via erroneous predictions. Similarly, if the dataset contains bias in the representation of the classes, the network could learn the bias between the classes which would lead to biased predictions. Furthermore, gathering a large labeled dataset is a time-consuming and expensive. It is also prone to labeling errors. Another risk that should be mentioned is not having a sufficient number of samples in the labeled dataset. This could easily be a bottleneck for the training of the network which could prevent the network from generalizing well. Gradient based backpropagation algorithm requires large amount of data to effectively train the network. These risks might be alleviated by carefully curating the data to prevent biases or mislabeled data which would require a domain expert. However, it is not easy to deploy domain experts to curate a large labeled dataset due to monetary costs. Moreover, applicability of labels could not be possible in every domain. Thus, it is imperative to search for alternative unsupervised learning paradigms to remove the dependence on labeled data and backpropagation. Recently, Hinton also pointed out that the backpropagation should be replaced with another learning algorithm that is plausible with how the brain works. His alternative approach to backpropagation removes the backward pass of the backpropagation training with two forward passes by using the data labels as positive or negative for weight updates. However, his proposal still uses derivatives and labels. This dissertation introduces an alternative algorithm for unsupervised training of CNNs (Erkoç, Eskil, 2023), specifically targeting the convolutional layers, without relying on backpropagation. In Chapter 4, we proposed our algorithm. The training process of the algorithm involves extracting novel features from a training set and iteratively adjusting their weights, all in a single pass through the training set, without the need for data labels. The entire filter extraction process is unsupervised and does not require backpropagation. Each convolutional layer is assumed empty (i.e., no filters) at the start of the process. The process starts with obtaining a set of filter candidates from a given image dataset. Following the formation of candidates set, the first candidate is selected as the first discovered filter for the current convolutional layer because there are not any filters to compare against. The remaining filters are subsequently identified from the pool of candidates using a similarity metric. The similarity metric is calculated as the dot product of the candidates against the already discovered filters of the current convolutional layer. The calculated similarity is then compared against a similarity threshold which is a value from $[0 - 1]$ which corresponds to not similarity to identical scale. If the calculated similarity is lower than the similarity threshold, then we accept this candidate as a new filter because there is a feature in the candidate that is not similar to any discovered filters. However, if the calculated similarity is higher than the similarity threshold, the candidate is not a new observation. Thus, we find the filter that has the highest similarity score to this candidate and update the weights of this filter with the candidate's. This process is repeated until the algorithm consumes all training images. We showed that the proposed unsupervised algorithm alleviates the need for labeled data to train the convolutional layers.

In supervised approaches, it is imperative to provide the number of filters of a convolutional layer and initialize them appropriately before the training starts. After the initialization of the filters, the backpropagation training for a large number of epochs commences. However, the determined number of filters might not be enough for representing the features in the dataset. In this case, hyperparameter optimization techniques are typically utilized to determine the appropriate number of filters for the convolutional layers. However, our proposed algorithm does not need the number of filters hyperparameter to be set before the training because this hyperparameter value is automatically determined by the filter extraction process. The self-discovery of the filters with the proposed algorithm also eliminates the need to initialize the filter weights with a proper weight initialization method. Furthermore, since the filter weights are not initialized with random values from a distribution, they do not require weight updates through backpropagation on multiple epochs. The proposed algorithm only looks at the images in the dataset once to obtain candidates so multiple forward and backward passes of the same data is no longer required to train the convolutional layers.

The experiments outlined in Chapter 5 reveal encouraging outcomes on diverse datasets without relying on data preprocessing, augmentation, or intricate architectures. These findings highlight the possibility of training convolutional layers using an unsupervised backpropagationless approach, where the training set images are processed in a single pass, eliminating the need for extensive iterations as required by supervised approaches. Moreover, the obtained results are on par with the state-of-the-art achieved through supervised learning, employing a simpler and more straightforward model that is easier to train.

Backpropagation alternatives for training neural networks is a novel research area that is gaining interest. Currently, our proposed algorithm is the only method which both omits data labels and backpropagation to train CNNs. Most of the research in this area is based on the recent forward-forward algorithm by Hinton (Geoffrey E. Hinton, 2022) which is still calculating derivatives based on data labels. We showed that it is possible to train the feature extractor part of the CNN architecture with a backpropagation free approach that does not use any labels. However, there are still open issues that needs to be solved in the future. As a future work, we should investigate how we can improve the classification performance when there are too many intraclass variations. Another future direction that can be investigated is the detection of anomalies in an unsupervised setting. Currently, the proposed algorithm produces good results with grayscale images, and it is an important direction for this research to extend this work to color images. Furthermore, we only applied grid search for similarity threshold optimization. It would be interesting to investigate the impact of hyperparameter optimization on hyperparameters like *filter_size* and number of layers over the proposed CCNN architecture.

REFERENCES

Agarap, A. F., Azcarraga, A. P. (2020) Improving k-means clustering performance with disentangled internal representations. In *2020 international joint conference on neural networks (ijcnn)* (pp. 1-8) doi: 10.1109/IJCNN48605.2020.9207192.

Akı, K. K. E., Erkoc¸, T., Eskil, M. T. (2017). Subset selection for tuning of hyper-parameters in artificial neural networks. In *2017 24th ieee international conference on electronics, circuits and systems (icecs)* (pp. 144–147).

Albus, J. S. (1971). A theory of cerebellar function. *Mathematical biosciences*, *10*(1-2), 25–61.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7700* LECTU, 437–478. doi:10.1007/978-3-642-35289-8-26. arXiv: 1206.5533

Byerly, A., Kalganova, T., Dear, I. (2021). No routing needed between capsules. *Neurocomputing, 463*, 74–80. doi:10.1016/j.neucom.2021.08.064

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In D. D. Lee, U. Luxburg, R. Garnett, M. Sugiyama, and I. Guyon (Eds.), *Nips'16: Proc. 30th int. conf. neural inf. process. syst.* (pp. 2180– 2188). Red Hook, NY, USA: Curran Associates.

Ciresan, D., Meier, U., Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *proceedings of the 25th ieee conference on computer vision and pattern recognition (cvpr)* (pp. 3642–3649).

Cireşan, D. C., Meier, U., Gambardella, L. M., Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation, 22*(12), 3207–3220. doi:10.1162/neco_a_00052

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the twenty-second international joint conference on artificial intelligence – volume two* (pp. 1237-1242). Barcelona, Catalonia, Spain: AAAI.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., Ha, D. (2018). Deep learning for classical japanese literature. CoRR, abs/1812.01718.arXiV:1812.01718.

Cohen, G., Afshar, S., Tapson, J., van Schaik, A. (2017). Emnist: Extending mnist to handwritten letters. In *2017 int. joint conf. neural netw. (ijcnn)* (pp. 2921–2926). Anchorage, AK, USA: IEEE.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 2933–2941). Curran Associates.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255).doi:10.1109/CVPR.2009.5206848

Denker, J., R. Gardner, W., Graf, H., Henderson, D., E. Howard, R., Hubbard, W., Jackel, L. D., Baird, H, Guyon, I. (1988). Neural network recognizer for hand-written zip code digits. In D. Touretzky (Ed.), *Advances in neural information processing systems, 1* (pp. 323–331), Morgan-Kaufmann.

Doersch, C., Gupta, A., Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In L. O'Conner (Ed.), *Proc. 2015 ieee int. conf. comput. vis. (iccv)* (pp. 1422–1430). Los Alamitos, CA, USA: IEEE Comput. Soc.

Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger (Eds.), *Proc. 27th int. conf. neural inf. process. syst., 1*, (pp. 766–774). Cambridge, MA, USA: MIT.

Duchi, J., Hazan, E., Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research, 12*(61), 2121–2159.

Erkoç, T., Eskil, M. T. (2022). Unsupervised similarity based convolutions for hand-written digit classification. In *2022 30th signal process. commun. appl. conf. (siu)* (pp. 1–4). doi:10.1109/SIU55565.2022.9864689

Erkoç, T., Eskil, M. T. (2023). A novel similarity based unsupervised technique for training convolutional filters. *IEEE Access, 11*, 49393–49408. doi:10.1109/ACCESS.2023.3277253

Foret, P., Kleiner, A., Mobahi, H., Neyshabur, B. (2021). Sharpness-aware minimization for efficiently improving generalization. Paper presented at the meeting of 10th int. conf. learn. representations (iclr 2021), Vienna, Austria.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics, 36*, 193–202.

Fukushima, K. (2003). Neocognitron for handwritten digit recognition. *Neurocomputing, 51*, 161–180. doi:10.1016/S0925-2312(02)00614-8

Fukushima, K. (2013). Training multi-layered neural network neocognitron. *Neural Networks, 40*, 18–31. doi:10.1016/j.neunet.2013.01.001

Fukushima, K. (2016). Margined winner-take-all: New learning rule for pattern recognition. In *2016 international joint conference on neural networks (ijcnn)* (pp. 977–984). doi:10.1109/IJCNN.2016.7727304

Fukushima, K., Hayashi, I., Léveillé, J. (2014). Neocognitron trained by winner-kill-loser with triple threshold. *Neurocomputing, 129,* 78–84. doi:10.1016/j.neucom. 2012.05.038

Fukushima, K., Wake, N. (1991). Handwritten alphanumeric character recognition by the neocognitron. *IEEE Transactions on Neural Networks, 2*(3), 355–365. doi:10.1109/72.97912

Gastaldi, X. (2017). Shake-shake regularization. CoRR, abs/1705.07485. arXiv: 1705.07485.

Gidaris, S., Singh, P., Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations. Paper presented at the meeting of 6th int. conf. learn. representations (iclr 2018), Vancouver, Canada.

Glorot, X., Bengio, Y. (2010). Understanding the difficulty of training deep feed-forward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), 9,* 249–256. doi:10.1.1.207.2059

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Bengio, Y. (2014). Generative adversarial nets. In Z. GhahramaniIn Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Q. Weinberger (Eds.), Nips’14: *Proc. 27th int. conf. neural inf. process. syst., 2,* pp. 2672–2680). Cambridge, MA, USA:MIT

He, K., Zhang, X., Ren, S., Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 ieee international conference on computer vision (iccv)* (pp. 1026–1034). doi:10. 1109/ICCV.2015.123

He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. *In 2016 ieee conference on computer vision and pattern recognition (cvpr)* (pp. 770–778). doi:10.1109/CVPR.2016.90

Hebb, D. (1949). The Organization of Behavior. A neuropsychological theory. The *Organization of Behavior, 911*(1), 335. doi:10.2307/1418888

Hinton, G. E., Dayan, P., J Frey, B., M Neal, R. (1995) The “wake-sleep” algorithm for unsupervised neural networks. *Science, 268,* 1158–61. doi:10.1126/science.7761831

Hinton, G. E. (2002) Training products of experts by minimizing contrastive divergence. *Neural Computation, 14*(8), 1771–1800. doi:10 . 1162 / 089976602760128018

Hinton, G. E. (2010) A Practical Guide to Training Restricted Boltzmann Machines A Practical Guide to Training Restricted Boltzmann Machines. *Computer, 9*(3), 1. doi:10.1007/978-3-642-35289-8 32

Hinton, G. E. (2022). The forward-forward algorithm: Some preliminary investigations. doi:10.48550/ARXIV.2212.13345

Hinton, G. E., Osindero, S., Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*(7), 1527–1554. doi:10.1162/ neco.2006.18.7.1527

Hochreiter, J. (1991). Untersuchungen zu dynamischen neuronalen Netzen, (Doctoral dissertation), Institut für Informatik, Tech. Univ. Munich, Munich.

Hou, B., Yan, R. (2018). Convolutional auto-encoder based deep feature learning for finger-vein verification. *In 2018 ieee int. symp. med. meas. appl. (memea)* (pp. 1–5). doi:10.1109/MeMeA.2018.8438719

Hubel, D. H., Wiesel, T. N. (1959). Receptive fields of single neurones in the cat’s striate cortex. *Journal of Physiology, 148,* 574–591. doi:10.1113/jphysiol.2009. 174151

Hutter, F. (2009). Automated Configuration of Algorithms for Solving Hard Computational Problems (Doctoral dissertation), The Faculty of Graduate Studies, The University Of British Columbia, Vancouver. doi:10.14288/1.0051652.

Ito, M., Sakurai, M., Tongroach, P. (1982). Climbing fibre induced depression of both mossy fibre responsiveness and glutamate sensitivity of cerebellar purkinje cells. *The Journal of Physiology, 324*(1), 113–134.

Jaehoon, C., Kim, Y., Jung, H., Oh, C., Youn, J., Sohn, K. (2018). Multi-task self-supervised visual representation learning for monocular road segmentation. In L. O’Conner (Ed.), *2018 ieee int. conf. multimedia expo (icme)* (pp. 1–6). Los Alamitos, CA, USA: IEEE Comput. Soc.

Kaneko, H., Funatsu, K. (2015). Fast optimization of hyperparameters for support vector regression models with highly predictive ability. *Chemometrics and Intelligent Laboratory Systems, 142,* 64–69. doi:10.1016/j.chemolab.2015.01.001

Keerthi, S. S., Lin, C.-J. (2003). Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel. *Neural Computation, 15*(7), 1667–1689. doi:10 . 1162 / 089976603321891855

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Technical Report TR-2009*, University of Toronto, Toronto.

Krizhevsky, A., Sutskever, I., Geoffrey E., H. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25* (NIPS2012), 1–9. doi:10.1109/5.726791. arXiv: 1102.0183

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. *Proceedings of the 24th international conference on Machine learning - ICML '07*, (2006), 473–480. doi:10.1145/1273496.1273556

Larsson, G., Maire, M., Shakhnarovich, G. (2016). Learning representations for automatic colorization. In B. Leibe, J. Matas, N. Sebe, and M. Welling (Eds.), *Comput. vis. - eccv 2016 Vol. LNCS 9910,* (pp. 577–593). Cham: Springer International.

Larsson, G., Maire, M., Shakhnarovich, G. (2017). Colorization as a proxy task for visual understanding. In L. O'Conner (Ed.), *2017 ieee conf. comput. vis. pattern recognit. (cvpr)* (pp. 840–849). Los Alamitos, CA, USA: IEEE Comput. Soc.

LeCun, Y., Bengio, Y., Hinton, G. E. (2015). Deep learning. *Nature, 521*, 436–44. doi:10.1038/nature14539

Lecun, Y., Boser, B., Denker, J., Henderson, D., E. Howard, R., Hubbard, W., Jackel, L. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation, 1,* 541–551. doi:10.1162/neco.1989.1.4.541

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2323. doi:10.1109/5.726791. arXiv: 1102.0183

LeCun, Y., Cortes, C. (2010). MNIST handwritten digit database. [Dataset] Retrieved from http://yann.lecun.com/exdb/mnist/

Liao, R., Schwing, A., Zemel, R., Urtasun, R. (2016). Learning deep parsimonious representations. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, R. Garnett (Eds.), *Nips'16: Proc. 30th int. conf. neural inf. process. syst.* (pp. 5083–5091). Red Hook, NY, USA: Curran Associates Inc.

Liu, X., Weijer, J., Bagdanov, A. D. (2019). Exploiting unlabeled data in cnns by self-supervised learning to rank. *IEEE Trans. Pattern Anal. and Mach. Intell., 41*, 1862–1878. doi:10.1109/TPAMI.2019.2899857

Lomo, T. (1966). Frequency potentiation of excitatory synaptic activity in dentate area of hippocampal formation. *In Acta physiologica scandinavica* (p. 128). Blackwell Science; Oxford, UK.

Maas, A. L., Hannun, A. Y., Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. Paper presented at the meeting of icml workshop on deep learning for audio, speech and language processing, Atlanta, Georgia, USA.

Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *5th berkeley symp. math. statist. probability* (pp. 281–297).

Mahon, L., Lukasiewicz, T. (2021). Selective pseudo-label clustering. In Edelkamp, S., Möller, R. Rueckert, E. (Eds.), *Ki 2021: Advances in artif. intell. Vol. LNAI 12873*, (pp. 158–178). Cham: Springer International.

Makhzani, A., Frey, B. J. (2014). K-sparse autoencoders. In Bengio, Y. and LeCun, Y. (Eds.), *2nd int. conf. learn. representations, iclr 2014, banff, ab, canada, april 14-16, 2014, conf. track proc.* Retrieved from http://arxiv.org/abs/1312.5663

Masci, J., Meier, U., Cires¸an, D., Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In Honkela, T., Duch, W., Girolami, M., and Kaski, S. (Eds.), *Artif. neural netw. mach. learn. – icann 2011 Vol. LNCS 6791*, (pp. 52–59). Berlin, Heidelberg: Springer Berlin Heidelberg.

McCulloch, W. S., Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. doi:10. 1007/BF02478259. arXiv: arXiv:1011.1669v3

Minsky, M., Papert, S. (1969). Perceptrons: An introduction to computational geometry. Cambridge, MA, USA: MIT.

Misra, I., Zitnick, C. L., Hebert, M. (2016). Shuffle and learn: Unsupervised learning using temporal order verification. In Leibe, B., Matas, J., Sebe, N., and Welling, M. (Eds.), *Comput. vis. - eccv 2016 Vol. LNCS 9910*, (pp. 527–544). Cham: Springer International.

Nair, V., E. Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *In Proceedings of ICML 27,* 807–814.

Noroozi, M., Favaro, P. (2016). Unsupervised learning of visual representations by solving jigsaw puzzles. In Leibe, B., Matas, J., Sebe, N., and Welling, M. (Eds.), *Comput. vis. - eccv 2016 Vol. LNCS 9910*, (pp. 69–84). Cham: Springer Inter- national.

Pad, P., Narduzzi, S., Kündig, C., Türetken, E., Bigdeli, S. A., Dunbar, L. A. (2020). Efficient neural vision systems based on convolutional image acquisi- tion. In O'Conner, L. (Ed.), *2020 ieee/cvf conf. comput. vis. pattern recognit.* (cvpr) (pp. 12282–12291). doi:10.1109/CVPR42600.2020.01230

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In O'Conner, L. (Ed.), *2016 ieee conf. comput. vis. pattern recognit. (cvpr)* (pp. 2536–2544). Los Alamitos, CA, USA: IEEE Comput. Soc.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, *12*(1), 145–151. doi:https:// doi. org/ 10 . 1016 / S0893 - 6080(98) 00116-6

Radford, A., Metz, L., Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In Bengio, Y. and LeCun, Y. (Eds.), *4th int. conf. learn. representations, iclr 2016, san juan, puerto rico, may 2-4, 2016, conf. track proc.* Retrieved from http://arxiv.org/abs/1511.06434

Ren, Z., Lee, Y. (2018). Cross-domain self-supervised multi-task feature learning using synthetic imagery. In *2018 ieee/cvf conf. comput. vis. pattern recognit. (cvpr)* (pp. 762–771). Los Alamitos, CA, USA: IEEE Comput. Soc.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386–408. doi:10 . 1037/h0042519

Rumelhard, D., Zipser, D. (1985). Feature discovery by competitive learning. *Cognitive Science, 9*, 75–112. doi:10.1016/S0010-4825(96)00018-2

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533–536. doi:10.1038/323533a0. arXiv: arXiv:1011.1669v3

Simonyan, K., Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556. arXiv: 1409 . 1556. Retrieved from http://arxiv.org/abs/1409.1556

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing Explorations in the Microstructure of Cognition, 1*(1), 194–281.

Snoek, J., Larochelle, H., Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *In Proceedings of the 25th international conference on neural information processing systems* (pp. 2951–2959). Lake Tahoe, Nevada: Curran Associates.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*(56), 1929–1958.

Srivastava, R. K., Greff, K., Schmidhuber, J. (2015a). Highway networks. arXiv: 1505.00387

Srivastava, R. K., Greff, K., Schmidhuber, J. (2015b). Training very deep networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M. and Garnett, R. (Eds.), *Advances in neural information processing systems 28* (pp. 2377–2385). Curran Associates.

Sutskever, I., Martens, J., Dahl, G., Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. In Dasgupta, S. McAllester, D.

(Eds.), *Proceedings of the 30th international conference on machine learning, 28*, (pp. 1139–1147). Atlanta, Georgia, USA: PMLR.

Theano Development Team. (2016). Theano: A python framework for fast computation of mathematical expressions. CoRR, abs/1605.02688. arXiv: 1605.02688

Tissera, D., Kahatapitiya, K., Wijesinghe, R., Fernando, S., Rodrigo, R. (2019). Context-aware multipath networks. Preprint at http://arxiv.org/abs/1907.11519.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research, 11*(110), 3371–3408.

Wan, L., Zeiler, M., Zhang, S., LeCun, Y., Fergus, R. (2013). Regularization of neural networks using dropconnect. *Icml, 1*, 109–111. arXiv: 1509.08985

Wang, X., He, K., Gupta, A. K. (2017). Transitive invariance for self-supervised visual representation learning. In L. O'Conner (Ed.), *2017 ieee int. conf. comput. vis. (iccv)* (pp. 1338–1347). Los Alamitos, CA, USA: IEEE Comput. Soc.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences, Ph.D. Dissertation, Harvard University, Cambridge.

Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. In Drenick, R. F. Kozin, F. (Eds.), *System modeling and optimization* (pp. 762–770). Berlin, Heidelberg: Springer Berlin Heidelberg.

Xiao, H., Rasul, K., Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. CoRR, abs/1708.07747. arXiv: 1708.07747. Retrieved from https://arxiv.org/abs/ 1708.07747.

Xie, J., Girshick, R., Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In Balcan, M. F. and Weinberger, K. Q. (Eds.), *Proc. 33rd int. conf. mach. learn., 48,* (pp. 478–487). New York, NY, USA: PMLR.

Xu, Y., McCord, R. (2021). Costa: Unsupervised convolutional neural network learning for spatial transcriptomics analysis. *BMC Bioinformatics, 22*. Article number: 397. doi:10.1186/s12859-021-04314-1

Yan, X., Misra, I., Gupta, A., Ghadiyaram, D., Mahajan, D. (2020). Clusterfit: Improving generalization of visual representations. In *2020 ieee/cvf conf. comput. vis. pattern recognit. (cvpr)* (pp. 6508–6517). Los Alamitos, CA, USA: IEEE Comput. Soc.

Yang, J., Parikh, D., Batra, D. (2016). Joint unsupervised learning of deep representations and image clusters. In *2016 ieee conf. comput. vis. pattern recognit. (cvpr)* (pp. 5147–5156). Los Alamitos, CA, USA: IEEE Comput. Soc.

Yann, L., Bottou, L., Orr, G. B., Müller, K.-R. (1998). Efficient BackProp. In: Montavon, G., Orr, G. B., Müller, K. R. (eds) Neural Networks: Tricks of the Trade. *Lecture Notes in Computer Science, vol 7700,* pp 9–48. Springer, Berlin, Heidelberg.

Zagoruyko, S., Komodakis, N. (2016). Wide residual networks. Wide residual networks. In Richard, E. R. H., Wilson, C. and Smith, W. A. P. (Eds.), *Proceedings of the british machine vision conference (bmvc)* (pp. 87.1–87.12).

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. CoRR, abs/1212.5701. arXiv: 1212.5701. Retrieved from http://arxiv.org/abs/1212.5701

Zeiler, M. D., Fergus, R. (2014). Visualizing and understanding convolutional networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8689* LNCS(PART 1), 818–833. doi:10.1007/978-3-319-10590-1 53. arXiv: 1311.2901

Zhang, R., Isola, P., Efros, A. A. (2016). Colorful image colorization. In B. Leibe, B., Matas, J., Sebe, N. Welling, M. (Eds.), *Comput. vis. - eccv 2016 Vol. LNCS 9910,* (pp. 649–666). Cham: Springer International

CURRICULUM VITAE