**DOKUZ EYLÜL UNIVERSITY**
**GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**

# HANDWRITTEN CHARACTER RECOGNITION AND DOCUMENT ANALYSIS USING DEEP NEURAL NETWORKS

**by**
**Barış KILIÇLAR**

**February, 2020**
**İZMİR**

# HANDWRITTEN CHARACTER RECOGNITION AND DOCUMENT ANALYSIS USING DEEP NEURAL NETWORKS

**A Thesis Submitted to the**
**Graduate School of Natural and Applied Sciences of Dokuz Eylül University**
**In Partial Fulfillment of the Requirements for the Degree of Master of Science**
**in Electrical and Electronics Engineering**

**by**

**Barış KILIÇLAR**

**February, 2020**
**İZMİR**

## M.Sc. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"HANDWRITTEN CHARACTER RECOGNITION AND DOCUMENT ANALYSIS USING DEEP NEURAL NETWORKS"** completed by **BARIŞ KILIÇLAR** under supervision of **ASST. PROF. DR. METEHAN MAKİNACI** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

.....

Asst. Prof. Dr. Metehan MAKİNACI

Supervisor

.....

Prof. Dr. Mehmet Kuntalp

(Jury Member)

.....

Asst. Prof. Dr. Nalan Özkurt

(Jury Member)

Prof. Dr. Kadriye ERTEKİN

Director

Graduate School of Natural and Applied Sciences

**ACKNOWLEDGEMENTS**

I would like to thank to Asst. Prof. Dr. Metehan MAKİNACI for all his valuable supervision and kind guiding as well as his comprehensive mastership to me throughout all this work. Thanks to all Dokuz Eylül University Electrical and Electronics Engineering former and actual academic staff, they are so helpful and sincere.

Thanks to my wife Tuba KILIÇLAR who support me devotedly. Thanks to my family and my mother who i owe forever.

Barış KILIÇLAR

**HANDWRITTEN CHARACTER RECOGNITION AND DOCUMENT ANALYSIS USING DEEP NEURAL NETWORKS**

## ABSTRACT

In this work, an application of deep learning techniques to recognize handwritten source code characters is presented. Although there are many works on the handwritten character recognition (HCR) problem, very few have been done about the offline handwritten source code character recognition. The problem includes the recognition of source code specific characters. An application designed and implemented, performing preprocessing, histogram based segmentation and normalization on the scanned documents of exam papers which include codes that were written in C programming language. Constructed dataset includes 7093 source code character samples. This dataset was enriched with character samples from the CROHME database by transforming them to offline samples. With resulting 95 classes of 17748 samples, several models of Convolutional Neural Networks (CNNs) were trained and tested. CNN is a deep learning architecture which is shown to produce state-of-the-art performance rates for handwritten character recognition tasks as in various other computer vision applications. Experimental evaluations gave performance rates between 92.33 percent and 98.82 percent. We conclude that CNN based classifiers are powerful tools for *recognition of handwritten source code characters* task.

**Keywords:** Handwritten character recognition, deep neural networks, convolutional neural networks, deep learning, handwritten source code classification

# DERİN SİNİR AĞLARI KULLANARAK EL YAZISI ALGILAMA VE BELGE ANALİZİ

## ÖZ

Bu çalışmada el yazısı kaynak kodu karakterlerini tanımak için derin öğrenme tekniklerinin bir uygulaması sunulmuştur. Her ne kadar el yazısı karakter tanıma problemi üzerine bir çok çalışma olsa da, çevrim dışı el yazısı kaynak kodu karakteri tanımada çok az çalışma bulunmaktadır. Problem kaynak koduna özel karakterleri tanımayı içermektedir. C programa dilinde yazılmış kodlar içeren sınav kağıtları üzerinde önişleme, histogram tabanlı bölütleme ve normalizasyon yapan bir uygulama tasarlanmış ve gerçeklenmiştir. Oluşturulan veri kümesi 7093 kaynak kodu karakter örneği içermektedir. Bu veri kümesi CROHME veri tabanından dönüştürülen çevrim dışı örneklerle zenginleştirilmiştir. Sonuçta oluşan 95 sınıflı 17748 örnek ile bazı Evrişimsel Sinir Ağı (ESA) modelleri eğitildi ve test edildi. ESA el yazısı karakter tanıma görevlerinde en ileri gelişmeleri yansıtan bir derin öğrenme mimarisidir. Deneyler yüzde 92,33 ilâ yüzde 98,82 arasında başarı dereceleri vermiştir. Bununla, CNN tabanlı sınıflandırıcıların *el yazısı kaynak kodu tanıma*da güçlü araçlar olduğu sonucuna varıyoruz.

**Anahtar kelimeler:** El yazısı karakter tanıma, derin sinir ağları, evrişimsel sinir ağları, derin öğrenme, el yazısı kaynak kodu sınıflandırma

**CONTENTS**

**Page**

## LIST OF FIGURES

**Page**

## LIST OF TABLES

**Page**

# CHAPTER ONE
# INTRODUCTION

## 1.1 Definition of the Field and Its Importance

Handwriting is still being the one of the most used, natural and convenient way for collecting, storing and transmitting the information. Automating the translation of handwriting document into the digital environment obviously has many useful applications. They include automatizing processes that involves handwritten document analysis, such as postal code handling, bank check reading, and digitally storing historical documents.

The term handwriting character recognition (HCR) is used for the progression from the document image analysis to content analysis. Analyzed document which contains handwritten source code can be used as input for computers. This method can have some advantages and convenience for certain circumstances over the techniques solely digital, like keyboard input.

## 1.2 Glance at the HCR

Although extensive amount of research has been made at recent years, handwritten character recognition field still needs to be improved (Arica & Vural, 2001). It is much challenging because there are so many factors affecting the performance such as: different persons have different handwriting characteristics, even one person use different forms in different contexts, correlation of segmentation in preprocessing with the resulting recognition performance. If not done properly, preprocessing can cause the loss of the localities such as contextual and positional information. It is also in effect that the field has many sub-fields regarding the parameters that are intrinsic on the field: on-line, off-line handwriting recognition, different contexts (postal cards, bank checks, street numbers, mathematical expressions, source codes, etc.), constrained or unconstrained handwriting, recognition of different characters for different languages.

Handwriting recognition field history dates back to 1960s (Zhi & Metoyer, 2017). For recognition, various classifiers are proposed in the basic application of pattern

recognition since that time such as Support Vector Machine (SVM) (Shruthi & Patel, 2015; Singh, Dhir, & Rani, 2011), Multilayer Perceptron (MLP) (Hussain & Kabuka, 1994), K Nearest Neighbor (KNN) (Singh, Dhir, & Rani, 2011), Self Organizing Map (SOM) (Liou & Yang, 1996) and Hidden Markov Model (HMM) (Chen, Kundu, & Zhou, 1994; El-Yacoubi, Gilloux, Sabourin, & Suen, 1999). Amongst others, deep learning techniques gained popularity because Convolutional Neural Networks (CNNs) presented outstanding success rates recently and as a result, CNNs are agreed to be the state of the art for HCR applications.

### 1.3 Material Description

We studied on the handwritten character recognition of scanned exam papers answered by the students of the Department of Electrical and Electronics Engineering at Dokuz Eylül University. Exam papers contain source codes which are written in C programming language. Documents are partially constrained since the exam papers are outlined by red guide lines. Although the name field has a grid of guide lines, answer field has only line guides. In this field, handwritten source code in C programming language is written line by line. Guidelines support segmentation methods in giving more accurate results.

There are many works done in the field of HCR (Cireşan, Meier, & Schmidhuber, 2012; Trier, Jain, & Taxt, 1996), but very few progresses have been made regarding the *handwritten source code character* recognition. To the best of our knowledge, only few papers have been published applying convolutional neural networks to the task of handwritten *source code character* recognition (Zhi & Metoyer, 2017). We extracted character images and constructed a dataset for this task and obtained results with different CNN models for comparison. We also compared our results to other systems; performance comparisons propound that CNN based classifiers are powerful tools for *recognition of handwritten source code characters* task.

## 1.4 Methodology, Reason of Choice and Previous Works on the Field

Examining the previous works done in the field of HCR, we can see that the processes can be grouped in phases as: preprocessing, segmentation, representation, training and recognition and postprocessing. Not all works implement all of these phases, some of them might be merged, omitted and some other phases might be added.

In training and recognition phase, mostly feature extraction and classification is used. Feature extraction is used to get a prominent representation of data to make classification performance high. Usually feature extraction process done manually because of the data variety between different problems. After getting fairly promising representation, classification process takes place which uses a different method than the feature extraction. Lately a different type of approach is increasingly used which integrates feature extraction and recognition phases into one process automatically (Palehai & Fanany, 2017). This is called as deep learning techniques and developed based on neural networks.

One of the architecture that implements deep learning techniques is convolutional neural networks (CNNs). They can extract hierarchical features from input data, automatically (Palehai & Fanany, 2017). In this work we utilized this competence of the CNN architecture. In training phase, document images after preprocessing phase are fed into the segmentation stage to extract character images. Without any feature extraction, these images along with the labels are directly used for training of the CNN with deep learning algorithm in a supervised learning manner. Test phase use the same scheme for character extraction. Then extracted character images are submitted to the CNN classifier which is trained in the training phase. Character labels are used to measure the performance of the system by checking the match with the classifier's output. Training and testing phases are shown in Figure 1.1.

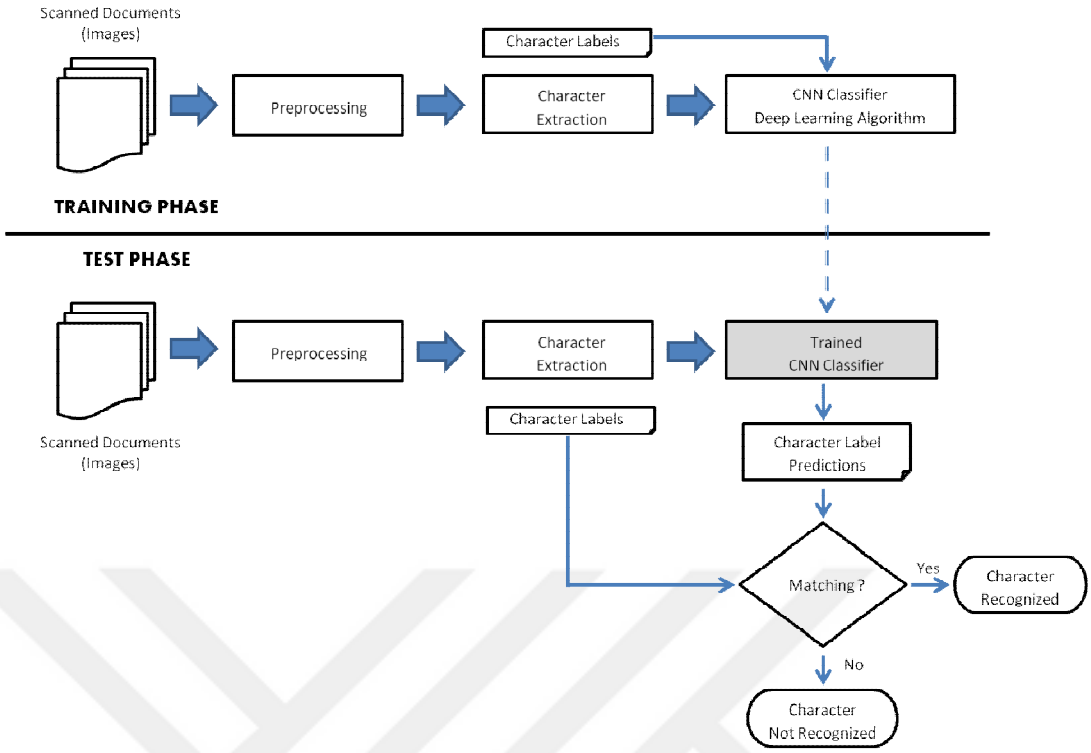



Figure 1.1 Used methodology for character recognition task with a deep learning architecture: Training and test phases

As in the field of other computer vision problems, such as 3D object recognition (Cireşan, Meier, Masci, Gambardella, & Schmidhuber, 2011b; LeCun, Huang, & Bottou, 2004), natural images (LeCun, Huang, & Bottou, 2004) and traffic signs recognition (Cireşan, Meier, Masci, & Schmidhuber, 2011), image denoising (Lefkimmiatis, 2017; Tian, et.al., 2019) and image segmentation (Chen, Papandreou, Kokkinos, Murphy, Yuille, 2014; Milletari, Navab, & Ahmadi, 2016), CNNs are also proven to be the most suitable method for HCR applications. Work of Simard, Steinkraus, & Platt (2003) is an example of this, which is done on the MNIST handwritten character dataset with 60000 samples of 10 classes for training and 10000 samples for test. Elastic image deformations was used to enhance the training dataset size, achieved 0.4% error rate using a fairly simple CNN architecture. Another and more recent work (Cireşan, Meier, Gambardella, & Schmidhuber, 2011) reported 0.27 ± 0.02% error rate on MNIST dataset and, 11.88% error rate on NIST SD 19 dataset of 62 classes (upper- and lower-case letters and digits), using committees of CNNs. Furthermore, some of the researchers successfully used

graphic cards (GPU) and CUDA code to reduce training time by a factor of 60 (Cireşan, Meier, Masci, Gambardella, & Schmidhuber, 2011b) over a compiler-optimized CPU version, which greatly increased time and energy efficiency of the process.

However, works on HCR were done mostly for plain texts. Some recent researches proposed for recognition of handwritten Mathematical Expressions (ME) (Lu & Mohan, 2015). In this work we used programming exam papers which contain source code written in C programming language as in the Figure 1.2. Exam papers guidelined on character based in the first part for writing ID info and line based in the second part for writing the source code. Thus, we can define this application as semi-constrained offline handwritten source code recognition. This application exhibits dissimilarities with the works about plain text and mathematical expressions. CNNs are used as an HCR tool on this application because of above mentioned benefits.

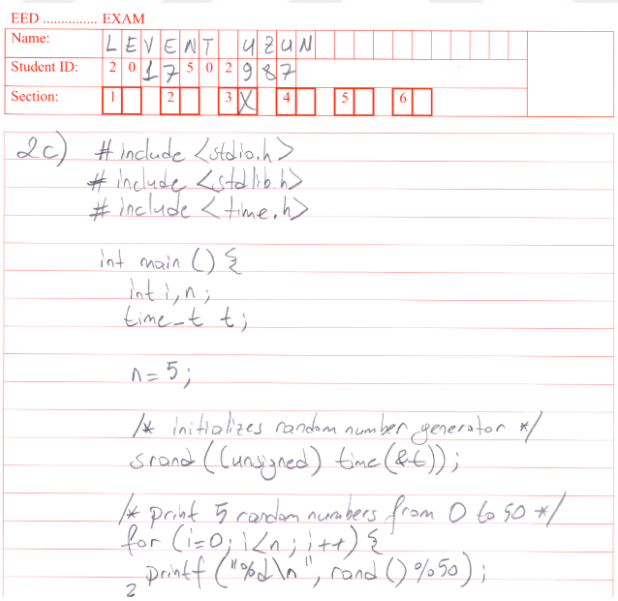



Figure 1.2 An example of analyzed documents (Student name is fictional)

In the preprocessing phase, filtering and orientation correction is applied on the document images. Then segmentation is carried out by making use of guidelines with histogram based approach (Santos, Clemente, Ren, & Cavalcanti, 2009) and

connected components (Lu & Shridhar, 1996; Pal & Datta, 2003; Thungamani & Kumar, 2012). After that point, we got dataset representations using normalization techniques on this segmented data. With this resulting labeled dataset we trained convolutional neural networks. CNN structure also used for classification by fully connected layer and final softmax layer. Test results for different CNN topologies are evaluated. Performance comparisons with the other works are presented at the "3.2.3 Test Results and Comparisons" section.

# CHAPTER TWO
# THEORETICAL BACKGROUND AND RELATED WORK

Handwritten character recognition is generally divided into major phases. According to the characteristics that the application bears, steps may differ in implementation or even may be omitted or populated with extra processes. Phases can be generalized as (Arica & Vural, 2001):

- Preprocessing
- Segmentation
- Representation
- Training and recognition
- Post processing

## 2.1 Preprocessing

Extracting discriminative features from data usually needs uniform format to process properly. For this purpose, various transformations are applied on the raw input data. These transformations are called as preprocessing and this is preliminary and basic step of almost every pattern recognition system (Rashid, 2014).

Preprocessing phase is carried on for the following purposes in general:

- Noise reduction
- Normalization of the data
- Compression of the data

These purposes can be achieved by the following techniques:

### *2.1.1 Noise Reduction*

In offline handwritten recognition applications, data acquisition is mainly performed by optical scanning devices. This technique may introduce noise in the form of disconnected line segments, distortions, spurious points and artifacts. These imperfections can reduce the performance of the training and recognition phase, so needs to be eliminated beforehand (Arica & Vural, 2001).

For this purpose, filtering or morphological operations can be used or noise can be removed by modeling it.

*Filtering:* In general, noise is a natural part of the system and it can be created in various ways. Some artifacts can also be created by the writer's unnecessary or unrelated pen strokes. Sometimes inadequately erased portion of script causes these kinds of artifacts. Noise and artifact elimination can be done with filtering techniques such as Gaussian filtering, median filtering, bilateral filtering etc.

### *2.1.2 Normalization*

Training and recognition phase works at best performance if the variations of data could be kept minimum. For handwriting, these variations mainly are size, slope and slant of the writing. Slope is the deviation angle of the baseline of the handwritten textline from the horizontal direction. Slant is the deviation angle of the downward strokes of handwritten character (as in the *"l"* and *"h"* letters) from the vertical direction. One of the methods to remove or reduce these variations is normalization (Vinciarelli, 2002). In present work, since there are guidelines on the document to be processed, we didn't need this normalization. Although slant normalization is useful for segmentation, we didn't use it because the recognition system we employed (convolutional neural network) is sufficiently immune to the slant of characters.

Size normalization is used to make the handwritten characters sizes the same, i.e., the same with which the network's input layer is designed for. Characters are resized so as to preserve their initial proportion. Extra padding used when the character's proportion is not a square and for the small sized characters which are not size-normalized.

### *2.1.3 Compression*

Compression is used mainly to reduce data volume to be processed and to increase processing speed. Although classical image compression techniques are used very common for volume reduction, it is well known that they transform the

image such that recognition performance may decrease because they originally were not designed to concern to preserve discriminative information. Preserving shape information is main requirement of a compression technique that would be used for recognition. Thresholding and thinning are generally used compression techniques which can be mentioned in this sense (Arica & Vural, 2001). Since the processing speed and storage capability of today's average computer is fairly sufficient for the present application, we didn't employ any compression technique in the present work.

### 2.2 Segmentation

Preprocessing yields low noise normalized data with the preserved information that will be used by the following processes. The next phase is the segmentation. In HCR applications, segmentation phase's ultimate goal is to produce properly isolated character images to be submitted for training and recognition phase. This can be done by an algorithm which aims to determine all the pixels belonging to the same character. Segmentation is an important process since it affects the recognition rate of the document directly.

Although many methods have been proposed, segmentation still remains as an unsolved problem (Arica & Vural, 2001; Santos, Clemente, Ren, & Cavalcanti, 2009). Character segmentation strategies can be viewed in three categories such as: Explicit segmentation (pure segmentation), implicit segmentation and segmentation free (holistic) strategies (Choudhary, Rishi, & Ahlawat 2013).

In implicit segmentation, recognition techniques are used. In this method, an image is analyzed to find the matching parts with the predetermined symbols (Arica & Vural, 2001). But this approach may lead to Sayre's paradox (Vinciarelli, 2002).

In the present work we used explicit (pure) segmentation. In this strategy, character's shape properties are used to cut the characters out (dissection). These properties usually depend on the nature of the symbols of the language used in the document to be segmented. Common dissection methods use white spaces,

connected component analysis, vertical projection analysis and landmarks (Arica & Vural, 2001).

### *2.2.1 Text Segmentation*

Document image segmentation can be examined at three hierarchical levels which are text line, word and character segmentation (Mehul, Ankita, Namrata, Rahul, & Sheth, 2014). All or some of these levels can be employed according to the requirements of the application at hand.

#### *2.2.1.1 Line Segmentation*

Line segmentation level is at the top of the hierarchy. For segmenting lines, image rows at pixel level are taken into account. Grouping pixels in a horizontal region gives text lines. Overlapping lines are predominant for the decrease in performance of the line segmentation. Slant and slope may also affect the result negatively if they couldn't be removed in the preprocessing stage.

#### *2.2.1.2 Word segmentation and Character Segmentation*

Other hierarchical levels are word and character segmentations. These have similar operations since both operations are carried out on vertical direction. Word segmentation can be relatively easy due to the gaps (whitespaces) between words.

Character segmentation methods have to deal with the touching characters in manuscripts or separation of the ligatures in cursive type handwriting.

As in the presented work, same algorithm can be used for both word and character segmentation to obtain characters instead of words.

### *2.2.2 Segmentation Algorithms*

A text document image can be segmented into lines, words, or characters using different algorithms. Some of the algorithms depend on the presence of the guidelines (Miah, Yousuf, Mia, & Miya, 2015; Palehai & Fanany, 2017) and other algorithms work without guidelines (Hossain, Naznin, Joarder, Islam, & Uddin,

2018; Sanchez, Suarez, Mello, Oliveira, & Alves, 2008). Segmentation can be relatively easy when the document is guidelined. Guidelines mostly eliminate skew (slope) of the text line and they constitute good candidates of boundaries where text line segmentation occurs. Character size and position are also determined according to the guidelines, which makes segmentation easier and more accurate.

*Pixel counting algorithm* is a simple technique which works on binarized images. In an ideal situation when two text lines are horizontally aligned with a gap between them, the gap can be determined by counting the number of white pixels (background) in a horizontal way, scanning every line of the image from top to bottom. But in real cases, text lines can have an inclination, characters of the successive text lines can overlap which causes this algorithm to fail.

*Histogram algorithm* segments the text line regions of a handwritten document (Mehul, Ankita, Namrata, Rahul, & Sheth, 2014). In general, every pixel row of the document image is scanned to create the profile (i.e. histogram) of horizontal projection. In an ideal case, there should be white spaces between text lines that create zero valued regions in the histogram. These regions are used to separate the text lines (Thungamani & Kumar, 2012). Instead of finding absolute whitespaces, some threshold value can be used to separate text lines when there are overlapping pixel projections along the direction of projection.

In this work we also used histogram approach on the colored image to detect red guideline locations. Location of red horizontal guidelines corresponds to the text line boundaries whereas location of red vertical guidelines corresponds to the character boundaries. A problem arises for the characters overflowing these boundaries. To overcome this problem, connected components analysis is used. Character is decided to belong to a text line if character's corresponding connected component label's centroid falls within that text line boundary.

We used histogram projection algorithm also to determine the location of vertical guidelines where they exist i.e. in the first ROI. In this process, vertical histogram projection is used instead.

Histogram method can be extended to segment words and characters (Mehul, Ankita, Namrata, Rahul, & Sheth, 2014; Thungamani & Kumar, 2012). This is accomplished in the present work, by taking into account the vertical projection profiles of each segmented text line which are calculated in the previous step. In this part it is assumed that there are gaps between discrete handwritten characters and no ligatures. This assumption can be loosened by accepting pixel count to be under a threshold, compared to the character's itself to segment overlapping or cursive handwriting. Applying this threshold value on the vertical histogram projection results vertical segmentation cuts giving rectangular regions in which a single character is possibly obtained. These rectangular regions are then used to extract single characters by the following phase.

### 2.3 Handwriting Recognition

Handwritten character recognition systems generally use pattern recognition techniques. Pattern recognition is, in general, assigning a label of a predefined class to an unknown symbol (Arica & Vural, 2001). For handwriting recognition, predefined classes could be words of a language or labels of a lexicon for a certain application. For plain texts, lexicon consists of simply alphabet letters and punctuations. For mathematical expressions, it will have additional mathematical symbols like plus sign or integral symbol. For source code, it will have programming language specific character set which includes braces, brackets, operators like percentage, underscore, hash sign etc.

HCR applications belong to two main fields: online and offline. Online applications use data which is recorded from the writing done on digital equipment such as tablet PC or smart phone. Offline applications use data usually obtained by a scanned document of handwritten paper. Typically, online data contain more information about the handwriting and therefore applications using online data can reach higher performance rates. Nevertheless, because of the prevalence of pen and paper, offline applications extend into wider range of real-world systems such as postal system automation (Vinciarelli, 2002; Wanchoo, Yadav, & Anuse, 2016) or

automatic bank check reading (Knerr, Augustin, Baret, & Price, 1998; Miah, Yousuf, Mia, & Miya, 2015; Palacios, Gupta, & Wang, 2004).

Some recognition systems use top down approach for recognizing full word so the system doesn't require the segmentation process of characters one by one. Other systems use analytic strategy using bottom up approach, starting from character detection, creating words from characters, and forming text lines or sentences. This strategy requires a robust character segmentation and detection algorithm which increases the computational complexity; on the other hand, can be applied on unlimited vocabulary. In this way, the recognition of isolated handwritten characters and other symbols can be accomplished with high accuracy (Arica & Vural, 2001). Mostly four general pattern recognition approaches are used for handwritten character recognition (Jain, Duin, & Mao, 2000):

- Template Matching,
- Statistical Techniques,
- Structural Techniques,
- Neural Networks.

These approaches generally have the following steps:

- Feature extraction
- Training
- Classification

Recognition systems based on HMM was accepted as the state of art and used frequently at the end of 90's. Later, by research and developments on the Neural Network (NN) systems, hybrid neuro-Markovian systems have been emerged. Developing NN systems are increasingly used in HCR applications. Recently, CNNs have outperformed traditional networks. Cireşan, Meier, Gambardella, & Schmidhuber (2011) applied CNNs on NIST SD 19 dataset which contains 62 classes of digits and letters, reported 11.88% error rate. There are many works using CNNs on the HCR of plain text application and few for ME.

## 2.4 Convolutional Neural Networks

Deep learning algorithms are proven to have good performance on the learning of the features of data which are better suit to task at hand. They are organized in a hierarchy with multiple levels each of which is associated with a different feature of data. Convolutional neural networks are one of the architectures implementing deep learning technique. They are a regularized variant of multilayer perceptrons (MLPs) and developed using insights from the work of Hubel and Wiesel on cat's visual cortex (Hubel & Wiesel, 1962). Their work reveals that cat's visual cortex has complicated structure of locally sensitive cells. Each group of these cells are sensitive to different subregions of the image appeared on the input space (retina). These subregions are called as receptive field of that group of cells. Local receptive fields are best suited to utilize the local correlations in the images since the images have strong 2D local structure and neighboring pixels are highly correlated. Utilization of spatial correlation yields better feature extraction in hierarchical levels, for example edges and ovals in lower levels and eyes and ears in higher levels. Fukushima proposed the first CNN model, called Neocognitron for the task of recognition handwritten digits (Fukushima, 1988). Reminiscent of cells in the Hubel and Wiesel work, for extracting local features of input data, Neocognitron employed locally receptive field idea such that each neuron connected only subregion of the preceding layer which consist of some neighboring neurons in that layer. LeCun et al. proposed another CNN model, LeNet-5, using gradient based back propagation learning algorithm (LeCun, Bottou, Bengio, & Haffner, 1998). CNN architectures generally adapt three concepts. These are local receptive fields, weight sharing and spatial subsampling which gives shift, rotation and distortion invariance at some extent (Rashid, 2014). Without doubt this quality of CNNs is very beneficial and best suited for the tasks involving recognition from 2-D images.

As a general design principle, almost every CNN model has the following sequential structure: convolutional layers, subsampling layers, one or more hidden layer of fully connected traditional neural networks and final output layer. Layers are the planes of organized units of neurons through which the input data is transformed, called also as *credit assignment path* (CAP), whereas the number of layers called as

CAP depth. Deep learning architectures generally uses substantial amount of CAP depth, hence the name *"deep"*. The number of convolutional and subsequent subsampling layers is chosen to best fit to the actual task of the network. In these layers each unit of cells are connected only to a small subset of neighboring units in the preceding layer which constitutes local receptive field of the unit concerned. Local receptive fields give units the ability to extract features hierarchically, i.e., low level features in former levels and high level features in latter levels. All cells of units in the layer plane share the same set of weights and are constrained to perform the same operation along all the portions of the input image. Outputs of these units constitute the feature map of this plane. Multiple features are extracted for each portion of the input image because a convolutional layer utilizes more than one feature map, and these feature maps have different weights. Resulting feature maps are fed into the hidden layer which is generally fully connected layer of cells. These cells are connected to the output layer. Output layer performs final classification based on its inputs. Number of outputs of this layer is determined by the number of specific target classes of the actual task. Figure 2.1 shows a diagram of CNN used in this work for handwritten character recognition task.

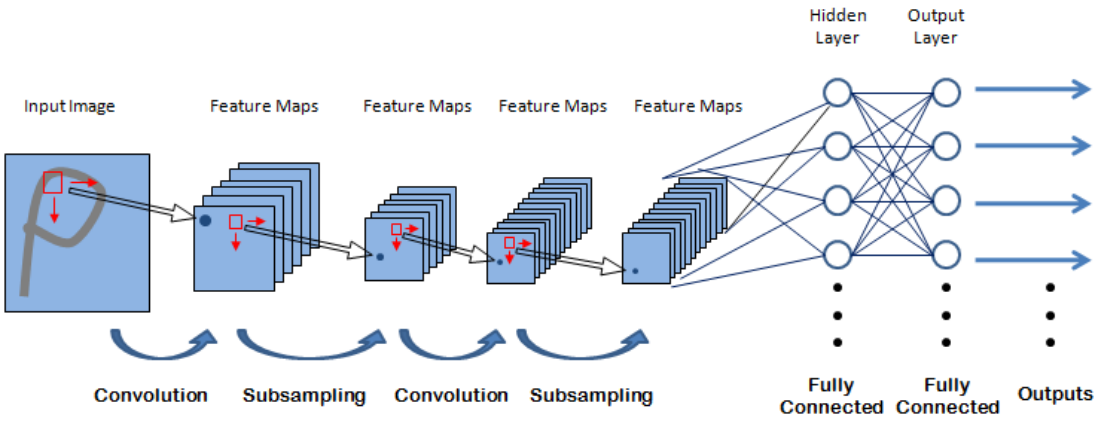



Figure 2.1 Representation of CNN architecture

Hierarchical learning capabilities of CNNs gives them advantage over other techniques for learning features automatically so that classification based on raw image data can be done.

Convolutional neural networks have two main parts in general. Convolutional layer which is responsible for feature extraction and fully connected layer which is responsible for the classification. There are some other layers that support CNN's operations. General aspects of layer types which could generally be found in a CNN's sequential structure explained in the following subtitles.

### *2.4.1 Convolutional Layer*

Convolutional layers are responsible for extracting features that corresponds to patterns to be learned and detected. In image processing, convolution is a mathematical operation and a sort of filtering which involves input matrix (image) and a filter matrix. This filter is also called as kernel in image processing terminology, and denoted as $h$, and image denoted as $I$ in the following formula. The indexes of rows and columns of the resulting matrix are marked with $m$ and $n$ respectively.

$$G[m,n] = (I * h)[m,n] = \sum_j \sum_k h[j,k] I[m-j, n-k] \tag{2.1}$$

where $j$ and $k$ take all the values so that the subscripts of $h$ and $I$ are valid.

In NN point of view, filter matrix or kernel elements are the weights of neurons which are subject to update by the learning algorithm that the network adopt. A convolutional layer has several of these filters. This number is proportional with the amount of features or patterns that the network is capable of recognizing.

The convolution process defined by the given formula can be viewed as shifting the filter through the input image matrix and calculating inner product at every shift. Shift size in terms of skipped pixel count is called as *stride*. In this process, the pixels near the edges have much smaller impact on the result than the ones near the center. To eliminate this effect, *padding* may be used. Padding is simply adding extra pixels to the edges of the input image. Size of padding and the values of added pixels are parameters to be decided by designer. Usually padding is done by adding zero valued

pixels. Consequently, convolutional layer's parameters are: the kernel size and count, padding size and padding values and stride. The relation between the dimensions of the output matrix, padding and stride is given in the following formula.

$$n_{out} = \left[\frac{n_{in} + 2p - q}{s} - 1\right] \tag{2.2}$$

where $p$ is padding, $q$ is the filter dimension (usually odd) and $s$ is stride. Resulting output matrices are the feature maps that are calculated at this layer. In this way, feature maps at the same layer share their weights but have different receptive fields. Subsequent layers of feature maps represent the hierarchical levels from bottom to up.

### *2.4.2 Activation Layer*

Feature maps which are produced by the convolutional layers are much smaller in size than the input matrix. Since the CNNs resemble NNs, they as well, adopt activation functions which are arranged in the layer called as *activation layer*. Though this layer is thought to be a successive part of the preceding convolutional layer, it is not shown explicitly in topological schemes since it is almost always used after a convolutional layer.

Activation functions determine the output of the neuron, based on the input value. Various activation functions exist for CNNs such as *linear function*, *sigmoid*, *tanh*, *binary step function* and *ReLu* (Rectifying Linear Unit). Non-linear activation functions are the most popular ones since they have localized outputs thus reducing the complexity of the input and make it possible for the network to generalize or adapt with data. They better suit with the backpropagation learning algorithms which allow the network to train itself making it capable to learn and perform more complex tasks.

*ReLu* is the most common activation function recently, which is defined as:

$$ReLu(x) = max(x, 0) \quad (2.3)$$

It is simply performing a threshold operation on the input and can be implemented and calculated more easily than *tanh* or *sigmoid* functions. Thresholding operation causes only few neurons to be activated at a time. These attributes of the *ReLu* function make it possible to reduce overall computational cost effectively.

### *2.4.3 Subsampling Layer*

The next layer is subsampling layer in CNN architecture. It is also called as downsampling layer. Subsampling layer operates on the output of the activation layer. The intuition for the subsampling operation is that the exact location of a feature is not important if it has been found. The result of subsampling operation is reduction in the size of feature maps while keeping the important information. That means there would be less parameter to be calculated, thus, subsampling too, helps to reduce the computational cost. Subsampling helps also to prevent overfitting.

Usually types of spatial pooling used in the subsampling layers. These might be one of the followings:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling is taking the largest element of the non-overlapping rectangular regions of the output of the activation layer. Average pooling and sum pooling take the arithmetic average or the sum of elements of the same region, respectively. Consequently, for the given pooling region of size ($P_x$, $P_y$), the input image is down sampled by a factor of $P_x$ and $P_y$ along each direction (Cireşan, Meier, Masci, Gambardella, & Schmidhuber, 2011a).

Max pooling is the most common used type of subsampling that supplies position invariance over larger local regions. Scherer, Müller, & Behnke (2010) states that

max pooling can converge faster, yield better generalization and select superior invariant features. General feature pooling's and max pooling's theoretical analysis can be found in the work of Boureau, Ponce, & LeCun (2010).

### *2.4.4 Dropout Layer*

Dropout layer functions only in the training phase of the network. Main purpose of using it is to prevent overfitting. This may occur especially when the training dataset volume is relatively low. When we train a network with a low quantity of samples, network might not find enough samples to generalize the data and learn the specific features of the given training data which are thought to be "noise" since these features don't represent the true nature of the data source. This effect can be thought to be "memorizing" the training data. This is sometimes called as "generalization error". Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov (2014) show that dropout is an efficient regularization technique for reducing this sort of error in many cases.

As its name implies, this technique simply drops out neurons at given probability in the training phase. Dropped out neurons' outputs are counted as zero on the forward pass and their weights are not updated on the backward pass. By dropping out, randomly different "thinned" networks are trained which are sharing the weights. As a result, the network becomes less sensitive to the specific weights of neurons. This makes the network capable of better generalization and less likely to overfit the training data. However, it is advised to use it only for fully connected layers and not for convolutional layers.

### *2.4.5 Batch Normalization Layer*

*Stochastic gradient descent algorithm* is one of the most used algorithms for training the deep neural networks. In this algorithm, training dataset volume is divided into mini-batches over which the loss calculated instead of using all the training data at once or using them one by one. This technique increases the training speed effectively. But this may give rise a problem which is called as *"internal covariate shift"* (Ioffe & Szegedy, 2015). The reason of this problem is having mini-

batches with different distributions. These distributions could have different means and deviations which may cause imbalanced gradients therefore cause *"exploding gradient"* effect. This brings about the instability and decrease in the learning speed since each layer tries to learn new distribution at every training step.

Batch normalization layer aims to solve this problem and thus increase the learning speed and stability. Generally speaking, normalization puts all the data points on the same scale. In a similar way as in the preprocessing of inputs for normalization before the input layer, batch normalization layer normalizes the previous layer's outputs for the next layer, by subtracting the batch mean and dividing by the batch standard deviation. This process forces the data points to have zero mean and standard deviation of one, according to the following calculations (Ioffe & Szegedy, 2015):

Activations for a mini-batch:

$$B = \{x_{1..m}\} \tag{2.4}$$

Mini-batch mean:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.5}$$

Mini-batch variance:

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x - \mu)^2 \tag{2.6}$$

Normalized activation:

$$\widehat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}} \tag{2.7}$$

where $\varepsilon$ is an arbitrarily small constant for numerical stability.

Batch normalization output:

$$y_i = \gamma \widehat{x_i} + \beta \tag{2.8}$$

where, $\gamma$ and $\beta$ are hyperparameters to be learnt in the training phase. Each unit has their own set of those parameters.

Batch normalization process helps to avoid weights and biases to become imbalanced with extremely high and low values which will over-influence the training process. It also prevents nonlinearities e.g. sigmoid to become saturated thus helps to solve *"vanishing gradient"* problem.

Increased stability makes the network less sensitive to how weights are initialized and makes possible to increase the learning rate. Moreover, since the scaling results in adding some noise within that mini-batch, it gives similar effect to dropout therefore provide some regularization and reducing the generalization error. It is not recommended to use batch normalization and dropout layers together since the normalized activations could be more noisy after dropout. Batch normalization layer may take place before or after the activation layer. This process is at batch basis hence the name *"batch"* normalization.

### *2.4.6 Fully Connected Layer*

A CNN model usually has several amounts of convolutional-activation-subsampling layers and one or more fully connected layers subsequently. Typically, they have much more parameters than the convolutional layers which is the one reason why they are generally used at the end of the architecture since the feature

maps sizes get smaller there. Another reason is that convolutional layers give localized results using *local receptive fields* principle by decreased number of connections. Presumably, fully connected layers can give the same results after much more training epochs.

In the input of first fully connected layer, resulting feature maps from the previous layer are flattened, i.e., reconstructed as a vector of one dimension and fed into the fully connected layer which is essentially a traditional neural network layer. Neurons in the fully connected layers are connected all the outputs (feature maps or also called as activations) of the previous layer, as its name implies. The output of this layer is calculated as the matrix multiplication of the input by a weight matrix and adding a bias vector. Learning and recognizing of the larger patterns happens at this layer since it combines all the information learned by the previous layers. Combining all the information gives network the ability to classify the whole input image correctly for classification problems.

Fully connected layer can be thought as a larger convolutional layer whose number of kernels (filters) is equal to the number of neurons in it with each kernel size is as big as its input's size. Therefore, it can be said that each neuron in the fully connected layer performs 1-dim convolutions. Adding more than one fully connected layer corresponds to adding more 1-dim convolutional layers which may increase overall learning capability of the network.

For the networks designed for classification problems, the output size of the last fully connected layer is equal to the number of classes of the dataset. Its outputs are probabilities for the possible labels that the image to be classified in (e.g. car, train, bus). The label corresponding to the output with the highest probability is the classification decision.

### *2.4.7 Softmax Layer*

CNN's layer structure generally ends with a consecutive softmax layer and a classification output layer, for the mutually exclusive multiple class problems. Softmax function acts as an activation function which is applied at this layer to the

preceding fully connected layer's output. Softmax layer's size is also equal to the total number of classes of the dataset as the final fully connected layer's size. Softmax layer assumes each input belongs to one and only one class. Each softmax layer's output will be in the interval (0, 1), and they will add up to 1, so that they can be interpreted as probabilities. Softmax function is defined as:

$$Softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \; for\; i = 1,..,K \; and \; \mathbf{z} = (z_1,..,z_K) \in \mathbb{R}^K \qquad (2.9)$$

where $K$ is the number of classes of the dataset, $\boldsymbol{z}$ is the output of the previous layer. This function enlarges differences in the input data and is also known as normalized exponential function. It is fully differentiable therefore is appropriate for training algorithms uses gradients such as gradient descent. The main objective of this layer is to turn the network's output to probabilities of each input element belonging to a class therefore output represents categorical distribution over class labels.

### *2.4.8 Classification Output Layer*

Remaining other calculations are carried out in a layer which is called as *classification output layer* that follows the softmax layer. Generally two functions attached to this layer. First, it assigns outputs a label according to the output of the softmax layer. Second function runs only in training phase to calculate *loss function* or a.k.a. *cost function*. Sometimes this layer is called as *loss layer* because of this functionality. Loss function is used in the training of the network by the backpropagation algorithm which tries to reduce the output of this function, or namely the "*loss"*…. There is several loss functions used in accordance with the specific application at hand, such as *Mean Absolute Error (MAE or L1 loss), Mean Squared Error (MSE or L2 loss), 0/1 loss, Huber loss, Hinge loss, cross entropy*, etc. Softmax layer is usually used with the cross entropy loss function for multi-class classification problems with mutually exclusive classes. Cross entropy loss is a

measure of total distance between the network label predictions and the original labels over all the samples.

Cross entropy for the correct output $y$ and network's prediction $\hat{y}$ for a sample is defined as:

$$H(y,\hat{y}) = -\sum_{m=1}^{M} y_m \log \hat{y}_m \tag{2.10}$$

where $M$ is the number of classes of the dataset. Here the network's prediction $\hat{y}$ is the value from the softmax function in this case.

Then the average total loss is calculated as:

$$loss = \frac{1}{N}\sum_{n=1}^{N} H(y_n, \hat{y}_n) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{M} y_{m,n} \log \hat{y}_{m,n} \tag{2.11}$$

where $N$ is the number of samples, $y_{m,n}$ is the correct output for the $n$th sample, $\hat{y}_{m,n}$ is the network's prediction for the $n$th sample.

For the problems involve prediction and classification, cross entropy loss function has some advantages over the other types of losses especially when gradient descent based learning algorithms are used. MSE loss function for example, penalizes the outliers strongly. Badly misclassified samples over-emphasized and shift the training too much.

## 2.5 Related Works

Related HCR applications are presented in the following subchapters. Works on the Bank check reading application are given in Section 2.5.1, postal applications are surveyed in Section 2.5.2, works on handwritten mathematical expression

recognition inspected in Section 2.5.3, a work about using CNNs for HCR is given in Section 2.5.4 and a work about online handwritten source code recognition system is given in Section 2.5.5.

### *2.5.1 Bank Check Reading*

Bank checks are usually written by handwriting. Bank check reading is one of the applications of the offline handwritten character recognition. Since automatic reading of check amounts is important and very advantageous for the banking industry, many works about HCR was made about this application. They are shown to have performances close to human's and some of them are deployed commercially. As in the many application of HCR, the segmentation plays an important role for bank check reading because of the usage of different handwriting styles in the fully unconstrained manner. Cursive letters, overlapping or touching characters make segmentation challenging. Another difficulty for check reading is that, they are written by different writers. HCR of bank check reading has an advantage since the amount is written in two different format, with words (legal amount) and digits (courtesy amount). This allows checking the results produced for one with other and can be used to improve the performance.

Table 2.1 Bank check reading performances

| Authors | Subject | Recognition System | Performance (%) |
|---|---|---|---|
| Knerr, Augustin, Baret, & Price, 1998 | Words | HMM | 89.2 |
| Miah, Yousuf, Mia, & Miya, 2015 | Digits | MLP | 93.4 |
| Palacios, Gupta, & Wang, 2004 | Digits | MLP | 93.1 |

### *2.5.2 Postal Applications*

Postal applications are one of the main applications where many works can be found about HCR. This field exhibits a challenging task since the handwriting on the envelopes is completely unconstrained; words can be cursive, handprinted, machineprinted or mixed and written by different writers. Moreover it could have multi-lingual and multi-script behavior. Recognition of zip codes plays an important role. When it is recognized, reading for further information may become redundant. El-Yacoubi, Gilloux, Sabourin, & Suen (1999) got 96.3% performance rate with an HMM based system. Akhand, Ahmed, & Rahman (2016) proposed CNN based system for digit recognition on the characters taken from handwritings of mail pincodes. They reported **98.98%** performance rate.

### *2.5.3 Handwritten Mathematical Expression (ME) Recognition*

The most similar problem domain with ours (handwritten source code character recognition) is the handwritten mathematical expression (ME) recognition. Mathematical symbols have similar handwritten characters as well as the same ones with source code characters yet still have specific differences. Handwritten ME recognition has been studied noticeably less, however, most of works in this area done on the online data, which is collected from digital medium such as tablet PCs or smart phones, where individual pen strokes are recorded along with timing information. This gives additional information about how writing is done and can be exploited so as to improve the accuracy and performance of recognition since offline recognition methods can also be applied on online data by simply discarding additional information such as temporal and structural information. One of the great source for the online handwritten ME is the CROHME competition, organized since 2011. Most work on this area used the database from this source. In our work, we also used CROHME dataset discarding online information to enhance our dataset.

Lu & Mohan (2015) used parameterized different size CNN models for online handwritten ME recognition and get **89.7%** test accuracy at best.

Nguyen, Le, & Nakagawa (2015) reached **91.28%** performance rate with 101 classes CROHME dataset, using deep learning techniques and combining both online and offline methods to improve classification performance.

With their works, Alvaro, Sánchez, & Benedí (2014) fill the gap of offline handwritten math symbols recognition, since many works have done using online or hybrid approach on the subject. They also test a novel set of features based on polar histograms and the vertical repositioning method for feature extraction as well as assess the performance of several well-known offline features for this task. A Recurrent Neural Network (RNN) has been used as the classifier because it outperforms Hidden Markov Model (HMM) in this task. As the result of their experiments they reported that offline features outperform online features and their combination significantly improves the recognition rate. Their best result for TOP1 is **87.1%** with combination of online and well-known offline features.

Hossain, Naznin, Joarder, Islam, & Uddin (2018) applied CNN based classifier to offline handwritten ME which are chosen to be only quadratic equations in this work, in the form of $ay^2 + by + c = 0$. Furthermore, their work finds the solution of the recognized quadratic expression. With more than 1000 test images, performance rate they reached was **91.08%** for character recognition rate.

### *2.5.4 Convolutional Neural Network Committees for Handwritten Character Classification*

In their work Cireşan, Meier, Gambardella, & Schmidhuber (2011) used a committee of seven deep CNNs for HCR problem on MNIST and NIST SD 19 databases. They trained CNNs on graphics cards (GPUs) and achieved significantly shorter training times with this technique. They got **11.88%** error rate on the 62 classes NIST SD 19 database.

### *2.5.5 Recognizing Handwritten Source Code*

Zhi & Metoyer (2017) pointed out the deficiency of handwritten source code recognition systems especially for online. They claimed that existing handwriting

recognition applications that running on the digital devices take generally natural language rules into account and produce erroneous results for source code recognition. Their work is to improve the recognition performance of a handwriting recognition application by augmenting it with the programming language grammar rules. With this improvement digital devices, such as smart phones and tablet PCs, can recognize online handwritten source code more accurately e.g. when using stylus input device. They explored the results of a state-of-the-art online handwriting recognition engine, *MyScript*, for handwritten source code. Their method is an additional postprocessing step for the engine output to improve them for Python programming language source code. They had 8.6% word error rate and 3.6% character error rate (**96.4%** performance). They used the following metric for word error rate (WER) and character error rate (CER), which is known as Levenshtein distance:

$$\frac{D + I + S}{L} \times 100\% \tag{2.12}$$

To calculate the CER, following values are used in this formula: The total characters count in the original document: $L$, deleted characters count: $D$, inserted characters count: $I$, substituted characters count: $S$. These values are determined by comparing the resulting recognized text against the original one, manually.

# CHAPTER THREE
# RECOGNITION OF HANDWRITTEN CHARACTERS OF EXAM PAPERS

As mentioned previously our work is done on the exam papers for C programming language course. A sample of it can be seen in Figure 3.1. We can separate ROI's according to which type of guideline it has. First ROI is the name field and has both vertical and horizontal guidelines. Second ROI is the answering field and has only horizontal guidelines. Guidelines are used mainly for segmentation. It gives also useful information about page orientation.

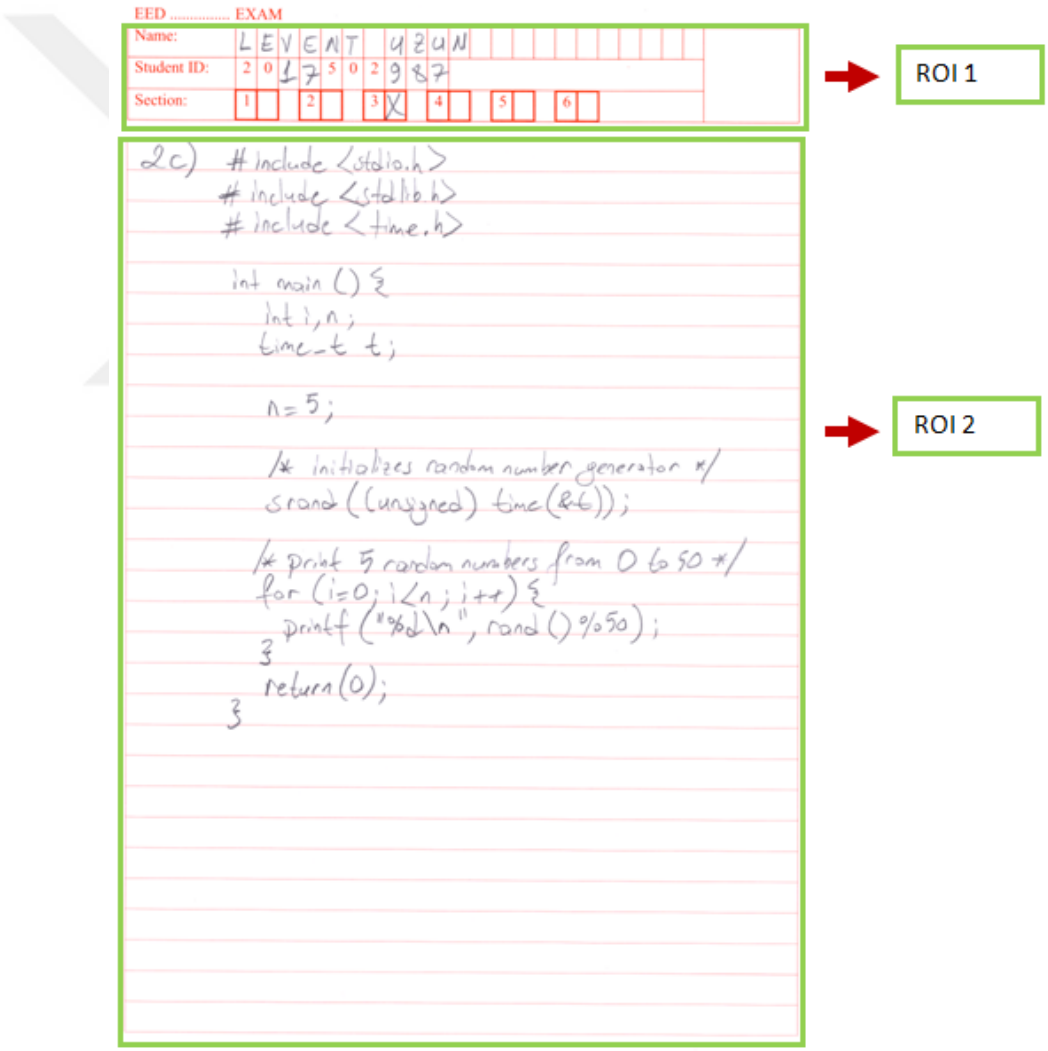



Figure 3.1 A sample document used in HCR processes

Our work has two phases. (a) Dataset construction, (b) training and testing CNNs by enriched dataset consists of ours merged with CROHME dataset.

The first phase can be divided into following steps:

- (a1) image pre-processing including guideline removal, orientation correction, median filtering and connected component based filtering
- (a2) image segmentation
- (a3) normalization
- (a4) manual labeling
- (a5) dataset enrichment by merging with CROHME dataset.

The second phase can be divided into below steps:

- (b1) choosing/designing and implementation of different CNN topologies.
- (b2) training CNNs for character level classification
- (b3) testing/ comparison

## 3.1 Dataset Construction

### *3.1.1 Image preprocessing*

#### *3.1.1.1 Handwriting image extraction and guideline removal*

We first detect guidelines by using their color difference to separate the handwritten information on the document. Guidelines have more red values in weight.

Digital images are represented as color pixels. One of its format is RGB images where each color pixel is actually three dimensional vector such as $[r\ g\ b]^T$. Each dimension corresponds to color weight of red, green or blue respectively. Weights are between 0 and 255. Therefore, this representation can handle $256^3$ or $2^{24}$ colors. This requires 24 bits wide memory space for each pixel. This type of images is called as 24 bit true color RGB images. For such an $m$ x $n$ sized RGB image $I$, there is $m$ number of row and $n$ number of column color pixels. This image $I$ can be represented

as *m* x *n* x 3 sized matrix. This means there is three *m* x *n* matrices each of which corresponds to one of red, green and blue values (weights) respectively.

For detecting pixels of red-weighted guidelines, we first found pixels of which red weights are bigger than a threshold. That is all pixels in the first *m x n* matrix which has the value bigger than the threshold.

$$P = \{[i\ j] \mid I_{i,j,1} > t; i = 1,..,m; j = 1,..,n\} \tag{3.1}$$

where *[i j]* represents vector of a pixel location *i, j* on image *I*, *t* is threshold for red weights. We empirically found 200 for red threshold value. Second, from these pixels we choose ones which has red weight bigger than green and blue by a difference value that will be determined.

$$Q = \{[k\ l] \mid I_{k,l,1} - I_{k,l,2} > d \wedge I_{k,l,1} - I_{k,l,2} > d; \forall [k\ l] \in P\} \tag{3.2}$$

where *[k l]* is a vector of pixel location *k, l* on image *I*; *d* is difference value we seek, between red and other values. We take the value 1 for the d, empirically. *Q* is the binary matrix having elements "1" or "0". The elements with the value "1" show that the corresponding pixel belongs to guidelines. Therefore, *Q* can be thought as a binary image showing only guidelines. Figure 3.2 shows this image for an exam paper. In this figure, "1"s in the *Q* matrix is represented as white pixels.

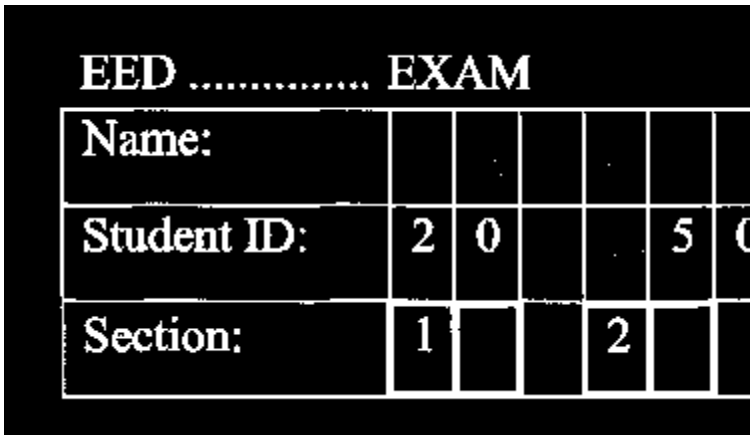
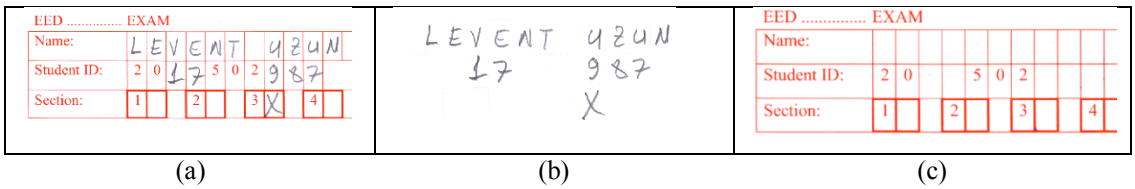



Figure 3.2 Binary image representing guideline locations

After detecting guideline pixels, we obtain handwriting image by equating these pixels to 255 at all 3 dimensions as in the Equation 3.3 below. This corresponds to the white color which is the background color:

$$I_{k,l,1} = 255,$$
$$I_{k,l,2} = 255, \qquad (3.3)$$
$$I_{k,l,2} = 255, \forall\, [\mathrm{k\ l}] \in \mathrm{Q}$$

Results of guideline removal steps for upper left part of an exam paper are shown in the Figure 3.3:




(a) (b) (c)

Figure 3.3 Guideline removal gives image of handwriting (b) and guidelines (c). Original image is depicted in (a)

*3.1.1.2 Orientation Correction*

By using guidelines image that we detected, page orientation, caused by bulk handling of exam papers of scanner device, was calculated. We first took an image of one horizontal guideline and found its horizontal and vertical pixel counts *h, v* respectively. Then orientation *o* is calculated as:

$$o = tan^{-1}(v/h) \tag{3.4}$$

Then images are rotated by *o* degree in the opposite direction of rotation.

*3.1.1.3 Grayscale Conversion*

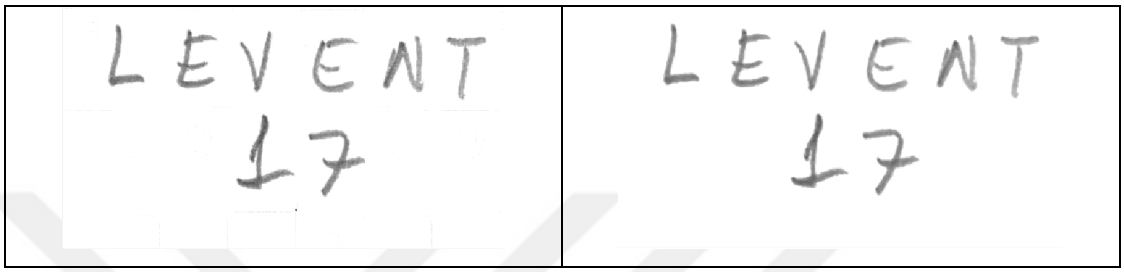
Next step is converting true color RGB images to the grayscale intensity images. RGB images have three values (weights) for a pixel whereas intensity images have only one value representing intensity within some range. Moreover, by this conversion, color information is lost. But for the current application, as for many applications, color information is not necessary, since the handwriting is done by only one color, besides it is already grayscale in our application. Representing a pixel as a single value needed in the subsequent steps and especially in CNN input. Grayscale conversion eliminates the hue and saturation information while retaining the luminance. It is calculated by forming a weighted sum of red, green and blue components, *R*, *G*, *B*, respectively, as:

$$Grayscale\ value = 0.2989 * R + 0.5870 * G + 0.1140 * B \tag{3.5}$$

*3.1.1.4 Filtering*

We applied median filtering on the resulting two dimensional images in the previous step. In median filtering, a pixel value is calculated as the median value of the pixels which are in the 3-by-3 neighborhood of the pixel under evaluation. This

filter size has been chosen empirically so as to preserve dots belong to handwritten characters. If there is two median values, average of them is used. Zero padding applied around the image edges before median value calculation. This introduces some distortion on the edges. Figure 3.4 shows the effect of median filtering. Differences are apparent on the computer screen and might not be observed exactly in the printed form.



(a) (b)

Figure 3.4 Median filtering effect: (a) before, (b) after

Median filtering is a nonlinear operation and used generally against the “salt and pepper” noise and smoothes the image. This type of noise might be introduced by natural reasons when handwriting, by scanner device quality or by previous steps such as guideline removal. If preserving edges is important while reducing the noise, median filtering has an advantage over filtering by convolution.

### *3.1.2 Segmentation*

For segmentation, we benefit from guidelines. Guidelines aid character level segmentation at the ROI 1 where horizontal and vertical guidelines exist. At the ROI 2, guidelines are used for textline level segmentation since there exist only horizontal guidelines in this ROI. Then we used histogram projection algorithm for character level segmentation.

To make use of guidelines, we took the guidelines image extracted previously at guidelines removal step. First we computed histogram projection of guidelines on both dimensions, vertical and horizontal. The histogram projection was calculated simply by summing all the values along the relevant dimension.

Vertical Histogram:

$$\boldsymbol{V}_i = \sum_j \boldsymbol{Q}_{i,j}\ i = 1,..,m\ ; j = 1,..,n \tag{3.6}$$

Horizontal Histogram:

$$\boldsymbol{H}_j = \sum_i \boldsymbol{Q}_{i,j}\ i = 1,..,m\ ; j = 1,..,n \tag{3.7}$$

where $\boldsymbol{V}$ is a raw vector showing vertical histogram projection, $\boldsymbol{H}$ is a column vector showing horizontal histogram projection and $\boldsymbol{Q}$ is the $m$ x $n$ size binary image of guidelines. Exemplary horizontal and vertical projections are shown in Figure 3.5 for the upper left part of guidelines image.

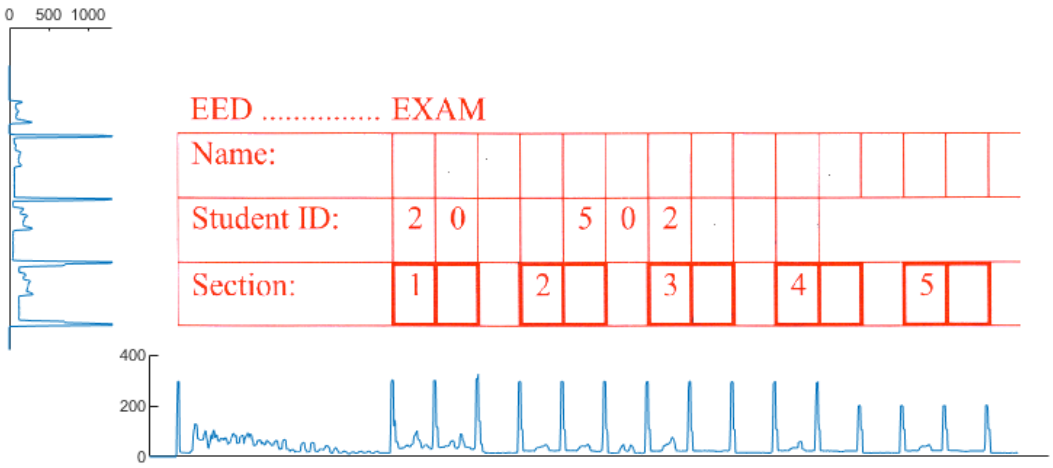



Figure 3.5 Vertical and horizontal histogram projections for upper left part of the guidelines

Then we calculated regions surrounded by guidelines by applying threshold on the histogram projections. Thresholds are determined empirically.

$$G_i = V_i > T_v \; ; \; i = 1, .., m \tag{3.8}$$

$$G_j = H_j > T_h \; ; \; j = 1, .., n \tag{3.9}$$

where $G_i$ is a binary raw vector showing vertical guideline locations, $G_j$ is a binary column vector showing horizontal guideline locations, $T_v$ and $T_h$ are threshold values for vertical and horizontal projections, respectively. Detected regions for a sample document are given in Figure 3.6.

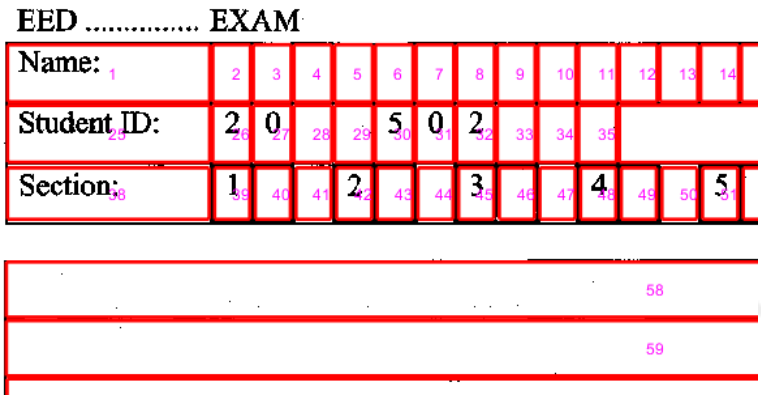



Figure 3.6 Segmented regions of image by guidelines

*3.1.2.1 Connected Components Labeling*

Connected components labeling, groups and labels the pixels of an image based on given pixel connectivity. Common pixel connectivities are known as 4-connectivity and 8-connectivity. In 4-connectivity, pixels in the neighborhood of the pixel under evaluation are assumed to be connected if they are touching by only edges. In 8-connectivity, touching by corners is also taken into account as assuming connectedness. Connectedness relies on the equality of the intensities of the pixels. Therefore, generally, intensity images converted to black and white images prior to labeling. This process works by taking all pixels sequentially from top to bottom and

from left to right, calculating all connections and assigning a label to all different groups found.

After converting the handwriting image of exam papers to black and white, we found labeling with 8-connectivity. Detected labels for a sample document are illustrated in the Figure 3.7. in which different labels are represented by different colors. Resulting labels' properties such as centers and bounding boxes are used to help segmentation of characters excessive regions surrounded by guidelines, as well as for filtering out small artifacts.

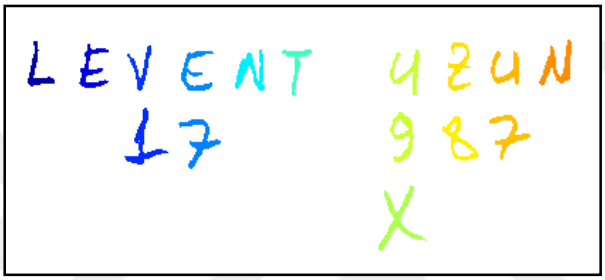


Figure 3.7 Connected component labeling (different colors show different labels)

Then we look at connected components labels at every region surrounded by the guidelines (red boxes in the Fig.8). For every region we found all the labels whose centers fall in that region. A *Label* is assumed to be in the *Region* if the followings hold:

$$Left_R \leq x_C \leq Right_R \tag{3.10}$$

$$Up_R \leq y_C \leq Down_R \tag{3.11}$$

where, $Left_R$ is the horizontal axis component of the *Region*'s left edge, $Right_R$ is the horizontal axis component of the *Region*'s right edge, $x_C$ is the horizontal axis component of the *Label*'s center, $y_C$ is the vertical axis component of the *Label*'s center, $Up_R$ is the vertical axis component of the *Region*'s upper edge and $Down_R$ is the horizontal axis component of the *Region*'s lower edge. This evaluation

is done for all the regions determined and all the labels found. Then we had a list about which label belongs to which region.

Figure 3.8 illustrates this process. The "}" character is assumed to be in the region which is surrounded by the guidelines since its center fall into that region, whereas the character "p"'s not.

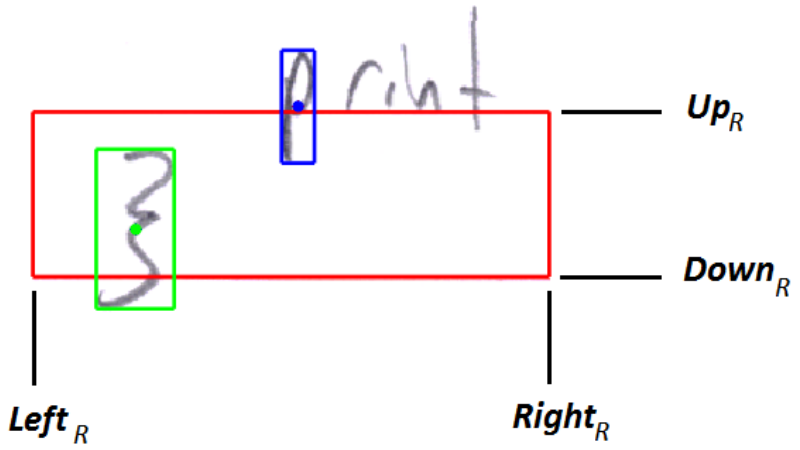



Figure 3.8 Illustration for assignment of labels to regions

A new image was constructed for a region from all labels found in this region by this procedure. According to this, the image of 'p' whereas image of '}' is taken for the image constructed for that region.

As mentioned earlier, segmentation procedure up to now gives handwriting image's character level segmentation for the part ROI 1 and text line level segmentation for the part ROI 2.

We then segmented the text lines into the characters by means of vertical histogram projection on each text lines which are segmented previously. Vertical histogram projection for a sample text line is depicted in the Figure 3.9.


Figure 3.9 Vertical histogram projection of a text line

By thresholding, vertically non-overlapping characters are segmented. Overlapping characters are discarded in the manual labeling step, later.

### *3.1.3 Normalization*

Normalization step consist of intensity adjustment and resizing to make all the character images the same size.

Intensities adjusted by multiplying with a constant and clipping values higher than the maximum intensity (255).

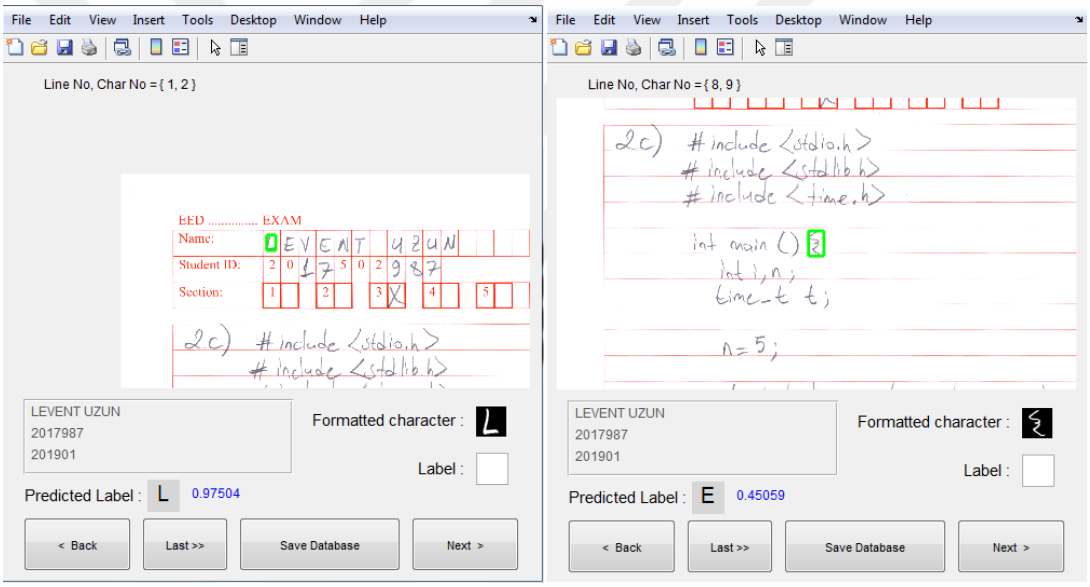
$$D_{i,j} = C_{i,j} * (255/63)$$
$$If\ D_{i,j} > 255,\ D_{i,j} = 255, i = 1,..,r\ ;\ j = 1,..,s \tag{3.12}$$

where *C* is the character image *r* x *s* sized matrix, *D* is contrast adjusted image matrix of the same size as the *C*.

Obtained character image sizes are adjusted to 28 x 28 by bicubic interpolation. The output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood. There is an exception for resizing of small characters like dot since resizing yields abnormal expansion of small characters and make them lose their meaning. We applied an exception and did not resize a character if their width and height is smaller than a threshold value which is some proportion of the average of text line heights. We used 0.2 for this threshold value.

### *3.1.4 Manual Labeling*

In this step we used our own developed application which implements all the steps discussed so far. In the application, segmented and normalized characters are shown and asked user to label it. Character's original location is also indicated on the original image to help the user make correct labeling. Additionally, a Neural Network's prediction is also shown to assist manual labeling procedure. This NN was trained by handwritten characters from EMNIST database which only have alphanumeric characters, i.e. digits and letters. Its predictions were wrong when it encounters with source code specific characters. Figure 3.10 shows two screenshots of the application in which wrong prediction and right prediction was made.




(a) (b)

Figure 3.10 Manual labeling application that implements all the steps so far. Two instances show correct prediction (a) and wrong prediction (b)

Formatted character images labeled by the user were saved to our database. Users discarded wrongly segmented characters here.

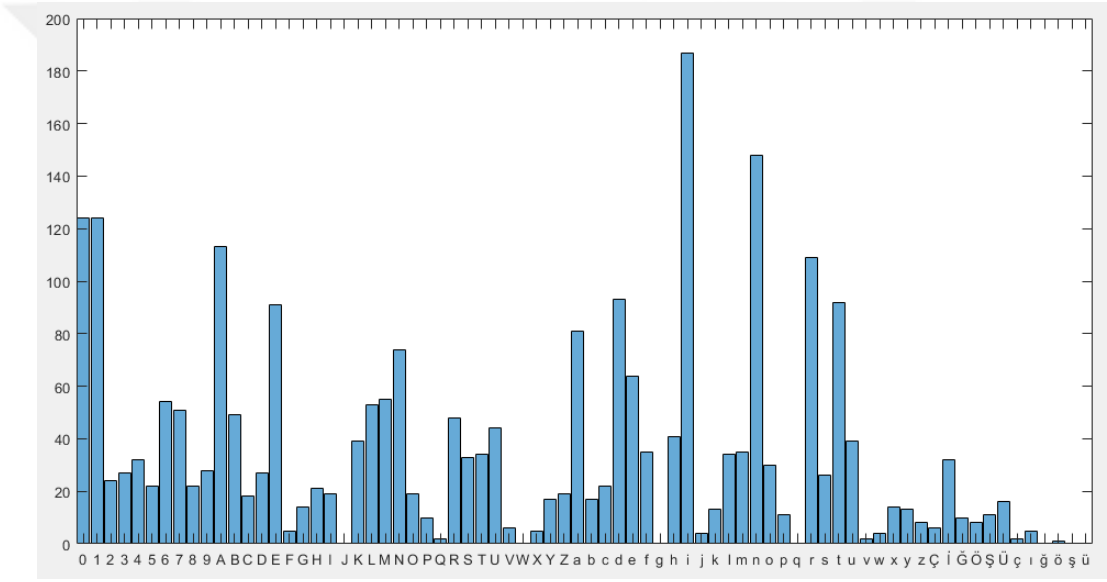
We processed and labeled 116 images by this application. We saved 2536 alphanumeric character samples, i.e. digits and uppercase and lowercase letters, including Turkish specific ones. And we saved 4557 handwritten non-alphanumeric

character samples such as punctuations, parentheses, and other C programming language source code specific characters.

Alphanumeric characters:
**0123456789ABCDEFGHIKLMNOPQRSTUVXYZabcdefhijklmnoprstuvwxyzÇİĞÖŞÜçıö**

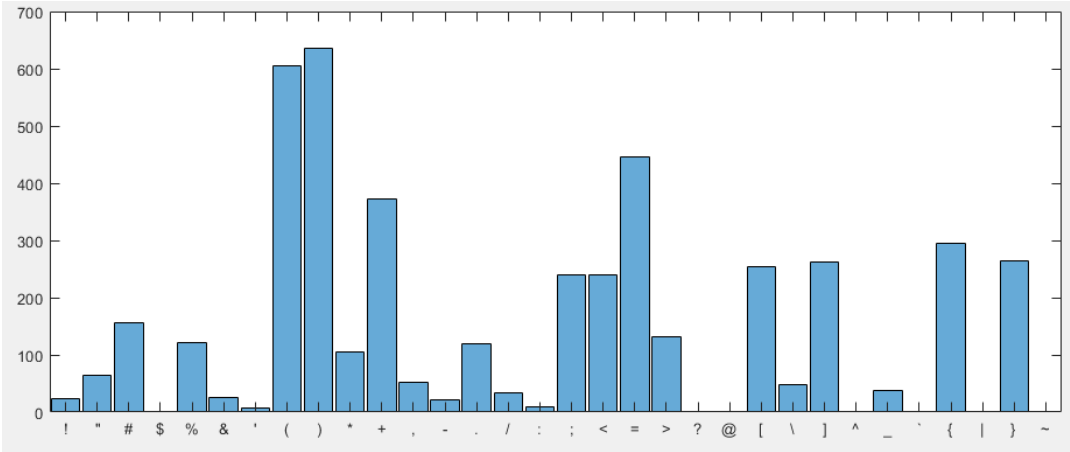
Non-alphanumeric characters: **!"#%&'()*+,-./:;<=>[\]_{}**

Our overall database is composed of 7093 labeled characters. Figure 3.11 shows samples and their amounts in our database, graphically.

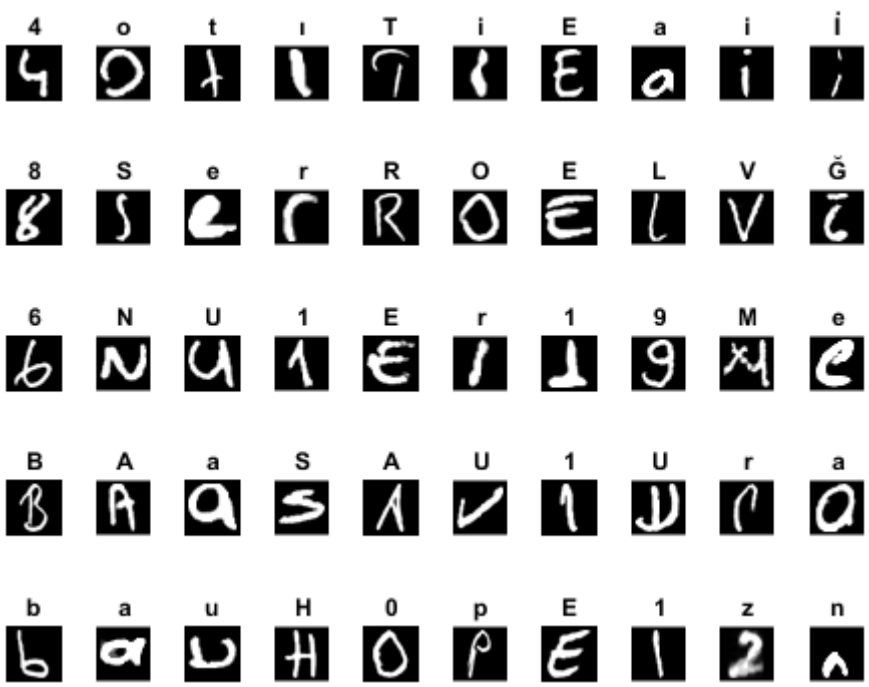



(a)

Figure 3.11 Histogram of character distribution of alphanumerics in our dataset (a), histogram plot of character distribution of non-alphanumerics in our dataset (b), samples of alphanumeric characters with labels (c), and samples of non-alphanumeric characters with labels (d)

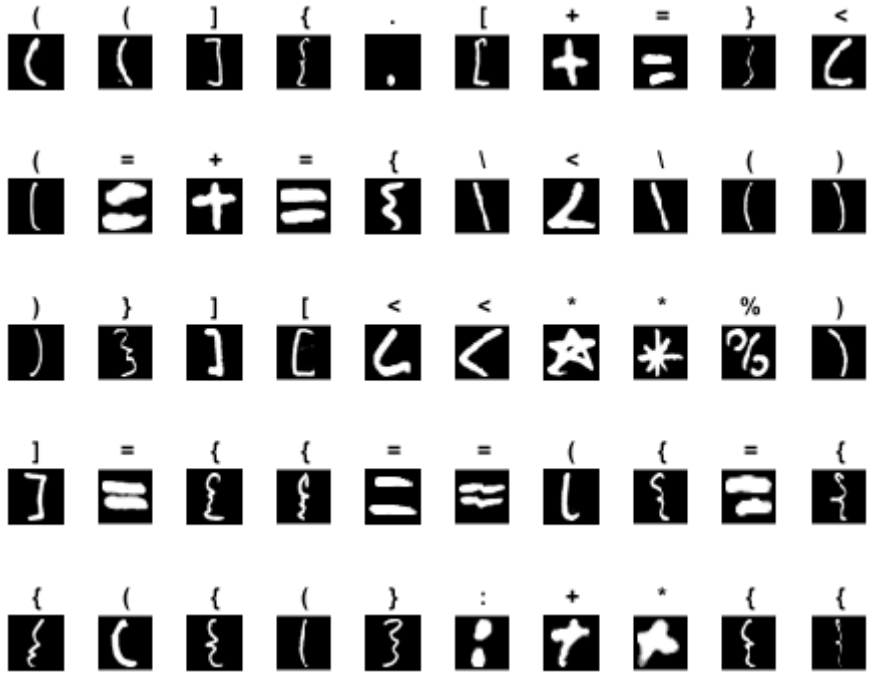



(b)




(c)

Figure 3.11 continues



(d)

Figure 3.11 continues

### *3.1.5 Dataset Enrichment, CROHME Dataset*

To increase number of samples, we enriched our dataset by using CROHME dataset which is publicly available. CROHME is Competition on Recognition of Online Handwritten Mathematical Expressions, being organized since 2011. All data is freely available for research purposes (*CROHME: competition on recognition of online handwritten mathematical expressions.* n.d.). CROHME samples are mathematical expressions in InkML format and saved in an INKML file. This file contains the following information:

- The traces which are called as the *"Ink"*. This is online handwriting information.
- Segmented symbols and corresponding labels contained in the expression.
- The structural information of the mathematical expression in MATHML format.

The last two information (symbols and mathematical structure) are entered manually. File may contain further information like writer information and LATEX if they are added. We didn't use that kind of information.

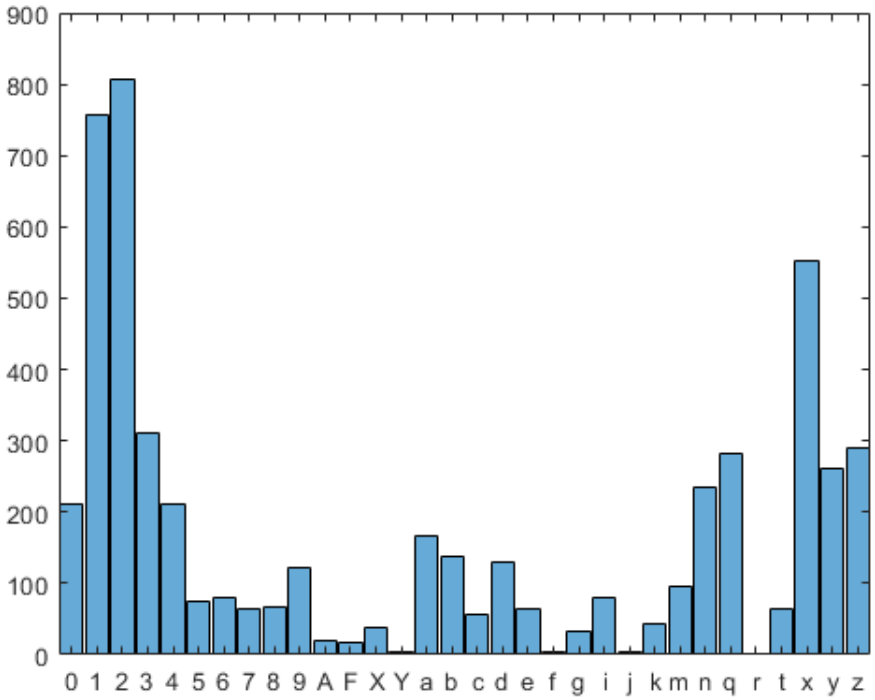
Although CROHME dataset is for recognition of online handwritten mathematical expressions, it has common symbols with the C programming language source code, such as mathematical operators and parentheses. We chose the segmented symbols and their labels information in InkML data and converted them to the same format with the sample characers' images of our dataset, i.e., 28x28 intensity images. But resulting images had only black and white colors and they didn't contain gray level information since it is absent in the InkML format.

We enriched our dataset by merging ones we took from CROHME dataset. We took 5281 alphanumeric and 5374 non-alphanumeric handwritten characters with their labels.

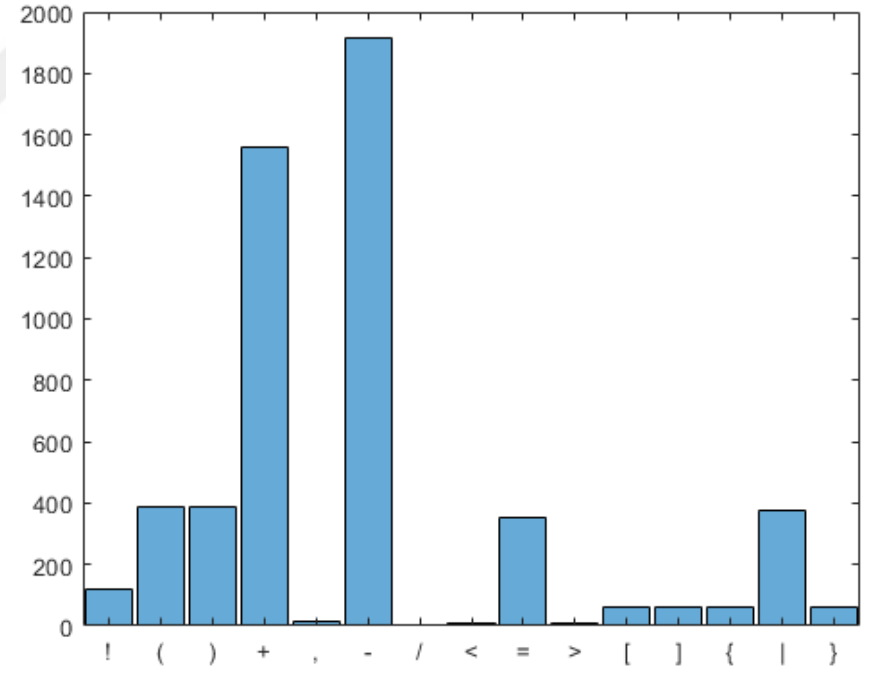
We took the following characters from CROHME dataset:

| | |
|---|---|
| Alphanumerics: | **0123456789AFXYabcdefgijkmnqrtxyz** |
| Non-alphanumerics: | **!()+,-/<=>[]{\|}** |

Figure 3.12 shows samples and their amounts taken from CROHME dataset, graphically.

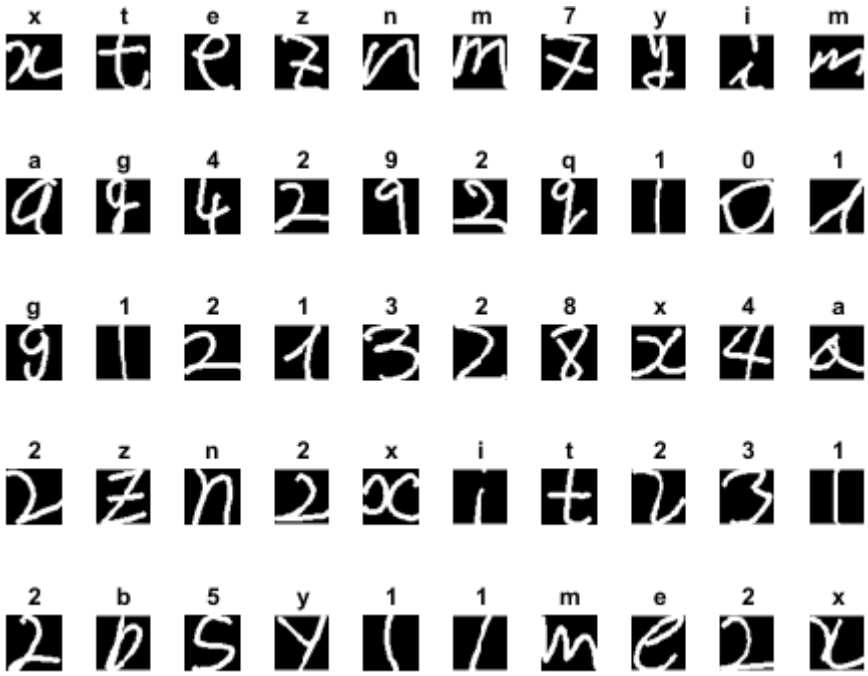



(a)

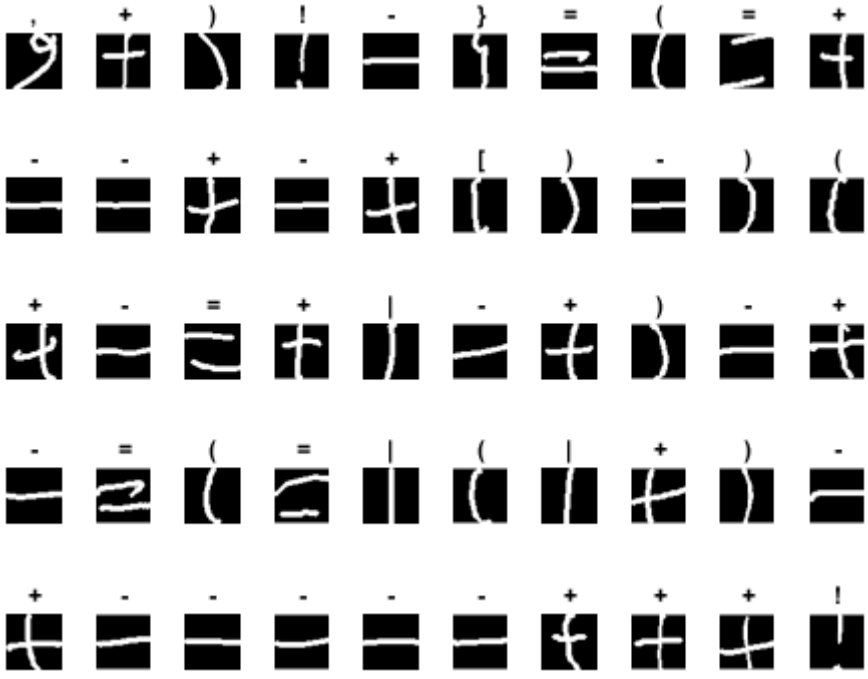



(b)

Figure 3.12 Histogram plot of all alphanumeric characters taken from CROHME dataset (a), histogram plot of all non-alphanumeric characters taken from CROHME dataset (b), random samples from CROHME alphanumeric characters (c), random samples from CROHME non-alphanumeric characters (d)




(c)




(d)

Figure 3.12 continues

After merging, dataset overall volume is 17748 offline handwritten characters with labels, and this was used to train and test of CNNs. Final character set after merge:

**0123456789ABCDEFGHIKLMNOPQRSTUVXYZabcdefghijklmnopqrstuvwxyzÇİĞÖŞÜçıö!"#%&'()*+,-./:;<=>[\]_{|}**

There are 95 classes in merged dataset.

### 3.2 Training and Test of CNNs

In this section we submit our methodology that we used for experiments to assess the handwritten source code character recognition performances of the given CNN topologies in the following subsection and a table of the results. We used *MATLAB Deep Learning Toolbox* as the computation platform.

In recognition stage, we used segmented character's images and classify them as one of the character in the lexicon. Previous character segmentation output is the input of the handwritten recognition system. Correct segmentation of characters is important for this stage to work properly. We obtained characters by above steps and eliminated wrongly segmented characters manually.

We applied CNNs on the *recognition of handwritten source code characters* task on which there is only few work done before.

#### *3.2.1 CNN Topologies Used*

We used different CNNs with different layers and parameters for comparison. We will use the symbols given in the Figure 3.13 to explain the topology of CNNs that we trained and tested. CNN models' topological presentations are given in Figures 3.14 – 3.19.

Please note that, all convolutional layers are followed by a batch normalization layer and a ReLu layer. All networks end with a softmax layer and a classification output layer sequentially. They are not shown explicitly.

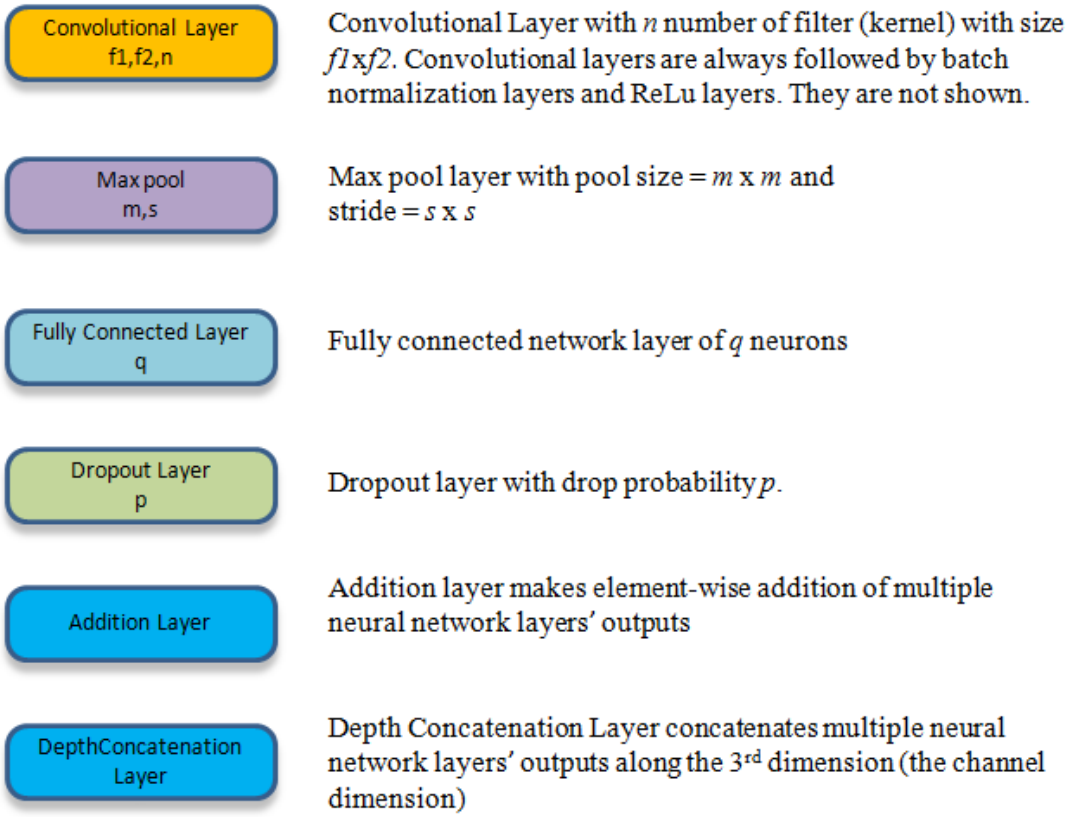



Figure 3.13 Symbols used to draw CNN topologies

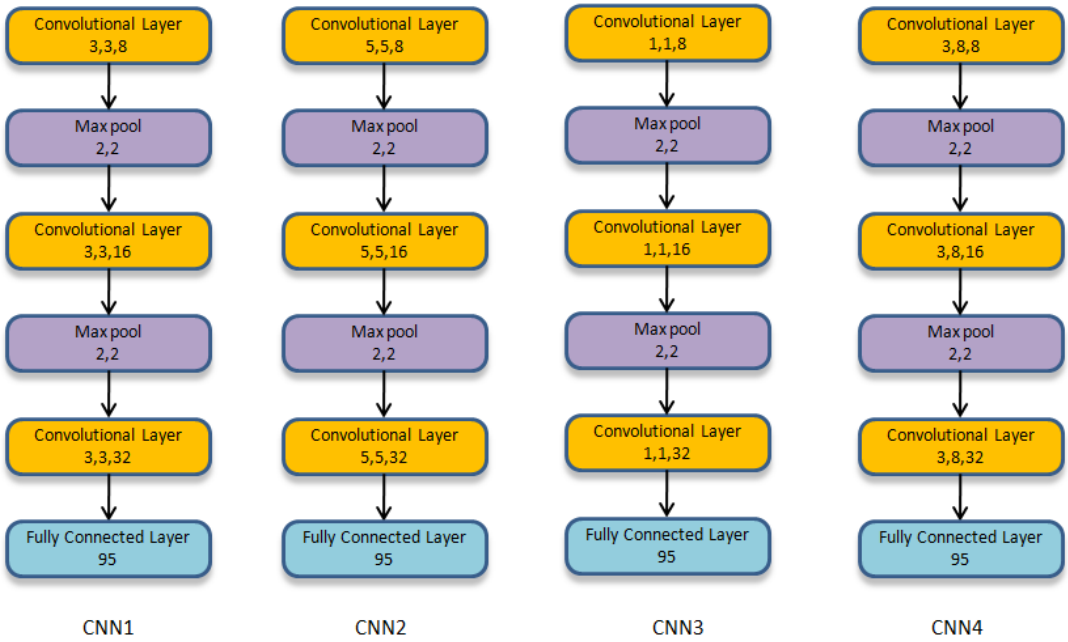



Figure 3.14 CNN topologies: CNN1-CNN4

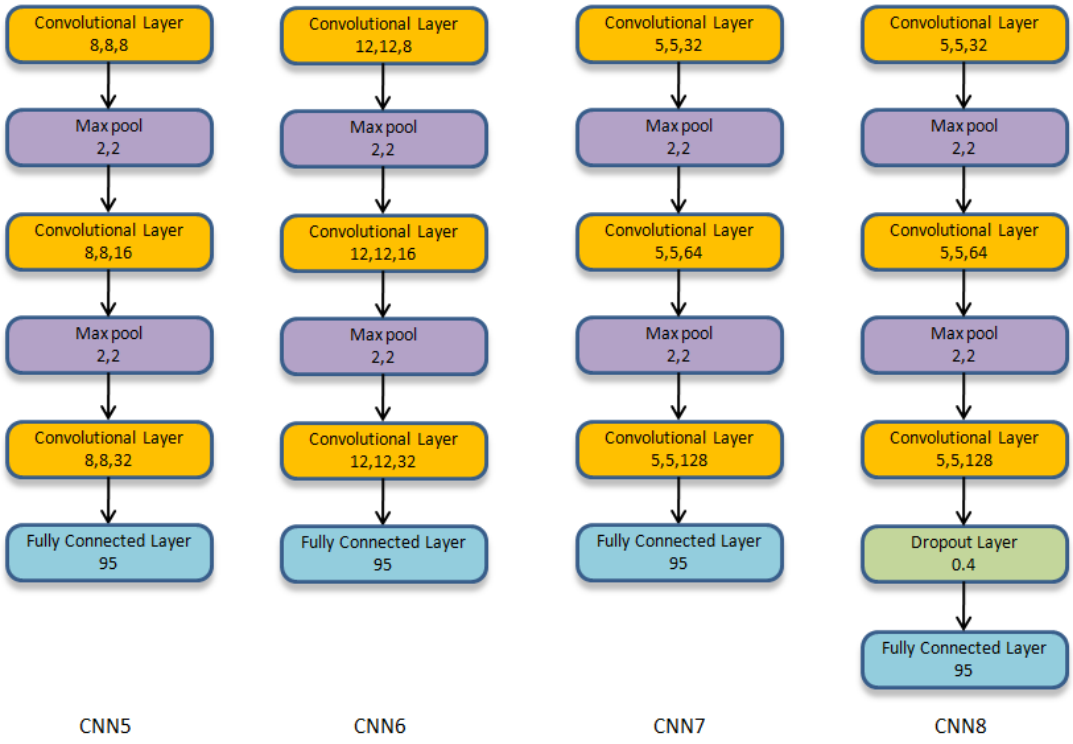



Figure 3.15 CNN topologies: CNN5-CNN8

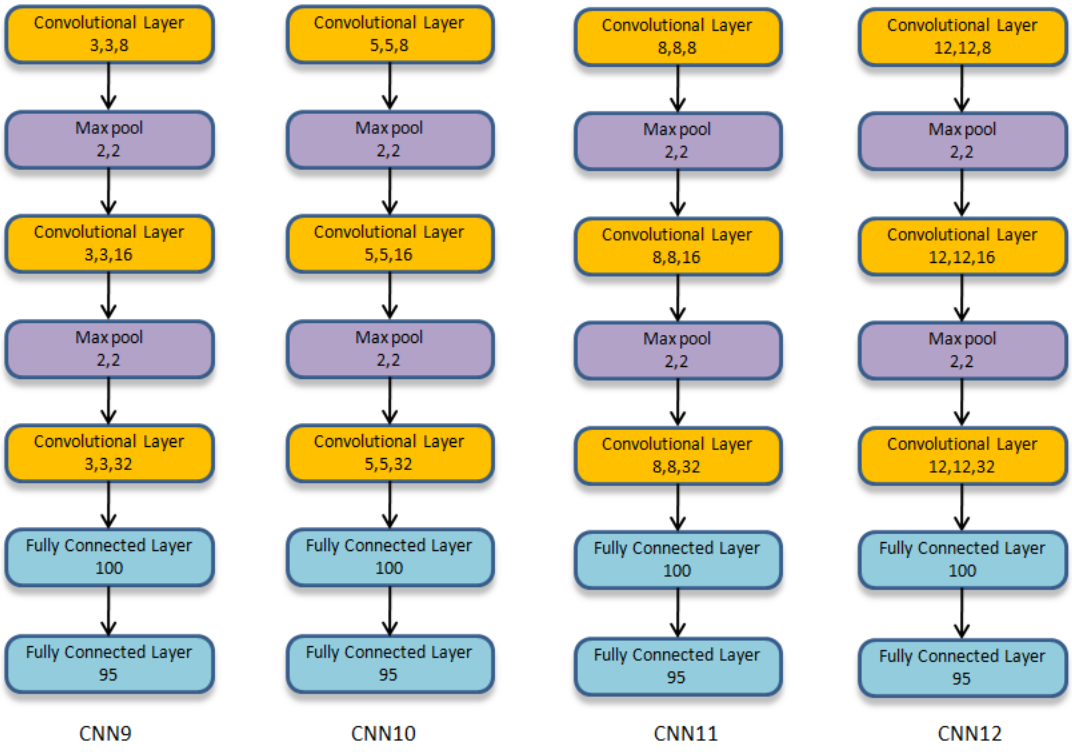



Figure 3.16 CNN topologies: CNN9-CNN12

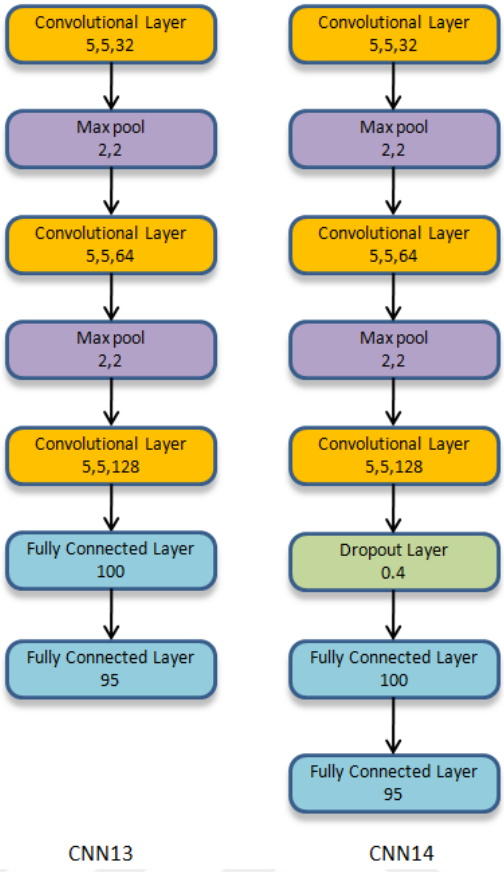



Figure 3.17 CNN topologies: CNN13 and CNN14

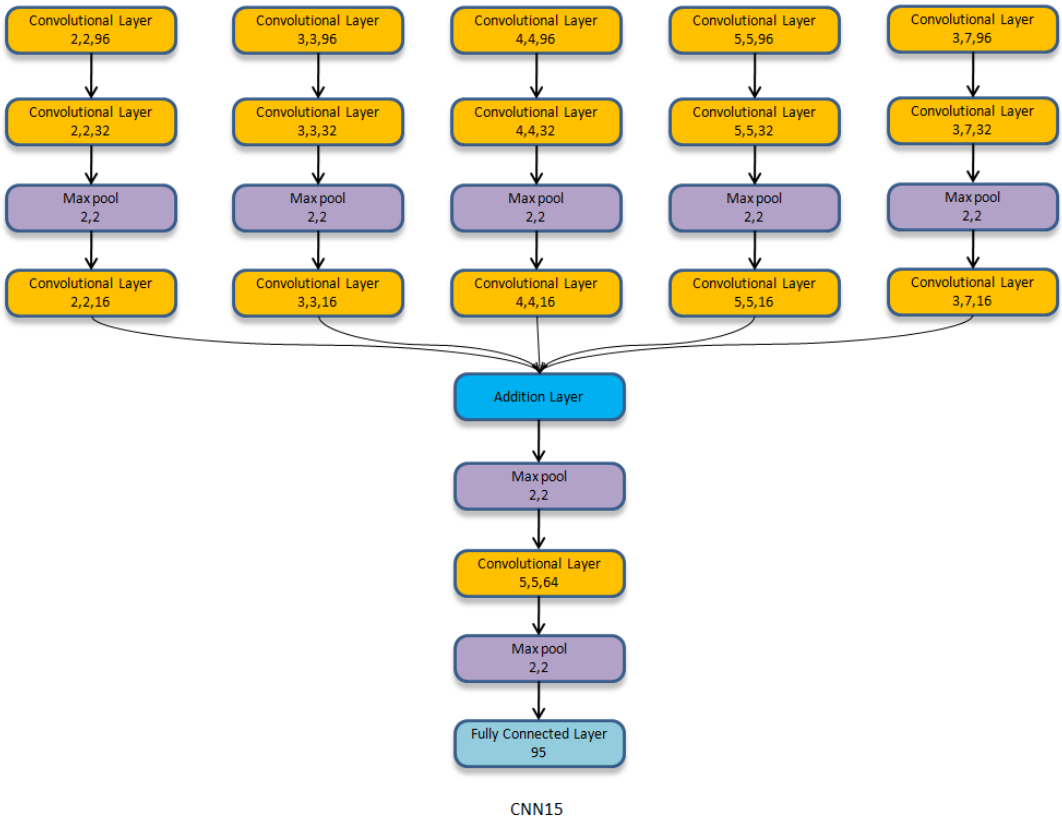



Figure 3.18 CNN topologies: CNN15

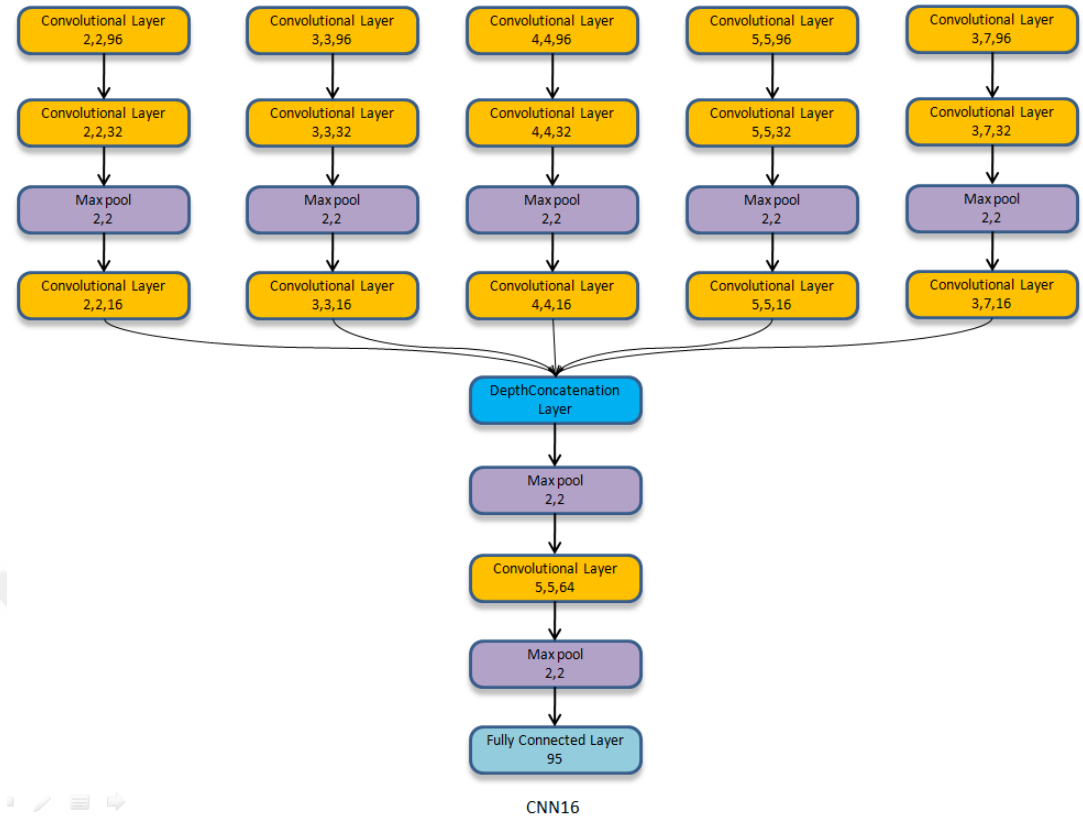



Figure 3.19 CNN topologies: CNN16

### *3.2.2 Training*

| | |
|---|---|
| All alphanumerics from our dataset: | 2536 samples |
| All non-alphanumerics from our dataset: | 4557 samples |
| All alphanumerics from CROHME dataset: | 5281 samples |
| All non-alphanumerics from CROHME dataset: | 5374 samples |

Totally 17748 samples (images with labels) are used for training, validation and test of the networks. Samples are allocated for each process by randomly subsampling from the entire dataset of 17748 samples. Validation and test sets are each chosen as 1/10 of the total. The remaining 8/10 is used for training.

Validation set was used to check the recognition accuracy of the network under training for multiple times during the training process to see if the network was getting over-fitted or over-trained. This may occur when the network “memorize” the

samples which is not desired. Below the Figure 3.20 depicts the graphs of accuracy and loss recorded progressively during the training process:

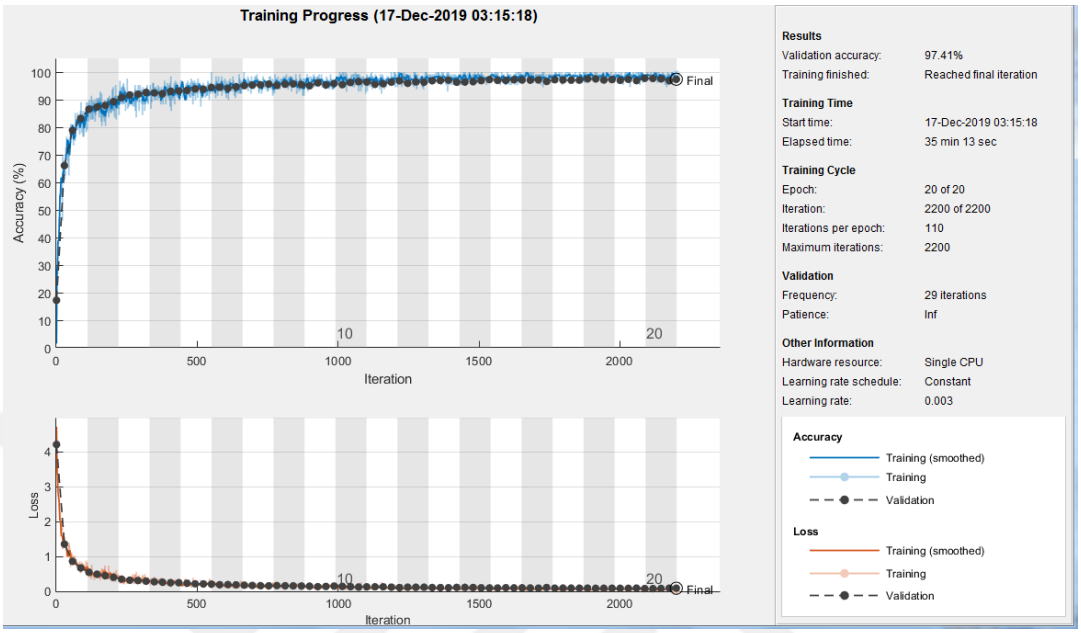



Figure 3.20 Accuracy and loss trends recorded during training of a network

We used "*stochastic gradient descent with momentum"* algorithm for training. This algorithm aims to find the global minimum of the *cost function* in terms of the learnable parameters of the network such as weights and biases. Training time is related to the number of those parameters. Larger networks with more parameters require longer training times. *"Stochastic"* term implies that the parameters are updated after mini-batches instead of waiting full batch. This technique increases the training speed.

Training process yields a network whose all learnable parameters are calculated so as to minimize the loss for the given training set. These parameters then are fixed and saved along with the network structure. Subsequent tests are carried out with these parameters.

*3.2.2.1 Deep Dream Images*

These images give an insight about a network in terms of its weights at different layers. They are calculated so as to activate corresponding kernel strongly. In this way, images represent the features that were learnt by the network.

Below Figures 3.21 – 3.23 show such exemplary images for all three convolutional layers of the CNN7 sequentially, which are calculated after the network was trained. It is interesting to notice how the details are increasing as we move to the deeper layers.

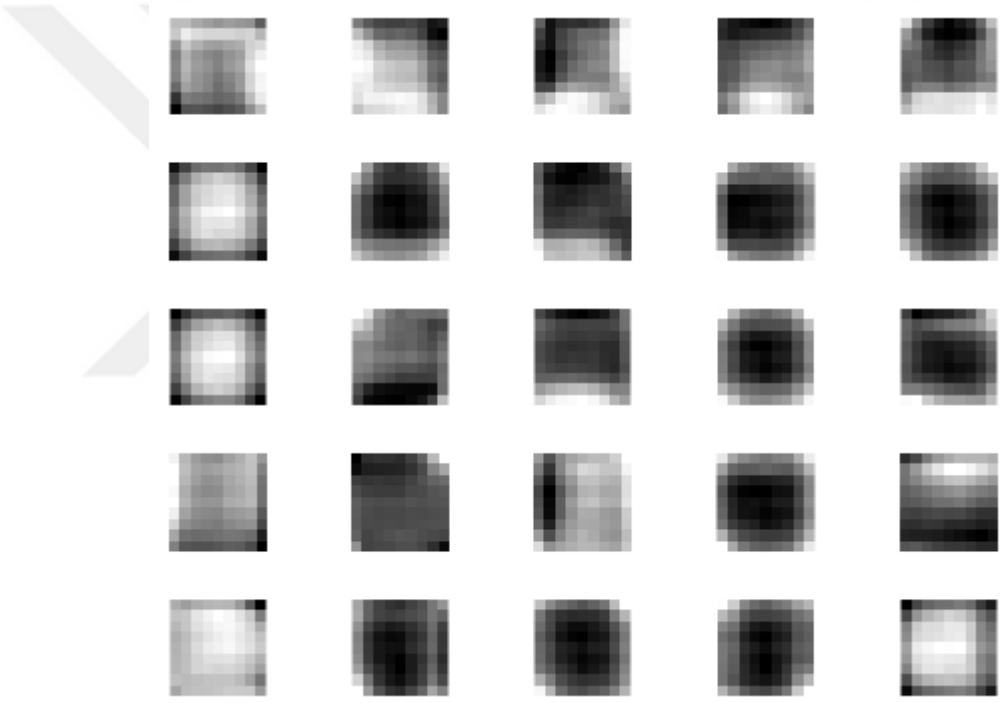


Figure 3.21 Deep dream images of convolutional layer-1 for the first 25 channels

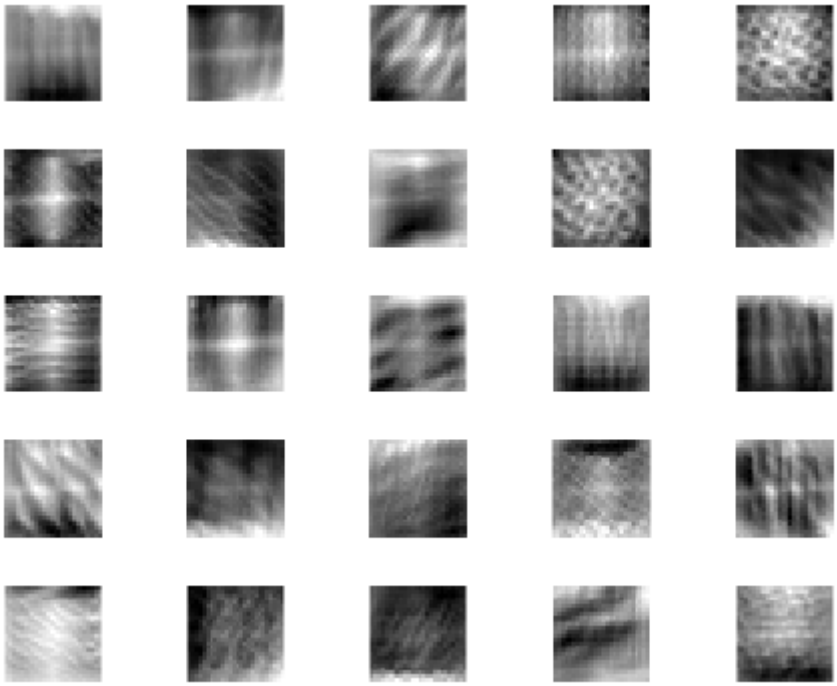


Figure 3.22 Deep dream images of convolutional layer-2 for the first 25 channels

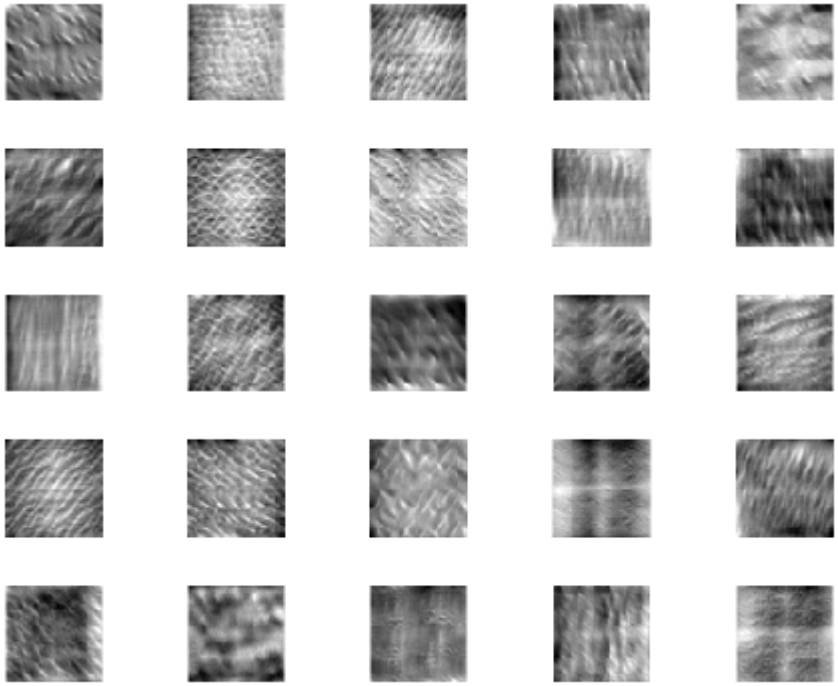


Figure 3.23 Deep dream images of convolutional layer-3 for the first 25 channels

*3.2.2.2 Activations*

Activation is the output of a certain unit in the network. It is worth investigating layer activations of the trained network for a specific input. This may give insight about layer's function. Examining the activations may help to figure out the features that the network learnt. As mentioned earlier CNN networks learn hierarchical features therefore they extract hierarchical representations of the input. Latter layers give higher-level features by means of lower-level features output from the former layers. We present three convolutional layers' activations sequentially in the Figures 3.25 – 3.27 for an exemplary input image which is shown in Figure 3.24. Each convolutional layer has several filters (or kernels) which are called as channels. Activations of a convolutional layer simply mean the filter responses of these channels. Whiter areas in the channel's activation image imply greater response of that channel to the corresponding areas in the input image.

Figure 3.24 Exemplary input image for activations

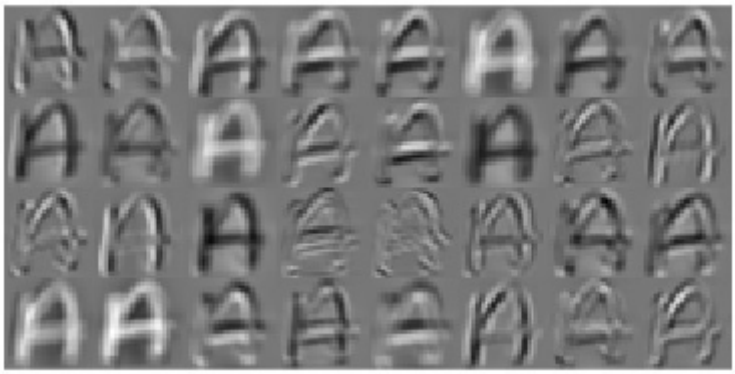


Figure 3.25 Activations of convolutional layer -1

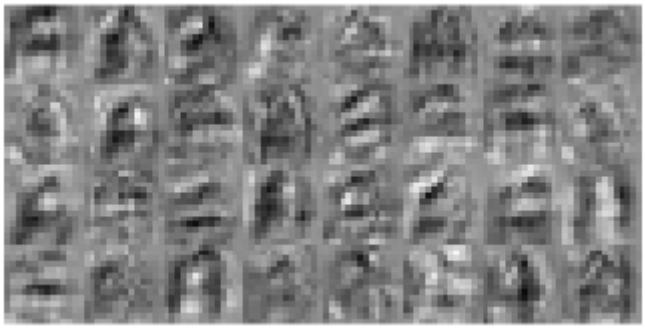


Figure 3.26 Activations of convolutional layer -2

Figure 3.27 Activations of convolutional layer -3

Some activations, especially in the first layer, appear to represent edge detections which are occurred in the corresponding convolutional layer. Activations of deeper layers are more difficult to make sense since they may represent non-visual features such as connections or other relational features. Moreover, outputs at deeper layers have smaller dimensions because of the reduction effect of intervening max pool layers and stride parameter of convolutional layers.

### *3.2.3 Test Results and Comparisons*

Test sets are used to get a reliable accuracy of the recognition with "unseen" data. That means, tests were conducted with samples which were not used before for training of the network under test.

First, we did 1 test for all the networks. Then we chose a successful one and did tests with 10-fold cross-validation to get more accurate results.

Test results for the CNN networks whose topological diagrams were given previously are shown in the Table 3.1 below. This table shows their validation and test performances for one training and test sequence (1-fold) of each.

Table 3.1 Validation and test accuracies of all the CNNs for 1-fold tests

| Network | Validation accuracy % | Test Accuracy % |
|---|---|---|
| CNN1 | 95.94 | 95.72 |
| CNN2 | 95.83 | 95.43 |
| CNN3 | 83.03 | 85.68 |
| CNN4 | 96.00 | 96.05 |
| CNN5 | 96.45 | 95.77 |
| CNN6 | 96.62 | 96.05 |
| **CNN7** | **97.97** | **98.20** |
| CNN8 | 96.45 | 97.13 |
| CNN9 | 93.07 | 92.33 |
| CNN10 | 93.46 | 93.18 |
| CNN11 | 93.86 | 94.08 |
| CNN12 | 93.69 | 93.63 |
| CNN13 | 97.01 | 95.89 |
| CNN14 | 96.28 | 95.38 |
| CNN15 | 97.63 | 97.18 |
| CNN16 | 97.63 | 97.52 |

10-Fold cross validation for CNN7 is given in the Table 3.2.

Table 3.2 Validation and test accuracies of CNN7 for 10-fold cross-validation

| Network | Validation accuracy % | Test Accuracy % |
|---|---|---|
| CNN7 | 97.46 | 98.82 |
| | 97.63 | 97.18 |
| | 96.67 | 96.34 |
| | 97.52 | 97.52 |
| | 97.41 | 97.52 |
| | 97.91 | 97.46 |
| | 97.13 | 97.29 |
| | 97.18 | 97.80 |
| | 97.01 | 97.46 |
| | 98.32 | 97.63 |
| Average (%): | 97.424 | 97.502 |
| Standard deviation (%): | 0.47 | 0.61 |

In 1-fold tests, test accuracies vary between 92.33% and 98.20%. Validation accuracies reached highest value of 97.97%. Both highest values belong to the same network model, CNN7. Therefore, we chose that model for further test at which we used 10-fold cross validation. That model got 98.82% test accuracy at most and 97.5% average of all 10 tests.

The Figure 3.28 below shows samples from test set which are misclassified by CNN7 after 97.97% test accuracy:

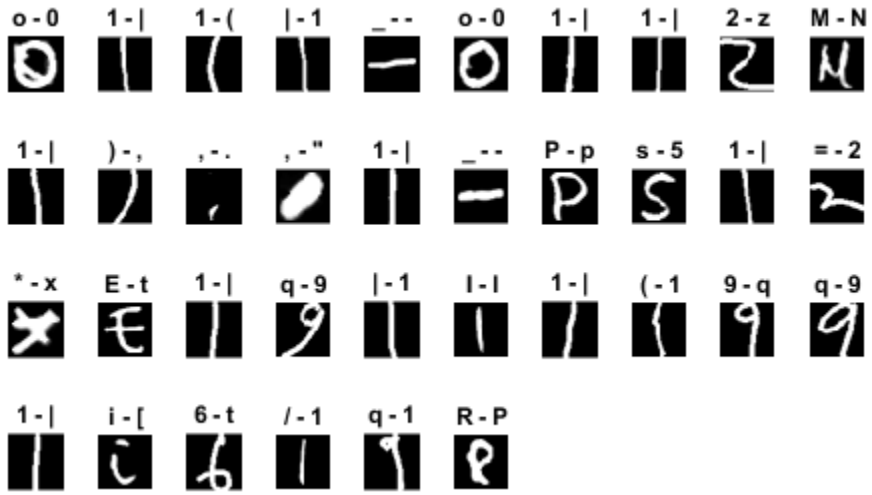



Figure 3.28 Misclassified samples by CNN7 after 97.97% test accuracy. Labels on the left are the correct ones; labels on the right are the predictions of the network in the titles of the character images

It can be useful to inspect the histogram of misclassified characters which is given in the Figure 3.29.

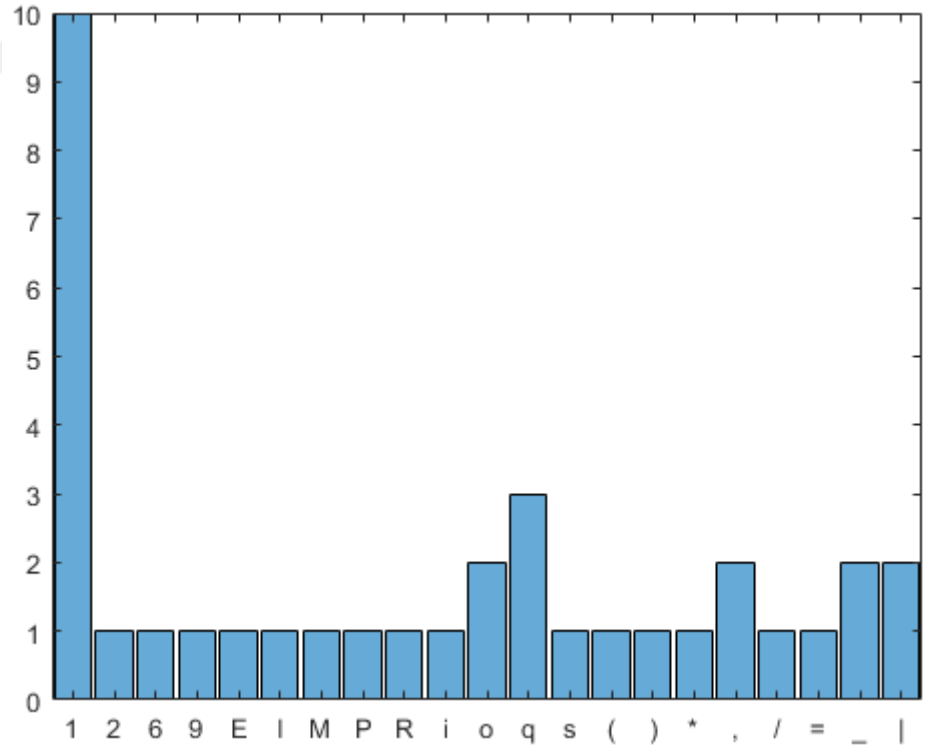



Figure 3.29 Histogram of misclassified characters

We think these results are quite satisfactory when character recognition performances in the literature are considered. We decided to review and tabulate the performances of some of the works mentioned in the *"2.5 Related Works"* section along with our results in the Table 3.3. Although there is not exact match between the tests, it may give an idea about how successful our results are.

Table 3.3 Performances of related works and ours.

| Authors | System | Data | Training Dataset Volume | Performance % |
|---|---|---|---|---|
| Lu & Mohan, 2015 | CNN | CROHME Handwritten ME with 75 classes | 33940 | 89.7 |
| Nguyen, Le, & Nakagawa, 2015 | Combined DMCN and BLSTM | CROHME Handwritten ME with 101 classes | 120341 | 91.28 |
| Alvaro, Sánchez, & Benedí, 2014 | RNN-BLSTM | CROHME Handwritten ME with 101 classes | 77400 | 87.1 |
| Cireşan, Meier, Gambardella, & Schmidhuber, 2011 | CNN Committees | NIST SD 19 database Handwritten characters with 62 classes | 482925 | 88.12* |
| Zhi & Metoyer, 2017 | Combined with an commercial product with an unknown mechanism | Source code samples collected from 15 participants and consisting of 555 lines of Phyton code | unknown | 96.4** |
| **Ours** | **CNN** | **Our own + CROHME, totally 95 classes** | **14200** | **97.5** |

* Converted from the error rate: (1- error rate).

** Converted from the CER: Character error rate obtained from the Levenshtein distance: (1-CER)

Although they use different metrics for performance measurements, we think our work got one of the most successful results.

# CHAPTER FOUR
# DISCUSSION

As mentioned earlier, we used exam papers of C programming language course for this work. In spite of guidelined design of the papers helped the processing of the scanned documents, dataset creation has taken considerably much time, especially the segmentation phase. We gathered all the image processing steps under an application which presents an interface to user for picking up segmented and formatted character images and assigning labels to them easily and quickly. We used that application and created our own database. Because of the segmentation algorithm's limited ability, especially overlapping handwritten characters couldn't be segmented correctly and discarded by the user. This reduced the number of samples collected from a paper.

Another difficulty was the plenty amount of improperly written characters by the students. We may also have selected uneven ones which could have affected test results negatively.

CROHME dataset contains originally online characters. Conversion of them to offline characters has some drawbacks which introduces some distortions on the samples. Converted samples also have some differences from our dataset's samples such as the lack of surrounding 2 pixel wide padding and absence of gray levels. These factors might have affected the performance of our work adversely.

First results of ours were relatively low (about 91%). After rearranging dataset ordering randomly, we got these higher results. We think that implies the importance of random subsampling.

We also tested other variants of the given networks with different hyper parameters. These tests reached the similar accuracies (97.91%, at most) with the above given tests but couldn't get more success than CNN7.

It can be seen from our results that dropout layer has negative effect on the performance. This might be caused by using it with batch normalization layer together which was discouraged in the literature.

Work of Zhi & Metoyer (2017) seems to be the most related work with ours and has the most successful result although it uses different metric for performance assessment of character recognition. But this work uses another recognition engine and augmented that with additional source code specific algorithm as a postprocessing phase. That recognition engine is called as *'MyScript'* which is formerly known as *"Vision Objects"* (Lu & Mohan, 2015). Its machine vision and recognition system is unknown since it is commercial. It probably uses combined techniques of NN and programmatic ones but we are not sure if it contains CNN or not. This engine works on touch devices such as smart phones or tablet PCs, so it very likely uses online information which distinguishes our work from it because we used fully offline methods only.

We noticed and tried to figure out the reason of our relatively higher results than the related works' performances given in the comparison table above. Although this question requires more detailed examination and cross check of their works with ours, one of the factors could be the batch normalization layer usage. We noticed that none of these works have used it. The reason may be the batch normalization technique was discovered later than these works. Other factor could be the random subsampling even though we are not sure whether they used it or not. We think that their results can be improved by these means and using other state of the art methods.

Results show that if CNNs with proper topology trained with sufficient amount of data and with correct parameters, they can succeed in automated reading of handwritten source code characters as with a human reader. Some characters are very difficult to discriminate such as "I" and "1" and "," and "/" and "|", "U" and "u", "-" and "_", "i" and ";", "(" and "C" and "<", "9" and "q" even for a human reader, without using contextual and positional information.

# CHAPTER FIVE
# CONCLUSION

Even though developing technology is making easy the information encoding and storage every day more and more, handwriting seems to maintain its prevalence for the future decades because of some advantages of it over other techniques. Therefore, recognition of handwritten characters will very likely be growing topic. Besides this, *recognition of handwritten source code characters* is a promising task that will be useful in various cases where pen based input is used such as automatic reading of exam papers for programming language courses as in our case, supporting software developers in various environments like in the evolving field of smart phones and tablet PCs etc. In this work we focused on this task with an ultimate goal of automatic grading of C programming language course's exam papers. Our work can be thought to be the first step of achieving this goal. It can be expanded by adding further steps such as fully digitization of exam paper, semantically extract source codes, processing and evaluating the correctness of it and grading for the identified person. This can be extended to further applications such as automatic reading and processing of source codes in different programming languages, flow charts or pseudocodes, handwriting identification, digitization of various types of documents and so on.

Deep learning technique was chosen as the character recognition tool in this task. Deep neural network architectures have gained increasing popularity in machine learning applications recently because of their high performances. CNN model is one of them which has widespread application fields involving 2D pattern recognition. We think that our work showed CNNs can successfully be utilized in *handwritten source code character recognition* task too.

There is only few works have been made about *handwritten source code character recognition* task by using CNNs. Our test results exhibit that the CNNs are the state of the art models for this task as with other various pattern recognition tasks. If more publicly available source code character databases are created, that would boost the new researches at this field.

Results can be improved by including positional information of the data which is lost by normalization process in the dataset construction phase.

Our dataset will be made publicly available for future works. Our dataset will be a valuable contribution in the research field of handwritten character recognition.

# CHAPTER SIX
# FUTURE WORKS

Some improvements could make the accuracy higher, such as using better filtering techniques in the preprocessing phase, using slant correction methods which we omitted, using higher volume dataset or enrichment of the existing ones by elastic deformations, eliminating the conversion drawbacks and differences of CROHME dataset samples, using more complicated network structures and topologies, testing with different hyper parameters and so on. We think that positional information preserving normalization methods can also improve the results. These are the open options which can be studied for getting the higher accuracies.

Additional symbols can be added to our dataset since we couldn't find samples for every class of characters. Furthermore, different datasets with another set of classes can be created for recognition source code characters of a different programming language.

A segmentation method can be developed such that it can use recognition accuracy of our work for assessing its segmentation accuracy and use it for training itself. This method may try to find largest area for which the recognition accuracy meaningfully high.

An application can be built which adds extra samples detected in a successfully recognized document to a publicly available database or to its own dataset to improve its own recognition performance. That means a self evolving application. This would be useful for learning specific handwriting features of the writer of the document currently being processed if it is capable of train itself when recognizing a document. This could increase the recognition performance by taking into account the writer's specific handwriting features by giving some higher priority to the features of sample characters of the current document upon the previous ones.

A new character dataset can be created which includes most overlapping character combinations and labeled as multiple characters. This method may be useful for overcoming segmentation difficulty especially for most overlapping characters which

is probably specific to a certain language. For example, overlapping and agglutinated characters could be “ft” or “ri” which can be very difficult to separate for current segmentation methods. It can be efficient also for cases in which handwriting of some characters are deformed with regard to its isolated form when used consecutively with some other characters. This approach can also be utilized for recognizing cursive handwriting.

## REFERENCES


Akhand, M.A., Ahmed, M., & Rahman, M.M. (2016). Convolutional neural network training with artificial pattern for Bangla handwritten numeral recognition. *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*, 625-630.

Alvaro, F., Sánchez, J., & Benedí, J. (2014). Offline features for classifying handwritten math symbols with recurrent neural networks. *22nd International Conference on Pattern Recognition*, 2944-2949.

Arica, N., & Vural, F.T.Y. (2001). An overview of character recognition focused on off-line handwriting. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *31* (2), 216-233.

Boureau, Y.L., Ponce, J., & LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML)*, 111-118.

Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A.L. (2014). Semantic image segmentation with deep convolutional nets and fully connected CRFs. *arXiv:1412.7062 [cs.CV]*,1-14.

Chen, M. Y., Kundu, A., & Zhou, J. (1994). Off-line handwritten word recognition using a hidden Markov model type stochastic network. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 16* (5), 481-496.

Choudhary, A., Rishi, R., & Ahlawat S. (2013). A new character segmentation approach for off-line cursive handwritten words. *Procedia Computer Science, 17*, 88-95.

Cireşan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2011). Convolutional neural network committees for handwritten character classification. *Proceedings of the 11th International Conference on Document Analysis and Recognition*, 1135-1139.

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2011a). Flexible, high performance convolutional neural networks for image classification. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, 2*, 1237–1242.

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2011b). High-performance neural networks for visual object classification. *arXiv:1102.0183 [cs.AI]*,1-12.

Cireşan, D. C., Meier, U., Masci, J., & Schmidhuber, J. (2011). A committee of neural networks for traffic sign classification. *International Joint Conference on Neural Networks*, 1918-1921.

Cireşan, D. C., Meier, U., & Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *IEEE Conference on Computer Vision and Pattern Recognition*, 3642-3649.

*CROHME: Competition on recognition of online handwritten mathematical expressions.* (n.d.). Retrieved September 2, 2020, from https://www.isical.ac.in/~crohme/CROHME_data.html

El-Yacoubi, M.A., Gilloux, M., Sabourin, R., & Suen, C.Y. (1999). An HMM-based approach for off-line unconstrained handwritten word modeling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 21* (8), 752-760.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, *1* (2), 119–130.

Hossain, M.B., Naznin, F., Joarder, Y.A., Islam, M.Z., & Uddin, M.J. (2018). Recognition and solution for handwritten equation using convolutional neural network. *Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, 250-255.

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology, 160* (1), 106–154.

Hussain, B., & Kabuka, M.R. (1994). A novel feature recognition neural network and its application to character recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 16* (1), 99-106.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on International Conference on Machine Learning, 37,* 448–456.

Jain, A.K., Duin, R.P., & Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22* (1), 4-37.

Knerr, S., Augustin, E., Baret, O., & Price, D. (1998). Hidden Markov model based word recognition and its application to legal amount reading on french checks. *Computer Vision Image Understanding 70* (3), 404–419.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86* (11), 2278–2324.

LeCun, Y., Huang, F.J., & Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (CVPR), 2*, II-104

Lefkimmiatis S. (2017). Non-local color image denoising with convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5882-5891.

Liou, C. Y., & Yang, H.C. (1996). Handprinted character recognition based on spatial topology distance measurement. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 18* (9), 941-945.

Lu, C., & Mohan, K. (2015). *Recognition of online handwritten mathematical expressions using convolutional neural networks.* CS221 final project, Computer Science, Stanford University, Standford.

Lu, Y., & Shridhar, M. (1996). Character segmentation in handwritten words—An overview. *Pattern Recognition, 1* (29), 77-96.

Mehul, G., Ankita, P., Namrata, D., Rahul, G., & Sheth, S. (2014). Text-based image segmentation methodology. *Procedia Technology, 14,* 465-472.

Miah, M.B.A., Yousuf, M.A., Mia, S., & Miya, M.P. (2015) Handwritten courtesy amount and signature recognition on bank cheque using neural network. *International Journal of Computer Applications 118* (5), 21-26.

Milletari, F., Navab, N., & Ahmadi, S. (2016). V-net: Fully convolutional neural networks for volumetric medical image segmentation. *Fourth International Conference on 3D Vision (3DV)*, 565-571.

Nguyen, H. D., Le, A. D., & Nakagawa, M. (2015). Deep neural networks for recognizing online handwritten mathematical symbols. *3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 121-125.

Pal, U., & Datta, S. (2003). Segmentation of Bangla unconstrained handwritten text. *Proceedings of the 7th International Conference on Document Analysis and Recognition,* 1128-1132.

Palacios, R., Gupta, A., & Wang, P.S. (2004). Handwritten bank check recognition of courtesy amounts. *International Journal of Image and Graphics, 4* (2), 1-20.

Palehai, D., & Fanany, M. I. (2017). Handwriting recognition on form document using convolutional neural network and support vector machines (CNN-SVM). *5th International Conference on Information and Communication Technology (ICoIC7)*, 1-6.

Rashid, S.F. (2014). *Optical character recognition - A combined ANN/HMM approach*. Phd Thesis, Technical University of Kaiserslautern, Kaiserslautern.

Sanchez, A., Suarez, P. D., Mello, C. A. B., Oliveira, A.L.I., & Alves, V.M.O. (2008). Text line segmentation in images of handwritten historical documents. *First Workshops on Image Processing Theory, Tools and Applications*, 1-6.

Santos, R. P. d., Clemente, G. S., Ren, T. I., & Cavalcanti, G. D. C. (2009). Text line segmentation based on morphology and histogram projection. *Proceedings of the 10th International Conference on Document Analysis and Recognition*, 651-655

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III,* 92–101

Shruthi A., & Patel M. S. (2015). Offline handwritten word recognition using multiple features with SVM classifier for holistic approach. *International Journal of Innovative Research in Computer and Communication Engineering, 3* (6), 5989-5995.

Singh, M.J.K., Dhir, R., & Rani, R. (2011). Performance comparison of Devanagari handwritten numerals recognition. *International Journal of Computer Applications, 22* (1), 1-6.

Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. *Seventh International Conference on Document Analysis and Recognition,* 958-963.

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

Thungamani, M., & Kumar, P.R. (2012). A survey of methods and strategies in handwritten Kannada character segmentation. *International Journal of Science Research*, *1* (1), 18-23.

Tian, C., Xu, Y., Fei, L., Wang, J., Wen, J., & Luo, N. (2019). Enhanced CNN for image denoising. *CAAI Transactions on Intelligence Technology, 4* (1), 17 – 23.

Trier, Ø. D., Jain, A. K., & Taxt, T. (1996). Feature extraction methods for character recognition - A survey. *Pattern Recognition, 29* (4), 641-662.

Vinciarelli, A. (2002). A survey on off-line cursive word recognition. *Pattern Recognition, 35* (7), 1433-1446.

Wanchoo, A., Yadav, P., & Anuse, A. (2016). A survey on Devanagari character recognition for Indian postal system automation. *International Journal of Applied Engineering Research, 11*, 4529-4536

Zhi, Q., & Metoyer, R. (2017). Recognizing handwritten source code. *Proceedings of the 43rd Graphics Interface Conference (GI '17)*, 163-170.