DEEP LEARNING BASED OFFLINE HANDWRITTEN CHARACTER RECOGNIZER SYSTEMS WITH A MULTILINGUAL HANDWRITTEN CHARACTER DATASET

Gaye EDİBOĞLU BARTOS

PhD Dissertation

Department of Computer Engineering

Programme in Computer Engineering

Supervisor: Prof. Dr. Yaşar HOŞCAN

(Co-Supervisor: Assoc. Prof. Dr. Éva NAGYNÉ HAJNAL)

Eskişehir

Eskişehir Technical University

Institute of Graduate Programs

January 2021

ABSTRACT

DEEP LEARNING BASED OFFLINE HANDWRITTEN CHARACTER RECOGNIZER SYSTEMS WITH A MULTILINGUAL HANDWRITTEN CHARACTER DATASET

Gaye EDİBOĞLU BARTOS

Department of Computer Engineering

Programme in Computer Engineering

Eskişehir Technical University, Institute of Graduate Programs, January 2021

Supervisor: Prof. Dr. Yaşar HOŞCAN

(Co-Supervisor: Assoc. Prof. Dr. Éva NAGYNÉ HAJNAL)

Despite decades of research, offline handwriting recognition is still an unresolved research problem. Advancements in deep learning led to a boost in image processing domain in general including the recognition of offline handwritings. In order to max out the capabilities of deep learning-based methods, a large input set is essential. However, these is a lack of publicly available handwriting datasets, especially in certain languages. Absence of handwritten character datasets in Turkish and Hungarian was a prompt to create a handwritten character dataset in those languages. In this work, a public domain multilingual handwritten character dataset is generated.

In addition to the proposed T-H-E Dataset, two different offline multilingual handwriting recognition systems were developed. The first one is a segmentation-based recognizer, using a novel deep learning architecture put forward in this study. In attempt to create a larger input for the network, synthetic characters based on the characters in T-H-E Datasets are generated. Deep Convolutional Generative Adversarial Networks (DCGANs) are adopted to create synthetic data for augmenting the existing dataset. Additionally, a segmentation-free handwriting recognizer is proposed as the second recognizer. The latest version of YOLO network for object detection, namely YOLOv5 is applied to the system. An object detection algorithm takes a large image containing one or multiple objects as an input and predicts the location and class label of the objects. Based on this assumption, the handwritten characters are systematically placed onto a 416×416-pixel image thus creating a suitable input to YOLOv5 network. The results indicate that a segmentation-based recognition is ideal for the recognition of non-cursive

single language handwritten texts. However, object recognition-based system outperforms the segmentation-based method in both single language and multilingual handwritten text recognition for cursive and non-cursive characters.

Keywords: Offline handwriting recognition, Deep Learning, CNN, YOLOv5, DCGANs, Deep generative models, Public dataset, Handwritten character dataset, Multilingual dataset, OCR, Segmentation free, Segmentation-based, Object Detection, Turkish, Hungarian.

ÖZET

DERİN ÖĞRENME TABANLI ÇEVİRİMDIŞI ETKİLEŞİMSİZ EL YAZILI KARAKTER TANIMA SİSTEMLERİ İLE ÇOK DİLLİ EL YAZISI KARAKTER VERİ SETİ

Gaye EDİBOĞLU BARTOS

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Bilim Dalı

Eskişehir Teknik Üniversitesi, Lisansüstü Eğitim Enstitüsü, Ocak 2021

Danışman: Prof. Dr. Yaşar HOŞCAN

İkinci Danışman: Doç. Dr. Éva NAGYNÉ HAJNAL

Yıllardır süregelen araştırmalara rağmen çevirimdışı (etkileşimsiz) el yazısı tanıma hala çözümlenememiş bir araşırma problemidir. Derin öğrenme konusundaki gelişmeler görüntü işleme konusunda çevirimdışı el yazısı tanımadayı da kapsayacak şekilde büyük gelişmeler sağlamıştır. Derin öğrenme yaklaşımları, kabiliyetlerini tam anlamı ile sergileyebilmek için büyük bir veri setine ihityaç duyarlar. Özellikle bazı dillerde yazılmış, genel kullanıma açık elyazısı veri setleri sayıca fazla değildir. Bu eksiklikten yola çıkarak bu çalışmada Türkçe ve Macarca el yazısı karakterlerini de içeren genel kullanıma açık çok dilli bir el yazısı karakter veri seti oluşturulmuştur.

Bu çalışmada, oluşturulan veri setinin yanısıra iki tane çevirimdışı elyazısı tanıma sistemi geliştirilmiştir. Birinci sistem ayrıştırma temelli bir sistem olup, yine bu çalışmada geliştirilen bir derin öğrenme mimarisi ile çalışmaktadır. Bu mimariye giriş setini olarak, çalışma için toplanan veri setindeki karakterler üzerinden Derin Evrişimli Çekişmeli Üretici Ağlar (Deep Convolutional Generative Adversarial Networks) kullanılarak sentetik veriler yaratılmıştır. Önerilen ikinci el yazısı tanıma sistemi ise bir nesne algılama algorıtması olarak yeni ortaya sürülen YOLOv5 algorıtması temelli ayrıştırmasız bir elyazısı tanıma sistemidir. Nesne algılama algoritmaları genelde büyük resimler üzerinde bulunan bir ya da birçok nesnenin yerini ve de sınıfını algılarlar. Çok dilli veri setindeki karakterleri de 416×416 piksel bir alana sistematik olarak yerleştirerek, YOLOv5 e uygun bir giriş oluşturulmuştur. Sonuçlara bakıldığında ayrıştırma temelli sistem, tek dilde ayrık yazılmış el yazısı tanımada ideal sonuçlar vermektedir. Fakat,

nesne tanıma tabanlı sistemi, hem tek dilli hem de çok dilli yazı tanımada, el yazısı ve ayrık yazılmış metinlerde ayrıştıma temelli yaklaşımdan daha iyi sonuçlar vermektedir.

Anahtar Sözcükler: Etkileşimsiz el yazısı tanıma,Çevirimdışı el yazısı tanıma, Derin öğrenme, CNN, Optik karakter tanıma, Ayrıştırmalı, Ayrıştırmasız, DCGANs, Derin üretken model, YOLOv5, Genel kullanıma açık veri seti, El yazısı karakter veri seti, Çok dilli veri seti, Nesne algılama, Türkçe, Macarca.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor Prof. Dr. Yaşar HOŞCAN and co-supervisor Assoc. Prof. Dr. NAGYNÉ HAJNAL Éva for their guidance, valuable experience and support which led me complete this work.

I am also very grateful to the thesis committee members Prof. Dr. Serkan GÜNAL and Asst. Prof. Dr. Mehmet KOÇ for their extensive knowledge, generous contribution and constructive suggestions for completion of this assignment. I would like to extend my sincere thanks to the jury members Prof. Dr. Cihan KALELİ, Assoc Prof. Dr. Alper Kürşat UYSAL, and Asst. Prof. Dr. Ahmet ARSLAN for their valuable contributions and practical advice at the thesis defence.

I owe my deepest gratitude to my family. Firstly, to my dear mother Aysel OFLAZOĞLU who always inspires me and my sister Dr. Ezgi EDİBOĞLU for their unwavering support, encouragement, and patience. Subsequently, to my husband Péter BARTOS for his understanding, support, and love during this long and painful process. Without him standing by me, this work would not have been accomplished. Additionally, to my sons Imre and Endre for being such good kids and always cheering me up. Finally, to all my family members who have always been there for me.

Additionally, I wish to acknowledge the help provided by the Scientific and Technological Research Council of Turkey (TUBITAK) for their partial financial support throughout this dissertation with grant TUBITAK 2211-A.

Finally, I would like to thank my friends especially Dr. Serel AKYOL, Dr. Zühal KURT and András KAUER for all their help and support.

Gaye EDİBOĞLU BARTOS

19/01/2021

TEŞEKKÜR

Öncelikle bana yol gösteren danışmanım Prof. Dr. Yaşar HOŞCAN'a ve eş danışmanım Doç. Dr. NAGYNÉ HAJNAL Éva'ya beni bu zorlu süreçte her zaman destekledikleri, bilgi ve deneyimlerini her daim benimle paylaştıkları, ve bana inandıkları için çok teşekkür ederim.

Ayrıca tez izleme komitesi üyeleri Prof. Dr. Serkan GÜNAL ve Dr. Öğr. Üyesi Mehmet KOÇ'a cömert katkıları ve bu tezin tamamlanmasına yönelik yapıcı önerileri için teşekkürü bir borç bilirim. Bunun yanı sıra, tez savunmamda önemli görüş ve önerileri ile tezime katkı sunan jüri üyeleri Prof. Dr. Cihan KALELİ, Doç. Dr. Alper Kürşat UYSAL ve Dr. Öğr. Üyesi Ahmet ARSLAN'a teşekkür ederim.

En derin minnettarlığımı aileme borçluyum. Öncelikle her zaman bana ilham veren sevgili annem Aysel OFLAZOĞLU'na ve kız kardeşim Dr. Ezgi EDİBOĞLU'na sonsuz destekleri, beni her şartta ve her zaman yüreklendirmeleri ve bitmek bilmeyen sabırları için minnettarım. Ardından, bu uzun ve sancılı süreçte anlayışı, desteği ve sevgisi için kocam Péter BARTOS'a ve beni her zaman neşelendirdikleri için oğullarım İmre ve Endre'ye ve son olarak her daim yanımda olan tüm aile üyelerime çok teşekkür ederim. Onların desteği olmasaydı bu çalışmayı tamamlamam mümkün olmazdı.

Doktora tez dönemim süresince 2211-A Genel Yurt İçi Doktora Burs Programı bursiyeri olmama fırsat vererek beni maddi yönden destekleyen Türkiye Bilimsel ve Teknolojik Araştırma Kurumu-TÜBİTAK'a teşekkürü borç bilirim.

Son olarak, başta Dr. Serel AKYOL, Dr. Zühal KURT ve András KAUER olmak üzere yardım ve destekleri için tüm arkadaşlarıma teşekkür ederim.

Gaye EDİBOĞLU BARTOS

STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES

I hereby truthfully declare that this thesis is an original work prepared by me; that I have behaved in accordance with the scientific ethical principles and rules throughout the stages of preparation, data collection, analysis and presentation of my work; that I have cited the sources of all the data and information that could be obtained within the scope of this study, and included these sources in the references section; and that this study has been scanned for plagiarism with "scientific plagiarism detection program" used by Eskişehir Technical University, and that "it does not have any plagiarism" whatsoever. I also declare that, if a case contrary to my declaration is detected in my work at any time, I hereby express my consent to all the ethical and legal consequences that are involved.

Gaye EDİBOĞLU BARTOS

CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

GLOSSARY OF SYMBOLS AND ABBREVIATIONS

OCR	:	Optical Character Recognition
HWR	:	Handwriting Recognition
BSM	:	Bozinovic- Shriari Method
HMM	:	Hidden Markov Model
SVM	:	Support Vector Machines
RNN	:	Recurrent Neural Network
CRT	:	Civil Registration and Nationality
HT	:	Hough Transform
WT	:	Watershed Transform
RST	:	Rough Sets Theory
BN	:	Bayesian Networks
UC	:	Upper-case
LC	:	Lower-case
MNIST	:	Modified-NIST
C-Cube	:	Cursive Character Challenge
T-H-E	:	Turkish- Hungarian- English
DL	:	Deep Learning
FCL	:	Fully Connected Layer
NLP	:	Natural Language Processing
KLT	:	Karhunen-Loéve Transform
ART	:	Angular Radial Transform
SDNLL	:	Size Dependent Negative Log-Likelihood
ANN	:	Artificial Neural Network
LSTM	:	Long Short-Term Memory
GPU	:	Graphics Processing Unit
TPU	:	Tensor Processing Unit
DBN	:	Deep Belief Networks
CNN	:	Convolutional Neural Network
ReLU	:	Rectified Linear Unit

RBM	:	Restricted Boltzmann Machines
DBM	:	Deep Boltzmann Machines
DBN	:	Deep Belief Networks
DEA	:	Deep Auto Encoders
VAE	:	Variational Autoencoders
MLP	:	Multilayer Perceptron
MDLSTM	:	Multi-Dimensional Long Short-Term Memory
BLSTM	:	Bidirectional Long Short-Term Memory
GANs	:	Generative Adversarial Networks
DCGANs	:	Deep Convolutional Generative Adversarial Networks
DPI	:	Dots Per Inch
R-CNN	:	Regions with CNN Features/ Region Based CNN
YOLO	:	You Look Only Once
CSPNet	:	Cross Stage Partial Networks
SPP	:	SpineNet
PaNet	:	Path aggregation network
ROI	:	Region of Interest
FRCNN	:	Faster CNN
RPN	:	Region Proposal Network
RESNET	:	Residual Neural Network
DenseNet	:	Densely Connected Networks
FPN	:	Feature Pyramid Networks
VGG	:	Very Deep Convolutional Networks
IS	:	Inception Score
FID	:	Frechet Inception Distance
MMD	:	Maximum Mean Discrepancy
NN	:	Nearest Neighbour
MSER	:	Maximally Stable Extremal Regions
GLD	:	Generalized Levenshtein Distance
WER	:	Word Error Rate
IoU	:	Intersection Over Union

1. INTRODUCTION

1.1. Chapter Overview

The definition of Optical Character recognition (OCR), offline Handwriting Recognition (HWR) and Deep learning are provided in this chapter. It is followed by the challenges in offline handwriting and contributions of the thesis. Finally, an overview of the structure of the thesis is provided.

1.1. Definition of Terms

Optical Character Recognition (OCR) is conversion of scanned images of machine printed or handwritten text, numerals, letters, and symbols into a computer processable format such as ASCII without any human intervention. There are two types of OCR in terms of the input namely, online, and offline recognition. In online recognition, the characters are recognised as they are drawn. Furthermore, the order of strokes are available and successive points are represented as a function of time [1][2]. On the other hand, offline recognition is performed after the writing or printing has been completed. In other words, its input is an image or a scanned document [3]. As mentioned above, input to an OCR system could refer to handwritings or machine printed documents. Handwriting recognition (HWR), on the other hand only refers to recognition of offline or online handwritten images on character or script level. Therefore, HWR is more challenging compare to the recognition of typed characters since there is no standard form for individuals' handwritings.

An OCR system consists of several components. Fig. 1 shows the elements in a typical OCR system. As can be seen from the Fig. 1, firstly the document is scanned through an optical scanner. Secondly the crucial pre-processing phase is applied. Pre-processing is critical for an OCR system since the outcomes of this step are going to be the input of the recogniser in the next step. Generally, in the pre-processing phase binarization, noise removal, normalization, segmentation, and feature extraction steps are performed. Finally, in the classification step, the recognition is performed. In addition to those steps, an extra post-processing phase could be adopted in which verification is performed in order to improve the performance.

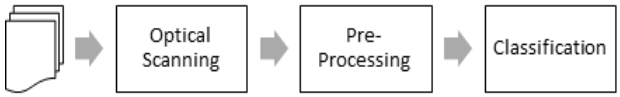


Figure 1 *Components of a typical OCR system*

The way people learn things is based on experience. The more the experience, the better the learning gets. Deep learning (DL) is a type of machine learning technique that uses experience based learning like humans do[4]. They learn patterns, features on large amount of data using a multi-layered neural network. The term "deep" derives from the depth and the complexity of hidden layers used in the network. Traditional neural networks usually have 2-3 hidden layers whereas deep networks generally consist of more complex and numerous layers as can be seen in the Fig. 2. The hidden layers extract key features using the output of the previous layer. The earlier features are usually more primitive like edges however having more hidden layers enables it to extract more complex and abstract representation of features[5]. Additionally, deep learning algorithms require a large set of data to provide desirable performance. Therefore, carrying out supervised or unsupervised learning tasks with deep networks generally require a big computational power.

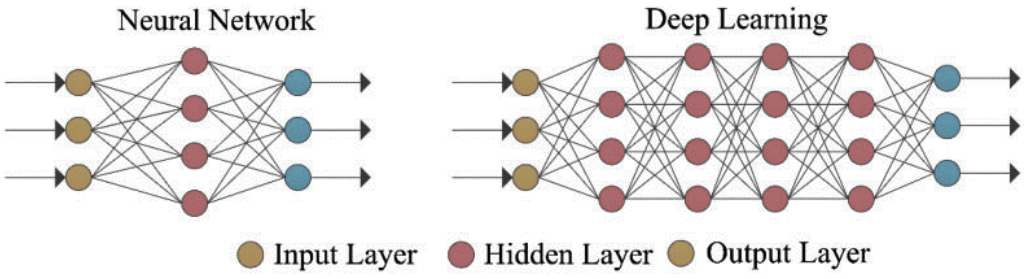


Figure 2 *Neural Network and Deep Learning Diagram[5]*

Unlike traditional machine learning algorithms, deep learning does not require a separate feature extraction step. Instead, features are extracted within the network. In terms of OCR, an input to a deep learning algorithm is the raw image not a set of features as in traditional machine learning algorithms. Results of deep learning provided a tremendous improvement in processing images, video, speech, and audio as well as in healthcare and self-driving cars.

1.2. Challenges in Offline Handwriting Recognition

Offline handwriting recognition is a task which is carried out after the handwriting has been performed. The time between the handwriting and recognition can be very short or it can be tens or hundreds of years as in the case of historical documents.

Tappert defines five difficulty levels for handwritten characters (Fig. 3) which are namely boxed discrete characters, spaced discrete characters, run-on (touching or overlapping) discretely written characters, pure cursive handwriting, and mixed cursive and discrete characters[6]. Recognition of stage 1-boxed discretely written characters are considered the easiest to recognise and stage 5-mixed cursive and discrete characters are the most difficult.

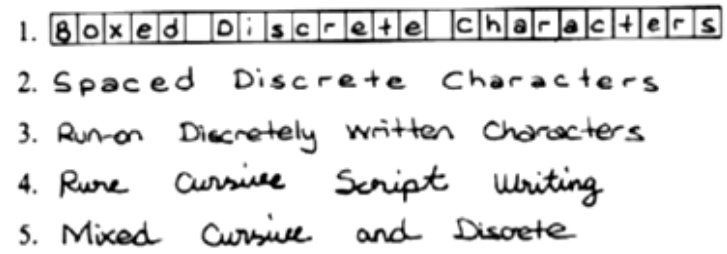


Figure 3 *Five stages of handwritten word recognition problem* [6]

In real life, handwritings can be performed in various ways depending on the writer such as writing same characters in different ways, using different styles of handwritings, quality of the ink, overlapping characters and readability. Readability is an issue that is worth mentioning since it may be challenging to read certain handwritings for human beings. The real problem with handwritings occur when it is performed cursively and/or the letters are joined-up(overlapping)[7]. For example, a cursively written letter *u* and *v* after one other can easily be mistaken as the letter *w*. Therefore, recognition of such handwritings without knowing the context of the document can be an issue.

The language of the handwriting is also an important factor determining the performance in HWR. In certain languages such as Hindi and Arabic, the characters are written in a way that is very difficult to segment. Additionally, high usage of diacritical marks also brings about a challenge in other languages. Availability of handwritten character, word and full text datasets in a language is also a defining factor for the recognition of that language. With advancements in machine learning especially deep learning algorithms, the need for large datasets has increased over the years.

Another challenge in HWR is the existence of multilingual documents. With globalisation, the number of multilingual documents is increasing and there is little study covering input. Advanced object detection-based systems started to offer good results on this domain.

The last issue to mention derives from the quality of the images. In offline handwriting recognition, a scanner or a digital camera is usually used to digitalise the document. Low resolution pictures and pictures taken from awkward angles also make the recognition task more difficult. Additionally, the condition of the original document and the quality of the ink are also issues to consider especially in the case of historical documents.

1.3. Main Contributions

Two separate deep learning based offline handwritten character recognizer systems which could possibly be applied to several languages are proposed in this study. The major contributions are put forward below:

i. A new CNN architecture for handwriting recognition
   A CNN architecture that outperforms other architectures on a multilingual handwritten character dataset was proposed.
ii. Generation of a multilingual handwritten character dataset (T-H-E Dataset)
   A large dataset consisting of Turkish, Hungarian, and English handwritten characters was generated and published.
iii. Augmentation of the dataset with synthetic characters
   Using Deep Convolutional Generative Adversarial Networks (DCGANs), synthetic characters based on the proposed dataset were generated and used for the recognizer system.
iv. Handwriting recognizer system for Turkish, Hungarian, and English
   A segmentation-based handwriting recognition system for Turkish, Hungarian, and English handwritings was created using the proposed CNN architecture.
v. Generation of a handwritten character dataset for Object Detection
   By systematically locating proposed characters in T-H-E Dataset onto a larger image (416×416) and tagging the location and label of the characters, a small dataset was proposed for training and testing Object Detection algorithms on such domain.

vi. YOLOv5 based handwritten text recognizer

Adopting a brand new and popular object detection algorithm, YOLOv5s, a segmentation free handwriting recognizer was proposed.

vii. Publications

During the thesis studies, five conference proceedings and one journal article based on the research conducted on the thesis were published. The details of the publications can be found in the Appendix section.

1.4. Organization of the Thesis

This study consists of eight chapters. The first chapter includes the definition of terms, main challenges in offline HWR and contributions of the thesis. The second chapter presents the concept of handwriting and alphabets including a brief explanation of Turkish, Hungarian, and English alphabets. It is followed by an overview of a general HWR system. The steps in traditional machine learning based HWR and DL approach are explained separately in the chapter. The experiments carried out on different stages of HWR and their results are put forward in the third chapter. Additionally, the generation process of a CNN architecture and evaluation of it is also included in this chapter.

The proposed multilingual dataset, T-H-E Dataset, is explained in the fourth chapter. In the fifth chapter, using DCGANs, the generation of synthetic characters based on the T-H-E Dataset is represented. A segmentation based multilingual handwriting recognition system based on the proposed CNN model is explained in the sixth chapter. In the seventh chapter, a segmentation free handwriting recognition system based on YOLOv5 is showed. The eight and the last chapter includes the conclusion and future works.

2. LITERATURE REVIEW

2.1. Chapter Overview

The concept of handwriting and alphabets are explained in this chapter focusing on the properties of three alphabets namely Turkish, Hungarian, and English. It is followed with a detailed overview of offline handwriting recognition system. Additionally, a detailed explanation of deep learning and popular DL algorithms are also given in this chapter.

2.1. The Concept of Handwriting and Alphabets

Alphabet is defined as a group of symbols and characters representing the phonemic structure of a language[8]. The earliest findings of an alphabet dates back to 1500 BC used by Phoenicians. In the modern world, the commonly used alphabets include Arabic, Chinese, Cyrillic, Greek, Hebrew, and Latin alphabets. Sample words from those alphabets are shown in the Table 1 below.

Table 1 *Representation of Different Alphabets*

Alphabet	***The representation of the sentence "how are you"***
Arabic	كيف حالك
Chinese	你好嗎
Cyrillic	Как поживаешь
Greek	Πώς είσαι
Hebrew	מה שלומך

In this thesis, mainly three alphabets are going to be mentioned namely the English, Hungarian and Turkish alphabet. All three alphabets are examples of Latin script alphabets meaning that they all use letters of Latin script. The characteristics of the above-mentioned alphabets and handwritings are explained below.

2.1.1. Properties of English Alphabet

English alphabet consists of 26 letters and every letter has its lower- and upper-case representation which can be seen in the Fig. 4 below.

Figure 4 *English Alphabet*

The 26 letters that comprise the English alphabet are used in the ISO basic Latin Alphabet which is a Latin script base alphabet used as an umbrella term in grouping Latin script base alphabets.

The concept of English handwritten character recognition is a very well-studied area, and recognition rates are satisfactory. Therefore, in this study the recognition of these letters is going to form the base for comparison using English benchmark databases.

2.1.2. Properties of Turkish Alphabet and Handwriting

Turkish Language consists of 29 letters as shown in the Fig. 5 below. 23 letters in this alphabet are the same as ISO basic Latin script and the remaining 6 letters are additions with diacritical marks which are symbols below or above a letter. The additional characters of the language generate a challenge for recognition purpose such as removal of the diacritical marks at the noise removal phase.

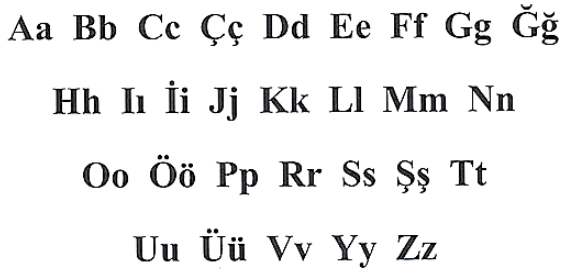


Figure 5 *Turkish Alphabet*

In Turkey, there is not a standard for the type of handwriting however using cursive handwriting is encouraged at primary school. Therefore, some people use cursive handwritings and others choose not to. Based on that, a successful Turkish handwriting recognition system should be able to perform well on all 5 categories of handwriting defined by Tappert[6].

2.1.2.1. Previous Studies on Turkish Handwriting Recognition

The first studies on Turkish handwriting recognition dates to 2003. The first paper in 2003 was on handwritten uppercase letters by Çapar et al..The data set for the study was gathered from the students in Istanbul Technical University which consisted of about 20000 uppercase and 7000 digit samples. For the feature selection many methods were used namely Karhunen-Loéve Transform (KLT), Zernike Moments, Angular Radial Transform (ART) and Geometric Features. As for classification Artificial Neural Networks, K-Nearest Neighbour, Nearest Mean, Bayes, Parzen and Size Dependent Negative Log-Likelihood (SDNLL) methods were employed. The results offered that KLT and ART classified by SDNLL gives the best result by 93,3 % for Turkish characters in their experiments [9]. In 2003, Yanıkoğlu and Kholmatov presented a study on Turkish handwriting recognition based on a lexicon-based recognition approach and a Turkish prefix recognizer. The performance of the system ranged from 53% to 57% depending on the writing type for the lexicon-based approach and between 38% to 41% for the parser-based approach. The performance was not surprising for the authors since the OCR engine was not trained with Turkish characters [10].

In 2004, Vural et al. proposed the online handwriting recognition for Turkish. In order to create the data set, they used a tablet PC interface. Hidden Markov Models (HMM) were adopted to train letter and word models. The proposed model performed 94% in recognizing handwritten words from a 1000-word lexicon [11]. This study is different from the others since it focuses on online handwriting recognition for Turkish where all the others focus on offline recognition. In 2006, Şekerci and Kandemir put forward another study for Turkish handwriting recognition. They collected the data from 172 people with 29 lowercase and 29 uppercase letters per person. Total of 9976 print characters (nun cursive) were collected. K-n neighbour method was used for classification and a dictionary was used for verification. The results showed that the system performed 100% on print letters [12]. Later in 2007, Şekerci wrote his master thesis on recognising connected and slant handwritten Turkish texts. Dataset was collected from 172 persons. Each person provided 29 uppercase, 29 lowercase characters and 10 digits. In total, 11696-character examples were gathered. In order to detect satire horizontal histogram graphic was adopted for satire finding and vertical histogram graphic was used for character finding. Later for character recognition, correlation algorithm was employed,

and K-n neighbour method was used for classification of the letters. The experiments resulted in 93% performance of digits, 90,4% for uppercase letters and 91,2% for lowercase letters [13]. Aydemir et al. developed two different Ottoman and Turkish handwritten recognition systems using Hidden Markov Model (HMM) and Recurrent Neural Network (RNN) in 2014. Many different data sets were applied namely IFN/ENIT dataset, which is created for Arabic language, is used because of the similarity between Ottoman and Arabic, IAM dataset is used which consists of Latin characters. Additionally, Civil Registration and Nationality (CRN) dataset was also adopted. Comparing the results of both system, the system using the RNN provided %8 higher accuracy rate then system using HMM [14].

2.1.3. Properties of Hungarian Alphabet and Handwriting

Hungarian Language consists of 44 letters (Fig. 6). All 26 letters from ISO basic Latin script are included in Hungarian alphabet and additionally 9 others have diacritical marks. The remaining 8 letters are examples of diagraphs such as “sz”, “ty” and 1 trigraph “dzs”. These characters of the language generate a challenge for recognition purpose such as removal of the diacritical marks at the noise removal phase. Another challenge in recognising Hungarian handwriting is that in Hungary there is a tradition of using cursive scripts. Cursive character of the handwriting brings about the challenge to the segmentation phase. Therefore, Hungarian Handwriting recognition is challenging compare to other Latin handwritten scripts. In addition to the challenges brought about with the way of writing, the lack of the studies conducted for Hungarian character recognition is another one. By the time this work was carried out, no studies conducted in English language in the field could be found apart from the ones carried out by the research group of the author of the thesis.

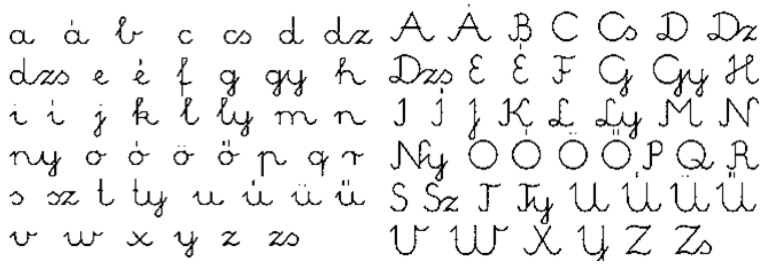


Figure 6 *Hungarian Alphabet*

2.2. Offline Handwritten Character Recognition

OCR refers to the recognition of characters which can be machine printed, or handwritten. Additionally, handwriting recognition also refers to two types of handwritings namely online and offline. Online handwriting recognition is carried out during the writing process generally on a tablet or using a special pen[15]. The recognition is performed after the writing process has been completed in offline handwriting recognition[16].

In traditional machine learning techniques, a handwritten document is firstly digitalized and then pre-processed to make it ready for feature extraction. Then, feature extraction is applied to get the distinctive features of the image. Finally, the image is classified based on the feature vector provided by feature extraction step. There can be an additional step in between feature extraction and classification called, feature selection which is applied to determine the most distinct features and eliminate the ones that do not have such value. In terms of classification, Support Vector Machines (SVM), Multilayer Perceptron (MLP) and Hidden Markov Models (HMM) are amongst popular classifiers[17]–[19].

In terms of deep learning algorithms, the digitalized document is process directly without a separate feature extraction phase. The deep network, extracts features and performs classification.

In this section, components of a traditional machine learning based, and deep learning based offline handwriting recognition systems are explained separately alongside with earlier works on offline handwriting recognition.

2.2.1. Optical Scanning

The first step of an offline OCR system is digitalizing of the physical documents. Optical scanners and digital cameras are commonly used to perform the conversion of the documents into a digital form. The quality of the scanning or the captured image plays a key role in the performance of the recognition since the result of this phase makes up the input for the other steps. Depending on the quality of the scanner, scanning process usually generates a more quality representation of the image for an OCR than capturing the image with a digital camera.

2.2.2. Pre-processing

Handwritten documents can be presented in several ways for example a historical handwritten paper can be found on a very large paper with a special ink and an exam paper of a student is usually written on an A4 sized white paper written with a pen. In order to create a standard way to represent different inputs, pre-processing techniques are adopted. In other words, raw data better be pre-processed to make it more suitable and, in some cases, applicable to an HWR system. This process establishes the path between handwriting and recognition. The pre-processing for offline handwriting recognition consists of many steps which usually include binarization, noise removal, normalisation, skew correction, slant removal and segmentation. According to different needs, different steps may be included or excluded.

2.2.3. Binarization (Thresholding)

Image binarization is used in order to convert a coloured or grayscale handwritten image into a binary image in which the image is only represented with black and white colours (black pixel value=0 and white pixel value=1). This process helps to reduce the size of the image and therefore speed the image processing up. In order to determine the colour of the pixel in binarization, a threshold value is used. The pixel values greater than the threshold is set as a white pixel and the rest are set as black. Two types of thresholding methods namely global and local (adaptive) thresholds are used in OCR systems [20]. In global thresholding, one threshold is used for the entire document. It is ideal for images with a bimodal histogram, minimal noise and not having a small object area. There are several techniques in the literature for picking the optimal global threshold for example Otsu's thresholding technique is amongst the most popular ones [21]. On the other hand, local thresholding uses different threshold values for each pixel according to the local area information [22].

2.2.4. Skew Correction and Slant Removal

When it comes to handwriting recognition, it is crucial to take into different handwriting styles into consideration. Performing handwritten texts with various skew and slant are amongst the most common ones. Skew correction is used when the document needs to be aligned with the coordinate system as can be seen in the Fig. 2. Inaccuracies in the scanning process and the style of handwriting may result in tilted or curved lines

within the image. Most popular approaches for skew correction includes correlation, projection and Hough transform [23], [24].

Slant refers to the tilt of the handwriting in the vertical axis as can be seen in the Fig. 7 Slant of handwritten texts varies from person to person (no slant, right slant, or left slant). Several methods are proposed for slant removal purpose. Examples of popular techniques are projection, calculation of the average angle of near-vertical elements, projection and Bozinovic- Shriari Method (BSM) [25], [26].

a) b)

Figure 7 *a) Skew and b) Slant*

2.2.5. Noise Removal

The quality of the document is improved by using noise reduction methods such as filtering and morphological operations (erosion, dilation etc.). Normalization also provides a tremendous reduction in data size by thinning and extracts the shape information of the characters [24], [27].

Most popular techniques for noise removal include filtering and morphological operations. Filters are usually aimed at smoothing, sharpening, thresholding, removing background and adjusting contrast [28]. Commonly used filters for noise removal include linear filtering, median filter, adapting filtering and averaging filter. As for the morphological operations, they replace the convolution operation by the logical operations. There are numerous morphological operations designed for noise removal such as thickening and thinning the characters; connecting the broken strokes and decomposing the connected strokes[25], [28].

2.2.6. Segmentation

Segmentation phase is used to determine the regions of the texts on the document which are going to be recognised in the following steps. It is necessary to identify the region of the handwritings on a document from figures and graphics[29]. This is a crucial step for character recognition considering that the success of the recognition highly depends on a successfully segmented image[30], [31]. Depending on the recognition level (character level or script level) and the algorithm used for the recognition, segmentation

can include various steps such as line segmentation, word segmentation, character segmentation. Correlatively, based on the algorithm, the number of segmentation types applied might also differ. For instance, in full text recognition line segmentation is generally adopted and in another case in which words are segmented using bounding boxes, line segmentation step can be skipped.

In terms of handwriting recognition, the document is usually divided into lines, then into words. Having the words segmented, there are two approaches to the segmentation namely holistic approaches and analytical approaches as can be seen in the Fig. 8 Holistic approaches (segmentation free) refer to the recognition of the words as a whole using a lexicon without further segmenting them. Analytical approaches can be categorised into two sections namely implicit and explicit segmentation.

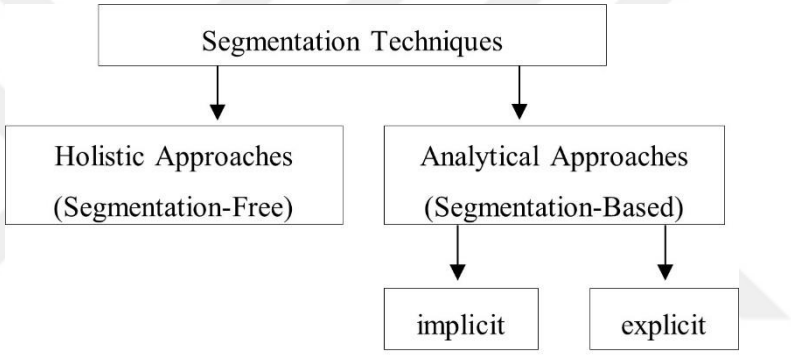


Figure 8 *Categorization of Segmentation Techniques*

Implicit Segmentation: Segmentation of the characters and recognition are carried out at the same time in this approach[31], [32].

Explicit Segmentation: This approach is the classical approach in which a word is divided into characters and then individual characters are recognised. Therefore, the performance of the explicit segmentation plays a key role in recognition[33].

2.2.7. Size Normalization

The image size of the input plays an important role in the performance of the recognition. The need for size normalisation derives from several reasons such as variation of handwritten characters and paper size. Extracting key features can be applied more accurately if there is a fixed size in the input image even in the case of the advanced deep learning-based systems.

2.3. Components of Traditional Machine Learning Based Handwriting Recognition

Different from DL, in traditional machine learning the features should be harvested manually using different techniques. Subsequently, a classifier is used to perform the recognition. In this section, popular feature extraction, feature selection and classification methods are explained.

2.3.1. Feature Extraction

In the feature extraction phase, significant features of a character are extracted. The result of the classification is directly affected by the features extracted since the feature vectors are going to be the input for the classifier. Therefore, it is crucial to extract the key features. Depending on the classifier, the features can be extracted manually and fed into the classifier such as Hidden Markov Model (HMM) or Support Vector Machines (SVM) or as in deep learning applications, the features can be extracted automatically while training the network.

Several feature extraction methods have been proposed for character recognition purpose in the literature. It is possible to group those into three categories namely distribution of points, structural analysis and transformations and series expansions.

2.3.1.1. Distribution of Points:

In this category, features are extracted based on the statistical distribution of points. These features are usually tolerant to distortions and style variations[34]. Key feature extraction methods in this category are explained below.

Projection Profiles:

Profiles refer to the distance from the border of the image until the next white pixel. As can be seen in the Fig. 4 below, different profiles can be used as feature vectors such as left, right, top, and bottom profiles.

Figure 9 *Right, left, top and bottom profiles of a character*

As can be seen in the Fig. 9, it is crucial to apply normalization to the images before extracting features otherwise it would be difficult to extract meaningful features from the images.

Extremas of the character image:

It returns the x and y coordinates of the 8 extremas of the image namely top-left, top-right, right-top, right-bottom, bottom-right, bottom-left, left-bottom and left-top as can be seen in Fig. 10.

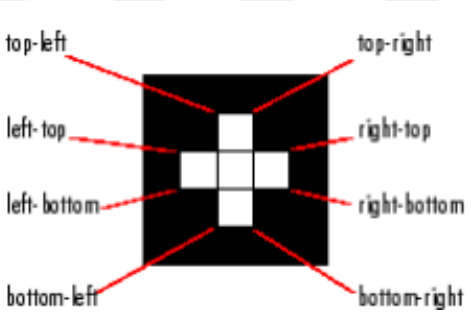


Figure 10 *Extremas*

Similarly to the previous method, it can be seen that normalization of the images before feature extraction is essential.

Centre of gravity:

It refers to the (x, y) values of the centre of gravity of the character image. In addition to those, the distance between the bottom of the character and the y coordinate of the centre of gravity and the distance from the left end of the character and x coordinate of the centre of gravity are also used as feature vectors.

Density:

The colour density of the character image is used as the feature vector.

Area:

It represents the actual number of pixels in the region, returned as a scalar.

The proportion of width and length:

It is the result of dividing the width of the character into the length of the character.

Number of regional minimas and maximas:

After applying horizontal and vertical projections, the number of regional maximas and regional minimas for both vertical and horizontal projection are used as features.

2.3.1.2. Structural analysis

This type of features represents the geometric and topological structures of a character. The most common types include endpoints, loops and strokes[29][35]. It is worth mentioning that these types of features are highly affected by any noise in the input data. As can be seen in the Table 2, any noise in the character image would cause a change in the feature vector thus a noise free input is crucial for the recognition.

Table 2 *An example feature-set of noise free character image and a noisy character image*

Character image	#endpoints	#connected components	#isolated small areas
	4	2	2
	5	3	3

No of endpoints:

It represents the number of pixels having only 1 connected neighbour in an 8 connected image.

No of branch points:

It represents the number of pixels having at least 3 neighbours that are 1s in an 8 connected image. An example of endpoint, branchpoint and loop is given in the Fig. 11 below.

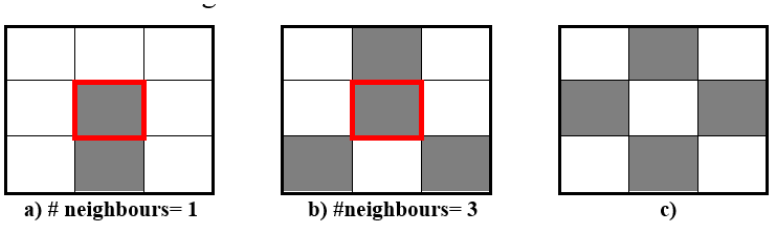


Figure 11 *a) endpoint, b) branch point, c) loop*

Euler number:

The Euler number represents the total number of objects in the image minus the total number of holes in those objects.

Number of loops:

The vector represents the number of holes in the image.

Number of small components:

The feature vector represents the number of isolated areas with the area smaller than 7 in the character image.

Sum of the area of small components:

It represents the sum of all small components with less than 7-pixel area (the number depends on the image size).

No of connected components:

It represents the number of connected components in the image.

2.3.1.3. Transformations and series expansions:

Unlike the other two type of techniques, this type of features is not affected by the deformations such as size and rotation since the features are represented as a continuous signal[29]. Several transformations and expansions are used for this purpose and the most popular ones are explained below.

Fourier Transforms:

The Fourier transform relates a signal sampled in time or space to the same signal sampled in frequency. Fourier analysis breaks down a signal into its frequency components. The reason behind is that a simpler signal representation as trigonometric

sine and cosine is more favourable than the original complex signal[36]. In handwriting recognition, it is generally used to calculate the contour of a character. In more detail, the first *n* coefficients of the transform can represent the contour and those can also be taken as the n-dimensional feature vector that represents the character. The more the number of coefficients is, the more details are represented[37].

Hough Transform:

In image processing applications, Hough transform is widely adopted for line and curve detection purposes[38]. Similarly, in handwriting recognition, it is used to extract base lines, curves, and circles in characters. The main principle of it is to find out a mapping between an image space and a parameter space.

Gabor Transform:

In Gabor transform is a window defined by a Gaussian function is used instead of the discrete size window in windowed Fourier Transform[39].

Wavelets:

The logic behind the Wavelets is very similar to Fourier transformation, the difference being the extra representations of the signal at different levels of resolution[40].

2.3.2. Feature Selection

The most distinctive features for classification are selected in the feature selection step. This is done by eliminating the irrelevant or noisy features from the feature set thus decreasing the number of features to be processed in the next step. It should be noted that, no relevant feature that would negatively affect the classification performance should be deleted in this step. The main goal in this step is reducing overfitting and training time and increasing the classification performance. Therefore, it can be beneficial to adopt this step after feature selection.

There are two types of feature selection methods namely wrapper and filter method. Filter method is used independently from the classification method and can be selected once and applied to different classifiers. However, wrapper method finds the distinctive features for a specific classifier. Therefore, wrapper model usually generates a more relative feature subset but they are more expensive in terms of time and computational

complexity than filter methods especially on large feature sets[41]. Additionally, wrapper method requires recalculation for each classifier unlike the filter method. Popular wrapper methods include Sequential Forward Selection and Genetic Algorithms. Correlation Based Feature Selection and Information Gain are amongst well-accepted filter methods.

2.3.3. Classification

Performance of a recognition system highly depends upon the classifier. The output of earlier stages serves the recognition stage therefore it is key to have a suitable classifier which can work efficiently on the provided feature set. Popular classifiers are explained below.

Multilayer Perceptron

The multilayer perceptron is a feedforward artificial neural network consisting of at least three layers namely, input, output and hidden layer which can be seen in the Fig. 12 below [42]. The input layer contains the inputs features of the network. The first hidden layer receives the weighted inputs from the input layer and sends data from the previous layer to the next one. Finally, the output layer presents the classification.

An MLP consisting of one hidden layer is a shallow network however having more hidden layers, it is one of the most traditional types of deep learning architectures having every element of a previous layer connected to every element of the next layer. In the next section, a detailed explanation of deep learning is going to be provided. In offline HWR domain, application of the multilayer perceptron (MLP) ensures high recognition accuracy when performing a robust training[19], [42]–[47].

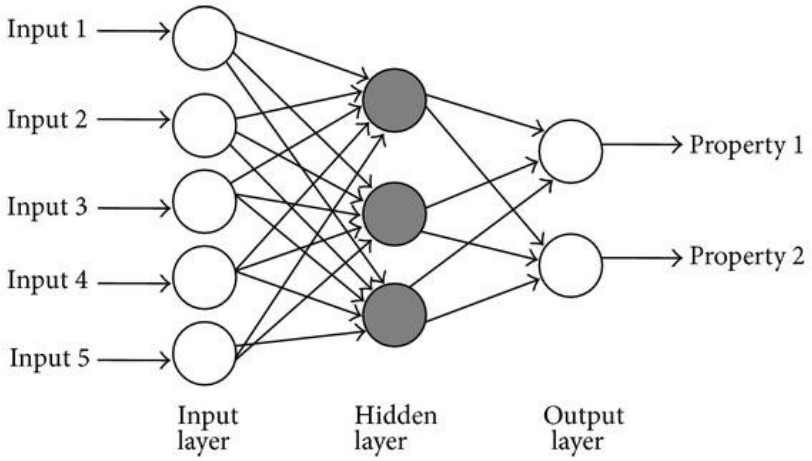


Figure 12 *Basic Architecture of a Multilayer Perceptron Neural Network*[48]

Support Vector Machines: SVM classifier carries out the classification by mapping all the input data to a value in a higher dimensional space. The data is classified by coming up with an optimum N-dimensional hyper plane which separates data into positive and negative examples[5]. Common applications of SVMs include text categorization, face detection and image classification[49]–[52]. Additionally, SVMs have successfully been used in recognizing handwritten characters as a single classifier [53]–[55] and in hybrid systems[56]–[58].

Rough Sets Theory: Rough Sets is a mathematical tool which deals with uncertainty and vagueness [6]. The idea is based on the assumption that with every object of the universe of discourse, it is possible to associate some information. Objects characterized by the same information are indiscernible in view of the available information about them. The indiscernibility relation generated in this way forms the mathematical basis of the theory. The rough sets theory provides a technique of reasoning from imprecise data, discovering relationships in data and generating decision rules [7]. Not requiring any preliminary or additional information about data like probability distributions in statistics is rough sets theory's main strength [8]. Hybrid applications of RST in offline HWR can be found[59]–[62].

Bayesian Networks: Bayesian classifiers are the statistical classifiers based on Bayes Theorem. They are able to predict class membership probabilities such as the probability that a given tuple belongs to a particular class[9]. Bayesian networks are a model representing uncertain knowledge about a complex phenomenon and allowing real

reasoning from data. They effectively represent a domain of knowledge, as a causal graph, permitting learning the dependency relationships that can help us make decisions and manage all incomplete data[10]. Bayesian Networks works especially well on offline HWR in which the characters are segmented [63], [64].

2.4. Deep Learning

In this section a general description of deep learning is provided followed by a brief history of the field. Then, DL architectures are represented with their application fields.

2.4.1. Definition

Deep learning is subtopic of machine learning that is capable of performing both supervised and unsupervised learning using a similar structure of human brain as it is self-teaching and learns by filtering information through multiple hidden layers [65]. DL is made of multiple processing layers in order to learn representations of data with multiple levels of abstraction [4]. It is based on a hierarchically layered system in which each layer of nodes is responsible for extracting distinct features using the previous layers output. The further you go with the layers; the more advanced features can be extracted. Applications of deep learning includes speech recognition, Natural Language Processing (NLP), image and video processing [66], [67]. While it is not necessarily new, deep learning has recently seen a surge in popularity as the computational power of machines have increased in the past years thus making it possible to work on large input data and execute such calculations in relatively short time.

2.4.2. A Brief History of Deep Learning

The studies can be linked to deep learning dates back to 1943. Pitts and McCulloch put forward a computer model based on the neural networks of the human brain "threshold logic" to mimic the thought process[68][69]. Later, at the beginning of the 1960s Kelly came up with a simple continuous Back Propagation Model[70]. Then it was followed by Bellman and Dreyfus in 1962, developing a simpler version based only on the chain rule[71]. In 1965, an early deep learning algorithm was proposed by Ivanhnenko and Lapa in 1965 [72]. They used models with polynomial (complicated equations) activation functions, that were then analysed statistically. From each layer, the best statistically chosen features were then forwarded on to the next layer[73], [74].

In 1979, Fukushima developed the first convolutional neural networks that which was a hierarchical, multi-layered artificial neural network (ANN) [75]. Not much later, Linnainmaa, a master student in Finland, wrote a code for backpropagation however it was not applied until mid-80s [76].

Support vector machine (SVM) was put forward by Cortes &Vapnik in 1995 [77]. [78]. Two years later in 1997, another popular deep learning algorithm, LSTM (long short-term memory) for recurrent neural networks was developed by Hochreiter and Schmidhuber [79]. By the end of the 1990s, developments in processing power of computers and GPUs led to an increase in the adaptation of deep learning algorithms that could be comparable with the adaptation of SVM in the field. In 2006, Deep Belief Networks (DBN) was introduced by Hinton et al. Two years later in 2009, Deep Boltzman Machines (DBMs) were introduced by Salakhutdinov & Hinton [80]. In 2010, 2011 and 2012 Cireşan et al. carried out several experiments using deep learning for handwritten character recognition on MNIST and NIST databases [19], [81]–[84]. In order to overcome overfitting in the set, a dropout method was introduced by Hinton et al. in 2012 [85]. The method was widely used not only because of its good performance but also because of its simplicity of implementation [73].

Deep learning today is commonly used in almost every field of life such as advertising, military, and medicine. Some of the popular applications include self-driving vehicles, social media (for example automatic friend tagging on Facebook) and Google Translate. Especially for the past 6 years, universities and large companies organised deep learning research groups and labs that can be seen in the Table 3 below.

Table 3 *Popular Deep Learning Research Groups/ Labs*

Research Group/ Lab	Field
Google Research/ Google Deepmind	NLP, IP
Facebook Fair	NLP, CV
Twitter Cortex	NLP, IP, CV
Microsoft DLTC	NLP, CV

In addition to above mentioned companies several prestigious universities such as University of Toronto, New York University, Stanford University, and Oxford University also have DL Research Groups.

There are several factors that led deep learning to such success today. Some of the factors that had an affect this success are presented below [86]:

- Emerging of high-quality labelled datasets,
- Parallel computing with GPUs,
- Variety in the available software platforms such as Tensorflow (Google Brain), Caffe 2 (by Berkeley Vision and Learning Center), CNTK (by Microsoft Research) and Theano (by University of Montreal),
- New regularization techniques resulting in less overfitting such as dropout, batch normalization, and data-augmentation.

2.4.3. Deep Learning Algorithms

In this section commonly used deep learning architectures such as Deep Boltzmann Machines and Deep Auto Encoders and their applications are briefly presented alongside with an in-depth explanation of Convolutional Neural Networks.

Convolution Neural Networks (CNNs): A classic CNN architecture consists of an input layer, an output layer and hidden layers. An input can be a 1D signal, 2D image or 3D video. Thereafter, the input goes through a serious of hierarchical layers including convolutional layers, pooling layers in order to extract distinct features in the input. Finally, extracted features form the input layer of a Fully Connected multilayer perceptron (MLP) at the very end of the architecture for recognition. A brief CNN architecture can be seen in Fig. 13 below.

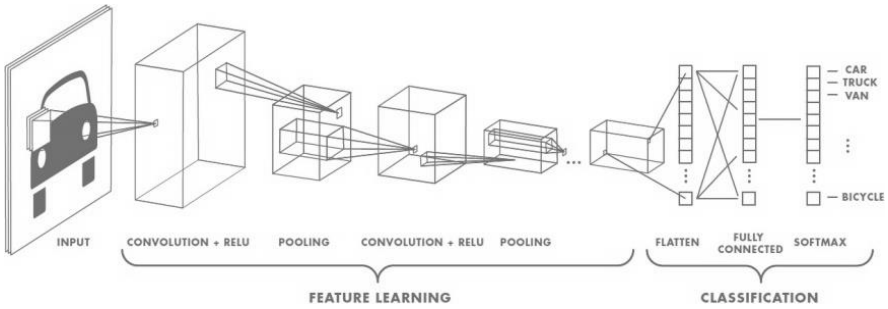


Figure 13 *Convolutional Neural Network Architecture* [87]

As can be seen from the Fig. 13 above, there can be several layers for feature learning purpose and output from one layer becomes the input for another. Based on this hierarchy, the further layers tent to produce higher level features. Three most common types of layers are namely convolutional layer, activation layer (ReLU-Rectified Linear

Unit) and pooling layer. Depending on the nature of the input, these layers can be applied in different numbers and sizes. However, it is notable that the greater the number of layers, the more difficult the calculations become.

Convolution Layer: In this layer, a convolution filter is applied to the input matrix to generate feature maps as can be seen in Fig. 14. The size of the filter is pre-determined according to the input matrix. For example, for a 32×32 matrix the first layer usually consists of a 5×5 filter. For larger sized inputs, filters that are 12×12 can also be found[88].

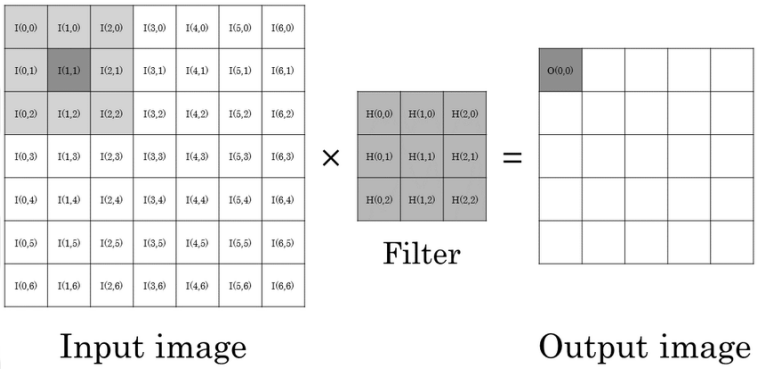


Figure 14 *Convolution Filter* [89]

Activation Layer (ReLU): ReLU operation replaces all negative pixel values in the feature map by zero [90]. Thus, it allows faster and more effective training.

Pooling Layer: Pooling layer aims at reducing the size of the feature maps for the next convolution layer generally by applying a sum, avg or max filter to the feature map as can be seen in Fig. 15. However, the reduction does not necessarily result in data loss, but eliminates the least significant data resulting in easier computation in the upcoming layers[90][91]. The operation performed by this layer is also called subsampling or downsampling.

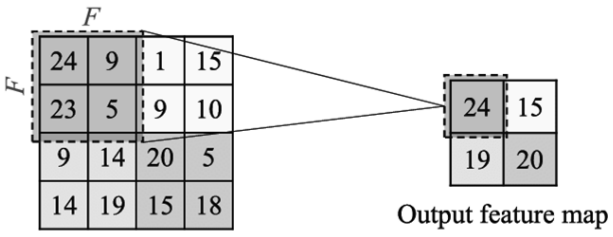


Figure 15 *A 2×2 Max Pooling Layer with Stride of 2* [92]

Applications of CNN are mostly focused in the field of Natural Language Processing (NLP) and image/ video processing.

Recurrent neural networks (RNNs):

In neural networks, all the inputs are independent from outputs. However, in RNNs the output from previous step is the input to the current step which solves issues in cases like when it is required to predict the next word of a sentence. Proposal of RNNs solved this issue with the help of a Hidden Layer [73].

This type of networks is more successful with sequential data such as speech recognition, music generation and DNA sequence analysis [93]–[95].

Long Short-Term Memory (LSTM):

An LSTM network is a type of recurrent neural networks (RNNs) [79]. An LSTM network for sequential data classification consists of a sequence input layer, an LSTM layer, a fully connected layer, a softmax layer, and a classification output layer.

LSTMs have high performance on learning, processing, and classifying sequential data. Common areas of application include sentiment analysis, NLP, speech and text recognition.

Restricted Boltzmann Machines (RBMs):

RBM are created based on BMs with some restrictions added onto it, therefore making it simpler. Its architecture consists of visible and hidden graphs that have a symmetrical connection different that the BMs by removal of the connections between hidden units, and the connections between visible units shown in Fig. 16 [66], [73], [80].

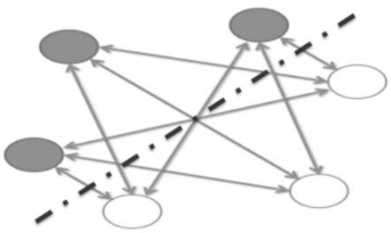


Figure 16 *Representation of Restricted Boltzman Machines* [73]

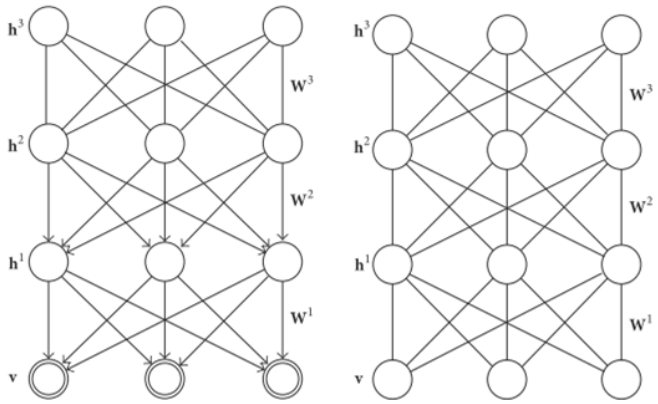
Their applications include dimensionality reduction, regression, classification, and feature learning.

Deep Boltzmann Machines (DBMs):

DBMs are formed by stacking RBMs together as can be seen in Fig. 8(b) below [80]. Similarly to the RBMs, DBMs also consists of visible and hidden graphs that have symmetrical connections. In a DBM, all connections are undirected between different layers.

Deep Belief Networks (DBNs):

Deep Belief Networks was proposed by Hinton et al. presenting that that DBMs can be trained in a stacked greedy manner [78]. Similar to the DBN architecture, every layer in DBNs is connected to the next and the previous layers. However, there is no vertical commination between nodes as can be seen in Fig. 17(a). Although DBN is referred as a stacked RBM, the architecture consists of one-layer RBM with extended layers devoted to generating patterns of data rather than putting one RBM on the top of another. Similarly, to the last layer of CNNs, the last layer for a supervised learning is usually softmax layer or clustering for unsupervised learning. DBN are vastly used in large scale image recognition and image production.



(a) (b)

Figure 17 *Representation of (a) Deep Belief Network and (b) Deep Boltzmann Machine* [66]

The difference between DBNs and DBMs can be seen in the Fig. 17 above. The top two layers of a DBN and DBM form an undirected graph however unlike undirected connections in all layers in DBM, in DBN the remaining layers form a belief network with directed, top-down connections. The undirected form of DBMs results in better learning performance on complex patterns compare to DBNs.

Deep Auto Encoders (DEAs) (Variational Autoencoders (VAEs)):

Deep Autoencoders are deep neural networks having the same number of neurons in the input and at the output layer as can be seen in Fig. 18 below.

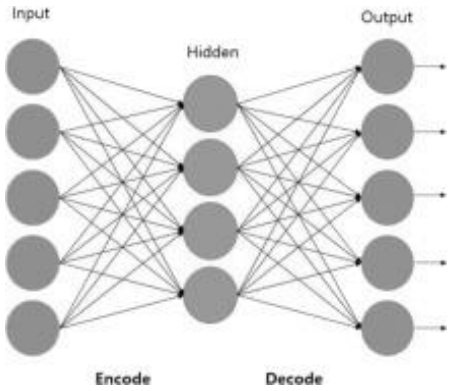


Figure 18 *Representation of Deep Autoencoders*[96]

As can be seen in the figure above, the layers of DEAs are RBMs. The encoder part of the architecture works as a dimensionality reducer by compressing the input data by making sure that significant data is not lost but the size of the data is reduced largely. The decoder part reproduces the uncompressed version of the data. Common uses of the network include NLP and image/ video processing.

3. EXPERIMENTS

3.1. Chapter Overview

In this section, experiments carried out to the evaluate above mentioned pre-processing methods are represented. In addition to that, steps for finding a suitable CNN architecture for handwritten character recognition is also included in this chapter.

3.2. Experiments on Binarization

To carry out the experiment, handwritten pages were scanned in 600 DPI and saved in .png file format. Subsequently, the images were binarized using two different global thresholding methods. Firstly, the threshold value was given manually by looking at the histogram of the image. Then, Otsu's Global Thresholding was adopted using imbinarize function provided by MATLAB 9.3 environment [21], [97].

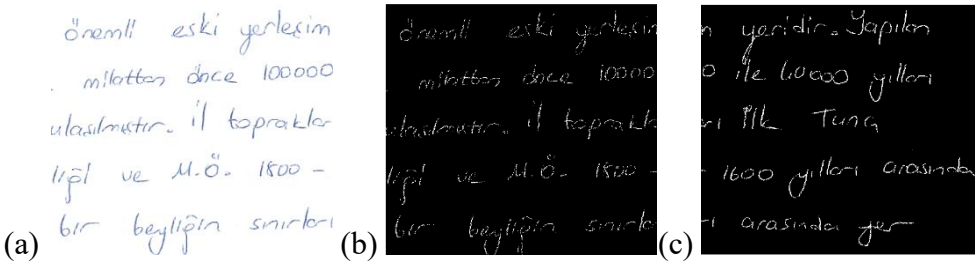


Figure 19 *Global Thresholding Applied to the Image: (a) original image, (b) manual thresholding, (c) Otsu's Thresholding*

As can be seen from the Fig. 19, Otsu's thresholding resulted in sharper images compare to manual thresholding.

3.3. Experiments Regarding Segmentation

This experiment adopts explicit segmentation and performs line, word, and character segmentation, respectively.

3.3.1. Line Segmentation

In this experiment, in order to perform line segmentation different methods have been applied such as smearing, Hough Transform (HT), Watershed Transform (WT) and Horizontal Projection. WT did not perform well since it tried to segment not only lines but also words and individual characters at the same time. The image is segmented into many regions which are not necessarily meaningful. In order to overcome this, WT was applied to a smeared image in which regions are more apparent. In order to perform the smoothing with smearing, run length smoothing algorithm was applied to the image. This algorithm smears consecutive white pixels along the horizontal direction [98]. WT was tried on different smearing levels. However, the results were not satisfactory to be adopted in our system. The results of the WT can be seen in the Fig. 20.

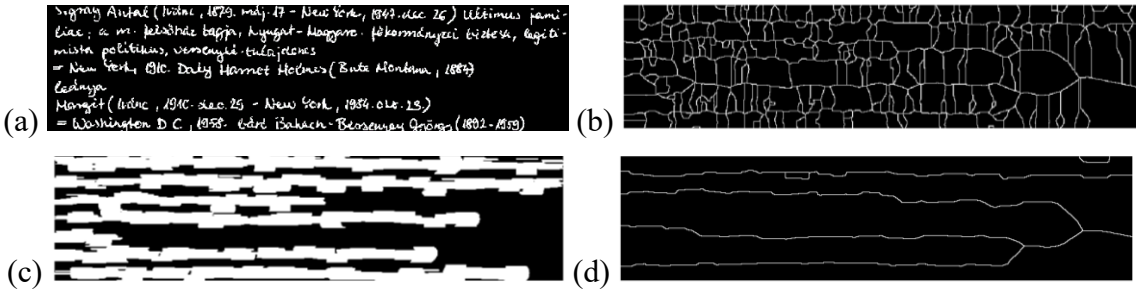


Figure 20 *Segmentation with Watershed Transform: (a) original image, (b) segmentation of the original image with watershed transform, (c) smeared image, (d) segmentation of smeared original image with watershed transform*

As another widespread method for several years, HT was applied to the image in order to find and then segment the lines [38], [99]. In contrary to WT, HT performed well for handwritings. The results can be seen in Fig. 21. The green lines show the detected lines.

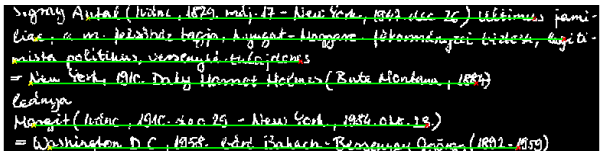
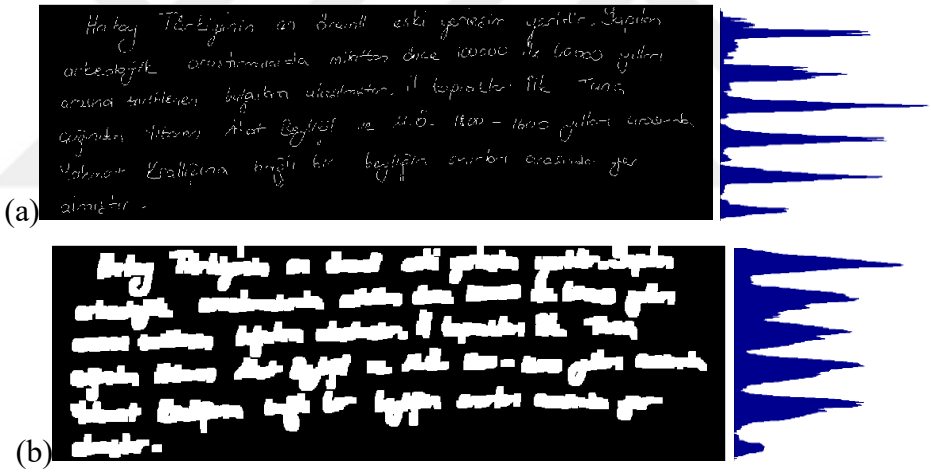


Figure 21 *Line Detection with Hough Transform*

Finally, Horizontal Histogram Projection was adopted for Turkish Handwriting Recognition purpose. This method is generally used for character, word and line segmentation [100]–[102]. Horizontal Projection provides the histogram of the sum of the black or white pixels. It can be seen from the Fig. 22, the histograms clearly indicate where a line ends and another one starts in both images.



(a)

(b)

Figure 22 *Horizontal Histogram Projection of the Images: (a) projection of the original image, (b) projection of the smoothened image*

A threshold value can be used separate the lines. However, any set threshold value might lead to cutting the tails of the words since there are overlaps between the lines caused by the tails of the letters. Instead, a very effective and simple method was adopted. As can be seen from the Fig. 23, after smoothing the histogram, finding the regional minima gives a great result for segmentation. The first histogram (a) is the histogram of the number of white pixels in each row and the second histogram (b) is the smoothened histogram a, by using ‘smooth’ operator. The last one (c) is the histogram of the regional minimas. In MATLAB, ‘imregionalmin’ sets the regional minimas into 1 where and the

rest into 0. This provides a straightforward way for finding the pixel where the lines should be separated.

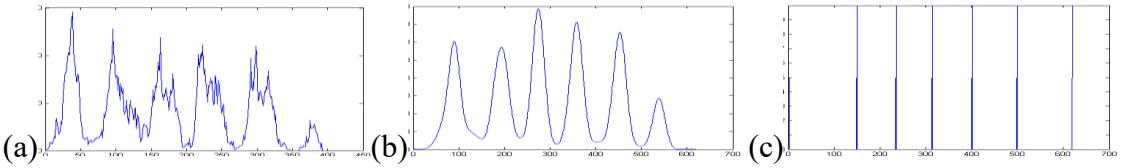


Figure 23 *Histograms of the lines: (a) histogram of the number of white pixels, (b) smoothened Histogram (a), (c) histogram of regional minimas*

3.3.2. Word Segmentation

Similarly to the line segmentation, a vertical projection method can be applied for word segmentation. As can be seen from the Fig. 24, vertical projection of an ordinary line has zero-pixel count at gaps between the words and within the words. In other words, any section of the line where there is not ink would be zero pixels at the vertical projection.

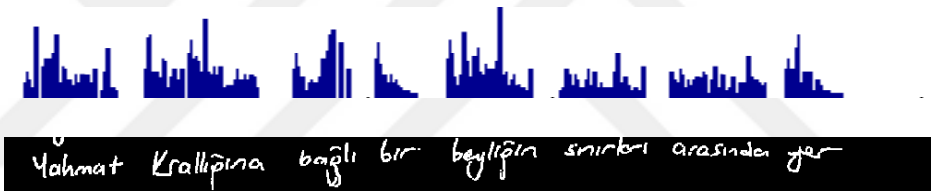


Figure 24 *Vertical Projection of a Line*

Gaps between the letters may also give zero-pixel count at vertical projection. In order to overcome the challenge, the length of the consecutive zero pixels were taken into consideration. For this purpose, mean of five smallest consecutive zero counts were set as the threshold. As an alternative threshold, a set 7 pixels also gave very similar result. Subsequently to this process, the words were segmented. A point worth mentioning is by segmenting the words this way; diacritical marks and parenthesis are also segmented. However, elimination of those at this stage would be more convenient than trying to eliminate those at the recognition phase. Therefore, another threshold was adopted in order to ignore segments including the diacritical marks which are smaller than the threshold. An example histogram of a line containing diacritical marks can be seen in Fig. 25.

Figure 25 *An example showing the histogram of an image with diacritical marks*

3.3.3. Character Segmentation

Several methods for character segmentation have been adopted over years. It is possible to group those into two categories as explicit segmentation and implicit segmentation [33]. Explicit Segmentation refers to partition of the word into individual characters. Then, the recognition is performed according to the results of segmentation. Therefore, the accuracy of the segmentation is very important for a successful recognition. Implicit Segmentation on the other hand tries to match any input over the image with its alphabet. It basically searches all possible locations across the whole word signal [103]. The recognition is performed at the same time with segmentation.

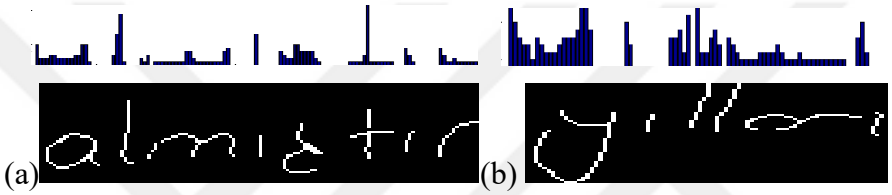


Figure 26 *Vertical projection of two words (a), (b)*

In our work, explicit segmentation was adopted. Vertical Projection of the words was calculated, and the individual characters were segmented according to the low pixel count at the point. However, when it comes to letters, a low number of pixels at a certain vertical line do not necessarily indicate a connection between letters. As can be seen from Fig. 26 and 27, it may also be the letter itself.

Figure 27 *Vertical projection of a Word*

In order to distinguish between letters and the connecting lines, another measure was adopted before segmenting according to the threshold. The threshold is the centre line which is acquired by the horizontal projection of the word. And the line containing the highest pixel run is considered as the centre line [20]. If the pixel count is below threshold at a certain vertical line, then location of the white pixels is checked. If they

belong to the upper part of the image, then there is no segmentation. On the other hand, if the white pixels are below or over the centre line then the segmentation is performed. This way, letters having low pixel count at their top are segmented correctly (h, m, n, r, t). Another addition to the algorithm is about the very last pixels. The algorithm ignores the low number of pixels in the very end of the image, in order to avoid segmentation of the ends of the letters into several pieces.

3.3.4. Evaluation of the Segmentation

Segmentation accuracy indicates how precise the prediction of the segmented image is. However, accuracy alone cannot be relied on in judging the performance of the given segmentation [104]. That is where the application of a confusion matrix can help. By setting the predicted segmentation against the true segmentation, not only the overall loss can be calculated, but the cost of different types of segmentation errors are also revealed by each cell of the matrix [105]. Table 4 shows the confusion matrix. In the table below, 'a' is the number of correct negative predictions, and 'b' shows the number of incorrect positive predictions, while in the following row 'c' is the number of incorrect negative predictions, and 'd' indicates the number of correct positive predictions [106].

Table 4 *Confusion Matrix*

		Predicted	
		Negative	*Positive*
Actual	*Negative*	a	b
	Positive	c	d

Four measures for evaluation were adopted namely accuracy, precision, recall and f measure. Table 5 represents those results of the line segmentation, word segmentation and character segmentation methods adopted. Line segmentation method worked very well with the data set. The accuracy for line segmentation is 96% for projection of histogram method. As can be seen in the Table 5, HT and WT did not perform as well as the projection of histogram method.

Table 5 *Results*

	Line		Word	Character	
Method	Projection of Histogram	HT	WT	Projection of Histogram	Projection of Histogram

Accuracy (%)	96	79	38	94	82
Precision	0,96	0,85	0,43	0,98	0,85
Recall	1	0,91	0,55	0,96	0,95
F-measure	0,98	0,88	0,55	0,97	0,89

In case of word segmentation, the accuracy was 94%. Words written very close to each other created the biggest miss-segmented words group. This challenge usually comes with cursive handwriting. Finally, character segmentation phase was completed, and the accuracy rate was only 82%. The method adopted did not perform very well on cursive handwritings. However, after modification certain difficulties were overcome. Segmentation of individual characters in cursive script is still problematic when it comes to real life documents with low quality.

3.4. Experiments on Feature Extraction and Classification Methods

In order to find the most suitable feature extraction methods for the study, the set of feature vectors are fed to the classifiers and the accuracy values of the classifications are used as the decision rule. Therefore, three different classifiers namely Support Vector Machines (SVM), Rough Sets Theory (RST) and Bayesian Networks (BN) are adopted for the classification purpose. The classification is carried out using the WEKA machine learning tool [4]. The reason behind the selection of the classifiers is that those classifiers take very little time to build the model in WEKA.

The classification task was carried out with and without applying feature selection. For the same data set, a supervised Correlation-based feature selection algorithm provided by WEKA is applied to the features. Additionally, the results of the classification without any feature selection are also given. Finally, the recognition is performed with three different cross validation values which are 5, 7- and 10-fold cross validation.

Results are represented in three different tables which represent features according to distribution of points, structural analysis, profile features, respectively. Although profile features belong to distribution of points group, they are represented by a large feature vectors therefore it can be beneficial to see if they are distinctive on their own. Thus, three different tables are created.

As can be seen from the Table 6, the accuracy for SVM with feature selection goes up to 94.4% followed by BN and RST. Although 94.4 % accuracy is the highest accuracy value recorded in three tables, it is not an ideal accuracy value for an OCR system. But it is a promising result considering that it is possible to combine the distribution of points features with others features in order to increase the accuracy. Thus, these features should be used for the system.

Table 6 *Accuracy values using the Distribution of Points Features*

	No Feature Selection			**Feature Selection**		
	5 fold	**7 fold**	**10 fold**	**5 fold**	**7 fold**	**10 fold**
SVM	88.2474 %	88.0412 %	88.866 %	92.7835 %	94.433 %	92.7835 %
RST	83.6654 %	83.6654 %	83.8125 %	91.6932 %	91.9965 %	91.9965 %
BN	85.567 %	85.567 %	84.9485 %	93.6082 %	93.4021 %	93.6082 %

Adopting only structural analysis features give the lowest accuracy values amongst the others due to the fact that the small number of features extracted. As can be seen in the Table 7, using only these features, accuracy can only go up to 69.8 percent. However, these features can be adopted with other values in order to distinguish certain characters such as characters with diacritical marks. Later on, it is planned to use these features in the second classifier where it is crucial to distinguish the letters with diacritical marks.

Table 7 *Results using the Structural Analysis features*

	No Feature Selection			**Feature Selection**		
	5 fold	**7 fold**	**10 fold**	**5 fold**	**7 fold**	**10 fold**
SVM	69.6907 %	67.6289 %	69.8969 %	There are not many features in		
RST	63.5052 %	63.7113 %	62.6804 %	total so there is feature selection		
BN	60.4124 %	60.2062 %	61.2371 %	significantly lowers accuracy		

Table 8 is an important table since it is based on one type of features. Although it is only based on profile features, it is a distinctive feature on its own. By only adopting profile features, up to 93.4 % of the characters were correctly classified. It can easily be said that projection profiles feature is one of the most distinctive features of all that have been adopted.

Table 8 *Results using the Projection Profiles Features*

	No Feature Selection			Feature Selection		
	5 fold	**7 fold**	**10 fold**	**5 fold**	**7 fold**	**10 fold**
SVM	86.3918 %	84.9485 %	87.4227 %	88.866 %	89.8969 %	89.4845 %
RST	85.0122 %	83.2548 %	84.0045 %	88,0412 %	88.4536 %	88,0412 %
BN	84.9485 %	84.1237 %	84.1237 %	93.1959 %	93.4021 %	91.3402 %

Looking at the Tables above it can be said that adaptation of one type of feature extraction methods is not enough for an OCR system. Hybrid methods should be tested, and the best combinations should be retrieved from the feature pool.

3.5. Finding the Suitable CNN Architecture

In this study, a CNN architecture is going to be adopted. Therefore, it is crucial to investigate the previous studies in the field of handwriting recognition to decide on the hyperparameters (number of layers, filter shapes, number of nodes) for the architecture. The literature review is provided in the next section.

Related Works

In this section, application of CNNs into the handwriting recognition problem is presented. One of the earlies applications of CNN like method into handwriting recognition was conducted in 1989 by LeCun et al. on Handwritten Zip Code Recognition [107]. Their model employed 3 hidden convolutional layers for feature extraction followed by a Backpropagation Neural Networks. The results showed 1.6% misclassification on training set, 8.1% on test set.

Classic CNN architectures include popular models such as LeNet-5 [108], AlexNet [109], GoogleNet [110] and VGG [111]. Out of these models, LeNet-5 originally designed and widely adopted for the field of handwriting recognition [108] (digit recognition on MNIST database [112]).

Simard et al. applied a CNN on to the MNIST database in 2003 [113]. Their CNN model consisted of two convolutional layers with 100 hidden units fed to the MLP. The results proposed that the error rate could go down to record breaking 0.4% using novel elastic training image deformations.

Ranzato et al. together with LeCun who was one of the creators of the LeNet model proposed another model for recognising digits on MNIST dataset and Caltech- 101 set of object categories in 2007 [108], [112], [114], [115]. They used a two layered method to extract features for a multi-stage Hubel-Wiesel type architecture. State- of-art accuracy was achieved on handwritten digits from the MNIST dataset with 0.62% error rate for supervised and 0.64% for unsupervised learning.

In 2010, Cireşan et al. applied 5 different MLPs with 2 to 9 hidden layers on the MNIST database for handwritten digit recognition [19], [112]. The best version out of the 5-variation performed 0.35% error rate exceling the previous works by 0.04 %. Later in 2011, Cireşan et al. conducted a study on handwritten characters using a seven deep CNNs trained on characters on NIST and MNIST databases in 2011 [84], [112], [81]. Their results indicate an advancement in the error rate compare to the previous studies by approximately 0.1% on digit recognition and 9% advancement in the 52 class Latin character recognition. Next year in 2012, Wu et al. applied their Cascaded heterogeneous 5 layered CNN model to the same MNIST database for handwritten digit recognition[116]. In comparison to the previous study conducted by Cireşan et al. on the same database [19],their study outperformed the previous one by bringing the error rate down to 0.23%.

Nui& Suen in 2012 adopted a hybrid CNN–Support Vector Machine (SVM) classifier for recognizing handwritten digits [58]. Feature extraction was performed using CNN and SVM was the classifier on the MNIST dataset. The recognition rate of the hybrid system was 99.81% without rejection, and a recognition rate of 94.40% with 5.60% rejection.

In 2015, Li et al. proposed a CNN-based handwritten character recognition framework and they applied it on handwritten digit (MNIST) and Chinese character (CASIA) datasets [112], [117]. The results showed that the error rate was 0.18% for MNIST dataset and 3.21% for CASIA dataset. Later in 2016, a study by Elleuch et al. was conducted on Arabic handwritten character datasets HACDB and IFN/ENIT by integrating two classifiers; CNN and SVM with dropout method in order to decrease overfitting [57]. Their model was similar to Nui& Suen's work since they also used 4 layered CNN for feature extraction and SVM as the classifier [58]. The model resulted in

2.09% error rate on 24 class HACDB, 5.83% on 66 class HACDB and 7.05% on 56 class IFN/ENIT database.

Proposed CNN Architecture

Various experiments have been carried out to figure out the architecture that works the best for such input. In order to have a base for the desired architecture, the Lenet5 architecture [108], which can be seen in the Fig. 28 below, was used with a modification in the output layer to match the class size of the input. It is important to use an established CNN model as a base to see the effects of the modifications on the architecture. LeNet-5 is a suitable model for the recognition of images with small input sizes and widely adopted for the field of handwriting recognition.

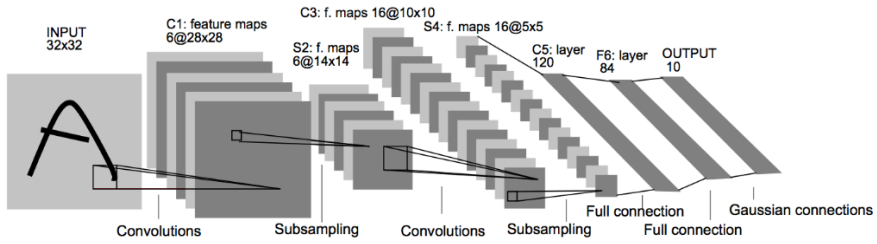


Figure 28 *LeNet-5 Model* [108]

A CNN has two types of hyperparameters as can be seen in the Table 9 below. The first type determines how the network is trained and the second type determines the network structure.

Table 9 *CNN Hyperparameters*[118]

Training Hyperparameters	Hyperparameters related to Network structure
Learning rate	Fully connected layers
Momentum	Convolutional layers
Learning rate decay	Number of filters
Maximum number of epochs	Filter size
Weight decay	
Early stopping patience	
Dropout rate	
Batch size	

In this work, size of the training set as well as maximum number of epochs, batch size, number and size of the convolutional layers, number and size of filters are going to

be changed for the experiments. Other hyperparameters that are going to be fixed are given in the Table 10 below.

Table 10 *Hyperparameters Kept the Same*

Hyperparameters	Values
Activation Function for Hidden Layers	ReLU
Activation Function at Output Layer	Softmax
Optimizer	Adam
Learning Rate	0,01

The experiments are run on Google Colab[119]. Google Colaboratory or "Colab" for short, lets anybody to write and execute python code through the web browser for free with some limitations. Limitations include the maximum 12 hours lifetime and disconnections from the virtual machines in case of leaving it idle for long. It is well suited for machine learning applications such as developing and training neural networks using popular libraries such as PyTorch, TensorFlow, Keras, and OpenCV. Executions are run on Google's cloud servers, using Google hardware including GPUs and TPUs.

Dataset:

The entire_augmented version from T-H-E Dataset including 156000 instances from 78 classes is used to find the most suitable architecture as can be seen in the Fig. 29 below. In depth details about the T-H-E Dataset is provided in the next chapter.

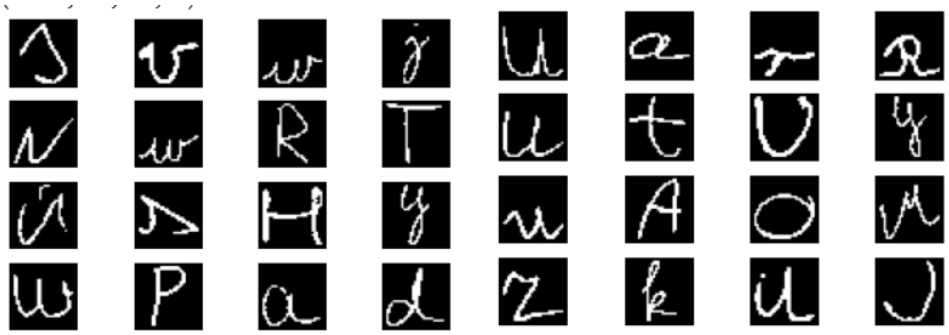


Figure 29 *Samples from the Dataset*

3.5.1. Experiment 1

The first experiment was carried out to find the optimal batch size on Lenet5 architecture. Batch size refers to the number of samples processed before the model is updated. Ideally, the batch size should be the power of 2 and a common use would be a

range between 16 and 256 for small datasets. In general, using a small batch size makes the training process faster since the weights are updated after each propagation. If we used all samples during propagation, we would make only 1 update for the network's parameter[120]. For instance, having 1000 training examples with batch size of 250, it would take 4 iterations to complete 1 epoch.

In this experiment the number of epochs was set to 10 since the aim of the experiment is not having the highest accuracy but finding the ideal batch size keeping the other hyperparameters fixed.

Table 11 *Experimental Results on the Batch Size*

Batch Size	Accuracy (%)	Confidence Rate
16	0.6415	±0.002773
32	0.6601	±0.002733
64	0.6718	±0.002726
128	0.6677	±0.002717
256	0.6304	±0.002697

3.5.2. Experiment 2

In the second experiment the number of epochs was determined using the LeNet5 model. Number of epochs is the number of times the entire training set goes past through the network. The maximum number of epochs are usually determined by the validation loss. As long as there is a decrease in loss, the learning should continue[118]. There is no limit to number of epochs, it can be between 1 and infinity however a very high number cause over fitting and very low number may not show the real potential of the network.

In this experiment, the batch size of 64 is going to be used principally based on the results in the previous experiment.

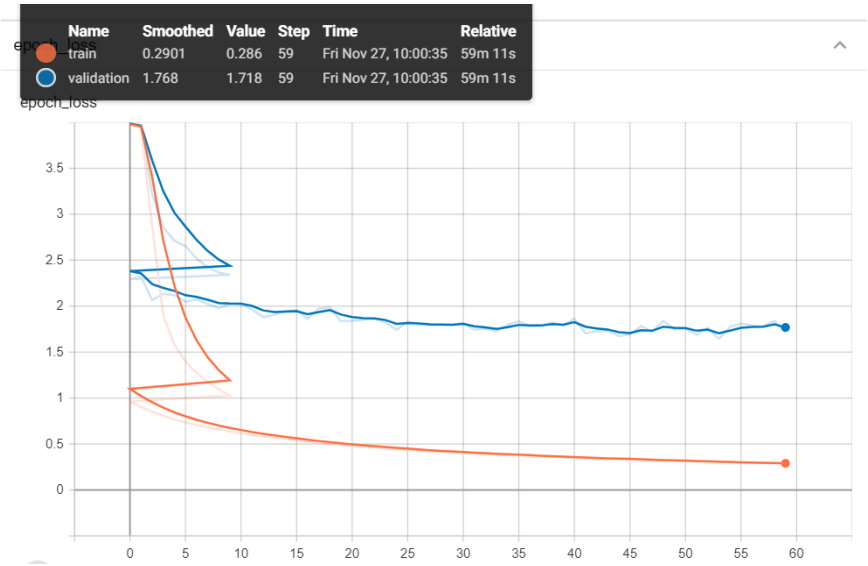


Figure 30 *Training and Validation Loss*

The figure above shows the change in training and validation loss over 60 epochs. Looking at the figure, 50 epochs are going to be used in the rest of the evaluation process.

3.5.3. Experiment 3

Pooling-layer parameters are examined in this experiment. Average filter and max filter are compared with two different size (2×2 and 3×3) and stride (1 and 2) values on Lenet5 model.

First, all four experiments were carried out for max filter. Based on the results, the better performing stride value for each size is applied using average filter. It should be noted that average filter results in blurring the image.

Table 12 *Results on Average and Max Filter*

Filter Type	Size	Stride	Accuracy (%)
Average	2×2	1	-
		2	0.49042
	3×3	1	0.5353
		2	-
Max	2×2	1	0.6069
		2	0.6801
	3×3	1	0.6171
		2	0.5905

3.5.4. Experiment 4

The first convolutional layer of a CNN extracts basic features such as edges from the input image. These features are called low-level features, providing minimal information about the input image. As we move further in the network, the extracted features get higher level, representing more meaningful information about the input. These features are called mid-level features. Complex, high-level features are found in the final layers of the network. Several experiments were run to find the ideal number of hidden layers to extract meaningful and distinct features. As a base, LeNet5 architecture in Fig 31 was used.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 78)                6630
=================================================================
Total params: 67,486
Trainable params: 67,486
Non-trainable params: 0
```

Figure 31 *The LeNet5 architecture used in the study*

Several modifications were applied to the base model starting with finding the best division in the input. Based on the commonly used distribution of the sets, 4 different distribution was tested. Additionally, the effect of shuffling the data was also evaluated as can be seen in the first 6 rows of the Table 13 below.

Table 13 *Evolution of the CNN Architecture*

#	Architecture	Accuracy	Precision	Recall	F-score
1	LeNet5-Shuffled Train 65%, Valid 15%, Test 20%	0.6505	0.7020	0.6182	0.6574
2	LeNet5-not shuffled Train 65%, Valid 15%, Test 20%	0.6809	0.7242	0.6582	0.6896
3	LeNet5-Shuffled Train 60%, Valid 20%, Test 20%	0.6709	0.7321	0.6316	0.6781
4	LeNet5-Not-Shuffled Train 60%, Valid 20%, Test 20%	0.6773	0.7031	0.6653	0.6837

5	LeNet5-Shuffled Train 70%, Valid 15%, Test 15%	0.6294	0.6748	0.6047	0.6378
6	LeNet5-Shuffled Train 70%, Valid 20%, Test 10%	0.6431	0.6851	0.6148	0.6480
7	Smaller filter size at shallow layers (3×3)	0.6857	0.7325	0.6577	0.6931
8	Double the convolutional layer at the beginning (2×2,3×3)	0.6758	0.6951	0.6649	0.6797
9	Double the convolutional layer at the beginning and increasing filter size (2×2×16, 3×3×32, 5×5×64)	0.7093	0.7228	0.7020	0.7123
10	Double the convolutional layer at the beginning and decreasing filter size (2×2×64, 3×3×32, 5×5×16)	0.5852	0.6585	0.5440	0.5958
11	Larger kernel size at the end 2×2,3×3,3×3,5×5 16,32,64,128	0.7291	0.7404	0.7237	0.7319
12	1 fully connected layer at the end	0.6879	0.7006	0.6813	0.6908
13	Insert one more layer	0.7122	0.7261	0.6066	0.6610
14	Use padding on pooling	0.7210	0.7327	0.7155	0.7240
15	Change size of Fully Connected layers	0.7325	0.7447	0.7262	0.7353

It is known that using small filters in the shallow layers lets us collect as much local information as possible and using larger filter size at the end of the network represents more high-level features[121], [122]. Based on that assumption, the size of the first convolutional filter was decreased in the row 7. Similarly, the size of the convolutional layer at deeper layers should increase. Affects can be seen in the row 11 of the table above.

In terms of fully connected layers, the original LeNet5 modified for this study has 400 (5×5×16) nodes connecting to 120 nodes and later those 120 nodes connecting to 84 nodes. Finally, 84 nodes connect to 76 nodes representing the class labels. In the Table 14 below, the changes made in the fully connected layers and the performance change is given. Convolutional layers have a greater impact on classification accuracy than fully connected layers. It should be noted that high sized fully connected layers drastically increase the number of parameters thus resulting in a costly training. Additionally, row

15 in the Table 13 above shows that having a single fully connected layer has a lower accuracy value than having two appropriately sized layers.

Table 14 *Effects of different sized fully connected layers*

Size of FCL 1	Size of FCL 2	Accuracy (%)
120	84	0.6809
150	90	0.6985
200	84	0.6631
300	84	0.6185

Based on the experiments carried out, the CNN architecture giving the best results for such input is shown in the Figure 32 below.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 31, 31, 16)        80
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 29, 29, 32)        4640
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 3, 3, 128)         204928
_________________________________________________________________
flatten_3 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_9 (Dense)              (None, 150)               172950
_________________________________________________________________
dense_10 (Dense)             (None, 90)                13590
_________________________________________________________________
dense_11 (Dense)             (None, 78)                7098
=================================================================
Total params: 421,782
Trainable params: 421,782
Non-trainable params: 0
_________________________________________________________________
```

Figure 32 *CNN Architecture with the best Results*

In the next section, the results of the experiments are explained.

3.5.5. Results

Several experiments were carried out to find a suitable CNN architecture which is able to extract the most distinct features and classifies the characters most accurately on such dataset. The LeNet-5 architecture was used as the base for the experiments and at every step, a parameter was changed to analyse the effect of the change. In the first two

experiments the ideal batch size and number of epochs were determined. The results show that batch size of 64 and 50 epochs are the optimal values for such input. The third experiment was conducted on finding the best pooling layer parameters. Max and average pooling were applied with different stride values (1 and 2) and filter sizes (2×2 and 3×3). Max pooling with stride of 2 and 2×2 filter size outperformed the other parameters. Finally, the ideal number of hidden layers are examined in the Experiment 4. Numerous changes were made to investigate the difference in the performance. Firstly, the ideal division of the input set was determined by adopting different sizes for training, validation, and test sets in the first 5 staps of the experiment. Additionally, the effects of shuffling the data were also tested in these experiments. The results showed that, 65% training, 15% validation and 20% test set on unshuffled data gave the highest accuracy by 68.07% and 0.6896 f-score value.

Differently from the LeNet-5, using small filters at the shallow layers give better accuracy considering the 32×32-pixel input size. Similarly, increasing the filter size and number of filters gradually has a positive effect on the classification performance. Additionally, using double convolutional layers at the end decreases accuracy whereas using double convolutional layers in the middle boosts the performance.

In terms of number and size of the fully connected layers, using a single layer instead of double appears to reduce the performance. Additionally, having a smooth decrease in the number of nodes also effects the performance positively.

Looking at the confusion matrix of LeNet-5 and the proposed architecture, the highly misclassified characters are mostly the upper and lower-case forms of the same characters. The highly misclassified characters are shown in the Table 15 below.

Table 15 *Highly Misclassified Characters*

	Accuracy	Highly Misclassified Characters
LeNet5	0.6809	c-C, i- İ j-J, o-O, p-P, ü-Ü, ű-Ü, x-X, v-V, z-Z
Proposed Architecture	0.7325	c-C, i- İ, l-I, s-S, o-O, p-P, x-X, v-V, w-W, z-Z

3.5.6. Evaluation

A CNN architecture was proposed based on numerous experiments in the previous section. Looking at the results of the experiments, effects of certain changes can be

generalised. Firstly, results indicate that using small filters the early steps help collection as much local information as possible. Similarly, the larger filters extract more distinct global features. Therefore, it is logical to use small filter sizes at the beginning and gradually increasing the size and the number of filters as the architecture goes deeper. Experiments showed that decreasing the filter throughout the architecture decreases the classification performance.

The effects of the filter size and number of filters are also investigated. The results show that, doubling the convolutional layers at the beginning with the same filter size and number of filters (2×2×16) gives lower accuracy value than doubling the layers with increasing filter size (2×2×16 followed by a 3×3×32). Additionally, not using padding on max pooling had a negative effect on the performance since it caused data loss. For example, a 15×15×32 input to max pooling without padding would give 7×7×32 output. However, adaptation of padding would allow us to get a 8×8×32 output for the same input thus preserving the important data. As can be seen in the Table 14, gradually decreasing the size of fully connected layers also have a positive effect on the performance. It should be noted that adding too many fully connected layers would result in an increase in the number of calculations since it contains most of the model parameters.

Looking at the highly misclassified characters, it might be beneficial to merge the upper and lower-case forms of some characters as in the T-H-E Dataset merged_augmented version in the HWR system and restore the upper and lower-case in the post-processing step to the original version. Thus, eliminate the errors caused by misclassifying upper and lower-case versions of the same letters. Additionally, certain misclassifications such as lower-case version of letter L, ”l”, and upper-case version of i, “I”, are going to be eliminated in the post-processing phase using the dictionary. It is natural for those letters which have almost the same representation to be misclassified in a context free recognition. Therefore, it is hoped that better classification performances are going be recorded with post-processing phase.

A CNN architecture was created in this section. The proposed architecture is going to be used in the deep learning based offline handwriting recognizer in the Chapter 6.

4. A MULTILINGUAL HANDWRITTEN CHARACTER DATASET

4.1. Chapter Overview

In this chapter, the details about the created multilingual handwritten character database are provided. Firstly, the motivation of creating such dataset and similar datasets are mentioned. Then, the structure of the proposed dataset is given. Lastly, the evaluation of the dataset is

4.2. T-H-E Dataset

Handwritten text datasets can be found in several forms, such as, full-page handwritten images, handwritten sentences, handwritten words and handwritten individual characters. However, the majority of the available datasets focus on a single language ignoring the existence of multilingual texts. As a result of globalization, multilingual handwritten texts are increasingly generated. By raising the number of multilingual datasets, the success on the single language handwriting recognition could similarly be achieved for multilingual handwritten texts. It is worth mentioning that offline handwriting recognition, of a single language, remains an unresolved problem, since there is no standard form in handwriting, unlike in print documents. Despite the available datasets for English characters, recognition of offline handwriting remains a challenge for several languages, such as, Turkish and Hungarian. In the case of Turkish, some researchers used datasets of their own which are not publicly available [9], [11], [12], [59], [123]. In order to be able to develop algorithms which deliver solid performance on handwritings with diacritical marks, handwritten character datasets on alphabets including a high number of diacritical marks are needed. In this paper we present a freely available character dataset consisting of 78 classes (Table 1) referring to 52 classes for English characters (26 upper-case+ 26 lower-case), 8 classes for special Turkish characters (4 upper-case+ 4 lower-case), 4 classes for Turkish and Hungarian joint characters (2 upper-case+ 2 lower-case) and 14 classes for special Hungarian characters (7 upper-case+ 7 lower-case) [59]. The two main reasons behind creating such dataset are lack of offline datasets for recognition of languages with special characters such as Turkish and Hungarian and secondly contributing to the existing Latin character datasets with a variety of handwritings collected from Hungarian and Turkish citizens. In addition to those motives, the proposed dataset offers an easier platform for designing multilingual unified recognition systems.

Handwritings collected from merely Turkish citizens mostly contain texts which are written using discrete characters only whereas texts written by Hungarians mainly consist of cursive characters. Gathering handwritings from both nations give the diversity to the dataset and such feature is believed to provide a positive impact on the classification process. In the next section, the earlier offline handwritten English character datasets and multilingual recognition systems are going to be represented.

4.3. Related Works

In machine learning, having access to right data in right format allows the researchers to develop, advance and assess their learning techniques. Therefore, regardless of the language of the handwriting, the condition and amount of the input data plays a crucial part in the performance of any handwriting recognition system. In this paper, we present a digitized, preprocessed, and labeled image dataset which consists of handwritten letters from three different languages. In the literature, handwritten character datasets can be found for several languages however, when it comes to multilingual handwritten character datasets, not many can be found. In order to be able to establish a multilingual recognition system, researchers either merge single language character/word datasets or adopt existing multilingual word datasets. The examples of multilingual handwriting recognition systems are presented in the following section. The majority of the studies focus on the recognition of English and French, due to the fact that there are existing datasets for those languages. In 2012, Wshah et al. used the IAM dataset [124] for English, the AMA dataset for Arabic [125] and the LAW dataset for Devanagari [126] together with a synthetic dataset in order to evaluate the proposed multilingual word spotting system [127]. Kozielski et al. carried out a study on recognizing real-world handwritten images in English and French in 2014 [128]. IAM, RIMES and Maurdor datasets [129] were used to train and evaluate their multilingual system. Bluche and Messina proposed a multilingual handwriting recognition system which was trained on datasets in English, French, Spanish, Portuguese, German, Italian and Russian in 2017 [130]. They used IAM, RIMES [131] and Maurdor datasets alongside with private collections they collected for those languages without available public or private datasets to evaluate their model. Lately in 2019, Swaileh et al. proposed a unified multilingual handwriting recognition system which was trained and evaluated using IAM and RIMES datasets for English and French respectively [132].

The abovementioned studies are all carried out on a combination of word or document-based datasets in different languages. The following section puts forward the most popular offline English handwritten character datasets. One of the earliest handwritten Latin character dataset, the CEDAR dataset, dates back to 1994, it consists of both handwritten words, such as, city names and postal codes and characters containing separated letters and numbers [133]. The separated letters and characters were put into 62 classes (26 upper-case+ 26 lower-case+ 10 digits) consisting of approximately 50000 samples. A year later, in 1995, the NIST Special Dataset 19 Hand printed Forms and character dataset was published containing full page binary image of handwritten forms and also characters (digits and letters) segmented from those forms (128×128). In the NIST dataset there are 62 labelled classes for digits '0-9', characters 'a-z' and 'A-Z'. Later in 1998, MNIST (Modified-NIST) dataset was created containing only digits (70000 samples) and it became a benchmark for digit recognition purposes [108]. In 2016, the 2nd version of the NIST dataset was published with full page binary images of 3699 handwritten sample forms and 814255 sample digits and characters of the same 62 classes [84]. It is possible to say that NIST dataset has become a benchmark for character recognition problem. Finally in 2017, EMNIST dataset, an extension of the MNIST dataset was published [83], [134]. EMNIST dataset is superior to its previous versions by many features such as number of instances, the balanced representation of characters, grayscale representation and the variety of classes provided. It contains 814255 samples of letters and digits (28×28). In addition to NIST and MNIST, EMNIST not only provides two class hierarchies namely By Class (every character into a different class with a different label) and By Merge (similar characters into the same class with the same label) but also provide four more options namely: balanced dataset which is easy to apply due to its balanced subset of all the By Merge classes; letters dataset generated to increase the number of errors occurring from case confusion by merging all of the uppercase and lowercase classes, to form a balanced 26-class classification task; digits dataset being a balanced subset of the digits dataset containing 28000 samples of each digit and a copy of MNIST dataset. Fig. 33 below shows the distribution of the different letters in the EMNIST By Class dataset.

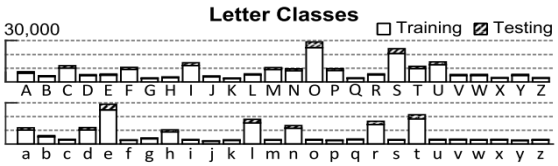


Figure 33 *Representation of the letters in the EMNIST By Class dataset* [83]

Finally in 2006, distinctly from previous datasets, a cursive character dataset C-Cube (Cursive Character Challenge) came out [135]. The C-Cube dataset includes 57293 characters including 26 upper and 26 lower case versions of each Latin letter. In our previous works, we adopted C-Cube data set after changing the way data was represented in the original dataset [136].

4.4. Data Set

The T-H-E Dataset includes handwritten letters from multiple alphabets, namely from English (ISO Basic Latin Alphabet), Turkish and Hungarian. However, since the dataset includes many Latin characters, it is very easy for other researchers to modify the data set for their needs (add/ remove special characters) and use it as a whole. The characters included in the dataset are presented in Table 16 below.

Table 16 *Characters in the Dataset*

	Lower case	Number of instances	Upper case	Number of instances
English Characters	a	2000	A	2000
	b	2000	B	2000
	c	2000	C	2000
	d	2000	D	2000
	e	2000	E	2000
	f	2000	F	2000
	g	2000	G	2000
	h	2000	H	2000
	i	2000	I	2000
	j	2000	J	2000
	k	2000	K	2000
	l	2000	L	2000
	m	2000	M	2000
	n	2000	N	2000
	o	2000	O	2000
	p	2000	P	2000
	q	2000	Q	2000

	r	2000	R	2000
	s	2000	S	2000
	t	2000	T	2000
	u	2000	U	2000
	v	2000	V	2000
	w	2000	W	2000
	x	2000	X	2000
	y	2000	Y	2000
	z	2000	Z	2000
Turkish Special Characters	ç	2000	Ç	2000
	ğ	2000	Ğ	2000
	ı	2000	İ	2000
	ş	2000	Ş	2000
Turkish and Hungarian Joint Characters	ö	2000	Ö	2000
	ü	2000	Ü	2000
Hungarian Special Characters	á	2000	Á	2000
	é	2000	É	2000
	í	2000	Í	2000
	ó	2000	Ó	2000
	ő	2000	Ő	2000
	ú	2000	Ú	2000
	ű	2000	Ű	2000
Total Number Of Characters	**39**	**78000**	**39**	**78000**

In order to generate the dataset, handwriting samples were collected, in an ethical way, from 200 participants who predominantly were at that time, high school and university students (Turkish and Hungarian citizens mixed), in a controlled environment. The participants were given a blank white paper and were asked to write the given text in their native language in their own handwriting. It can be said that there was less noise found in the images, since the paper used was new and blank. Then, the papers were scanned at 300 DPI. Subsequently, the images were thickened using morphological thickening provided by the MATLAB 9.3 environment [97] and line, word and character segmentation was performed [137]. These steps usually include a noise removal step, in order to get rid of the noise occurring in the scanned documents. However, the noise

removal step was skipped in order to maintain every diacritical mark in the images. The character segmentation phase includes several processes, namely, separating each character, getting rid of the white space around each character and binarization of the character, using Otsu's Algorithm [21]. Finally, every character is normalized to a 28×28 pixel shape. A representation of the sample characters, after the normalization step, can be found in Fig. 34.

Figure 34 *Sample Characters from the T-H-E Dataset*

4.5. Structure of the Dataset

Including characters from several alphabets, the T-H-E dataset is established in six versions, to provide for ease of use and flexibility when switching between alphabets, for different researchers with different approaches. The abovementioned six versions are explained below:

entire_augmented: This version represents the entire dataset. It includes all the 28×28 pixel binary characters from the three alphabets together forming a balanced dataset with 156000 characters belonging to 78 classes (Table 16).

tr_augmented: It consists of merely 12 Turkish special characters (6 upper-case and 6 lower-case). 2000 samples of each character can be found in this version forming a 24000-character dataset.

hu_augmented: Similar to the tr_augmented version, this includes 18 Hungarian special characters only (9 lower-case and 9 upper-case) forming a 36000-character dataset.

en_augmented: The fourth version includes 2000 samples of 52 English characters (26 upper-case and 26 lower-case) forming a 104000-character dataset.

This representation enables us to merge English letters with Hungarian special characters and work only on Hungarian characters by just putting two versions together. A fair warning should be provided about the Turkish alphabet; putting tr_augmented and en_augmented together does not result in the Turkish alphabet since there are no letters 'q', 'w' and 'x' in the Turkish alphabet. The users may want to exclude those 3 letters (3

lower-case and 3-upper-case) from the en_augmented in order to work on Turkish alphabet accurately.

merged_augmented: This version is derived from the entire_augmented version which includes all the characters from different alphabets together. The characters having a similar way of representation in their upper-case and lower-case form are put into the same class in this version such as lower case ‘o’ and upper case ‘O’. The characters merged are shown in the Table 17 below. In this group there are 55 classes and 156000 samples. However, only in this version are the number of instances, in each class, not balanced. Some classes have 2000 samples, while merged ones, are represented in 4000 samples.

Table 17 *Merged Characters*

	Merged Classes	Number of Instances		Merged Classes	Number of Instances
1	c- C	2000	13	s-S	2000
2	i-I	2000	14	ş-Ş	2000
3	í- Í	2000	15	u-U	2000
4	ı-İ	2000	16	ú -Ú	2000
5	j-J	2000	17	Ü-Ü	2000
6	k-K	2000	18	ű- Ű	2000
7	m-M	2000	19	v-V	2000
8	o-O	2000	20	w-W	2000
9	ó- Ó	2000	21	x-X	2000
10	Ö-Ö	2000	22	y-Y	2000
11	ő- Ő	2000	23	z-Z	2000
12	p-P	2000			

entire_raw: The original handwritten characters (1000 instances for every 78 classes) are put forward in the sixth version. Using this version, it is possible to experiment different distortion techniques and their impact on the classification performance can be tested. 78000 characters from 78 different classes, can be found in this version.

One important point to be noted is that there are 4 special characters (ü, ö, Ü and Ö) which are used both in Turkish and Hungarian, therefore, they repeat in tr_augmented

and hu_augmented versions. Another crucial point was discovered during the handwriting collection process concerning those 4 joint characters. In the Hungarian alphabet, there are two special characters ‘ő’ and ‘ö’ which are apparently represented in one single character ‘ö’ in Turkish alphabet (Similarly, letter ‘ü’ and ‘ű’ are presented as ‘ü’). The shape of the accent over the letter does not make a difference in the Turkish alphabet (based on the handwritings collected), however, they represent two different characters in the Hungarian alphabet. Therefore, it is crucial to understand the differences before carrying out the recognition. In order to avoid confusion, in this dataset, the Turkish and Hungarian joint characters, ‘ö’ and ‘ü’, were carefully segmented by adding only short, slanted versions into these classes, by avoiding some of the Turkish participants’ handwritings. Users might want to merge the classes ‘ö’ and ‘ő’ into one single class, if they are training for a Turkish recognition, instead of just discarding the letter ‘ő’ (the same applies for ‘ü’ and ‘ű’).

4.6. Data Augmentation

Augmenting the input image by applying distortions in order to increase the variance and therefore, performance, is a very common use both in character and text recognition [113], [138], [139]. Examples of different distortions such as shifting, scaling, skewing, and compression is represented in the popular MNIST dataset.

As represented in the previous section, the T-H-E dataset contains 2000 samples of every character. However, this number includes 1000 original handwritten characters and 1000 generated characters from those 1000 original characters. The number of handwritten characters was increased, by augmenting the existing characters by applying distortions on the original characters.

The augmentations include affine transformations and elastic distortions. Every single handwritten character is distorted randomly once using one of the distortions. If it is desired to have an even larger dataset, the same random distortion algorithm can be run on the original set time after time, generating 78000 randomly distorted character images at every attempt (the source code used for randomly generating images can be downloaded together with the dataset). The distortion methods applied on two different characters can be seen in the Fig. 35 below.

Tilting randomly to the left or right using Piecewise Linear Transformation: tilting right (50% chance) refers to moving the top left corner to the right randomly by 7 to 14 pixels and lowering it randomly by 5 to 12 pixels; tilting left (50% chance) refers to moving the bottom right corner to the left and top randomly by 1.1 to 1.5 times 28. After the tilt operation image is resized to the 28×28 pixel keeping the aspect ratio [140].

Adding Fisheye Effect: This effect was given by applying either barrel (50% chance) or pincushion effect (50% chance) randomly to the original image with a random distortion amount between 0.1 and 0.9 [141].

Rotating: Rotation of the images randomly to the left (50% chance) or right (50% chance) is applied by MATLAB [97]. Rotating to the left and right refers to randomly rotating the input by 5,10,15,20 or 25 degrees then resizing the result to fit the 28×28 matrix.

Figure 35 *Adopted Distortion Methods Applied to Two Different Characters*

4.7. Evaluation of the T-H-E Dataset

Deep learning is subtopic of machine learning that is capable of performing both supervised and unsupervised learning, using a feature, similar to the human brain, which is the ability to grasp patterns and recognize things accordingly [65]. Recent studies propose that deep learning algorithms outperform the traditional machine learning algorithms in the case of image classification since they do not deal with handcrafted features as can be seen in the Fig. 36 [66], [112], [142]–[144].

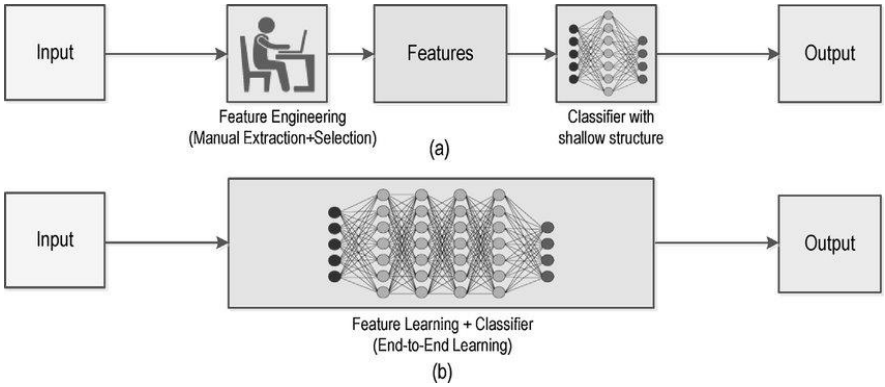


Figure 36 *(a) Traditional Machine Learning Workflow vs. (b) Deep Learning Workflow* [142]

Deep learning is made of multiple processing layers in order to learn representations of data with multiple levels of abstraction [4]. It is based on a hierarchically layered system, in which, each layer of nodes, is responsible for extracting distinct features using the previous layers' output. The further you go with the layers; the more advanced features can be extracted. In this study, a deep learning algorithm called Convolutional Neural Networks (CNN) is going to be adopted in order to evaluate the T-H-E dataset [75], [107].

A CNN architecture consists of an input layer, an output layer and hidden layers. An input can be a 1D signal, 2D image or 3D video. Thereafter, the input goes through a serious of hierarchical layers including convolutional layers, pooling layers in order to extract distinct features in the input. Finally, extracted features form the input layer of a Fully Connected MLP at the very end of the architecture for recognition. A brief CNN architecture can be seen in Fig. 37 below.

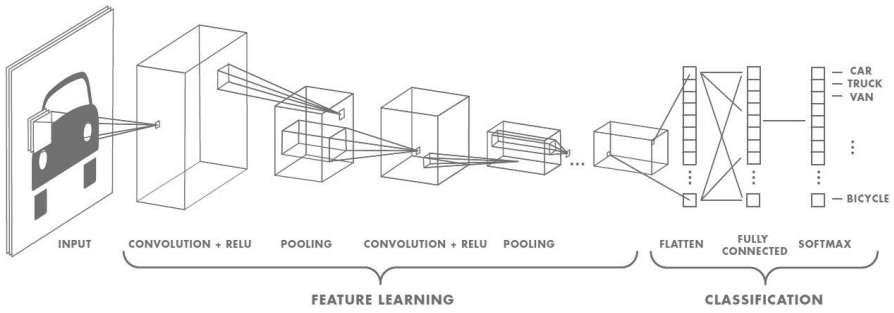


Figure 37 *Convolutional Neural Network Architecture [77]*

Classic CNN architectures include popular models such as LeNet-5 [108], AlexNet [109], GoogLeNet [110] and VGG [111]. Out of these models, LeNet-5 is the most

suitable model for the recognition of images with small input sizes and widely adopted for the field of handwriting recognition [108].

Convolution Layer: In this layer, a convolution filter is applied to the input matrix to generate feature maps as can be seen in Fig. 38. The size of the filter is pre-determined according to the input matrix [88].

Activation Layer (ReLU): ReLU operation replaces all negative pixel values in the feature map by zero [90]. Thus, it allows faster and more effective training.

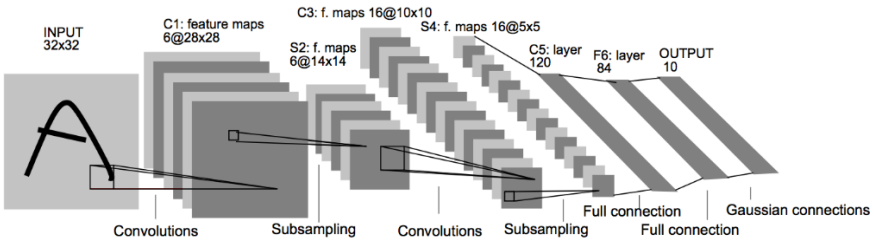


Figure 38 *LeNet-5 Model [56]*

Pooling Layer: Pooling layer aims at reducing the size of the feature maps for the next convolution layer generally by applying a sum, average or max filter to the feature map. However, the reduction does not necessarily result in data loss, but eliminates the least significant data resulting in easier computation in the upcoming layers [90], [91]. The operation performed by this layer is also called subsampling or downsampling.

4.8. Experiments

In this section two small scale experiments are represented, to confirm the validity and applicability of the proposed dataset. Additionally, third experiment compares the entire_augmented and merged_augmented datasets. As mentioned in the related works section, EMNIST dataset [83] has become a standard benchmark for handwriting character recognition purpose. Therefore, in order to evaluate the proposed dataset, the same LeNet-5 architecture was applied on both on the proposed dataset and EMNIST By_Class dataset with 20 epochs in the MATLAB 9.3 environment [97]. One important point to mention is LeNet-5 architecture requires 32×32-pixel images as the input. For that reason, all 28×28 images were widened to 32×32 images by adding black pixels to the margins of the images (left, right, bottom, and top). Another point to mention is the difference in the color of the input images in two datasets. The proposed dataset consists of characters 28×28-pixel binary images for every 78 class. However, the EMNIST

By_Class dataset includes 28×28 grayscale images for 52 classes representing the lower and upper case of English Alphabet (see Fig. 1).

In terms of validation parts, a similar validation partition to the evaluation of the EMNIST dataset [83] is applied. Every class was divided into two parts namely train and test parts without using validation. The training part consists of 900 and testing part 100 characters (90% and 10% for the experiment 3).

It should be noted that the first two experiments are carried out in order to evaluate the usability of the T-H-E dataset by comparing its results with a part of the EMNIST dataset which is the benchmark in the field. Having comparable results with EMNIST dataset is the main goal of the experiments. Therefore, the performance of the recognition is not paramount.

4.8.1. Experiment 1

The first experiment represents the comparison of en_augmented set and EMNIST By_Class dataset under equal conditions in terms of the input size and the colors of the input images. In order to have the same sample size for both datasets, 1000 characters out of 2000 characters for each class label in en_augmented set were randomly picked (52×1000=52000). As mentioned above, in EMNIST dataset characters are represented in 28×28 grayscale images in comparison to the binary 28×28 images in the T-H-E dataset. Therefore, for the first experiment, randomly chosen 1000 characters from all 52 letter classes (26 upper case and 26 lower case) from the EMNIST By_Class dataset were binarized using Otsu's algorithm [21].

4.8.2. Experiment 2

The second experiment is carried out very similarly to the first one. The only difference being that the original grayscale input images from the EMNIST dataset are kept as they are.

4.8.3. Experiment 3

In the last experiment entire_augmented and merged_augmented datasets are compared using the LeNet-5 architecture. Although the input sizes are the same in both versions (156000), entire_augmented has 78 class labels whereas merged_augmented only has 55 class labels. The difference in the size of the output is expected to result in the favor of the merged_augmented version with smaller class labels however, it should

also be noted that merged_augmented is an unbalanced set referring to the fact that not every class has the same number of instances (some have 2000 characters and others 4000). One of the previous studies conducted by the authors showed that the unbalanced nature of the dataset has a negative impact on the classification performance [136].

4.9. Results

This section puts forward the results of above-mentioned three experiments. MATLAB 9.3 environment was used for carrying out the experiments using the LeNet-5 architecture for feature extraction and classification. In first two experiments, the input was classified into 52 classes, whereas in the third experiment, two inputs had a different number of output sizes (78 and 55). Although, 20 epochs were set for the network, the experiments stopped after the 17th epoch in all 4 cases. The Fig. 39 below is a screenshot from the results of the en_augmented version of the proposed dataset in the 1st experiment. In the image, every column represents an epoch, and it can clearly be seen that the accuracy does not change significantly after the 3rd epoch.

The classification accuracies, 95% confidence intervals and highly misclassified letters, for all five inputs, in all three experiments, are shown in the Table 18 below. By looking at the results of Experiment 1 and 2, it can be seen that the portion randomly picked from the en_augmented dataset performed the best under such conditions compare to the randomly picked 52000 characters from the EMNIST By_Class dataset. Having the same input size and number of classes, the difference in the results could be explained by the fact that characters in the T-H-E dataset mainly consists of the handwritings of high school and university students. This may have brought about a more standardized way in handwriting. Although, the classification accuracy is slightly lower than 80%; as can be seen in the Table 19 below; misclassified letters are predominantly the same letters with their upper- or lower-case versions.

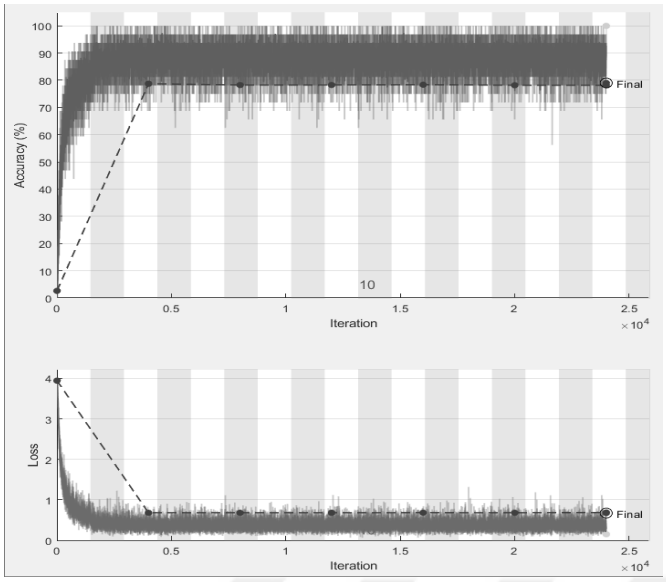


Figure 39 *Classification Performance of the T-H-E Dataset*

Looking at the different representations of EMNIST by_class dataset in Experiment 1 and Experiment 2, grayscale representation of the input images gave slightly better performance than the binary versions of the same images. As mentioned earlier, in this section the performance of the classifier was not crucial since the comparison was on the input not on the classifier. We believe that, adopting more sophisticated methods for classification and using a larger input set, with the participation of a more diverse group of people, rather than substantially students, could have a positive impact on the classification performance.

Table 18 *Classification Performances*

	Input	Input Size and #Classes	Classification Accuracy	95% Confidence Interval	Misclassified Letters
Exp.1	en_augmented	52000- 52	79.12%	1.10%	y-Y, z-Z, x-X
	EMNIST binary	52000- 52	74.77%	1.18%	p-P, t-T, J-m
Exp.2	EMNIST grayscale	52000- 52	75.58%	0.75%	p-P, t-T, J-m
Exp.3	entire_augmented	156000- 78	71.65%	0.60%	x-X, y-Y,p-P, ő-Ő
	merged_augmented	156000- 55	82.49%	0.71%	(i-I)-(í-Í), (ö-Ö)-(ő-Ő), (z-Z)-(x-X), r-(v-V)

The comparison of the 78-class entire_augmented set and 55-class merged_augmented set in the experiment 3 resulted in favor of the merged set. The overall accuracy for the entire_augmented version was recorded 71.65% whereas; the merged_augmented version had 82.49% accuracy. The performance difference in both datasets was mainly caused by the misclassification of the upper and lower-case letters. More specifically in the entire_augmented version, the letters 'x', 'y', 'p' and 'ő' were highly misclassified with their uppercase forms as can be seen from Table 18. However, in the merged_augmented form of the dataset, most of the misclassification was caused by inaccurately classifying similar letters such as 'ö' and 'ő'. A clearer and more detailed representation of the highly misclassified letters are demonstrated in the Table 19 below.

Table 19 *Detailed Description of the Highly Misclassified Letters*

Input	Letter	Accuracy	Letter	Accuracy
en_augmented	c	72.8%	C	90.1%
	x	53.9%	X	48.2%
	y	25.6%	Y	5.6%
	z	68.4%	Z	62.1%
EMNIST binary	p	52.5.%	P	54.1%
	t	51.9%	T	52.3%
	m	54.7%	J	53.3%
EMNIST grayscale	p	47.8%	P	49.4%
	t	56.0%	T	52.5%
	m	58.7%	J	55.6%
entire_augmented	x	50.3%	X	50.2%
	y	53.2%	Y	67.5%
	p	31.2%	P	79.8%
	ő	41.7%	Ő	26.5%
merged_augmented	i-I	81.8%	ı- İ	73.8%
	ö-Ö	87.7%	ő-Ő	82.2%
	z-Z	89.6%	x-X	75.6%
	r	86.3%	v-V	61.9%

Looking at the results, it can be said that merging upper and lower-case characters, have a positive effect on the class performance. Additionally, application of more

sophisticated classifiers might contribute to the elimination of the errors as well as increasing the size of the input images for the merged version. An interesting point is seen by looking at the results as the letter 'Y' only has 5.6% accuracy rate for the en_augmented input, whereas, it has 67.5% accuracy rate for entire_augmented input. Considering that both inputs are derived from the same characters, such a difference stands out. By looking deeper into the results, it can be seen that 83.3% of the letter 'Y's in en_augmented, are misclassified as the letter 'y'. The only apparent explanation for such gap can be the variation in the sample size in two inputs. As can be seen in Table 18, entire_augmented has 2000 samples of each character forming a 156000-character set whereas en_augmented used in the experiment 1 has only 1000 samples of each character forming a 52000-character set. The difference in the input size of the classifier and the sample size for each character may explain the difference in the recognition performance of the letter 'Y'. By carrying out the three experiments, the usability of the proposed dataset was evaluated in this section.

4.10. Conclusion

In this paper, a free-to-use, multilingual handwritten character dataset, compatible with different platforms and classifiers, is presented. The handwritings were collected in an ethical way, from 200 participants, representing a diverse mixture of Turkish and Hungarian citizens. The pre-processing and segmentation phases were described and in addition to those steps, the augmentation techniques used for the letters, are described herein. Finally, the evaluation of the T-H-E dataset is carried out in three different experiments. In the first two experiments, the English letters proposed in the T-H-E dataset, are compared to the EMNIST by_class dataset, which is the benchmark for English handwriting recognition. The results of the experiment 1 and 2 demonstrated that the T-H-E dataset outperformed the randomly chosen part of the EMNIST by_class dataset. This could result from the fact that the handwritings in the T-H-E dataset, may be more standardized, since the people contributing to it were mostly high school and university students or alternatively, the T-H-E dataset might include a greater variety in handwritings since it is collected from Turkish and Hungarian Citizens, thus, presenting more distinct examples for the deep learning algorithm, to learn from. Besides outperforming the other dataset, the en_augmented version, presented very few misclassifications between different letters. Having a 79.12% accuracy rate, majority of the errors were caused by misclassifying the same letters, with their upper- and lower-

case versions. This could easily be overlooked by merging the upper- and lower-case classes or at the post processing phase, of the recognition, by using a dictionary. As for the last experiment, the same LeNet-5 architecture was applied to two out of six different versions of the proposed dataset, namely, the entire_augmented and merged_augmented versions. Both versions had the same 156000-character input size, however, the output sizes differed. The version representing letters from three different alphabets separately both in upper- and lower-case classes included 78 letters whereas, the merged version had only 55 letters, merging similarly written upper- and lower-case letters into one class. Naturally, merging two classes into one, resulted in imbalance in the dataset, having 2000 samples for unmerged classes and 4000 samples in merged classes. Although merged_augmented has an imbalanced nature, it outperformed the entire_augmented version with over a 10% difference in accuracy rates, having only a 0.71% confidence score. As mentioned for the en_augmented version above, lower- and upper-case versions of the same characters form the highest misclassifications in the experiments. Therefore, having both versions put in the same class as in the merged_augmented version eliminates such inaccuracies. Having the six different versions provided in the T-H-E dataset makes it possible to test the performance of a classifier using the entire dataset, as well as carry out more specific tasks, such as, effects various distortions of characters, Turkish handwriting recognition and Hungarian and English mixed handwriting recognition.

Consequently, it is possible to say that the T-H-E dataset can be an alternative for earlier datasets, in terms of English character recognition, and outperforms those in terms of the variety of letters provided. In addition to being an alternative, it is the only handwritten Turkish and Hungarian handwritten character dataset in the field. The T-H-E dataset could be adopted for single language recognition purposes, namely, Turkish, Hungarian or English character recognition systems, as well as multilingual recognition systems. We believe, creation of multilingual character datasets will contribute to advancements in recognition systems, thus to the recognition of multilingual texts. Alongside with handwriting recognition, it could be used as an input, to evaluate other supervised and unsupervised learning systems.

In the next versions of the T-H-E dataset, we will aim at increasing the number of handwritten characters, as well as augmentation with meta-data regarding the participants, namely, the age, gender, occupation, left or right-handed, level of education

and nationality of the participants. It should be noted that finding a large number and diversity in participants, for such a purpose, is a major challenge, since the collection of the handwriting should be in person. While overcoming such a challenge, it may be possible to add other special characters from different languages, such as, Portuguese and/or French. We plan to add more alphabets to widen the scope of the dataset and we, in conjunction, plan to generate a handwritten document dataset consisting of handwritten documents in Turkish, Hungarian, and English.

4.11. Dataset Availability

All the data generated in this study including the paper are publicly available at https://github.com/bartosgaye/thedataset[145]. Additional data related to this paper may be requested from the authors.

5. AUGMENTATION OF T-H-E DATASET WITH SYNTHETIC CHARACTERS-A DEEP HANDWRITTEN CHARACTER GENERATOR

5.1. Chapter Overview

In this chapter, using Deep Convolutional Generative Adversarial Networks (DCGANs), synthetic characters are created. Firstly, generative models are described with popular deep generative models. Then, the structure and principles of DCGANs are explained followed by evaluation methods for synthetic data. Finally, experiments and results are presented.

5.2. Generative Models

Generative models are used to generate new examples based on an existing distribution of samples. Deep generative models represent the generative models and algorithms based on deep learning concepts. Such algorithms have become very popular in the past 5 years with impressively high performing applications such as painting, music generation, voice generation, photo editing and image generation such as human face and 3D images. The popularity also applies at the synthetic handwriting generation. Collection of handwriting from real users is a challenging task and there are not many examples of handwritten datasets in every language. Therefore, generative models are also used to generate handwritings. Commonly used deep generative models include Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), Deep Boltzmann Machines (DBMs) and Deep Belief Networks (DBNs). Detailed descriptions of the generative models are provided in the deep learning section of the study.

In this chapter, a way generation of handwritten characters is represented alongside with popular deep generative models and the evaluation techniques for such models.

5.3. GANS

In 2014, GANs were introduced by Goodfellow et al. [146]. They are trained using two neural network models for generative modelling. The first network captures the data distribution, and the second model decides whether a sample is from the training data or has been generated by the first model as can be seen in the Fig. 40 below. In other words, the output generated by the generative model tries to fool the discriminator.

The performance of the two models are inversely proportional in a way, since the better the generator performs, the harder it is for the discriminator to distinguish the real example or a generated one.

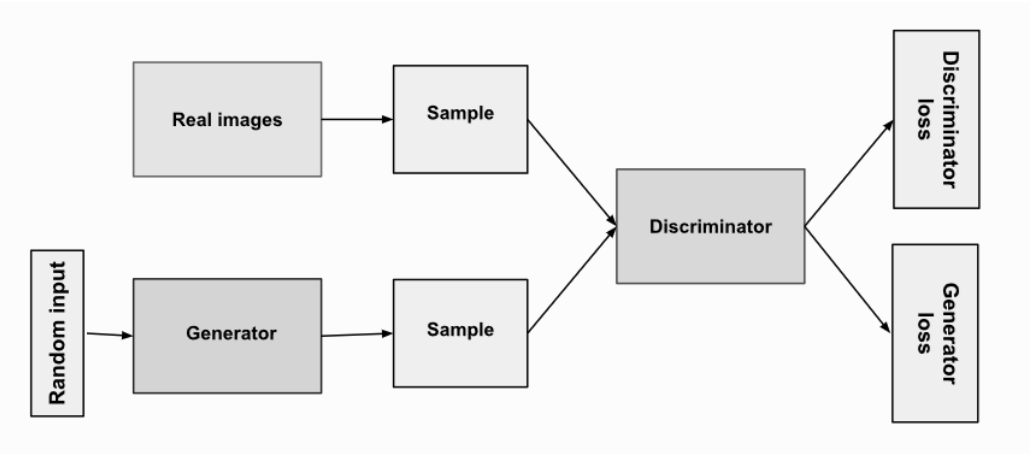


Figure 40 *Structure of the GANs* [147]

In 2015, Radfort et al. proposed Deep Convolutional Generative Adversarial Networks (DCGANS) using the transposed convolution technique instead of the simple fully connected network in GANs [148], [149]. Convolution and transposed convolution are reverse operations. Convolution takes a large input size and decrease the size of the input at every step; however, the transpose convolutions increase the size. As can be seen in the Fig. 41 below, DCGANs generates clearer images than GANs on MNIST dataset [148]. In 2020, Çelik and Talu published a study in which compare to other seven GAN architectures namely GAN, InfoGAN, SGAN, ACGAN, WGAN-GP, LSGAN; digits generated by DCGANs gave the second best classification accuracy after LSGANs and resulted in sharp images with the least amount of noise with WGANGPs[150]. Therefore, in this study having a simple architecture, DCGANs were used to generate new characters based on the characters in the T-H-E dataset.

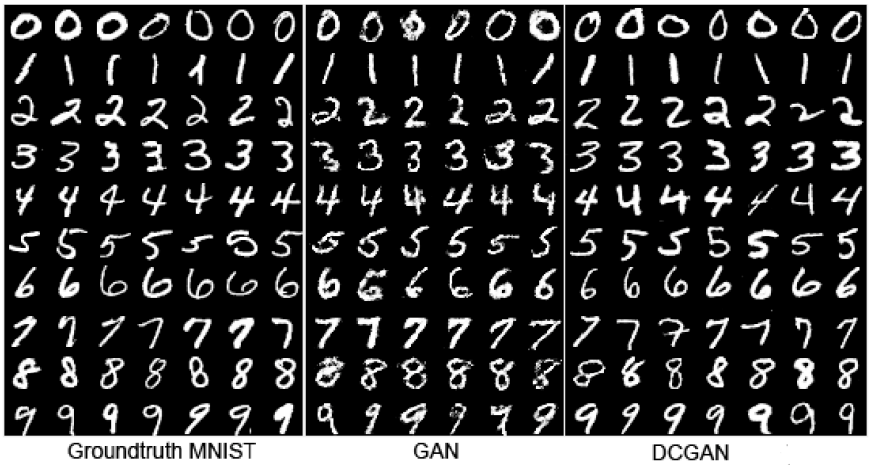


Figure 41 *Comparison of the Generated Characters by GANs and DCGANs* [148]

The structure of a DCGAN can be seen in the Fig. 42 below. The generator uses a multiple layers of transposed convolutions and the output is the generated image. On the other hand, the discriminator uses a convolution to classify the input image into two classes namely real image or generated image.

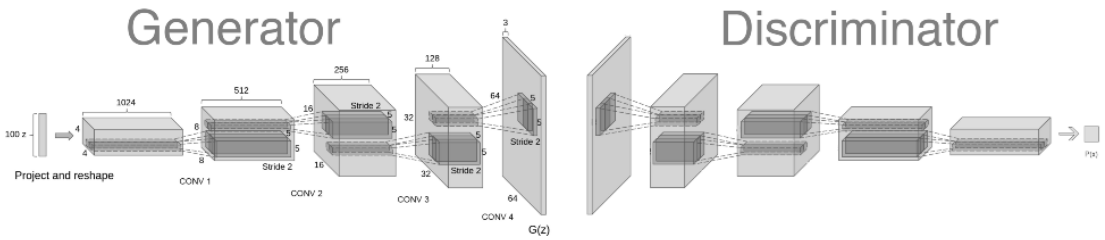


Figure 42 *Structure of a DCGAN for the Generation of a 64×64×3 image* [148]

Different to the CNN layers used in the previous section, the DCGAN adopts transposed convolution, Leaky ReLu and Tanh activation layers which are explained below:

Transposed Convolution: In convolutional layer, size of the input is decreased by applying a kernel. For example, a 3×3 input with a 2×2 kernel would result in a 2×2 output using a convolution. However, the transposed convolutional layer increases the input size by reversing the process. A 2×2 input with a 2×2 kernel would result in a 3×3 output using a transposed convolution as can be seen in the Fig. 43 below.

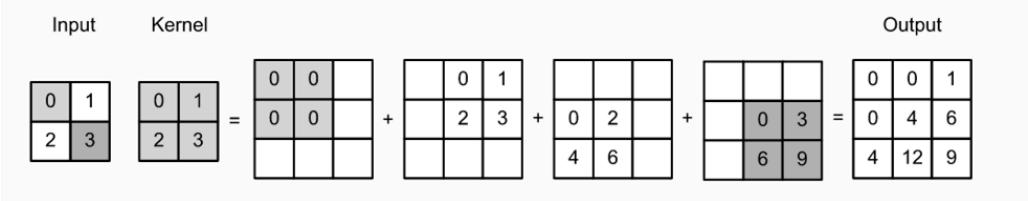


Figure 43 *Transposed Convolution* [151]

Leaky ReLu Activation Layer: A threshold operation is applied to the input smaller than zero by multiplying it with by a fixed scalar (usually the value of the scalar is 0.01).

Tanh (hyperbolic tangent) Activation Layer: It pushes the input value to the range [-1, 1] using Tangent Hyperbolic function.

5.4. Evaluation of GANs

Evaluation of the generated images by GANs is an ongoing research problem. Manual visual inspection has been a popular method for the purpose however it has its limitations such as, subjectivity, need for an expert in the field and time. Therefore, several qualitative and quantitative models have been proposed for the evaluation. Popular evaluation metrics are presented in two categories below.

Quantitative Evaluation Methods

These type of evaluation metrics are calculated based on specific numerical scores used to summarize the quality of generated images[152]. A detailed explanation of popular methods is given below.

Inception Score (IS): Firstly, the class probabilities are calculated by using the Inception Net[110] trained on the theImageNet[153] on the generated images and these represent the conditional probabilities. Thereafter, marginal probability is used which is the probability distribution of all generated images. Finally, the KL divergence (relative entropy) between the conditional and marginal probability distributions give the inception score[152], [154], [155]. The inception score is ranked between 1.0 and the highest value of the number of classes supported by the classification model in which 1.0 referring to the lowest possible performance.

Precision Recall and F1 Score: In terms of the evaluation of the GANs, precision refers to the quality of generated images, recall on the other hand presents how well the generated images cover the distribution of the original image[156]. It is different from the

IS in this case since IS does not take the original image into consideration. The F1 score is calculated by the harmonic average of precision and recall. These metrics rank between 0 and 1 and ideally a high precision recall value represents a high performance.

Frechet Inception Distance (FID): The FID attempts at improving the IS by comparing the statistics of generated images with the original images[157]. Similarly, FID uses the same Inception Net with the theImageNet input[110], [153]. However, in FID the calculations are done on both the generated and original images. The mean and covariance of images are calculated as a multivariate Gaussian for real and synthetic images. Thereafter, the Frechet distance (Wasserstein-2 distance) is calculated for two Gaussians. A lower FID score shows a better performance and 0.0 indicates the highest performance.

Maximum Mean Discrepancy (MMD): This method computes the distance between the generator distribution and the reference distribution[158]. When the value is large, it shows the likeliness of the samples being from different distributions.

Qualitative Evaluation Methods

These types of methods are not calculated and mainly require a human judge to evaluate the results[159]. Commonly used methods in the literature to qualitatively inspect the performance are explained below.

Nearest Neighbour (NN): This method calculates the distance between selected real images with generated images typically using Euclidean distance.

Rapid Scene Categorization: The synthetic and real images are shown to judges for a very short time frame and they are asked to distinguish between them[146].

5.5. Experiments

In this experiment, the aim is to generate 500 synthetic samples of characters from each class. Generated images DCGANs do not provide a class lable. Therefore each class was fed into the network manually and the generated images were then manually labeled. The dataset used to train the generator is entire_augmented version of T-H-E Dataset including 156000 binary samples of handwritten characters belonging to 78 classes (78×2000=156000).

First, the input is reshaped into a 2000×28×28×1 and then batched into batches of size 256. The generator model takes a 100×1 noise as input and generates a 28×28×1 character output in 50 epochs using the following steps in the Fig. 44 below.

The first attempt was carried out using Python 3.7 environment [160] on a laptop having Intel Core i7 8550 U 1.8 GHz processor, 8GB DDR4 2400 MHz Memory and NVIDIA GeForce930MX 2000MHz dedicated and Intel UHD Graphics 620 integrated GPU. However, the execution was extremely time consuming (time for epoch 1 was 30583.19077372551 sec-8 hours 49 minutes). It should be noted that DCGANs take a long time to train on regular computers therefore using cloud GPU services are recommended. Therefore, the environment was changed to Google Colab[119].

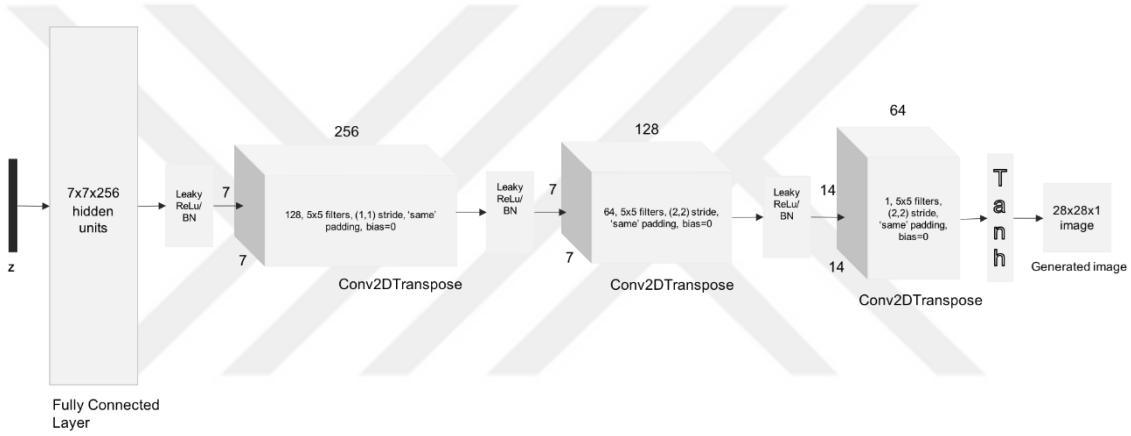


Figure 44 *The Structure of the Generator Model Adopted* [161]

As can be seen in the Fig. 44 above, the 100×1 noise vector **z** is the input for the fully connected layer (7×7×256 hidden units). The output of it if the input for the Batch Normalisation followed by a Leaky ReLu Activation. Later, the output is reshaped to 7×7×256 which is the input for the transposed convolutional layer. The output again goes through the Batch Normalisation and Leaky ReLu Activation. Then, the output is fed to the second transposed convolutional layer. Finally, the final Batch Normalisation and Leaky ReLu Activation is applied followed by the last transposed convolutional layer. The final output of size 28×28×1 goes through a Tanh Activation resulting in a generated image.

After the generation, the discriminator model takes generated and real images as input and using convolution it classifies those as real or fake (0 for fake image, 1 for the real image). The model adopted can be seen in the Fig. 45 below.

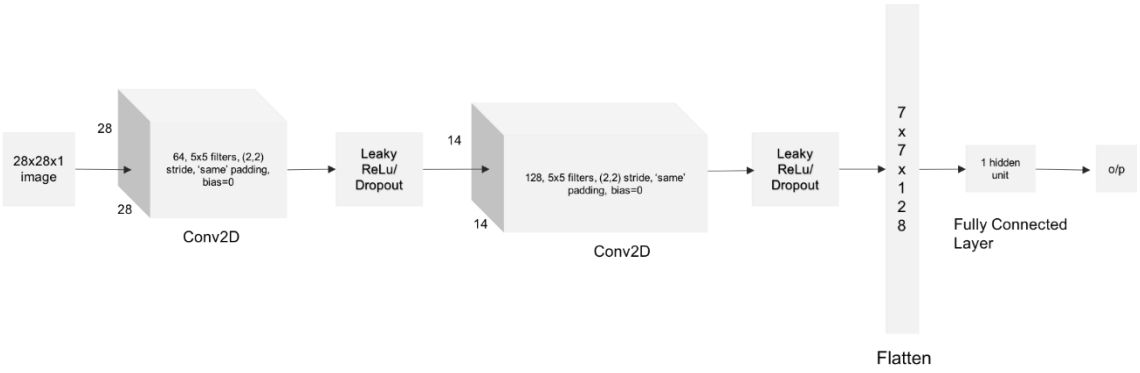


Figure 45 *The Structure of the Discriminator Model Adopted* [161]

5.6. Results

In order to determine the number of epochs on the experiments, the results were saved and subjectively evaluated for quality after every five epochs. Its because, generative models might start to create less similar images if trained long. Therefore, a manual inspection is needed. In the case of generating handwitten characters, the manual inspection is possible however, it may be challenging in case of generating more complex and large images. As mentioned earlier, generative models take long to run therefore running the model for 50 epochs gave a good enough result based on manual inspection. However, qualitative evaluation on its own might be misleading. Therefore, to evaluate the performance of the generated images, the metrics mentioned in the paper by Xu et al. [154] are used, using the repository for the same paper [162]. The Table 20 below includes sample images from epoch 1, 10, 20, 50 and 100 and the FID metric for the images generated at that epoch. Since the model was run on a single class at once, the numbers represented in the table below belong to a single class only.

Table 20 *Character Generation at Different Epochs and FID Metric*

	Epoch 1	Epoch 10	Epoch 20	Epoch 50	Epoch 100
Visual					
FID	145.45	106.01	98.77	59.14	62.36

The aim of generating synthetic data was to increase the number of handwritten characters in the T-H-E dataset. For this reason, 500 synthetic samples of each character are created. In the Fig. 46 below, the generated character after 50 epochs can be seen as well as the original characters from the T-H-E Dataset.

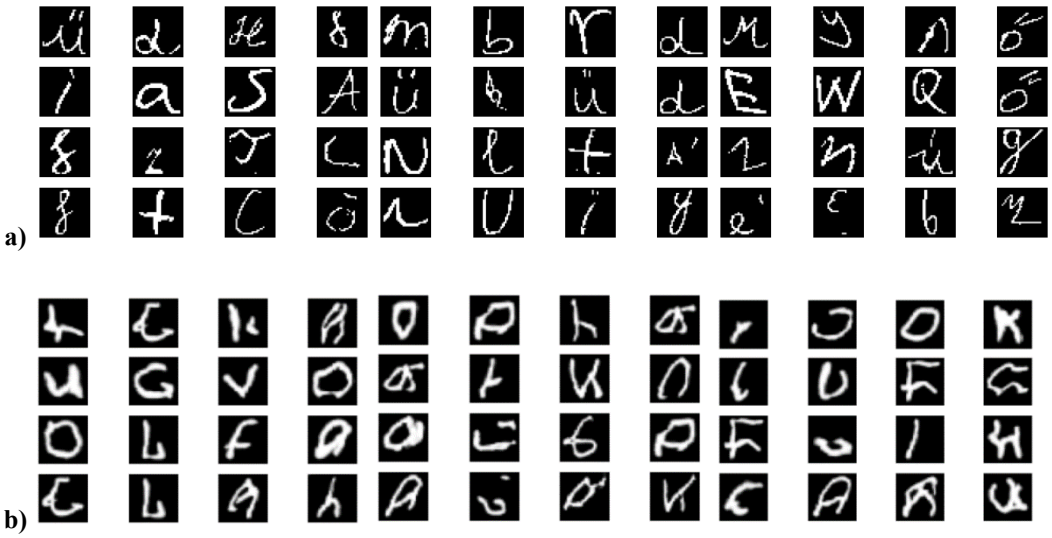


Figure 46 *a) Original Characters from the T-H-E Dataset, b) Generated Characters*

As mentioned in the previous chapter, the entire_augmented version of the proposed dataset contains 2000 (1000 original and 1000 augmented) samples of 78 characters from all three languages forming a 156000-character dataset. Using the original LeNet5 architecture, original dataset with 156000 characters and augmented dataset containing 195000 characters including 500 synthetic characters for each class are going to be classified and the performance difference is going to be evaluated. In both cases, the dataset is first shuffled, and the set is split into 60% training set, 20% validation set, and 20% test set and the experiment are run for 50 epochs with the batch size of 64. The accuracy values recorded can be seen in the Table 21 below.

Table 21 *Experimental Results on The Effects of Adding Synthetic Characters to the Database*

	Accuracy (%)
Original dataset 156000 characters	67.09
Dataset augmented with synthetic letters 195000 characters	67.32

5.7. Evaluation

Using DCGANs, 500 synthetic characters for each letter in three alphabets were generated in this chapter. In this section, the performance of the generative model and the augmentation of the T-H-E Dataset with the generated characters is going to be evaluated.

To evaluate the generated images qualitative and quantitative methods are used. In this study both methods were adopted to inspect the performance of generated characters. The manual inspection of the characters is not too challenging compare to more complex images. The change in the characters were watched every five epoch and 50 epochs gave an acceptable enough performance. There are two reasons behind the decision. Firstly, it is costly to run generative models and secondly, generative models might start to create less similar images if they are trained long. Additionally, the decision is supported using FID score.

The classification results for the original dataset and the augmented dataset can be seen in the Table 21 above. The results show that augmentation has a positive effect on the recognition performance. Although, the change in the accuracy value might seem little, it proves that the generated characters do not mislead the classifier. It is a known fact and can be seen in the classification results, more variety and high number in the training set adds to the classification performance. Collecting handwritings from people and processing those into a standard format with class labels are time consuming works. Generative models bring about an ease in this field by generating original characters without the need for the abovementioned processes. Based on these, the synthetic characters are a good alternative for augmenting the T-H-E Dataset.

6. DEEP LEARNING BASED OFFLINE HANDWRITING RECOGNIZER

6.1. Chapter Overview

In this chapter a segmentation based offline handwriting recognizer is given. The recognizer uses the deep learning architecture and the dataset augmented with synthetic characters mentioned in the previous two chapters. Firstly, the earlier works are mentioned followed by the elements in the recognizer system. Finally, examples from the recognizer are shown with results.

6.2. Literature Review

There are two approaches for the handwriting recognition problem. The first approach tries to solve it by classifying single-characters as in the case of MNIST and the second approach is full-word classifications. Recent studies on both approaches include deep learning algorithms. In this section, key studies on offline handwriting recognition on Latin characters are presented.

Graves and Schmidhuber combined multidimensional recurrent neural networks and connectionist temporal classification for handwriting recognition purpose in 2008[16]. The input the Multi-Dimensional Long Short-Term Memory Recurrent Neural Networks (MDLSTM) system is raw pixel data unlike previous systems thus it is applicable to any language. The system outperformed any other system attending the ICDAR 2007 Arabic recognition contest. Next year in 2009, Graves et al.[18] published a study in which they compare the HMM based online and offline handwriting recognition with their proposed approach for hard to segment data based on a novel type of recurrent neural network. On IAM database using bidirectional LSTM(BLSTM) architecture to gain long-range bidirectional contextual information and the CTC output layer allowing the system to train on unsegmented sequence data, the proposed system had around 74%-word accuracy in offline recognition compare to 64% for HMM based approach. In 2014, Zamora-Martínez et al. applied two different system for offline HWR namely BLSTM and hybrid HMM/ANN models[163]. The mail aim of the study was to determine how the dictionary size (55K and 103K) and application of Neural Network Language Models (NN LMs)[164] to estimate word probabilities effect the overall system. The results indicated that hybrid HMM/ANN system performed better with large vocabularies and worked better with NN LMs than the BLSTM NN system. A report was published by Yan in 2016 on offline HRW and she applied Faster R-CNN (FRCNN) object detection model for

handwritten character and word recognition[165]. The database used was IAM Database with synthetic handwriting she generated. The study showed that such method was not very useful on cursive handwritten images without knowing the context of the text. The same year, Doetsch et al.[166] proposed an attention based RNN[167] based end-end offline handwriting recogniser. The experiments were carried out on the RIMES[131] database. Their research consisted of two parts, the first part was the recognition of isolated word recognition with class label of 82 characters and full text lines as the second having 96 characters as the class labels. The network was applied on the line images or word images. The result of the study showed that end-to-end systems are also a good alternative to traditional hybrid HMM systems. Similarly to the previous study, Bluche et al. proposed an attention based end to end recognition system[168]. In the system the attention is implemented with a MDLSTM[16]. Differently from the previous study by Doetsch et al., this study applies the network onto the paragraph images using IAM Database[124]. In 2018, Wigington et al.[169] presented a model that jointly learns text detection, segmentation, and recognition using Region Proposal Network to find the start position of text lines and CNN-LSTM network using 2017 ICDAR HWR dataset[170], IAM and RIMES datasets. The proposed system outperformed the winner of the 2017 ICDAR handwriting recognition competition. In 2019. Chung and Delteil proposed a CNN- BLSTM[171] based system tested on IAM Database[172]. The results showed that the proposed system reduced computational costs compared to existing methods.

6.3. Proposed Method

The proposed recognizer consists of three main components namely, pre-processing, classifier and post-processing as can be seen in the Fig. 47 below.

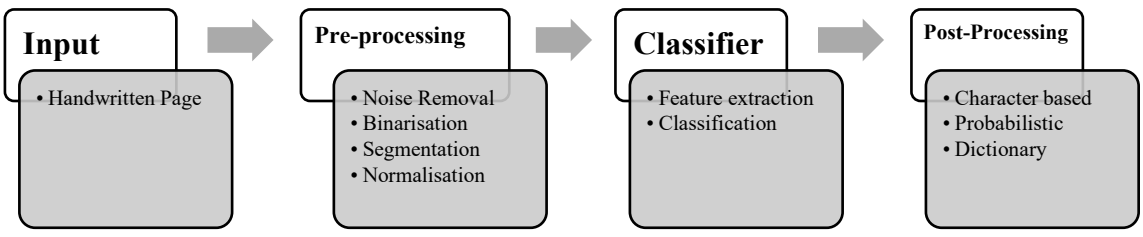


Figure 47 *Structure of the Recognizer*

6.4. Input

The input to the proposed system is a handwritten document in English, Hungarian or Turkish language. The system is capable of recognising documents in mix languages as well, therefore the document can also be multilingual. The document is digitalised preferably with a scanner with at least 600 DPI however the system also works well with high resolution pictures taken with a mobile phone or digital camera.

6.5. Pre-Processing

This step includes several processes namely noise removal, binarization, segmentation of lines and characters, and normalisation of the segmented characters.

6.5.1. Noise Removal

In this step, the input image goes through a morphological closing operation.

6.5.2. Binarization

Otsu's thresholding is applied for binarizing the image[21].

6.5.3. Segmentation process

The segmentation process starts with the line segmentation. The binary image is segmented into separate lines using Maximally Stable Extremal Regions (MSER) algorithm[173]. MSER detects regions which stay nearly the same through a wide range of thresholds.

Detection of separate lines is followed by character segmentation step. Firstly, the core-zone of the segmented line is detected. The zone between the upper baseline and lower baseline is called core-zone and the majority of the connections are carried out in this zone in cursive handwritings. The logic behind the core-zone is finding the zone in which most of the handwritings are performed. The upper and lower baselines are calculated using horizontal projection of the text line as mentioned in the section 3.3.1. Looking at the projection of the line from top to bottom, where there is a big increase in the number of white pixels is the upper baseline; and looking from bottom to top, the place where there is a big jump in the white pixel count is the lower baseline. The big jump is calculated by finding the first and the last regional minima on the smoothened histogram. The core zone is the zone between these two lines. Segmentation of the characters are performed based on the core zone.

Like the logic in the section 3.3.1, the number of white pixels is calculated for every vertical column in order to have a clearer understanding of the regions on the core-zone. Additionally, taking only core zones into consideration eliminates the segmentation errors caused by the tails of letters such as “y” and “g”. The columns containing zero white pixels are removed at the beginning of each line. After several experiments, two threshold values are set. The first one is to determine the minimal space between two characters and the second one to detect the space between words which is later used at the post-processing step. The character segmentation is performed in three rounds. In the first round, looking at the sum of the white pixels, starting from the left to the right every zero-pixel count place is assumed as a cut point representing separate characters. The segmented characters are saved together with the value of the beginning pixel and the ending pixel of the character. The consecutive zero pixels are opted out. Additionally, the black pixels from the top and bottom of the characters are removed. After that process, If the cut character has less than 3-pixel width or 4-pixel height, that character is opted out as noise. In the second round, touching characters are detected. By looking at the sum of the white pixels, places having a 1-pixel count are taken into consideration to investigate if they are in fact a connection. If the character has a width larger than the second threshold value and has a 1-pixel count sum within, then the first one-pixel count from the left is considered a cut point. And as in the first round, the beginning and the end pixels are saved for both characters after segmentation. In the third round, every character is sorted based on their starting pixel value and the distance between the end of each letter and beginning of the next letter is calculated. Using the second threshold value which is minimum 12 pixels, if the distance between characters is higher than 11, number of the ending pixel is saved as the place of the space. The result can be seen in the Fig. 48 below.

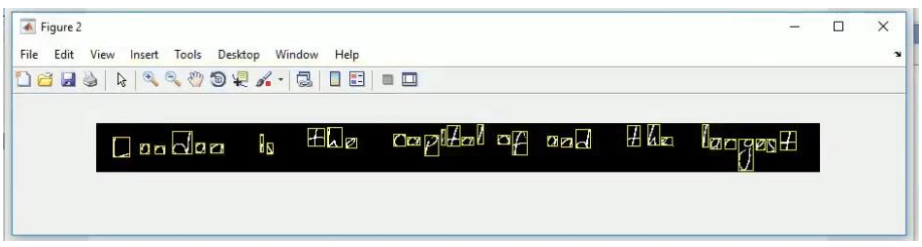


Figure 48 *Character Segmentation Process*

After performing the segmentation, the cut characters are normalised to 28×28 size using the algorithm below:

```
    [x y]= size of the image

    if

    x <= 28 && y <= 28

    then add black pixels to necessary sides until the size   is
28 (left-right or upper-lower)

    else

    x>28 || y>28

    resize the larger one to 28 keeping the aspect ratio add
black pixels to bring the smaller size up to 28

    end
```

6.6. Feature Extraction and Recognition

To carry out the recognition, the CNN architecture proposed in the Chapter 3 is adopted. The architecture of the network can be seen in the Fig. 49 below. Depending on the class size, the last layer representing the number of classes in the below mentioned architecture is going to be modified.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 31, 31, 16)        80
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 29, 29, 32)        4640
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 3, 3, 128)         204928
_________________________________________________________________
flatten_3 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_9 (Dense)              (None, 150)               172950
_________________________________________________________________
dense_10 (Dense)             (None, 90)                13590
_________________________________________________________________
dense_11 (Dense)             (None, 78)                7098
=================================================================
Total params: 421,782
Trainable params: 421,782
Non-trainable params: 0
_________________________________________________________________
```

Figure 49 *CNN Architecture with the best Results*

The recognizer is going to be tested on the recognition of texts in several languages. Therefore, the model is trained on different datasets for the recognition of different

languages. The versions used to train the network for the recognition of Turkish text, Hungarian text, English text, and multilingual text can be seen in the Table 22 below. It should be noted that synthetic characters are also added to train the network.

Table 22 *The Datasets used for training the Object Detection Algorithm*

Text Language	***Version used from the T-H-E Dataset***	***#classes***
Turkish	en_augmented tr_augmented	64
Hungarian	en_augmented hu_augmented	70
English	en_augmented	52
Multilingual	entire_augmented	78

6.7. Post-processing

The post-processing unit is used to overcome misclassifications. A list of words in the form of a text file is used as the dictionary in this step. The structure of the text file can be a list of words or basically a text containing several words such as a book. In our work a book for English post-processing and list of words collected from several sources for Turkish and Hungarian post-processing is used [174], [175], [176].

The output of the recognition is fed to the postprocessor line by line together with the position of spaces between characters represented with the characters as can be seen in the example below:

Output of recognition:LondonisthecaPitai,6,8,11

In the next step, every letter in the output of the recognition step and every word in the dictionary is converted to lower-case characters in order to make sure that the distance is calculated with minimal errors. Finally, the lower-case form of the words is fed to the post-processor.

Levenshtein distance is calculated for the recognised words and every single word in the dictionary and the minimum distanced match is given as the output[177]. This algorithm is used to find similarities between two strings by insertion, deletion and substitution of symbols and used in various areas such as text retrieval, speech recognition

and spell correction. Based on its performance on comparing two strings with low cost, the Generalized Levenshtein Distance (GLD) algorithm which is often called as “edit distance” algorithm is applied to calculate the distance between the recognised word and the words in the dictionary[178].

The calculation is performed recursively using the formula below, in which *a* and *b* refer to the string 1 and string 2; *i* and *j* refer to the character position of the string 1 and string 2 respectively. In the calculation, every edit referring to insertion, deletion and substitution of characters or symbols has cost of 1. In case there are multiple matches with the same distance, then the first one in the dictionary is the output.

Equation 1 *The Edit Distance Algorithm*

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

A comparison of the word *“cat”* and *“mate*” and the calculation of the distances in the matrix is demonstrated in the Table 23 below.

Table 23 *The Edit Distance Table*

	#	C	A	T
#	0	1	2	3
M	1	1	2	3
A	2	2	1	2
T	3	3	2	1
E	4	4	3	2

In the final step, the matching word is compared with the original dictionary and it is normalised into its form in the dictionary to be able to maintain the capital letters in certain words.

***Output of recognition**: LonDon*

***Input to post-processing:** london*

***Output of the post-processing:** London*

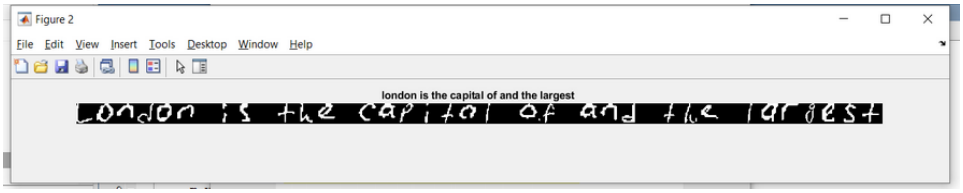


Figure 50 *Output of the Recognizer*

The evaluation of the post-processing method is carried out by calculating the word error rate before and after the implementation of the post-processer. The results are provided in the next section.

6.8. Evaluation of the Recognizer

The performance evaluation is carried out separately for Turkish, Hungarian, English, and multilingual texts. As for the input, the handwritten pages to generate the T-H-E Dataset were used for single language recognition purpose (20 pages for each language). As for the multilingual handwriting recognition, a small set of handwritten pages (20 pages) containing text from all three languages were generated by 10 participants.

Since the detection is carried out line by line, the evaluation was also carried out line by line with manual supervision. Since the pages used to evaluate the recognizer do not belong to a dataset, they were not tagged with the actual words. Therefore, an expert converted those pages into strings by manually adding each word into a .txt file format. Every handwritten document is saved into a separate file containing one word per row. Then, the .txt files were used to evaluate the performance.

The word error rate (WER) for each language is calculated to measure the performance. As mentioned in the previous section, the Levenshtein distance for each word was calculated for post-processing purpose. WER refers to the Levenshtein distance of the words divided by the number of words. The WER metric gives a better understanding of the misclassification than merely the number of misclassified words divided by the number of total words since it takes the distance into consideration. The Table 24 below shows the WER and percentage of the misclassified words for each language.

As mentioned in the previous section, in order to evaluate the effectiveness and the necessity of the post-processing step, the evaluation metrics are calculated before and after the post-processing step. The Table 24 below includes the results of both cases.

Table 24 *Word Error Rate of Handwritten Texts in Different Languages Before and After Post-processing*

	Input	WER (%)	Percentage of misclassified words	#words
Without Post-processing	**Turkish Text**	30.39	20.85	4758
	Hungarian Text	32.96	23.16	5012
	English Text	23.66	16.51	4826
	Multilingual Text	46.97	36.76	4062
With Post-processing	**Turkish Text**	11.37	8.89	4758
	Hungarian Text	14.01	11.55	5012
	English Text	8.21	6.24	4826
	Multilingual Text	20.56	16.05	4062

Looking at the Table 24 above, it is clear that both the WER and percentage of misclassified words are significantly higher in all four inputs when there is no post-processing step applied. Therefore, it can be said that the post-processing step improves the performance of the recognizer and it is going to be applied in both deep learning based recognizers in this study.

The results show that in terms of detection of single languages, the lowest performance was recorded in Hungarian text. The cursive nature of Hungarian handwriting brings about a challenge in the character segmentation phase. Therefore, the overall recognition is negatively affected by mis-segmented characters. The number of mis-classified words in Hungarian is 597 and the total Levenshtein distance of those 597 words is 702. Which means that most of the words had approximately one change to be classified correctly. Additionally, Hungarian alphabet consists of several letters with accents. These letters are highly likely to be misclassified as one-another. This also adds to the high number in the WER in Hungarian text recognition.

The reason behind English text having the highest recognition rate can be based on two reasons. The first one is that the number of classes are smaller in the English alphabet and there are not many characters that have an accent. This should have a positive effect

on the classification performance. The second one is that the nature of English text in the dataset consists of mostly print characters which are rarely overlapping thus making the character segmentation easier.

Multilingual text recognition yields the highest error rate overall. The size of the dictionary in multilingual recognition is approximately triple the size of the single language dictionary. Therefore, the chances of misclassification of words are higher. In addition to it, having the largest class size also has a negative effect on the accuracy. Compare to English text recognition with 52 class size, multilingual text detection is expected to have a lower accuracy rate since it includes 78 class labels.

In order to create a base of comparison for the proposed method, the pre-trained model on the merged_augmented version of the T-H-E dataset was used to recognize English handwritings in the IAM Database[124]. The IAM database includes full English sentences collected from approximately 400 participants, containing 82,227 words. Since the input to post-processor is only lower-case letters, the upper-lower case problem caused by the merged_augmented version is eliminated at the post-processing. The WER of the proposed model with the results of published models are provided in the Table 25 below.

Table 25 *Results on IAM Dataset*

Model	**Test WER (%)**	**Language model**
Proposed Model	17.4	Yes
Bluche[179]	16.4	Yes
Wigington[169]	23.2	No

Looking at the Table 25, it should be noted that although a language model was used in the study by Wigington[169], the results were only provided for the version without language model. The difference in the WER can be explained by the lack of the language model. Different from the previous study, Bluche[179] provided for the both with and without a language model. Even though the WER without the language model was 24.6%, using a language model lowered the WER by over 8%. Looking at the results, the proposed model has 17.4%-word error rate with a 1% difference with the best resulting model.

Looking at the results, it is clear that post-processing step has a positive effect on the performance of the recognizer therefore it should be adopted in the system. Based on the results of the evaluation, it can be said that a segmentation-based approach is ideal for a non-cursive single language handwriting however when it comes to cursive and multilingual handwritings, holistic (segmentation free) approaches might yield better results. Additionally, testing the model on a public dataset, performance of the model is not too far from the state-of-the-art results. The next chapter puts forward an object detection-based handwriting recognition system.

7. OBJECT DETECTION BASED OFFLINE HANDWRITING RECOGNIZER

7.1. Chapter Overview

In this chapter, an object detection algorithm-based segmentation free offline handwriting recognition system is presented. Firstly, two main object detection methods are explained. Later, the structure of the recognizer is given. Finally, experiments and the results are put forward.

7.2. Object Detection

Object detection refers to two different computer vision tasks carried out together namely image classification and object localisation. Image classification task is used to put a class label on the object in the image and object localisation is used to locate the object or multiple objects in an image and put a bounding box around the object.

Popular methods for real-time object detection include methods based on Regions with CNN features (R-CNN) and You Look Only Once (YOLO) [180]–[185]. As mentioned in the previous chapter, object detection approach is also used in the handwriting recognition domain. The majority of the studies focus on R-CNN based methods. Therefore, in this chapter, we propose a YOLOv5s based handwriting recognition system. In this chapter, the R-CNN and YOLO based object detection is explained followed by the proposed model.

R-CNN Based Approach

Regions with CNN Features or Region-Based Convolutional Neural Network was proposed by Girshick et al. in 2014[180]. The model consisted of three modules namely, region proposal, feature extraction and classifier. The region of interest are determined in the region proposal step, using a selective search algorithm[186]. Approximately 2000 regions that might include an object are put into rectangular boxes in the first module. These regions form the input of the next module which is feature extraction. A large CNN extracts a fixed length feature vector (4096) from each region. Lastly, the feature vector is fed into a set of class- specific linear SVMs.

The downsides of R-CNNs derive from the fact that it is time consuming since approximately 2000 regions should be classified for each image. Thus, in 2015 a Fast R-CNN was proposed Girshick[181]. In this version, the region proposal step using selective search algorithm was left out. Instead, the feature maps are created by feeding the image

to the CNN. In order to make sure the size of the feature map is fixed; a Region of Interest (ROI) pooling layer is adopted. Finally, the fix sized vector is given to the fully connected layer as an input and two outputs are generated using two separate layers namely softmax and bounding box regressor. Class of the proposed region is generated as a result of the softmax layer and offset values are determined by the bounding box regressor. This version proves to be faster than the R-CNNs, however the need to calculate possible regions for each image is still time consuming. Later in 2016, a Faster R-CCN (FRCNN) was proposed by Ren et al.[182]. In this version, instead of the selected search algorithm a Region Proposal Network (RPN) was put forward. RPN is used for proposing the regions and the type of the object to be found in the region based on pre-defined shapes called anchor boxes. Then, the Fast R-CNN is used to extract the features and finally give the two outputs namely class label and bounding box for the proposed region.

YOLO Based Approach

Other popular approach in object detection is YOLO which was put forward by Redmond et al. in 2016. The researcher Ross Girshick who was one of the mail contributors and authors in the development of R-CNN is also a contributor and author in the development of YOLO. In the first version, class probabilities and the location of the objects in bounding boxes are calculated with a single neural network architecture using the full image[187]. After application of the original image as a whole, the bounding boxes are predicted, and each box includes 5 predictions. The first two are the *(x, y)* coordinates of the centre of the box and the following two are *w* and *h* stand for the width and height of the box and lastly the confidence is the last prediction[188]. YOLO being a single stage object detector, consists of three components namely backbone, neck, and head. Backbone is a CNN which usually works as the feature extractor, the features prepared in the neck for the detection by combining and mixing those and finally the head generates the bounding boxes and predictions[189]. Commonly used backbones include Residual Neural Networks (ResNet)[190], Densely Connected Networks (DenseNet)[191] and Very Deep Convolutional Networks (VGG)[111] which are pre-trained on image classification datasets and then fine-tuned on the detection dataset. Popular models for the neck are Path Aggregation Network (PaNet) [192] and Feature Pyramid Networks (FPN) [193] helping to extract different feature maps at different stages. As for the head responsible for classification and regression, YOLO[183], Region

Proposal Networks (RPN)[182] and RetinaNet[193] are amongst the widely adopted models.

The popularity of the model derives from the fact that it is very fast, thus allowing real-time object detection since the calculation does not take as much time as it takes using R-CNN based methods. Later in 2017, Redmond and Farhadi proposed the YOLOv2 also known as also YOLO 9000[194]. The name YOLO9000 is used because this version is capable of detecting over 9000 object categories. In addition to the improvements in the number of object categories, other improvements such as addition of batch normalization and working with high-resolution input images are also made. Thereafter in 2018, Redmond and Farhadi announced the next version, the YOLOv3[184]. This version predict boxes at 3 different scales thus allowing better detection for small objects compare to the prior versions[195]. In 2020, Bochkovskiy et al. Proposed the YOLOv4 using CSPDarknet53[196] as the backbone, SpineNet(SPP)[197] and PaNet[192] for the neck and YOLOv3[184] as the head. This version outperformed the older one by 10% in accuracy and 12% in speed. The latest version YOLOv5 was proposed approximately a month after the previous version and it is still being developed and new packages and additions are added on regular basis. The latest release is v3.1, published on 29th of October 2020[185]. In this version, Cross Stage Partial Networks(CSPNet) are adopted as the backbone[196] as in the previous version. As for the neck and head similarly to the previous version, PaNet and YOLOv3 were used. In terms of performance of networks, YOLOv4 and YOLOv5 perform similarly. However, YOLOv5 appears to be easier to apply and faster to train whereas YOLOv4 trains more slowly and in the long run gives better accuracy.

7.3. Literature Review

In terms of offline handwriting recognition using versions of YOLO is still not common. In this aspect, this work proposes one of the first offline handwriting recognizers using YOLO based models. The reason behind the lack of research in this field can be the difficulty of the training of the model. There are datasets containing objects such as bicycle and car with proper labels for YOLO models however there are no datasets available for handwriting recognition purpose. In terms of offline handwriting recognition, Santoso et al. adopted YOLOv3 object detection to recognize Kawi character on copper inscriptions in 2020[198]. Although, the input in the study is a historical copper

inscription, the main difference with our study is the nature of handwriting. The Kawi characters on the inscription are spaced and discretely written unlike the handwritings in our study. The results showed that their model had an average accuracy of 97.93% in recognition of Kawi characters. There are currently no other studies conducted on offline handwriting recognition however there are three studies conducted on handwritten digit recognition using YOLO based approaches. It should be noted that in digit recognition, the domain is generally discretely written, and the number of classes are only 10 unlike in handwritten characters. Therefore, it is possible to say that digit recognition is less challenging that character recognition. In 2020, Sun et al. proposed a YOLO based handwritten steel billet identification number recognizer[199]. This study is similar to the previous one in terms of the surface the handwriting was performed on. For training YOLO object detector, 120 images and as for testing 99 images were used. The results indicated that recognition of every number had at least 97% accuracy rate. In 2021, Kusetogullari et al. proposed a new historical handwritten digit database called DIDA and a novel deep learning architecture, named DIGITNET[200]. In their study, they also adopted the YOLOv3 and YOLOv3- tiny as a base of comparison for their proposed recognizer. Their results showed that YOLOv3 had a 48.56% and YOLOv3- tiny had 64.48% and the proposed DIGITNET had 70.15% accuracy rate on a 200000-image dataset. Lastly, Hochuli et al. put forward a comprehensive study on handwritten digit recognition[201]. The proposed study included five different datasets with different difficulty levels and recognition of those included several segmentation-based and segmentation-free strategies including YOLO. The results showed that, YOLO outperformed other three segmentation free approaches on three datasets out of five. In the remaining two datasets, the difference in the performance of YOLO and the best performing method was very small.

The following sections provide information how the data was prepared to train the object detector and how the post processing was performed followed by the experiments and the evaluation.

7.4. Experiments

Looking at the recent success in object detection and the performance of YOLO based object detection systems was an encouragement to try this approach on the handwriting recognition problem. If we consider a handwritten page or simply a

handwritten word/ character, the two components for object detection problem namely object location and classification of the object are present. However, it should be noted that in object detection, the difference between classes is more obvious unlike in the case of characters. As for the object detector, based on its ease of implementation and speed, YOLOv5s model was used for handwriting recognition. The modifications made in the architecture and the input are explained below.

7.4.1. Data-Processing

Unlike the characters represented in T-H-E Dataset, the input to YOLOv5 is a large image preferably containing more than one character at different locations over the image. Therefore, the 28×28 pixel binary images from T-H-E Dataset were used to create suitable input for the object detection algorithm using some of the pre-processing steps from the GitHub repository of YOLOv3 implementation [202]. Firstly, the empty space meaning black pixels from right, left, bottom and top of the character are removed. Then, the character to be placed onto the 416×416-pixel space is resized by multiplying its size by one of the 7 fix values (0.5, 0.8, 1, 1.5, 2, 3, 4) picked randomly. Then the character is randomly placed onto the space. Every character has its label and its location saved in the format below and every line represents one character:

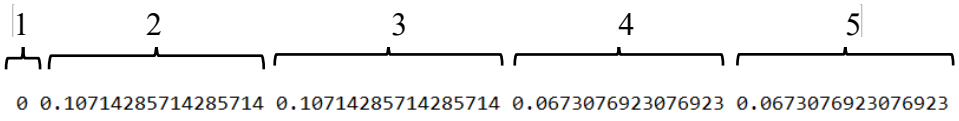


Figure 51 *Representation of the Location and the Label of the Characters*

The first number refers to the class label of the image, the second number refers to the location of the centre of the character on the *x* axis and the third number refers to the *y* axis of the centre. The fourth number is the width of the character divided by the width of the image (416) and the last one is the height of the character divided by the height of the image (416). This particular example in Fig. 51, represents a 28×28-pixel sized letter "a" which is placed on the top-left corner of a 416×416 image, this the class label is "0" referring to character "a" and the last two numbers are the width and height 28 divided into 416 (28/416=0.0673076923076923).

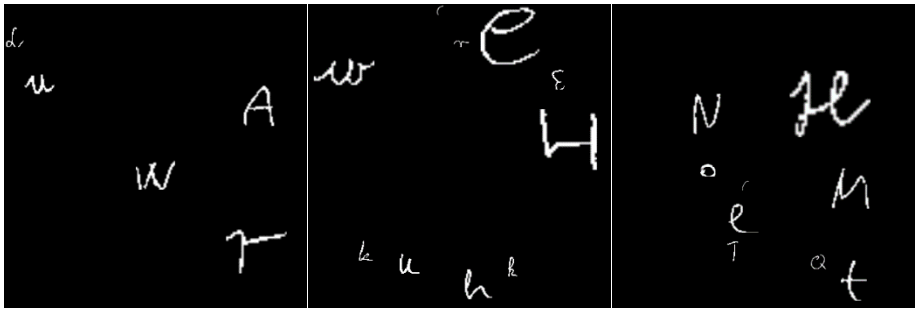


Figure 52 *Data Prepared for the Training*

There is no fix number defining the number of characters to be placed onto the space. The process of placing characters onto the 416×416-pixel space continues until one character is placed onto another exceeding the threshold which on average results in having 4 to 10 characters onto the space. The threshold is set to 0.02 intersection over union (IoU). Three examples of training data can be seen in the Fig. 52 above.

The input images are prepared by the above-mentioned steps. Then using these images, the network is trained.

7.4.2. Post-Processing

The post-processing phase aims at excluding misclassifications generated due to the flaws in the recognition system. In this section, the post-processor in the previous chapter is used with minimal changes.

The output of the object detection phase is fed to the postprocessor word by word. An example can be seen in the Fig. 53 below. Firstly, the letters in the output of the recognition step and every word in the dictionary is converted to lower-case characters in order to make sure that the distance is calculated with minimal errors. Finally, the lower-case form of the words is fed to the post-processor. Like in the previous chapter, the Levenshtein distance is calculated for the recognised words and every single word in the dictionary and the minimum distanced match is given as the output[177]. The same dictionaries as in the previous chapter are adopted in this section.

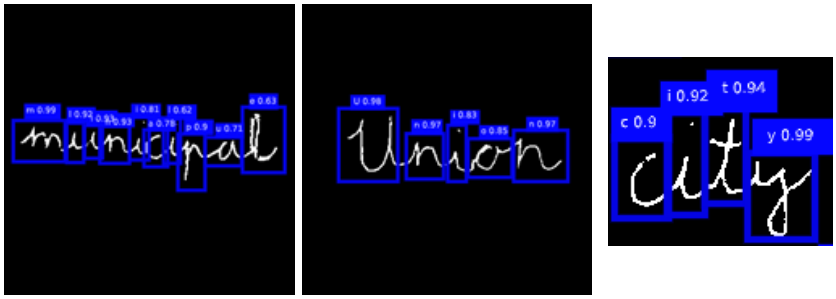


Figure 53 *Sample Results of Object Detection*

In the final step, the matching word is compared with the original dictionary and it is normalised into its form in the dictionary to be able to maintain the capital letters in certain words.

7.4.3. Experiments

The experiments were carried out using Google Colab**.** YOLOv5s repository in GitHub was used with certain changes for such input[203]. Firstly, the training and test sets were determined as 90% training and 10% test (validation) set. Then, the anchor sizes were changed for the needs of our input as can be seen in the Fig. 54 below.

```
# anchors
anchors:
  - [10,10, 16,16, 16,28]  # P3/8
  - [28,16, 32,28, 32,32]  # P4/16
  - [64,32, 64,64, 128,128]  # P5/32
```

Figure 54 *Modified Anchor Sizes in YOLOv5*

The performance of the recognizer is going to be tested on Turkish, Hungarian and English languages separately and on a multilingual text containing all three languages. Therefore, the network was trained separately for every input. For Turkish handwriting recognition, the en_augmented and tr_augmented versions from the T-H-E Dataset were combined and the network was trained using a 64-character (52 English+ 12 Turkish) dataset. It should be noted that although the Turkish alphabet does not contain letter “x”, “w“ and “q” they were not excluded from the dataset. As for the Hungarian text, hu_augmented and en_augmented versions were combined forming a dataset with a class label of 70. Finally, for English and multilingual texts, en_augmented and entire_augmented versioned were used, respectively. The versions used can be seen in the Table 22 in the previous chapter. It should be noted that the synthetic characters generated in the Chapter 5 are also included in the training sets.

The trained network is going to be used to detect the characters given as a word onto a blank 416×416-pixel image. The words are segmented as mentioned in the previous chapter. Firstly, using MSER algorithm the lines are detected and then the words are segmented based on the number of black pixels in between. The segmented words then are resized to fit into 416×416 image keeping the aspect ratio of the word and each word is placed onto a 416×416-pixel blank image resulting in a word in each image. This process is carried out for all the words in the 80-page long input mentioned in the previous chapter (20 pages for each language).

7.4.4. Evaluations

Similarly to the previous chapter, the performance evaluation is carried out separately for Turkish, Hungarian, English, and multilingual texts. Prepared images containing every word in a 416×416-pixel format are used as the input for the recognition. Since the pages used to evaluate the recognizer do not belong to a dataset, they were not tagged with the actual words. Therefore, an expert created the labels of the pages by converting those into strings as mentioned in the previous chapter.

The word error rate (WER) for each language is calculated to measure the performance. The Table 26 below shows the WER and percentage of the misclassified words for each language.

Table 26 *Word Error Rate of Handwritten Texts in Different Languages*

	WER (%)	**Percentage of misclassified words**	**#words**
Turkish Text	6.58	5.55	4758
Hungarian Text	9.08	7.36	5012
English Text	5.93	4.35	4826
Multilingual Text	15.07	11.99	4062

Looking at the results, it can be seen that similarly to the previous recognizer, multilingual text detection gives the lowest accuracy. The reason behind the high number of misclassifications derives from the size of the dictionary being very large. Additionally, since the text containing all three languages is arbitrary, a context-based dictionary could not be used. Using a context-based dictionary would decrease the WER in our opinion. Additionally, multilingual text detection is carried out with a network

trained on 78 letters therefore, the high number in class label also contributes to the high error rate.

As for the single language recognition, recognition of English texts has the highest performance with only 5.93 % WER and 4.35 % of the words misclassified. The discrete nature in the English handwritings in the dataset and lower-class size (52 classes) have a positive effect on the classification performance. On the other hand, Hungarian text recognition has the highest error rate with over 9%. As mentioned in the evaluation of the previous chapter, the cursive nature of the language has a negative effect in terms of recognition performance. Additionally, having letters that are written in a similar way with an accent difference such as letter “u” - “ú” and “e” and “é” also makes the chances of misclassification higher. Increasing the number of training set might help with elimination of errors caused by the accents.

Looking at the results, it is possible to say that object detection-based recognition has a great potential at single language recognition. The error rates are at an acceptable level and they can further be decreased with a larger training set. Additionally, instead of training the network with single letters, training the network with words may yield better results.

8. CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK

8.1. Chapter Overview

In this chapter, a general conclusion of the thesis is put forward followed by future research directions.

8.2. Conclusions

Offline handwriting recognition is an unresolved research problem due to several factors such as use of diacritical marks, cursive nature of the handwriting, lack of handwritten datasets and input language. The input language plays a crucial role in the success of the recognition. For instance, it is possible to find very accurate handwriting recognition software for English language but there are not many that yields high performance in languages such as Turkish and Hungarian. In addition to single language handwriting recognition, with the number and the importance of multilingual handwritten text rising in the recent years as a result of globalisation, solutions for multilingual handwritten text recognition has come into prominence. This brought about a need for handwritten text or character datasets in several languages to develop multilingual handwriting recognition software.

In this study, a multilingual handwritten character dataset was created and published on public domain. The proposed dataset, T-H-E Dataset, includes handwritten characters in Turkish, Hungarian and English languages in six different versions as mentioned in the Chapter 4. Several experiments to evaluate the proposed dataset were carried out and the results show that in terms of handwritten English character datasets, the T-H-E Dataset can be a good alternative. In addition to being an alternative, it is the only handwritten Turkish and Hungarian handwritten character dataset in the field. The proposed dataset could be adopted for single language recognition purposes, namely, Turkish, Hungarian, or English character recognition systems, as well as multilingual recognition systems.

In addition to human-generated characters, synthetic characters based on the handwritten characters in the T-H-E Dataset were also generated using generative models namely DCGANs in the Chapter 5. The experiments on the dataset augmented with synthetic characters show that including the generated characters in the input improve the overall recognition accuracy.

In the past years, advancements in deep learning methods concomitantly led to improvements in offline HWR. In this study, using the handwritten and synthetic characters, two deep learning based offline handwriting recognition systems were proposed in the Chapter 7 and 8. The first system is a segmentation based offline handwriting recognizer using a CNN architecture which was proposed in the Chapter 3. The system consists of pre-processing, recognition, and post-processing steps. The proposed system works on Turkish, Hungarian, and English texts however the performance of the system highly depends on the style of handwriting since the system is segmentation based. The results indicated that the segmentation-based model yields good results in non-cursive single language recognition and poor results in cursive handwritings and multilingual texts. In order to overcome the drawback in the system, an object detection-based recognition is proposed in the Chapter 7. YOLOv5 algorithm is used to recognise handwritten texts. Firstly, the system is trained with the characters from the T-H-E Dataset and synthetic characters from Chapter 5 systematically scattered on a 416×416 blank image and then handwritten words are detected using the trained algorithm. The results indicate that, YOLOv5 outperforms the segmentation-based method proposed in the previous chapter in recognition of single languages as well as the multilingual texts. As for the cursive handwritings, only 8.36 % of the words are misclassified compare to 11.55 % in the segmentation-based recognizer. Similarly, the WER of the recognizer is 5.65 % lower than the previous recognizer. In multilingual text recognition, the proposed system performs a little lower than the ideal. Although, the character level recognition is at an ideal level, the dictionary being very large results in misclassifications in the word level. Since the multilingual text is artificially created and it does not have a real context, it is difficult to decrease the dictionary size. In a real-life sample, a context-based dictionary can be used to overcome the vast of the misclassified words.

It can be concluded that, the proposed handwritten character dataset in this study constitutes a big potential for multilingual handwriting recognition purpose as well as contributing to Turkish, Hungarian, and English character databases. Alongside with that, the proposed recognition systems get very close to the optimum recognition rates on handwritten single and multilingual texts. Additionally, it is hoped that the contribution of this thesis can be a start for further improvements and developments in the field

especially for Turkish and Hungarian handwriting recognition since there is still a lot to accomplish.

8.3. Future Works

Due to time constraints, some of the planned work could not be completed. The augmentation of the T-H-E Dataset was supposed to be completed. However, due to the COVID-19 pandemic the process was disturbed, and data collection for the dataset had to be postponed. In this section, the recommendations on further improving the T-H-E Dataset is presented.

Augmentation of the T-H-E Dataset

T-H-E Dataset consists of handwritten binary Turkish, Hungarian, and English characters as mentioned in the previous section. However, the dataset did not include any meta data regarding the user writing the characters. Therefore, the applicability of the set was limited to image classification applications. In order to extend the scope of the application of the dataset, some meta data about the user is going to be linked to the handwritten characters namely the age, gender, occupation, handiness (left or right-handed) and nationality of the users. Thereby, the T-H-E dataset could also be used for other research fields such as handwriting identification or statistics regarding the nationalities, handiness, occupations, and genders.

In addition to the meta data included in the dataset, the second version is going to include characters in grayscale unlike in the first version. Since, grayscale images provide more information than a binary image, it is believed to increase the performance of the classifiers as well.

Training the Object Detection Model with Handwritten Words

The object-detection based recognized in the Chapter 7 was trained with the handwritten characters scattered around a 416×416-pixel space. However, training the network with actual handwritten words with the tagged characters might result in better recognition performance. Since the handwritten characters lose some of its shape after segmentation. Due to time constraints, this part of the training could not be accomplished.

9. REFERENCES

[1] J. Pena, S. Letourneau, and F. Famili, “Application of Rough Sets Algorithms to Prediction of Aircraft Component Failure,” in *Advances in Intelligent Data Analys*, no. i, 1999, pp. 473–484.

[2] R. Plamondon and S. N. Srihari, “On-line and off-line handwriting recognition: A comprehensive survey,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 63–84, 2000.

[3] P. K. Charles, V. Harish, M. Swathi, and C. H. Deepthi, “A Review on the Various Techniques used for Optical Character Recognition,” *Int. J. Eng. Res. Appl.*, vol. 2, no. 1, pp. 659–662, 2012.

[4] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nat. Methods*, vol. 13, no. 1, p. 35, 2015.

[5] W. Xing and D. Du, “Dropout Prediction in MOOCs: Using Deep Learning for Personalized Intervention,” *J. Educ. Comput. Res.*, vol. 57, no. 3, pp. 547–570, 2019.

[6] C. C. Tappert, “Adaptive on-line handwriting recognition,” in *7th International Conference on Pattern Recognition*, 1984, pp. 1004–1007.

[7] M. Gumah, “Off-line Arabic Handwriting Recognition System Using Fast Wavelet Transform,” Universiti Teknologi Petronas, 2010.

[8] D. Diringer and D. R. Olson, “Alphabet,” *Britannica*. 1998.

[9] A. Çapar, K. Taşdemir, Ö. Kılıç, and M. Gökmen, “A Turkish Handprint character recognition system,” in *Computer and Information Sciences*, 2003.

[10] B. A. Yanikoglu and A. Kholmatov, “Turkish handwritten text recognition: a case of agglutinative languages,” *Doc. Recognit. Retr.*, pp. 227–233, 2003.

[11] E. Vural, H. Erdogan, O. Oflazer, and B. Yanikoglu, “An online handwriting recognition system for Turkish,” *Proc. IEEE 12th Signal Process. Commun. Appl. Conf.*, pp. 607–610, 2004.

[12] M. Şekerci and R. Kandemir, “Sözlük Kullanarak Türkçe El yazısı Tanıma,” *Elektr. Bilgi. Sempozyum (ELECO 2006)*, pp. 2–6, 2006.

[13] M. Şekerci, “Birleşik ve eğik Türkçe el yazısı tanıma sistemi,” Trakya University, 2007.

[14] M. S. Aydemir, B. Aydın, H. Kaya, İ. Karlıağa, and C. Demir, “Tübitak Türkçe - Osmanlica El Yazisi Tanima Sistemi Tübitak Turkish - Ottoman Handwritten Recognition System,” 2014, pp. 1918–1921.

[15] H. Tanaka, K. Nakajima, K. Ishigaki, K. Akiyama, and M. Nakagawa, “Hybrid pen-input character recognition system based on integration\nof online-offline recognition,” *Proc. Fifth Int. Conf. Doc. Anal. Recognition. ICDAR ’99 (Cat. No.PR00318)*, vol. 1, no. c, pp. 1–4, 1999.

[16] A. Graves and J. Schmidhuber, “Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks,” *Adv. Neural Inf. Process. Syst.*, pp. 545–552, 2008.

[17] C. Bahlmann, B. Haasdonk, and H. Burkhardt, “Online handwriting recognition with support vector machines - A kernel approach,” *Proc. - Int. Work. Front. Handwrit. Recognition, IWFHR*, pp. 49–54, 2002.

[18] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, “A novel connectionist system for unconstrained handwriting recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 5, pp. 855–868, 2009.

[19] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep, big, simple neural nets for handwritten digit recognition,” *Neural Comput.*, vol. 22, no. 12, pp. 3207–3220, 2010.

[20] N. Arica and F. T. Yarman-Vural, “Optical character recognition for cursive handwriting,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 6, pp. 801–813, 2002.

[21] N. Otsu, “A Threshold Selection Method from Gray-Level Histograms,” *IEEE Trans. Syst. Man, Cybern.*, vol. 9, pp. 62–66, 1979.

[22] P. K. Sahoo, S. Soltani, and A. K. C. Wong, “A Survey of Thresholding Techniques*,” *Comput. Vision, Graph. Image Process.*, vol. 41, no. 2, pp. 233–260, 1988.

[23] N. Priyanka, S. Pal, and R. Mandal, “Line and Word Segmentation Approach for Printed Documents,” in *IJCA Special Issue on Recent Trends in Image Processing and Pattern Recognition-RTIPPR*, 2010, pp. 30–36.

[24] B. Mağden and S. Telçeken, “Probabilistic Rough Sets in Turkish Optical Character Recognition,” in *6th World Conference on Soft Computing*, 2016, no. 3, pp. 170–173.

[25] R. K. Nath and M. Rastogi, “Improving Various Off-line Techniques used for Handwritten Character Recognition : a Review,” *Int. J. Comput. Appl.*, vol. 49, no. 18, pp. 11–17, 2012.

[26] R. M. Bozinovic and S. N. Srihari, “Off-line cursive word recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 1, pp. 68–83, 1989.

[27] R. J. Rodrigues, A. Carlos, and G. Thom, “Cursive character recognition – a character segmentation method using projection profile-based technique,” in *The 4th World Multiconference on Systemics, Cybernetics and Informatics SCI 2000 and The 6th International Conference on Information Systems, Analysis and Synthesis ISAS*, 2000.

[28] S. Mo and J. Mathews, “Adaptive, quadratic preprocessing of document images for binarization,” *IEEE Trans. image Process.*, vol. 7, no. 7, pp. 992–999, 1998.

[29] L. Eikvil, “OCR — Optical Character Recognition,” no. December, 1993.

[30] B. Verma and M. Blumenstein, “Fusion of Segmentation Strategies for Off-Line Cursive Handwriting Recognition,” in *Pattern Recognition Technologies and Applications: Recent Advances*, New York: Information Science Reference, 2008, pp. 1–16.

[31] A. R. Khan and M. Zulkifli, “A Simple Segmentation Approach for Unconstrained Cursive Handwritten Words in Conjunction with the Neural Network,” *Int. J. Image Process.*, vol. 2, no. 3, pp. 29–35, 2008.

[32] M. Eltay, A. Zidouri, and I. Ahmad, “Exploring Deep Learning Approaches to Recognize Handwritten Arabic Texts,” *IEEE Access*, vol. 8, pp. 89882–89898, 2020.

[33] A. Choudhary, “A Review of Various Character Segmentation Techniques,” vol. 4, no. 6, pp. 559–564, 2014.

[34] G. Chopra, Salin ; Ghadge, Amit; Padwal , Onkar;Punjabi , Karan;Gurjar, “Optical Character Recognition, 2014,” *Citeseer. Ist. Psu. Edu/142042. Html*, vol. 4, no. 5, p. 13, 1982.

[35] N. Arica and F. T. Yarman-Vural, “An overview of character recognition focused on offline handwriting,” *IEEE Trans. Syst. Man Cybern. Part C (Applications Rev.*, vol. 31, no. 2, pp. 216–233, 2001.

[36] M. Faraggi and K. Sayadi, “Time series features extraction using Fourier and Wavelet transforms on ECG data,” *OCTO Technology*, 2019. [Online]. Available: https://blog.octo.com/en/time-series-features-extraction-using-fourier-and-wavelet-transforms-on-ecg-data/. [Accessed: 23-Oct-2020].

[37] M. S. M. El-Mahallawy, “A Large Scale HMM-based Omni Font-Written OCR System for Cursive Scripts,” no. April, p. 150, 2008.

[38] R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures.,” *Commun. ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[39] Y. Hamamoto, S. Uchimura, K. Masamizu, and S. Tomita, “Recognition of handprinted Chinese characters using Gabor features,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1995, pp. 819–823.

[40] S.-W. Lee and Y.-J. Kim, “Multiresolution recognition of handwritten numerals with wavelet transform and multilayer cluster neural network,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1995, pp. 1010–1013.

[41] A. G. Karegowda, A. S. Manjunath, and M. A. Jayaram, “Comparative Study of Attribute Selection Using Gain Ratio and Correlation Based Feature Selection,” *Int. J. Inf. Technol. Knowl. Manag.*, vol. 2, no. 2, pp. 271–277, 2010.

[42] S. Ben Driss, M. Soua, R. Kachouri, and M. Akil, “A comparison study between MLP and convolutional neural network models for character recognition,” *Real-*

Time Image Video Process. 2017, vol. 10223, p. 1022306, 2017.

[43] S. España-Boquera, M. J. Castro-Bleda, J. Gorbe-Moya, and F. Zamora-Martinez, "Improving offline handwritten text recognition with hybrid HMM/ANN models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 4, pp. 767–779, 2011.

[44] C. K. Dewa, A. L. Fadhilah, and A. Afiahayati, "Convolutional Neural Networks for Handwritten Javanese Character Recognition," *IJCCS (Indonesian J. Comput. Cybern. Syst.*, vol. 12, no. 1, p. 83, 2018.

[45] R. M. O. Cruz, G. D. C. Cavalcanti, and T. I. Ren, "An Ensemble Classifier For Offline Cursive Character Recognition Using Multiple Feature Extraction Techniques," *2010 Int. Jt. Conf. Neural Networks*, no. July 2015, pp. 1–8, 2010.

[46] G. Singh and M. Sachan, "Multi-layer perceptron (MLP) neural network technique for offline handwritten Gurmukhi character recognition," *2014 IEEE Int. Conf. Comput. Intell. Comput. Res. IEEE ICCIC 2014*, pp. 1–5, 2015.

[47] A. Dhomne, R. Kumar, and V. Bhan, "Gender Recognition Through Face Using Deep Learning," *Procedia Comput. Sci.*, vol. 132, pp. 2–10, 2018.

[48] J. Djuriš, D. Medarević, M. Krstić, I. Vasiljević, I. Mašić, and S. Ibrić, "Design space approach in optimization of fluid bed granulation and tablets compression process," *Sci. World J.*, vol. 2012, no. May 2014, 2012.

[49] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 130–136, 1997.

[50] H. Jee, K. Lee, and S. Pan, "Eye and face detection using SVM," *Proc. 2004 Intell. Sensors, Sens. Networks Inf. Process. Conf. ISSNIP '04*, pp. 577–580, 2004.

[51] G. M. Foody and A. Mathur, "The use of small training sets containing mixed pixels for accurate hard image classification: Training on mixed spectral responses for classification by a SVM," *Remote Sens. Environ.*, vol. 103, no. 2, pp. 179–189, 2006.

[52] O. Uslu and S. Akyol, "Türkçe Haber Metinlerinin Makine Öğrenmesi Yöntemleri Kullanılarak Sınıflandırılması Türkçe Haber Metinlerinin Makine Öğrenmesi

Yöntemleri Kullanılarak Sınıflandırılması," *J. Estud. Inf.*, no. January, 2021.

[53] A. Alalshekmubarak, A. Hussain, and Q. F. Wang, "Off-line handwritten Arabic word recognition using SVMs with normalized poly kernel," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7664 LNCS, no. PART 2, pp. 85–91, 2012.

[54] G. Katiyar, "Off-Line Handwritten Character Recognition System Using Support Vector Machine," *Am. J. Neural Networks Appl.*, vol. 3, no. 2, p. 22, 2017.

[55] M. Kumar, M. K. Jindal, and R. K. Sharma, "SVM based offline handwritten Gurmukhi character recognition," *CEUR Workshop Proc.*, vol. 758, no. November 2014, pp. 51–62, 2011.

[56] J. Dasgupta, K. Bhattacharya, and B. Chanda, "A holistic approach for Off-line handwritten cursive word recognition using directional feature based on Arnold transform," *Pattern Recognit. Lett.*, vol. 79, pp. 73–79, 2016.

[57] M. Elleuch, R. Maalej, and M. Kherallah, "A New design based-SVM of the CNN classifier architecture with dropout for offline Arabic handwritten recognition," *Procedia Comput. Sci.*, vol. 80, pp. 1712–1723, 2016.

[58] X. X. Nui and C. Y. Suen, "A novel hybrid CNN–SVM classifier for recognizing handwritten digits," *Pattern Recognit.*, vol. 45, no. 4, pp. 1318–1325, 2012.

[59] G. Ediboğlu Bartos and É. Hajnal, "Optical Character Recognition for Turkish - a Survey," in *Vasil Levski National Military University Annual Scientific Conference 2017*, 2017.

[60] W. Czajewski, "Rough Sets in Optical Character Recognition," in *Rough Sets and Current Trends in Computing 98*, 1998, pp. 601–604.

[61] Z. Pawlak and A. Skowron, "Rudiments of rough sets," *Inf. Sci. (Ny).*, vol. 177, no. 1, pp. 3–27, 2007.

[62] J. Tian, W. Liu, and H. Wang, "Offline handwritten Chinese character recognition using genetic algorithm and rough set," in *2014 IEEE 7th Joint International Information Technology and Artificial Intelligence Conference, ITAIC*, 2015, pp. 74–78.

[63] K. Jayech, M. A. Mahjoub, and N. Ghanmi, “Application of Bayesian Networks for Pattern Recognition : Character Recognition Case,” in *International Conference on Sciences of Electronics, Technologies of Information and Telecommunications*, 2012, vol. 3, no. March, pp. 748–757.

[64] K. M. Lakshmi, K. Venkatesh, G. Sunaina, D. Sravani, and P. Dayakar, “Hand Written Telugu Character Recognition Using Bayesian Classifier,” *Int. J. Eng. Technol.*, vol. 9, no. 3S, pp. 37–42, 2017.

[65] H. Wehle, “Machine Learning, Deep Learning, and AI: What’s the Difference?,” in *Data Scientist Innovation Day*, 2017, no. July.

[66] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep Learning for Computer Vision: A Brief Review,” *Comput. Intell. Neurosci.*, vol. 2018, pp. 1–13, 2018.

[67] A. Şeker, B. Diri, and H. H. Balık, “Derin Öğrenme Yöntemleri ve Uygulamaları Hakkında Bir İnceleme,” *Gazi Mühendislik Bilim. Derg.*, vol. 3, no. 3, pp. 47–64, 2017.

[68] W. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity.,” *Bull. ofMathematical Biophys.*, no. 7, pp. 115–133., 1943.

[69] J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015.

[70] H. J. Kelley, “Gradient Theory of Optimal Flight Paths,” *ARS J.*, vol. 30, no. 10, pp. 947–954, 1960.

[71] R. E. Bellman and S. E. Dreyfus, “Applied Dynamic Programming.pdf,” 1962.

[72] A. G. Ivakhnenko and V. G. Lapa, *Cybernetic Predicting Devices*. New York: CCM Information Corporation, 1965.

[73] H. Wang and B. Raj, “On the Origin of Deep Learning,” pp. 1–72, 2017.

[74] K. D. Foote, “A Brief History of Deep Learning,” 2017. [Online]. Available: https://www.dataversity.net/brief-history-deep-learning/. [Accessed: 04-Jun-2019].

[75] K. Fukushima, “Neural network model for a mechanism of pattern recognition unaffected by shift in position Neocognitron.,” *Trans. IECE*, vol. 62, no. 10, pp. 658–665, 1979.

[76] S. Linnainmaa, “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors.,” University of Helsinki, 1970.

[77] C. Cortes and V. Vapnik, “Support-Vector Networks,” *Mach. Learn.*, vol. 20, pp. 273–297, 1995.

[78] G. E. Hinton, S. Osindero, and Y. W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[79] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[80] R. Salakhutdinov and G. E. Hinton, “Deep Boltzmann Machines,” *Int. Conf. Artif. Intell. Statics*, no. 3, pp. 448–455, 2009.

[81] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Convolutional neural network committees for handwritten character classification,” *Proc. Int. Conf. Doc. Anal. Recognition, ICDAR*, vol. 10, pp. 1135–1139, 2011.

[82] D. Cireşan, U. Meier, and J. Schmidhuber, “Multi-column Deep Neural Networks for Image Classification,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2012.

[83] G. Cohen, S. Afshar, J. Tapson, and van A. Schalik, “EMNIST : an extension of MNIST to handwritten letters,” 2017.

[84] P. J. Grother and K. K. Hanaoka, “NIST Special Database 19,” pp. 1–30, 2016.

[85] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” pp. 1–18, 2012.

[86] A. Beam, “Deep Learning 101 - Part 1: History and Background,” 2017. [Online]. Available:

https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html. [Accessed: 04-Jun-2019].

[87] MathWorks, “Convolutional Neural Network 3 things you need to know.” [Online]. Available: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html?s_tid=srchtitle. [Accessed: 04-Jun-2019].

[88] L. Lab, “Convolutional Neural Networks (LeNet),” 2018. [Online]. Available: http://deeplearning.net/tutorial/lenet.html.

[89] C. Baskin, N. Liss, E. Zheltonozhskii, A. M. Bronstein, and A. Mendelson, “Streaming architecture for large-scale quantized neural networks on an FPGA-based dataflow platform,” *Proc. - 2018 IEEE 32nd Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2018*, no. September, pp. 162–169, 2018.

[90] R. Haridas, “Convolutional Neural Networks : A Comprehensive Survey,” vol. 14, no. 3, pp. 780–789, 2019.

[91] X. Zhang, C. Xv, M. Shen, X. He, and W. Du, “Survey of Convolutional Neural Network,” 2018.

[92] X. Jin, P. Cheng, W. L. Chen, and H. Li, “Prediction model of velocity field around circular cylinder over various Reynolds numbers by fusion convolutional neural networks based on pressure on the cylinder,” *Phys. Fluids*, vol. 30, no. 4, 2018.

[93] D. Quang and X. Xie, “DanQ: A hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences,” *Nucleic Acids Res.*, vol. 44, no. 11, pp. 1–6, 2016.

[94] A. M. and G. H. Alex Graves, “Speech Recognition with Deep Recurrent Neural Networks , Department of Computer Science, University of Toronto,” *Dep. Comput. Sci. Univ. Toronto*, vol. 3, no. 3, pp. 45–49, 2013.

[95] Ş. Işık, Z. Kurt, Y. Anagün, and K. Özkan, “Recurrent Neural Networks for Spam E-mail Classification on an Agglutinative Language,” *Int. J. Intell. Syst. Appl. Eng.*, vol. 8, no. 4, pp. 221–227, 2020.

[96] Y. J. Yoo, “Hyperparameter optimization of deep neural network using univariate

dynamic encoding algorithm for searchesNo Title," *Knowledge-Based Syst.*, vol. 178, 2019.

[97] The MathWorks Inc, "MATLAB and Statistics Toolbox Release 2017b." The MathWorks, Inc., Natick, Massachusetts, United States, 2017.

[98] K. Y. Wong, R. G. Casey, and F. M. Wahl, "Document analysis system," *IBM J. Res. Dev.*, vol. 26, no. 6, pp. 647-656., 1982.

[99] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognit.*, vol. 13, no. 2, pp. 111-122., 1981.

[100] M. H. Glauberman, "Character recognition for business machines," *Character Recognit. Bus. Mach. Electron.*, vol. 29, no. 2, pp. 132–136, 1956.

[101] R. Kasturi, *Image analysis applications.*, 24th ed. CRC Press, 1990.

[102] O. D. Trier, A. K. Jain, and T. Taxt, "Feature extraction methods for character recognition - A survey," *Pattern Recognit.*, vol. 29, no. 4, pp. 641–662, 1996.

[103] C. Viard-gaudin and É. Caillault, "Using Segmentation Constraints in an Implicit Segmentation Scheme for On-line Word Recognition," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[104] N. Xue, "Chinese word segmentation as character tagging," *Comput. Linguist. Chinese Lang. Process.*, vol. 8, no. 1, pp. 29–48, 2003.

[105] D. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. MIT Press, 2001.

[106] "Confusion Matrix." [Online]. Available: http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html. [Accessed: 07-Jan-2015].

[107] Y. LeCun *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Comput.*, vol. 1, pp. 541–551, 1989.

[108] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffiner, "Gradient based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 86(11):2278-2324, 1998.

[109] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep

Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, vol. 1, no. 4, 2012.

[110] C. Szegedy *et al.*, "Going deeper with convolutions," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 1–9, 2015.

[111] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations*, 2015, pp. 1–14.

[112] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research," *IEEE SIGNAL PROCESSING MAGAZINE*, no. November, pp. 141–142, 2012.

[113] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," in *Seventh International Conference on Document Analysis and Recognition (ICDAR 2003)*, 2003.

[114] M. A. Ranzato, F. Huang, Y. Boureau, and Y. Lecun, "Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition," in *CVPR*, 2007.

[115] L. Fei-Fei, R. Fergus, and P. Perona, "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories," *CVPR Work.*, vol. 5, no. 7, 2004.

[116] C. Wu, W. Fan, Y. He, J. Sun, and S. Naoi, "Cascaded Heterogeneous Convolutional Neural Networks for Handwritten Digit Recognition," *Int. Conf. Pattern Recognit.*, no. Icpr, pp. 657–660, 2012.

[117] L. Chen, S. Wang, W. Fan, J. Sun, and S. NAoi, "Beyond human recognition: A CNN-based framework for handwritten character recognition," in *3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015.

[118] N. Mboga, C. Persello, J. R. Bergado, and A. Stein, "Detection of informal settlements from VHR images using convolutional neural networks," *Remote Sens.*, vol. 9, no. 11, 2017.

[119] Google, "What is Colaboratory." 2017.

[120] Flow2k, “What is batch size in neural network?” [Online]. Available: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network.

[121] S. Ramesh, “A guide to an efficient way to build neural network architectures,” 2018. [Online]. Available: https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7.

[122] B. A. A. and S. Amidi, “Convolutional Neural Networks cheatsheet,” *Stanford University*. [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks.

[123] M. Şekerci, “Birleşik ve Eğik Türkçe El Yazısı Tanıma Sistemi,” Trakya University, 2007.

[124] U. V. Marti and H. Bunke, “The IAM-database: An English sentence database for offline handwriting recognition,” *Int. J. Doc. Anal. Recognit.*, vol. 5, no. 1, pp. 39–46, 2003.

[125] “Applied Media Analysis,” *Arabic-Handwritten-1.0.*, 2007. [Online]. Available: http://appliedmediaanalysis.com/Datasets.htm.

[126] R. Jayadevan, S. R. Kolhe, P. M. Patil, and U. Pal, “Database Development and Recognition of Handwritten Devanagari Legal Amount Words,” in *2011 International Conference on Document Analysis and Recognition*, 2011, pp. 304–308.

[127] S. Wshah, G. Kumar, and V. Govindaraju, “Multilingual word spotting in offline handwritten documents,” in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, 2012, pp. 310–313.

[128] M. Kozielski, P. Doetsch, M. Hamdani, and H. Ney, “Multilingual off-line handwriting recognition in real-world images,” *Proc. - 11th IAPR Int. Work. Doc. Anal. Syst. DAS 2014*, pp. 121–125, 2014.

[129] S. Brunessaux *et al.*, “The Maurdor Project: Improving Automatic Processing of Digital Documents,” in *2014 11th IAPR International Workshop on Document*

Analysis Systems, 2014, pp. 349–354.

[130] T. Bluche and R. Messina, “Gated Convolutional Recurrent Neural Networks for Multilingual Handwriting Recognition,” in *14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, 2017, pp. 646–651.

[131] E. Augustin, J. Brodin, M. Carré, E. Geoffrois, E. Grosicki, and F. Prêteux, “RIMES Evaluation Campaign for Handwritten Mail Processing,” *Work. Front. Handwrit. Recognit.*, no. 1, pp. 1–5, 2006.

[132] W. Swaileh, Y. Soullard, and T. Paquet, “A Unified Multilingual Handwriting Recognition System using multigrams sub-lexical units,” *Pattern Recognit. Lett.*, vol. 121, pp. 68–76, 2019.

[133] J. J. Hull, “A Database for Handwritten Text Recognition Research,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 5, pp. 550–554, 1994.

[134] M. Thoma, “The HASYv2 dataset,” pp. 1–8, 2017.

[135] F. Camastra, M. Spinetti, and A. Vinciarelli, “Offline Cursive Character Challenge : a New Benchmark for Machine Learning and Pattern Recognition Algorithms .,” *18th Int. Conf. Pattern Recognit.*, vol. 2, pp. 913–916, 2006.

[136] G. Ediboğlu Bartos, É. Hajnal, and Y. Hoşcan, “Performance Analysis of Character case-sensitive and case-insensitive Classification in Handwritten Character Recognition,” in *13th International Symposium on Applied Informatics and Related Areas (AIS 2018)*, 2018.

[137] G. Ediboglu Bartos and É. Hajnal, “Pre-processing Techniques for Hungarian Handwriting Recognition,” in *11th International Symposium on Applied Informatics and Related Areas (AIS 2016)*, Budapest: Óbudai Egyetem, 2016, pp. 94–99.

[138] C. Wigington *et al.*, “Data Augmentation for Recognition of Handwritten Words and Lines Using a CNN-LSTM Network,” in *14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, 2017, pp. 639–645.

[139] K. Dutta, P. Krishnan, M. Mathew, and C. V. Jawahar, “Improving CNN-RNN hybrid networks for handwriting recognition,” *Proc. Int. Conf. Front. Handwrit.*

Recognition, ICFHR, vol. 2018-Augus, pp. 80–85, 2018.

[140] MathWorks, "Create a Gallery of Transformed Images," 2019. [Online]. Available: https://www.mathworks.com/help/images/creating-a-gallery-of-transformed-images.html#GalleryTransformedImagesExample-7. [Accessed: 05-Jul-2019].

[141] de J. Vries, "barrel and pincushion lens distortion correction," *MathWorks*, 2012. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/37980-barrel-and-pincushion-lens-distortion-correction. [Accessed: 05-Jul-2019].

[142] N. O'Mahony *et al.*, "Deep Learning vs. Traditional Computer Vision," *Adv. Intell. Syst. Comput.*, vol. 943, pp. 128–144, 2019.

[143] Y. Lecun *et al.*, "Comparison of Learning Algorithms for Handwritten Digit Recognition," *Int. Conf. Artif. Neural Networks*, pp. 53–60, 1995.

[144] Y. LeCun *et al.*, "Learning Algorithms For Classification: A Comparison On Handwritten Digit Recognition," *Neural Networks Stat. Mech. Perspect.*, pp. 261–276, 1995.

[145] G. Ediboğlu Bartos, Y. Hoscan, A. Kauer, and É. Hajnal, "A Multilingual Handwritten Character Dataset: T-H-E Dataset," *Acta Polytech. Hungarica J. Appl. Sci.*, vol. 17, no. 9, 2020.

[146] I. J. Goodfellow *et al.*, "Generative Adversarial Networks," pp. 1–9, 2014.

[147] Google Developers, "Overview of GAN Structure." [Online]. Available: https://developers.google.com/machine-learning/gan/gan_structure. [Accessed: 08-Nov-2019].

[148] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," pp. 1–16, 2015.

[149] N. Shibuya, "Having Fun with Deep Convolutional GANs," 2017. [Online]. Available: https://medium.com/activating-robotic-minds/having-fun-with-deep-convolutional-gans-f4f8393686ed. [Accessed: 08-Nov-2019].

[150] G. ÇELİK and M. F. TALU, "Çekişmeli Üretken Ağ Modellerinin Görüntü

Üretme Performanslarının Incelenmesi," *Balıkesir Üniversitesi Fen Bilim. Enstitüsü Derg.*, vol. 22, no. 1, pp. 181–192, 2020.

[151] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. 2019.

[152] A. Borji, "Pros and cons of GAN evaluation measures," *Comput. Vis. Image Underst.*, vol. 179, pp. 41–65, 2019.

[153] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[154] Q. Xu *et al.*, "An empirical study on evaluation metrics of generative adversarial networks," 2018.

[155] K. Shmelkov *et al.*, "How good is my GAN ?," in *ECCV 2018 - European Conference on Computer Vision*, 2018, pp. 218–234.

[156] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, and S. Gelly, "Are Gans created equal? A large-scale study," *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, no. Nips, pp. 700–709, 2018.

[157] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, pp. 6627–6638, 2017.

[158] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *J. Mach. Learn. Res.*, vol. 13, pp. 723–773, 2012.

[159] J. Brownlee, "How to Evaluate Generative Adversarial Networks," *Machine Learning Mastery*, 2019. [Online]. Available: https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/. [Accessed: 21-Sep-2020].

[160] The Python Software Foundation, "Python." 2018.

[161] R. Gowda, "DCGAN Implementation in Keras explained," 2019. [Online]. Available: https://medium.com/@ramyahrgowda/dcgan-implementation-in-keras-explained-e1918fc930ea. [Accessed: 07-Nov-2019].

[162] Xuqiantong, “GAN-Metrics,” *GitHub*, 2019. [Online]. Available: https://github.com/xuqiantong/GAN-Metrics?fbclid=IwAR3CpFms44TikRJzKFLCbtXN0swgFGPIqR_aa2W2kgEn3uuynhQvfLLXT6U. [Accessed: 10-Oct-2020].

[163] F. Zamora-Martínez, V. Frinken, S. España-Boquera, M. J. Castro-Bleda, A. Fischer, and H. Bunke, “Neural network language models for off-line handwriting recognition,” *Pattern Recognit.*, vol. 47, no. 4, pp. 1642–1652, 2014.

[164] H. Schwenk, “Continuous space language models,” *Comput. Speech Lang.*, vol. 21, no. 3, pp. 492–518, 2007.

[165] L. Yan, “Recognizing Handwritten Characters,” 2016.

[166] P. Doetsch, A. Zeyer, and H. Ney, “Bidirectional decoder networks for attention-based end-to-end offline handwriting recognition,” *Proc. Int. Conf. Front. Handwrit. Recognition, ICFHR*, pp. 361–366, 2016.

[167] D. Bahdanau, K. H. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

[168] T. Bluche, J. Louradour, and R. Messina, “Scan, Attend and Read: End-To-End Handwritten Paragraph Recognition with MDLSTM Attention,” *Proc. Int. Conf. Doc. Anal. Recognition, ICDAR*, vol. 1, pp. 1050–1055, 2017.

[169] C. Wigington, C. Tensmeyer, B. Davis, W. Barrett, B. Price, and S. Cohen, “Start, follow, read: End-to-end full-page handwriting recognition,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11210 LNCS, pp. 372–388, 2018.

[170] J. andreu Sánchez, V. Romero, A. H. Toselli, M. Villegas, and E. Vidal, “Dataset for ICDAR2017 Competition on Handwritten Text Recognition on the READ Dataset (ICDAR2017 HTR),” Jul. 2017.

[171] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional LSTM networks BT - Neural Networks, 2005. IJCNN ’05. Proceedings. 2005 IEEE International Joint Conference on,” vol. 4, pp. 2047–

2052, 2005.

[172] J. Chung and T. Delteil, “A Computationally Efficient Pipeline Approach to Full Page Offline Handwritten Text Recognition,” in *International Conference on Document Analysis and Recognition Workshops (ICDARW)*, 2019.

[173] J. Matas, O. Chum, M. Urban, and T. Pajdla, “Robust wide-baseline stereo from maximally stable extremal regions,” *Image Vis. Comput.*, vol. 22, no. 10 SPEC. ISS., pp. 761–767, 2004.

[174] A. C. Doyle, “The Adventures of Sherlock Holmes,” *Project Gutenberg*, 1999. [Online]. Available: http://www.gutenberg.org/ebooks/1661. [Accessed: 15-May-2020].

[175] G. Orosz, “Awesome NLP Resources for Hungarian,” 2017. [Online]. Available: https://github.com/oroszgy/awesome-hungarian-nlp?fbclid=IwAR2d4Mv-KVGqGArtlkYvflO7Lofi4A9jgklfkDZHJdC3tjXn5FjFBg-bLA8. [Accessed: 20-Jun-2020].

[176] N. Çarkacı, “TDKDictionaryCrawler,” *GitHub*, 2016. [Online]. Available: https://github.com/ncarkaci/TDKDictionaryCrawler.

[177] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Sov. physics-Doklady*, vol. 10, pp. 707–710, 1965.

[178] R. A. Wagner and M. J. Fischer, “The String-to-String Correction Problem,” *J. Assoc. Comput. Mach.*, vol. 21, no. 1, pp. 168–173, 1974.

[179] T. Bluche, “Joint line segmentation and transcription for end-to-end handwritten paragraph recognition,” *Adv. Neural Inf. Process. Syst.*, pp. 838–846, 2016.

[180] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 580–587, 2014.

[181] R. Girshick, “Fast R-CNN,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015.

[182] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks Shaoqing,” *IEEE Trans. Pattern Anal.*

Mach. Intell., vol. 39, no. 6, pp. 1137–1149, 2016.

[183] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016.

[184] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv*, 2018.

[185] G. Jocher *et al.*, "ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements." 2020.

[186] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders, "Selective Search for Object Recognition," *Int. J. Comput. Vis.*, vol. 104, no. 2, pp. 154–171, 2013.

[187] J. Du, "Understanding of Object Detection Based on CNN Family and YOLO," *J. Phys. Conf. Ser.*, vol. 1004, no. 1, 2018.

[188] Ankushsharma, "YOLO V1 Architecture," 2020. [Online]. Available: https://medium.com/@ankushsharma2805/yolo-v1-v2-v3-architecture-1ccac0f6206e. [Accessed: 10-Nov-2020].

[189] A. Anka, "YOLO v4: Optimal Speed & Accuracy for object detection A review of a state-of-the-art model for real-time object detection," 2020. [Online]. Available: https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50. [Accessed: 10-Nov-2020].

[190] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016.

[191] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 2261–2269, 2017.

[192] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path Aggregation Network for Instance Segmentation," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 8759–8768, 2018.

[193] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318–327, 2020.

[194] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6517–6525, 2017.

[195] M. H. Yap *et al.*, “Deep Learning in Diabetic Foot Ulcers Detection: A Comprehensive Evaluation,” 2020.

[196] C. Y. Wang, H. Y. Mark Liao, Y. H. Wu, P. Y. Chen, J. W. Hsieh, and I. H. Yeh, “CSPNet: A new backbone that can enhance learning capability of CNN,” *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2020-June, pp. 1571–1580, 2020.

[197] X. Du *et al.*, “SpineNet: Learning scale-permuted backbone for recognition and localization,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 11589–11598, 2020.

[198] R. Santoso, Y. K. Suprapto, and E. M. Yuniarno, “Kawi Character Recognition on Copper Inscription Using YOLO Object Detection,” in *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM) IEEE*, 2020, pp. 343–348.

[199] Q. Sun, D. Chen, S. Wang, and S. Liu, “Recognition Method for Handwritten Steel Billet Identification Number Based on Yolo Deep Convolutional Neural Network,” in *2020 Chinese Control And Decision Conference (CCDC) IEEE*, 2020, pp. 5642–5646.

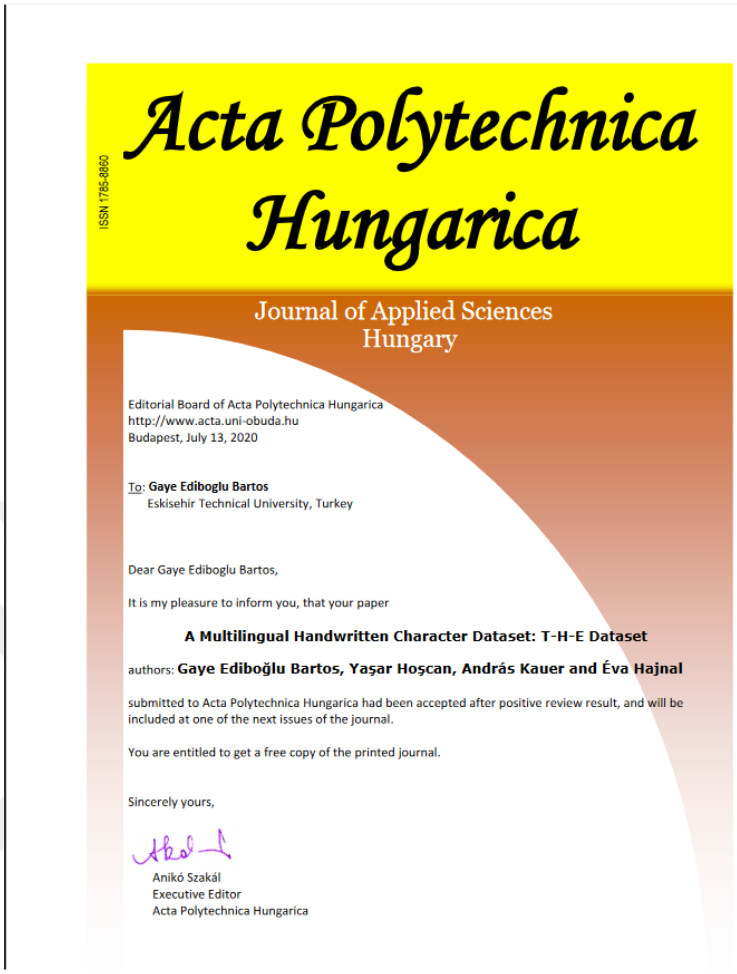
[200] H. Kusetogullari, A. Yavariabdi, J. Hall, and N. Lavesson, “DIGITNET: A Deep Handwritten Digit Detection and Recognition Method Using a New Historical Handwritten Digit Dataset,” *Big Data Res.*, vol. 23, p. 100182, 2021.

[201] A. G. Hochuli, A. S. Britto, D. A. Saji, J. M. Saavedra, R. Sabourin, and L. S. Oliveira, “A comprehensive comparison of end-to-end approaches for handwritten digit string recognition,” *Expert Syst. Appl.*, vol. 165, no. December 2019, 2021.

[202] Pythonlessons, C. Escudero, S. Bijawe, and Tkwant, “TensorFlow-2.x-YOLOv3 and YOLOv4 tutorials,” *GitHub*, 2020. [Online]. Available: https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3/blob/master/mnist/make_data.py. [Accessed: 15-Dec-2020].

[203] G. Jocher, “YOLOv5 in PyTorch,” *GitHub*, 2020. [Online]. Available: https://github.com/ultralytics/yolov5/releases.

10. ACCEPTANCE LETTER OF THE ARTICLE



Acta Polytechnica Hungarica

ISSN 1785-8860

Journal of Applied Sciences
Hungary

Editorial Board of Acta Polytechnica Hungarica
http://www.acta.uni-obuda.hu
Budapest, July 13, 2020

To: **Gaye Ediboglu Bartos**
Eskisehir Technical University, Turkey

Dear Gaye Ediboglu Bartos,

It is my pleasure to inform you, that your paper

A Multilingual Handwritten Character Dataset: T-H-E Dataset

authors: **Gaye Ediboğlu Bartos, Yaşar Hoşcan, András Kauer and Éva Hajnal**

submitted to Acta Polytechnica Hungarica had been accepted after positive review result, and will be included at one of the next issues of the journal.

You are entitled to get a free copy of the printed journal.

Sincerely yours,

Anikó Szakál
Executive Editor
Acta Polytechnica Hungarica

Can be accessed via:

https://eur02.safelinks.protection.outlook.com/?url=http%3A%2F%2Facta.uni-obuda.hu%2FIssue106.htm&data=02%7C01%7Cgayeediboglu%40eskisehir.edu.tr%7C6979aca560c6415f2c5108d858a4554c%7Cbb6a84b1d9b64d469e052ca9be1aa7b9%7C0%7C0%7C637356810114515504&sdata=7exOzpVBlME%2Fjh3B6sY%2FH64P1TqpGsIgb%2FRROb66tX0%3D&reserved=0

11. APPROVAL OF THE ETHICAL COMMITTEE

Eskişehir Teknik Üni. Evrak Tarih ve Sayısı: 12.01.2021 - 575

T.C.
ESKİŞEHİR TEKNİK ÜNİVERSİTESİ REKTÖRLÜĞÜ
Genel Sekreterlik

Sayı : E-53913633-050.99-575
Konu : 30/12/2020 tarihli 16/2 sayılı Etik Kurul Kararı

12.01.2021

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ MÜDÜRLÜĞÜNE

İlgi : 10/12/2020 tarih ve E.28007 sayılı yazı

İlgi yazınıza istinaden Rektörlüğümüze gönderilen Prof.Dr.Yaşar HOŞCAN'ın yürütücülüğünü yaptığı **"Bilgi Sistemleri için Kaba Kümeler Tabanlı Tahmin Sistemi Tasarımı"** başlıklı araştırma projesine ilişkin başvurusu Fen ve Mühendislik Bilimler Bilimsel Araştırma ve Yayın Etiği Kurulu tarafından incelenmiş olup etik açıdan uygun bulunmuştur.

Bilgilerinizi ve uygulama dosyasının hazırlanmasında, ilgili kurumun bulunması halinde Etik Kurulu Yönergesinin dikkate alınması konusunda gereğini rica ederim.

Prof. Dr. Gürsoy ARSLAN
Sosyal ve Beşerî Bilimler Bilimsel
Araştırma ve Yayın Etiği Kurulu Başkanı

Bu belge, güvenli elektronik imza ile imzalanmıştır.

Belge Doğrulama Kodu :BELCK0TM8 Belge Takip Adresi : http://belgedogrulama.eskisehir.edu.tr/BelgeDogrulama.aspx

Adres:İki Eylül Kampüsü Tepebaşı/Eskişehir
Telefon:+90 222 321 35 50 Faks:+90 222 323 95 01
e-Posta:gensek@eskisehir.edu.tr Web:www.eskisehir.edu.tr
Kep Adresi:eskisehirteknikuniversitesi@hs01.kep.tr

Bilgi için: Arzu BALCI
Unvanı: Memur
Tel No: 1119

CURRICULUM VITAE

Name- Surname : Gaye EDIBOGLU BARTOS

Languages: English-Advanced, Hungarian-Pre-Intermediate

Education:

Óbudai University Alba Regia Technical Faculty, Székesfehérvár, Hungary
Erasmus+ Internship programme, PhD Research (11th July 2016- 31st May 2017)

Eskisehir Technical University, Eskisehir, Turkey
PhD Computer Engineering (2013-present), Faculty of Engineering

University of Leeds, Leeds, UK
MSc Computing and Management (2011-2012), Faculty of Engineering

University College of London, London, UK
Pre- Sessional Language Course (3rd May- 23rd September 2011)

Eurocentres Languge Centre, London Lee Green , UK
10 weeks of General English Course, C1 English Language Certificate (2011)

Nicolaus Copernicus University, Torun, Poland
One semester Erasmus exchange student (2008-2009) at Faculty of Education

Cukurova University, Adana, Turkey
BSc Computer Education and Instructional Technologies (2005-2009)

Publications:

G. Ediboğlu Bartos, Y. Hoscan, A. Kauer, and É. Hajnal, "A Multilingual Handwritten Character Dataset: T-H-E Dataset," Acta Polytech. Hungarica J. Appl. Sci., vol. 17, no. 9, 2020.

G. Ediboğlu Bartos, É. Hajnal, and Y. Hoşcan, "Comparison of Feature Extraction Techniques for Handwriting Recognition," in *IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI 2018)*, Temesvár: IEEE Hungary Section; IEEE Romania Section, 2018, pp. 405-410 PG–6.

G. Ediboğlu Bartos, É. Hajnal, and Y. Hoşcan, "Performance Analysis of Character case-sensitive and case-insensitive Classification in Handwritten Character Recognition," in *13th International Symposium on Applied Informatics and Related Areas (AIS 2018)*, 2018.

G. Ediboğlu Bartos, Y. Hoşcan, and É. Hajnal, "Performance Comparison of

Different Classifiers for Hungarian Handwriting Recognition," in *AIS 2017 - 12th International Symposium on Applied Informatics and Related Areas (AIS 2017)*, 2017.

G. Ediboğlu Bartos and É. Hajnal, "Optical Character Recognition for Turkish - a Survey," in *Vasil Levski National Military University Annual Scientific Conference 2017*, 2017.

G. Ediboglu Bartos and É. Hajnal, "Pre-processing Techniques for Hungarian Handwriting Recognition," in *11th International Symposium on Applied Informatics and Related Areas (AIS 2016)*, Budapest: Óbudai Egyetem, 2016, pp. 94–99.

Thakker, D., Dimitrova, V.,Ediboglu, G. Introducing cultural prompts in semantic data browser. Accepted for the Internation workshop on Intelligent Exploration of Semantic Data (IESD'12) in conjuction with EKAW 2012, Galway (Ireland), 9 October, 2012.