Hochschule Rhein-Waal

Rhine-Waal University of Applied Sciences
Faculty of Technology and Bionics

Prof. Dr. Ronny Hartanto
Prof. Dr. Matthias Krauledat

Chess Piece Recognition using Machine Learning Technique

A Thesis Submitted in
Partial Fulfillment of the
Requirements of the Degree of
Bachelor of Science
in
Mechanical Engineering

By

Anoshan Indreswaran

Address
In den Langwiesen 70a
66849 Landstuhl
Deutschland

Matriculation No.:
13491

Date of Submission:
16-01-2016

Hochschule Rhein-Waal

Rhine-Waal University of Applied Sciences
Faculty of Technology and Bionics

Prof. Dr. Ronny Hartanto
Prof. Dr. Matthias Krauledat

Chess Piece Recognition using Machine Learning Technique

Bachelor Thesis

By

Anoshan Indreswaran
Matriculation No.: 13491

Abstract

The paper explains the development of a classification program which is to be integrated to a robot that will autonomously play chess. The problem is to perform a classification on a 12 class data set of chess pieces which works on a real-time video feed. Various approaches have been discussed for the problem; finally Convolutional Neural Network was chosen to develop the solution.

The paper develops two different approaches to solve the problem. One approach uses a single Convolutional Neural Network for classification. The other divides the problem into two classification task, classifying of the chess piece color and classifying the chess piece based on shape. The models developed from both the approaches are compared based on accuracy and computational time.

Cross validation was used to ensure the classification accuracy results are repeatable. To understand the accuracy in classifying each class, a confusion matrix was generated and the important features were visualized to understand the learnt features. The computation time for all the function calls on the program were measured and the average was calculated. Finally, the robustness of the 1-Step model to the change in object size was experimentally determined.

Declaration

I hereby declare in lieu of an oath that I have produced this work by myself. All used sources are listed in the bibliography and content taken directly or indirectly from other sources is marked as such. This work has not been submitted to any other board of examiners and has not yet been published. I am fully aware of the legal consequences of making a false declaration.

Place/Date

Signature

Acknowledgements

I would never have been able to finish my dissertation without the guidance of the Professors, help from friends, and support from my family.

I would like to express my deepest gratitude to my advisor, Prof. Dr. Matthias Krauledat and Prof. Dr. Ronny Hartanto and Dipl.-Inform. Regina Wolff, for their excellent guidance, patience, and support during the thesis.

I would like to thank Prof. Hartanto, who gave me access to the robotics lab to let me expand my knowledge to the subject of thesis and explore the vast field of robotics. Under his guidance, I was able to learn the practical issues beyond the textbooks. I would also like to thank Prof. Krauledat for encouraging my interest in pursuing a thesis in robotics, for guiding my research, and helping me ease through the problems faced during programming. I would also like to convey my gratitude to Ms. Wolff, for her patience and for her precious time she provided during the thesis with suggestions to improve the model.

I would also like to give a special thanks to the Professors, Members of the Faculty, and the University staff for their support and guidance throughout my Bachelor's degree. I would like to thank my family. They were always supporting me and encouraging me with their best wishes. I would also like to thank all my friends, especially Fabrice Hupez, Ilya Kamenschikov, Mariya Kamenschikova, and Lipika Kanojia, and my brother Itharshan Indreswaran who were always willing to help and give their best suggestions. My research would not have been possible without their helps.

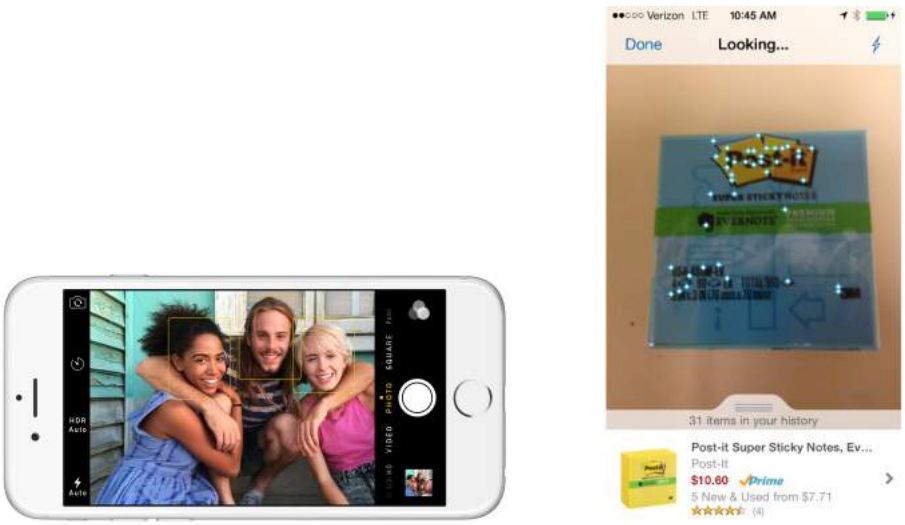
Contents

List of Figures

List of Tables

1 Introduction

In the recent years, image recognition has become successful in a short time span due to successful breakthrough in neural networks. Image processing has become a part of everyday life. We can see various utilities and uses of this platform; it is employed in field ranging from sales to computer vision.For example, Amazon, an electronic commerce firm, has integrated image recognition in their mobile application for iPhone so products can be searched by a click of an image as shown in figure 1.1b [13]. Further, as shown in figure 1.1a, mobile phones have face recognition to detect faces when capturing pictures. Furthermore, researches are being conducted on autonomous vehicles, humanoid robots, industrial manipulators and many other applications in which object recognition technology could play a vital role.



(a) Face detection (b) Amazon app

Figure 1.1: Two examples of image recognition

(a): iPhone camera uses image recognition to enclose the faces detected in yellow boxes [14]. (b): Amazon mobile app for iPhone uses image recognition to search for products using the image search option [13].

1.1 Motivation

Our goal is to develop a chess playing robot which uses camera to visually sense the chess pieces and the surroundings. The robot of interest is BAXTER, available in the robotics lab of the Rhine-Waal University of Applied Sciences for academic and re-

search purposes. The robot consists of two manipulator arms with a camera at each end, and a main display with camera as its head. Therefore, our main focus shifts from the hardware to the software development.

Figure 1.2: Baxter Robot playing Connect Four

Baxter playing Connect Four with a human at Robotronica, a robotics event in Brisbane, Australia on August 23rd, 2015 [2].

The software development for a chess playing robot can be simplified to the tasks given below:

- Object recognition- Identifying the different chess pieces, their positions, and the surrounding.
- Object position estimation- Identifying the coordinates of the chess pieces in 3D space of the chess pieces relative to the robot for manipulating the chess pieces.
- Object tracking- Keeping track of the chess moves from the opponent to identify the changes in the game.
- Chess game- The program would include the game methodology and rules for the execution of a well defined chess game.
- Object manipulation- The program to manipulate the chess pieces.

From the aforementioned tasks, our aim is to study the field of object recognition.

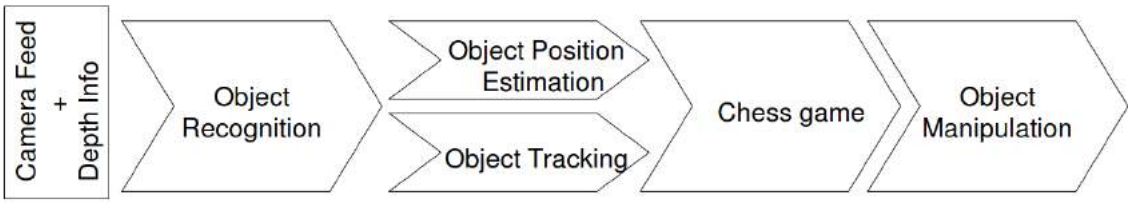


Figure 1.3: Typical model of the Software of the Chess playing robot.

The figure represents the processes flow of a chess playing robot.

Figure 1.3 shows the process flow of a chess playing robot. First, step is to recognize the objects using the camera feed and the depth information from the sensor. This information is used to estimate the position of the chess pieces an keep track of the changes in chess piece positions. Next, the algorithm to play the game devices a suitable move which will be handled by the object manipulation program.

In an image there will be multiple objects of which only some are of interest. The challenge is to identify these objects of interest and their position on the image. The problem of identifying the object is object classification and the problem of locating the object is object detection.

1.2 Objective

This paper focuses on the task of object classification. The objective is to develop an object classification architecture with the following key-points:

- Minimum computation time
- High classification accuracy
- Model should be able to learn new objects

In addition, the classification shall be performed from limited number of images that are shot with mobile phone camera. Further, machine learning techniques shall be used to develop the program.

1.3 Related Work

Serre and colleagues [15] construct a set of features for object recognition which is inspired by the primates visual cortex. Initially, a set of Gabor filters with 16 different scales, and 4 different orientations are applied for each of the input images to develop 64 feature maps called S1. The S1 maps are arranged in 8 bands that contain the feature maps of 2 consecutive scales with all four orientations for each. In the next step, the maximum of each orientation is taken to create 4 maps in every band; these are the C1 maps. From this, a random number of patches of various sizes, and all four orientations are extracted during the training, which will be used during testing.While testing, number of patches are extracted randomly from all positions of the image.

The Euclidean distance, a mathematical formula used to calculate the similarity between the images,extracts patch to each patch of the training set, which is calculated in every band to form the S1 feature maps. Finally, the maximum of all positions is calculated for every training patch. That implies, the most similar feature to the feature from the training is selected, and this is passed on to a classifier for classification.

Lillywhite, Lee & Beau Tippets [11] introduced Evolution-Constructed (ECO), features a novel approach for feature selection in object detection using genetic algorithm. In genetic algorithms a population of creatures, made up of genes, undergo evolution through a predetermined number of generations with only the fittest surviving. The evolution is carried out by creating cross-overs of the genes, and random mutations to its genes. While the survival of a creature is determined by a fitness score, the creatures that have a fitness score above a specified thresholded passes on to the following generation. An image region (x1, y1, x2, y2) and the transforms make-up a creature that undergoes the evolution. The fitness scores are assigned to the creatures using a single perceptron. After the evolution process, the remaining creatures will be the region of interests with apt ECO features that will be further processed for the object recognition. AdaBoost algorithm is used to enhance the classification of the perceptrons. Compared to the other available techniques which either focus on the domain of shape, texture or color, this algorithm combines various transforms which could assimilate in color, shape, and orientation. This approach intuitively seems to fit the task of image recognition because humans use a combination of such properties for differentiating between objects. Thus, in this paper we shall use this approach to identify the chess pieces.

1.4 Overview

Section 2: This section explains in detail, the theory that is necessary in understanding the image classification problem, and the technique used in the paper.

Section 3: The Methodology contains the solutions attempted for the classification problems, and a detailed description of the design of the Convolutional Neural Network used for classification.

Section 4:In this section, the hardware and software used int he program is explained. Further, details on the preparation of the data set is given and the steps in training

the Neural Network is explained.

Section 5: Experiments were conducted to study the overall accuracy of the model, the accuracy in classifying individual classes and the computation time. Further analysis was performed to understand the learned features. Further, it provides the results of the experiment to check the robustness of the model to changes in size of the object in the image.

Section 6: This section concludes the paper with a summary of the experiments conducted, and the results.

Section 7: The section suggests how the model can be improved on accuracy, modularity, and computational time.

2 Theory

Object classification is the ability of a system to identify an object in an environment (image, video, or other representation). The task sounds trivial as a human, but it is still a challenge to create a computer algorithm that would perform with similar efficiency as the human vision. This is because for a computer, an image is an array of numbers, interpreting this data into useful information is the task of image classification [10].

Some of the major challenges [7] in object recognition are given below:

- Viewpoint variation: An object may have different features depending on the different perspective.
- Varying illumination: Illumination plays a vital role to recognize an object. For example, an objects' colour varies depending on the illumination. In image recognition, illumination is a decisive factor in the image quality.
- Scale: An object can occur in different sizes in an image, so it becomes difficult to identify the target object.
- Occlusion: An image may not necessarily contain a single object. Other objects can occlude some parts of the main object.

The apparent solution for the classification problem is to compare the array of the image to be classified, to the known image. However, computationally, this is not the most efficient method. Thus, a common approach would be to use the features that would help in the classification [10].

This section explains in detail, the theory that is necessary in understanding the image classification problem, and the technique used in the paper.

2.1 Features

Feature extraction is the translation of matrix data into information such as edge or shape that could be used in identifying objects. Usually a group of features is used for better performance. Information from the group of features is used to form a hypotheses to model an object [7]. Most features are either template-based or histogram-

based. Template based features are suitable to capture a variety of object classes, but performs poorly when there is a transformation in the image.

Comparatively, the histogram based features are robust to transformations, but is not good at detecting objects in a variety of classes. In implementing an object classification algorithm the selection of features also plays a vital role in the performance [15].

2.1.1 Gabor filter

Gabor filter also extracts frequency information of an image, however instead extracting global information. The gabor filters extracts local information which is more robust to changes in image, yielding to better description of an object. Using the parameters in Gabor filter, features that are of particular orientation can be highlighted in an image. Gabor filters are commonly used in feature extraction and selective orientation extraction [1]. Figure 2.1 shows the output of Gabor filter with a randomly generated value for the kernel size on an image. The output of the filter is a blurry version of the original image.

Figure 2.1: Gabor Transform

Gabor filter applied on an image with a chess piece with randomly generated parameter values for kernel size.

2.1.2 Discrete Fourier Transform

Fourier transform is the representation of the image in its frequency domain. The transform stores the information as the number of times a pixel values occurs in the

image. The output of Fourier transform contains a magnitude, and a phase image; the magnitude values contains the geometrical information of the image [1]. An example of the magnitude image obtained by applying on a chess piece is given in the figure 2.2.

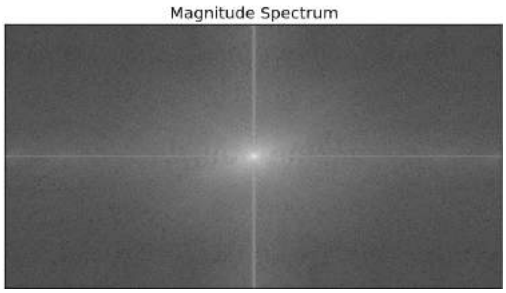


Figure 2.2: Discrete Fourier Transform

The image is the magnitude image obtained from the Fourier Transform applied on a chess piece image.

2.1.3 Canny Edges

Canny edges is a edge detection technique used in defining an object. The technique uses the derivative of the intensity information to locate edges. The changes in intensity will give rise to a higher gradient. The reason to choose Canny edges over the other detection algorithms is that it minimizes multiple edges and gives a clear output of edges [1]. Figure 2.3 shows the edges of a bishop extracted using the Canny edge technique.

Figure 2.3: Canny edges

Edges of the chess piece bishop extracted using the Canny edge function.

2.1.4 Color Histogram

Histogram of color was used to represent the color information of an object. On a global scale this does not act as a good feature, but when used in local, to define a particular region of an image, this is very vital, as color is an important feature in differentiating objects [1]. The histogram obtained from an image of a black chess piece is shown in figure 2.4. The histogram spikes at pixel value ranging from 0-50 and then again from 200-255 to value around 10,000.

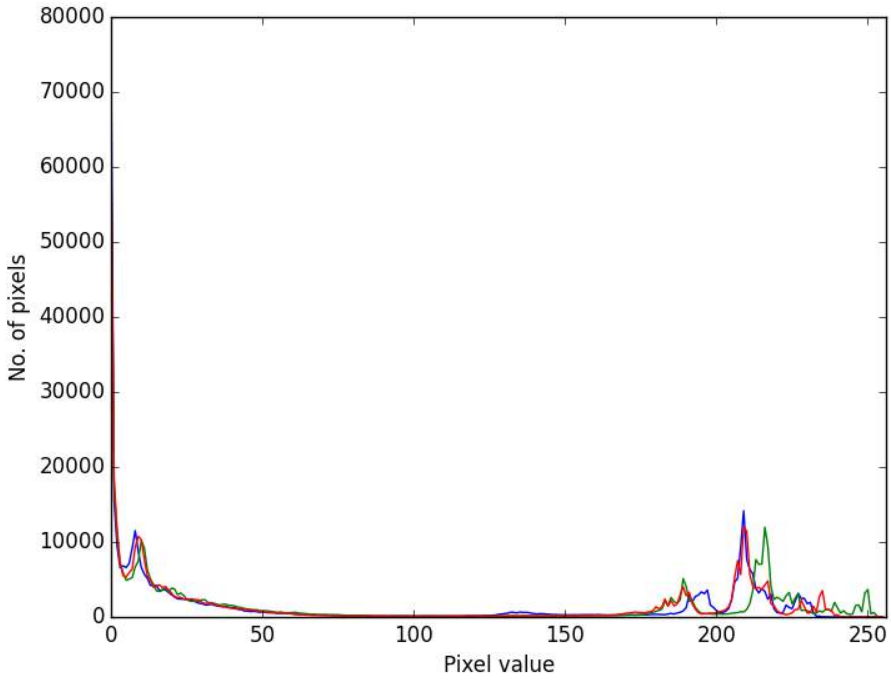


Figure 2.4: Hue Histogram

The hue values of an image containing the chess piece bishop have plotted as an histogram.

2.1.5 Gaussian Difference

Gaussian Difference extracts the potential features of an object and is invariant to scale and orientation, which makes it useful in object classification task because change in scale and orientation are two of the challenges in object classification. The difference of Gaussian can be associated with edge detection, as it removes the noise in an image and the homogeneous areas of an image [1]. The edges of the bishop are shown as white thick edges on figure 2.5, and the remaining image is black with some colored spots which are the noise from the original image.

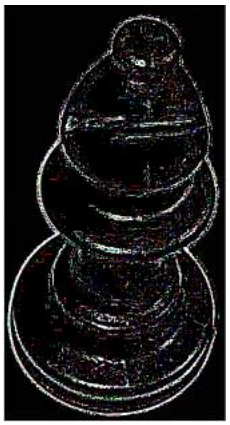


Figure 2.5: Gaussian Difference

Gaussian difference applied on an image of a bishop.

2.2 Genetic Algorithm

Genetic algorithm is a machine learning technique that assimilates the natural selection process through which it selects the fittest individual of the population. Individual is a possible solution in a set of possible solutions called population. An individual is defined by a set of parameters called genes that . The fitness value is a numerical score which is calculated for all the individuals using a predefined function based on the problem [12].

In object classification, a feature that defines the object is an individual and how well the feature helps in classification is the fitness score of the feature. The solutions are altered by mutation and crossover to generate possible new solutions and threshold using the fitness scores. Then from all the available solutions a selected number of solutions is chosen at random. The above mentioned processes are repeated till desired fitness score is achieved, a preset number of iterations or a predefined amount of variation in the individuals [12].

2.3 Neural Network

Therefore we use a data driven approach, that is we write an algorithm that learns from the priory labelled data to classify data. One such technique is the Neural Network, also called the Multi-Layer Perceptron [10].

A neural network has two components: a score function that maps the raw data into class information, and a loss function that quantifies the agreement between the pre-

diction and the true label [10].

A neural network consists of hidden layers and an output layer, which classifies the input based on linear combination of the input [10].

2.3.1 Score function

A single layer of the neural network can be represented as a linear mapping function as follows:

Score Function

$$f(x_i, W, b) = Wx_i + b$$

W: Weight Matrix

b: Bias Matrix

x_i: *Input Vector*

The input to the neural network is a vector of size $[D \times 1]$ and the size of the weight matrix is $[K \times D]$. The weight matrix quantifies the significance of every pixel in the classification, thus the size of the weight matrix can be understood as the mapping of the significant pixels (D) for the different classes (K). The matrix multiplication of the input image and the weight matrix is added with the bias vector of size $[K \times 1]$ to the network [10].

These are the typical operations in a single layer of a neural network. In a feed forward network multiple layers are used and the intermediate classifications are random classifications which are trained using the loss function attached to the output layer [10].

2.3.2 Loss function

The loss function is a measure of the performance of the classifier, it will be high if the model is performing poorly. This measure is used to update the weights to achieve better classification accuracy, and this in return influences the score vector such that the score of the right class is high. The two common loss functions are multi-class Support Vector Machines(SVM) loss and the Softmax classifier [10].

Hinge Loss Function

$$L_i = \sum_{j \neq y_i} max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)$$

Loss Function: L_i

Score Function: f(x,W)

Classes: j

True class: y_i

Fixed Margin Value: Δ

In the hinge loss, the difference between the score and the true label's score is calculated which is then summed with a fixed margin value δ. This value is threshholded for maximum value with 0. In the same manner, the calculation is done for all score in the score vector and the sum is the Hinge loss [10].

$$L_i = -\ln\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

On the other hand, the cross entropy loss takes the natural exponent for all the scores and then normalizes the values to add up to one. Then the negative log of the calculated value gives the loss [10].

The difference in calculating the losses makes them ideal for different data sets. The cross entropy loss will have a loss greater than zero even if the score of the correct class is high, on the other hand the output of the hinge loss will be zero if the difference in the correct class score and the other labels are higher than the fixed margin value δ. Further, using the Softmax classifier the confidence value for all the classes can be calculated, which will allow us to compare the confidence of the predicted classes to the rest [10].

2.3.3 Solver

The cross entropy loss function was used to calculate the performance of the weights, yet the performance need to be optimized. This can be done by updating the weights to lead to a lower loss, however the problem cannot be analytically solved due to the high dimensionality of the loss function which is a linear combination of the weights. Thus, the weight updates should be in the direction to reach the minimum, which is done be following the gradient. The gradients are calculated numerically using a step size of α, which is called the learning rate [10].

The learning rate is to be determined experimentally. Since the function is multidimensional, and not differentiable, there are many local minima's, and if a higher learning rate is used minima's may be missed. On the other hand, using a smaller learning rate will slow down the learning. Thus a learning rate which is suitable for the model should

be set during the training phase empirically. This process of updating the weight based on the gradient is called gradient descent [10]. According to Bottou, gradient descent is an ideal solver when the training time is the bottle neck. Even though, the training set was not large to select the Stochastic Gradient Descent(SGD) solver, the essence of time lead to the choice of the SGD [3].

To achieve a lower loss, the weights at each layer needs to be updated based on how much the layer had affected the classification. For this purpose, the back-propagation algorithm is used. That is, using back propagation, the required change in a weight can be calculated to achieve the desired result. This is done by chain rule on the partial derivatives [10].

2.3.4 Activation function

The human sensory system receives many information, of which not all are necessary in perceiving the environment. So the neurons function as a processing unit to extract the useful information from the multitude of data it receives. For creating a computation model, the simplified process happening in a neuron can be explained as a point where the weighted sums of the data are thresholded using a function which fires if the total sum, is above a certain value. In artificial intelligence, the function which is used to threshold the weighted sum is the Activation function [10].

The activation function is used immediately after performing weighted sum operations such as a fully connected layer, and a convolution layer. It can be understood as a function which evaluates the importance of the filters [10].

Initially, sigmoid function was the most commonly used activation function. This function thresholds the values between 0 and 1 [10]. However, it has drawbacks such as:

The outputs are not zero centered: The the outputs are almost always positive, which affects the dynamics of the gradient update during the learning process [10].

Saturation: The outputs of a Sigmoid function are almost always a zero or one. During back-propagation, the product of the gradient is multiplied with the output at the node. The value of a gradient is decisive factor in learning. So if the gradient is almost zero, the back-propagation algorithm's output will always be zero, which means the weights are not being updated. As a result, the Convolutional Neural Network does not learn that feature [10].

There are many activation functions such as the Sigmoid function. At the moment, the Activation function- Rectified Linear Unit (ReLU) is widely used due to its simple nature in calculation and it does not have the cons of saturation. The basic ReLU function is a maximum threshold function of zero and the x-value. Yet, a major drawback when using the ReLU activation is that the learned feature can die; it become completely zero. This may occur, if the gradient is very high. However, the suggested solution to this is to use smaller learning rates, which ensures no steep change in gradients are experienced by the Convolutional Neural Network [10].

2.4 Convolutional Neural Network

Convolutional Neural Network is a variant of the Multilayer Perceptron which is pecially used in computer vision [10].

In Convolutional neural network the images are filtered through a number of convolution and pooling layer before it arrives at a fully connected layer which performs the classification [10].

2.4.1 Convolutional Layer

A convolution layer is a filter (that is, a kernel or a matrix) coupled with weights which are learned during training. The kernel convolves through all the channels of the images and over the entire height and width of the image [10].

Example of Convolution process:
The dot product of the (5×5) image matrix and the (3×3) kernel is calculated for every position in the matrix, resulting in an activation map of size (3×3).

$$\begin{bmatrix} 250 & 230 & 220 & 226 & 187 \\ 234 & 23 & 65 & 23 & 96 \\ 12 & 45 & 23 & 34 & 124 \\ 234 & 23 & 65 & 23 & 96 \\ 12 & 45 & 23 & 34 & 124 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 323 & 111 & 184 \\ 80 & 102 & 181 \\ 322 & 111 & 184 \end{bmatrix}$$

The convolutional layer learns features of the object; the first convolution layer acts as an edge detector, and the Convolutional layers further in the net learns abstract features.

2.4.2 Pooling Layer

A Pooling layer is used in the Convolutional Neural Network to reduce the spatial size of the activation maps, thus reducing the computation cost. Furthermore, using a maximum pool layer expedites the learnig of the network [4].

A kernel of size $[a \times a]$ is coupled with a function, which is commonly a maximum or an average function, is slid over the activation map following which the function output is passed on to the next layer [10]. The parameters that can be set are the filter size, and the stride. The size of the output can be calculated using the following equation:

Pooling Function

$$W_o = \frac{(W_i - F)}{S} + 1$$

Output Layer Size: W_o

Input Layer Size: W_i

Filter Size: F

Stride: S

As an example, using the above equation, if we calculate the output size of an image of size $[256 \times 256]$ pixels through a pooling layer of filter size $[2 \times 2]$, and a stride of two will be $[128 \times 128]$ pixels. By taking the ratio of the number of final pixels to the initial pixels, it is evident that the size has reduced by a three-fourth, which a huge loss in information. Therefore, smaller pooling filters, with a lower stride, is a better choice to not lose vital data [10].

3 Methodology

The Methodology contains the solutions attempted for the classification problem and a detailed description of the design of the Convolutional Neural Network used for classification.

3.1 Classification using ECO-features

Despite of Convolutional Neural Networks being the state-of-the-art in image classification, the ECO-Features were intended to be used for the classification task. This was because, in ECO-features the filters that define the feature set are manually define. Intuitively, this seemed ideal because by selecting filters that defines an image using its shape, color and texture a good set of features could be generated.

Gabor filters, Canny Edges, and Gaussian Difference were implemented for extracting shape and orientation features. For color information, histogram of colors of HSV-color space was used, and histogram equalization technique was used to achieve better contrast in colors. While developing the genetic algorithm to create ECO-features for the chess pieces, it was found, from the full report of the ECO feature, that when deployed with sliding window technique, classification will take more than a minute. Therefore, the development of a model using genetic algorithm was put on a halt and other alternatives were considered.

The immediate alternative was Convolutional Neural Network. Thus from this point onwards, the paper will discuss how the Convolutional Neural Network was used for the classification problem.

3.2 Classification using Convolutional Neural Network

There are many factors affecting the performance of a Convolutional Neural Network, such as: the weight initialization, the activation function, the classifier, and the architecture of the network.

Weight initialization is the initial assigning of weights to the neurons. Weights are initialized by assigning constant values, or at random for example, Gaussian weight filler which uses a fixed standard deviation value was introduced. When these techniques were

used, the weights were not converging when the number of layers in a network increased. Thus, Glorot and Bongio, suggested a technique, which is called Xavier initialization in the Caffe library [5], [6]. This fills the weights based on the number of inputs, and number of outputs in a particular layer. In this project, the weights were initialized using Xavier initialization as it speeds up the training and has a higher probability for convergence [5]. The biases in the fully connected layers were initialized with a constant value of zero.

For activation, as it was explained in subsection 2.3.4, Rectified Linear Units (ReLU) are a better choice compared to the other available functions like sigmoid or hyperbolic tangent because there is no saturation of weights which would produce dead filters. In addition, the ReLU function is computationally cheap and it converges fast. In the report, we have treated the ReLU layer as a part of the layer on which the activation function is applied, even though while defining the architecture the ReLU layers were defined as separate layers.

For classification, a Softmax classifier was used in all the models. A Softmax classifier is a feed forward network which uses Softmax loss function to train the network. This classifier was preferred over the Support Vector Machine (SVM) because of the loss functions nature to always reduce the back propagated loss value. This means, that the output could be interpreted as a confidence level in the classification.

An image is fed into a feature constructor which learns features from the image, that are then flattened into a vector and passed onto the classifier for prediction. This is the simplest structure in every Convolutional Neural Network. This project presents two architectures for classifying the chess pieces: The 1-Step Model which has a single classifier and the 2-Step Model has two classifiers. The models are explained in detail in the following subsections.

3.2.1 1-Step Model

The 1-Step Model works with a color image to predict the results, unlike in the 2-Step Model (elaborated below), in this model no prior knowledge is used to breakdown the logic. The input is a HSV image from which features are learned and then used for classification.

In figure 3.1 the architecture of the 1-Step Model is presented. The neural network

consists of 7 layers excluding the input layer and considering the ReLU layers as a part of the layer (i.e. convolution layer or fully connected layer) which it activates. The first 5 layers are for constructing features and the last two layers are for classifying.

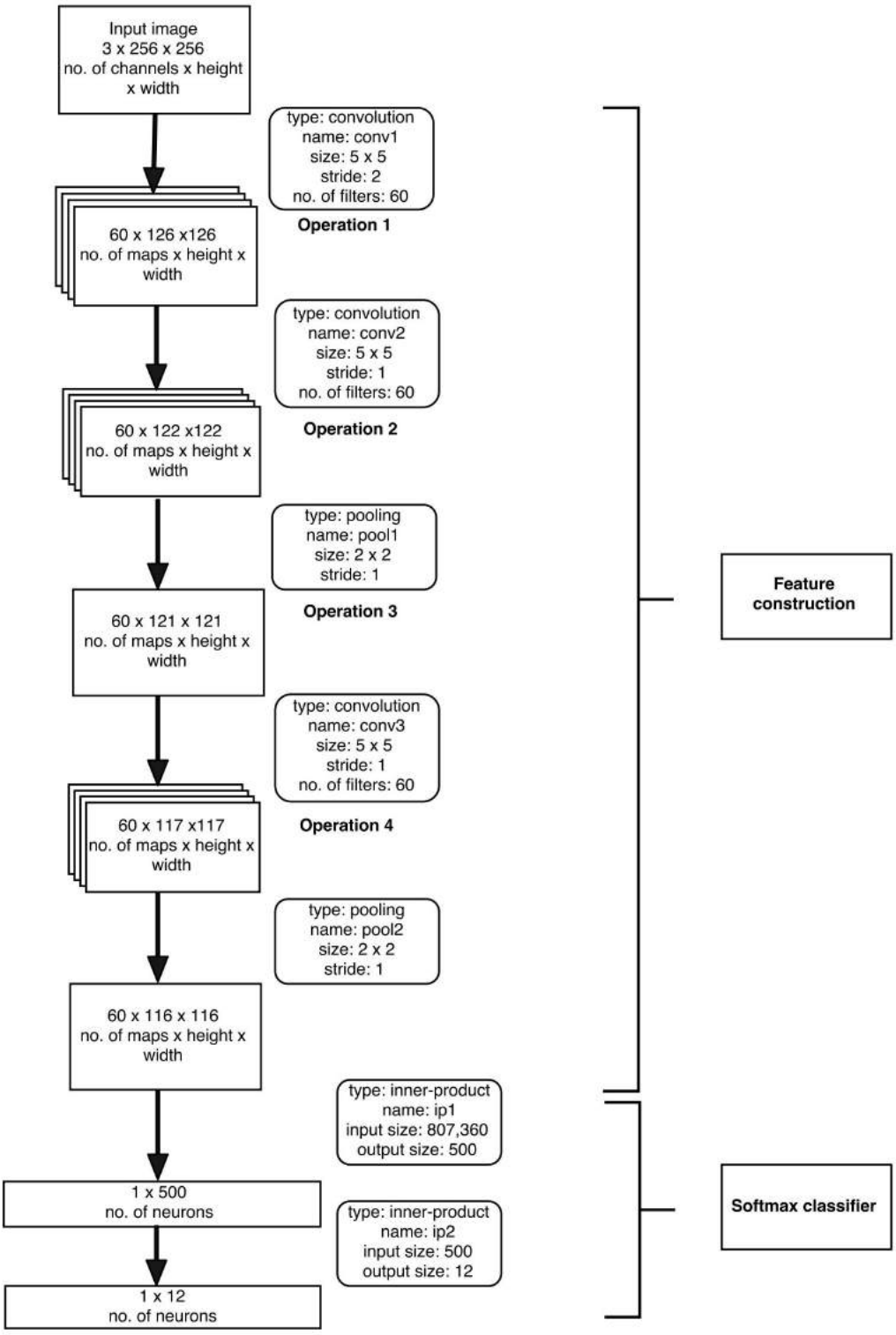
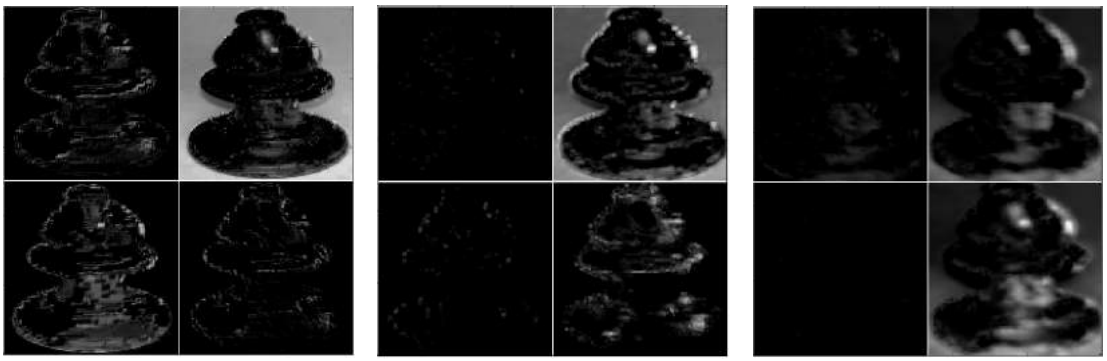
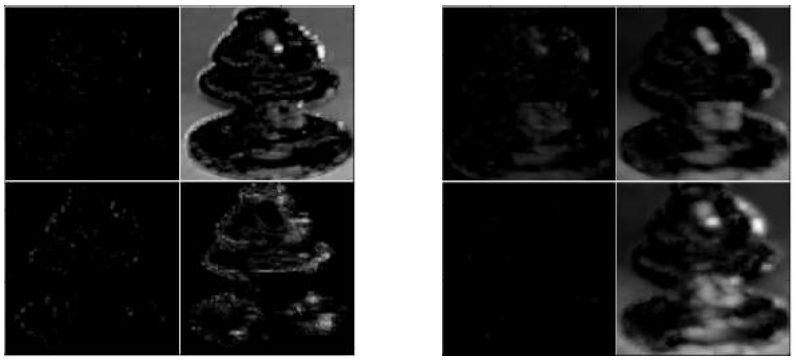


Figure 3.1: Architecture of the 1-Step Model

The layers, and the parameter settings of the 1-Step Model are given in the figure. The figure also includes the input and output at each layer and their size. The model classifies the images into the 12 chess pieces.



(a) convolution layer 1 (b) convolution layer 2 (c) pooling layer 1



(d) convolution layer 3 (e) pooling layer 2

Figure 3.2: Outputs of the feature constructing layers in the 1-Step Model

The figure displays the outputs of all the layers constructing features that are forwarded to the classifier. (a) Output of the first layer that learns lower level features such as lines, and edges. (b) Output of the second layer that learns higher level features. (c) Output of the third layer that reduces the image size. (d) Output of the fourth layer that learns a complex features. (e) Output of the fifth layer in which the data size is further reduced.

An image of size 256×256 pixels is input to the feature constructor which first passes through two consecutive convolution + ReLU layer with 60 convolution filters of size 5×5 in each convolution layer. For the first convolution layer a stride of two was used, however, for the second, a stride of only one was used so no important information is lost. The first convolution layer was intended to work as a edge detector while the second one was intended for a higher feature such as a line or a edge. Then the output is pooled for the maximum with a filter of size $[2 \times 2]$ and a stride of one. Next, the output of the pooling layer is input to the convolution+ReLU layer with a Convolutional layer consisting of 60 filters of size 5×5 and a stride of one. Which is followed by the final pooling layer of size $[2 \times 2]$ and stride 1 which also pools for the maximum. The pooling layers were used to reduce the size of the image, and filter only the important information. Further pooling layers help in reducing over-fitting of data. Figure 3.2

shows the output of all the convolution and pooling layers of the 1-Step Model.

The classifier is a simple feed forward network with two layers- one hidden+ ReLU layer, and the output layer. There are 500 neurons in the first layer which receives an input of size 807,360 pixels followed by the output layer of size 12 representing the 12 object classes.

3.2.2 2-Step Model

The chess pieces can be grouped into 2 classes based on colour, and six different classes based on geometry. This information was used to develop a model which classifies the feed based on color - the Binary classifier and the one based on geometry- the piece-wise classifier.

The Binary classifier processes an image in HSV color-space and predicts the color of the chess piece. The piece-wise classifier processes an gray-scale image and classifies the input images according to the piece geometry. Then the prediction is made depending upon the combination of the output of both the classifiers. The figure 3.3 shows the architecture of the Binary classifier. This model has two layers for features extraction and two layers for classification; in total, a number of 4 layers in the whole network.

The layer for feature extraction is a convolution layer of size $[3 \times 3]$ with a stride of two and 20 filters. Since only the information of color of the piece was required using a higher stride in a low filter size was acceptable. The pooling layer is of size $[3 \times 3]$ with stride one pooling for the maximum.

A Softmax classifier with a hidden layer of size 200 neurons was used which received an input of 16,820 pixels. The output of the 200 neurons is fed into the output layer which had 2 neurons according to the number of classes, black and white.

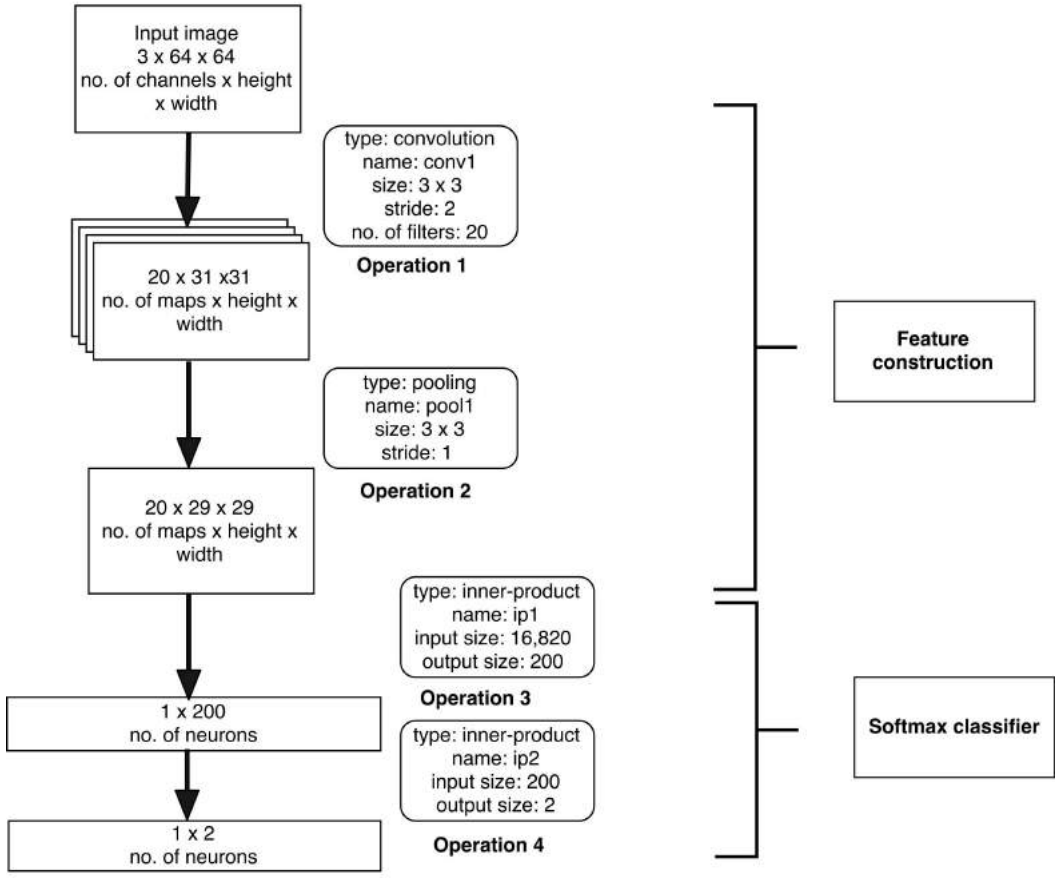


Figure 3.3: Architecture of the Binary classifier in the 2-Step model

The layers of the network and their parameter values with the inputs and outputs are shown in the figure. The network performs a binary classification to predict the color of the chess piece.

The figure 3.4 depicts the piece-wise classifier's structure. The network has a depth of six layers excluding the input layer and considering the ReLU layers as a part of the layer (i.e. convolution layer or fully connected layer) which it activates. The first 4 layers are the feature constructing layers and the last two layers are for classifying.

The feature constructor is made of two consecutive convolution+ ReLU+pooling layer combinations which receives an input of gray-scale image of size 256×256 pixels. Each convolution layer has a filter size of $[5 \times 5]$ with a stride of one, with the first convolution layer having 20 filters and the last one having 50 filters. The pooling layers are of filter size $[2 \times 2]$ with stride two to achieve lower computation time in the network by reducing the data required to be processed.

The classifier is similar to the ones in previous models with two layers- one hidden+ ReLU layer, and the output layer. The hidden layer has 500 neurons with an input of

size 186,050 pixels which is then passed on to the output layer with six neurons for the six object classes.

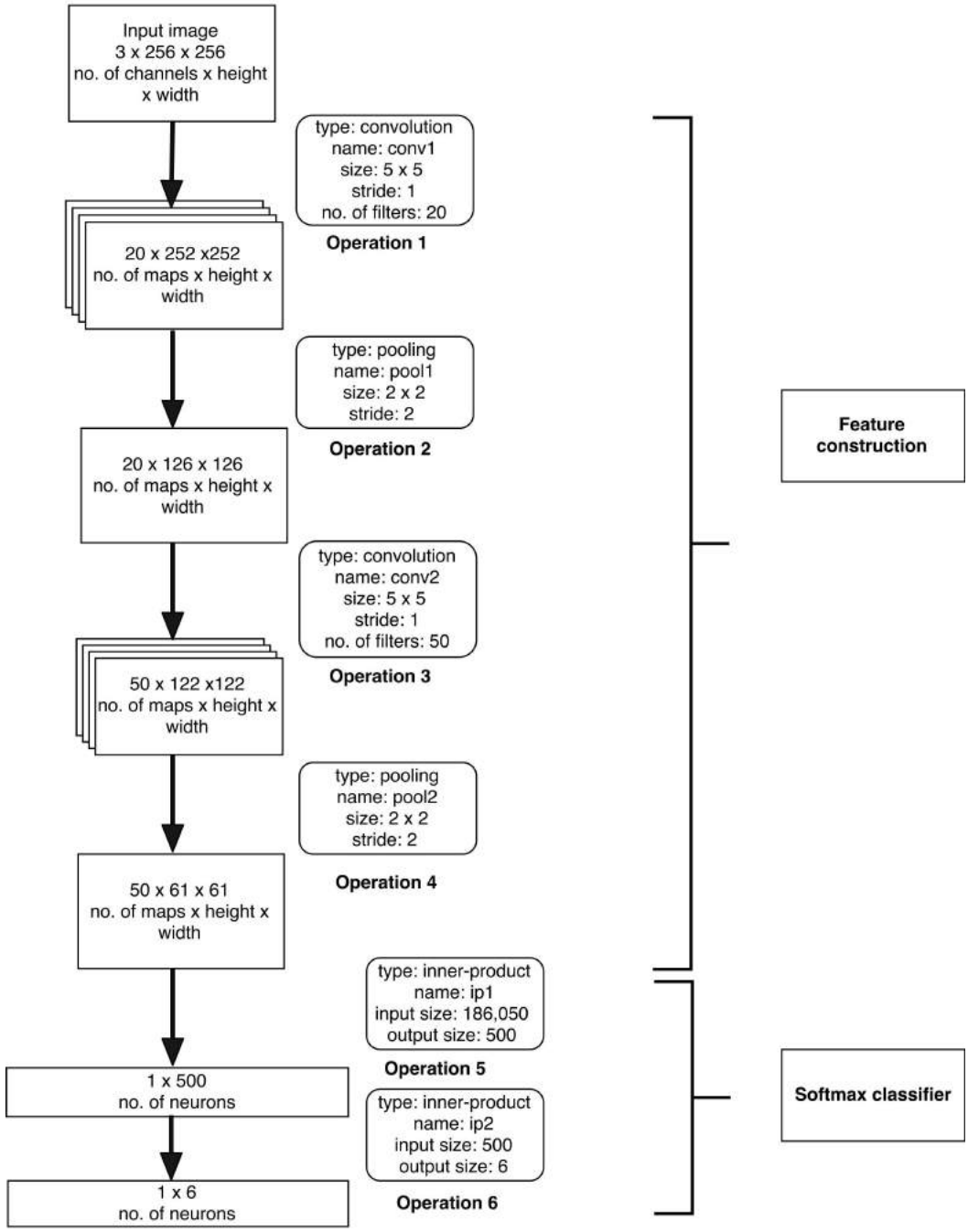


Figure 3.4: Architecture of the Shape-based Classifier of the 2-Step Model

4 Implementation

In this section, the hardware and software used int he program is explained. Further, details on the preparation of the data set is given and the steps in training the Neural Network is explained.

4.1 Hardware and Software

The hardware requirement for the project was low, as the focus was on software. The hardware used for the project was a laptop, and a camera. The software's used in the program were C++, python and their libraries.

The laptop had an Intel i3 technology processor of frequency 2.1 GHz and 4 GB of memory. The laptop was used to develop, train and test the Convolutional neural network. The camera was used to capture the necessary video and images for training and testing the network. It was the rear camera of Samsung Note II with 8 MP.

Libraries such as Numpy, OpenCV, Matplotlib, and Caffe were used to ease the process. Numpy was used in array operations and for statistical calculations. OpenCV was used in the initial phase to develop the feature set for ECO-features. During the use of Convolutional neural networks, OpenCV was used in converting the color space of the image. Matplotlib was used for visualizing the outputs of the network and to plot graphs from the data collected data. Caffe is a C++ library used to develop the neural network. Further details of Caffe is explained in detail in the sub-subsection 4.1.1.

4.1.1 Caffe

Caffe is an open-source deep learning C++ framework with Python and MATLAB interface. The Caffe deep learning model definitions are written in Protocol Buffer Language which makes it simple for implementation [8].

The main components of a Caffe model are the blobs, layers, solver. Caffe stores data in 4 dimensional arrays called blobs, which are passed on to layers where the data is processed and passed forward to the following layer. In addition, the layers calculate the gradient depending on its input and the parameters and passes it on to the previous layers. Some of the available layers are Convolutional layer, pooling layer, inner products, rectified linear, and Softmax loss. The solver is responsible for optimizing the model by

parameter updates [9].

Caffe was the choice of library because it was developed for computer vision purpose and is more dedicated to Convolutional neural networks. There are several pretrained models and examples of object classification, and detection using Caffe. As a beginner, it was helpful to get an understanding of the basics in using the library.

In addition, Caffe is highly modular in design. Since model definitions are done in Protocol Buffer language, changing the architecture of a model or the parameters can be done without writing lines of code [8].

Further, since it is a C++ library with MATLAB and Python bindings, there was a flexibility in switching between the different programming languages [8]. For example, for faster classification, the Caffe C++ implementation is better. As for analysis of the network, python is preferable because with the least code a network can be analysed by exploiting other available libraries of Python such as numpy for numerical operations, Matplotlib for plotting diagrams, and os for shell commands.

Another reason to select Caffe was that new custom layers can be implemented in Caffe architecture which shall be useful when experimenting with new techniques areas such as image segmentation and object detection [9].

4.2 Dataset

The data set was prepared using the 8MP rear camera of Samsung Galaxy Note II from the chess pieces shown in figure 4.1. First, 60 second long videos of frame size 1920×1080 pixels of all 12 distinct chess pieces were captured on a white background. The video was taken in well lit area with day light. The chess pieces were recorded from all around to have different orientations.

Next, a database of 50 images per class was encoded in JPEG by extracting frames from the video using FFmpeg tools. The images were then cropped manually to center the object in the image, this resulted in images of various sizes starting from a size of 300×300 pixels.

Next, using python script and OpenCV library the cropped images were converted to a grayscale and a HSV color space image data sets. The contrast of the grayscale images were increased for clarity. Figure 4.2 shows the results of the individual processes on the image.

The data sets were re-sized to a size of 256×256 and encoded with labels as LMDB files. Three different types of LMDB data was generated: 12 class HSV, two class HSV, and six class grayscale images.

The HSV data set was encoded with 12 labels each denoting a different chess piece.

$[\ 0 : 11\ ] = \{$Black Bishop, Black Castle, Black King, Black Knight, Black Pawn, Black Queen, White Bishop, White Castle, White King, White Knight, White Pawn, White Queen$\}$

Another data set was prepared with two labels to denote black and white pieces.

$[\ 0 : 1\ ] = \{$Black, White$\}$

From the gray images, a six class data set which defined the chess pieces regardless of color.

$[\ 0 : 5\ ] = \{$Bishop, Castle, King, Knight, Pawn, Queen$\}$

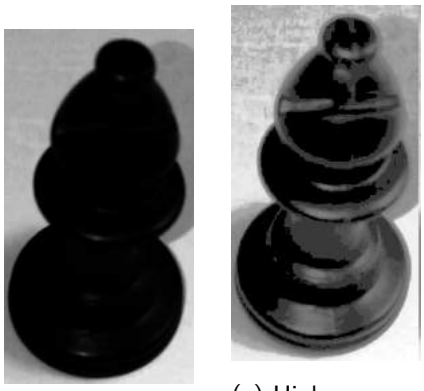
(a) BK (b) BQ (c) BB (d) BKn (e) BC (f) BP

(g) WK (h) WQ (i) WB (j) WKn (k) WC (l) WP

Figure 4.1: Chess Pieces/ data set

The chess pieces that were used to prepare the data set.

(a) Original (b) Image crop (c) HSV



(d) Grayscale (e) High contrast

Figure 4.2: Image pre-processes

The figure shows the different processes performed on the original image. (a) The original image captured from the video. (b) The image was cropped to fit the image to the object size. (c) The cropped image was converted to HSV color space to perform 12 class classification and binary classification. (d) The cropped image was converted to grayscale to perform six class classification. (e) The contrast of the grayscale image was enhanced for clarity.

4.3 Training

The network learns from examples and regulates its learning by losses passed through the network. The learning method, the learning rates, and the required training time are important factor to be considered during training.

The stochastic gradient method was chosen for weight updates as it converges fast when used in networks with ReLU activation. The stochastic gradient descent method is a suitable choice if lower training time is necessary.

Since lower learning rate converges slow and the higher learning rate may settle at a local minimum of the loss, the right learning rate for a task is to be found through trial

and error [10].

Initially a learning rate of 0.001 was used, but the program stopped with an error due to very high loss. This is usually caused due to divergence in learning that is when the weight values are diverging, which can be understood as the classification not improving over the iterations, and in turn the Loss function producing higher losses to improve the classification.

The learning rate was reduced to 0.00001 and an average accuracy of 35% was achieved to classify among six classes on grayscale images. The same network architecture as the 1-Step Model, except in the output layer in which the number of neurons were reduced to 6, was used in the grayscale image training. On the other hand the 1-Step Model yielded an average accuracy of 70% when trained with the same learning rate.

A probable reason for the difference in performance was that the shape features are not clear enough in the grayscale images as in color images because the edges in color images are well defined comparatively. Thus, to increase the clarity in the images, the contrast in the grayscale images were increased by histogram equalization technique.

The grayscale images with higher contrast were now used for training and an average accuracy of 62% was reached. Since our goal is to achieve better accuracy with the minimalistic processing power and memory usage. A new model with fewer layers and filters were developed and trained with the high contrast grayscale images, a classification accuracy of 58% was reached. This model is the piece-wise classifier in the 2-Step model which was explained in the subsection 3.2.2.

Training the network was the most time consuming task in developing a neural network. In the project, the time required to train the 1-Step Model was on average 5 hours and the time required. The time required to train the six class classifier of the 2-Step model was on average 3 hours, and 5 minutes for the Binary classifier of the same model. The 2-Step model is faster to train compared to the 1-Step Model.

5 Experiments and Results

Experiments were conducted to study the overall accuracy of the model, the accuracy in classifying individual classes and the computation time. Further analysis was performed to understand the learned features. Further, it provides the results of the experiment to check the robustness of the model to changes in size of the object in the image.

5.1 Accuracy

The data was split into five subsets from which one subset was used for testing and the remaining four subsets were used for training. This was done in a repetitive fashion by choosing a new subset from the initial split as the testing set and the remaining as the training until all the subsets were used as a test set. This technique ensures the repeatability of the model for this classification task and the performance of the model in an independent testing set.

The average accuracy of the 1-step model 12 classes is 70%, while the overall average accuracy of the 2-step model is 58%. In the 2-step model, the Binary classifier reached 100% accuracy, and the six class classifier processing on gray images reached an average accuracy 58%.

The tables below gives an displays the average achieved by each classifier on all the training sets and the average.

Table 1: Cross Validation Results of the 1-Step Model

The table gives the results of the five fold cross validation performed on the 1-Step Model and model's average accuracy.

Iteration	**1st**	**2nd**	**3rd**	**4th**	**5th**	**Average**
Accuracy	0.7	0.6	0.75	0.75	0.7	**0.7**

5.2 Classwise Accuracy

The accuracy of the model obtained through cross validation still lacks information on how well does the architecture classifies the individual classes. So a classification test

Table 2: Cross Validation Results of the Binary 2-Step Model

The table below is the five fold cross validation results of the 2-Step Model. The Binary classifier's accuracy represents how well the model predicts the class and the Piece-wise classifier's accuracy represents how well the piece name is predicted right. The product of both the accuracies gives the Overall accuracy.

Iteration	**1st**	**2nd**	**3rd**	**4th**	**5th**	**Average**
Binary classifier's Accuracy	1.0	1.0	1.0	1.0	1.0	**1.0**
Piece-wise classifier's Accuracy	0.5	0.8	0.7	0.6	0.3	**0.58**
Overall Accuracy	0.5	0.8	0.7	0.6	0.3	**0.58**

was run through the 600 image data set which contained 50 images of each class and the prediction of each image was recorded. Using this data, the confusion matrix was generated; the columns are the predicted classes, and the rows are the object's real label.

From table 3, it can be inferred that the average classification accuracy for the black chess pieces is 91.0% with a standard deviation of 4.9%. However, the classification accuracy for the white chess pieces is 77.3% with a standard deviation of 9.0%. The difference in the accuracies suggests there may be a bias towards the black chess pieces. Yet from the table it is clear that the white pieces are never misclassified into a black piece, but to another white pieces, mostly being the white bishop. Thus, to visualize the trade off between the accuracy and precision, we plot the accuracy vs precision graph for all the classes.

From the figure 5.1, we can observe that the white pawn and white knight has a bad accuracy, and the white bishop has an accuracy above 90% but a precision less than 60% due to many false positives. Thus, we can assume, the network has a higher bias for the white bishop. Besides, the classification accuracies of the remaining chess pieces are acceptable.

Table 3: Confusion Matrix of the 1-Step Model

The rows represent the right labels of the image and the colums represent the predicted labels by the model. In an ideal case, the results would be a diagonal matrix. The values besides diagonal are the wrong classification of one class as another which is known as false positives. The accuracy is the number of right predictions as a ratio to the number of tested images. The precision is the ratio of the number of right predictions to the total number of positive predictions of that label.

Label	**Prediction**												**Accuracy**
	BB	**BC**	**BK**	**BKn**	**BP**	**BQ**	**WB**	**WC**	**WK**	**WKn**	**WP**	**WQ**	
BB	47	0	2	0	1	0	0	0	0	0	0	0	**0.94**
BC	0	44	2	0	0	4	0	0	0	0	0	0	**0.88**
BK	0	0	46	0	0	4	0	0	0	0	0	0	**0.92**
BKn	0	0	6	42	0	2	0	0	0	0	0	0	**0.84**
BP	0	0	2	2	45	1	0	0	0	0	0	0	**0.90**
BQ	0	0	1	0	0	49	0	0	0	0	0	0	**0.98**
WB	0	0	0	0	0	0	46	0	1	0	2	1	**0.92**
WC	0	0	0	0	0	0	3	38	3	0	5	1	**0.76**
WK	0	0	0	0	0	0	11	0	38	0	0	1	**0.76**
WKn	0	0	0	0	0	0	9	0	8	32	1	0	**0.64**
WP	0	0	0	0	0	0	4	4	3	0	38	1	**0.76**
WQ	0	0	0	0	0	0	5	0	4	0	1	40	**0.80**
Precision	1.0	1.0	0.78	0.95	0.98	0.81	0.58	0.9	0.67	1.0	0.81	0.91	

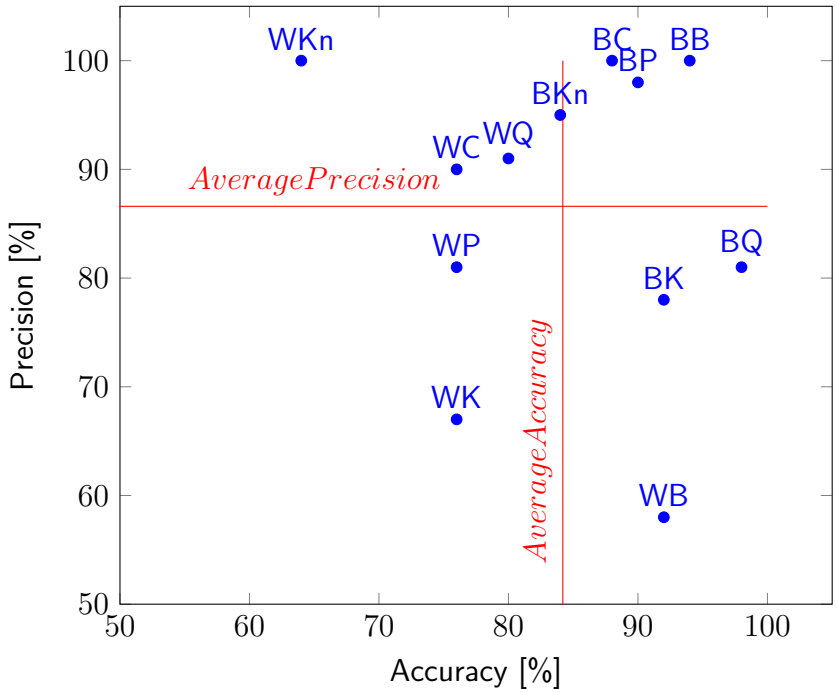


Figure 5.1: Precision vs. Accuracy Plot of 1-Step Model

The plot was generated from the data presented on table 3. The scatter plot gives information regarding the accuracy and the precision of the image classes. The average accuracy line and the average precision line are drawn to clearly see the classes on which the model performs below the average.

Table 4: Confusion Matrix of the Binary classifier in the 2-Step Model

The table gives the statistics of the performance of the classifier in predicting each class. The rows of the table are the object class and the columns are predicted object class. Accuracy is the ratio between the number of tested images of the class to the number of images that were rightly classified. Precision is the percentage of the right predictions over the positive predictions.

Label	**Prediction**		**Accuracy**
	White	**Black**	
White	300	0	**100**
Black	0	300	**100**
Precision	100	100	

Table 5: Confusion Matrix of the Piece-wise Classifier in the 2-Step Model

The table gives the statistics of the predictions of the Piece-wise classifier of the 2-Step Model. The rows are the object class and the columns are the predicted class. Accuracy is the percentage of images that were predicted with the right class. Precision is the percentage of the right predictions over the positive predictions.

Label	**Prediction**						**Accuracy**
	B	**C**	**K**	**Kn**	**P**	**Q**	
B	39	5	24	5	6	21	**39**
C	1	67	13	5	11	3	**67**
K	1	3	85	6	1	4	**85**
Kn	0	0	3	30	57	10	**30**
P	0	0	7	7	84	2	**84**
Q	0	4	39	1	7	49	**49**
Precision	95	85	50	56	51	55	

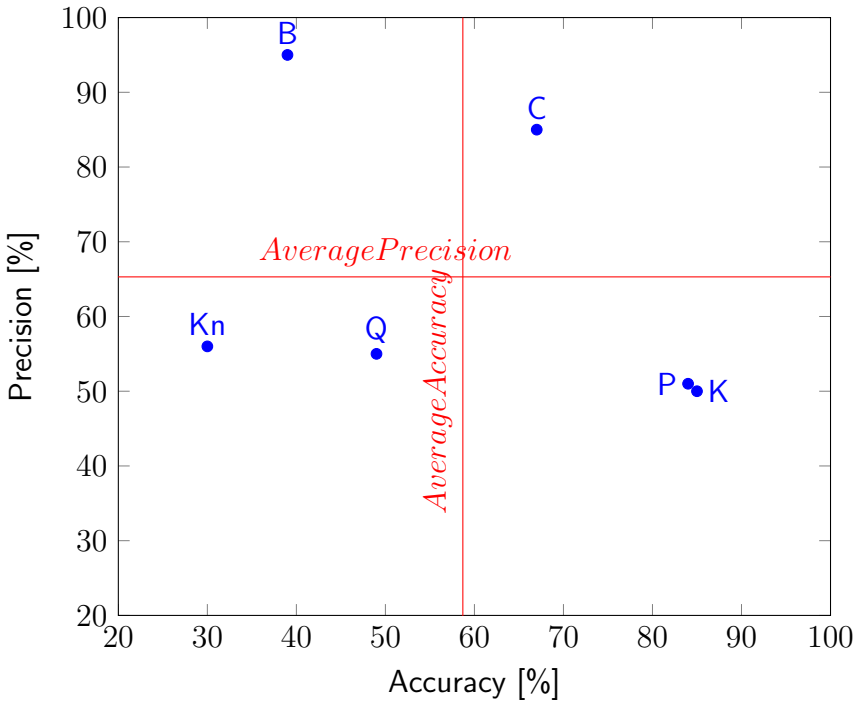


Figure 5.2: Precision vs. Accuracy plot of 2-Step Model

The plot shows the accuracy and the precision of each class of the Piece-wise classifier of the 2-Step Model. The plot was generated from the data produced on table 5. The average accuracy and the precision line are plotted to compare the results for the individual classes to the average.

Table 5 shows that the 58% accuracy achieved by the model is mainly because of the king and pawn, which has accuracies of 85% and 84% respectively. The accuracy of 3 chess classes are less than 50%. The precisions of 4 classes are below average which means there will be many wrong positive classifications.

Considering the class wise accuracy and the overall accuracy, the 2-Step model is performing well.

5.3 Computation Time

The total classification time is important as the motion of the robot and the chess game algorithm depends on the information it receives from the visual feed. For instance, to be able to classify on live video stream without lag, the total time from the capture of video till the output of classification should be less than 500 ms to achieve a prediction with a frame rate of 2 frames per second.

To asses the performance of our system, the time taken at every step in the process: loading of the image to the program, resizing of the image, converting the color space of the image, rearranging the color channels to feed into the network, loading of the image to the network, and classifying the image was measured by running a test on 60 images using a python script. The results of the individual models are given in Table 6 to Table 8.

Table 6: Computation time of the functions in the 1-Step Model

The average time and the standard deviation of the computation time of the processes in classifying an image using the 1-Step model is given in the table. The data was collected from recording the running time for 60 images.

Process	**1-Step Model**	
	Mean time [s]	**Std. dev. [s]**
Loading the image	0.0284	0.0193
Resizing the image	0.0291	0.0089
Converting the image color	0.0018	0.0048
Preprocessing the image	0.0046	0.0012
Inputting the image	0.0005	9.1147
Classifying the image	3.7573	0.0227
Total	3.8218	0.0486

Table 7: Computation time of the functions in the Binary classifier of the 2-Step Model

Using the data from 60 images the average computation time and the standard deviation were calculated for functions in the Binary classifier of the 2-Step Model.

Process	**2-Step Model Binary classifier**	
	Mean time [s]	**Std. dev. [s]**
Loading the image	0.0165	0.0037
Resizing the image	0.0130	0.0022
Converting the image color	0	0
Preprocessing the image	0.0003	0
Inputting the image	0.0003	0
Classifying the image	0.0421	0.0040
Total	0.0722	0.0080

Table 8: Computation time of the functions in the Piece-wise classifier of the 2-Step Model

The average computation time and the standard deviation was calculated for the functions in the Piece-wise classifier of the 2-Step Model using 60 images for data. The results are given in the table.

Process	**2-Step Model Piece-wise Classifier**	
	Mean time [s]	**Std. dev. [s]**
Loading the image	0.0798	0.0226
Resizing the image	0.0964	0.0133
Converting the image color	0.0010	0.0003
Preprocessing the image	0.0003	0
Inputting the image	0.0016	0.0003
Classifying the image	1.3273	0.0165
Total	1.5065	0.0401

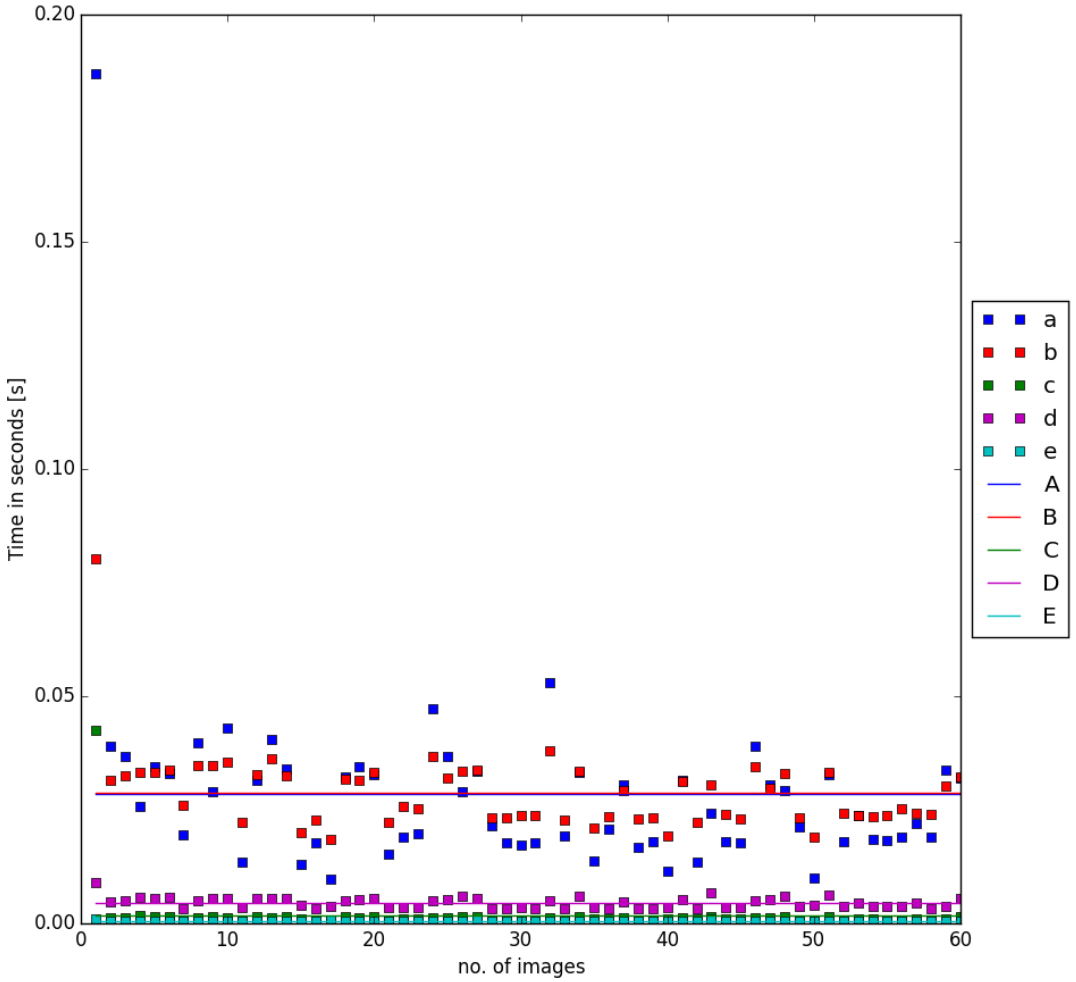


Figure 5.3: Time vs. number of images plot for the pre-processing in the 1-Step Model

The computation time in the 1-Step Model for each image is given in the plot. The data are of the time for (a) loading the image to the program, (b) resizing the image, (c) converting the color space of the image, (d) transposing and swapping the color channels of the image, and (e) for inputting the image to the network. The lines are the average time of the above mentioned processes. The average time for the above mentioned processes are given by (A) time for loading the image is 28.4 ms, (B) time for resizing the image is 29.1 ms, (C) time for converting the color space of the image is 1.8 ms, (D) time for transposing and swapping the color space of the image is 4.6 ms, and (E) time for inputting the image to the network is 0.5 ms.

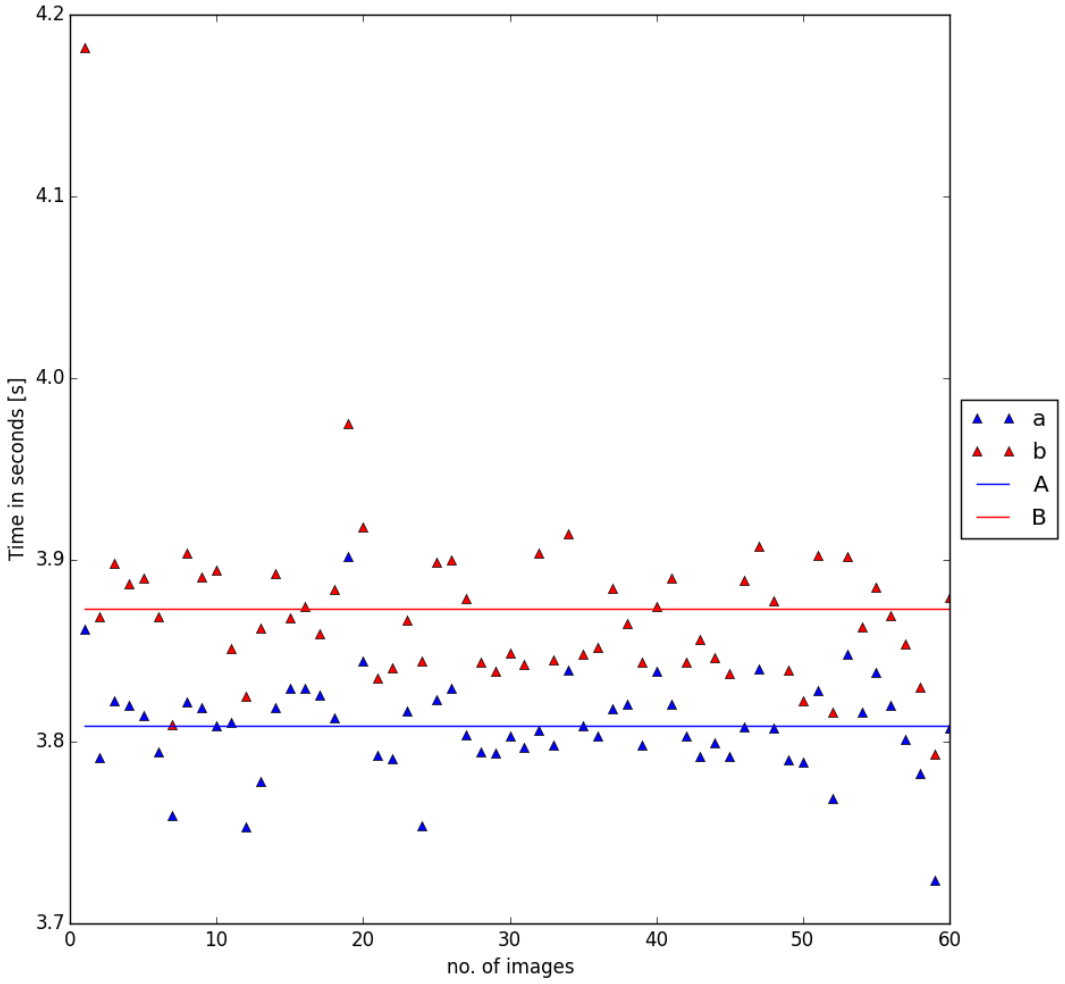


Figure 5.4: Time vs. number of images plot for classification in the 1-Step Model

The computation time in the 1-Step Model for each image is given in the plot. The data are of the time for (a) classifying the image, and (b) the total time of all the processes. (A) The average time for classifying is 3.7573 s, and (B) the average total time is 3.8218 s.

Figure 5.3 and 5.4 are the time vs. number of images plot of the 1-Step model. From the data obtained, the average time required for the individual processes before classification are below 30 ms. The total time of all these processes is below 100 ms. From the table 6 it can be seen that the total classification time is 3.82 s which was due to the classification step, when the image is processing through the neural network, which consumes on average a time of 3.76 s.

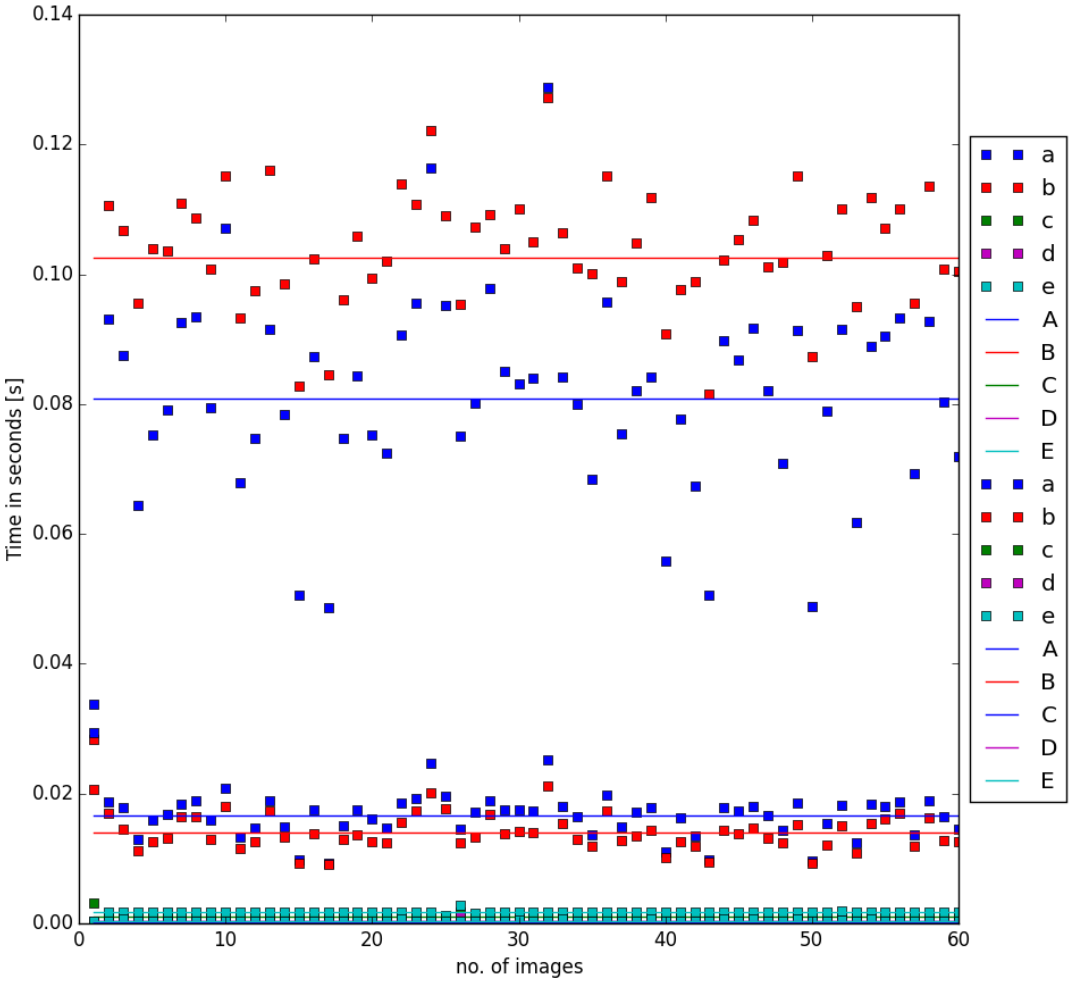


Figure 5.5: Time vs. number of images plot for the pre-processing of the Binary classifier in the 2-Step Model

The computation time of the Binary classifier in the 2-Step Model for each image is given in the plot. The data are of the time for (a) loading the image to the program, (b) resizing the image, (c) converting the color space of the image, (d) transposing and swapping the color channels of the image, and (e) for inputting the image to the network. The lines are the average time of the above mentioned processes. The average time (A) for loading the image is 16.5 ms, (B) for resizing the image is 13.0 ms, (C) for converting the color space of the image is 0.0 ms, (D) for transposing and swapping the color space of the image is 0.3 ms, and (E) for inputting the image to the network is 0.3 ms.

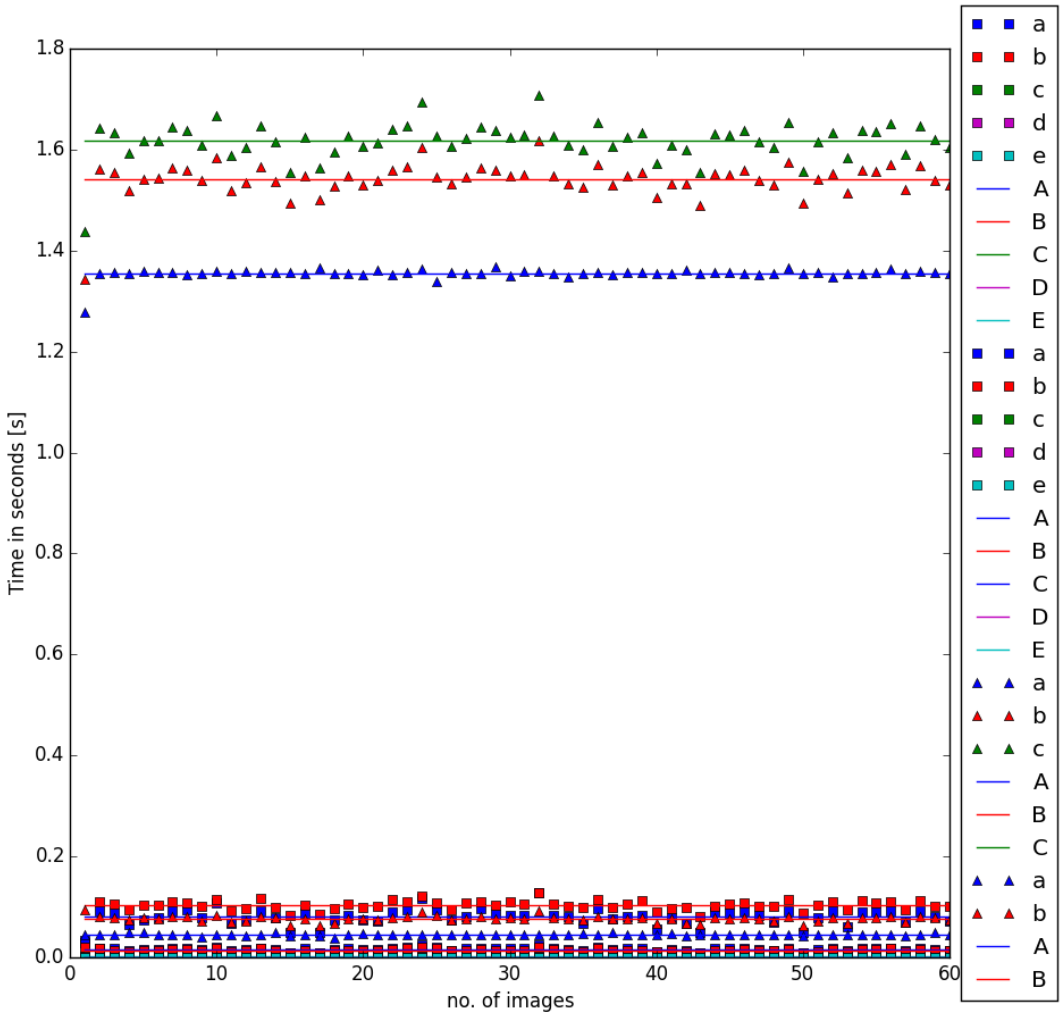


Figure 5.6: Time vs. number of images plot for classification in the Binary classifier in the 2-Step Model

The computation time of the Binary classifier in the 2-Step Model for each image is given in the plot. The data are of the time for (a) classifying the image, and (b) the total time of all the processes. (A) The average time for classifying is 42.1 ms, and (B) the average total time is 72.2 ms.

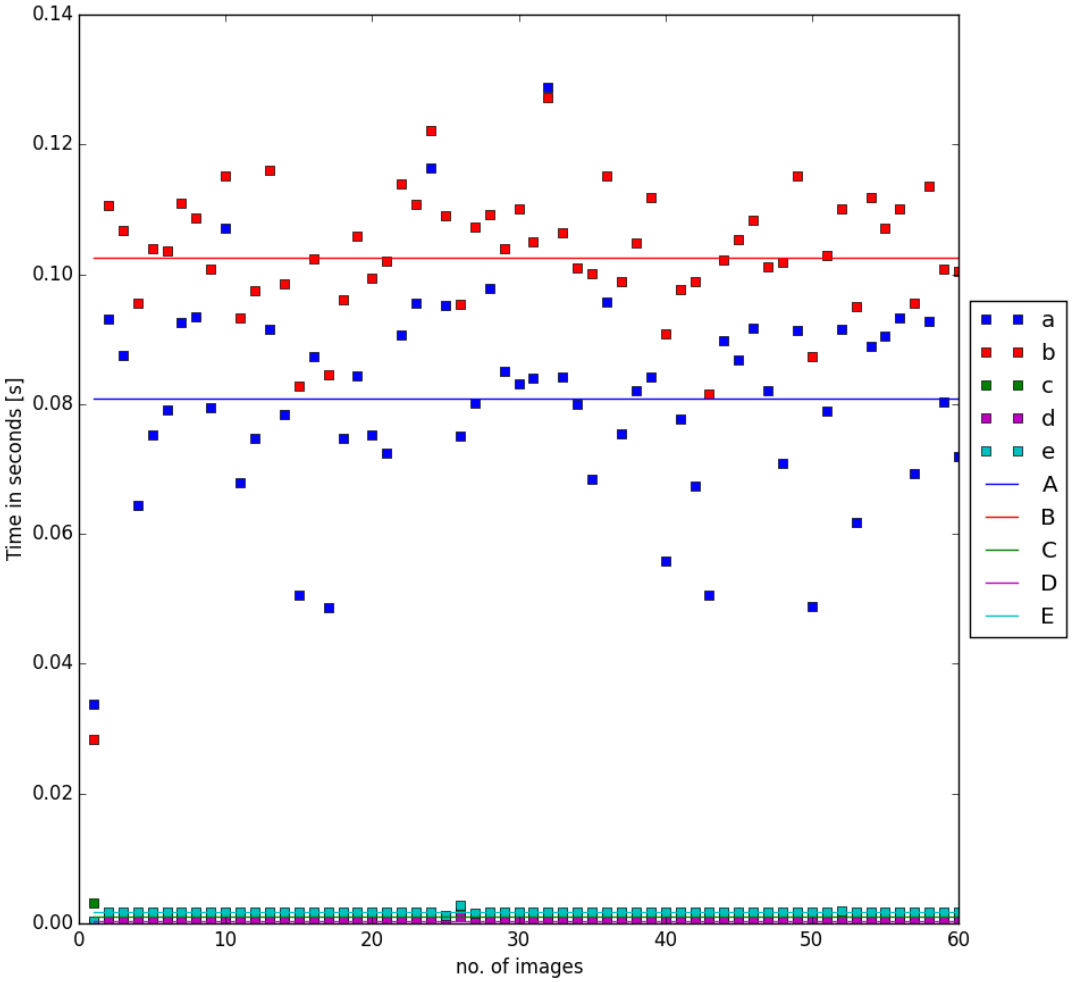


Figure 5.7: Time vs. number of images plot for the pre-processing of the Piece-wise classifier in the 2-Step Model

The computation time of the Piece-wise classifier in the 2-Step Model for each image is given in the plot. The data are of the time for (a) loading the image to the program, (b) resizing the image, (c) converting the color space of the image, (d) transposing and swapping the color channels of the image, and (e) for inputting the image to the network. The lines are the average time of the above mentioned processes. The average time (A) for loading the image is 79.8 ms, (B) for resizing the image is 96.4 ms, (C) for converting the color space of the image is 1.0 ms, (D) for transposing and swapping the color space of the image is 0.3 ms, and (E) for inputting the image to the network is 1.6 ms.

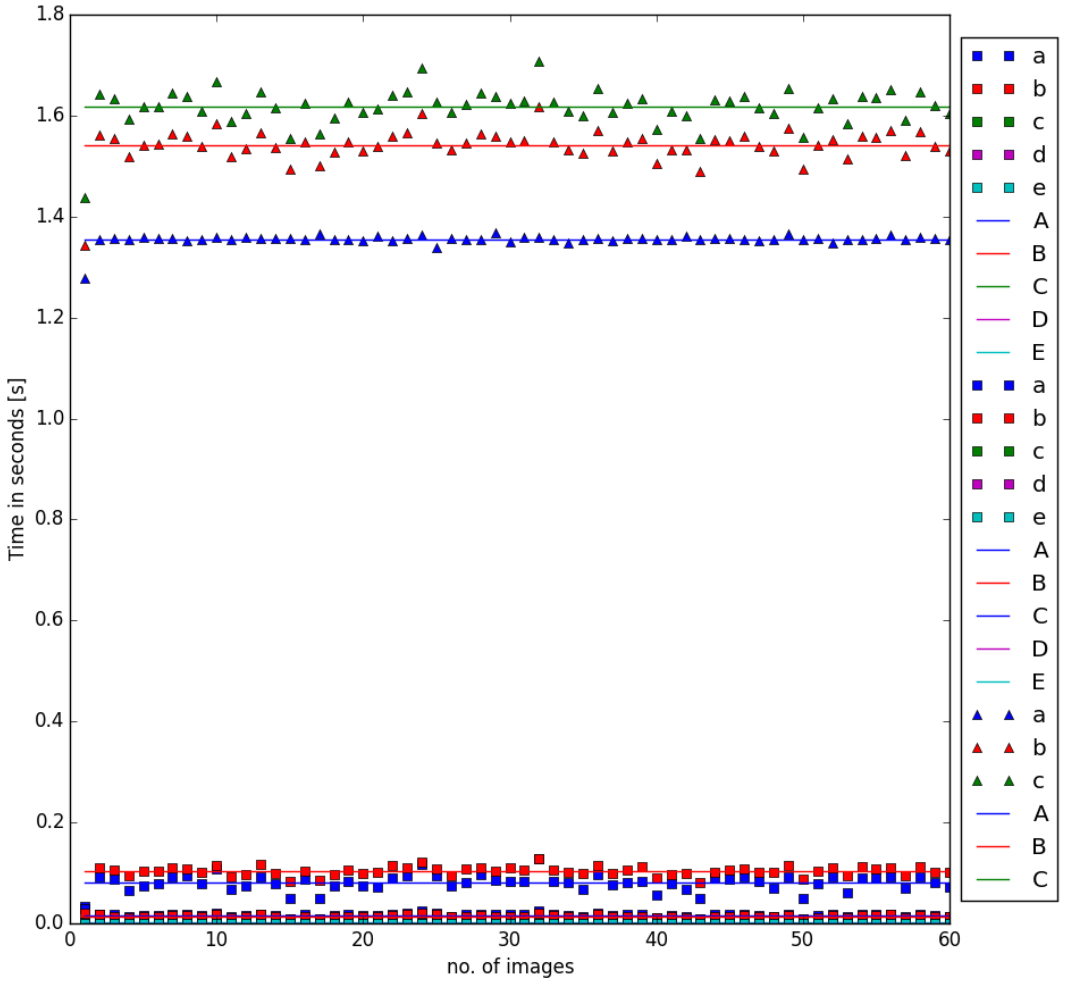


Figure 5.8: Time vs. number of images plot for classification in the 2-Step Model

The computation time of the Piece-wise classifier in the 2-Step Model for each image is given in the plot. The data are of the time for (a) classifying the image, (b) the total time of all the processes, and (c) the total time for classification using the 2-Step model. (A) The average time for classifying is 1.3273 s, (B) the average total time is 1.5065 s., and (C) the average of the total time for classifying using the 2-Step model is 1.5787 s.

Figure 5.5 to figure 5.8 are the time vs. number of images plot of the 2-Step model. The total average time required for predicting the color of the piece is 72 ms. To predict the chess piece based on geometry the total time consumed by the network was 1.33 s and the total classification time was 1.51 s. Since to predict the class of the chess piece we need the classification from both the Binary classifier and the Piece-wise classifier, the total time for classification by the 2-Step model is the summation of the average total time of both the classifiers, which is 1.58 s. The time required for the processes

before classification in the 2-Step Model is 209 ms, which is high compared to the time required for the 1-Step model. The possible reason is that the 2-Step model had 2 neural networks running in parallel which would have caused the response time for the functions to slow down.

When compared the total time taken for classification, the 2-Step model performs better than the 1-Step model by 2 folds.

5.4 Pixelwise Classification

It is clear from the results of the confusion matrix that weights are biased more towards some classes compared to the other. Therefore, to understand which regions of the image are mainly learnt as features for the classes we use a simpler version of the technique used by Zeiler and Fergus in understanding Convolutional neural networks [16]. This information is useful to decide if the networks learnt features are sensible.

They change the fully connected layers of a network, to Convolutional layers by keeping the number of weights a constant. The number of filters of the convolution layer are set to the number of neurons of the changed fully connected layer. For the convolution layer from the first hidden layer, the filter size is set to be the square root of the number of inputs of the fully connected layer divided by the number of neurons of the fully connected layer. In the following convolution layers the filter size is set to be 1. Through these network settings it is ensure the number of learned weights of the fully connected layer is equal to the number of weights of the convolution layer.

Then the edited network is input with an image of size greater than the initially set image size for the neural network to receive an array of classifications as the convolution filter slides over the image. The process can be thought analagous to a sliding window technique in which a whole image is searched for an object from a part of the image at a position and then moving over all the positions step by step.

The technique was deployed on the two multi class classifiers and was fed with images of size $[300 \times 300]$ pixels for pixelwise classification. The color image classifier classifies using a filter of size $[116 \times 116]$ pixels and the grayscale image classifier uses a filter of size $[61 \times 61]$. The output of the classifier is a matrix of size $[23 \times 23]$ with the prediction classes and the confidence of the prediction. This information is used to plot a heatmap to visualize the important regions of a picture during classification.

Below are the heat maps of the individual chess pieces with the input image right above. The dark blue regions were misclassified regions and the remainig regions are classfied right with the blue shades representing lower confidence and the red shades representing the high confidence region. However, it should be noted the region of interest will vary image to image because depending on the object lightning, size, and orientation the important features differ.

From figure 5.9 to 5.14, the first row contains the input images and the second row contains the classification heatmaps of the pixelwise classifier.

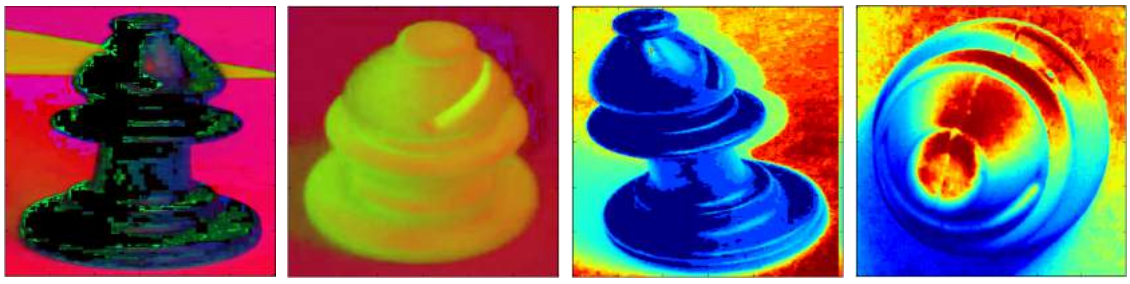
(a), (b) - HSV image

(c), (d) - Intesity image displayed as heatmaps

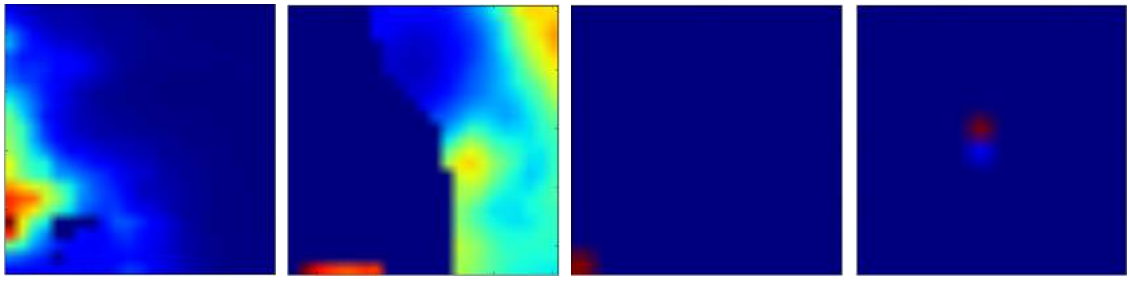
(e), (f) - Classification heatmap of the HSV image (a), (b) respectively

(g), (h) - Classification heatmap of the HSV image (c), (d) respectively

Figure 5.9 shows the pixelwise classification of the class bishop. The heat map of the



(a) BB (b) WB (c) BB (d) WB



(e) BB (f) WB (g) BB (h) WB

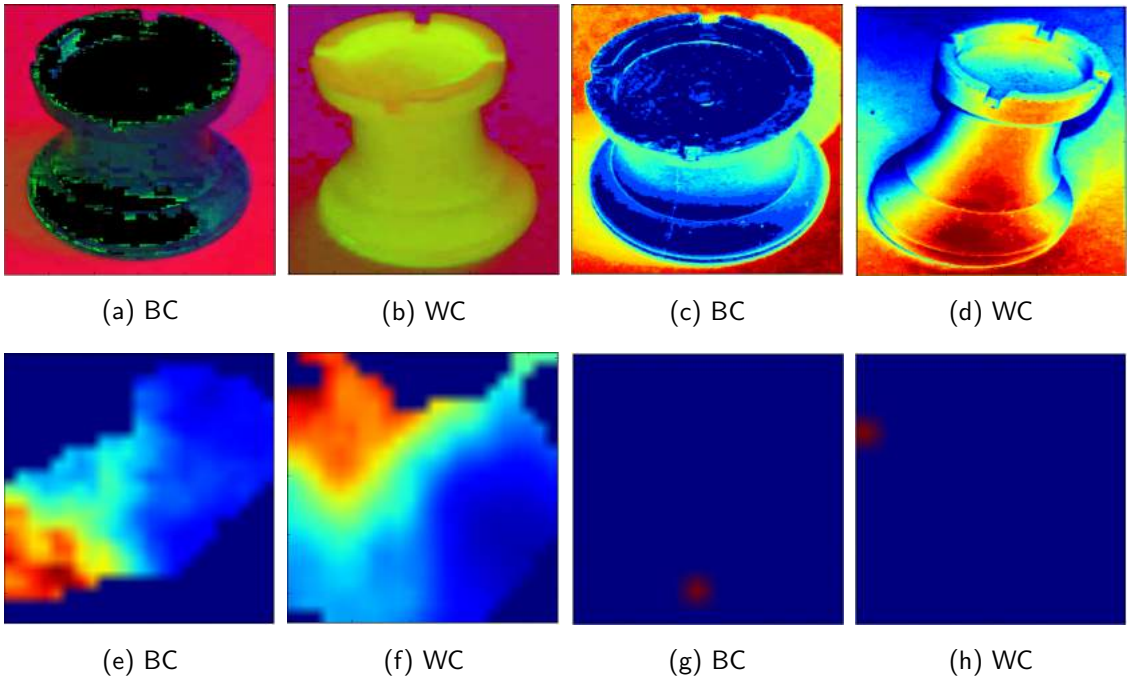
Figure 5.9: Pixelwise classification of class bishop

HSV images show that the region of interest in the bishop lies to the side with a cut through at the top. In addition, it classifies at least half of the image with the right label which shows that the features learned by the network are distributed over the object and not at a point. Thus, even if some part is occluded a right classification is possible.

However, the learned features of the grayscale image are bad. First, it only covers a small region which would result in poor classification performance in situations where

the orientation differs or the point of interest is occluded. Second the region of interest in classification of the black bishop is mostly covering the bottom shape and the background of the image. The bottom shape of all the chess pieces are similar which may lead to wrong classification. It is clear the features extracted for learning in the 2-Step model for bishop is bad.

In figure 5.10 the pixelwise classification of the class castle is shown. The learned



(a) BC (b) WC (c) BC (d) WC

(e) BC (f) WC (g) BC (h) WC

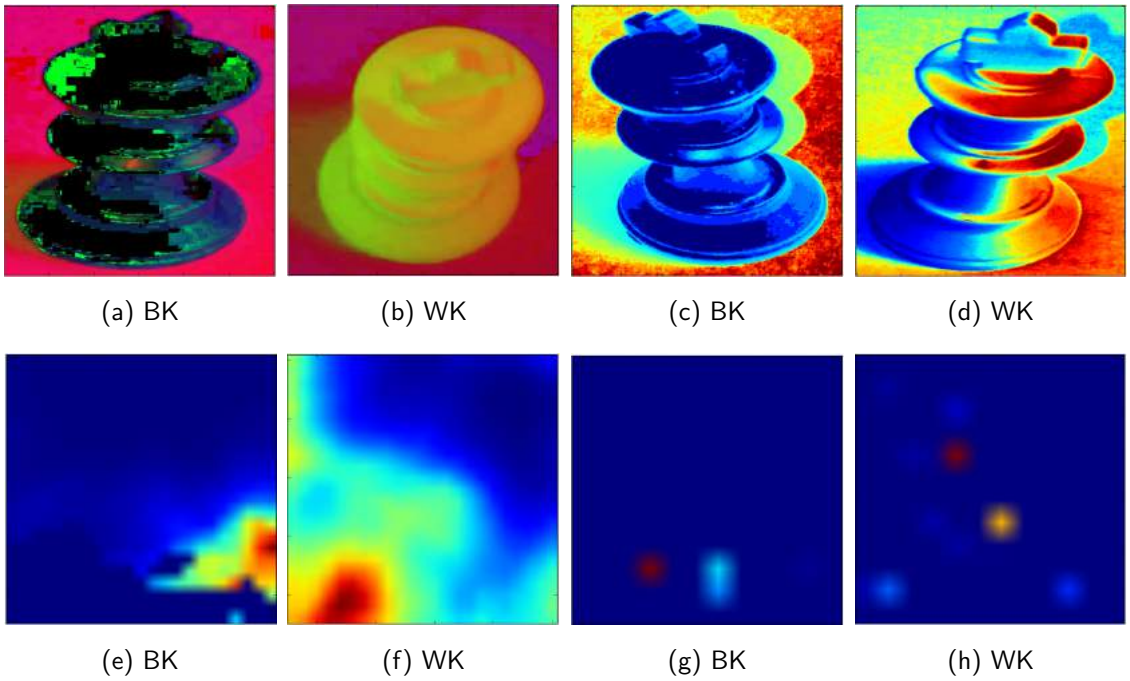
Figure 5.10: Pixelwise classification of class castle

features of the 1-Step model is distributed through the image, which is good. However, for the black castle, the strong features are centered on the lower part of the object which may result in poor classification if there is a change in orientation.

The features of the grayscale image yielded similar results for the castle as for the bishop. The learned features are around a single point.

The figure 5.11 is the pixelwise classification of the class King. The positive prediction distribution is predicted in at least one fourth of the image, especially for the class white king. However, the problem of strong classification on the lower region is present in this class as well.

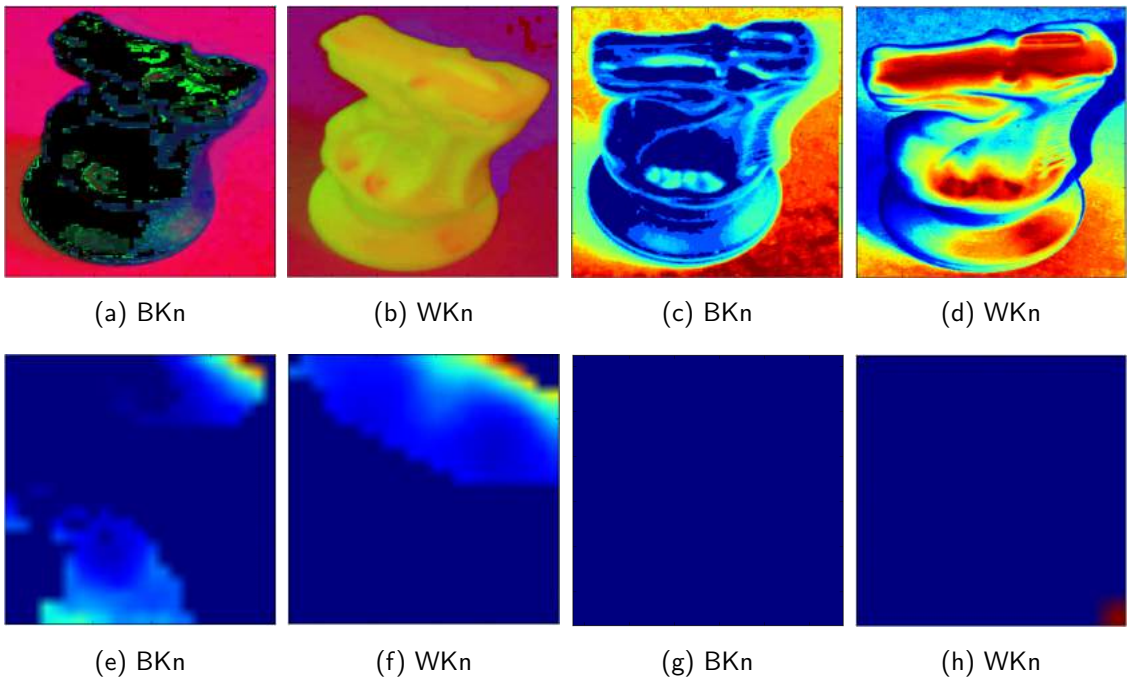
The heatmap shows the prediction of the king is ditributed over the image, which is better comapred to the other classes of the grayscale image. However, the amount of positive prediction for the while picture is still low.



(a) BK (b) WK (c) BK (d) WK

(e) BK (f) WK (g) BK (h) WK

Figure 5.11: Pixelwise classification of class king



(a) BKn (b) WKn (c) BKn (d) WKn

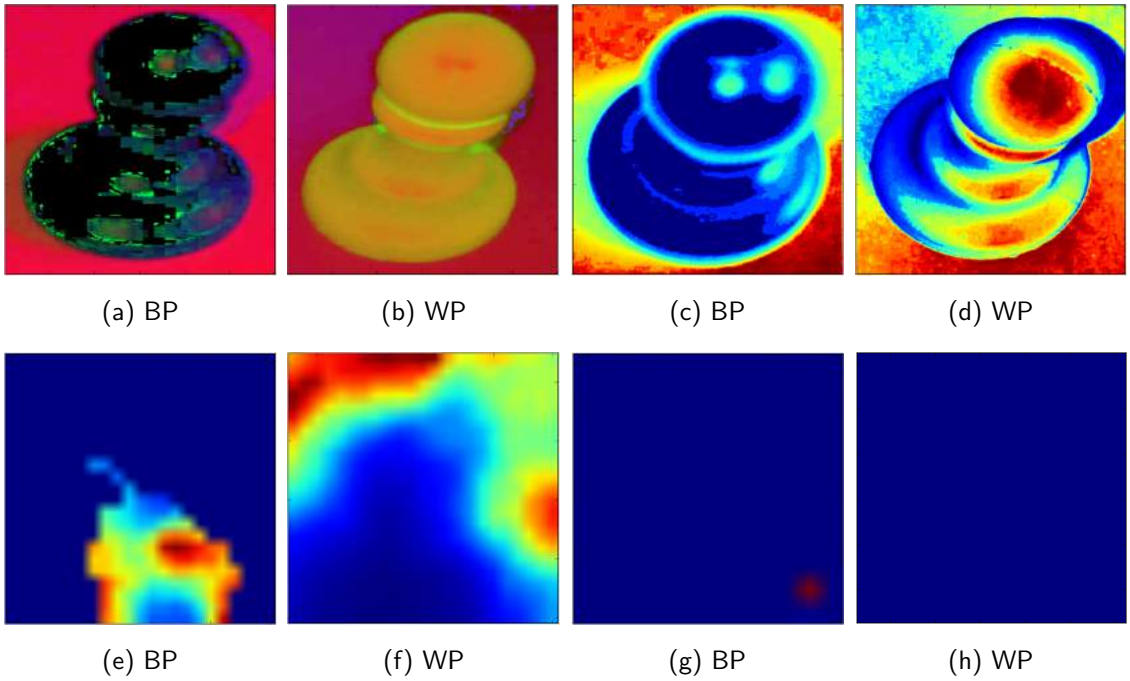
(e) BKn (f) WKn (g) BKn (h) WKn

Figure 5.12: Pixelwise classification of class knight

The pixelwise prediction of class Knight is displayed in figure 5.12. In the 1-Step model, the region from which the features of knight are learned are good because the head region of the knight is very distinct compared to the other pieces.

The 2-Step model have no classification on the black knight, and has only a small region of classification for the white knight.

The figure 5.13 displays the heatmap for the classification of the class pawn. In the



(a) BP (b) WP (c) BP (d) WP

(e) BP (f) WP (g) BP (h) WP

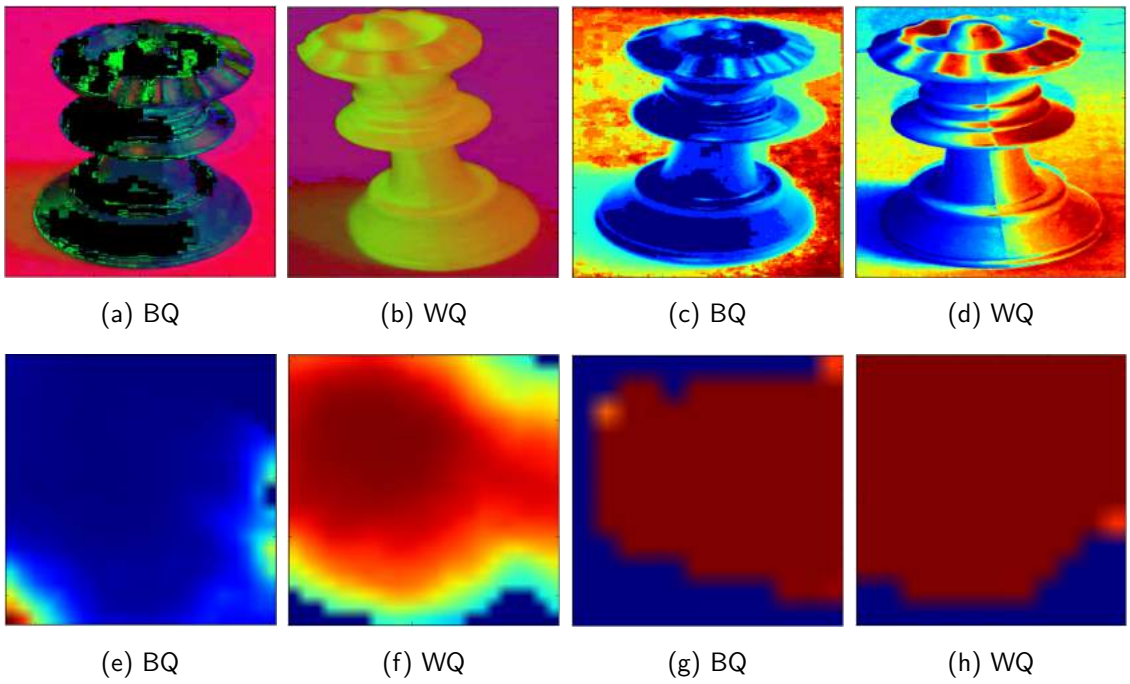
Figure 5.13: Pixelwise classification of class pawn

color images, the white pawn prediction spans over a large area, while the prediction of the black pawn is about one fourth of the image.

The 2-Step model has no positive prediction on the white pawn and only a small region in the black pawn.

The heat map for classifying the Queen is presented in figure 5.14. The black queen has strong features at the left bottom region of the piece while the white queen has it through the most part of the image. This could be a reason for the white queen to have a class wise accuracy of 80% and a precision of 91%. No other class under the white chess pieces have both a high class wise accuracy value and precision.

In the grayscale image, both the white and black queen has a large area of positive classification. A possible reason is that the chosen image had the right features comapred to the other classes as the class queen has only an accuracy of 49% and a precision of



(a) BQ (b) WQ (c) BQ (d) WQ

(e) BQ (f) WQ (g) BQ (h) WQ

Figure 5.14: Pixelwise classification of class queen

55%.

It is clear that the Piece-wise classifier in the 2-Step model needs improvement because the features it has learned are not suitable and too few. The 1-Step model has learned good features, but in the white classes the distribution seems to be too distributed which might be a reason for the poor class wise accuracy in the white pieces.

5.4.1 Scale invariant

From the previous experiments, we can conclude that the 1-Step Model can be chosen over the 2-Step model for the classification task. Thus, an experiment was performed on the 1-Step Model to study how robust it is to change in size of the object on the image. Objects, as shown in figure 5.15, of scale: 70%, 80%, and 90% of the image size from each chess class were shot in daylight condition for the experiment. The set of images were then classified using the 1-Step Model.

The predictions of the model was right only 22% of the time on the scaled set of images. Most of the images were misclassified as Black Knight, Black King, White Queen or White Bishop. A possible explanation for this after visually evaluating the data set was that the diversity of images in every class was very low. There was a monotony in the size and orientation of the training images. Since the neural network is a data driven approach to identify objects, unless the neural network has been trained with similar

images below, the network would not identify the image. So the reason for the bad performance of the model is lack of information for different scales of the object.

(a) Scale: 70% of the image size

(b) Scale: 80% of the image size

(c) Scale: 90% of the image size

Figure 5.15: Chess piece picture of three different size for the experiment

The pictures shows objects of three different scales in the image. This was used to test the robustness of the 1-Step model to variation in scale of the object.

6 Conclusion

To classify a data set of 12 classes, two different models were generated. The first one used a single Convolutional Neural Network for the classification with an input of HSV images, while the other performed the classification using 2 classifiers. One classifier predicts the colour of the chess pieces, and the other predicts the class based on shape. The combination of the prediction is used for the final prediction. The single step classifier uses HSV image, and the 2-Step Model uses HSV image for predicting the colour, and a gray-scale image for predicting the chess pieces.

The average accuracy reached by the colour model was 70%, and the 2-Step model was 58%. Furthermore, from the confusion matrix, we observed that the accuracy, and the precision of the 1-Step model were in the acceptable range. Whereas, for the 2-Step model, the accuracies of the king, and the pawn were above 80% while the accuracies for other chess pieces were low. From the precision values of the chess pieces in the 2-Step model, it was certain that there would be many false positives. Thus, in terms of accuracy, the 1-Step model performed better.

The average computation time of the 1-Step model was 3.8 s while that of the 2-Step model was 1.8 s. Therefore, in terms of computation time, the 2-Step model performed better.

Based on the qualitative analysis performed, it was clear that the features learnt by the 1-Step model were distributed over the object, and that of the 2-Step model were concentrated at a small region. In addition, by performing an analysis to classify the chess pieces with object size, which were within the range of 70%-90% of the image size, it was found that the network can classify objects that are almost centred and of the complete size of the image. After observing the training images, it was certain that the data set was not diverse enough to cover different orientations and sizes.

To conclude, the 1-Step model has performed better in most of the aspects compared to the 2-Step model. However, it still needs a significant reduction in classification time to implement in a real time system.

7 Future Scope

The section suggests how the model can be improved and integrated with other systems. The improvements are suggested for accuracy, modularity, and computational time.

7.1 Improving the Model

The goal of the project was to use the program to develop an autonomous chess playing robot. However, there are still many short comings from the model developed in this project such as the computation time, and classification accuracy.

Computation time is important because a robot which identifies its environment through visual data feed will need to process multiple frames in a second for fluid control. The chess playing robot should identify the chess pieces on the board, for this it needs to identify all the chess pieces on a chess board and their positions. So the classification program at least should predict the classes of the pieces in less than a second. According to Caffe tutorials [9], the implementation of a model in C++ will yield twice the speed of a Python implementation. Further it mentions, that by using a Graphical Processing Unit (GPU), the computation time can be reduced by ten folds. So these two are possible solutions to improve the computation time.

The classification accuracy is 70% in average at the moment. Further, the model is not robust to changes in scale of the object. The first step in improvising the accuracy shall be to prepare a data set which shall contain different scales and orientation of the object such that the program is robust if there are any changes in scale and orientation of the object in the image. In addition, the architecture of the model needs to be further studied and improved, such that the model learns good features.

The above mentioned points shall be the first steps in improving the model.

7.2 Integration to ROS

Robotic Operating System is an Open source framework to widely used in developing robotic software. It has a collection of tools and libraries such as Gazebo- a 3D simulation environment, OpenCV - an image processing library, and many more which simplify the work of the user. So I attempted to import the Caffe library on the C++ code of the ROS environment.

The attempt initially failed to build as it could not find a library by the name cblas which was necessary for Caffe. However, later this problem was solved by explicitly defining the path to the file on the make file of ROS.

Next, a segmentation error halt the program during run-time. To find the problem, the program was loaded to ROS on debug mode and the problem was found to be on an OpenCV function. Furthermore, to find the specific reason of the problem, as OpenCV was already included in the ROS environment and it is working well, a code snippet was added to print the version of the OpenCV used in the current program. This showed that Caffe is built on OpenCV 3.0 version while the ROS environment relies on OpenCV 2.4.8 version. So theoretically, if Caffe is built again with the same version of OpenCV as in ROS then Caffe and ROS can be used in parallel.

Due to the time limit in thesis no further attempts were made, for the moment, to work with Caffe and ROS together. However, if the two frameworks can run in parallel, then ROS framework can be exploited to implement the model on a robot to perform tasks such as object location and manipulation using live camera video feed.

References

[1] No Author. "Open Source Computer Vision (OpenCV) Online Documentation". In: (n.d.). URL: http://docs.opencv.org/master/.

[2] A. Bogle. "Baxter the robot can beat you at Connect Four". In: *Mashable* (Aug. 2015). Image. URL: http://mashable.com.

[3] L. Bottou. "Neural Networks Tricks of the Trade". In: ed. by G. Montavon, G. Orr, and K. Müller. Second Edition. Springer Berlin Heidelberg, 2012. Chap. Stochastic Gradient Descent Tricks, pp. 421–436. URL: http://link.springer.com/.

[4] D. C. Ciresan et al. "Flexible, High Performance Convolutional Neural Networks for Image Classification". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. 2011. URL: http://ijcai-11.iiia.csic.es/.

[5] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *13th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 9. Springer International Publishing Switzerland 2014, 2010. URL: http://jmlr.org/.

[6] K. He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. Feb. 2015. URL: http://arxiv.org/abs/1502.01852.

[7] R. Jain and R. Kasturi. "Machine Vision". In: New York: McGraw-Hill, 1995. Chap. Object Recognition, pp. 459–491. URL: http://www.cse.usf.edu.

[8] Y. Jia et al. *Caffe: Convolutional Architecture for Fast Feature Embedding*. Oct. 2012. URL: http://arxiv.org/abs/1408.5093.

[9] Y. Jia et al. *Caffe Tutorial*. n.d. URL: http://caffe.berkeleyvision.org/tutorial/.

[10] A. Karpathy, F. Li, and J. Johnson. *CS231n: Convolutional Neural Networks for Visual Recognition*. Online Course. 2016. URL: http://cs231n.stanford.edu/.

[11] K. Lillywhite, D. Lee, and B. Tippetts. "Improving Evolution-COnstructed features using speciation for general object detection". In: *2012 IEEE Workshop on the Applications of Computer Vision (WACV)* (2012). DOI: https://www.ieee.org/.

[12] K. F. Man, K. S. Tang, and S. Kwong. "Genetic Algorithms: Concepts and Applications". In: *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS* 43.5 (Oct. 1996), pp. 519–534. URL: http://ieeexplore.ieee.org/.

[13] H. McCracken. "Amazon's iPhone App Uses Image Recognition to "See" Real-World Products You Want to Buy". In: *TIME* (Feb. 2014). Image. URL: http://techland.time.com.

[14] A. Murabayashi. *The Unsettling Future of Facial Recognition*. Blog. Image. Nov. 2015. URL: http://blog.photoshelter.com/author/allen3/.

[15] T. Serre, L. Wolf, and T. Poggio. "Object Recognition with Features Inspired by Visual Cortex". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (2005). URL: cbcl.mit.edu/cbcl/.

[16] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". In: *Computer Vision ECCV 2014*. Ed. by D. Fleet et al. .Part I. Springer International Publishing Switzerland 2014, Sept. 2014. URL: https://www.cs.nyu.edu/.