PATTERN RECOGNITION APPLICATIONS FOR IMAGE CLASSIFICATION

by

Mert Çetinkaya

B.S., Industrial Engineering, Galatasaray University, 2017

B.S., Computer Engineering, Galatasaray University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2021

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere thanks to my thesis advisor Assoc. Prof. Wolfgang Hörmann for the time that we studied on this thesis. I am really grateful to him for his communication, interest, helpful approach and guiding ideas to improve my thesis even during Covid-19 pandemic.

I also would like to sincerely thank Prof. Tankut Acarman and Assist. Prof. Mustafa Gökçe Baydoğan for agreeing to join the thesis committee, their time and valuable comments.

Lastly, I would like to thank my teachers in the Industrial Engineering Department of Boğaziçi University for making the time I spent during my master program very precious and to contribute me to improve my vision. For me, it will always be a great honour to hold a master degree from this tremendous department and university.

ABSTRACT

PATTERN RECOGNITION APPLICATIONS FOR IMAGE CLASSIFICATION

Pattern recognition and image classification applications are the topic of this thesis. Its focus is on neural network based techniques, in other words, convolutional neural networks, and they are tried to be used in the most efficient way. In addition to convolutional neural networks, some experiments using more classical pattern recognition and classification techniques are conducted and their results are evaluated. The applications are made for traffic sign recognition and medical image recognition using different public datasets that were also preferred in previous studies. In the context of these applications, the obtained results are also compared to the results available in the literature.

As an additional experiment, an optical character recognition and a real text digitalization method is also proposed as a proof of concept study. These experiments again consist of similar steps, but, also a character segmentation approach is developed on real text images, to extract the characters to be classified.

During all these experiments, image processing and pattern recognition and classification techniques are used, and some proposal are brought to augment their efficiency.

ÖZET

GÖRÜNTÜ SINIFLANDIRMA İÇİN ÖRÜNTÜ TANIMA UYGULAMALARI

Bu tezde örüntü tanıma ve görüntü sınıflandırma uygulamaları yapılmıştır. Tez boyunca, sinir ağları tabanlı tekniklere, bir başka ifadeyle, evrişimsel sinir ağlarına daha fazla odaklanılmıştır ve en verimli şekilde kullanılmaya çalışılmıştır. Evrişimsel sinir ağlarına ek olarak, daha eski örüntü tanıma ve sınıflandırma teknikleriyle de deneyler yapılmıştır ve sonuçları değerlendirilmiştir. Uygulamalar farklı çalışmalarda da kullanılmış olan ve herkese açık bir şekilde bulunan trafik levhası tanıma ve tıbbi görüntü tanıma veri setleri kullanılarak yapılmıştır. Bu uygulamalar kapsamında, elde edilen sonuçlar aynı zamanda açık literatürde bulunan diğer çalışmalar ile de sonuçların validasyonu için kıyaslanmıştır.

Ek bir deney olarak, optik karakter tanıma ve bir gerçek metin görüntüsü dijitalleştirme yöntemi de önerilmiştir ancak bu öneri ve ilgili deneyler daha çok yöntemin uygulanılabilirliğini ispatlama amaçlı olarak daha basit seviyede gösterilmiştir. Bu deneylerde de bir önceki deneylere benzer adımlar bulunmaktadır ancak, ek olarak, gerçek metin görüntüleri üzerinde, sınıflandırılacak karakterleri bulmak için bir karakter segmentasyonu yaklaşımı da içermektedir.

Bütün bu deneyler sırasında, görüntü işleme ve örüntü tanıma ve sınıflandırma teknikleri kullanılmıştır ve verimlerini artırmak için bazı öneriler getirilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

b	Bias
C	Inverse regularization parameter
K	Kernel
L	Loss
p	Probability
w	Weight matrix
x	Input
y	Output
α	Momentum coefficient
γ	Forgetting factor
Δw	Change in weight matrix
$\nabla L(w)$	Gradient of loss with respect to weight
ϵ	Small positive scalar
η	Learning rate
ϕ	Kernel function

LIST OF ACRONYMS/ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
ANN	Artificial Neural Networks
BTSC	Belgium Traffic Sign Classification
CE	Cross Entropy
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
EML	Extreme Machine Learning
ENet	Efficient Neural Network
FN	False Negative
FP	False Positive
GB	Gigabyte
GHz	Gigahertz
GPU	Graphics Processing Unit
GTSRB	German Traffic Sign Recognition Benchmark
HOG	Histogram of Oriented Gradients
HSI	Hue Saturation Intensity
HSV	Hue Saturation Value
K-d	K Dimensional
LDA	Linear Discriminant Analysis
LR	Logistic Regression
MLE	Maximum Likelihood Estimation
NB	Naive Bayes
NLP	Natural Language Processing
OCR	Optical Character Recognition
PPC	Pixel per Cell
RAM	Random Access Memory

ReLU	Rectified Linear Unit
ResNet	Residual Networks
RGB	Red Green Blue
RNN	Recurrent Neural Networks
SVC	Support Vector Classifier
SVM	Support Vector Machines
TN	True Negative
TP	True Positive
VGGNet	Visual Geometry Group Networks
VG-RAM	Virtual Generalizing Random Access Memory
WNN	Weightless Neural Networks

1. INTRODUCTION

In this thesis, it is focused on pattern recognition for different image classification problems about traffic sign recognition, medical image recognition and OCR. In OCR section, we also present our character segmentation method from real text images, thus presenting our own simple text image digitalization approach.

Together with their common points, each of the three application areas has its own characteristics. For example, in traffic sign classification, there are lots of classes and in medical image recognition there are very few number of classes. In the medical image recognition applications, we dealt with malaria recognition and breast cancer recognition and in the datasets that we used there are 2 classes as positive cases and negative cases. In addition to this great difference between class numbers, in a medical image recognition problem, obviously, the cost of missing a positive case is quite high and number of positive cases can be likely to be less than number of negative cases. So slightly different solution approaches are required. The third problem, OCR, is special as these characters are easily expressed by a binary representation. At the same time, some characters are quite similar and open to be mixed up, that's why, we needed to develop some additional ideas to distinguish them. Moreover, in this problem, we also developed a method for character segmentation from real text images to extract the characters to be classified in order to obtain a simple OCR engine. In order to do this, we made use of some standard image processing techniques and our own simple algorithm. We ran our simple OCR engine under different conditions and noise effects and showed that it is promising as a proof of concept level study.

As classifier, we mostly used neural networks based classifiers, in other words, CNN, and tried to get most out of it using it as efficient as possible trying to find the best network architecture, using the best and mostly recommended network elements like activation functions, loss functions etc. and adding some problem specific and simple tricks that we developed. In today's image recognition and deep learning problems,

we can see CNN as a very powerful and widely used feature extraction and classification method [1–3]. Because of this situation, we made lots of experiments using CNN and tried to find the best CNN model in an elaborate and detailed way. In addition to CNN, more classical approaches like logistic regression, Naive Bayes, SVM with linear kernel and standard ANN were also used for classification after feature extraction using more classical methods like HOG as a shallow machine learning approach for image classification. However, we made these experiments in a more superficial and coarse way and not as detailed as the CNN experiments. Showing the superiority of CNN over other methods and fine-tuning it as good as possible are between the aims of this thesis.

In the applications in this thesis regarding traffic sign recognition and medical image recognition, we used publicly available datasets that were also used in some other studies found in the open literature. At the end, we also compared our results to the results found in these studies as the validation of our results. Whereas, for OCR, we produced our own dataset using our own alphabet that contains Turkish and English characters, numbers, punctuations and some special characters and showed the performance of our classifiers and presented our results via this dataset.

We conducted all the experiments in an ordinary laptop computer with Intel i5 8250U 1.60 GHz CPU, 16 GB RAM and NVIDIA GeForce 940 MX GPU. We used Python programming language and the data science, image processing and deep learning modules which are Pandas [4], Numpy [5], Scikit-Learn [6], Matplotlib [7], Keras [8], OpenCV [9] and Scikit-Image [10]. To reach the codes, necessary github links and the related information are presented at the end of the related sections.

2. RELATED LITERATURE

Usage of pattern recognition techniques in image classification is a popular approach and in this chapter related literature for the applications in this thesis will be presented. For traffic sign recognition and medical image recognition, we worked with well known and often considered public datasets. For traffic sign recognition we used GTSRB and BTSC dataset. For medical image recognition we used malaria recognition dataset and breast cancer recognition dataset. Therefore the studies considering these datasets are mainly introduced. The numerical results of the literature are also presented in the respective sections in order to compare them with our results.

2.1. Traffic Sign Recognition

Berger *et al.* [11] used virtual generalizing random access memory weightless neural networks as an effective machine learning technique. This is a kind of special RAM-based neural network that can store knowledge inside of their neurons. They used this network and made some preprocessing on images to carry out the recognition task.

Gudigar *et al.* [12] used higher order spectra supported with texture based features and developed an efficient traffic sign recognition model. These features represented the shape and content information of the traffic signs. Then, they used a subspace learning method with graph embedding. In this system, they made use of linear discriminant analysis to increase the discrimination success between traffic symbols from different variety and, so, for traffic sign recognition.

Zakloutta *et al.* [13] used HOG features and distance transforms for feature extraction and K-d trees and random forests to carry out traffic sign recognition.

Wang and You [14] combined boosting learning logic and SVM and used Haar

features and HOG features for traffic sign recognition. Their method was an effective and efficient method that works reducing data dimensionality.

Yin *et al.* [15] proposed a special CNN architecture that includes a special block layer structure which combines network-in-network and residual connection. Their whole network consisted of 10 layers of which 7 belonged to convolutional part and the last 3 belonged to the fully connected part.

Mehta *et al.* [16] produced their own network and made some experiments to see the effect of dropout regularization, different optimizers and different activation functions on this network. They found that adam optimizer and softmax function gave good results.

Lu *et al.* [17] presented a graph embedding algorithm that catches a balance between local miscellaneous features and global discriminative information. They designed a novel graph structure to show the local features of traffic signs with various appearances and to intuitively model discriminative information between classes. Thanks to this graph structure, their algorithm effectively learns a compact and discriminative subspace. In addition, their proposed algorithm could keep the sparse representation property of the original space after graph embedding, and, in this way, it generates a more accurate projection matrix.

Aziz *at al.* [18] used HOG features, Gabor features and compund local binary pattern features and performed classification using extreme machine learning. They stated that extreme machine learning is a new simple learning algorithm for single layer feedforward neural networks. Their learning is faster and they can obtain better generalization than single layer feedforward neural networks.

2.2. Medical Image Recognition

Rajaraman *et al.* [19] applied transfer learning and in this context assessed the performance of pre-trained CNN based deep learning models as feature extractors in classifying parasitized and uninfected cells for malaria recognition. They made experiments and determined the optimal model layers and architecture for feature extraction from the given data. They used some statistical validation techniques and demonstrated that the use of pre-trained CNN is a promising tool for feature extraction on this purpose. They also used their customized CNN model in their experiments.

In a different study, Rajaraman *et al.* [20] evaluated the performance of custom and pre-trained CNN and constructed an optimal ensemble model to make predictions reducing bias and, in this way, improving generalization for classification of parasitized and normal red blood cells in thin-blood smear images for malaria recognition. They showed that the proposed ensemble model which combines multiple deep learning models reached nice predictive performance that could not be achieved by any of the individual models. This was the first study constructing and statistically evaluating an ensemble model that classifies a large clinical dataset of parasitized and uninfected cells.

Qayyum *et al.* [21] used their own CNN based networks for malaria recognition. They firstly started with a 5-layer simple CNN model for classification. Then, they modified this model changing kernel sizes in a linearly dilated way in each layer and in another model they changed the kernel sizes in a Fibonacci series dilated way in each layer. At the end, they found that their last model, the Fibonacci series-wise dilated model, performed the best.

Akilotu *et al.* [22] used transfer learning approach for malaria recognition. They used pre-trained AlexNet and VGGNet networks and their different versions. At the classification step they used SVM to carry out the classification process.

Cruz-Roa *et al.* [23] used their own 3-layers CNN architecture that works with a softmax classifier for breast cancer recognition problem. While running their CNN, they also made an input pre-processing converting the images from RGB to YUV color space. They also extracted some hand-crafted features and used a random forest classifier with these features. At the end, they showed that CNN based classification outperformed the other approaches using hand-crafted features and random forest.

Janowczyk and Madabhushi [24] made a research on different image analysis challenges in digital pathology on different diseases and investigated usage of deep learning on these problems. Breast Cancer recognition was among these tasks and they used AlexNet configuration. They also made use of additional rotations to make a synthetic data augmentation against the imbalanced data problem between negative and positive cases.

Reza and Ma [25] focused on 2 different breast cancer recognition tasks using deep learning. In both tasks, the data was highly imbalanced and they used different sampling approaches to cope with this problem. The network that they used in the experiments was keras-team of Francois Chollet's setups for mnist digit classification with 3 convolutional layers. At the end, they found that oversampling is generally the best sampling approach to increase the CNN performance for highly imbalanced data problems.

Romano and Hernandez [26] dealt with the problem of breast cancer recognition applying some sampling and data augmentation approaches. They firstly applied downsampling against the imbalanced data problem to make number of negative and positive cases equal. Then, they made a synthetic data augmentation using some transformations. They also used deep learning for the classification task. They produced their own model and in this model, in one layer, they used a new pooling strategy called accept-reject pooling rather than traditional pooling approaches in order to minimize the over-fitting problem.

Romero *et al.* [27] also dealt with the problem of breast cancer recognition and in their paper they proposed a model derived from the deep learning architecture named Inception. In this model, they proposed a multi-level batch normalization module between each convolutional steps. They used this module as a base block for feature extraction in a CNN architecture and they used softmax classifier. Their experiments showed that their approach performed better than the original Inception module.

For medical image recognition, at least with regards to the examined papers, we see that usage of deep learning is much more popular than other pattern recognition techniques. In their paper, Bakator and Radosav [28] also showed that deep learning methods have a wide application in the medical field and medical image analysis like detection, segmentation, classification, prediction and other. They also expressed that the future development of deep learning is quite promising with more applications in various fields of medicine, and, particularly, in the domain of medical diagnosis, deep learning can provide a very nice support for experts in the medical field during decision making.

2.3. Optical Character Recognition

Phangtriastu *et al.* [29] worked on only capital letter images and their classification problem. To solve this problem, they used zoning, projection and HOG feature extraction approaches. They used these features one by one and as double or triple combinations and used SVM and ANN as classifiers. At the end, overall, they showed that ANN gives better accuracy than SVM. However, the best accuracy was reached with using SVM with projection and projection + HOG combination.

Roy *et al.* [30] used a simple, but efficient method for digitalizing texts. They used binary representation of texts and segmented characters from texts and matched the flattened and resized representation of extracted character with the ones in training data to find its label. They only looked at the number of matching binary pixels and labelled the character to the one that it is the most closest with regards to this approach.

They also used some noise reduction techniques and produced a model which is robust to noisy texts.

Xie [31] used robust gray value normalization and SVM with radial basis function kernel for character segmentation from text images and classification. He tested his model using some text images and showed that it works well.

Afroge *et al.* [32] used some median filtering for noise reduction and binary image for character extraction drawing rectangles around the borders of images. Then, they made resizing operation on extracted characters and flattened binary characters to obtain related input. They trained an ANN in this way and obtained their well-functioning OCR mechanism.

Vamvakas *et al.* [33] worked on handwritten Greek characters. They only worked for character recognition like Phangtriastu *et al.* [29]. They worked with binary characters and made 2 types of feature extraction and combined them in a hybrid fashion. In the first one, they divided the character image into a set of zones and calculated character pixels' densities in each zone. In the second one, the area that was formed using projections of upper and lower, and also, of left and right character profiles was calculated. After feature extraction, they used SVM with radial basis function kernel for classification.

Wei *et al.* [34] used some deep neural network approaches and made some experiments on noisy or poor quality text images. They firstly made some preprocessing applications like transforming to a binary one and dilation operation for character segmentation and they used their own deep neural networks architecture and inception v3 pre-trained model as a transfer learning approach on extracted characters. They showed that the best performance was obtained with inception v3 architecture and their system worked well for digitalization of poor quality text images.

Sevik *et al.* [35] worked on Turkish letters and made use of AlexNet deep learning

architecture. They trained the pre-trained Alexnet using their own dataset. They used this model to complete their OCR engine. For character segmentation step, they made use of image preprocessing techniques like obtaining binary image and morphologic operations. They also defined some special rules for accented or dotted characters.

Lastly, there also exists some more advanced studies where texts with more complex backgrounds are detected and digitalized. These texts that are hard to find and extract can be found in more complex natural images as a small part of these images [36–39]. However, in this thesis, we realized the OCR applications on simpler images. Also, an important tool is the Tesseract OCR engine and open source library [40].

3. METHODOLOGY

In an image classification problem the standard procedure is firstly extracting features from image and making the classification using these features as input for each image. In this thesis, we benefited from HOG features and deep learning feature extraction in traffic sign recognition and medical image recognition and binary representation of images in OCR. We used HOG features and binary representation of images with Naive Bayes, Logistic Regression and SVM as more classical classification approaches and the usage of HOG features with ANN was also presented as a transition between classical methods and deep learning. In traffic sign recognition and medical image recognition we used deep learning as is by many authors considered a very powerful and modern approach for image recognition problems [1–3]. Deep learning can be seen as a combination of two types of neural networks: CNN and ANN. They work together for feature extraction and classification.

3.1. Image Feature Extraction Methods

In this thesis, HOG features, deep learning based features and binary representation of images for feature extraction are used. However, there are some more methods as well for feature extraction like Haar like features, histogram of hue values, Wavelet transform, Fourier transform, Local Binary Pattern etc. Between these more classical methods HOG features are accepted to be more powerful and it is more popular and widely used [13, 14, 18]. That's why, we decided to use HOG features among these classical image feature extraction methods. For only OCR, we preferred binary representation of images, because this approach is quite common in this research domain [30,32,33]. Below, we explain HOG features and transformation to a binary image as feature extraction methods. We also explained usage of deep learning, but we explained it under classification methods title, because deep learning is a comprehensive name for feature extraction and classification.

3.1.1. Histogram of Oriented Gradients

HOG features were firstly presented by Dalal and Triggs [41] in 2005 to solve human detection problem. In their problem they used Linear SVM and HOG features. They proposed a 5-stage descriptor. These stages are briefly explained below [41].

1. Global Image Normalisation: This optional step may be helpful to reduce some illumination effect on the image. Pixel values are normalised using some normalisation techniques like gamma normalisation, square-root normalisation etc.
2. Gradient computation: The image gradients are calculated in both the x and y directions thanks to some convolutional operations. Then, using these gradients, the final gradient magnitude image and orientation of the gradient for each pixel is calculated.
3. Weighted votes in each cell: The gradient image is divided into cells and for each cell in the image, histogram of oriented gradients is constructed using gradient magnitudes and orientation information of the pixels in that cells. Here, the histograms are constructed according to a pre-defined orientation number. It is generally preferable to use unsigned gradients in the range [0, 180] with orientation number in the range [9, 12]. In a cell, each pixel contributes a weighted vote to the histogram and the weight of a vote can be simply thought as the gradient magnitude at that pixel.
4. Contrast normalization over blocks: The cells are grouped into larger and connecting blocks that commonly overlap. This means that each cell contributes to the final vector more than once. In a block, for each cell, the corresponding gradient histograms are concatenated and this entire concatenated feature vector is normalised using for example L1 or L2 normalisation. This block normalisation is done against lighting variations and it increases the descriptor performance.
5. Calculating the final HOG feature vector: In this step, after all blocks are normalised, the resulting histograms are concatenated and they are treated as the final vector for the image.

If we try to explain this procedure in a simpler way we can say that cells consist of pixel values and blocks consist of cells. HOG values are extracted in each cell using the gradient magnitude and orientation of pixel values and these histograms are concatenated and normalised in blocks. Then, normalised vectors of blocks are concatenated and the final feature vector is obtained. The most important parameters are pixel per cell value, cell per block value and orientation number value in a histogram [41].

One of the images used in traffic sign recognition and the related representative HOG image is given in https://github.com/mertcetinkaya/master_thesis_codes to show the main idea. The image can be found in images folder in the link. In this image, the traffic sign image is taken from GTSRB dataset and in the HOG image, a line segment is given for each cell at the centre of that cell and intensity of these line segments is proportional to the corresponding histogram value.

3.1.2. Binary Representation of Images

Binary representation of images and usage of these binary representation as features is a very simple and popular approach especially for optical character recognition [30, 32, 33]. In this logic the image is firstly converted to grayscale. For example, an RGB image can be converted to grayscale using Equation (3.1) for each pixel [32]:

$$Gray = 0.299 * R + 0.587 * G + 0.114 * B. \tag{3.1}$$

Then, this grayscale image is converted to black and white image using the obtained grayscale pixel values. For example, if the pixel value is lower than 127 than that pixel can be accepted as 0 (black), else it can be accepted as 255 or 1 (white) [32].

3.2. Classification Methods

In this thesis, we carried out some image recognition applications. We used some classical approaches and deep neural networks to create different classifiers and compared the results. The machine learning based classifiers are mainly examined under 2 different categories [42]. As a general knowledge, the discriminative classifiers are more successful in terms of accuracy.

3.2.1. Discriminative vs. Generative Classifiers

Generative classifiers learn a model of joint probability, $p(x, y)$, between input x and label y and they make their predictions using Bayes rules calculating conditional probabilities $p(y|x)$. Then, the most likely label y is chosen. Discriminative classifiers model the posterior $p(y|x)$ directly, or they learn a direct map between inputs x and the labels y [43]. Discriminative models directly assume functional form for $p(y|x)$ and parameters of the function are estimated using training data to draw a boundary between classes.

The Naive Bayes approach is a generative classifier and is a very well-known and classical method [43]. We also used logistic regression, SVM and neural networks and these methods belong to discriminative classifiers [1, 43–45]. Between these methods, neural networks show us the performance of deep learning. Also, logistic regression and SVM are 2 other popular methods in machine learning. We used SVM with linear kernel and comparison of logistic regression and SVM with linear kernel is also interesting as both are linear classifiers.

3.2.1.1. Naive Bayes. According to Bayes Rule, the probability for an example $x = (x_1, x_2, ..., x_n)$ to be from class y is shown in Equation (3.2) [46]:

$$P(y|x_1, x_2, ..., x_n) = \frac{P(y)P(x_1, x_2, ..., x_n|y)}{P(x_1, x_2, ..., x_n)}. \tag{3.2}$$

The above expression can be re-written as in Equation (3.3) using the independence assumption:

$$P(y|x_1, x_2, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, x_2, ..., x_n)}. \tag{3.3}$$

Because the expression $P(x_1, x_2, ..., x_n)$ is constant given the input, $P(y|x_1, x_2, ..., x_n)$ becomes proportional to $P(y)\prod_{i=1}^{n} P(x_i|y)$ and Equation (3.4) can be used for classification:

$$\hat{y} = argmax\ P(y)\prod_{i=1}^{n} P(x_i|y). \tag{3.4}$$

Here, when dealing with continuous training data, a typical assumption is to distribute it according to a normal (or Gaussian) distribution. In this way, $P(x_i|y)$ values can be obtained to use in the formula. Another way to deal with continuous training data may be discretizing it into bins. In the applications in this thesis, we used Gaussian distribution assumption [46].

Training of Naive Bayes is quite fast, because only some probability information is calculated and used and the independence assumption which is the naive part of this approach helps to alleviate problems sourced by curse of dimensionality in these calculations. There does not exist any coefficient to estimate using optimization procedures. However, Naive Bayes is generally known to be a weak classifier [46].

3.2.1.2. Logistic Regression. Logistic regression is a linear model for classification. As an optimization problem, $l2$ penalized logistic regression minimizes the following cost function given in (3.5) [47, 48]. In this cost function there exists a penalty term and logistic loss. Practically, C is taken as 1:

$$\min \frac{1}{2}w^T w + C\sum_{i=1}^{n} log(exp(-y_i(w^T x_i + b)) + 1). \tag{3.5}$$

Similarly, $l1$ regularized logistic regression deals with the following cost function given in (3.6):

$$\min \sum_{i=1}^{n} |w_i| + C \sum_{i=1}^{n} log(exp(-y_i(w^T x_i + b)) + 1). \tag{3.6}$$

Also, there is elastic net regularization. It is a combination of $l1$ and $l2$ and it minimizes the following cost function given in (3.7) [47, 48]:

$$\min \frac{1-\rho}{2} w^T w + \rho \sum_{i=1}^{n} |w_i| + C \sum_{i=1}^{n} log(exp(-y_i(w^T x_i + b)) + 1). \tag{3.7}$$

3.2.1.3. Support Vector Machines. SVMs are a set of supervised learning methods and they are used for classification and regression. We use linear SVM in this thesis.

An SVM constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space and these can be used for classification or regression tasks. Intuitively, a good separation is obtained by the hyper-plane that has the largest distance to the nearest training data point of any class [47].

An SVC solves the following primal problem given in (3.8):

$$\begin{aligned} \min \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i, \\ \text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \\ \zeta_i \geq 0, i = 1, ..., n. \end{aligned} \tag{3.8}$$

Here, we are trying to maximize the margin (by minimizing $||w||^2 = w^T w$), while giving a penalty when a sample is misclassified or within the margin region.

Ideally, the value $y_i(w^T \phi(x_i) + b)$ would be $>= 1$ for all samples, which indicates a perfect prediction. However, as expected, problems are usually not always this much

perfectly separable with a hyperplane. That's why, some samples are allowed to be at a distance from their correct margin boundary and the strength of penalty given for this case is controlled by the penalty term C [47].

The dual problem for the primary problem is given in (3.9):

$$\begin{aligned} &\min \frac{1}{2}\alpha^T Q\alpha - e^T\alpha, \\ &\quad \text{subject to } y^T\alpha = 0, \\ &\quad\quad 0 \leq \alpha_i \leq C, i = 1, ..., n. \end{aligned} \tag{3.9}$$

In (3.9), e is the vector of all ones and Q is an n by n positive semidefinite matrix. $Q_{ij} \equiv y_i y_j K(x_i, x_j)$ with $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel. The terms α_i are called the dual coefficients, and their values are bounded from above by C. This dual representation shows the situation that training vectors are represented in a higher (maybe infinite) dimensional space thanks to the function ϕ in an implicit way. Using function ϕ, kernel trick can be applied and its aim is to project the data in a higher dimension space. This can be done using standard feature transformation methods and adding new features as well, but this can potentially blow up the dimension number and kernel trick is a more elegant and easier way to do this. Using this data transformation a linearly separable data is obtained and it is separated by a hyper-plane. Here, usage of duality gives us the chance to apply kernel trick thanks to involving the input features via inner products ($K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$) [49]. In this new problem given in (3.9), an equivalent quadratic programming problem to the primal problem given in (3.8) is obtained.

When this new optimization problem is solved, the output for a given sample x becomes $\sum y_i\alpha_i K(x_i, x) + b$ and the predicted class corresponds to its sign. Here, summing only over the support vectors that are the points found in the margin region is needed because the dual coefficients, α_i, become 0 for the other points.

In the applications in this thesis, we always used SVM with linear kernel. In

this case, the primal problem can be equivalently formulated as the following, given in (3.10), where the hinge loss is used:

$$\min \frac{1}{2} w^T w + C \sum_{i=1}^{n} \max(0, 1 - y_i(w^T \phi(x_i) + b)). \tag{3.10}$$

This approach does not include inner products between points, so the famous kernel trick cannot be realized and ϕ becomes the identity function [47].

To make an analogy between logistic regression and Linear SVM, we can think the cost function as a sum of loss and penalty/regularization expressions and in the context of this analogy, the loss function for logistic regression becomes logistic loss and the loss function for SVM becomes the hinge loss presented above. When $w^T x + b$ is greater than 1, SVM loss becomes 0, since this corresponds to an observation that is on the correct side of the margin. Both loss functions seem quite similar and logistic loss is actually a smoothed version of SVM loss. Due to the similarities between their loss functions, the 2 classifiers often give similar results. In case of well separated classes, SVM tends to be more successful than logistic regression and in more overlapping cases, logistic regression is more preferable [48].

3.2.2. Deep Learning

As a widely used approach in today's pattern recognition problems like image recognition [1–3] deep learning is based on usage of neural networks mostly and these neural networks can be in different types like ANN, CNN and RNN. Especially in the context of image recognition, usage of CNN and ANN is very common. As a convention, firstly CNN is used and a new image is obtained thanks to some convolutional operations and some other operations on the input image. Then, this new image is given to ANN or fully connected layer (or dense layers) after being flattened to a 1D vector and this vector is processed and classified by this ANN. CNN and ANN work together in cooperation as components or parts of a model or a network and their weights are updated together and with regards to each other, iteratively, to reduce the

error as much as possible [50]. A simple representative image for deep learning is given in [51].

To obtain a good understanding, we firstly explain basic deep learning elements (belonging to CNN and ANN). Here, we start with Fully Connected Layer which is actually the ANN part of the standard deep learning structure used in image recognition. This standard structure is formed by combination of CNN and ANN. We must also add that ANN is the most basic topic of deep learning and neural networks [50, 52].

3.2.2.1. Fully Connected Layer. The fully connected layer can be thought as traditional ANN. As a convention, the 'image' obtained at the end of the CNN section is flattened to a 1D vector and given to this layer and classification is carried out. In most of the applications, CNN based features are used with ANN, but these features can be used with other classification methods as well.

Here, we must add that ANN is another feature extractor and in its last layer the classification is done using a proper activation function like softmax or sigmoid. CNN part will be explained after this part but, we must say that the feature extraction process of ANN is much simpler than CNN. It works with 1D inputs and in each layer the result of $w^T * x + b$ is put in an activation function like ReLU, sigmoid etc. in each neurone and the output of this activation function (neurone) becomes the result (or output) of related layer and input of the next layer. This process can continue during several layers and, at the end, classification is done by the last layer and a loss is calculated comparing predicted and real results as a supervised machine learning approach. Using this loss function the weights of the neural networks are updated through a backward signal and this approach is called back propagation. As an iterative process, this process continues, preferably, until a convergence is found in loss function. The same training logic is also kept when CNN part (convolutional layers) is added before ANN part and loss calculation and weight updating are done on the whole network using this logic [50, 52].

Here, the loss function is an important element. Preferably, cross entropy based loss functions are used in deep learning. To talk a bit about cross entropy, it is a measure from the field of information theory and it calculates the difference between two probability distributions [1].

The formula for cross entropy is given in Equation (3.11) where P is the target distribution and Q is its approximation. In the loss calculation, in practice, average cross entropy across all training examples is calculated:

$$CE(P,Q) = -\sum_{i=1} P(x_i)log(Q(x_i)). \tag{3.11}$$

Usage of cross entropy loss function instead of more classical approaches like sum of squares based approaches leads to faster training and improved generalization in classification problems [1]. It improved model training process that previously suffered from saturation and slow learning [2].

When it comes to target and predicted probability distribution, it is also necessary to add that in most supervised learning problems the goal is to estimate a conditional probability $P(y|x;\theta)$ in order to predict y given x and the MLE framework is quite important and useful to find model parameters. Thinking this way, in this concept, MLE becomes equivalent to minimization of cross entropy. Most modern neural networks are trained according to maximum likelihood idea and this means that the cost function is formulated as the cross entropy between the training data and the model distribution [2].

Here, the aim is minimizing the loss value (cross entropy loss) and stochastic gradient descent based optimization algorithms are used for this purpose during iterations and epochs in neural networks [50, 52]. These optimization approaches for minimizing the loss will be explained later. Now we continue with the CNN part and explain its characteristic elements.

3.2.2.2. Convolution. For CNNs a convolution is applied on small, local regions of the image. This is done sliding a filter over the image spatially and computing dot products for small chunks of the image using the formula $w^T * x + b$. At the end of each convolution operation, one number is obtained and using and putting these numbers together an activation map is obtained with one pixel depth [52]. We see an example of this operation in [52]. Using several filters, several activation maps are obtained and they are put back to back.

In a convolution layer, after obtaining all activation maps, the activation maps are put in an activation function (mostly ReLU activation is preferred [3]) and the image obtained after this activation function operation becomes input of the next layer as the final output of convolution layer. The formula for ReLU activation function is given in Equation (3.12) [52]:

$$r(x) = max(0, x). \tag{3.12}$$

3.2.2.3. Stride. Stride option can be used to reduce the size of output of convolution layer. Manipulating stride size, overlaps can be diminished and this causes output size to reduce. Stride can be defined as step size of convolution filter in each movement and in the examples above, we used 1 stride size. For an input with size $N \times N$ with filter size $F \times F$ and stride S, output size is $(N - F)/S + 1$ for one border [52].

3.2.2.4. Padding. Convolution operation shrinks the image a little and during convolution operation, the information on corners and on borders of the image is used less. To overcome this problem, padding strategy, mostly zero-padding, is used. The borders of an input are filled with zeros and this input undergoes convolutional operations. In this way, it also gives the chance to manage the output size. Thinking P as the number of layers of zero-padding for a border $(N + 2P - F)/S + 1$ becomes the new output size and this output can be used as input of another convolution layer [52]. For example, P is 1 in Figure 3.1.

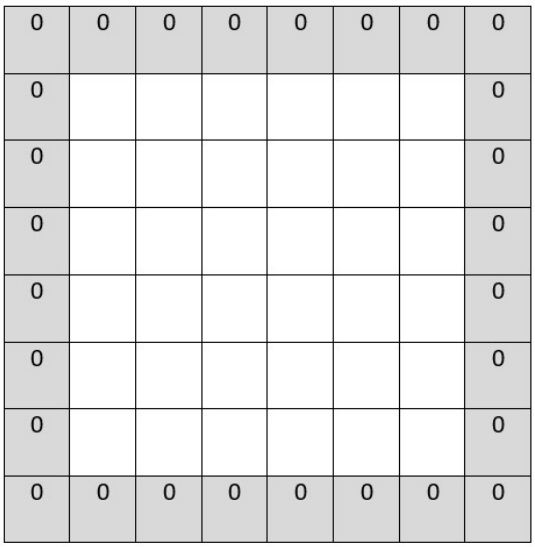

Figure 3.1. Zero Padding.

3.2.2.5. Pooling. Pooling is applied to make the representations of convolution layer outputs smaller and reduce the complexity for operation ease. It is done over each convolution layer output independently and it is, in fact, a downsampling operation on these outputs. A representative image is given in [52]. The most common type of pooling is max-pooling. It partitions the image to sub-region rectangles, and it only returns the maximum value inside of that sub-region. These sub-regions generally become of size 2x2 and stride size becomes 2 like it is given in [52].

3.2.2.6. Dropout. Dropout is a regularization technique used for neural networks. Normally, after some training, each neuron in a neural network starts to be tuned for a specific feature according to input and output and this situation can lead to overfitting. In dropout approach, in each pass, randomly chosen neurons are dropped out of the network during training and specialization of a neuron for a feature is blocked in a certain extent. As a result, the network becomes less sensitive to specific features in input and it becomes capable of better generalization [52].

3.2.2.7. Optimization in Deep Learning. In today's deep learning problems, stochastic gradient descent based optimization is of core practical importance and it is used a lot in many fields of science and engineering [53]. In these approaches a loss value is calculated and using its gradient with respect to weight values, each weight value is updated in the backward direction and to the required extent under the concept of back propagation. This weight updating is done through a backward signal after loss calculation and after updating each weight on the whole network the same training procedure is repeated until reaching a pre-defined training termination condition. Target is finding the minimum point for the loss value and this is done trying to find the point where the gradient of loss function with respect to weight values is equal or close to 0. There may be some problems during this operation like long convergence time, getting stuck in local optima etc. [52,53]. Figure 3.2 is an image to represent this weight adjustment to reduce loss aka Gradient Descent.

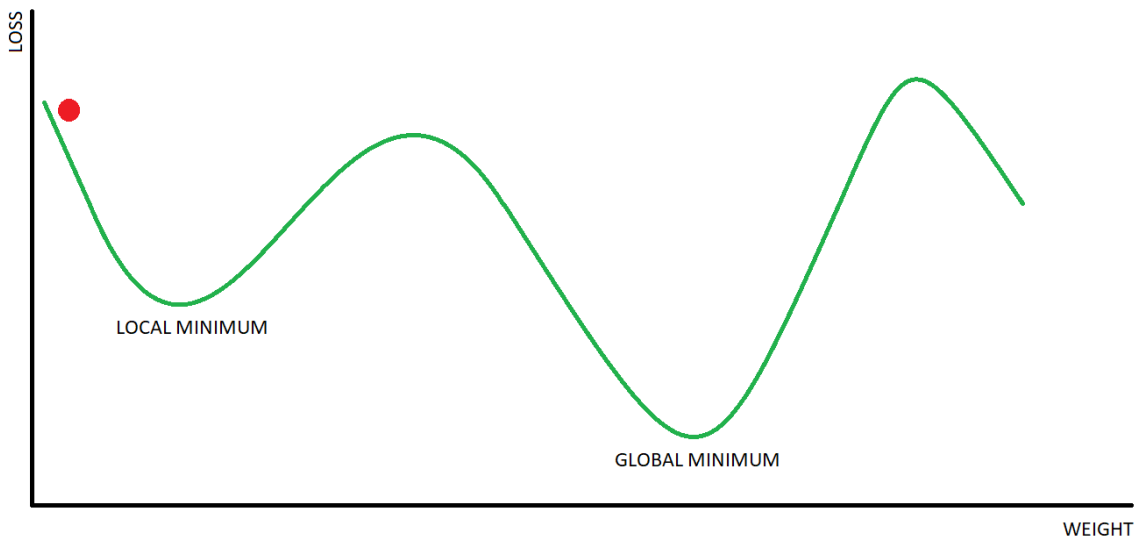

Figure 3.2. Gradient Descent.

The gradients of Loss function with respect to weight values are found using the 'chain rule'. As an example for this situation, the computational chain given in [52] can be used.

In that computational chain, $q(x, y) = x + y$ and $f(q, z) = qz$. To calculate the derivative of f with respect to x or derivative of f with respect to y, Equations (3.13) and (3.14) can be used and this approach is used to calculate the gradients of Loss function with respect to other weights [52]:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}, \tag{3.13}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}. \tag{3.14}$$

Now, if we look at these gradient descent approaches with its general principles, we can say that **Stochastic Gradient Descent** [54] is the most basic one with the following formula given in Equation (3.15). Here, η is the learning rate. The gradient of loss function is calculated and the weight value is modified accordingly:

$$w = w - \eta \nabla L(w). \tag{3.15}$$

As a weakness of Stochastic Gradient Descent, we can say that this algorithm can get stuck in local optima easily because if gradient is around or equal to 0, it will not be able to leave a local optimum and there will not be an important change in the weights. That's why usage of **Momentum** [55] is a nice approach with the below formula given in Equation (3.16). Here, the last change in weight is also added with a momentum coefficient α and this can be useful for solving the local optima problem of stochastic gradient descent. Usage of momentum also accelerates the learning process:

$$w = w - \eta \nabla L(w) + \alpha \Delta w. \tag{3.16}$$

As another weakness of Stochastic Gradient Descent, we can say that it does not include any learning rate update. It is a good idea to decay learning rate as getting closer to the optimal point in order to avoid oscillation around this optimum point and **AdaGrad** (adaptive gradient algorithm) [56] applies this learning rate decay using the

following Equations in (3.17) and (3.18):

$$v = v + (\nabla L(w))^2, \tag{3.17}$$

$$w = w - \frac{\eta}{\sqrt{v + \epsilon}} \nabla L(w). \tag{3.18}$$

As a modification to AdaGrad, **RMSProp** (Root Mean Square Propagation) [57] approach was suggested. In Adagrad, the learning rate is decayed very aggressively and after a while the weight updates will be so small that the training gets very slow. To avoid this, the learning rate decay using the Equations in (3.19) and (3.20), is done in a more gentle way. Here, γ is the forgetting factor:

$$v = \gamma v + (1 - \gamma)(\nabla L(w))^2, \tag{3.19}$$

$$w = w - \frac{\eta}{\sqrt{v + \epsilon}} \nabla L(w). \tag{3.20}$$

As a combination of Momentum and RMSProp approach **Adam** (Adaptive Moment Estimation) [53] optimization algorithm was introduced. The following Equations in (3.21), (3.22), (3.23), (3.24) and (3.25) are used and we can think that numerator represents the momentum part while denominator represents the RMSProp part while updating w in the last step:

$$m = \beta_1 m + (1 - \beta_1) \nabla L(w), \tag{3.21}$$

$$v = \beta_2 v + (1 - \beta_2)(\nabla L(w))^2, \tag{3.22}$$

$$\hat{m} = \frac{m}{1 - \beta_1}, \tag{3.23}$$

$$\hat{v} = \frac{v}{1 - \beta_2}, \tag{3.24}$$

$$w = w - \eta \frac{\hat{m}}{\sqrt{\hat{v} + \epsilon}}. \tag{3.25}$$

Combining momentum and RMSProp together, Adam optimization algorithm is a nice and popular approach. It was introduced by Knigma and Ba [53] and their empirical results demonstrated that Adam works well in practice and gives better results than other stochastic optimization methods. In this thesis we also used Adam optimizer. Good default settings are η= 0.001, β_1=0.9, β_2=0.999 and $\epsilon = 10^{-8}$ [53].

3.2.3. Randomness in Deep Learning

Up to now, we have tried to present deep learning highlighting its important aspects. Here, it is also necessary to mention the notion 'randomness'. There is always a randomness in deep learning sourced by several reasons and this randomness may come into prominence while evaluating different deep learning architectures. Actually, there can always be some randomness in all machine learning methods, but because it has lots of parameters and has a more complicated structure, handling the randomness in deep learning is more difficult. Ending up with different models for likewise trained architectures is a very common issue in the deep learning workspace. Here, we must add that it can be a bit complicated task to remove this randomness in deep learning software libraries. This randomness in deep learning comes from the randomness in weight and bias initialization in layers, stochastic structure of the optimization algorithm, random batch selection during the iterations and randomized dropout (if used) etc. [58]. In order to handle this randomness and to make better model evaluations, the same architecture with exactly the same logic can be trained from scratch several times and statistical properties like minimum, maximum, mean, median etc. for accuracy values (for example if it is a classification problem) can be analysed to see the performance and validity of that architecture or training approach [19,20]. In this way, an efficient network architecture selection can be realized.

3.2.4. What Makes Deep Learning Different?

There are some points that make deep learning different from other machine learning approaches and this improves its performance. As the core and most basic difference of deep learning, it facilitates feature engineering and makes it more efficient. Traditionally, like in the HOG feature case, the features of an image are extracted manually in a handcrafted way and are then used for learning algorithms. This concept is called shallow learning. In this concept, features are extracted according to some specific rules. This feature engineering part is often time-consuming and requires expertise.

In deep learning however, the feature extraction process (also called feature engineering), is carried out automatically. Thus it can be designed in a more powerful and complicated way using a deepenable layer by layer structure. These layers are updated together in an iterative way following each others' needs during training thanks to a feedback signal which represents the error. In this way, the layers can jointly learn the data. Here, each layer or filter can specialize for the extraction of a particular feature and this specialization of layers or filters allows to recognize more complex and secret features in the deeper layers. During this process, the network transforms the input image into representations that are increasingly different from the original image and each representation is only a simple transformation of the previous one. As an analogy, deep network can be seen as a multistage information-distillation operation. During this operation information goes through the successive filters and becomes more and more purified in each step. See [50], where this principle is illustrated with a figure of a digit-classification model.

The figure given in [52] is another and a more detailed representation of image classification by CNN. Convolution, ReLU, pooling and fully connected operations and their results are given more clearly.

This jointly and automatic learning mechanism and complex structure working

in harmony makes the model capable of extracting complicated and invisible relations in the given data. This situation becomes important if the data are more difficult to understand or to resolve like it is in image recognition or speech processing. Together, these properties make deep learning often more successful than other machine learning approaches. Also, removing the manual feature engineering part makes its application easier [50, 59].

3.2.5. Choosing the Best Deep Learning Architecure

Choosing the best deep learning architecture and its fine-tuning is not an easy task because there are lots of parameters and lots of combinations are available, each creating a (slightly) different model. This is generally made in a heuristic way using some general guidelines, but there exist no precise rules how to obtain a good model. In our image recognition problem a very popular deep learning logic combining CNN and ANN is used. For choosing the best architecture, starting from some popular deep learning models is a common approach. These models can be used in two ways: a model can be used as a baseline and some modifications can be obtained by trial and error to get a very good model. On the other hand a pre-trained model for a different problem can be imported and retrained freezing some layers and changing the weights of others. This second approach is called transfer learning [2].

In the applications in this thesis, we produce our own models using VGGNet as a baseline and we construct our models on this baseline making some modifications. There are actually several famous deep learning models like LeNet, AlexNet, VGGNet, ResNet and Inception. Between them we decided to choose VGGNet. VGGNet is currently the most preferred choice in the community for feature extraction from images and it has been used in many applications and challenges as a baseline feature extractor [60].

VGG is an abbreviation for 'Visual Geometry Group' and the idea of VGGNet emerged from the Visual Geometry Group at Oxford University by Simonyan and

Zisserman [61]. In a VGG Block there exists a sequence of convolutional layers, followed by a max pooling layer. They used convolutions with 2 sequential 3x3 kernels and 2x2 max pooling with stride, together with fully connected layers at the end of the CNN part. For an input of 224x224 RGB images they used between 64 and 512 filters in convolutional layers. They experimented especially with architectures using between 11 and 19 convolutional layers.

In the experiments, we used the logic in VGGNet as a baseline while constructing our network architecture, but we used a smaller version with 32x32 RGB images and less convolutional layers and filters. We used a simpler VGGNet at the baseline and added some modifications on that with regards to training and validation results. We made these modifications to decrease the model size and to obtain light models that can be easily adaptable for usage in embedded systems like inside of an autonomous car or inside of a medical device (in terms of size and real time work). In this way, we tried to carry out a more realistic application. In addition to its usage, training of heavy models is also computationally expensive and for recognition of small images, usage of heavy models is not efficient [62]. Here, we also made the mentioned modifications to facilitate and accelerate the computations for training a model and saw that we are able to obtain good results using small images and light models.

4. EXPERIMENTS AND RESULTS

In this section we present our approaches and experiments on 3 different problems.

4.1. Traffic Sign Recognition

For traffic sign recognition, we used GTSRB dataset and BTSC dataset. They are two public datasets popular in this research area.

4.1.1. German Traffic Sign Recognition Benchmark Dataset

GTSRB is a multi-category classification competition held at International Joint Conference on Neural Networks in 2011 and Stalkamp *et al.* [63] presented the GTSRB dataset for this competition. This is a comprehensive and lifelike dataset containing more than 50,000 traffic signs from 43 classes with unbalanced class frequencies. The dataset was created using approximately 10 hours of videos recorded driving on different roads in Germany during daytime in March, October and November 2010. The resolutions of the images in the dataset change between 15x15 and 250x250 [64].

Examples for the traffic sign classes and image types can be seen in Figure 4.1.

Figure 4.1. Examples from GTSRB Dataset.

4.1.1.1. Experiments with Classical Methods. The average of width and height length for images in our training data is approximately 50 pixels. In the experiments we resized all images to 50x50 resolution. We used different pixel per cell (PPC) values to express the level of detail extracted. We used 5, 10 and 15 for PPC (cell size is 5x5, 10x10 or 15x15 pixels) and fixed cell per block to 2 (block size is 2x2 cells) and orientation to 9 to extract HOG features. These are also popularly used and recommended parameters for HOG feature extraction [41,63]. For PPC 5, 10 and 15 the number of features extracted (the length of vector) became 2,916, 576 and 144, respectively, for each image. For all training data feature extraction took around 90 minutes, 20 minutes and 5 minutes, respectively. Here, we tried to examine for different levels of detail extracted from an image by different PPC values. There are also 3 different HOG feature sets published by the official source of the dataset. Among these sets, two of them contain 1,568 features for each and in one of them there are 2,916 features. In our dataset there are 39,209 training images and 12,630 test images. Table 4.1 shows the training results and Table 4.2 shows the test results. In this part we made our experiments for logistic

regression (LR), linear SVM, Naive Bayes (NB) and ANN with HOG feature inputs taking the penalty term C as 1 in LR and SVM [47]. Here, it is also necessary to add that there is always some randomness in ANN and the results given for ANN change in every time that we train the same model architecture from scratch and this change is in a very small interval. The results for ANN are given for 1 model trained and they can be thought as approximate results. This subject is examined with more details in the next subsection which is about the experiments with deep learning. In the experiments with ANN we used a model with 2 layers. In the first layer there are 128 neurons and the second layer is output layer that determines the classification result. We trained the model during 25 epochs. This ANN is actually the same with the one used in the fully connected layer part of deep learning experiments. These ANN experiments can be seen as a nice transition between usage of HOG + ANN combination and CNN + ANN combination to see the exact contribution of CNN features over HOG features. More details about this ANN model will be given below when discussing deep learning experiments that combine CNN and ANN.

Table 4.1. GTSRB Dataset Training Results.

	PPC 5	**PPC 10**	**PPC 15**
LR Accuracy	0.99	0.90	0.70
SVM Accuracy	0.99	0.94	0.81
NB Accuracy	0.92	0.83	0.72
ANN Accuracy	0.99	0.97	0.89
LR Training Time (s)	42	17	11
SVM Training Time (s)	69	21	7
NB Training Time (s)	2	0.38	0.08
ANN Training Time	77	34	27

Table 4.2. GTSRB Dataset Test Results.

	PPC 5	**PPC 10**	**PPC 15**
LR Accuracy	0.92	0.82	0.66
SVM Accuracy	0.92	0.85	0.74
NB Accuracy	0.81	0.72	0.63
ANN Accuracy	0.92	0.87	0.80

Regarding Table 4.1 and Table 4.2, we see that the best results are obtained using ANN and it is the most robust one to decreases in level of details extracted. As another important result, we see that increasing level of detail always influences positively the classifier performance and, as expected, it also increases the training time. We see that LR and SVM have similar performance but SVM seems a bit better. We also see that with its 'naive' approach NB has the lowest accuracy, but it is quite fast, much faster than the other methods due to its simple structure.

4.1.1.2. Experiments with Deep Learning. To find the best deep learning model, we use a simple VGGNet architecture as baseline and construct our model on that by adding some new layers and making some modifications with regards to training and validation results. Our baseline model is given in Figure 4.2. In that Figure, None value shows us the batch size and it is given None by default on the model plot by the software module.

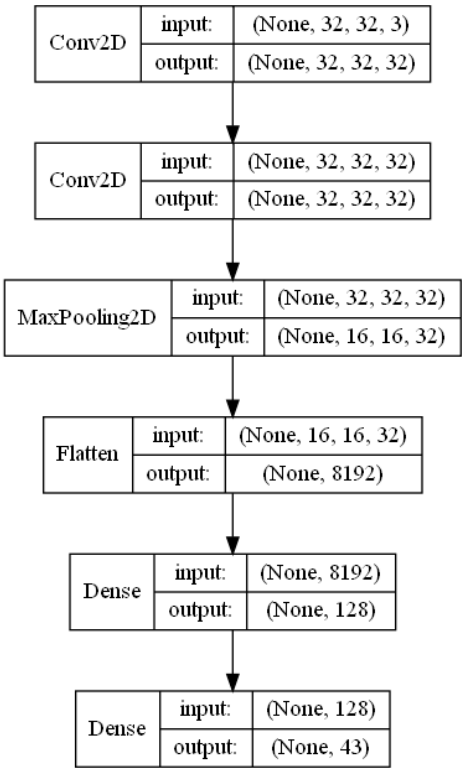

Figure 4.2. Baseline Model for GTSRB Data.

This is a very simple model including 2 convolutional layers that consist of 32 filters in each with 3x3 kernel size, 2x2 max pooling layer, one dense layer with 128 neurons and a classification layer. In classification layer softmax function was used and in the other layers ReLU activation function with He initialization was used as recommended [3]. We used adam optimizer and batch size 32 during iterations in each epoch. We calculated the loss according to cross entropy function and the use of cross entropy loss with softmax output improves the learning performance of models in terms of fast learning and robustness [2]. To recall it, the formula for ReLU activation function is again given in Equation (4.1):

$$r(x) = max(0, x). \tag{4.1}$$

Also, the formula for softmax classifier and the formula for cross entropy loss with

softmax classifier (with its special name, categorical cross entropy loss) are given below in Equations (4.2) and (4.3). This is the loss function to be minimized in our problem. In these formulas x_i is the CNN score for the related class of instance x. In the trials while choosing the model architecture we made our experiments with 10 epochs and used 5 models trained from scratch while giving the results and looked at our simple statistical analysis on training and validation accuracies:

$$s(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}, \tag{4.2}$$

$$L(x_i) = -log(\frac{e^{x_i}}{\sum_j e^{x_j}}). \tag{4.3}$$

During the experiments with CNN, we resized the images to 32x32 resolution and used 75% of the training data in training set and 25% of it in validation set in a stratified manner. Our class distribution plot for training, validation (val) and test data is given in Figure 4.3.

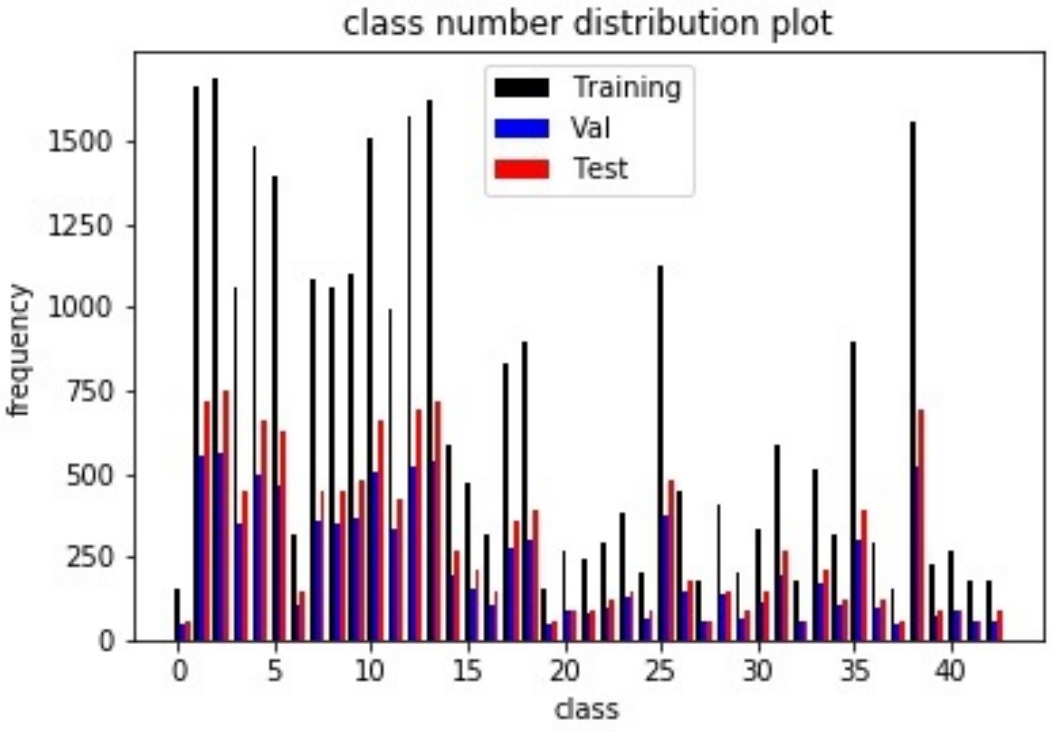

Figure 4.3. Class Distribution Plot for GTSRB Data.

As a preprocessing step, we simply normalized the pixel values to 0-1 scale from 0-255 scale. We did not make data cleaning like detecting and removing outliers in any classification experiment during this thesis. The reason is that because images are high dimensional data, in these kind of data, outlier detection becomes more complex and most existing outlier detection algorithms fail for this kind of datasets [65]. Moreover, at the end of the literature survey given at the beginning of the thesis, we also saw that researchers mostly forego outlier detection and data cleaning process in image classification studies. As an interesting and different outlier detection approach, another deep learning approach, usage of autoencoder at the basis and its enhancements may be a good idea while working with an image dataset. Autoencoders learn to reconstruct an image, and hence can classify those images as anomalies, for which the difference between the reconstructed image and the original image is larger than a threshold [65–67]. Some representative images for autoencoders are given in [68, 69].

Regarding the experiments, if we look at the results for our first and basic model given in Figure 4.2, the interval for our training accuracy became [0.9955, 0.9998] and for validation accuracy [0.9837, 0.9902]. We found the median value for validation accuracy as 0.9877 and validation error interval [0.0098, 0.0163]. Training time was found around 420 seconds for one model and the related codes for this model architecture is given in Figure A.1, in the Appendix. About the training time, training duration for each epoch is almost equal in these kind of networks and, here, it is around 42 seconds.

In our second trial, we added 2 more convolutional layers with 64 filters and 3x3 kernel size for each layer and a max pooling layer with the same principle following the max pooling layer of the previous model. We saw an improvement in the results with training accuracy interval [0.9956, 0.9988], validation accuracy interval [0.9882, 0.994] and validation accuracy median 0.9928.

In our next trial, we added 2 more convolutional layers with 128 filters and 3x3 kernel size for each layer and a max pooling layer with the same principle of the previous model. This time we found training accuracy interval [0.9725, 0.9974], validation accuracy interval [0.9669, 0.9933] and validation accuracy median 0.9871.

There is a performance decline at the end of the third trial. That's why we decided to continue with the second model. Adding extra layers may cause vanishing/exploding gradient problems in the chain rule because of consecutive multiplications of gradients and this may cause a lower performance model [70]. Our case may be an example for this situation. ResNet deep learning architecture was built as a precaution against this kind of a problem. It uses shortcut connections on convolutional blocks or layers like it is represented in [70] in addition to sequential data flow [70]. However, in this thesis, we use a VGGNet architecture at the basis and it works with sequential data flow.

We continued with the second model and we added some dropout to the second model and obtained training accuracy interval [0.9989, 0.9997], validation accuracy interval [0.9946, 0.9967] and validation accuracy median 0.9954.

We added batch normalization to the previous model and obtained training accuracy interval [0.9991, 0.9999], validation accuracy interval [0.995, 0.9972] and validation accuracy median 0.9964. Batch normalization is used to increase the stability of the neural network normalizing the output of a layer. In this way, generalization is improved and also a faster convergence of loss function is targeted [52].

We also added learning decay with the decay rate equal to the learning rate/epoch number (epoch number is 10) to the last model and found training accuracy interval [0.9998, 0.9999], validation accuracy interval [0.9972, 0.9978] and validation accuracy median 0.9974. Until now, we have always used 0.001 as the learning rate, but, here, we added the mentioned learning rate schedule. As stated before, learning rate schedule is used to decay the learning rate to avoid oscillation around the optimum point and to find that point properly as training continues. As it is said we used Adam optimization and a learning rate decay is already applied in Adam optimization, but usage of additional learning rate decay is deemed as a nice heuristics.

At the moment, the very last model is found to be the best model and it is given in Figure 4.4. In this Figure, the layer group on the right hand side is a sequel to the layer group on the left hand side and the two groups form the model together. We see that usage of dropout, batch normalization and learning decay influenced positively the performance of our model. This time, training time was found around 1200 seconds for one model and the related codes for this model architecture is given in Figure A.2. As an additional analysis, its validation error interval became [0.0022, 0.0028] and this is approximately one-fifth of the error interval found for the baseline architecture which is the first architecture.

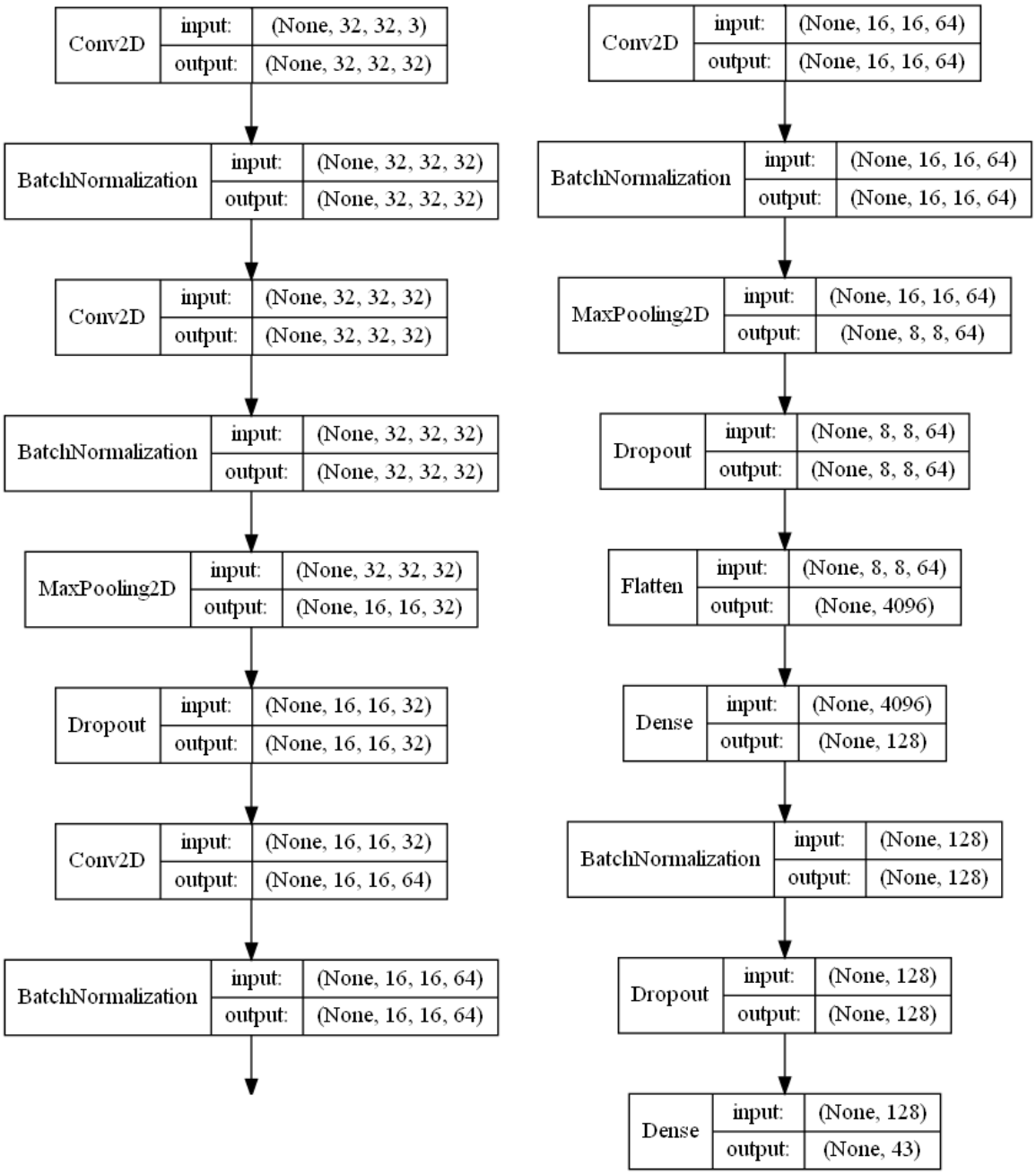

Figure 4.4. Our Model Found for GTSRB Data.

When we trained this model during 25 epochs and 5 times we found training accuracy is almost always 1, validation accuracy interval is [0.9976, 0.9983] and validation accuracy median is 0.9977. If we look at the test data, we saw that test accuracy interval is [0.9808, 0.9841] with median 0.9814. For our 5 different models, we obtained

the graphic in Figure 4.5.

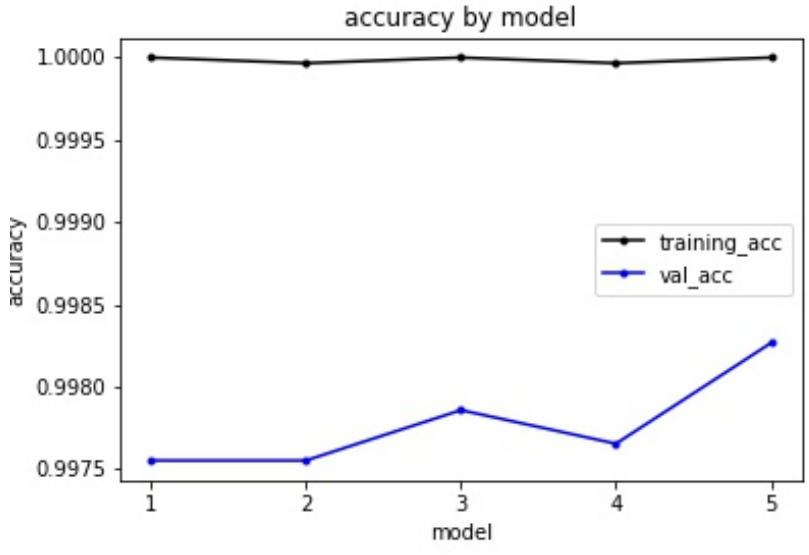

Figure 4.5. Accuracy Values by Model for GSRB Data.

As it can be seen in Figure 4.5, our 5 models are quite close to each other with regards to their classification accuracy performance on our data. To have an idea about their training history, Figure 4.6 can be examined. It is given for the fifth model whose validation accuracy is the best in Figure 4.5, but it can be intuitively thought that these training history graphics will be all very similar for these 5 models. In this kind of a training plot given in Figure 4.6, validation accuracy values are given according to the model obtained at the end of each epoch and whole validation data. Whereas, training accuracy values are given as a kind of average of accuracy values of all training batches and all different models found at the end of each iteration during each epoch.

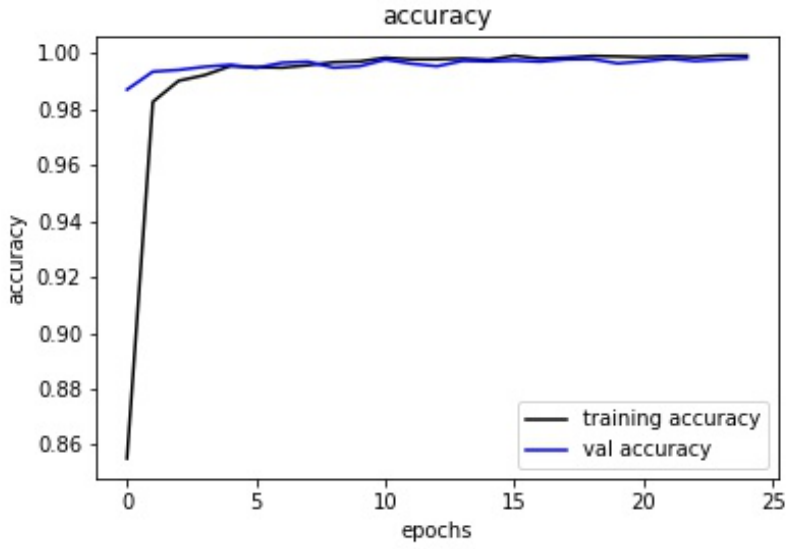

Figure 4.6. Training History for GTSRB.

After finding these models, we combined them in a proper way. These models are quite close to each other with regards to their classification accuracy performance on our data. However, because of the randomness that we discussed before in the nature of deep learning, each of them makes different progresses. As a result, in a class where one model is very successful, another model may not be that successful and while learning well the data overall, each model may specialize itself in a different class. For example, Figure 4.7, Figure 4.8 and Figure 4.9 show us the confusion matrices for misclassified samples in validation data. In these figures, we can see the rate of misclassified samples for each actual class and in which wrong class they are classified. These figures are drawn for the first, the third and the fifth models, respectively. We chose these models as the best ones, regarding training and validation accuracies. In the figures, lighter colors show that the related rate value increases and darker colors show that the related rate value is close to 0. The rate values are in [0, 0.02] interval approximately.

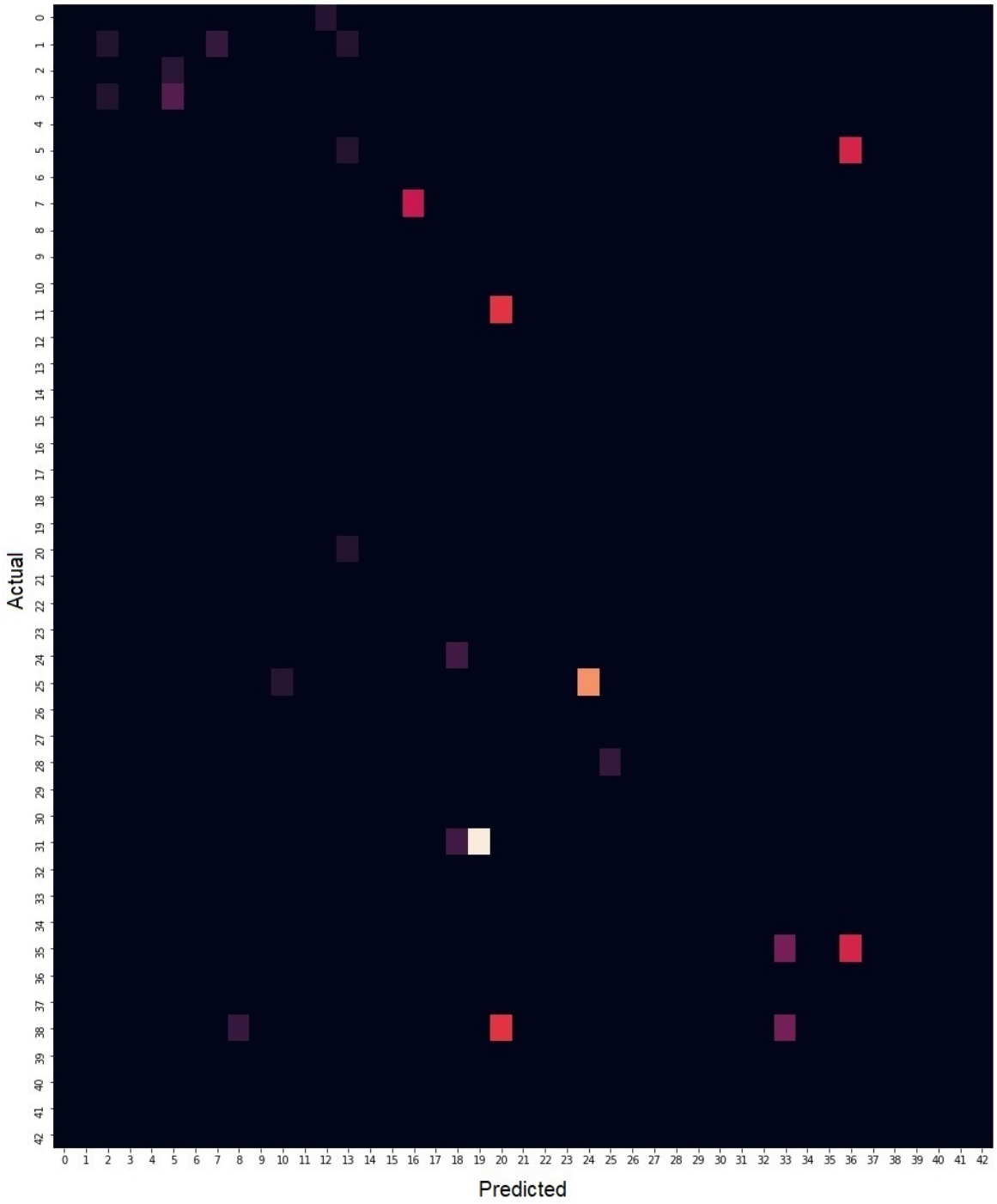

Figure 4.7. Misclassification Results for the First Model.

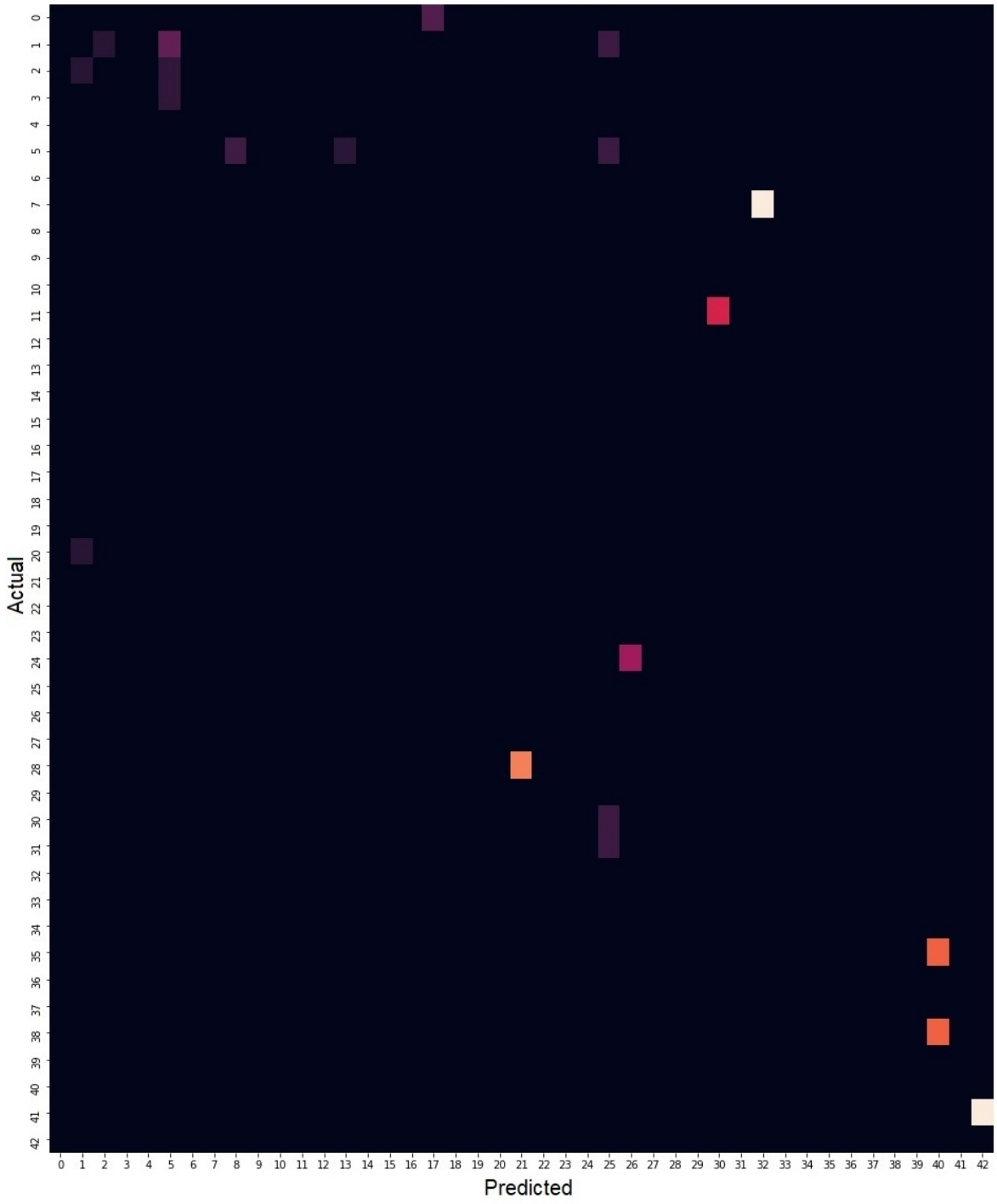

Figure 4.8. Misclassification Results for the Third Model.

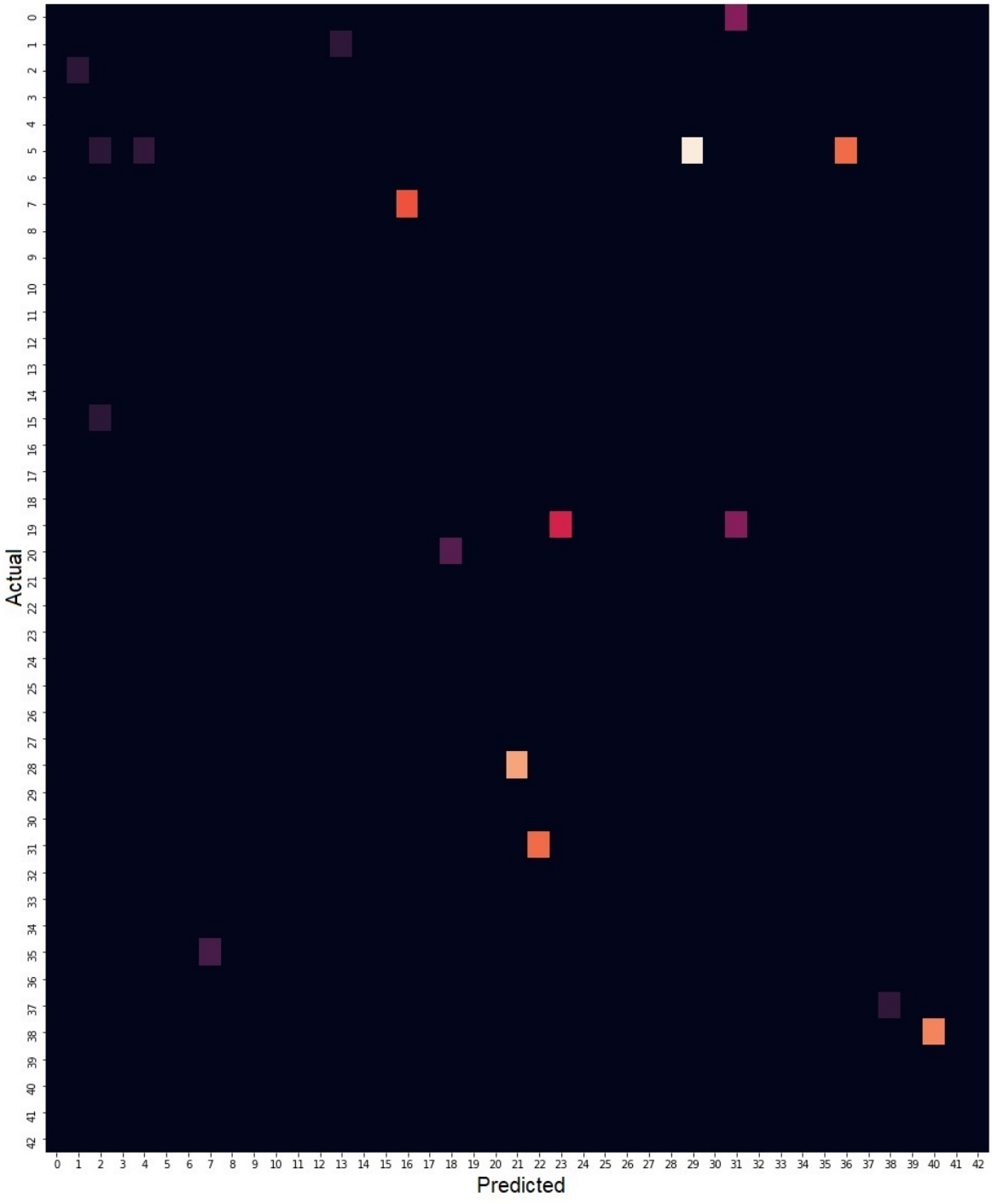

Figure 4.9. Misclassification Results for the Fifth Model.

When we look at the above figures, we see that there are some dissimilarities between these misclassification maps which means while one model is weaker for a

class another model can be strong for this class. In this way, different models can fix each others' mistake when they are together. Using this idea, we combined the 5 different models using a simple grid search mechanism. We used the simple grid search mechanism to determine the weights of each model and we determined the best weight combination according to the validation data results. We firstly defined a weight list, got its cartesian product 5 times (because we have 5 different models.) and normalized it to 1. To shorten the computations (to decrease the combination number), we used a simple weight list and we defined it as [0, 0.5, 1]. To explain via en example, if the cartesian product result becomes [0.5, 0.5, 1, 0, 1] then it is normalized to [0.17, 0.17, 0.33, 0, 0.33] by dividing each element in the product by their sum. In this way, we obtained each possible combination and tried each unique combination on validation data and chose the combination with the highest accuracy as optimum combination for the model weights. During these trials, we simply used models' prediction probability values for each sample and each class and multiplied them with the model weights and summed these scores of each model up. At the end, the class with the biggest score for a sample was determined as combined models' prediction for that sample.

Using this logic, we applied our grid search based model combination. At the end of our trial, we found that usage of the weights [0.2, 0.2, 0.2, 0.0, 0.4] respectively for our 5 models gives the best validation accuracy. In this way, the validation accuracy rose to 0.9991 and the related test accuracy rose to 0.9871. These validation and test results are better than the individual results of the 5 models and this shows that our model combination approach worked well.

In https://github.com/mertcetinkaya/master_thesis_codes, the related detailed confusion matrix for our models' combination with the determined weights and the test data can be found in images folder. In that figure, the colors change according to the sample number value written in the related cell. Larger numbers are shown with lighter color while smaller numbers are shown with darker colors.

In addition, Table 4.3 shows us the precision, recall and f-measure results for this

combination for each class, like a summary classification report. Precision, recall and f-measure (or f1-measure) metrics are widely used metrics in classification problems and they are used more often especially in binary classification problems for positive cases. However, they can be applied in multiclass classification problems as well and in these problems they can be calculated for each class. Equations (4.4), (4.5) and (4.6) show the formulas to calculate these metrics. We can simply think recall to be about detection rate and precision to be about false alarm rate (high precision points to less false alarms) for a class. Also, f-measure is a combination of precision and recall. All of these metrics are targeted to be as close to one as possible:

$$Precision = \frac{TP}{TP + FP}, \tag{4.4}$$

$$Recall = \frac{TP}{TP + FN}, \tag{4.5}$$

$$F - Measure = 2\frac{Precision * Recall}{Precision + Recall}. \tag{4.6}$$

The results given in Table 4.3 can also be extracted using the confusion matrix found in images folder of https://github.com/mertcetinkaya/master_thesis_codes.

Table 4.3. Classification Report for GTSRB Test Data.

class	**precision**	**recall**	**f-measure**	**class**	**precision**	**recall**	**f-measure**
0	1.0000	1.0000	1.0000	1	0.9638	0.9986	0.9809
2	0.9987	0.9987	0.9987	3	1.0000	0.9600	0.9796
4	1.0000	0.9894	0.9947	5	0.9488	1.0000	0.9737
6	1.0000	0.8800	0.9362	7	0.9781	0.9911	0.9845
8	0.9956	0.9978	0.9967	9	0.9856	0.9979	0.9917
10	0.9985	0.9939	0.9962	11	0.9767	0.9976	0.9870
12	0.9971	0.9971	0.9971	13	0.9972	0.9972	0.9972
14	1.0000	1.0000	1.0000	15	0.9953	1.0000	0.9976
16	1.0000	1.0000	1.0000	17	1.0000	0.9972	0.9986
18	0.9974	0.9897	0.9936	19	0.9836	1.0000	0.9917
20	0.8333	1.0000	0.9091	21	1.0000	0.9667	0.9831
22	0.9831	0.9667	0.9748	23	0.9868	1.0000	0.9934
24	1.0000	0.9778	0.9888	25	0.9650	0.9771	0.9710
26	0.9634	0.8778	0.9186	27	0.9091	0.6667	0.7692
28	0.9933	0.9933	0.9933	29	1.0000	1.0000	1.0000
30	1.0000	0.8867	0.9399	31	0.9890	1.0000	0.9945
32	0.9375	1.0000	0.9677	33	0.9953	1.0000	0.9976
34	1.0000	0.9917	0.9958	35	1.0000	1.0000	1.0000
36	1.0000	0.9917	0.9958	37	1.0000	1.0000	1.0000
38	0.9986	0.9986	0.9986	39	1.0000	1.0000	1.0000
40	0.9535	0.9111	0.9318	41	1.0000	0.8833	0.9381
42	0.9890	1.0000	0.9945				

Here, as an additional improvement proposal, Table 4.3 and the confusion matrix can be produced for validation data and some mechanism or algorithms for extra correction can be developed especially for the classes being mixed up with each other. In this way, the predictions can be optimized more and better classification results can

be obtained in test data. This kind of a logic is used in the OCR application in this thesis.

Table 4.4 shows us some results found by the other researchers. With regards to this table, we see that our approach works well and our test result which is 98.71% is between the best ones.

Table 4.4. GTSRB Result Comparison.

Research	**Method**	**Test Accuracy**
Yin *et al.* [15]	CNN	98.96%
Berger *et al.* [11]	VG-RAM WNN	98.73%
Gudigar *et al.* [12]	graph LDA	97.84%
Zaklouta *et al.* [13]	Random Forests	96.14%
Wang and You [14]	Boosting SVM	93.6%
Zaklouta *et al.* [13]	K-d Trees	92.70%

4.1.2. Belgium Traffic Sign Classification Dataset

Timofte *et al.* [71] generated the Belgium Traffic Sign dataset using a van with eight roof-mounted cameras capturing images every meter. BTSC dataset is a subset of that dataset and BTSC dataset contains cropped images for 62 different classes of traffic signs. There are 4,575 images in the training set and 2,520 images in the test set. The resolution of the images changes between 22x21 to 674x527 [64].

Some examples from this dataset can be seen in Figure 4.10.

Figure 4.10. Examples from BTSC Dataset.

4.1.2.1. Experiments. The average of width and height length for the images in our training data is approximately 100 pixels. We resized the images to 32x32 and, thinking the same problem nature, used the best model found for GTSRB dataset, and repeated our experiments with the same principle: 5 times and 25 epochs in each training. We found a training accuracy in interval [0.9971, 1.0], validation accuracy in interval [0.972, 0.9904] and validation accuracy median 0.9869. For our 5 different models, we obtained the graphic in Figure 4.11. When we choose the best one (the third one) according to training and validation results between these 5 models, we found 0.9758 test accuracy. (All test scores: 0.9659, 0.9726, 0.9758, 0.9635, 0.9694 respectively)

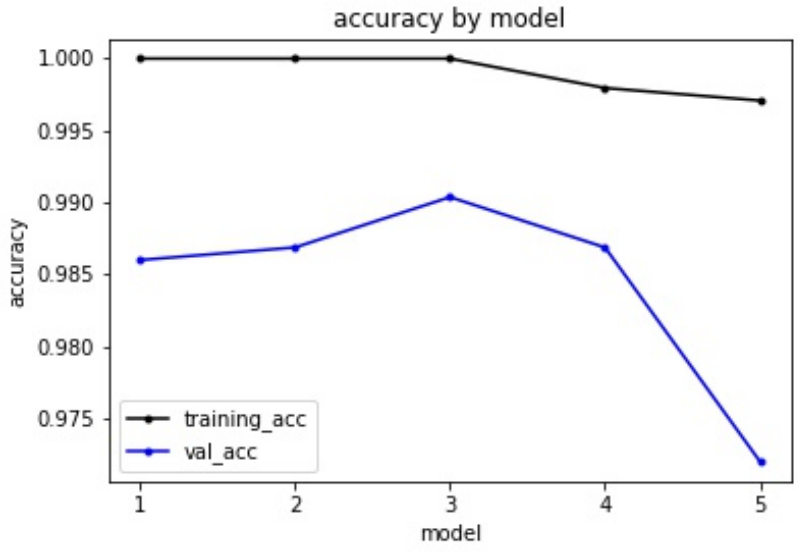

Figure 4.11. Accuracy Values by Model for BTSC Data.

Here, we again used 25% of training data in validation data and the rest of it was used in training data. The data split was done in a stratified manner. Our class distribution plot for training, validation (val) and test data is given in Figure 4.12.

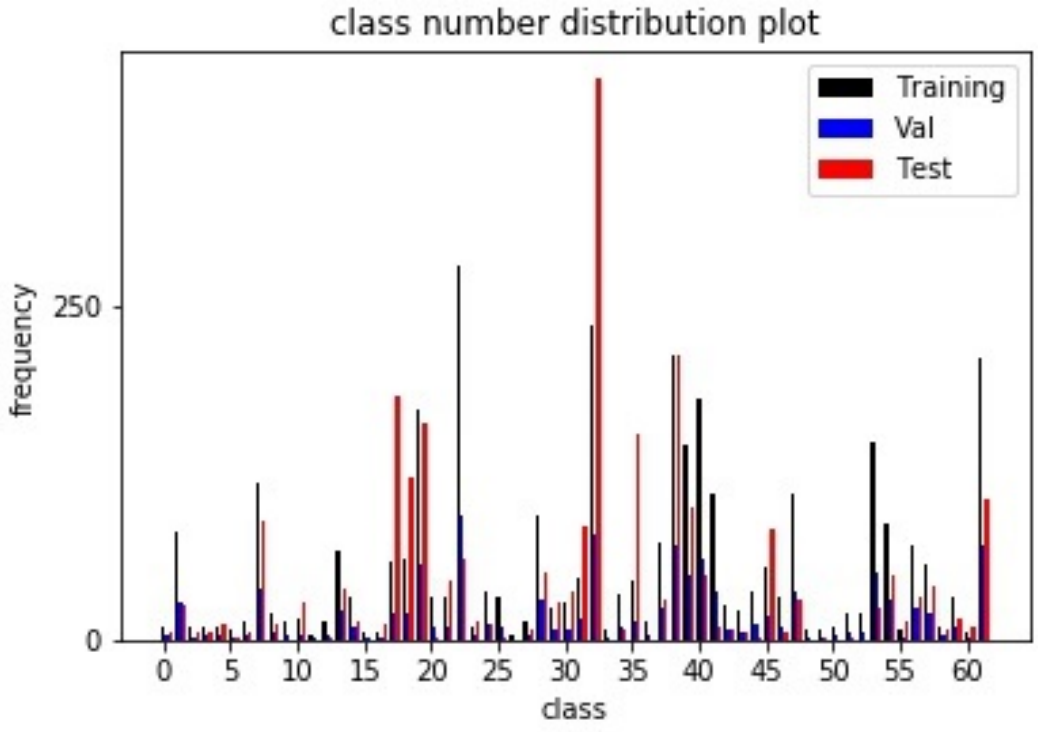

Figure 4.12. Class Distribution Plot for BTSC Data.

It is obvious that the BTSC dataset is a bit small (especially when compared to the GTSRB dataset). One way to improve accuracy can be data augmentation [2]. Moreover, we see that although it is the same problem type, especially the validation results change in a comparatively larger interval (especially for the last validation result) than for GTSRB. This possibly indicates that the model structure allows some improvements as there is an insufficient and unstable learning. We therefore increased the number of kernels in each layer; training during more epochs can be another idea. Instead of 32 and 64 kernels in convolutional layers, we used 256 and 512 kernels and obtained a higher level and more intelligent system that is able to extract more information from this limited dataset. This network obtained [0.9985, 1] training accuracy, [0.9878, 0.9965] validation accuracy and 0.9939 as validation accuracy median for 5 models and 25 epochs of training for each model. For these models, we obtained the graphic in Figure 4.13. With regards to this graph, we see that this time validation

accuracy changes in a smaller interval with a smaller standard deviation and validation results got better as a sign of better training. We also see that the first one seems to be the best model and when we choose this model we get 0.9825 test accuracy (All test scores: 0.9825, 0.9813, 0.9829, 0.9837, 0.9674 respectively).

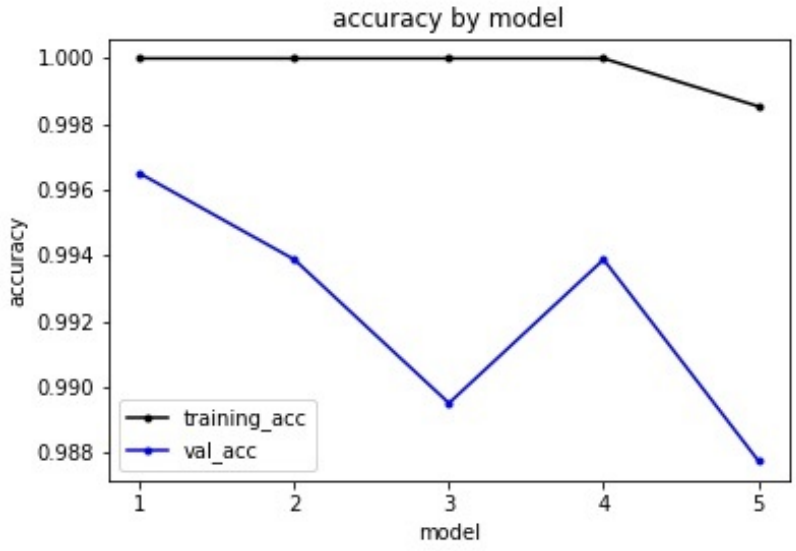

Figure 4.13. Final Graphic for Accuracy Values by Model for BTSC Data.

The training history for this model (the first model) is given in Figure 4.14. This model is found as the best model.

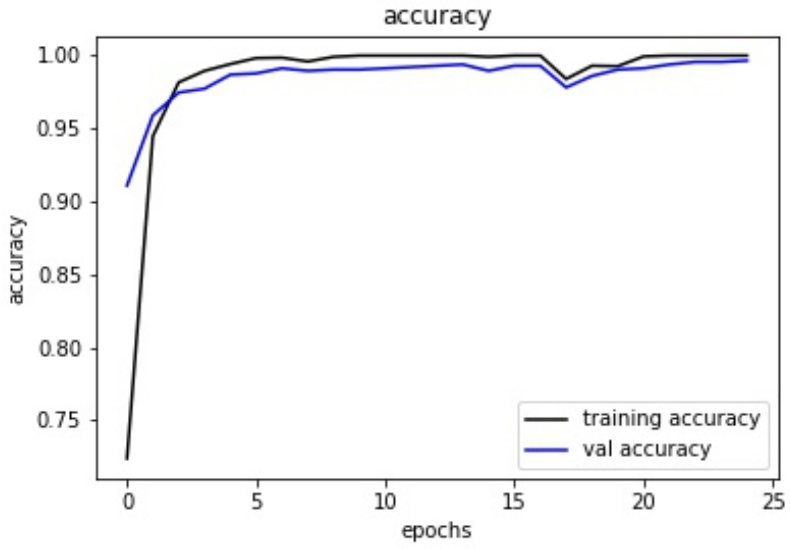

Figure 4.14. Training History for BTSC.

Related detailed confusion matrix and brief precision, recall and f-measure results are given in https://github.com/mertcetinkaya/master_thesis_codes (in images folder) and in Table 4.5 for test data.

Table 4.5. Classification Report for BTSC Test Data.

class	**precision**	**recall**	**f-measure**	**class**	**precision**	**recall**	**f-measure**
0	1.0000	1.0000	1.0000	1	1.0000	1.0000	1.0000
2	1.0000	0.7143	0.8333	3	1.0000	0.6667	0.8000
4	0.8571	1.0000	0.9231	5	1.0000	1.0000	1.0000
6	0.7143	0.8333	0.7692	7	0.9886	0.9667	0.9775
8	0.9231	1.0000	0.9600	10	1.0000	1.0000	1.0000
12	1.0000	1.0000	1.0000	13	1.0000	0.9744	0.9870
14	0.9375	1.0000	0.9677	16	1.0000	1.0000	1.0000
17	0.9839	1.0000	0.9919	18	0.9917	0.9754	0.9835
19	0.9939	1.0000	0.9969	20	1.0000	1.0000	1.0000
21	1.0000	1.0000	1.0000	22	1.0000	1.0000	1.0000
23	1.0000	1.0000	1.0000	24	1.0000	0.7692	0.8696
25	0.5000	1.0000	0.6667	27	1.0000	1.0000	1.0000
28	1.0000	1.0000	1.0000	29	1.0000	1.0000	1.0000
30	0.9737	1.0000	0.9867	31	1.0000	1.0000	1.0000
32	1.0000	0.9976	0.9988	34	1.0000	1.0000	1.0000
35	0.9931	0.9351	0.9632	37	0.9688	1.0000	0.9841
38	0.9631	0.9812	0.9721	39	0.9899	0.9899	0.9899
40	0.9796	1.0000	0.9897	41	1.0000	1.0000	1.0000
42	1.0000	1.0000	1.0000	43	1.0000	1.0000	1.0000
44	1.0000	1.0000	1.0000	45	0.9625	0.9167	0.9390
46	0.4286	0.5000	0.4615	47	0.9688	1.0000	0.9841
49	0.4000	0.6667	0.5000	51	0.7500	1.0000	0.8571
53	0.9600	1.0000	0.9796	54	0.9796	1.0000	0.9897
55	1.0000	1.0000	1.0000	56	0.9429	1.0000	0.9706
57	1.0000	0.9512	0.9750	58	1.0000	1.0000	1.0000
59	1.0000	1.0000	1.0000	60	1.0000	1.0000	1.0000
61	1.0000	1.0000	1.0000				

Table 4.6 shows some performance results reported in the literature. With regards to this table, we see that our approach works well and our test result which is 98.25% is quite close to the best ones.

Table 4.6. BTSC Result Comparison.

Research	**Method**	**Test Accuracy**
Gudigar *et al.* [12]	graph LDA	98.50%
Aziz *et al.* [18]	EML	98.30%
Mehta *et al.* [16]	CNN	97.06%
Lu *et al.* [17]	Graph	96.29%

4.1.3. Folder and File Structure for Traffic Sign Recognition

Figure 4.15 shows the folder and file structure used in traffic sign recognition experiments. In hog and hog_test folders the produced hog dataframes are found for training and test data. In Train folder, there are folders with class numbers and in each of these class folders there are related training images. In Test folder there are test images. Test.csv file contains image and class id information for test data. The codes can be reached using the link https://github.com/mertcetinkaya/master_thesis_codes and this folder and file structure must be created to run the codes properly. The readme file on github contains additional information.

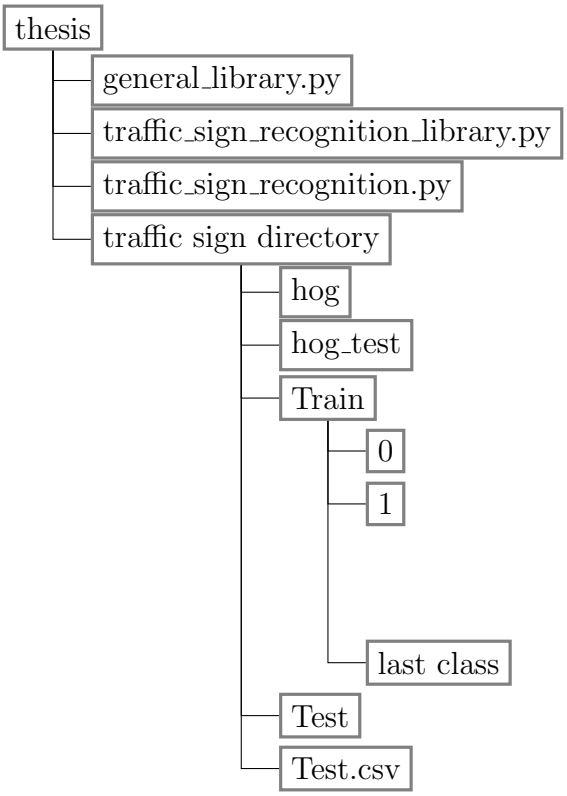

Figure 4.15. Traffic Sign Recognition Folder and File Structure.

4.2. Medical Image Recognition

Here we describe the experiments we made for malaria recognition and breast cancer recognition.

4.2.1. Malaria Recognition

Malaria is an infectious disease that causes more than 400,000 deaths per year and it is a true endemic in some areas of the world. This means that the disease is regularly found in the region. Especially, the middle parts of Africa mostly suffer from this disease and it is a mosquito-borne blood disease caused by the Plasmodium parasites transmitted through the bite of the female Anopheles mosquito. The dataset that we used for malaria recognition is the same dataset that Rajaraman *et al.* [19] used in their publication in 2018.

In their study Rajaraman *et al.* [19] used a dataset that consists of 27,558 blood cell images with equal instances of parasitized and uninfected cells. The minimum, average and maximum values for width and height of images are around 40, 130 and 390 pixels respectively and images are in a square-like form.

Examples for uninfected (negative) and parasitized (positive) cells can be seen in Figure 4.16 and in Figure 4.17.

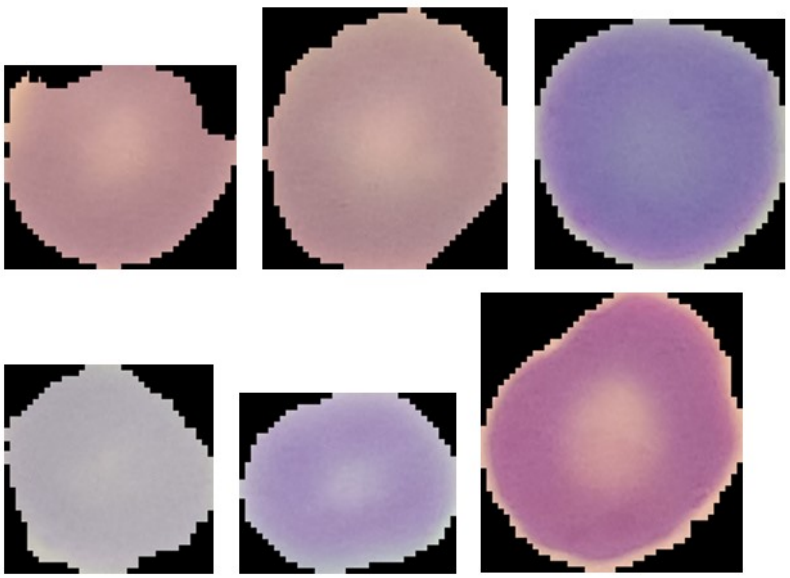

Figure 4.16. Examples of Uninfected Cells.

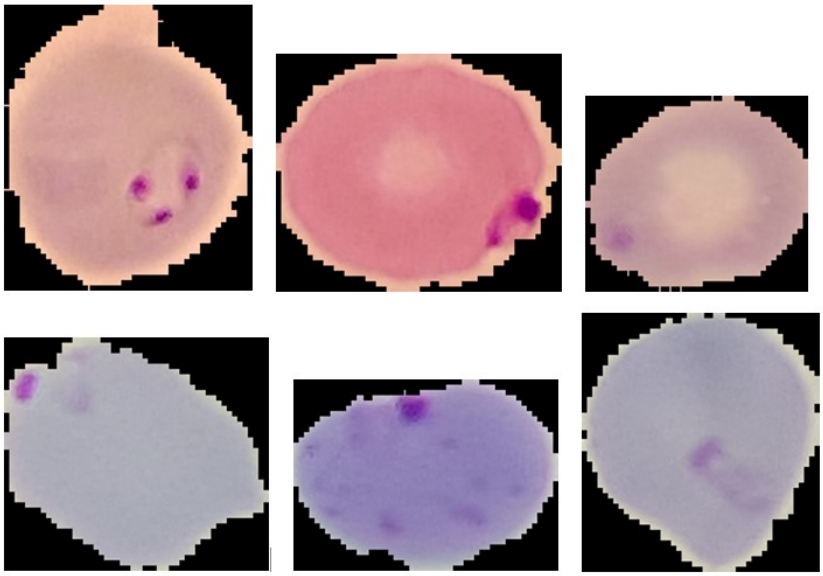

Figure 4.17. Examples of Parasitized Cells.

4.2.1.1. Experiments with Classical Methods. In our experiments, we used 80% of the data for training and 20% of the data for test making a stratified data split. In their study, in 2019, Rajaraman *et al.* [20] reported an improvement to the study in 2018 using the same dataset and in this second study they used images resizing them to 100x100 pixels. Regarding this situation and real sizes of the images, in our experiments we also resized the images to 100x100 pixels and extracted HOG features for PPC 5, 10 and 15 fixing cell per block to 2 and orientation to 9. In this way, we extracted HOG features with sizes 12,996, 2,916 and 900 for PPC 5, 10 and 15, respectively.

Table 4.7 shows the training results. In these experiments, the same basic initial properties (like using the same penalty term etc.) used in GTSRB experiments were given to the classifiers. Table 4.8 shows the test results.

Table 4.7. Malaria Recognition Training Results.

	PPC 5	**PPC 10**	**PPC 15**
LR Accuracy	0.89	0.80	0.76
SVM Accuracy	0.96	0.83	0.78
NB Accuracy	0.68	0.72	0.73
ANN Accuracy	1	0.99	0.96
LR Training Time (s)	28	8	2
SVM Training Time (s)	229	29	4
NB Training Time (s)	4	0.9	0.26
ANN Training Time (s)	130	37	18

Table 4.8. Malaria Recognition Test Results.

	PPC 5	**PPC 10**	**PPC 15**
LR Accuracy	0.72	0.77	0.75
SVM Accuracy	0.72	0.78	0.76
NB Accuracy	0.66	0.71	0.72
ANN Accuracy	0.72	0.82	0.85

With regards to Table 4.7 and Table 4.8 we see that again ANN is the best classifier of all in terms of accuracy. Also, SVM seems better than LR in terms of accuracy with a small difference, but its training time increases a lot when dimension size gets bigger. From this point of view, LR seems a better choice than SVM. NB again works quite fast, but it gives again the worst results. In addition, we see that accuracy performance difference between NB and other methods is smaller than for the GTSRB dataset. Here, it is also interesting that the test performance increases and training performance decreases in most of the times as level of extracted detail decreases. It seems

that this may be because of the 'curse of dimensionality' and decreasing dimension number in input data may increase classifier performance. It seems that the 'curse of dimensionality' causes some problems and may prevent classifiers from showing their best performance. Thus, the mentioned dimensionality reduction seems to help the classifiers for this reason. In this case, accuracy performance difference between NB and others could increase as well.

Moreover, as another interesting point, when we look at Figure 4.16 and Figure 4.17 we see that there exists one or more darker blobs of colour when a cell is parasitized. Searching for these blobs and trying to extract them can be an idea to decide if a cell is parasitized or not. In order to do so, some classical image processing techniques like Canny Edge Detection [72], Hough Transformation [73] or some colour space transformation between RGB and HSV or RGB and HSI can be used. For example, in Figure 4.18 Canny Edge Detection algorithm is run on a parasitized blood cell and the darker blob is detected and a bounded box is drawn around this activated area. In addition, in Figure 4.19 RGB colour space is transformed to HSV and a proper mask is produced. Using this mask, the blob around which a bounded box is drawn is detected. During these operations, some parameter tuning is required and some noise reduction like blurring can also be applied.

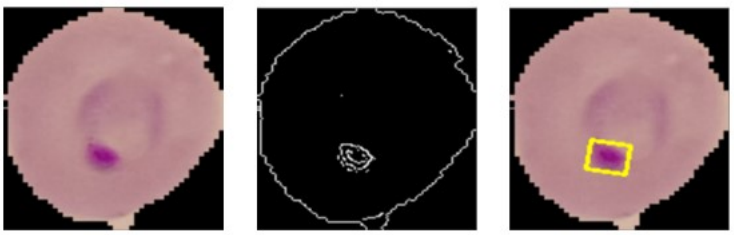

Figure 4.18. Canny Edge Detection.

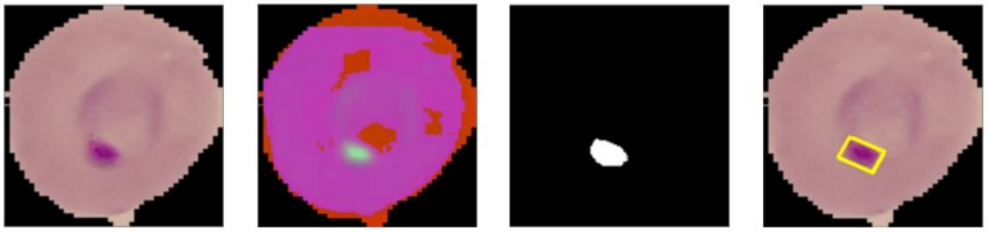

Figure 4.19. HSV Transformation.

However, searching for a pattern like a blob, as it is in this example, is not easy with standard image processing approaches for this kind of classification problems. This process requires some parameter tuning which is not easy and our experimental results show that because of the variations sourced by colour, shape, size etc., this tuning operation gets quite difficult and fragile in order to find a general rule across all data. Obviously, it does not present an efficient learning and generalization. As a much more automatic and successful algorithm, deep learning can be used again.

In medical image analysis, searching for a specific area like diseased area and usage of deep learning in this context is a popular research area. In modern research, this is actually a topic of semantic image segmentation which means assigning class label for each pixel and it allows pixel-wise classification in an image [74]. Architectures like UNet [75] and ENet [76] are nice examples for semantic and medical image segmentation. According to some research, in addition to their success, they can work fast and their results are quite promising. It seems that in the future their usage will become widespread in hospitals as a successful and much faster approach than experts or radiologists for segmentation of specific areas, diseased areas, tumors etc. in medical images or films. Using these architectures successful and detailed segmentations can be made [28, 77, 78].

For example, Karimov *et al.* [78] used UNet, ENet and their customized ENet (BoxENet) for semantic segmentation of mast cells in histological slice scans and they showed that deep learning architectures work quite well for this task.

These deep learning based techniques can also be applied to the malaria recognition problem, at least as a decision support approach for more advanced level studies. In the following section, we propose our classification experiments with deep learning following a similar procedure as for traffic sign recognition.

4.2.1.2. Experiments with Deep Learning. In this section we again used the CNN + ANN combination and used a procedure similar to that we used for GTSRB dataset while improving the neural network model. We again used the same training and test data as for the HOG feature experiments above, but this time we used 25% of the training data as validation data. This means that we used 20% of the whole data in test, 20% of the whole data in validation and 60% of the whole data in training. In our experiments we always splitted the data in a stratified way and resized the images to 32x32 resolution. Our class distribution plot for training, validation (val) and test data is given in Figure 4.20. Also, a difference about model training is that the classification layer of this model is for two outputs because of the problem nature.

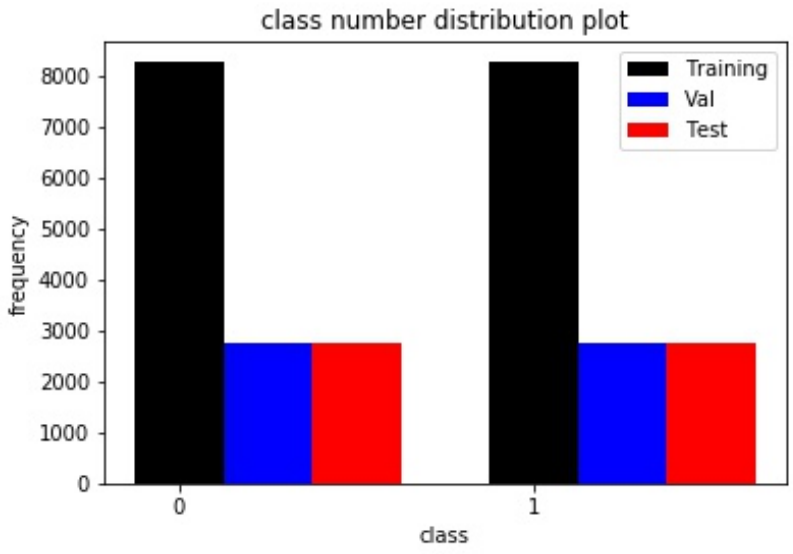

Figure 4.20. Class Distribution Plot for Malaria Recognition.

Table 4.9 shows us the results for step-by-step model development. We repeated

the experiments from scratch 5 times and during 10 epochs in each time in Table 4.9.

Table 4.9. Malaria Recognition Model Development.

Model	**Training Accuracy Inteval**	**Validation Accuracy Interval**	**Validation Accuracy Median**
Baseline	[0.9693, 0.9933]	[0.9196, 0.9441]	0.9434
Baseline +2 Conv.	[0.9736, 0.9823]	[0.949, 0.959]	0.955
Baseline +4 Conv.	[0.9604, 0.975]	[0.9478, 0.9601]	0.9583
Baseline +4 Conv. +Dropout	[0.9573, 0.9669]	[0.9485, 0.9603]	0.9528
Baseline +2 Conv. +Dropout	[0.9683, 0.9707]	[0.9577, 0.961]	0.9594
Baseline +2 Conv. +Dropout +Batch Norm.	[0.9386, 0.9687]	[0.9376, 0.9563]	0.9548
Baseline +2 Conv. +Dropout +Learn. Decay	[0.9685, 0.9736]	[0.955, 0.9614]	0.9583

Here, we chose the architecture with baseline + 2 convolutional layers + dropout that is shown in Figure 4.21 and when we trained this model during 25 epochs and 5 times we found training accuracy interval is [0.9877, 0.9921], validation accuracy interval is [0.9581, 0.9606] and validation accuracy median is 0.9595.

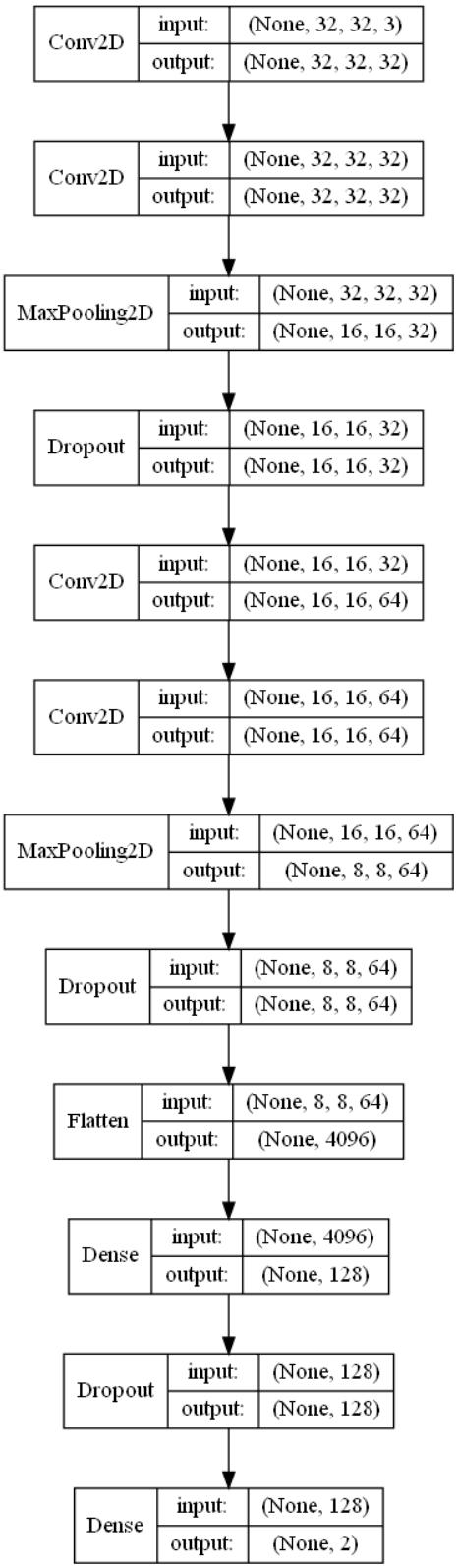

Figure 4.21. Our Model for Malaria Recognition.

For our 5 different models, we obtained the graphic presented as Figure 4.22.

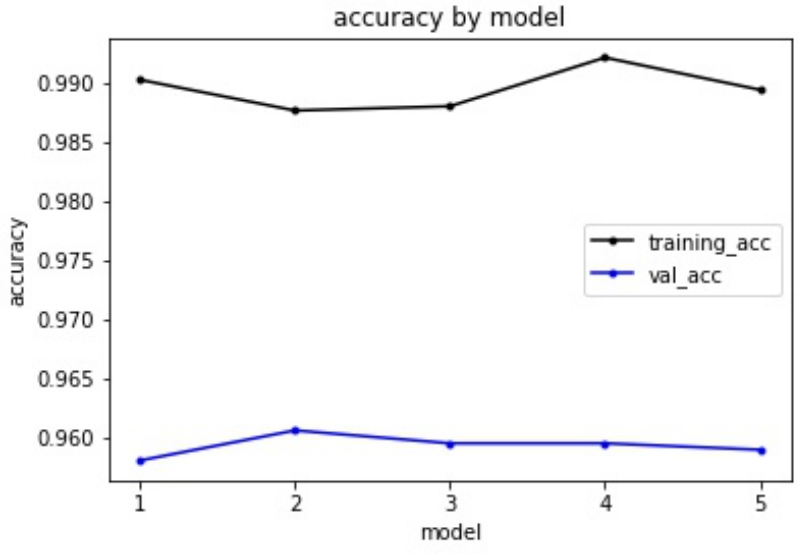

Figure 4.22. Accuracy Values by Model for Malaria Data.

In order to augment the success of our models, we applied data augmentation on our training data. Data augmentation is an important approach in deep learning to improve the learning performance. For the data augmentation, we blurred, rotated by 90 degrees in clockwise and counter clockwise and rotated by 180 degrees each of the images and we increased the amount of training data 4 times. As a representative image for data augmentation, in Figure 4.23, we see the original image on the leftmost and the others are produced under data augmentation concept.

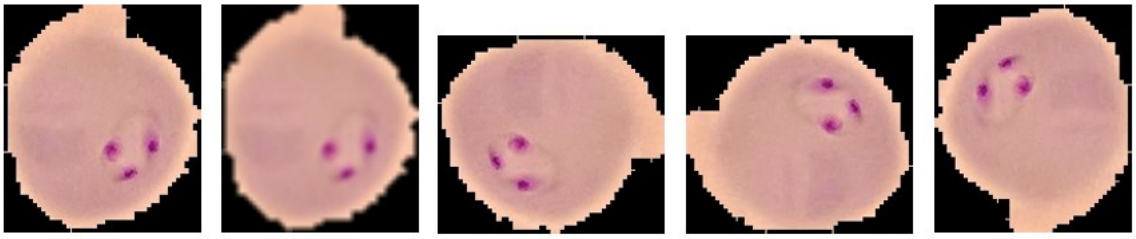

Figure 4.23. Data Augmentation for Malaria Data.

After this data augmentation, we trained the chosen model architecture using the same (the previous) principal from scratch and obtained the graphic in Figure 4.24. The training accuracy interval is [0.9847, 0.9871], validation accuracy interval is [0.9628, 0.9669] and validation accuracy median is 0.9644. We see that there is an improvement in validation scores and a decline in training scores. However, the improvement in validation scores shows us a better learning. We see that data augmentation acted like a regulator against overfitting that prevented our model from memorizing and it provided a better generalization.

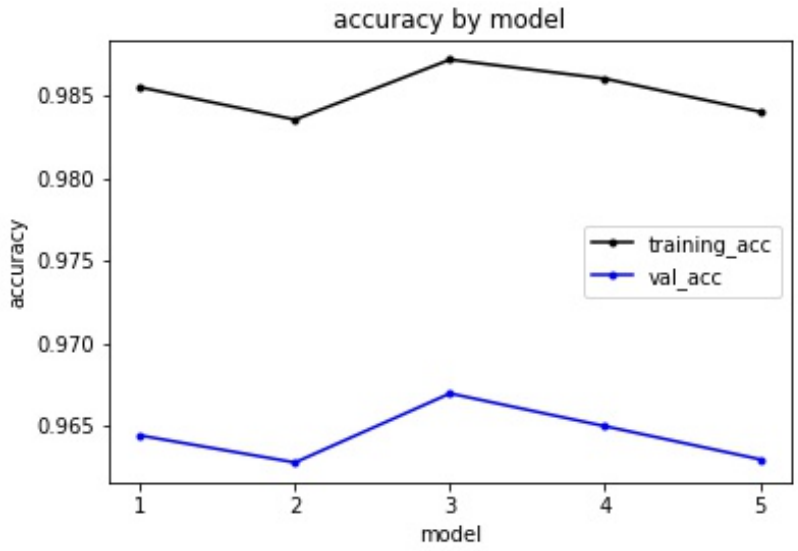

Figure 4.24. Training History for Malaria Data after Data Augmentation.

With regards to Figure 4.24, we see that there is an improvement in validation scores thanks to data augmentation and the third model is the best one to choose. The test accuracy for this model is 0.9663 (all test scores: 0.9628, 0.9637, 0.9663, 0.9634, 0.9643). The confusion matrix of this model for test data is presented in Figure 4.25.

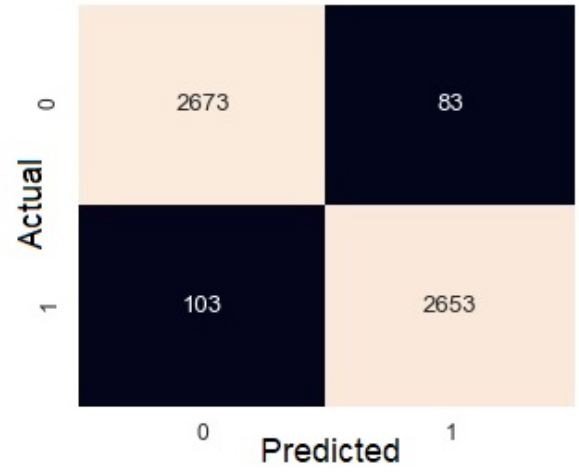

Figure 4.25. Confusion Matrix for Malaria Test Data.

We obtained precision, recall, f-measure and balanced accuracy values as 0.9696, 0.9626, 0.9661 and 0.9663 for the test data. A metric often used in binary classification is the arithmetic mean of the recall results for the 2 classes. It is called balanced accuracy metric. As a convention, we gave our recall results for positive class in this binary classification problem, but it can be calculated for both classes. In this way, these recall calculations get more special names like sensitivity and specificity. The sensitivity formula is used for recall calculation for positive class. The formulas for these new metrics are given in Equations (4.7), (4.8) and (4.9):

$$Sensitivity = \frac{TP}{TP + FN}, \tag{4.7}$$

$$Specificity = \frac{TN}{FP + TN}, \tag{4.8}$$

$$Balanced\ Accuracy = \frac{Sensitivity + Specificity}{2}. \tag{4.9}$$

We see that as our data are balanced the balanced accuracy result are equal to

the accuracy result. These results can also be obtained via Figure 4.25. The training history for this model (the third model) is given in Figure 4.26.

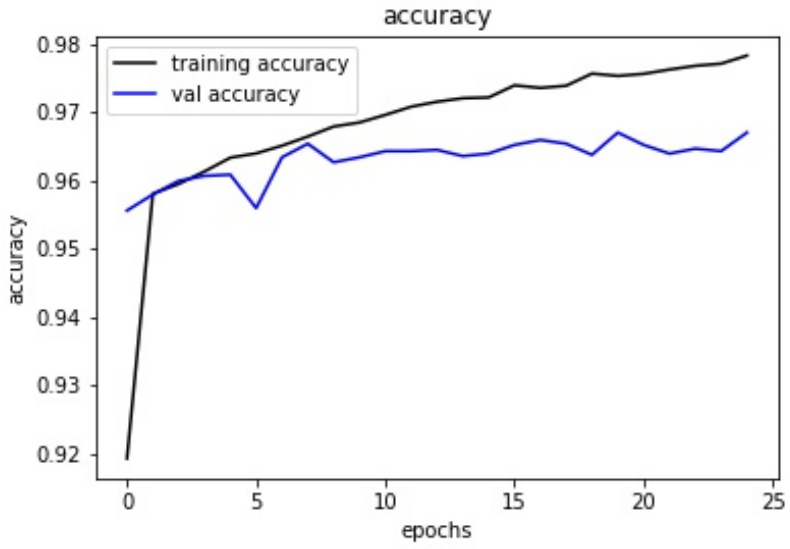

Figure 4.26. Training History for Malaria after Data Augmentation.

Table 4.10 shows us some results found by the other researchers. However, here, it is necessary to say that the training and test data that we used are different than the ones used in the other researches (because they are not given in the dataset) and this separation between training and test data are not always clearly explained in the papers. That's why, this comparison is not 100 percent fair but gives only a general idea. With regards to this table, we see that our approach works well and our test result which is 96.63% is one of the best.

Table 4.10. Malaria Recognition Result Comparison.

Research	**Method**	**Test Accuracy**
Rajaraman *et al.* [20]	Transfer Learning CNN	99.32%
Rajaraman *et al.* [20]	Custom CNN	99.09%
Qayyum *et al.* [21]	Custom CNN	96.05%
Akilotu *et al.* [22]	Transfer Learning CNN	96%
Rajaraman *et al.* [19]	Transfer Learning CNN	95.7%
Rajaraman *et al.* [19]	Custom CNN	94%

In this kind of a learning problem, it is obvious that the cost of missing a disease would be quite high. That's why, producing a model that gives more importance to detect as many positive cases as possible is desirable. This means that the recall metric is more important than the precision metric. In order to obtain this kind of a model, it is possible to: increase the number of positive cases compared to the number of negative cases used in the training set or to increase the class weight of positive cases during the training. An alternative is also to use a different probability threshold.

Here, f-measure can also be a nice metric and calculating an f-measure where the weight of the recall metric is larger is a nice approach. The best point between precision and recall can be determined using aforementioned techniques and this kind of an f-measure. This f-measure is also called f-beta measure and the f-measure (or f1-measure) that has been used so far is actually a special f-beta measure where beta is 1. The formula of the f-beta measure is given in Equation (4.10). Three common values for beta are 0.5, 1 and 2 and by changing the beta value, the weights of precision and recall in the f-beta measure calculation can be adjusted:

$$F - beta\ Measure = \frac{(1 + beta^2) * Precision * Recall}{beta^2 * Precision + Recall}. \tag{4.10}$$

Here, we directly continued from the last model and made some additional experiments moving the probability threshold. Normally, the probability threshold is 0.5 while deciding between positive and negative cases. However, there is always a trade-off between precision and recall and in this kind of problems recall is a more important metric because the cost of missing a positive case is much higher than the cost of giving a false alarm for a negative case. That's why, equating beta parameter to 2, we defined our f-beta measure and increased the importance of recall in the f-measure calculation. Normally, the f-measure uses a beta parameter equal to 1, which means that precision and recall have the same weights. After defining the f-beta measure in this new way we drew the precision-recall curve and found the best probability threshold in the training data equal to 0.2533 to obtain the biggest f-beta measure possible. This probability threshold is for positive cases and our precision-recall curve and the best point according to the f-beta measure for this precision-recall curve is given in Figure 4.27.

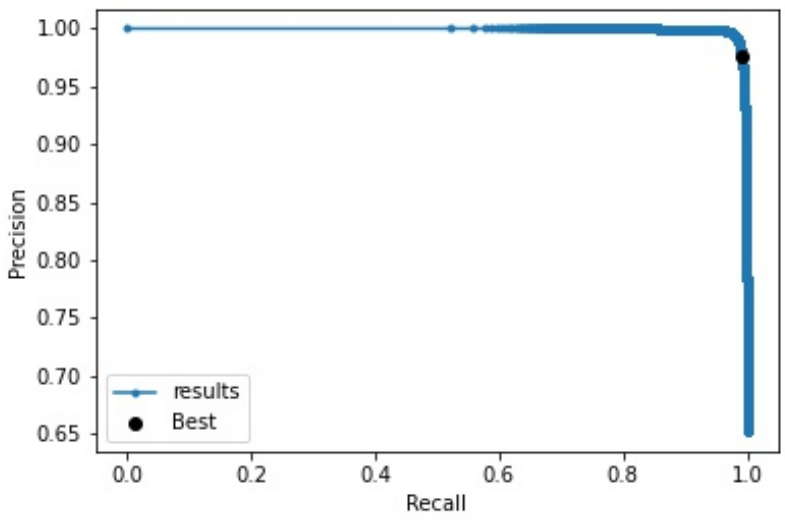

Figure 4.27. Precision-Recall Curve.

As it can be seen in Figure 4.27, the best point for a probability threshold is in the favour of making the recall value greater than the precision and this probability was found to be 0.2533 for the training data.

After defining the best probability threshold in this way we obtained the results for precision, recall and f-beta measure (see Table 4.11).

Table 4.11. Malaria Recognition Precision, Recall and F-Beta Measure Results.

	Precision	**Recall**	**F-Beta Measure**
Training	0.9762	0.9915	0.9884
Validation	0.9502	0.9706	0.9664
Test	0.9499	0.9716	0.9672

For this probability threshold, we obtained 0.9837 training accuracy, 0.9599 validation accuracy and 0.9602 test accuracy. We see that there is a decline in the accuracy scores, but our model is more successful in not missing positive cases. Our validation recall rose to 0.9706 from 0.9641 and our test recall rose to 0.9716 from 0.9626.

4.2.2. Breast Cancer Recognition

The dataset that we used in this experiment is for Invasive Ductal Carcinoma (IDC) which is the most common type of breast cancer. It originally was studied by Cruz-Roa *et al.* [23] and it consists of 162 whole mount slide images of Breast Cancer specimens scanned at 40x. From that, 277,524 patches of size 50x50 were extracted with 198,738 IDC negative and 78,786 IDC positive.

Examples for IDC negative and IDC positive are presented in Figures 4.28 and 4.29.

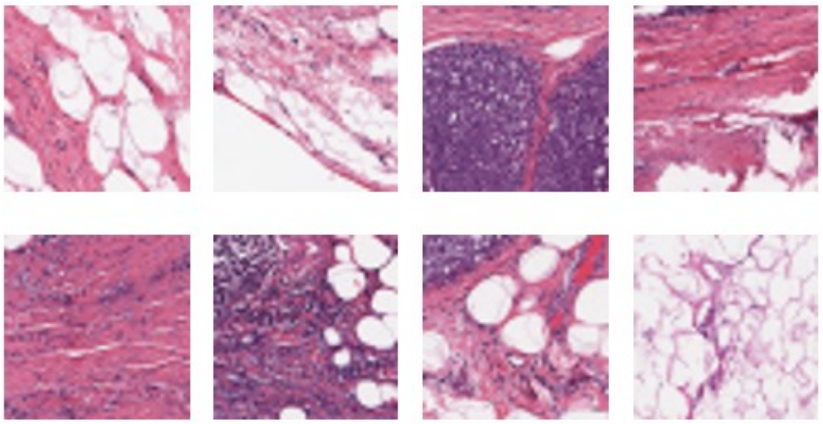

Figure 4.28. Examples for IDC Negative.

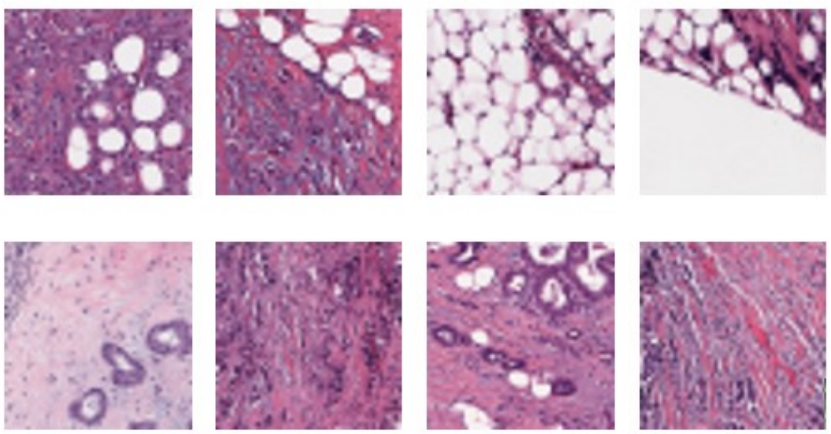

Figure 4.29. Examples for IDC Positive.

4.2.2.1. Experiments. In our experiments, we again used 60% of the whole data for training, 20% of the whole data for validation and 20% of the whole data for test and we made this splitting in a stratified manner between negative and positive classes. We resized our images to 32x32 resolution and made a downsampling to equate the number of positive and negative samples in the training set after the aforementioned data splitting. At the end, we ended up with 94,542 samples with equal positive and negative cases in our training set. In the validation set, we used 55,505 samples and in

the test set we used 55,506 samples. Our class distribution plot for training, validation (val) and test data is given in Figure 4.30.

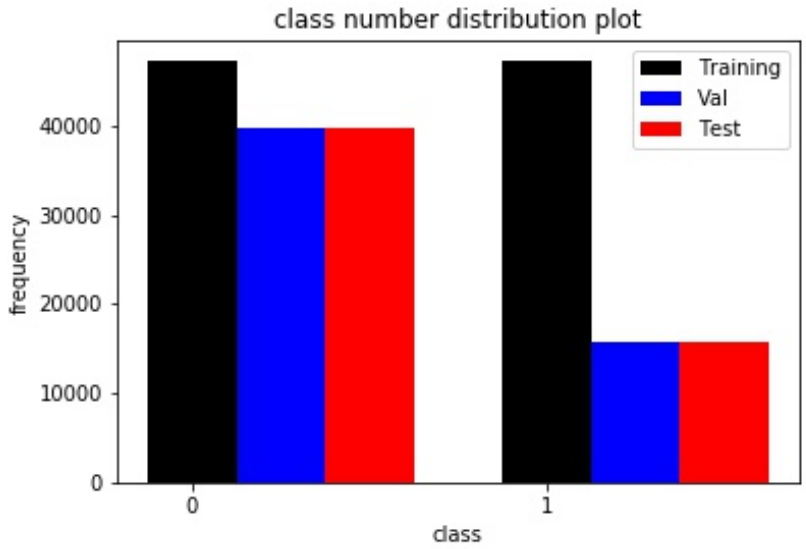

Figure 4.30. Class Distribution Plot for Breast Cancer Recognition.

As the two problems seem similar, we used the best model architecture we found finally for malaria recognition. During our experiments we saw that the training history graphic does not become as flattened as it was found in the previous experiments as a sign that the data are not easy to learn. Also, as a more important point, the validation accuracy graphic fluctuates a lot and it is quite oscillated. These points show us that it is difficult to obtain convergence for our model with the malaria data. Clearly the use of the model would probably not result in the best possible test performance than it. We therefore decided to train our model once from scratch with 50 epochs of training and then selected the model with the best validation score. The training history is presented in Figure 4.31.

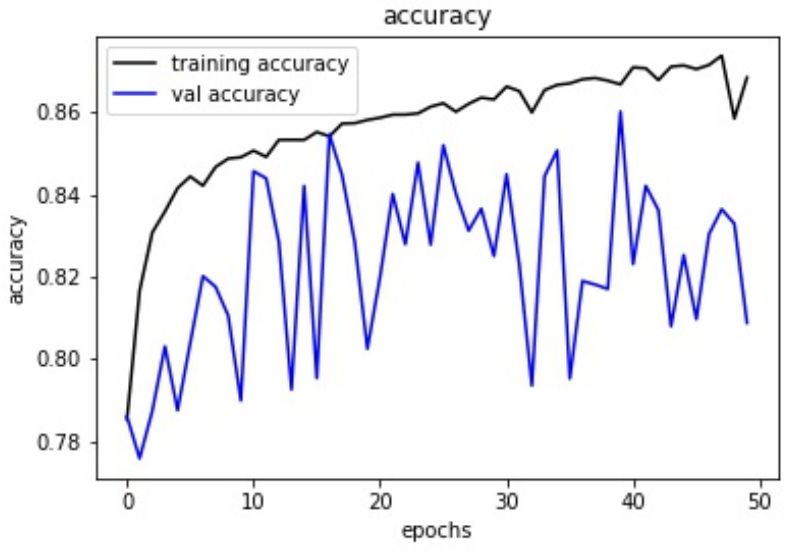

Figure 4.31. Training History for Breast Cancer Data.

With regards to Figure 4.31, we see that the best validation score was obtained in the 40^{th} epoch with 0.8602. Using the model found in the 40^{th} epoch, training accuracy was found to be 0.8867 and test accuracy was found to be 0.8594. The confusion matrix of this model for test data is presented as 4.32.

Figure 4.32. Confusion Matrix for Breast Cancer Test Data.

For the test data we obtained precision, recall, f-measure and balanced accuracy values of 0.8566, 0.7088, 0.7757, and 0.8585. These results can also be calcuated using the results of Figure 4.32. It seems that for this experiment the model was trained in a way to be likely to make negative predictions. We understand this from higher precision and lower recall results. However, we made a totally balanced experiment between these 2 classes regarding class weights and probability threshold. Considering this situation and our other empirical results, we can say that this propensity for negative class was obtained randomly. However, again, using some approaches like probability threshold moving applied in Malaria recognition, our model can be modified in a way to increase recall and decrease precision.

Table 4.12 shows us some performance results reported in the literature. With regards to this table, we see that our approach is competitive, our test result (a balanced accuracy of 85.85%) is only outperformed by the best model. However, it is again necessary to stress that the training and test data are not given in the dataset and the selection of training and test data splitting is not always clearly explained in these papers. That's why, this comparison is not exact and can give only a general idea.

Table 4.12. Breast Cancer Recognition Result Comparison.

Research	**Method**	**Balanced Test Accuracy**
Romero *et al.* [27]	Custom CNN	89%
Reza and Ma [25]	Custom CNN	85.48%
Romano and Hernandez [26]	Custom CNN	85.41%
Janowczyk and Madabhushi [24]	Transfer Learning CNN	84.68%
Cruz-Roa *et al.* [23]	Custom CNN	84.23%

4.2.3. Folder and File Structure for Medical Image Recognition

Figure 4.33 shows the folder and file structure used in our medical image recognition experiments. In hog and hog_test folders the produced hog dataframes are found for training and test data. Under binary_images folder, in training_all folder, there are folders with class numbers (0 and 1) and in each of these class folders there are related training images. Under binary_images folder, in test_all folder, there are folders with class numbers (0 and 1) and in each of these class folders there are related test images. Test_All.csv file contains image and class id information for test data. The codes can be reached using the link https://github.com/mertcetinkaya/master_thesis_codes and this folder and file structure must be created to run the codes properly. The readme file on github contains additional information.

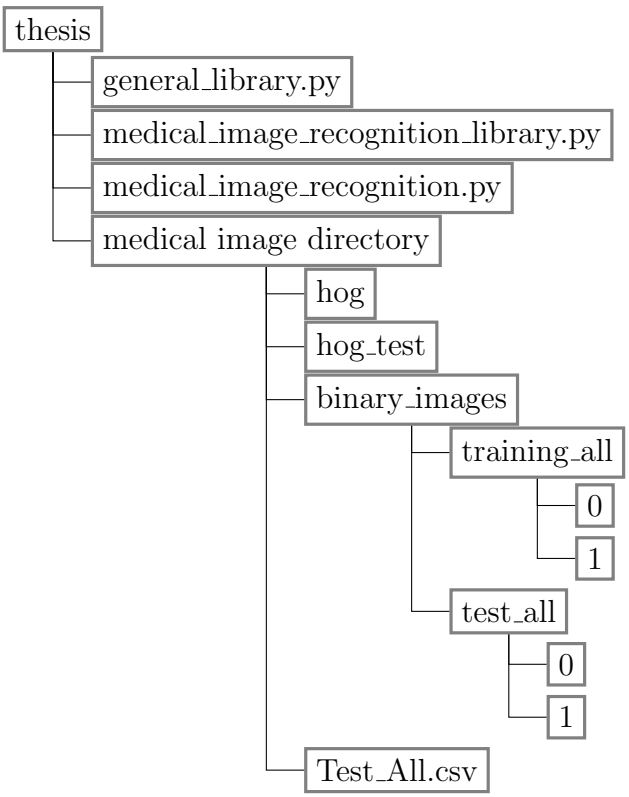

Figure 4.33. Medical Image Recognition Folder and File Structure.

4.3. Optical Character Recognition

In this part of the thesis, we develop a real text digitalization application. Thus the aim is not only to recognize the given letter, but also to extract first these letters one by one from the given text image. Thus it will be possible to digitalize real text from a text image. We can call this a proof of concept study for producing an OCR engine.

For this purpose, firstly, a simple character segmentation approach was produced. Our algorithm for this segmentation is given in a simplified and step by step way in Figure 4.34. Before starting character segmentation, we first convert our image to a binary image. That means that after that conversion the characters must all be black pixels and the remaining pixels must all be white. Our character segmentation algorithm for such a binary image is given below.

- Start from the top pixel line of the image and examine each line pixel by pixel.
- When at least one black pixel is recognized in a pixel line that means that the top border of a line that includes characters (the top border of a character line) was found.
- After the top border of a character line was found, we continue to check the below pixel rows till the first pixel row without a black pixel is recognized. Thus the bottom border (ie exit) of a character line was found.
- The same logic is kept until the end of the image and horizontal borders (character lines) are found.
- The same logic is used inside of extracted character lines, this time, to find the vertical borders in each line. Thus it is possible to determine the absolute rectangular borders of every character and to extract them one by one from the image.

Figure 4.34. Character Segmentation Algorithm.

As an example, the result of this segmentation algorithm applied to the image of Figure 4.35 is presented in Figure 4.36.

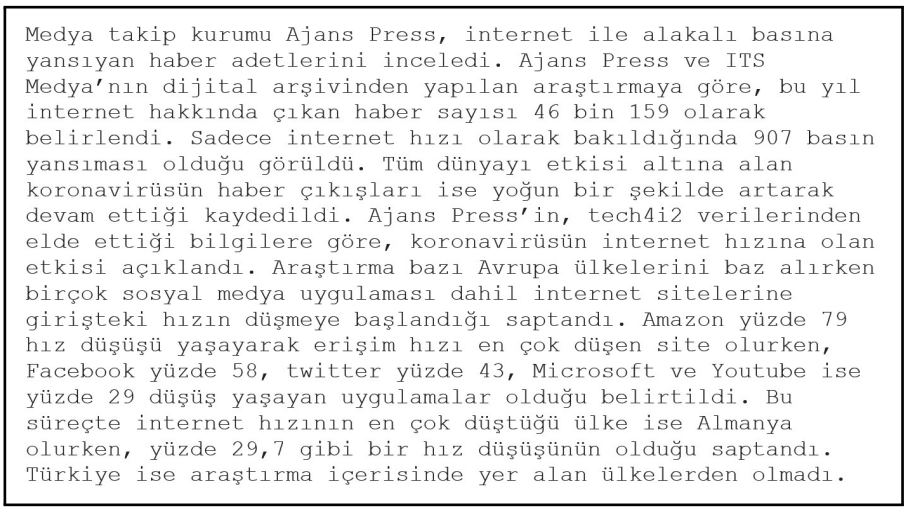

Figure 4.35. Sample Image Before Segmentation.

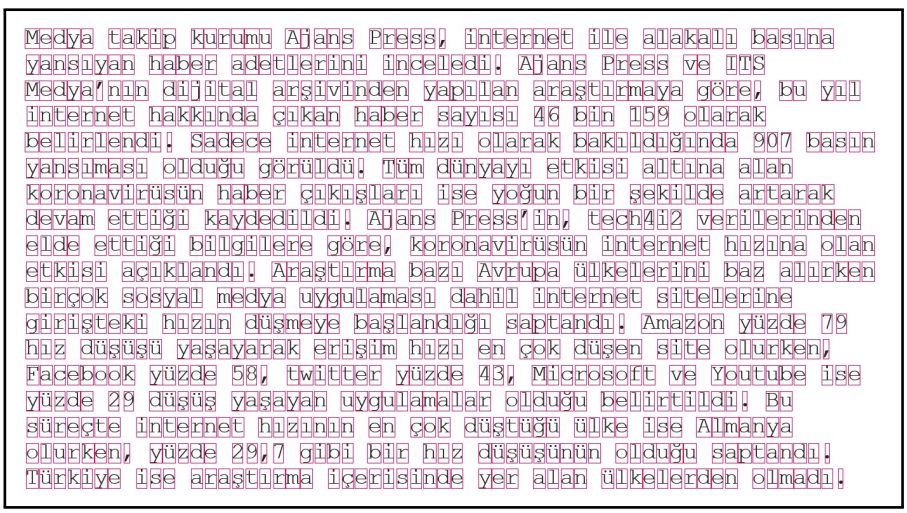

Figure 4.36. Sample Image After Segmentation.

We can see in Figure 4.36 that after the segmentation operation there can always remain fully white row above and below the character. These white spaces can decrease the performance of the classifier. Therefore we add here a preprocessing step, that

removes these white spaces by raising the lower boundary line and/or lowering the upper boundary line. A resizing operation is necessary after this space removal.

For the next step, the classification task, we used logistic regression (LR), Naive Bayes (NB) and SVM with linear kernel. For training and using these classifiers we always used the binary and flattened representation of the input and we created our own dataset. It is based on the following alphabet given in Figure 4.37 containing all Turkish and English letters, the numbers, the punctuation signs and some special characters.

AaBbCcÇçDdEeFfGgĞğHh
IıİiJjKkLlMmNnOoÖöPp
QqRrSsŞşTtUuÜüVvWwXxYyZz0123456789-.,;:?!'*()

Figure 4.37. Alphabet.

As we created our own training set, its size is not large. Considering also that the inputs to classify are less complicated compared to the images in the previous sections, in this classification process, we did not apply deep learning methods but continued with our three simpler classifiers.

Training is conducted after the segmentation and the preprocessing step to extract the letters from images like Figure 4.38. This figure also shows the alphabet. We produced several of them using different resolutions, fonts and also blurring effects. As fonts Courier, Times New Roman, Bahnschrift, Arial, Cambria and Verdana were used. Each of the extracted characters is then put into the data set in binary, flattened form to get more data. Figure 4.38 presents such a 'training' image using Courier font.

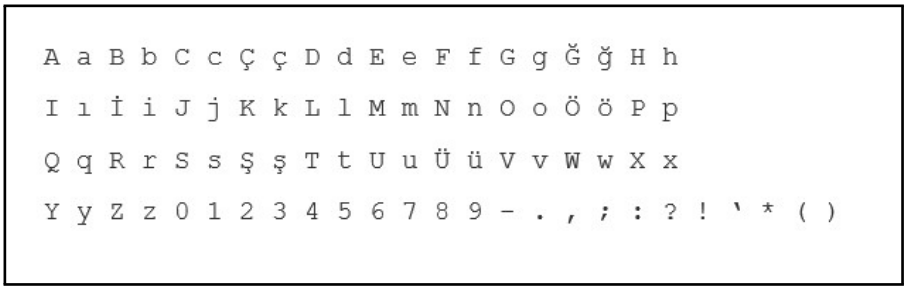

Figure 4.38. Alphabet Image.

After preparation of the segmentation and the preprocessing algorithm and training a classifier, our system is ready to digitalize text images. Length of the alphabet is 85 and there exist 3,060 characters in our dataset. We used 25% of the characters in our dataset as test set and used the rest as training set. Figure 4.39 presents the width and height of the characters observed after segmentation and space removal. Using those results and some empirical experiments we decided to resize all characters to 30x30 as another preprocessing step after space removal. Finally, we got the following classification results in Table 4.13.

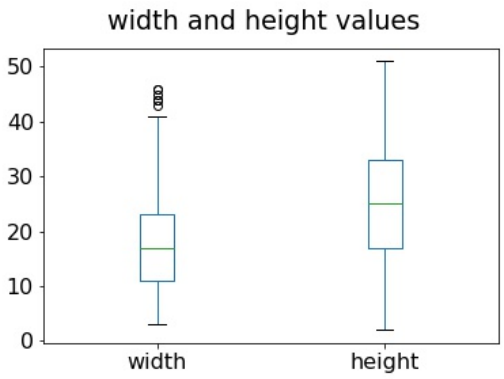

Figure 4.39. Width and Height Information for Characters.

Table 4.13. OCR Results for Classification.

	Training Time (s)	**Training Accuracy**	**Test Accuracy**
LR	3	0.9961	0.9333
SVM	17	0.9969	0.9346
NB	0.04	0.9382	0.6588

According to Table 4.13 logistic regression is the best classifier considering training time and accuracy values together. That's why, we continued with logistic regression through this section. The related confusion matrix of logistic regression is given for test data in https://github.com/mertcetinkaya/master_thesis_codes (in images folder).

Here, to augment the classification accuracy we also defined some rules for extra correction, in order to prevent the system from mixing up characters like i, İ, ı, I, l, 1 or ., ,, -, ' or the upper case and lower case of the characters of c, o, p, s, u, v, w, x and z. We selected these character groups using experimental text digitalization results and the confusion matrix in https://github.com/mertcetinkaya/master_thesis_codes. We defined these rules with regards to the original height of character found just after segmentation, the first and the second cutting point for preprocessing (space removal), the probability predictions for different classes made by classifier and some empirical results. It is also necessary to add that normally SVM is not a probabilistic method, but it can be used in probabilistic way by probability calibration and probabilities are calibrated using Platt scaling [79] at the basis and extending it [80]. This kind of probability calibration approaches can be used for any non-probabilistic estimator to make it probabilistic. However, it increases the training time. The training time difference for SVM given in Table 4.13 is mostly sourced by this situation.

In Figure 4.40, some elements used in this extra correction mechanism are presented via an 's' and "' taken from Figure 4.36. These characters are presented before

and after space removal. Origin is always taken to be 0 and the first cutting point and the second cutting point values are determined by counting pixel values (pixel values are increasing one by one towards the bottom). The given height values are also simply pixel count. Figure 4.40 is also an example for the claim given above. We clearly see that after the space removal (and also resizing), it is not possible (or very difficult) to distinguish between 's' and 'S' or "' and ','. That's why, we present an extra correction mechanism between mixable characters using the ratios between the elements presented in Figure 4.40.

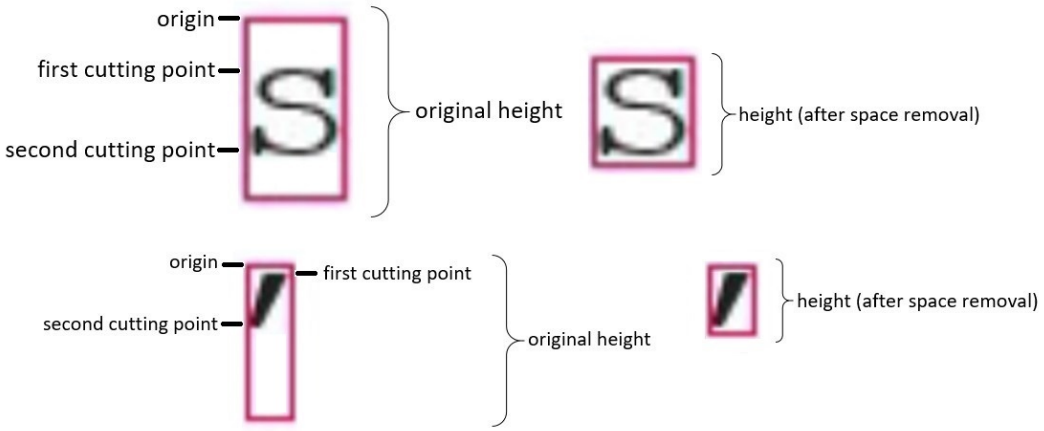

Figure 4.40. Elements Used in Extra Correction.

In Figure 4.41, the related ratio values are given for the characters used in the training set. These ratios and some empirical results are benefited during this extra correction to determine some decisive threshold. For example, if a character is found to be 's', but if its height (after space removal) is large, then it is updated to be 'S'. In order to understand, if the height of a character is large, simply first cutting point / original height ratio can be used. If this ratio is small (compared to the determined threshold), then, we can say that height of that character is large. As another example, if a character is found to be "', but the positional information is not logical, the character with the biggest probability between '.' and ',' is chosen for this character. The positional information can be found using again first cutting point /

original height ratio. If this ratio is large, it means that first cutting point is large and this is a sign to be a '.' or ','. According to our results, these 3 characters can frequently be mixed up between each other. A similar situation also exists between 'i', 'İ', 'ı', 'I', 'l' and '1'. The positional and the probabilistic information is again used to distinguish between these characters as well. We use this kind of approaches in this extra correction mechanism. The whole code and the remaining details for this mechanism can be found in Figures A.3 and A.4.

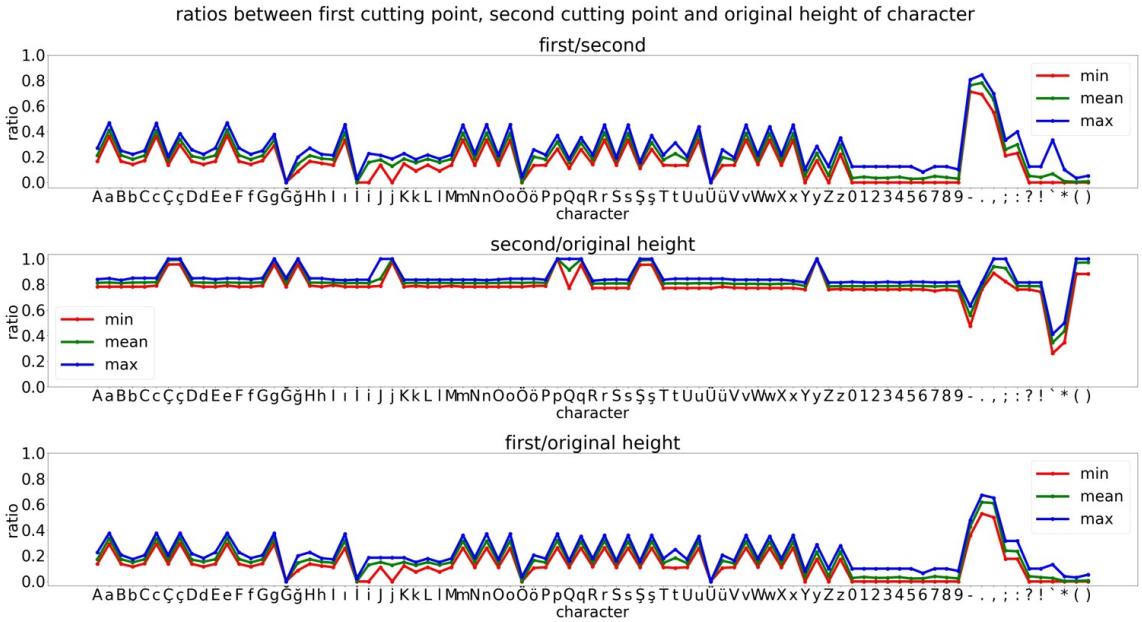

Figure 4.41. Ratios Used in Extra Correction.

This extra correction mechanism improves the accuracy in our character recognition test data by approximately 4%. Table 4.14 shows the results after this extra correction mechanism.

Table 4.14. OCR Results after Extra Correction.

	Training Accuracy	**Test Accuracy**
LR	0.9834	0.9712
SVM	0.9838	0.9725
NB	0.9625	0.7006

The related confusion matrix of logistic regression is given for the test data in https://github.com/mertcetinkaya/master_thesis_codes (in images folder) after this extra correction mechanism.

To assess the quality of our method we report here some result from the literature. Afroge *et al.* [32] used a similar procedure as here and they produced their own dataset. They worked on English characters including the digits (0 to 9), the uppercase letters (A to Z) and the lowercase letters (a to z) and obtained around 93% accuracy on their test data using neural networks. In another study, Phangtriastu *et al.* [29] used English uppercase letters taken from a ready dataset and they reached around 95% classification accuracy using SVM. Sevik *et al.* [35] worked on Turkish letters. Using different fonts, they produced their own dataset for Turkish letter classification similar to this thesis and they used AlexNet as a deep learning architecture. They report a success of nearly 100% percent for their test data. Lastly, in his master thesis, Afşin [81] also used different fonts and produced his own dataset using almost all Turkish and English characters with the lower and the upper cases, the digits, some special characters and the punctuations. In his application, he used CNN for classification and obtained 96% test accuracy.

It is clear that the comparison can only give a general overview as the set of characters used, the training sets and the test sets are all different. We can claim that the produced dataset in this thesis is, compared to the studies we found in the

literature, quite comprehensive. It can be used for a real OCR engine as it includes all of the lowercase and uppercase letters, the digits, the special characters and the punctuation signs. Especially important is that it includes all Turkish characters as they are only covered in very few studies.

4.3.1. Text Digitalization Experiments

To give a general idea about the performance of the OCR system we look now at some real text digitalization experiments. In these examples, we used different images of the same Turkish text to test the performance of our method under different conditions. We used a Turkish text because the Turkish characters have a big variety and are so seldom considered in the literature. The proposed text digitalization process is given in Figure 4.42.

- Obtain the proper binary text image of the original image.
- Apply character segmentation.
- Apply preprocessing for space removal on the segmented characters and resize them to 30x30.
- Classify the characters and apply the extra correction mechanism if necessary to obtain the true classes.
- Write the letters according to the class information, in a proper way, to obtain the digitalization of the text image.

Figure 4.42. Text Digitalization Process.

For example, the output of the algorithm for Figure 4.35 is shown in Figure 4.43. The text in Figure 4.35 is written with Courier font and, in this section, the following results and all of the other results were obtained using logistic regression and the extra correction mechanism presented. During these experiments we generally preferred using images with low resolution. For example, Figure 4.35 is of 1387x756 resolution and the below image shows us the digitalization for that image.

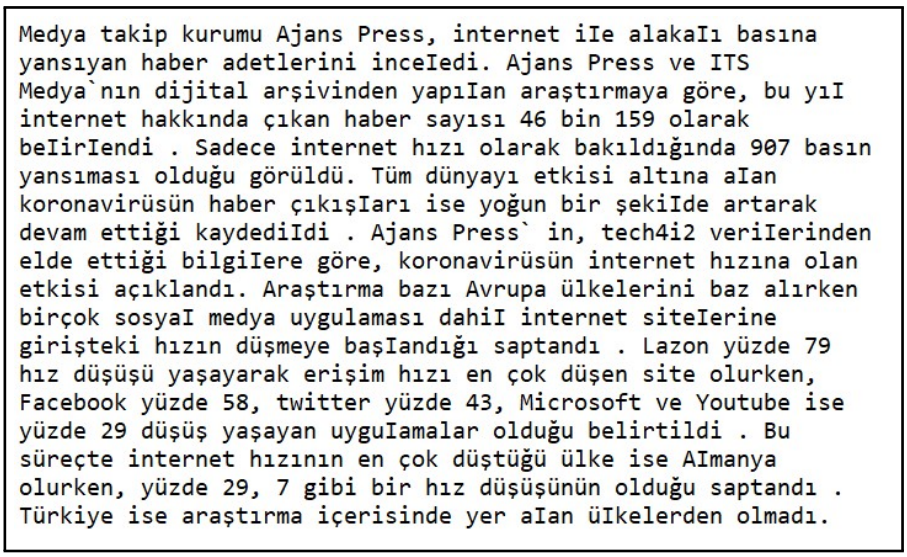

Figure 4.43. Digitalization for Figure 4.35.

It is interesting that our model works also well for text images written with fonts that are not included in the training set. For example in Figure 4.44 we see a text written with Consolas font. The figure is of 1411x742 resolution and it shows the same text used in Figure 4.35. Figure 4.45 shows the segmentation results and Figure 4.46 shows the output of our algorithm as digitalization of Figure 4.44.

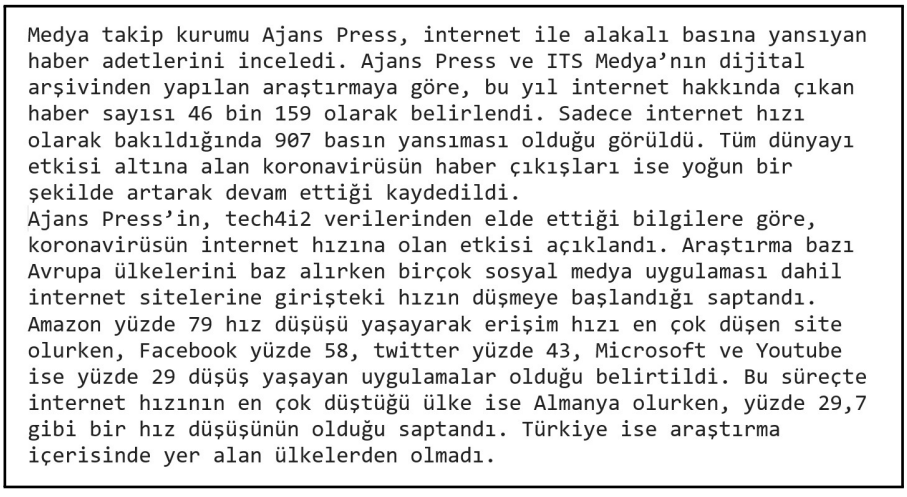

Figure 4.44. Sample Image Before Segmentation.

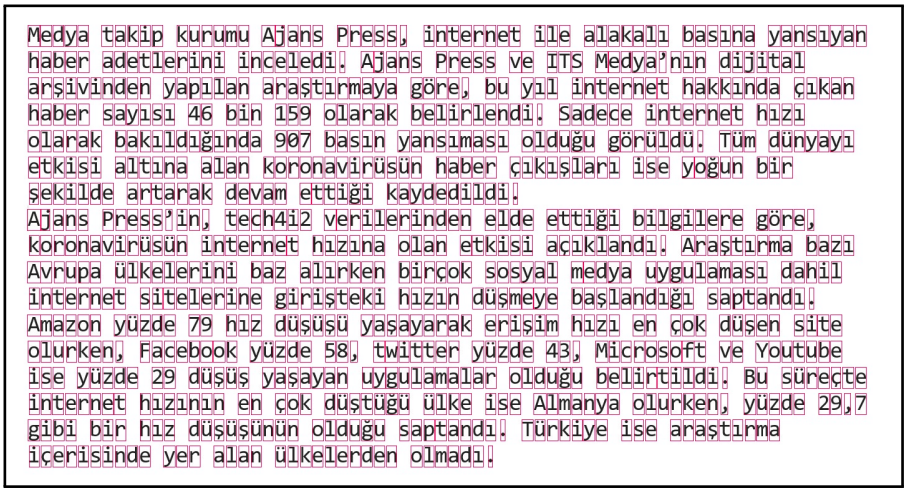

Figure 4.45. Sample Image After Segmentation.

```
Medya takip kurumu Ajans Press, internet ile alakalı basına yansıyan
haber adetIerini inceledi. Ajans Press ve ITS Medya`nın dijital
arşivinden yapıIan araştırmaya göre, bu yıl internet hakkında çıkan
haber sayısı 46 bin 159 olarak belirlendi. Sadece internet hızı
olarak bakıldığında 907 basın yansıması olduğu görüldü. Tüm dünyayı
etkisi altına aIan koronavirüsün haber çıkışIarı ise yoğun bir
şekilde artarak devam ettiği kaydedildi.
Ajans Press`in, tech4i2 verilerinden elde ettiği bilgilere göre,
koronavirüsün internet hızına olan etkisi açıklandı. Araştırma bazı
Avrupa ülkelerini baz alırken birçok sosyal medya uygulaması dahil
internet sitelerine girişteki hızın düşmeye başlandığı saptandı.
Amazon yüzde 79 hız düşüşü yaşayarak erişim hızı en çok düşen site
olurken, Facebook yüzde 58, twitter yüzde 4B, Microsoft ve Youtube
ise yüzde 29 düşüş yaşayan uygulamalar olduğu beIirtiIdi. Bu süreçte
internet hızının en çok düştüğü ülke ise Almanya olurken, yüzde 29,7
gibi bir hız düşüşünün oIduğu saptandı. Türkiye ise araştırma
içerisinde yer alan ülkeIerden olmadı.
```

Figure 4.46. Digitalization for Figure 4.44.

In addition, in Figure 4.47 we see another text with low resolution, this time written with Calibri font. The image is of 1420x603 resolution. Figure 4.48 shows the segmentation results and Figure 4.49 shows the output of our algorithm as digitalization of Figure 4.47.

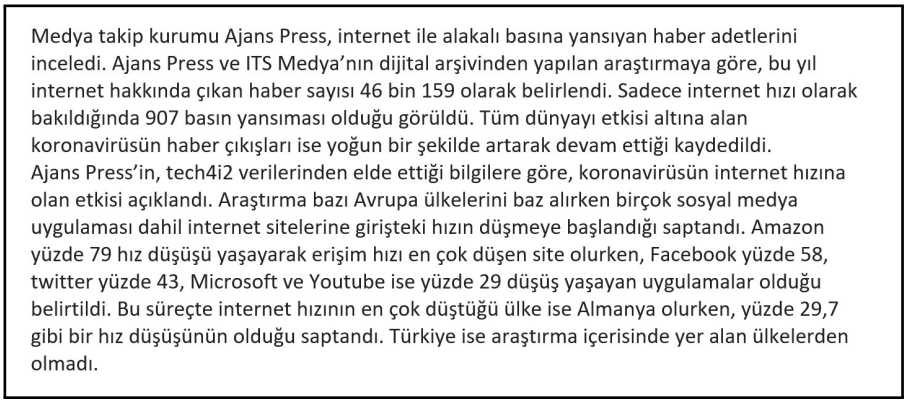

Figure 4.47. Sample Image Before Segmentation.

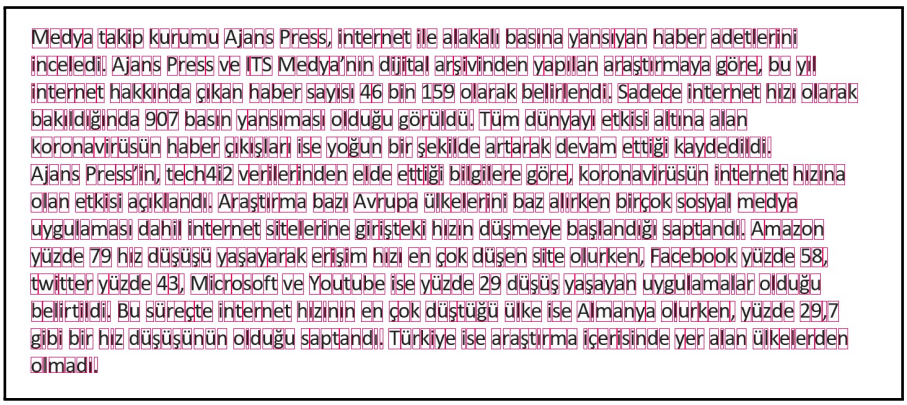

Figure 4.48. Sample Image After Segmentation.

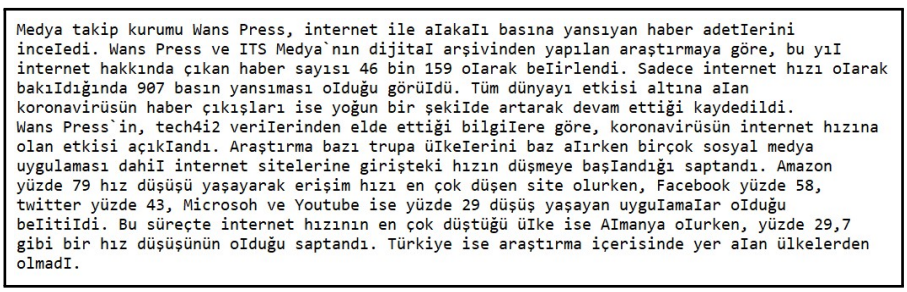

Figure 4.49. Digitalization for Figure 4.47.

We made these experiments getting also print-out of these texts and scanning them and found similar results at the end, but because of the resolution decline and

some noises or small smears emerging during scanning, there has been a little performance decline in this case.

The model is also robust to noise and blurring effects. For example, in Figure 4.50 we see the image of Figure 4.44 with some salt and pepper noise and in Figure 4.51 we see the related segmented image. Here, some blurring tricks are used firstly to remove the salt and pepper noise as an extra preprocessing on the image and the standard digitalization procedure given in Figure 4.42 is applied on this remaining cleaned image. This mentioned cleaned image is given in Figure 4.52. The digitalization results are again quite similar to the original version of the image which is Figure 4.44.

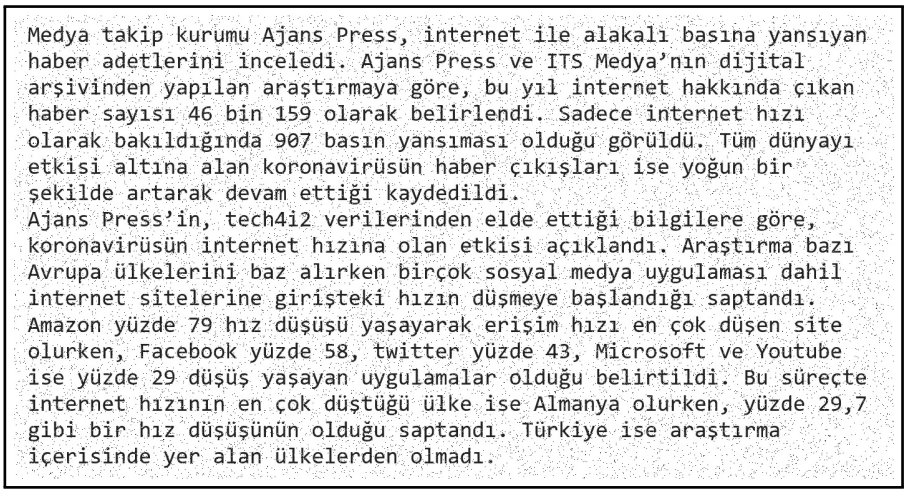

Figure 4.50. Sample Image Before Segmentation.

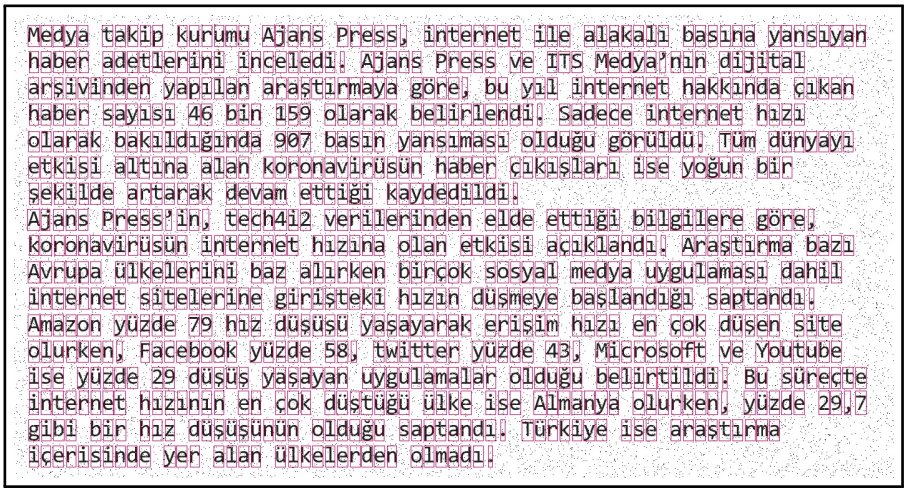

Figure 4.51. Sample Image After Segmentation.

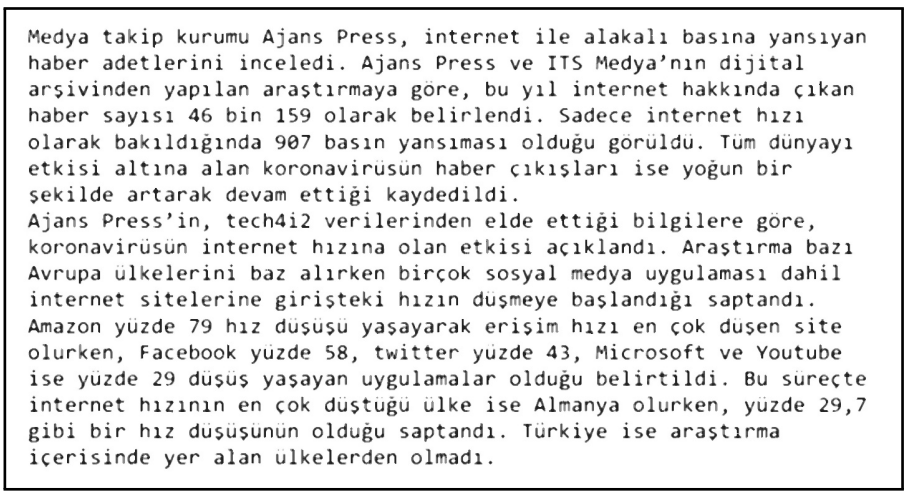

Figure 4.52. Cleaned Image by Blurring to Remove Noise.

Also, in Figure 4.53 we see the image in Figure 4.44 with some blurring effect and in Figure 4.54 we see the related segmented image. Here, the threshold was changed while transforming the image to its binary version and the standard procedure is applied on this image. The digitalization results are again quite similar to the original version of the image which is Figure 4.44.

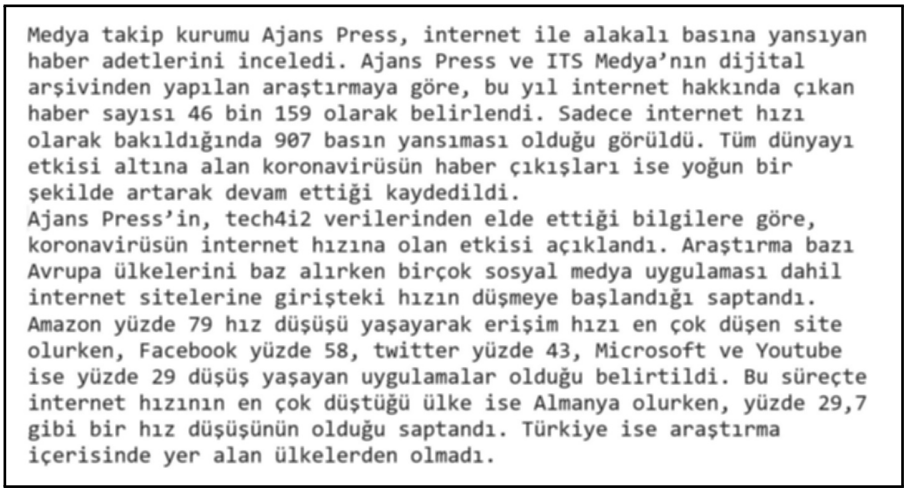

Figure 4.53. Sample Image Before Segmentation.

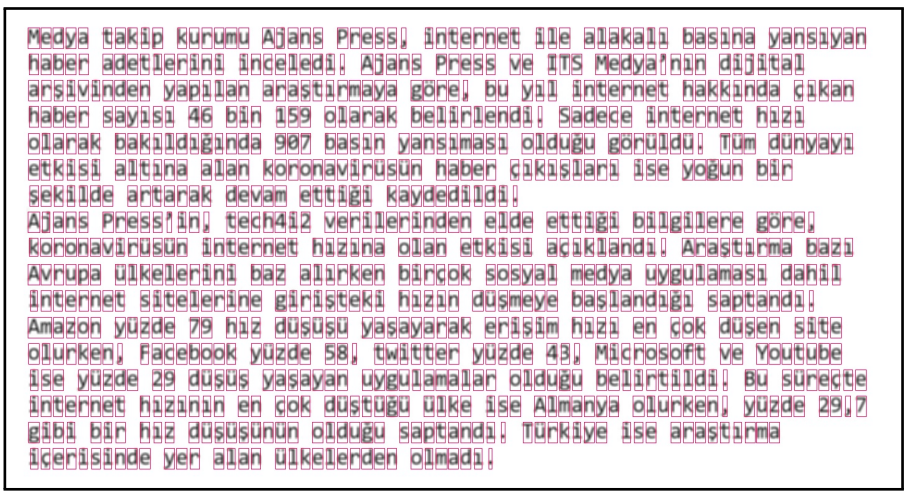

Figure 4.54. Sample Image After Segmentation.

Lastly, the algorithm can work with other colours as well. For example, in Figure 4.55 we see an example of a colour image written with Candara font and in Figure 4.56 we see the related segmented image. The binary version of the image that the algorithm runs through is given in Figure 4.57. Up to now, in the given examples, because greylevel images were used, binary images have always been quite similar to the original or cleaned images, but in the case of a colour image we see a real image transformation while obtaining the binary version. At the end, Figure 4.58 shows the

output of our algorithm as digitalization of Figure 4.55.

Figure 4.55. Sample Image Before Segmentation.

Figure 4.56. Sample Image After Segmentation.

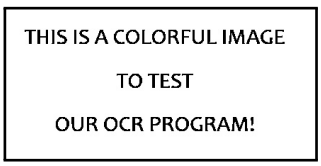

Figure 4.57. Obtaining Binary Version.

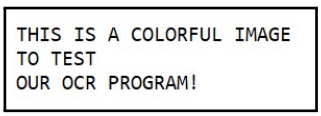

Figure 4.58. Digitalization for Figure 4.55.

There are many other experimental setings possible. For example to digitalize images with noisy lines or containing tables that include straight lines can be quite useful in practice. In these cases additional image processing techniques must be applied to detect and clear the lines. With regards to our experiments presented so far, we can say that our approach works well and gives generally nice results as a proof of

concept study. The presented segmentation and preprocessing approach has also some similar points to the one proposed by Afroge *et al.* [32]. However we add a correction mechanism that increases the success rate in the test set. Also, in addition to English characters, Turkish characters and (in addition to letters and numbers) punctuation and special characters are also used to obtain a complete text digitalizer. Lastly, the presented approach is tested on more complex and diversified text images. The presented method is open to improvement. Especially the error rate when digitalizing texts should be reduced in practice when using NLP with word match and sentence context match.

4.3.2. Folder and File Structure for Optical Character Recognition

Figure 4.59 shows the folder and file structure used in optical character recognition experiments. In ocr_character folder segmented and preprocessed characters from text image are stored for text digitalization. In ocr_images folder alphabet figures like Figure 4.38 are found to use for model training and in ocr_training/images folder the segmented and preprocessed characters from supplied training images are stored. The image to test (to digitalize) must also be stored in ocr_images folder. The codes can be reached using the link https://github.com/mertcetinkaya/master_thesis_codes and this folder and file structure must be created to run the codes properly. The readme file on github contains additional information.

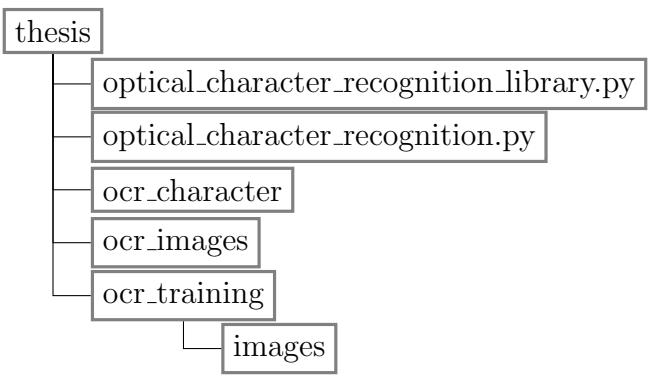

Figure 4.59. Optical Character Recognition Folder and File Structure.

5. ABOUT MACHINE LEARNING AND OPTIMIZATION

Machine learning is actually a subset of artificial intelligence and it is an area of research that gives computers the ability to learn without being explicitly programmed [82]. Its aim is to develop systems or algorithms that have the ability to learn and to improve themselves automatically through experience. Machine learning algorithms build a mathematical model based on the data used and optimization is an indispensable component of machine learning because machine learning needs to solve these mathematical models efficiently [83].

In machine learning problems a loss function is defined that has to be minimized. This indicates that machine learning and optimization have a close relationship and optimization techniques are used for machine learning. Domingos [84] wrote the equation $MachineLearning = Representation + Optimization + Evaluation$ and listed common examples of these components.

The main differentiation is that for machine learning most important that the obtained method is generalizing also to the test set whereas in optimization accuracy is the main objective. An optimization algorithm tries to minimize the loss of a training set, but machine learning is related to minimizing the loss of the test set. This means it searches for a generalization and learning [85].

This difference of perspective separates pure optimization and machine learning communities. For example, machine learning people may favor a local minimizer or an early stopping in training data with regards to validation data results. The validation data results are quite important in machine learning community for model training. However, validation data would not be that important for pure optimization people. They focus more on minimizing the training error [83] and therefore maximizing the training accuracy with regards to the modelled problem. In addition to accuracy and finding the best results for the problem at hand, mathematical programming puts a

premium on speed and robustness [86]. This can be seen as pure mathematics applied to develop the best algorithm that reaches the exact result as fast as possible. However, if we look at the machine learning side, generalization is the bottom line and accuracy and small speed improvements are not a big concern. Simpler algorithms that work in plausible computational time for specific classes of problems are preferred [86]. To say it shorter: From the machine learning perspective a good optimization algorithm must be simple, easy to implement and not too slow together with good generalization, and fast convergence to an approximate solution. However, for the pure optimization community, instead of an approximate solution, the most accurate solution is searched while trying to propose a fast and robust algorithm.

Finally we can also see an interplay between both fields. In machine learning the mathematical optimization techniques are used for better training. Thus some specific needs and increasing interest for some specific models in machine learning motivate the optimization community to focus more on these models. They aim to develop more advanced optimization techniques in order to solve those problems relevant for machine learning. This interaction also encourages people from these 2 different communities to learn and better understand the needs and aims of the other group. It is clear that both fields will continue to grow stronger if they continue being open to each other and keeping this mutual relationship [83].

6. CONCLUSION

In this thesis, we dealt with different image classification problems. With regards to our experiments on traffic sign recognition and medical image recognition, we clearly observed that, as often claimed in the literature, CNN outperformed the classical methods. As reason for this successful performance of CNN seems to be the feature engineering of CNN with its more complex structure. Thus the units of CNN can work together and can be deepened to augment its learning capacity. Thanks to these points CNN can make better feature engineering and thus extract more information from the input that for a successful classification. Ordinary shallow machine learning rests quite weak against CNN, especially when the data size is large enough to utilize the great learning potential of CNN and when the nature of the data are involved like the complex image data used in traffic sign recognition or in medical image recognition.

For these 2 problems we were also able to observe that elaborate CNN fine-tuning and the use of problem specific approaches can improve the performance of CNN.

In the experiments on OCR, we worked on a simpler dataset that we produced to accommodate the characters of our alphabet in order to make real text digitalization. In these experiments, we simply used binary representation of the characters and our classical classifiers and we saw that they give good results for OCR in this context. In these applications, we used simple approaches for character segmentation and classification. We also proposed an extra correction mechanism to improve the performance of the classifier for our OCR application. Making use of basic image processing techniques, we also proposed our simple OCR engine to make real text digitalization as a proof of concept level study and showed that it gives nice and promising results.

In this thesis, we completed a general study about pattern recognition and image classification. We focused mainly on CNN, but we experimented also with classical techniques. We saw that the usage of CNN is thanks to its automatic and flexible

mechanisms a sensible decision, especially for complex and large datasets. We also saw that elaborate fine-tuning of a classifier can be useful to improve its performance. In the third part of this thesis, we developed an OCR engine. We demonstrated that combining a segmentation algorithm, logistic regression for character recognition and some special rules to improve the classification of very similar characters leads to a good and robust OCR software. These main building blocks may be further improved to build a more advanced OCR engine for practical use.

Lastly, the codes of all our experiments are available on Github using the link https://github.com/mertcetinkaya/master_thesis_codes. Necessary information for users can be found in the readme files of the Github repository.

REFERENCES

1. Bishop, C. M., *Pattern Recognition and Machine Learning*, Springer, Cambridge, 2006.

2. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, MIT Press, Cambridge, 2017.

3. He, K., X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", *arXiv 1502.01852*, 2015.

4. Reback, J., W. McKinney, J. V. d. Bossche, T. Augspurger, P. Cloud, S. Hawkins, M. Roeschke, A. Klein, T. Petersen, J. Tratner, C. She, W. Ayd, S. Naveh, M. Garcia, J. Schendel, A. Hayden, D. Saxton, V. Jancauskas, M. Gorelli, R. Shadrach, A. McMaster, P. Battiston, S. Seabold and K. Dong, *pandas-dev/pandas: Pandas*, Zenodo, 2020.

5. Oliphant, T., *Guide to NumPy*, 2006, http://www.numpy.org/.

6. Buitinck, L., G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. Vanderplas, A. Joly, B. Holt and G. Varoquaux, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830, 2011.

7. Hunter, J. D., "Matplotlib: A 2D Graphics Environment", *Computing in Science & Engineering*, Vol. 9, No. 3, pp. 90–95, 2007.

8. Chollet, F., *Keras*, GitHub, 2015, https://github.com/fchollet/keras.

9. Bradski, G., "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

10. v. d. Walt, S., J. L. Schonberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner,

N. Yager, E. Gouillart and T. Yu, "Scikit-Image: Image Processing in Python", *PeerJ*, Vol. 2, p. e453, 2014.

11. Berger, M., A. Forechi, A. F. D. Souza, J. D. O. Neto, L. Veronese and C. Badue, "Traffic Sign Recognition with VG-RAM Weightless Neural Networks", *International Conference on Intelligent Systems Design and Applications*, pp. 315–319, 2012.

12. Gudigar, A., S. Chokkadi, U. Raghavendra and U. R. Acharya, "Local Texture Patterns for Traffic Sign Recognition Using Higher Order Spectra", *Pattern Recognition Letters*, Vol. 94, pp. 202–210, 2017.

13. Zaklouta, F., B. Stanciulescu and O. Hamdoun, "Traffic Sign Classification Using K-d Trees and Random Forests", *International Joint Conference on Neural Networks*, pp. 2151–2155, 2011.

14. Wang, C. and W. You, "Boosting-SVM: Effective Learning with Reduced Data Dimension", *Applied Intelligence*, Vol. 39, pp. 465–474, 2013.

15. Yin, S., J. Deng, D. Zhang and J. Du, "Traffic Sign Recognition Based on Deep Convolutional Neural Network", *Computer Vision*, pp. 685–695, Springer, 2017.

16. Mehta, S., C. Paunwala and B. Vaidya, "CNN Based Traffic Sign Classification Using Adam Optimizer", *International Conference on Intelligent Computing and Control Systems*, pp. 1293–1298, 2019.

17. Lu, K., Z. Ding and S. Ge, "Sparse-Representation-Based Graph Embedding for Traffic Sign Recognition", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 13, No. 4, pp. 1515–1524, 2012.

18. Aziz, S., E. A. Mohamed and F. Youssef, "Traffic Sign Recognition Based on Multi-Feature Fusion and ELM Classifier", *Procedia Computer Science*, Vol. 127, pp. 146–153, 2018.

19. Rajaraman, S., S. Antani, M. Poostchi, K. Silamut, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, M. A. Hossain, R. M. ., S. Jaeger and G. Thoma, "Pre-Trained Convolutional Neural Networks as Feature Extractors Toward Improved Malaria Parasite Detection in Thin Blood Smear Images", *PeerJ*, Vol. 6, 2019.

20. Rajaraman, S., S. Jaeger and S. Antani, "Performance Evaluation of Deep Neural Ensembles Toward Malaria Parasite Detection in Thin Blood Smear Images", *PeerJ*, Vol. 7, 2019.

21. Qayyum, A. B. A., T. Islam and M. A. Haque, "Malaria Diagnosis with Dilated Convolutional Neural Network Based Image Analysis", *IEEE International Conference on Biomedical Engineering, Computer and Information Technology for Health*, pp. 68–72, 2019.

22. Akilotu, B., Z. Kadiroğlu, A. Sengur and M. M. Kayaoğlu, "Malaria Detection using Both Convolutional Neural Networks and Transfer Learning Method", *International Engineering and Science Symposium*, 2019.

23. Cruz-Roa, A., A. Basavanhally, F. González, H. Gilmore, M. Feldman, S. Ganesan, N. Shih, J. Tomaszewski and A. Madabhushi, "Automatic Detection of Invasive Ductal Carcinoma in Whole Slide Images with Convolutional Neural Networks", *Progress in Biomedical Optics and Imaging - Proceedings of SPIE*, Vol. 9041, 2014.

24. Janowczyk, A. and A. Madabhushi, "Deep Learning for Digital Pathology Image Analysis: A Comprehensive Tutorial with Selected Use Cases", *Journal of Pathology Informatics*, 2016.

25. Reza, M. S. and J. Ma, "Imbalanced Histopathological Breast Cancer Image Classification with Convolutional Neural Network", *IEEE International Conference on Signal Processing*, pp. 619–624, 2018.

26. Romano, A. M. and A. A. Hernandez, "Enhanced Deep Learning Approach for

Predicting Invasive Ductal Carcinoma from Histopathology Images", *International Conference on Artificial Intelligence and Big Data*, pp. 142–148, 2019.

27. Romero, F. P., A. Tang and S. Kadoury, "Multi-Level Batch Normalization in Deep Networks for Invasive Ductal Carcinoma Cell Discrimination in Histopathology Images", *IEEE International Symposium on Biomedical Imaging*, pp. 1092–1095, 2019.

28. Bakator, M. and D. Radosav, "Deep Learning and Medical Diagnosis: A Review of Literature", *Multimodal Technologies and Interaction*, Vol. 2, p. 47, 2018.

29. Phangtriastu, M., J. Harefa and D. Tanoto, "Comparison Between Neural Network and Support Vector Machine in Optical Character Recognition", *Procedia Computer Science*, Vol. 116, pp. 351–357, 2017.

30. Roy, S., A. Chatterjee and K. Goswami, "Printed Text Character Analysis Version-II: Optimized Optical Character Recognition for Noisy Images with the New User Training and Background Detection Mechanism", *International Journal of Advanced Computer Research*, Vol. 4, pp. 601–610, 2014.

31. Xie, J., "Optical Character Recognition Based on Least Square Support Vector Machine", *International Symposium on Intelligent Information Technology Application*, Los Alamitos, CA, USA, 2009.

32. Afroge, S., B. Ahmed and F. Mahmud, "Optical Character Recognition Using Back Propagation Neural Network", *International Conference on Electrical, Computer & Telecommunication Engineering*, Rajshahi, Bangladesh, 2016.

33. Vamvakas, G., B. Gatos, I. Pratikakis, N. Stamatopoulos, A. Roniotis and S. J. Perantonis, "Hybrid Off-Line OCR for Isolated Handwritten Greek Characters", *IASTED International Conference on Signal Processing, Pattern Recognition, and Applications*, Innsbruck, Austria, 2007.

34. Wei, T., U. Sheikh and A. Rahman, "Improved Optical Character Recognition with Deep Neural Network", *International Colloquium on Signal Processing & its Applications*, Penang, Malaysia, 2018.

35. Sevik, A., P. Erdogmus and E. Yalein, "Font and Turkish Letter Recognition in Images with Deep Learning", *International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism*, pp. 61–64, 2018.

36. Ye, Q., Q. Huang, W. Gao and D. Zhao, "Fast and Robust Text Detection in Images and Video Frames", *Image and Vision Computing*, Vol. 23, No. 6, pp. 565–576, 2005.

37. Gllavata, J., R. Ewerth and B. Freisleben, "Text Detection in Images Based on Unsupervised Classification of High-Frequency Wavelet Coefficients", *International Conference on Pattern Recognition*, Vol. 1, pp. 425–428, 2004.

38. Liu, C., C. Wang and R. Dai, "Text Detection in Images Based on Unsupervised Classification of Edge-Based Features", *International Conference on Document Analysis and Recognition*, Vol. 2, pp. 610–614, 2005.

39. Yao, C., X. Bai and W. Liu, "A Unified Framework for Multioriented Text Detection and Recognition", *IEEE Transactions on Image Processing*, Vol. 23, No. 11, pp. 4737–4749, 2014.

40. Smith, R., "An Overview of the Tesseract OCR Engine", *International Conference on Document Analysis and Recognition*, Vol. 2, pp. 629–633, 2007.

41. Dalal, N. and B. Triggs, "Histograms of Oriented Gradients for Human Detection", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 1, pp. 886–893, 2005.

42. Chandrakala, S. and C. C. Sekhar, "Classification of Multi-Variate Varying Length Time Series Using Descriptive Statistical Features", S. Chaudhury, S. Mitra, C. A.

Murthy, P. S. Sastry and S. K. Pal (Editors), *Pattern Recognition and Machine Intelligence*, pp. 13–18, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

43. Ng, A. and M. Jordan, "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes", *Advances in Neural Information Processing Systems*, pp. 13–18, 2002.

44. Wang, J. and A. Cherian, "Discriminative Video Representation Learning Using Support Vector Classifiers", *arXiv 1909.02856*, 2019.

45. Chen, C. H., *Handbook of Pattern Recognition and Computer Vision*, World Scientific, 2016.

46. Zhang, H., "The Optimality of Naive Bayes", *International Florida Artificial Intelligence Research Society Conference*, Miami, Florida, USA, 2004.

47. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-Learn: Machine Learning in Python", *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830, 2011.

48. James, G., D. Witten, T. Hastie and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, NY, 2013.

49. Hastie, T., R. Tibshirani and J. Friedman, *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, Springer, NY, 2009.

50. Chollet, F., *Deep Learning with Python*, Manning, Shelter Island, NY 11964, 2018.

51. Maeda-Gutiérrez, V., C. E. Galván-Tejada, L. A. Zanella-Calzada, J. M. Celaya-Padilla, J. I. Galván-Tejada, H. Gamboa-Rosales, H. Luna-García, R. Magallanes-Quintanar, C. A. G. Méndez and C. A. Olvera-Olvera, "Comparison of Convolu-

tional Neural Network Architectures for Classification of Tomato Plant Diseases", *Applied Sciences*, Vol. 10, No. 4, 2020.

52. Li, F. F., R. Krishna and D. Xu, *CS231n: Convolutional Neural Networks for Visual Recognition*, Stanford University, 2020, http://cs231n.stanford.edu/index.html.

53. Kingma, D. P. and J. L. Ba, "Adam: A Method for Stochastic Optimization", *arXiv 1412.6980*, 2014.

54. Murphy, K. P., *Probabilistic Machine Learning: An introduction*, MIT Press, 2021.

55. Rumelhart, D., G. Hinton and R. Williams, "Learning Representations by Back-Propagating Errors", *Nature*, Vol. 323, pp. 533–536, 1986.

56. Duchi, J., E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, Vol. 12, pp. 2121–2159, 2011.

57. Ruder, S., "An Overview of Gradient Descent Optimization Algorithms", *arXiv 1609.04747*, 2017.

58. Scardapane, S. and D. Wang, "Randomness in Neural Networks: An Overview", *WIREs Data Mining and Knowledge Discovery*, Vol. 7, No. 2, p. e1200, 2017.

59. Wang, J., Y. Chen, S. Hao, X. Peng and L. Hu, "Deep Learning for Sensor-Based Activity Recognition: A survey", *Pattern Recognition Letters*, Vol. 119, pp. 3–11, 2019.

60. Vasudevan, C., *Concepts and Programming in PyTorch: A Way to Dive into the Technicality*, BPB, New Delhi-110002, 2018.

61. Simonyan, K. and A. Zisserman, "Very Deep Convolutional Networks for Large-

Scale Image Recognition", *arXiv 1409.1556*, 2014.

62. Truong, T., V. Nguyen and M. Tran, "Lightweight Deep Convolutional Network for Tiny Object Recognition", *International Conference on Pattern Recognition Applications and Methods*, pp. 675–682, 2018.

63. Stallkamp, J., M. Schlipsing, J. Salmen and C. Igel, "The German Traffic Sign Recognition Benchmark: A Multi-Class Classification Competition", *International Joint Conference on Neural Networks*, pp. 1453–1460, 2011.

64. Temel, D., T. Alshawi, M. H. Chen and G. AlRegib, "Challenging Environments for Traffic Sign Detection: Reliability Assessment under Inclement Conditions", *arXiv 1902.06857*, 2019.

65. Aggarwal, C. C. and P. S. Yu, "Outlier Detection for High Dimensional Data", *ACM SIGMOD Record*, Vol. 30, No. 2, pp. 37–46, 2001.

66. Chen, Z., C. K. Yeo, B. S. Lee, C. T. Lau and Y. Jin, "Evolutionary Multi-Objective Optimization Based Ensemble Autoencoders for Image Outlier Detection", *Neurocomputing*, Vol. 309, pp. 192–200, 2018.

67. Beggel, L., M. Pfeiffer and B. Bischl, "Robust Anomaly Detection in Images Using Adversarial Autoencoders", *arXiv 1901.06355*, 2019.

68. Flores, S., *Variational Autoencoders are Beautiful*, Stanford University, 2019, https://www.compthree.com/blog/autoencoder/.

69. Serengil, S. I., *Convolutional Autoencoder: Clustering Images with Neural Networks*, Stanford University, 2018, https://sefiks.com/2018/03/23/convolutional-autoencoder-clustering-images-with-neural-networks/.

70. He, K., X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", *arXiv 1512.03385*, 2015.

71. Timofte, R., K. Zimmermann and L. V. Gool, "Multi-View Traffic Sign Detection, Recognition, and 3D Localisation", *Workshop on Applications of Computer Vision*, pp. 1–8, 2009.

72. Canny, J., "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, pp. 679–698, 1986.

73. Hough, P. V. C., "Method and Means for Recognizing Complex Patterns", *US Patent 3,069,654*, 1962.

74. Liu, X. and Z. D. Y. Yang, "Recent Progress in Semantic Image Segmentation", *Artificial Intelligence Review*, Vol. 52, No. 2, pp. 1089–1106, 2018.

75. Ronneberger, O., P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", *arXiv 1505.04597*, 2015.

76. Paszke, A., A. Chaurasia, S. Kim and E. Culurciello, "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation", *arXiv 1606.02147*, 2016.

77. Puttagunta, M. and S. Ravi, "Medical Image Analysis Based on Deep Learning Approach", *Multimedia Tools and Applications*, 2021.

78. Karimov, A., A. Razumov, R. Manbatchurina, K. Simonova, I. Donets, A. Vlasova, Y. Khramtsova and K. Ushenin, "Comparison of UNet, ENet, and BoxENet for Segmentation of Mast Cells in Scans of Histological Slices", *International Multi-Conference on Engineering, Computer and Information Sciences*, 2019.

79. Platt, J., "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods", *Advances in Large Margin Classifiers*, Vol. 10, 2000.

80. Wu, T., C. Lin and R. C. Weng, "Probability Estimates for Multi-Class Classifi-

cation by Pairwise Coupling", *Journal of Machine Learning Research*, Vol. 5, pp. 975–1005, 2004.

81. Afşin, E., *Derin Öğrenme ile İki Boyutlu Optik Karakter Tanıma*, Süleyman Demirel University, 2017.

82. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, Vol. 3, No. 3, pp. 210–229, 1959.

83. Lin, Z., "How Can Machine Learning and Optimization Help Each Other Better?", *Journal of the Operations Research Society of China*, Vol. 3, pp. 341–351, 2020.

84. Domingos, P., "A Few Useful Things to Know About Machine Learning", *Communications of the ACM*, Vol. 55, pp. 78–87, 2012.

85. Roux, N. L., Y. Bengio and A. Fitzgibbon, *Optimization for Machine Learning*, MIT Press, 2012.

86. Bennet, K. P. and E. Parrado-Hernandez, "The Interplay of Optimization and Machine Learning Research", *Journal of Machine Learning Research*, Vol. 7, pp. 1265–1281, 2006.

APPENDIX A: CODE EXAMPLES

```
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten
model = Sequential()
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding='same',
kernel_initializer='he_uniform',input_shape=(32,32,3)))
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding='same',
kernel_initializer='he_uniform'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(128,activation='relu',kernel_initializer='he_uniform'))
model.add(Dense(43,activation='softmax'))
```

Figure A.1. Code for Baseline Architecture of GTSRB Classification.

```
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten, Dropout,
BatchNormalization
model = Sequential()
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding='same',
kernel_initializer='he_uniform',input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding='same',
kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same',
kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same',
kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128,activation='relu',kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(43,activation='softmax'))
```

Figure A.2. Code for The Best Architecture of GTSRB Classification.

```
#boundaries list includes first cutting point, second cutting point and original
#height information respectively for each segmented character.
#predictions list shows the prediction for each input as index number in the alphabet
#probabilities list shows the predicted probability values for each input for each
# character in the alphabet
import numpy as np
for i in range(len(predictions)):
    if(alphabet[predictions[i]] in ['.',','] and boundaries[i][0]<=0.1*boundaries[i][2]):
        predictions[i]=alphabet.index("'")
    elif(alphabet[predictions[i]]=='‘' and boundaries[i][0]>0.1*boundaries[i][2]):
        x=np.argmax(np.array([probabilities[i,alphabet.index('.')],
        probabilities[i,alphabet.index(',')]]))
        if(x==0): predictions[i]=alphabet.index('.')
        elif(x==1): predictions[i]=alphabet.index(',')
    elif(alphabet[predictions[i]]=='-' and boundaries[i][1]>0.7*boundaries[i][2]):
        x=np.argmax(np.array([probabilities[i,alphabet.index('.')],
        probabilities[i,alphabet.index(',')]]))
        if(x==0): predictions[i]=alphabet.index('.')
        elif(x==1): predictions[i]=alphabet.index(',')
    elif(alphabet[predictions[i]] in ['ı','I','i','İ','l','1']):
        if(boundaries[i][0]>0.5*boundaries[i][2]):
            x=np.argmax(np.array([probabilities[i,alphabet.index('.')],
            probabilities[i,alphabet.index(',')]]))
```

Figure A.3. Code for Extra Correction in OCR.

```
        if(x==0): predictions[i]=alphabet.index('.')
        elif(x==1): predictions[i]=alphabet.index(',')
    elif((boundaries[i][1]-boundaries[i][0])<=0.35*boundaries[i][2]):
        predictions[i]=alphabet.index("'")
    elif(0.35*(boundaries[i][2])<(boundaries[i][1]-boundaries[i][0])<=
    0.6*(boundaries[i][2])):
        predictions[i]=alphabet.index('ı')
    elif(0.6*(boundaries[i][2])<(boundaries[i][1]-boundaries[i][0])):
        x=np.argmax(np.array([probabilities[i,alphabet.index('I')],
        probabilities[i,alphabet.index('i')],probabilities[i,alphabet.index('İ')],
        probabilities[i,alphabet.index('l')],probabilities[i,alphabet.index('1')]]))
        if(x==0): predictions[i]=alphabet.index('I')
        elif(x==1): predictions[i]=alphabet.index('i')
        elif(x==2): predictions[i]=alphabet.index('İ')
        elif(x==3): predictions[i]=alphabet.index('l')
        elif(x==4): predictions[i]=alphabet.index('1')
elif(alphabet[predictions[i]] in ['C','Ç','O','P','S','Ş','U','V','W','X','Y','Z']
and boundaries[i][0]>=0.175*boundaries[i][2]):
    predictions[i]=predictions[i]+1
elif(alphabet[predictions[i]] in ['c','ç','o','p','s','ş','u','v','w','x','y','z']
and boundaries[i][0]<=0.175*boundaries[i][2]):
    predictions[i]=predictions[i]-1
```

Figure A.4. Code for Extra Correction in OCR (cont.).