Universität Passau
Fakultät für Informatik und Mathematik

Classification Of Visualization In Scientific Literature

Masterarbeit zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

Lehrstuhl für Intelligent Systems und Lehrstuhl für Data Science
der Fakultät für Informatik und Mathematik
der Universität Passau

Name:	Arnold Adaboo Azeem
Matrikelnummer:	79176
Fachbereich:	Informatik
Studiengang:	Master Informatik
Erstprüfer:	Prof. Dr. Christin Siefert
Zweitprüfer:	Prof. Dr. Michael Granitzer
Date:	July 17, 2018

Abstract

Chart image classification serves as an initial step towards comprehending, extracting and further analysing raw data visualized using charts. These chart images are regularly embedded in scientific papers and journals to describe research findings. The biggest challenge in chart classification lies with charts having different forms, patterns and ancillary structures like text, legends, and axis. Chart classification consists of extracting chart features which are then used for training a model to identify the chart type.

Previous approaches for dealing with chart classification depended on hand-crafted features which is not robust especially when dealing with a large amount of data with variable content structure. This thesis, therefore, proposes using Convolutional Neural Networks(CovNets), a deep learning-based approach that automates the feature acquisition step. We present a modified version of the AlexNet [3] CovNet architecture for chart image classification.

Two techniques were considered in this thesis, the first is a CovNet model obtained by training on only the dataset we created, this dataset consisted of 17,357 chart images of 4 classes, and the retraining with the created dataset of a model that was pre-trained by Google using 1.2 million images. With results showing an accuracy of 88% and 97%, respectively, the proposed methods in this thesis proved to be very efficient.

Acknowledgement

I would like to thank my supervisor Prof. Dr. Christin Siefert of University of Twente. When Professor Siefert was in University of Passau, anytime I had questions or trouble her door was always open for me, and even after she left to University of Twente, she continued addressing my concerns via email. She steered me in the right direction but allowed this thesis to be my own work.

I would also like to thank Prof. Dr. Michael Granitzer for his support and availability through out the writing of this thesis. I would also like to thank Julian Stier and Sahib Sulka for their patience and advice throughout this the writing of this thesis.

Finally, I must express my very profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you

Arnold Adaboo Azeem, Passau 13/08/2018

Contents

List of Figures

List of Tables

1 Introduction

“The human brain processes images 60,000 times faster than text, and 90 percent of information transmitted to the brain is visual” [8]. As human beings, we have been communicating amongst each other for a much longer period of time than we have been using written words. Our brain is, therefore, inclined more to processing visual data faster than text.

Due to this fact, most scientists exploit the brain’s ability to capture visualized content faster by representing their research findings using chart images and then embedding these images in their publications. Since data is ever growing and more visualizations are being used, there is a growing need for these graphs to be analyzed faster and automatically. Identifying or classifying these charts serves as a foundation for further analysis of the content of these charts.

The aim of this thesis is to develop a reliable dataset consisting of chart images which will enable us to train a model capable of classifying chart images accurately as an initial step towards decoding the content by a machine. The structure of this thesis comprises three main sections: chart creation, image processing, and machine learning techniques.

The initial section in this chapter gives an overview of chart image classification and the complications and difficulties faced in chart image classification. In the other sections of this chapter the research motivation, objectives, contributions and an outline of the proposed framework in this thesis is presented.

1.1 Overview

A basic description of a chart is a graphical representation of data, where the data is represented by some type of symbols. These symbols could be rectangles in the case of histograms or bar charts, slices that make a circle in the case of a pie chart, and dots, stars, diamonds and an asterisk in the case of a scatter plot. Charts are often used to visualize complex data for easy understanding, interpretation, and to find patterns and trends in data. Charts can usually be read more quickly than the raw data visualized. They are used in almost every field especially in science, marketing, and maths. Charts are mostly created with the help of a computer but can also be created by hand. There are a variety of charts and therefore, to present your data in the right way, the right type of chart must be employed. For example bar charts are best used when comparing data, line charts work better when we want to find trends in data and scatter plots are used when we want to find relationships in data. Figure 1.1, which was inspired by a diagram from an article written by Few S.(2018) [9], shows which charts should be chosen depending on what task we would like to describe. For example, if we want to show a relationship between two variables in some data, a scatter plot is the best option, as seen below.

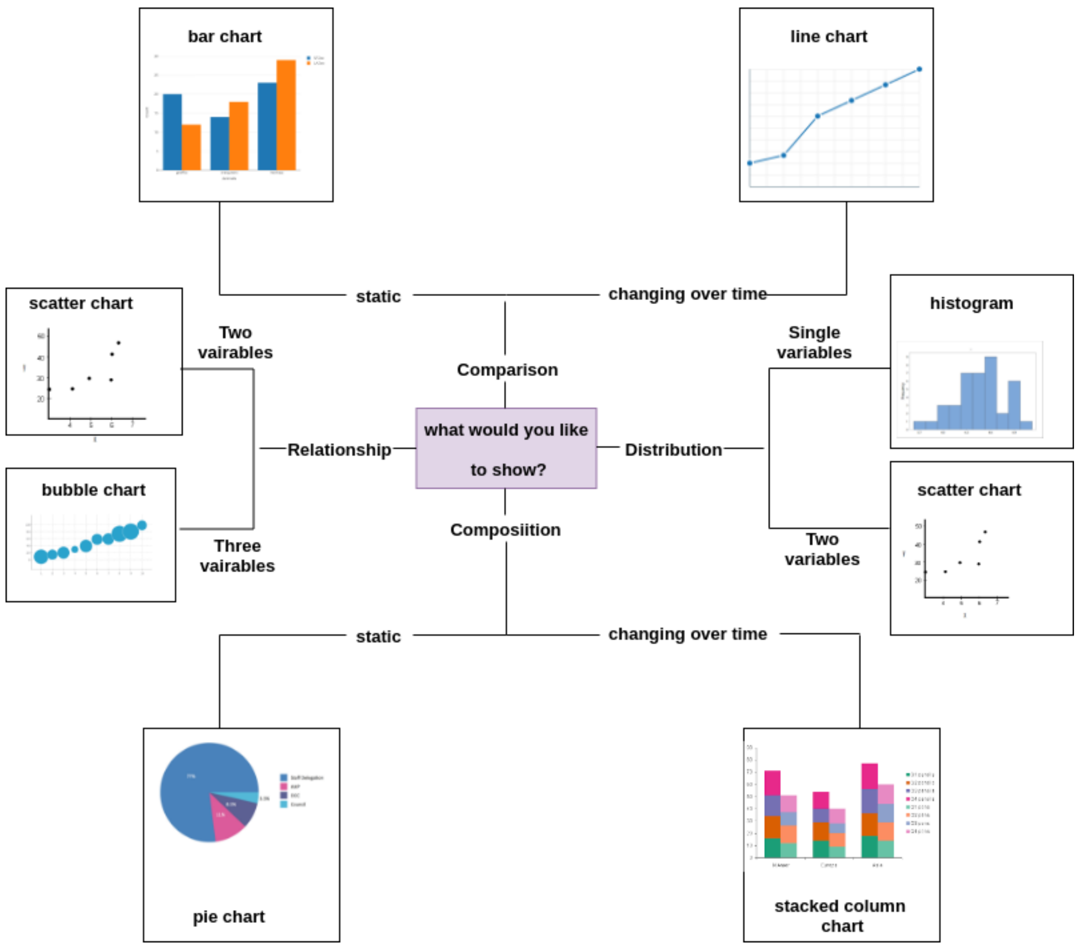
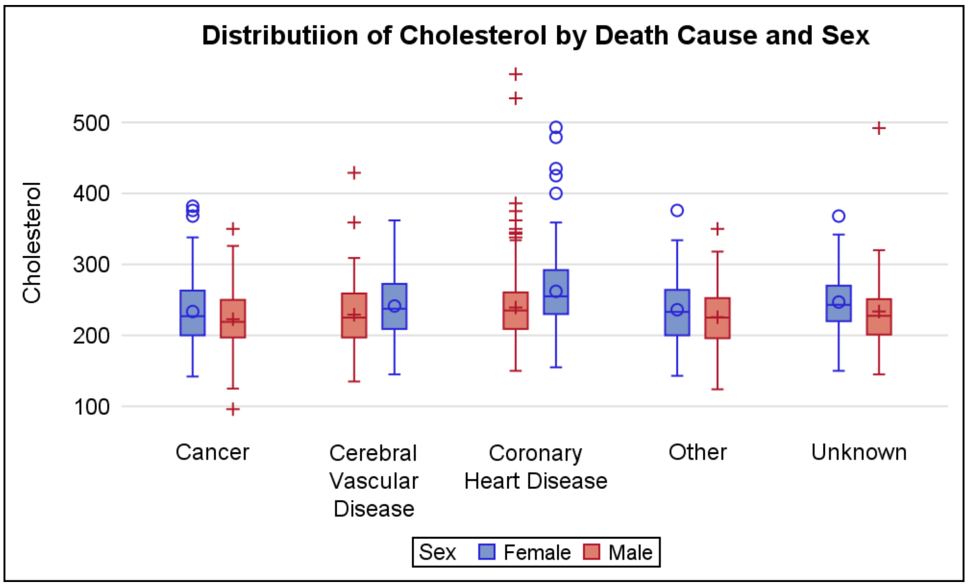


Figure 1.1: How to Select the approapriate chart for visualizing your data

Charts usually appear in different forms. Nonetheless, there are some features that are mostly present in every chart. Charts are mostly graphical with a little bit of text used to annotate the data. One important use of text, visible in most charts, is to give the chart a title. The title describes what the graphic represents and is usually situated above the main graphical part. Text is used less because the brain processes visual information sixty thousand faster [8] as mentioned earlier. Most graphs have one line at the side and another usually at the bottom. These lines if used are called axes of the graph. The one at the bottom is called the x-axis, and the other line is the y-axis. Each graph usually indicates numerical or periodic sequence shown by periodic graduations. There are also texts usually outside or beside the axis describing the measurements. Some charts include what we call a legend, which is usually present in situations where there are multiple variables to be described. Figure 1.2 shows a typical chart that has the features mentioned above. The legend showing the two variables female and male represented in the chart, the label on the left side annotating the cholesterol levels with the different measurements of the cholesterol shown on the y-axis, and to the bottom categorical measurements shown on the x-axis. Finally, the title that is the text describing the whole chart is mostly shown at the top of the visual graph.

Figure 1.2: A box plot with a legend, x-axis, and y-axis from [1]



Despite most features being present in most charts, there are some charts that are totally different from the rest. For example, a pie chart does not have an x-axis or y-axis. Also, the designers of a particular chart have a choice of visual encoding and styling. This results in a wide variety of charts, even in charts of the same type. To be able to identify the different types of charts accurately will mean considering all the wide range of the different visual encodings and styles of these charts.

1.2 Motivation

The examples described in the introductory section have shown that chart images are very relevant for visualizing data and there is a need to classify chart images as a basis for further analysis of the charts. Charts contain a wealth of information that other researchers will like to extract and interpret because:

- The information when extracted can be used by another researcher to validate the authenticity of the experiment done and the results obtained.
- The information of one type of chart can be decoded and presented in another form to a different audience.
- Comparing two charts is more effective when we have access to the raw data.

Even though the identification, analysis, and interpretation of chart data can easily be done by a human being. Machines, on the other hand, find it difficult to decode them due to the variations in their appearance and their variable structure [6]. There is a growing need in many fields for machines to be able to perform tasks that human beings are easily doing with regards to chart analysis [10].

In spite of this growing interest, there have been little groundbreaking results achieved due to the complexity of the problem [6]. Previous research on chart classification exploited the structural content of charts for classification. For example, Manollis Savva, et al. [11] proposed a model to classify charts using extracted low-level features (small rectangular image regions)and textual features. After extracting the features, a Support Vector Machines (SVMs) classifier is used for the classification step. This method was limited since most charts contain the same types of low-level features like axes, grid lines, and legends. In V. Shiv Naga Prasad's work [12] classification was based on using features based on the shape and spatial relationships of their primitives. This work was limited due to the inconstancy in which data in most charts can be depicted. In the paper of Alex Krizhevsky et al. [3] a deep CovNet which combines both feature extraction and classification was used and this technique achieved good results. However, the dataset size of 3377 chart images of 11 classes is a bit limited due to the complexity of the problem.

These problems and others addressed in Chapter 2 was the motivation for the search of a more efficient solution. This report is therefore an attempt to look for a more efficient solution.

1.3 Objective

The aim of this thesis is to develop algorithms using image processing and machine learning methods for the purpose of chart image classification. Different approaches will be considered in order to choose the most favorable one.

The goal of this thesis is to try to identify or recognise chart images with the highest accuracy. To achieve this goal this thesis seeks to answer the following question:

> How well can we classify the four different types of plots (Line-Charts, Bar-Charts, Scatter-plots, and Box-plots) in scientific literature?

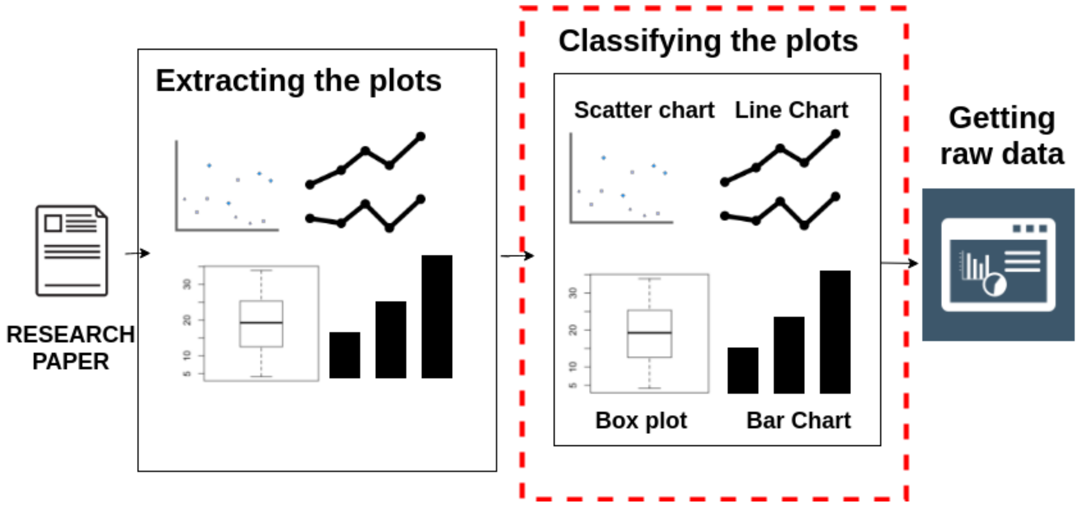
To answer this question, the following objectives are set:

- To obtain raw data gathered from real world occurrences.
- To create a dataset of chart images with the collected data, using different plotting programs (Python, Matlab, R, and Java) and different libraries supported by these plotting programs.
- To train and evaluate a machine learning model that can recognise unseen charts with high accuracy.

1.4 Research Significance and Contribution

Figure 1.3 shows the significance of this work and what it will lead to. This thesis focuses mainly on four chart images. These plots are scatter plots, bar charts, line charts and box plots. The first three were selected because they are the most popular image charts. Box plots were also chosen because it shows skewness and unusual observations in a dataset. This information is especially relevant when dealing with large data, which is mostly the case in machine learning. The first part of Figure 1.3 involves obtaining the four different types of plots mentioned earlier, after which we then label our plots and train a neural network model to classify with high accuracy any of the four plots if shown to the model. Then finally, the raw data can be extracted from the detected plot using other processes and techniques. The focus of this thesis, however, is shown in the figure with red dotted lines, which entail gathering a dataset of chart images, labeling them and training a model that can classify the plots.

Figure 1.3: Overview of the process to obtain raw data from scientific papers



2 Background and Related Work

2.1 Overview

In this chapter, the literature on the classification of chart images is presented and explored. Old techniques that were previously used and how they evolved are discussed. Section 3.2 introduces the two main categories of image classification, while in 3.3 and 3.4 the two categories are discussed in detail, also, CovNets which are inspired by the cat's visual cortex and have performed really well in the field of image processing is talked about in detail. Lastly, Section 3.5 presents the important approaches and techniques employed by other researchers for chart image classification.

2.2 Background

Classification involving charts images has been a topic of huge interest in recent times. This is due to the fact that, most scientific publications if not all, are embedded with these charts as a way of conveying complex research findings in more visualizable and easy to understand format. As a result of this popularity, chart images have gained, different techniques have over time been used for classifying these chart images, these methods have over the past few years evolved. The techniques in recent years have been inspired by the field of image processing which employs raster to vector conversion and layout analysis of the images amongst others. These techniques can be put into two main categories, Model-Based Approaches and Machine Learning Based Approaches [10].

In the next section, the approach used in this thesis, and the works that inspired us to choose these techniques are explained in details.

2.3 Model Based Approaches

The following is written with inspiration from Boyle et al. work [13]. Numerous chart images even of the same type, are different in sense of their context structure. This is as a result of the variability of the positions of the context structure of a chart. There is exists no real standard for all charts to follow, content like legends, axes, grids etc., have no fixed position and sometimes are not even present in some charts. As a result of this non-standardization of charts, classifying becomes a difficult task. Initially, chart classification exploited the structure of charts, this method is generally called model-based. The model-based approach is divided into two steps. Firstly, a number of predefined object classes are created with abstract models and then an image is matched with the created model. For example, a model for a 2D/3D pie chart consists of line segments (radii and circular/elliptical arcs), and these line segments are used to create the model, however, just using these line segments won't be enough, so some constraints are introduced into the model, example of constraints of a pie chart include; the center of the pie chart is where all radii meet; all 2D pie charts have radii of equal length; arcs mainly form parts of the same 2D pie chart circle, or if its a 3D then the arcs form part of the ellipse. After this, a goodness-of-fit is introduced to help measure the discrepancy between the image and the conditions for the model. For example, a goodness-of-fit of a pie chart is as follows: the difference in length of radii, the difference in how the arcs are curved, the distance between a center of a circle and that of a radii. When a chart is presented for classification, the edges are detected, thinned, linked, vectorized, extracted and compared to the various edge models created and the best match is selected. After this, the good-to-fit criteria is used to measure the difference between the edges of the model and the image. Finally, a voting is performed base on the value of the good-to-fit results. The model-based approach has some drawbacks though. The main drawback is that human intervention is needed to develop the various models. This could be a cumbersome and time-consuming procedure, due to the limitation of this approach a machine learning approach was introduced to handle some of the limitations of this approach.

2.4 Machine Learning Based Approaches

This method was developed in recent times, and most works in the area of image analysis and processing employ this approach. This method involves using handcrafted features extracted from the charts for the classification task. In

Zhou and Tan's [14] work, features like a legend, x-y-axis labels, chart title, and values of the bar of a bar-chart were extracted for purposes of classifying a bar-chart. In Shao and Futrelle's [15] work, graphical elements of the charts like colors, tick marks on an axis and data point markers are the handcrafted feature used for classification. Another instance where handcrafted features were used was in Inokuchi et al. work [16]. In this work, regular appearing substructures of a chart are extracted and used as features for classification. After the feature acquisition step, the obtained features are then represented in vector form for the classification step. The features are stored in a matrix vector-like structure. Algorithm 1 shows an example of the structure in which the features are stored. This structure has multiple columns containing the features and one column indicating the target or label of the chart. This vector-like structure is then fed as input into a machine learning technique for the model training step. In Karthikeyani and Nagarajan's paper [7] after the features were obtained SVM Classification [1], MLP Classification [2], and KNN Classification[3] were used to create a function that maps the set of features to a predefined label. This technique achieves good results but, the drawback is that there is an over-reliance on handcrafted features. This drawback makes it difficult when a large amount of data consisting of a variable context structure is involved, and this situation is evident in most cases. Recently, however CovNets [4], a machine learning technique which removes the drawback of using handcrafted features by learning important features anywhere in the image automatically. CovNets performance on image classification keeps on improving and in recent times, results are being compared to human performance levels. CovNets were used in this work, in the next section they will be explained in details.

2.5 Convolutional Neural Network

2.5.1 Artificial Neural Networks

This section is inspired by a book called the nature code [17]. Artificial neural network a very important tool in machine learning was developed with inspiration from the inner workings of the human brain to solve certain kinds of problems. There are certain kinds of problems that are easy for a computer to solve but rather difficult for human beings. There are also on the other hand

[1]http://www.statsoft.com/Textbook/Support-Vector-Machines
[2]http://www.iro.umontreal.ca/ bengioy/ift6266/H12/html/mlp_en.html
[3]https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/
[4]https://deeplearning4j.org/convolutionalnetwork

problems that are easier for human beings to solve but difficult for computers. Show me a picture of a cow today and tomorrow I can pick a different picture of a cow from a group of pictures of animals. But when its time for a machine to perform the same task it becomes more complicated. There has been a lot of research into finding solutions for tasks such as the one mentioned above.

Artificial neural networks have been found to perform really well in tasks difficult for computers but easier for human beings. One property that makes neural networks standout is their ability to learn. A neural network is a system that can change its inner workings based on the information flowing through it. This is possible because it can adjust what we call its weights. Figure 2.1 is a representation of a typical neural network. In the diagram, each line indicates a path of flow of information. These lines are represented by numbers(weights). In an event where there is a bad output, these weights are adjusted by the system in other to improve subsequent results. This ability to learn and make an adjustment in other to improve its results is what makes neural networks special in machine learning. Neural networks perform really well in fields like pattern recognition, anomaly detection, signal processing, and, image classification.

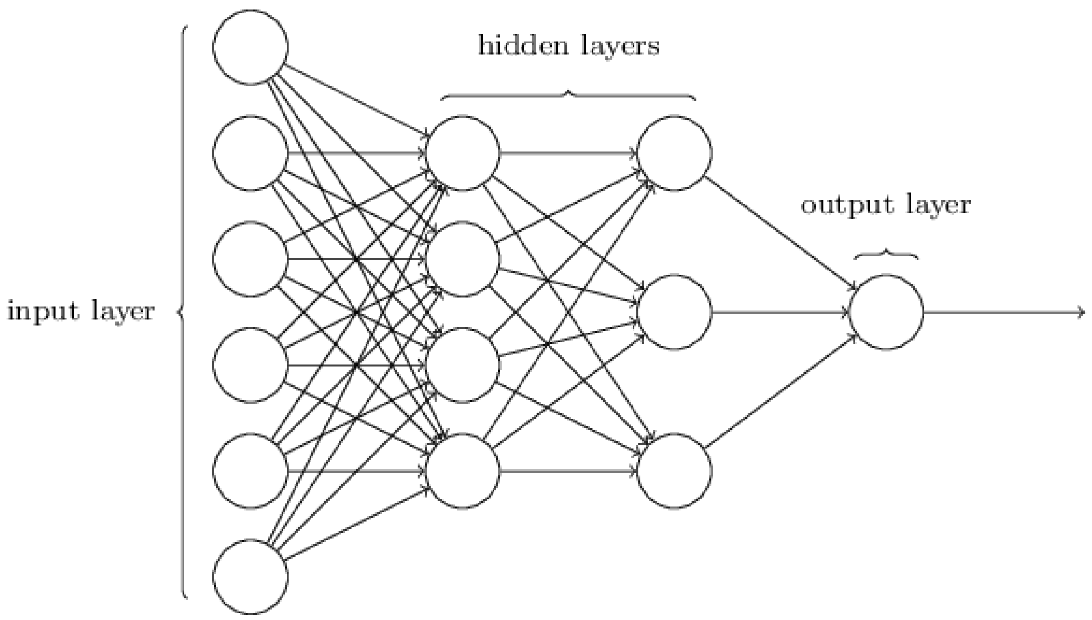


Figure 2.1: How a neural network looks like

2.5.2 Convolutional Neural Network

CovNets are a category of neural networks that have archived state of the art results in the area of image classification and analysis. Neural networks have fallen out of use due to the fact that information flows from input to output in one direction. This results in a very complex structure when dealing with image and text due to the reason that all neurons are connected. The network becomes even more complex when dealing with a huge dataset. This resulted in the development of other variants of neural networks. CovNets a variant of neural networks was inspired by a cat's visual cortex. In a cat's visual cortex there are receptive fields made up of sub-regions which are layered over each other. When a cat looks at an object the layers act as filters which process the viewed object and then pass the signals to subsequent layers. The arrangement of the neurons in the cat's cortex proves to be a simpler way to carry signals as opposed to the initial way done by artificial neural networks. A CovNet has neurons just like the visual cortex and these are organized into layers. Each layer tries to identify a part of the input image. For example, one layer identifies the edges, another identifies the curves and so on. This is done by employing the spatial relation that exist in the pixel of an image. All the layers are connected to each other. There are four main layers in the CovNet:

- Convolution
- Non-Linearity (ReLU)
- Pooling or Sub Sampling
- Classification (Fully Connected Layer)

Before the four main layers are explained, how images are perceived by a computer has to be described to understand the layers. Unlike human beings computers see an image as a matrix of numbers. Images can be gray-scale or color. Gray images are 2D matrix whilst color images have a 2D matrix of 3 channels ie. RGB (red, green, blue). Figure 2.2 shows how a computer perceives part of the image of a cat.

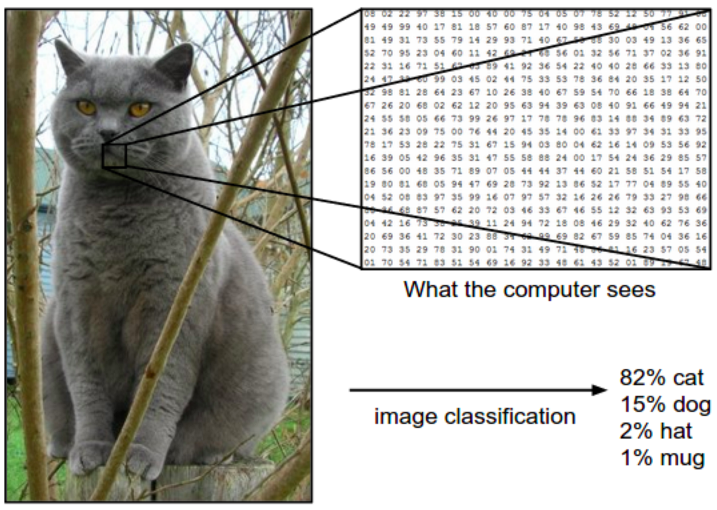


Figure 2.2: How a computer perceives an image-source [2]

2.5.3 Convolutional Layer

The Convolutional layer will be explained using the following example, let consider the image as a magical cake of height and width say 48x48, and there is a cake cutter of size say 5x5. The cutter is used to cut the cake from the top left. The cutter is referred to as a filter in machine learning. The cutter, in this case, is also an array of numbers called weights and for the maths to work the cutter must have the same depth as our cake (5x5x3). Lets first consider the cutter being in the first position ie. the top left of the cake. The values in the cutter are multiplied by the values of the cake (element-wise multiplications). These multiplications are all summed up and result in a single number. This single number is only for the first part and this process is repeated throughout the whole cake (Next step would be moving the cutter to the right by 1 unit, then to the right again by 1, and so on) keep in mind its a magical cake so you can cut a part more than once. After using the cutter on the whole of the magical cake we result in a new cake of size 28x28x1, the results is called a feature map. The filters are low-level feature identifiers. Which identified features as it was passing through the whole cake (straight edges, simple colors, and curves).

2.5.4 Activation Functions

The activation functions are an important part of a CovNet, they are mostly added to the output end of a Convolutional layer. They usually map the output values between 0 and 1 or -1 and 1. The activation functions can be divided into two categories; linear and non-linear functions. The non-linear functions are the most used because most of the real-world data are non-linear. There 3 main types of this function that are mostly used. They are the sigmoid, tanh and ReLu. The sigmoid map the output value between 0 and 1, the tanh maps the output between -1 and 1 and finally the ReLu the most used activation function. The Rectified Linear Units (ReLU) is found in all layers during the Convolution phase, this layer implements element-wise nonlinearity to our Convolutional layer, this just means it helps to handle situations where the relation between the input values and the CovNet output is non-linear. The ReLU has a function $f(x) = max(0, x)$ which means if you give it a value x, it will return 0 if x is negative and will return the value itself if its positive. ReLU is mostly used because it speeds up the training process significantly and it does not saturate(the gradient is small if the input is elevated or small), unlike the tanh and sigmoid [18]. Figure 2.3 shows a plot by Krizhevsky et al. [3] paper, the figure shows that there is a 6x advancement in convergence

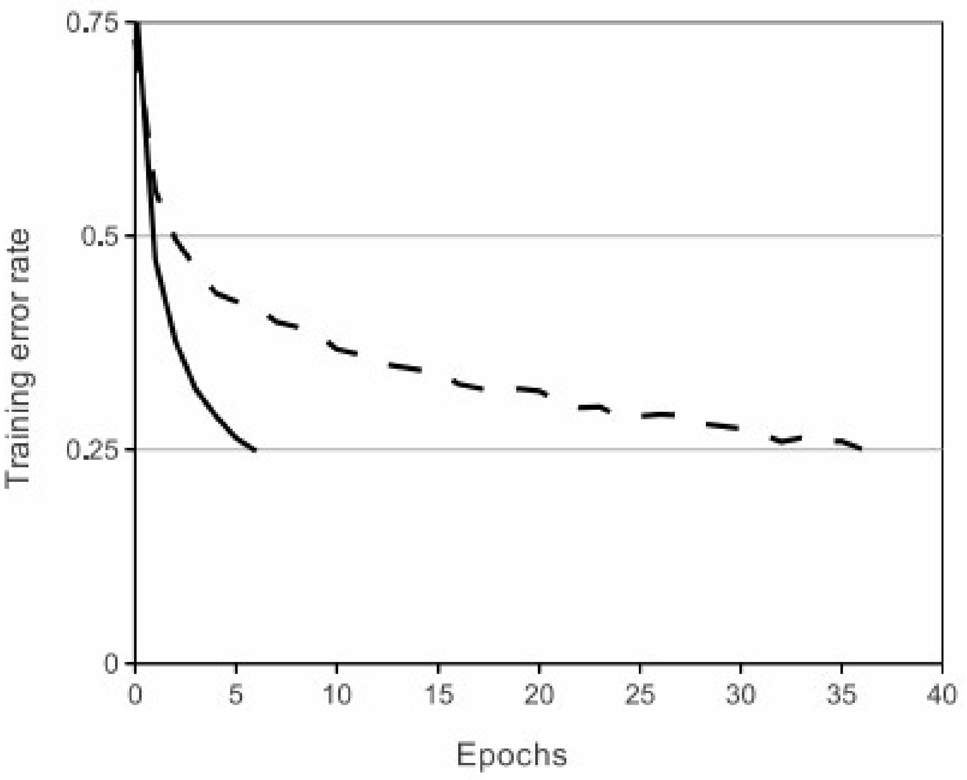


Figure 2.3: ReLU unit compared to the tanh unit from Krizhevsky et al. paper [3]

with the ReLU unit compared to the tanh unit. The bold line represents the ReLU and tanh is the dashed line.

2.5.5 Pooling Layer

The pooling layers basically have two main functions, the first one is that they help the CovNet to locate features regardless of which part of the image the features are located. This results in the model being robust against small changes in the position of the features of the images. The second function is that it also helps to reduce the size of the feature map. Therefore computations in the futures layers are relatively less complex. One way of performing pooling is using max pooling technique other less used techniques include average and sum pooling, the idea in max-pooling is like sliding a window through the feature map, the windows fills a number of arrays in the feature map therefore, you pick the largest among all the numbers and disregard the rest of the numbers. This technique is employed in the first, second and last Convolutional layers of the AlexNet architecture.

2.5.6 Dropout Layers

The following is inspired by an article on Medium [19]. This layer is used for regularising your network. The dropout is used in our CovNet to prevent over-fitting. The Fully connected layers eventually handle more parameters and therefore neurons become co-dependent on one another, and this leads to over-fitting during the training phase. When dropout is implemented in a network, it does not use all neurons during a particular forward or backward pass during training but, chooses random neurons at a specified probability for each pass. Figure 2.4 shows a neural network with an implemented dropout and another without dropout.

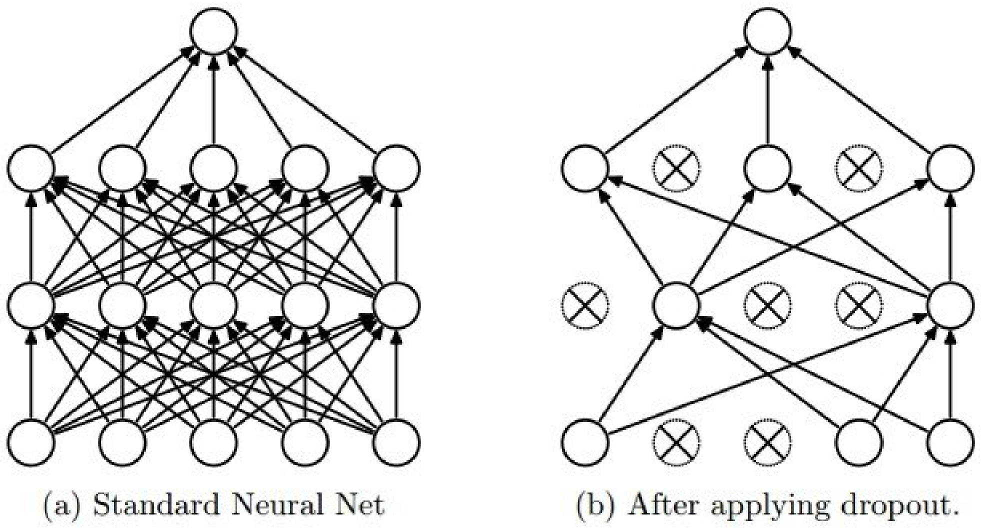


Figure 2.4: Left: shows a standard neural net. Right: shows a neural network where dropout has been applied on. Crossed units are dropped units. figure from Krizhevsky et al. paper [4]

2.5.7 Fully Connected Layer

Fully connected simply implies that every neuron from the previous layer is connected to every other neuron in the next layer. The output of this layer are high-level features of the image presented to the network. The output of this layer is a probability distribution of all the classes in the network and this is ensured by using an activation function, usually a softmax activation function. We use a softmax activation as opposed to other activation functions because, it handles low invigoration in images (say blurry charts) of your neural net with rather uniform distribution and to high invigoration in images (say large numbers, like sharp images) with prediction probabilities between 1 and 0. To put in simple terms, the Convolutional and pooling layers act as feature extractors and the last layer, the fully connected layer act as the classifier.

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.1}$$

The equation 2.1 is a softmax function, (z) represents a vector, so the function takes (in z) and crushes it to a probability value between zero and one [2].

2.6 Adding more steps

The layers mentioned above are a standard for every CovNet and with this kind of setup and a good dataset, most simple experiments are most likely to turn out with good results. However, when dealing with more complex problems in real-world other strategies like stacking more Convolutional layers and adding max-pooling at some points to reduce the dimensions of your data. The more Convolutional layers your network has the more it learns complex features, but also the more computational power you need. So the question then arises, how do you know how many layers to add so as to achieve the best results with the least possible computational power? The answer to this is experimentation, a lot of experiments and testing should be done in other to get the optimal structure [20]. Another step that improves image classification is some processing of the images before passing them as input into the machine learning algorithm. In the next section, some image processing techniques performed on the dataset before feeding the images to the network are discussed.

2.6.1 Image Processing Techniques

CovNets are very powerful and have the ability to learn features with some level of spatial invariability, this ability, however, is limited when it comes to choosing parameters in an efficient manner [21]. To further improve robustness to spatial invariance during testing, some preprocessing techniques like data augmentation are applied on the train data.

One relevant feature of CovNet is its multi-layer representation, which is responsible for the classification phase. This multi layer representation is not engineered but rather reliant on the data presented to it [22]. The multi-layer representation determines the kind of features important for the classification task. To find the right multi-layer representation, the network must be presented with a variety of instances of the image, so as to be able to capture the different appearance of the image. There are some parts of the image that can be varied and these parts are: the location, the viewpoint, and the size of an object or pattern. We, therefore, perform some preprocessing techniques in other to capture the 3 main ways images can be altered so that our model is better generalized.

- Data Augmentation: CovNets require a large train dataset in other to learn. The formation of such a dataset is a strenuous and expensive task. Data augmenting eases this task by creating label preserving transformations samples by transforming, rotating, flipping or even scaling the original samples. This technique helps the model to be robust to changes in position and orientation [23].

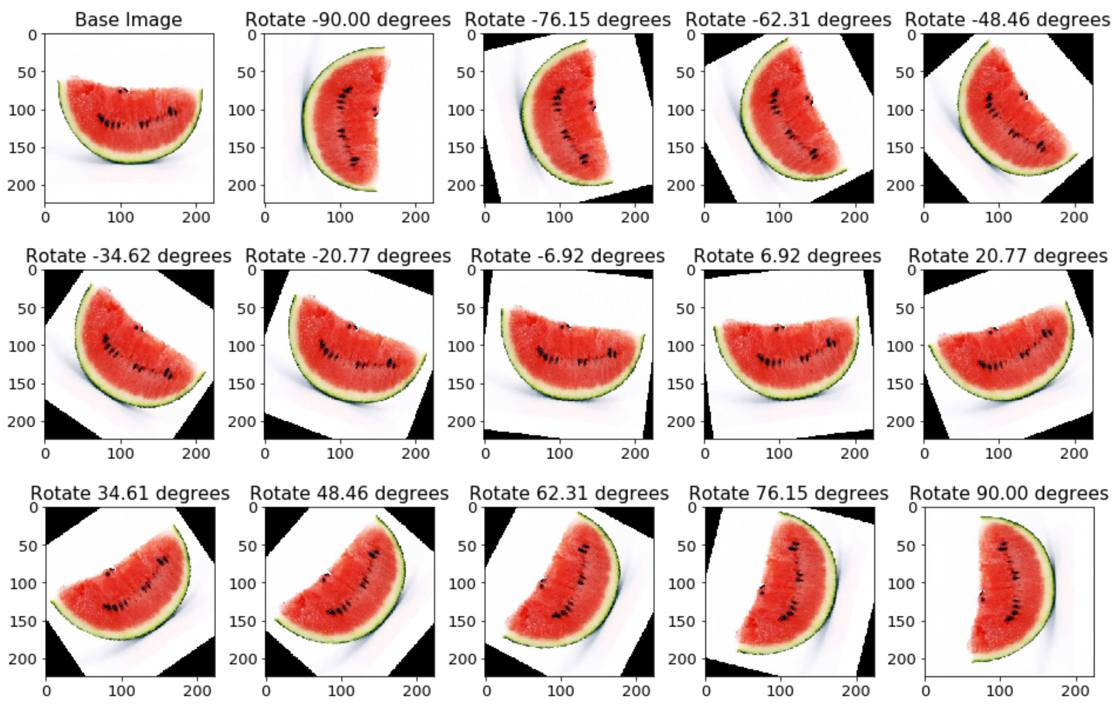


Figure 2.5: Data augmentaion technique of Rotation (at finer angles) source [5]

- Normalization: The process where the range of pixel intensity values is changed, this is done to bring the pixels to a range that our senses are more familiar to. The motivation for normalizing image data is to achieve consistency in the range of pixel values in the data. There are mainly two types of normalization techniques: the first one is Scaling, that is putting the values on the same scale and secondly centering, which involves balancing the data around a particular point. Both techniques have different benefits and should be selected based on our needs. For instance, scaling speeds up convergence and centering fights the problem of vanishing gradient [24].

2.7 Validation in Machine Learning

The following argument is based on Ricardo Gutierrez-Osuna's lecture [25]. Every machine learning algorithm has some parameters that can be tuned to adapt to a specific problem. For instance, in CovNets those parameters are the number of hidden layers, weights, and biases. During the training process of a machine learning algorithm, these tunable parameters are given specific configurations (model), we then check the classification performance of the algorithm with these estimated configurations. Classification performance is mostly estimated by finding the error rate of the entire dataset. We, therefore, try to tune our parameters to classify with the least error rate. In an ideal situation where our dataset consist of all possible variants of the samples to be classified, the error rate on the entire dataset will be the true error rate, but with real-world data this situation is unrealistic. The model will perform well on samples it has seen but will fail on new samples, this situation is known as over-fitting. In other to overcome this limitation some validation techniques are adapted during the training phase. These techniques are presented below.

2.7.0.1 Holdout

The holdout method involves splitting the dataset into two disjoint groups. Train set used to train the model and test set used to estimate the error rate of the model. The train to test split ratio can be a 60/40 or 70/30 or 80/20 split on the dataset. One drawback of this method is that if we have a small dataset, we cant afford to set aside a portion for testing only. Another drawback could be that if the difficult patterns end up in the test set the accuracy might be misleading.

2.7.0.2 Cross-validation

In this method the data is divided into say k subsets, the holdout method explained above is then repeated k times i.e each time one of the k subsets is used for testing and the other k-1 subsets are combined and used for training. The true error rate of the model is the average error across all the k trials performed.

2.7.0.3 Leave-one-out cross validation

The leave one out technique can be likened to the cross validation method. The main difference is that the k subsets are equal to the number of samples in the dataset. So, therefore, the model is trained on all the samples except one, and a prediction is made from that point. The true error rate, in this case, is also the average of all the k trials. The major drawback is that it can be computationally expensive.

2.8 Performance Evaluation

The following argument is based on Mohammed Sunasra's article on Medium [26]. After going through the process of putting together a dataset, estimating the best parameters for a machine learning algorithm, implementing a model and getting some output prediction on the classes trained on, the next step will be to find how well our model performed using some type of metrics. There exist quite a number of performance metrics that can be employed to evaluate a machine learning model. Some of these metrics that are used to evaluate classification problems are described below.

2.8.0.1 Confusion Matrix

The Confusion Matrix is an intuitive way used to describe the accuracy of a model. This metric is useful in problems where there exist multiple types of classes. A confusion matrix is a table that shows the number of correct and wrongly predicted values in each classification class. It gives insight into the type of errors being made by the classifier on each class.

	class 1 Predicted	class 2 Predicted
class 1 Actual	TP	FN
class 2 Actual	FP	TN

Table 2.1: Confusion Matrix

The numbers inside the Confusion matrix can be used to calculate all the performance metrics. The terms used in the table and how they are used to calculate the other metrics are explained below.

- TP (True Positive): Observations predicted as positive, that are actually positive.
- FP (False Positive): Observations predicted as positive, but are negative.
- TN (True Negative): Observations predicted as negative, but are negative.
- FN (False Negative): Observations predicted as negative, but are positive.

2.8.0.2 Accuracy

In any classification problem, the accuracy is the number of correct predictions made overall the predictions made. Equation 2.2 shows how the values in the confusion matrix are used to calculate the accuracy of a model. Accuracy is mainly used when there is an even number of samples in each class.

$$Accuracy = \frac{TP}{TP + FP + TN + FN} \tag{2.2}$$

2.8.0.3 Precision

Precision tells us if the portion of the sample in a particular class that was predicted as positive were actually positive. So it considers samples predicted as positive (TP and FP), and those that were actually positive TP. Equation 2.3 shows how the precision of a model is calculated.

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

2.8.0.4 Recall

Recall or Sensitivity tries to answer the question, of all the positive samples what portion were actually identified correctly. It considers the actual positives (TP and FN) and samples classified as positive (TP). Equation 2.4 shows how the precision of a model is calculated.

$$Precision = \frac{TP}{TP + FN} \tag{2.4}$$

2.8.0.5 F1 Score

F1-score is just a way to represent both precision and recall as a single value. It is taking the harmonic mean [5] of the precision and recall of the model. Equation 2.5 shows how the F1 score is calculated.

$$Precision = \frac{2 * Precision * Recall}{(Precision + Recall)} \tag{2.5}$$

There are many other metrics used in machine learning to evaluate a model but the ones mentioned above are mostly used when it comes to classification problems.

2.8.1 Deep Learning Frameworks

This section is inspired by a blog, 10 Best Frameworks and Libraries for AI [27]. Machine learning has become a friendlier field due to the amount of interest it has gained in recent times. Due to this interest, there has been a huge improvement and a lot of new developed libraries and frameworks. There is, therefore, a headache when choosing a framework for a particular. In this section, some of the most popular frameworks that are used for CovNets are discussed.

2.8.1.1 Theano

Theano is a library used with python and is flexible. There also exist very good documentation free to use. It is really good with numerical operations with multi-dimensional arrays.

Pros

- It is optimized for CPU and GPU.
- Performs numerical computations efficiently.

Cons

- Need to be combined with other high-level libraries like keras for better abstraction
- it's a low-level library

[5]http://www.statisticshowto.com/harmonic-mean/

2.8.1.2 TensorFlow

Tensorflow is built on C++ and Python programming languages and available in python. It is developed and maintained by Google developers. It is open source and very efficient when carrying out numerical tasks using data flow graphs. Tensorflow is arguably the most used neural network framework.

Pros

- The inner working can be visualized with Tensorboard.
- Performs numerical computations efficiently.

Cons

- Speed is a problem as python is not soo fast
- Limited access to pre-trained models

2.8.1.3 Caffe

Caffe is a very fast and efficient deep learning framework. It very easy to use especially for image classification.

Pros

- has great performance.
- you can train a model without actually writing any additional code.

Cons

- does not work well with new architectures
- also bad for implementing with recurrent neural networks.

2.8.1.4 Torch

Torch is another open-source library that works well with scientific and numerical calculations. Torch is based on the Lua programming language. It works well with operations such as slicing and indexing since it has a potent N-dimensional array. The main disadvantage of Torch is that access to documentation is very unclear. Also, there is no readily available code we can just use.

As we have seen there is a wealth of frameworks that exist, these frameworks facilitate easier the running of deep network code. Selecting the right one for your work will depend on your requirements. Either you go for speed, handling of computational complex calculations, less computational power or memory efficient frameworks.

2.8.2 Training process

This section is inspired by a blog written by ujjwalkarn [28]. In machine learning, a model is an artifact created during the training phase, this model can be likened to a function that maps specific features to their respective labels, the model is trained with a portion of the dataset called the train set. As the model is trained, the validation set is used to decide and pick which metric out of hyper-parameters, early stopping and architecture considerations yield the best performance. This helps to adjust and optimize the model [25]. Finally, the test-set in machine learning is the other portion of the dataset which was not used during the training phase. The test-set checks how well the trained model performs on unseen data by giving an unbiased assessment of the model. Most machines learning datasets are therefore divided into train, validation and test set. The process by which a CovNet performs classification is summarised below.

- All parameters (weights, filter, and biases) are initialized randomly.
- The network is then given as input the image. The forward pass is then performed (convolution, ReLU, pooling and fully connected layer), the output is a probability of each class.
- here the error in the output is calculated.

$$TotalError = \sum \frac{1}{2}(targetprobability - outputprobability)^2$$

- This step reduces the total error by performing back-propagation, the gradients of the error with respect to the weights are calculated. This is used to update parameters and therefore results in reducing the total error.
- In the final step, steps 2 to 4 are repeated for all images in the training dataset. The best parameters are then stored for the prediction of a new image given to the network.

2.9 Review of chart image Classification Techniques

In the rest of this chapter, four related works that performed chart image classification with unique methods and also achieved good results are described in details.

2.9.1 ImageNet Classification with Deep Convolutional Neural Networks

The first related work is the work of Krizhevsky et al. [3]. In this work, they show that a large, deep CovNet can achieve good results on complex tasks using purely supervised learning. The dataset used was the famous ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) image dataset, this dataset consists of roughly 1.2 training images, 50,000 validation images, and 150,000 testing images. All images were down-sampled to a fixed size of 256 $\times$ 256. The only pre-processing technique used was centering the image and nothing else. The architecture of the CovNet comprised of five convolutional and 3 fully connected layers. Every convolutional and fully-connected layer was followed by the ReLU non-linearity function. The first two convolutional layers are also followed by normalization layers. Max-pooling layers followed the normalization layers and the last convolutional layer. In other to reduce over-fitting, a dropout of 0.5 was used in the first and second fully connected layers. A learning rate of 0.01 which was later reduced by three times prior to termination was used. The training was set to 90 cycles and this took five to six days to train all 1.2 million images on two NVIDIA GTX 580 3GB GPUs. The results archived were better than the previous state of the art. An error rate of classifying on one of the classes considered most probable by the classifier as the top-1 was 37.5% and top-5 error rates on 5 classes was 17.0%.

2.9.2 Machine Learning Classification Algorithms to Recognize Chart Types in Portable Document Format(PDF) Files

Karthikeyani and Nagarajan's work [7] focuses on classifying charts found in pdf files. The steps employed for the classification task involves extracting texture features from the charts and then, using a machine learning technique for the classifying phase. The dataset consists of images extracted from various pdf's. The dataset is made up of 155 256*256 RGB images. Table 2.2 shows the details of the dataset used. The technique used involved extracting handcrafted features from the chart image and then, using these features as input for a model to be trained and tested. For the feature extraction phase, Gray Level Co-Occurrence Matrix (GLCM) is employed. GLCM is a technique which uses co-occurrence matrix to extract texture features of an image with the use of statistical equations. GLCM was used to extract eleven features, these included: area, median, minimum and maximum intensity, contrast, homogeneity, energy, entropy, mean, variance, standard deviation, and correlation. These extracted features are correlated to the pixels of the image. These extracted features are then stored using a 2-dimensional matrix-vector data structure. This structure has thirteen (13) columns and 'x' rows, where x is the size of the dataset. The features are stored in twelve (12) columns and the label is stored in the thirteenth column. The features and labels were stored in the below structure 1.

Chart Type	No of Charts	Chart Type	No of Charts
2D Bar chart	40	Doughnut 2D	7
3D Bar chart	16	Doughnut 3D	11
2D Pie chart	13	Line	35
3D Pie Chart	20	Mixed Chart	13

Table 2.2: Details on Dataset from [7]

Algorithm 1 This is the structure used to store the features

```
Struct FeatureVector {
float feature1; float feature2;
float feature3; float feature4;
float feature5; float feature6;
float feature7; float feature8;
float feature9; float feature10;
float feature11; float feature12;
int target;
}
```

This vector-like structure is then fed as input into three classifiers SVM, MLP neural network, and K-Nearest Neighbor. These classifiers are then trained and a classification model is formed for the recognition step. After this, to see if the model works well a test set consisting of new records is fed into the model for the model to predict their labels. Three metrics were used to check the performance of the model, these metrics are the error rate, classification accuracy, and speed of classification. The error rates obtained were MLP

0.30, K-NN 0.22 and SVM 0.23. The scores of the accuracy metrics were KNN (78.06%), MLP (69.68%) and SVM (76.77%) and finally the speed metric. The speed metric which is the sum of training and test time together was recorded with MPL having the slowest time of 8.38, followed by SVM with 0.31secs and KNN with 0.26secs. Even though these results were good, the paper proposed extracting features which are related to shape and curves since these features will carry more unique information to distinguish charts.

2.9.3 Architecture proposal for data extraction of chart images using Convolutional Neural Network

Our work is inspired by the technique used in the work of De Freitas et al. [29]. They proposed a way to extract the raw data visualized in different chart images. The paper talks about two main stages of accomplishing this task. Firstly, the classification of charts is done since it allows the different variety of charts to be detected automatically allowing the next step, which is the extraction of data from the classified charts. The paper, however, focuses on the first step which only involves the classification of charts. In this paper, a Convolutional Neural Network is used for the classification task. The Convolutional neural network encapsulates the characterisation and classification processes during its learning process, unlike other old techniques. The dataset used for this task were searched and downloaded by using Google image search. Table 2.3 shows the chart types which were collected and the number of train and test sets which the respective charts were divided into.

For the classification, a variant of CovNet called LeNet-based CovNet model is used. The model was implemented using Tensorflow[6], LeNet-based CovNet has an architecture which is comprised of 3 convolutional layers, followed by a fully connected layer. The model is trained in a way that the dataset is divided into mini-batches, samples of fixed sizes(100) are selected and fed into the CovNet, as a result of this process the model becomes robust since it learns to generalize from the different min-batches which are fed into the model. Also, all the images are converted to jpeg and resized to 224x224x3, that is, 224 pixels of height, 224 of width and 3 layers of output. The other parameters used were 1000 epochs and a learning rate of 0.003. The accuracy at the end of the training process was 70% which is pretty decent.

[6]https://www.tensorflow.org/

Chart Type	Test	Train
Area Chart	50	555
Bar Chart	50	657
Line Chart	50	489
Map	50	476
Pareto Chart	50	261
Pie Chart	50	361
Radar Chart	50	454
Scatter Chart	50	552
Table	44	236
Venn Diagram	48	304
Total	498	4345

Table 2.3: Number of Train and Test Dataset collected

2.9.4 Chart classification by combining deep convolutional networks and deep belief networks

In another paper by Liu et al. [6], a new approach was proposed for the process of chart classification. The process involves using CovNet to extract deep hidden features of charts and then employing a deep belief network to use the extracted features to predict the labels of the charts. Due to a difficulty in acquiring a large number of charts as training data, the famous ImageNet model trained with 1.2 million natural images was retrained with just over 5,000 collected charts images. The types of charts collected were pie charts, scatter charts, line charts, bar charts, and flowcharts. The architecture of the CovNet used is made up of five Convolutional layers, two fully-connected layers, and an output layer. The preprocessing steps for the images involve down-sampling them to 256 x 256 x 3, after which each is cropped to a size 227 x 227 from the center and its horizontal flip is extracted as the input of the CovNet, other parameters used for the CovNet include a learning rate that starts with 0.01 initially and later decreased by a factor of 0.1 after every lOOk iterations, the weight decay parameter was set at 0.0005 and a dropout rate of 0.5. This results in an output of a 5-way softmax which produces the distribution over the 5 class labels and this is used as input for the deep belief network. The deep belief network architecture has three hidden layers, whose dimensions are 5000, 500 and 2000. This results in a softmax predicting the probability distribution over the 5 categories of charts as output. The training process was done with 4000 randomly selected images and the rest were used as test set. The accuracy of the model after the evaluation was 75.4%. Table 2.4 shows the results gained. It shows the results after the training was done pre-trained with the natural images without deep belief networks, and also shows the results gained when the training was done with only the chart dataset but with deep belief networks.

Chart	CovNets	CovNets+DBN without pre-training	CovNets+DBN
Bar Chart	75.6%	45.6%	74.2%
Flow Chart	88.3%	56.8%	91.3%
Line Chart	71.2%	22.3%	67.9%
Scatter Chart	69.8%	44.5%	84.2%
Pie Chart	58.1%	50.1%	59.4%
Ave. Accuracy	72.6%	43.9%	75.4%

Table 2.4: Comparing Results of Proposed Framework from (Liu et al. [6])

2.10 Summary

This chapter discussed the background literature on chart image classification. The two main approaches to chart images classification were mentioned and explained in detail. In Section 2.5, CovNet a machine learning approach used commonly for visual image recognition is reviewed and discussed, also some additional steps like image processing techniques that improve image classification are mentioned and discussed. In Section 2.7 validation techniques to help choose the tunable parameters in a machine learning algorithm are discussed. In Section 2.8 some performance evaluation metrics that help us know how well a model is performing are presented and discussed. Towards the end of this chapter, some related works that inspired this thesis were discussed.

3 Approach

3.1 Overview

In this chapter, we present the proposed approaches used in this thesis for chart image classification. As discussed in the previous chapters, the proposed approaches involve two main steps, namely, the image processing step and classification step. How the dataset was created is discussed in section 3.2 and Section 3.3. In Section 3.4 the process which the train, validation and test data are acquired are discussed. In Section 3.5 the proposed architecture and important parameters of the CovNet is discussed. The processing techniques used for the images are discussed in Section 3.6. Lastly, the various parameters used for the training of the network, and how the training was done is discussed.

3.2 Dataset

The following paragraph is inspired by a blog written on what to look for in training data [30]. The aphorism 'Garbage in Garbage out' is a valid statement when it comes to creating a dataset for machine learning. The machine learning technique learns from whatever data is fed to it. So if a dataset of good quality is fed into the algorithm, then the model created will also be of good quality. The dataset creation stage is, therefore, an important stage. In most works that involved chart image classification, the dataset used consisted of chart images downloaded from Google and a few others were obtained by extracting chart images from pdfs. This approach we believe is limited since, we do not know the programming languages, the libraries used and parameters used in creating these charts. This missing information is relevant since it tells us how diverse our resultant dataset is. For example, how are we sure that all the charts that were downloaded from Google, were not only created in Python or Java?, and in such a case how well will a new chart created with Matlab or R be classified. For this reason, the dataset comprised of image charts created with different libraries and a variety of programming languages. Some random chart images from other works were manually labeled and added to our created dataset. In the next sections, the various datasets and the languages used in plotting are described. To make the dataset as diverse as possible, charts created in each programing language used a different set of CSV files (raw data).

3.2.1 Dataset for Matlab

The Data used for creating the plots in Matlab were randomly chosen from Project Dataset [31], a free CSV data repository, DatPlot [32] and Plotly CSV repository in github [33]. The datasets are multidimensional and compiled from normal day to day activities like dating, what makes people happy etc, and objects like cameras and cars. On the average, the datasets used contain about 500 instance and 5 different columns. The biggest dataset is called Speed dating data. It is made up of over 8,000 observations of answers to survey questions about how people rate themselves and how they rate others on several dimensions. The smallest dataset used has 33 instances and 12 columns. It contains information about cars. The number of gears and speed, just to name a few attributes.

3.2.2 Dataset for R

For the plots in R, 13 random CSV files where downloaded from an archive of datasets distributed with R called Rdatasets [34]. Rdatasets is a collection of dataset distributed with R. On the average there are 80 instances and 5 columns in each dataset. The biggest CSV file is the Australian athletes' dataset. It's made of 203 instances and 14 columns and contains attributes like sex, height, weight, and sports. The smallest dataset is the Canadian Women's Labour Force Participation. This dataset has 30 rows and 7 columns. It contains information like average wages of women, percent of adult women in the workforce etc.

3.2.3 Dataset for Python

The data used for creating the plots in Python were 15 randomly seleted csv files also from Rdatasets [34]. The biggest dataset among the 15 is the Monoclonal gammapothy data, it contains data about the natural history of patients with monoclonal gammapothy of undetermined significance. The dataset is made up of 1384 observations with 10 columns, it has attributes like age, sex, time of death and last contact in months. On the average, each dataset contains about 200 instances and 7 columns of multi-dimensional data. The smallest dataset, however, contains only 33 instances with 11 columns and is called the Nuclear Power Station Construction Data. The data in this file is made of information about the construction of 32 light water reactor (LWR) plants constructed in the U.S.A in the late 90's.

3.2.4 Dataset for Java

For the plots created in java, I used the dataset made available by Plotly [33], a github repository of CSV datasets used in the Plotly API examples. 14 random CSV files were downloaded, the biggest file has 1002 instances and 9 columns, and on the average, each file contains about 100 instances and 9 columns. The smallest file, however, is made of 33 instances and 12 columns called the mtcars file. It contains information about a variety of different car models like the number of gears, speed etc. The table 3.1 contains the names of all CSV files that were used in the different languages with the different plotting programs.

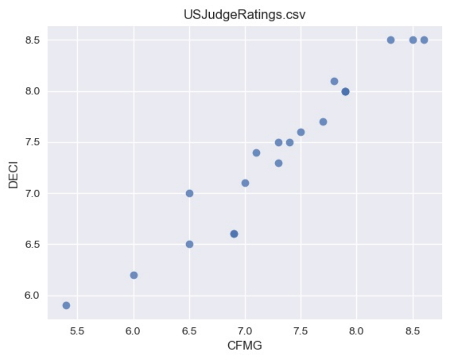
Datasets			
Python	Matlab	R	Java
3d_line_sample.csv	Camera.csv	ais.csv	3d-line-plot.csv
LightFordward.csv	Cars.csv	Angell.csv	3d-scatter.csv
line_3d_dataset.csv	speedDating.csv	Baumann.csv	2011_flight_paths.csv
longley.csv	Cereal.csv	Bfox.csv	2011_us_export.csv
loti.csv	happiness.csv	cane.csv	auto-mpg.csv
lung.csv	TestData1.csv	carprice.csv	candlestick_data.csv
nuclear.csv	TestData2.csv	Chirot.csv	finance-charts-
timeseries.csv	mpg.csv	Davis.csv	apple.csv
USJudgeRatings	okcupid-	Ericksen.csv	globe_contours.csv
WVSCultural.csv	religion.csv	Florida.csv	hobbspearsontrial.csv
wind_rose.csv	spectral.csv	Highway1.csv	motor_trend_test.csv
volcano.csv	stockdata.csv	Pottery.csv	nz_weather.csv
uspop2.csvm	subplots.csv	Prestige.csv	volcano.csv
tips		salinity.csv	iris.csv
		urine.csv	mtcars.csv

Table 3.1: Names of datasets used in each plotting program

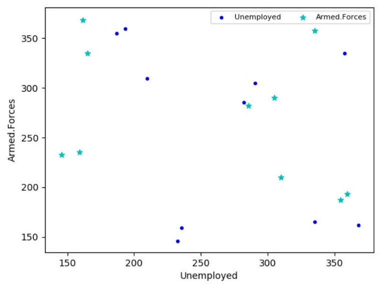
3.3 Creating Plots

The motivation for creating a variety of plots to capture all type of plots used in scientific papers was acquired by inspecting the datasets of Architecture proposal for data extraction of chart images using CovNet paper [29] and Analysing visual information in the scientific literature [35]. Scripts in various languages were written to handle the plotting and labeling process automatically.

All datasets for a particular plot (example scatter plot for python) are put into one folder. The scripts read each CSV file column by column while creating the plots. Table 3.2 describes how the plots were created in each language. The type column describes the different variety of a particular plot, for example, bar charts can be of type stacked, grouped, vertical and horizontal bar charts, also scatter plots types can be a scatter plot consisting of one type of marker, one scatter plot with multiple markers and finally a scatter plot with a line showing the correlation between the plots. The parameter column shows some parameters that were used to create the plots. For example, geom_point is a parameter available in the Ggplot library in R to specify the color, shape and size of the markers in scatter plots. Markerstyle parameter shows the type of marker used in the line or scatter plot. The LegendPosition as the name suggests indicated which part of the chart the legend should appear. In most cases, however, the default parameters of the various libraries were used to create the plots. Figure 3.2 shows two different types of bar charts. Figure 3.2a is a stacked bar chart and figure 3.2b a normal vertical bar chart. The library column shows the different plotting libraries used, the parameter column describes parameters that were changed and finally the number of plots created was also added. The images below are sample images that exist in our dataset of created plots for each language.



(a) Python scatter plot with circular markers



(b) Java scatter plot with circular and diamond markers

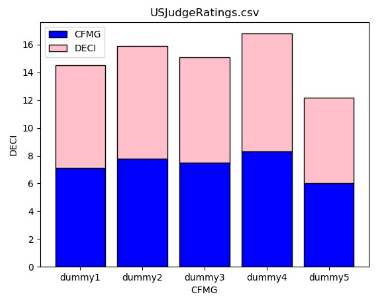
Figure 3.1: Sample scatter plots in the dataset

Table 3.2: Overview of the varied parameters and libraries used for creating plots in the different plotting programs

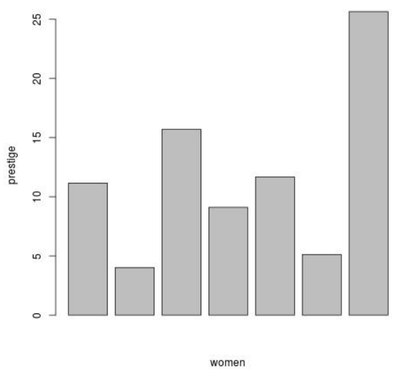
	Language	Library	Parameters
Scatter plot	Python	Matplotlib v2.1.2 Plotly v2.5.1 Seaborn v0.8.1	MarkerStyle [’o’, ’*’, ’.’, ’+’,’x’]
	MATLAB	Default Plotly	MarkerStyle [’o’, ’*’,’+’,’x’,’s’]
	R	Plotly Lattice Ggplot2	MarkerStyle [’o’, ’*’, ’+’,’x’,’s’] geom_point(shape(1,2,16), size (3-4))
	JAVA	XChart 3.5.1 jfreechart 1.0.1	MarkerSize (15 -18)
Bar charts	Python	Matplotlib v2.1.2 Plotly v2.5.1 Seaborn v0.8.1	
	MATLAB	Default	Width of bar(14-16)
	R	Default,Plotly R Library ggplot2	space (0-3)
	JAVA	XChart 3.5.1 jfreechart:1.0.192 javafx.scene	PlotOrientation (vertical or horizontal) with error bars
Line chart	Python	Matplotlib v2.1.2 Plotly v2.5.1 Seaborn v0.8.1	Linestyle [’-’, ’–’, ’-.’, ’:’]
	MATLAB	Default Plotly	MarkerStyle [’o’, ’*’, ’.’, ’+’,’x’,’s’] markersize [8-10]
	R	Default,Plotly R Library ggplot2	
	JAVA	XChart 3.5.1 javafx JFreeChart	MarkerSize (12-16)
Box Plots	Python	Matplotlib v2.1.2 Plotly v2.5.1 Seaborn v0.8.1	
	MATLAB	Default	
	R	Default,Plotly R Library ggplot2	
	JAVA	XChart 3.5.1 Jfree smile	LegendPosition (topleft,topright)

Scatter Plots		
Language	Number of plots	Type
Python	1165	Unique markers, With legends, multiple markers regplot
Matlab	1008	
R	1009	
Java	1002	
Bar Charts		
Language	Number of plots	Types(bar)
Python	1018	Horizontal and Vertical, Stacked, Grouped bar charts Error bars
Matlab	1063	
R	1022	
Java	1143	
Line Charts		
Language	Number of plots	Types(Line with)
Python	1019	Markers, Multiple Lines
Matlab	1013	
R	1132	
Java	1089	
Box Plots		
Language	Number of plots	Types(Box with)
Python	1012	Notches, Multiple Boxes
Matlab	1032	
R	1113	
Java	1036	

Table 3.3: Overview of the number of plots and different varieties plots in the different plotting programs

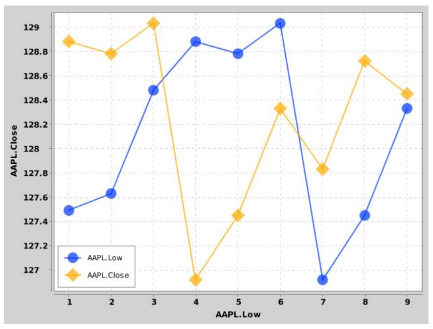


(a) Matlab stacked bar chart (bar width 16)

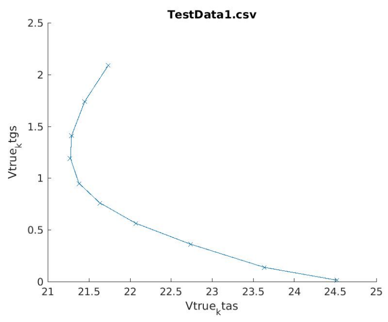


(b) R horizontal bar chart (bar width 16)

Figure 3.2: Sample Bar Charts in our dataset

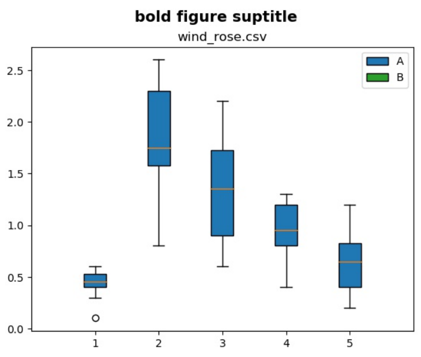


(a) Java line chart with diamond and circular markers

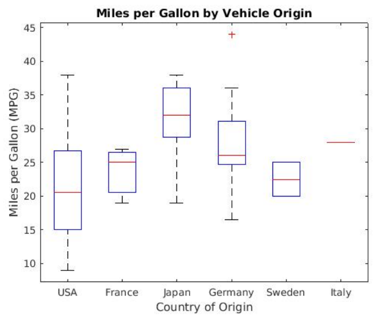


(b) simple Matlab line chart with Asterix marker

Figure 3.3: Sample Line Charts in our dataset



(a) Vertical multiple boxplots in python

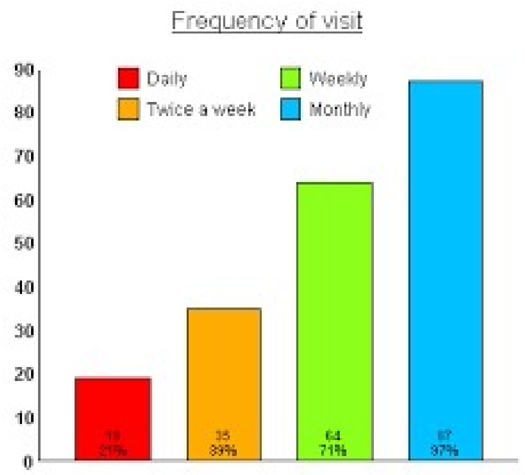


(b) Vertical multiple boxplots in Matlab

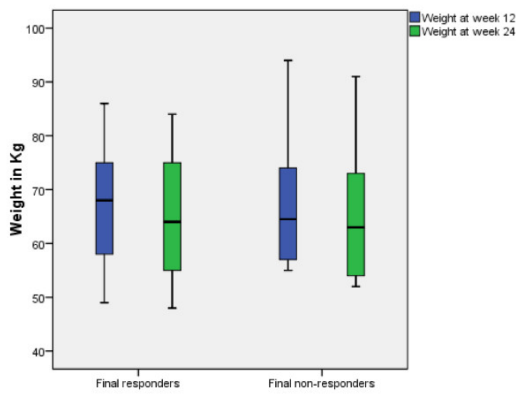
Figure 3.4: Sample Box Plots in our dataset

3.3.1 Random Charts

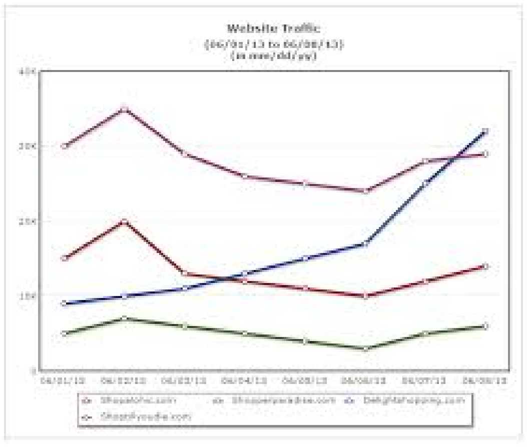
In other to have the most diverse dataset, so as to achieve a robust model, we decided to pick 120 random images for each chart type from Junior et al. [29] and Po-shen et al. [36]. The line charts, bar charts, and scatter plots were already labeled. These plots were sampled from the research of Junior et al. and the box-plots which we had to label manually, were taken from the work of Po-shen et al. since, the prior did not contain box-plots. The randomly picked chart images, were then added to the already created dataset for training and testing. Figure 3.5 shows sample images from the already created chart images from the paper of Po-shen et al. and Junior et al.



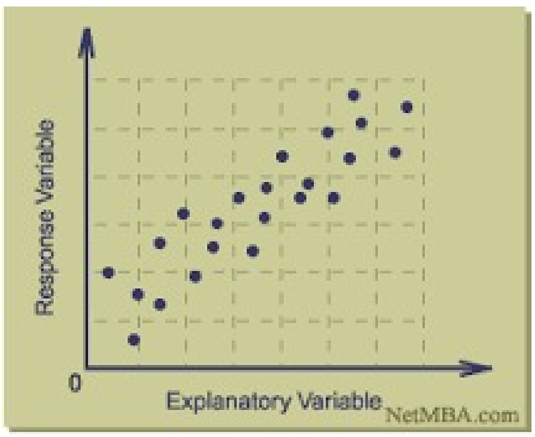
(a) A bar-chart



(b) A box-plot



(c) A line-chart



(d) Scatter-plot

Figure 3.5: Example Chart images from Po-shen et al. and Junior et al.

3.4 Training, Validation and Test set

Figure 3.6 summaries how our dataset was created and split into various portions for the training and testing stages. After creating the various plots. 80% of each class was randomly chosen as train data and the rest was used as test data. Also 96 of the 120 random chosen charts mentioned in the previous section was randomly picked and added to their respective classes in the train set, and the rest of the 24 which forms 20% were added to their respective test samples. For the validation phase, 20% of the train set is split randomly for validation during the training phase. Table 3.4 shows a summary of how many samples were added to each class for training and testing.

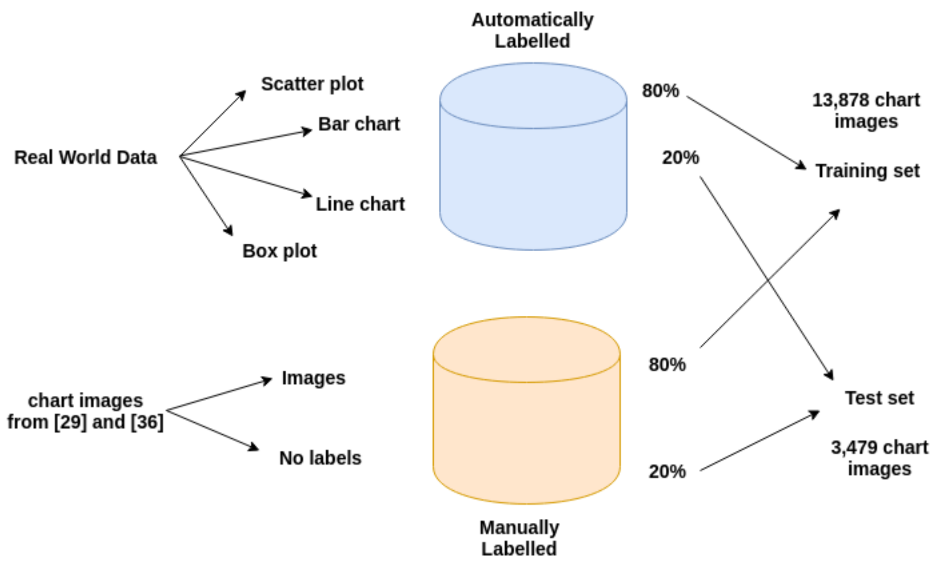


Figure 3.6: How the train and test dataset was created

	Train set	Test Set
Scatter plot	3442	862
Bar Chart	3491	875
Line Chart	3497	876
Box plot	3448	866
Total	13878	3479

Table 3.4: Summary of the dataset used for training and testing

3.5 The Architecture

As mentioned in previous sections, in our first experiment, our model is trained with a CovNet. The architecture of the CovNet used is inspired by the famous AlexNet CovNet architecture [3]. The AlexNet architecture was used in the famous ImageNet LSVRC-2010 contest to classify 1.2 million into 1000 classes. Our CovNet architecture just like the AlexNet architecture contains 8 learned layers, these 8 layers are made up of five Convolutional layers and 3 fully connected layers. A Relu is used after some Convolutional layers and fully connected layers. Another parameter employed in our architecture is a dropout, a dropout is applied in the first and second fully connected layers of our network and finally, max pooling is applied in some of the layers. A summary of our CovNet architecture used to train our model is shown in Figure 3.7. The preprocessing steps, the parameters used, and why they were employed are explained in details in the rest of this chapter.

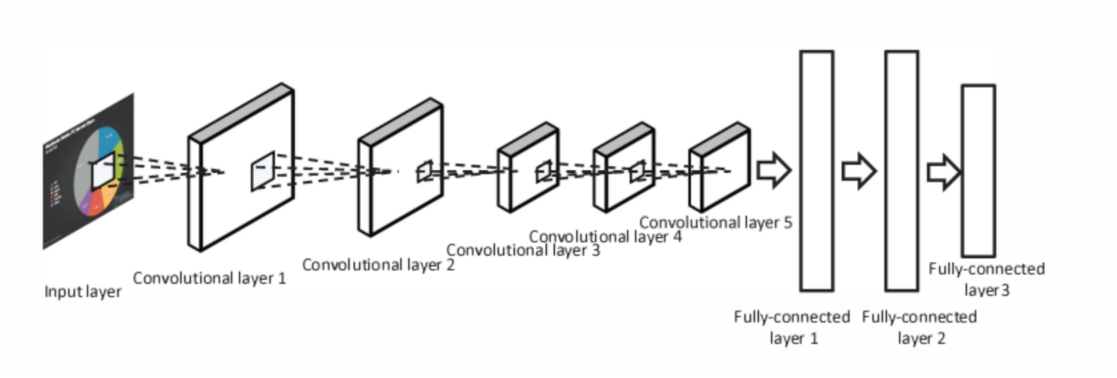


Figure 3.7: Architecure of CovNet used to train our model source [6]

In our second experiment we retrained a model called MobileNets. The following argument is based on the work of Howard et al. [37]. MobileNet is built on depthwise separable convolutions. This means it factorizes a standard convolution into a depthwise convolution and pointwise (1x1) convolution. Table 3.5 shows the architectural structure used to train the MobileNet model. Dw means depthwise and s1 and s2 stand for number of stride.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw /	s1 $3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 3.5: Body Architecture of MobileNet

3.6 Preprocessing Techniques

Archiving high performance by any machine learning technique is reliant on the quality and quantity of the data fed into the network. Datasets must comprise of diverse data, this helps the network to generalize well during the testing phase. To help increase the quantity and also generate more diverse data we employ some image preprocessing techniques. The preprocessing techniques used in this thesis are described in the rest of this section.

- Image Resizing: The images have varying sizes and due to the presence of a fully connected layer, we resize all the images to the size of 227 x 227. With this fixed image size processing in the dataset in batches is also possible.
- Padding: Padding was set to SAME in the convolutional layer, which means all images are padded with zeros so that the output dimensions are equal.
- Image Scaling: All images were rescaled by a factor of 1.0/255, to enable the network to handle the images in the same way.

3.6.1 Data Augmentation

In this thesis, no data augmentation techniques were used. This mainly because we do not expect our images to be presented in different angles and under different lighting conditions as real world images are normally presented. This is due to the fact that images are expected to be extracted with a pdf extractor and all the bounding boxes of the chart images are expected to be horizontally aligned.

3.7 First Method

The first experiment was trained using the Tensorflow framework. The dataset are all read in with OpenCV [1] and resized to 277 x 277 x 3 since they are RGB image. The CovNet model used is the AlexNet model with some modifications.

For the training process used, we defined the building blocks of the network as follows. The layer weights and biases are initialized randomly. We chose a batch size of 32 to train our data for computational simplicity. In tensorflow

[1] http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_

tf.nn.conv2d function can be used to build the convolutional layer. It takes as input the images, weights, and biases. We then apply a max-pooling to the convolution to reduce the dimensions. After that a Relu is used as our activation function, we just take the output of the max-pooling and apply Relu on it. This convolution layer is repeated 5 times each learning different features. The features after they are learned are reshape to 1D vector by the flatten layer before being sent to the fully connected layers. This layer also takes as input random weights and biases and performs the standard z=wx+b operation here. We set dropout probability to 0.5 in our network, this means nodes are dropped out with a probability of 1-05. We add a condition that applies a Relu on first two of the three fully connected layers. We then define a cost function to choose our optimum weights. There is a simple function in tensorflow **softmax_cross_entropy_with_logits** which takes the output of the fully connected layer and the real labels to find these weights. With a learning rate of 0.0001 to minimize cost we then optimize our model. We then call a softmax when we have the lowest cost to get the probability of each class. The detailed layer parameters and layer order are shown in Table 3.6.

Operation	Filter	Depth	Stride
Conv1 + Relu	11 * 11	96	4
Max Pooling	3 * 3		2
Conv2 + Relu	5 * 5	256	4
Max Pooling	3 * 3		2
Conv3 + Relu	3 * 3	384	4
Conv4 + Relu	3 * 3	384	4
Conv5 + Relu	3 * 3	256	4
Max Pooling	3 * 3		2
Dropout(rate 0.5)			
Fc6 + dropout 0.5 + ReLU		4,096	
Fc7 + dropout 0.5 + ReLU		4,096	
Fc8 + softmax output			

Table 3.6: The parameters used in the AlexNet with some modifications. The order of layers is the order in which data is passed through. depth is the amount of output feature maps or neurons, filter is kernel size

3.8 Second Method

For the second experiment, the transfer learning technique used in the work of Liu et. al [6] was employed. We used a model pre-trained by Google called the MobileNet [2]. MobileNet was chosen because of its small size and less computational power needed to run. The MobileNet_0.50_224 model was the version used for this thesis. The MobileNet model was trained with 1.2 million images with 1000 classes from the ILSVRC-2012-CLS image classification dataset [38]. Some of the classes in this dataset are rule, zebra, Dishwasher, and dalmatian.

For the training process, a retrain python script originally developed by the TensorFlow authors for the purpose of using your images to train was used. The retrain scripts downloads the MobileNet model, adds a new model and retrains the new model with the chart image dataset we created. The scripts is customizable and below are the parameters we customized.

- The validation_percentage parameter is the part of the dataset used for validation and 20% was used for this thesis.
- testing_percentage parameter is the part of the dataset used for testing of final model and 20% was used for this thesis.
- test_batch_size=-1 was chosen, which means all the images in reserved for testing was used.
- print_misclassified_test_images: was used so that the misclassified images where shown.
- how_many_training_steps is the number of epochs the network should run and 200 was used in this thesis.
- learning_rate: This is something you'll want to play with. I found 0.0001 to work well.
- validation_batch_size: It was Set to -1, this tells the script to use all our data reserved for validation to validate on.

3.9 Summary

In this chapter we discussed our approach to classifying chart images. How our dataset was created and which raw data samples were used to create our

[2]https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html

chart images was discussed initially. In section 3.4 we discussed how the full dataset was split into train, validation and test sets for the experiments to be performed. We then talked about our modification of the AlexNet architecture used for our model training. Afterwards, the preprocessing techniques used on the dataset was explained and finally the two experiments chosen to help answer our research question were explained in details.

4 Experimental Results

4.1 Overview

This chapter presents the results of the experiments performed in the previous chapter. We answer the research question "How Well Can We Classify the Four Different Types of Plots (Line-Charts, Bar-Charts, Scatter-plots, and Box-plots) in Scientific Literature? " by presenting the results obtained after the experiments were performed. In Section 4.2 we present various metrics on the first methods used and Section 4.3 shows the results obtained in the second experiment.

4.2 Results of First Experiment

We trained with different epoch sizes and realized after 4500 epochs there was no significant increase in accuracy. After testing on 3478 images we achieved an accuracy of 88.4%. 88.4% was considered a good score since we had a wider variety of chats for each chart type, therefore, more complex dataset as compared to the work Amara et al. [10] that got a score of 89%. Since we needed to gain more insight into the data to know the strengths and weakness of our model we computed the confusion matrix and used the scores to compute other metrics like precision, recall and f1 score. Figure 4.1 shows the precision, recall and F1 measure of the results obtained after the experiments. What is interesting is the huge difference in scores between the precision and recall of the Line charts. From the confusion matrix, we saw that it was due to the fact that most of the line charts were misclassified as scatter plots. The other charts, however, had pretty good scores on these metrics. We then visualized the results of the confusion matrix with a heat map. Figure 4.2 shows a heat map derived with the confusion matrix. 0 stands for the bar chart, 1 stands for box plot, 2 for line plot and 3 for scatter plot. Again we can see that bar charts where mostly classified correctly followed by box plot and line charts in that order and finally scatter plot were the least classified correctly. Another interesting observation was that most line charts were misclassified as box plots.

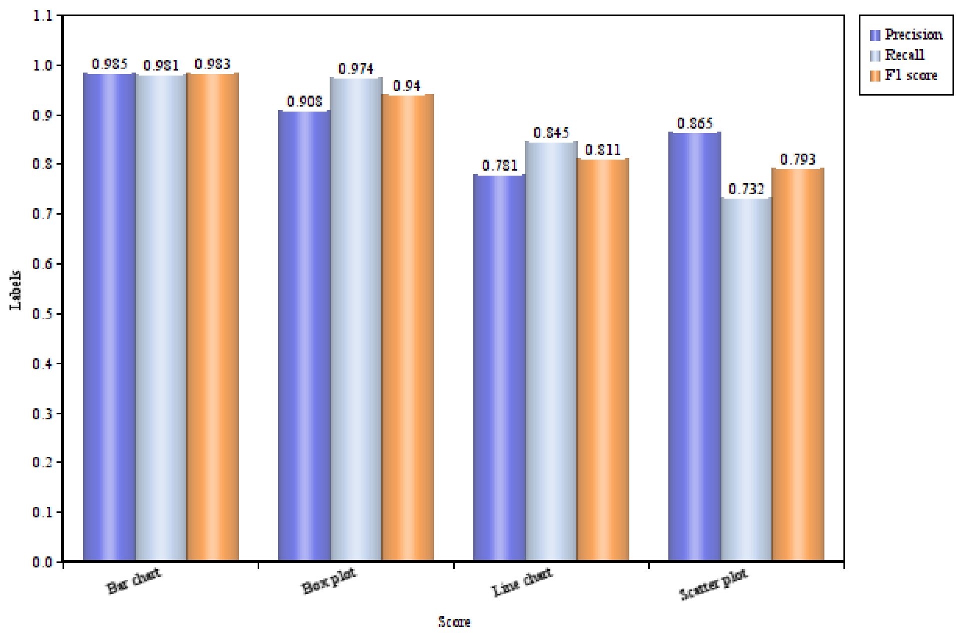


Figure 4.1: A bar chart showing the precision recall and F1 score of the experiment

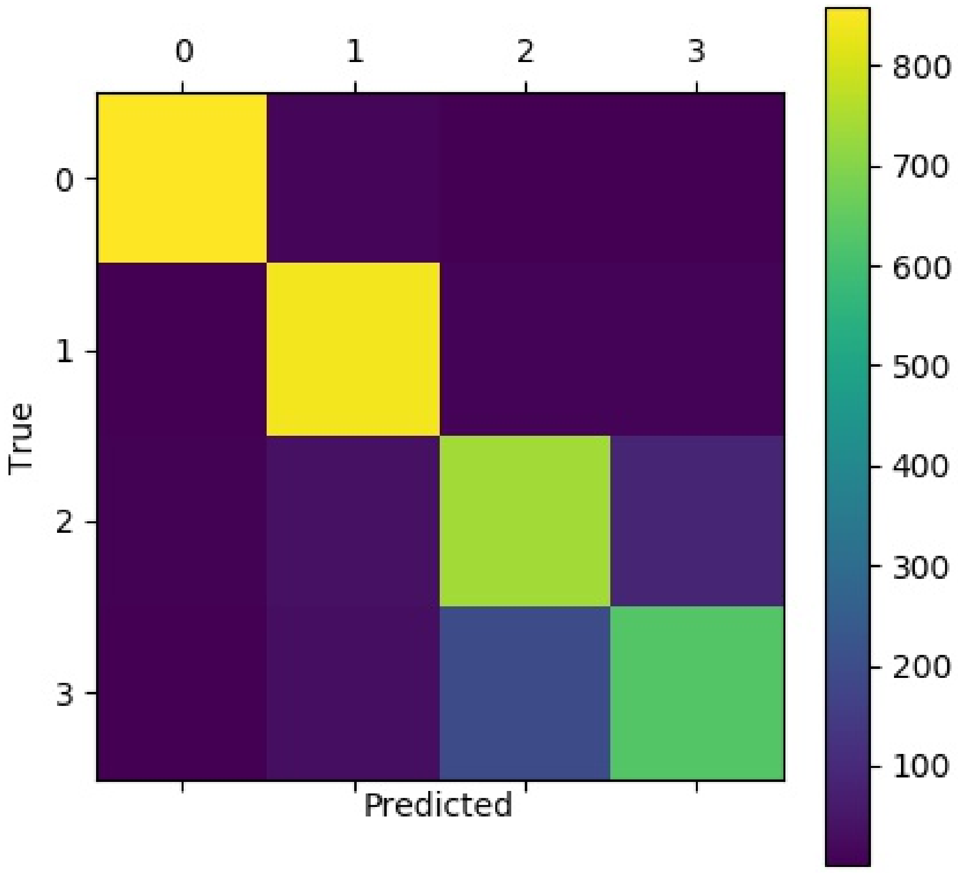


Figure 4.2: Results obtained shown with a Heat Map

4.3 Results of Second Experiment

The second method which involved using the pre-trained ImageNet model and retraining it with our created chart image dataset executed with just 200 epochs and got an accuracy of 97% with a cross entropy of 0.04. For retraining, 60% of the whole dataset was split for training, 20% for validation and 20% was for testing. Table 4.1 shows the confusion matrix generated with the various scores obtained after testing our model on 3485 random samples.

		Predicted			
		Bar chart	Box chart	Line chart	Scatter plot
Truth	Bar chart	850	12	2	2
	Box plot	19	822	5	20
	Line Chart	1	10	854	24
	Scatter plot	1	1	7	855

Table 4.1: Confusion matrix generated with results obtained from the test data

We used the scores from the confusion matrix to calculate the precision, recall and f1 score of the model. Figure 4.3 shows the visualized scores of the precision, recall, and f1 score to better understand our results. At a glance, it can be observed that the difference between the precision and recall of the scatter plot is relatively huge. Scatter plot classification had a higher recall but a lower precision which is more relevant for the future step of data extraction. Box plots and Line charts, however, had a higher precision which is better in relation to our thesis.

From the confusion matrix, we observe that box plots were the least classified correctly with most of the predictions on box plots being misclassified as Scatter plots and Bar charts. Another interesting observation is that most of the misclassified samples of Bar charts were classified as Box plots.

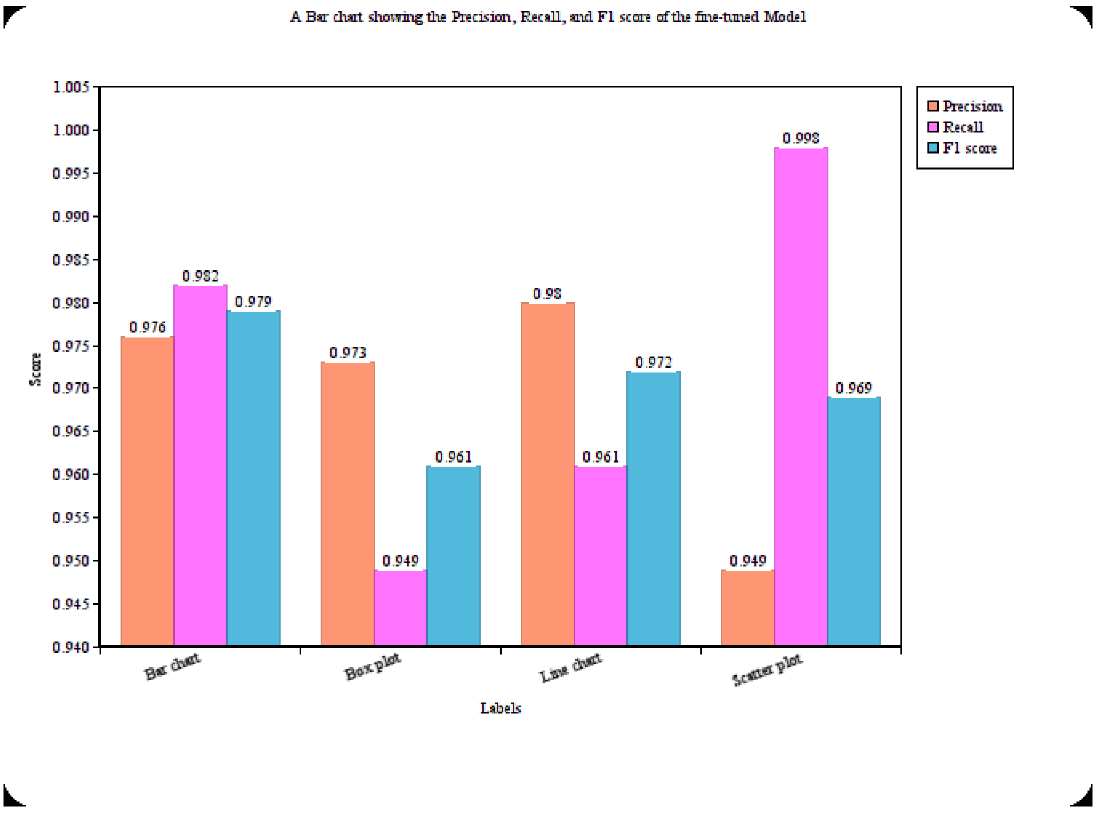


Figure 4.3: A bar chart showing the precision recall and F1 score of our fine-tuned model

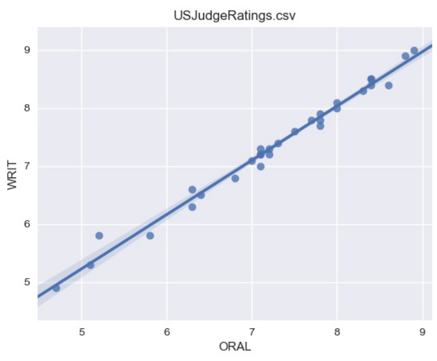
4.4 Discussion

Our results show that fine-tuning on the pre-trained model achieved better results. We believe the reason for the better results was because the pre-trained model learned edges, curves, and sharp corners from the huge dataset given to it. These learned weights were then transferred to our task. Table 4.2 shows how our experiment compares to the work of Amara et al. [10]. As shown in the table our second experiment performed best and the score of the first experiment was not too far off that of Amara et al. work. We, therefore, believe we gained better results because of the transferred knowledge gained from the pre-trained model.

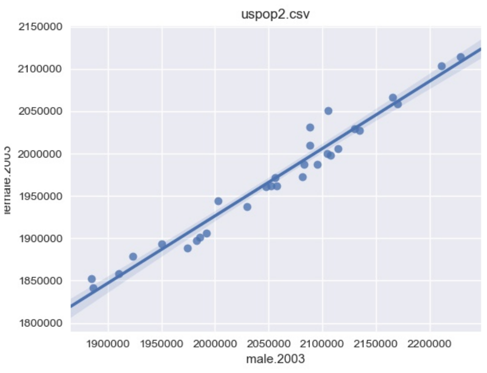
Model	Bar chart		Box chart		Line chart		Scatter plot	
	P	R	P	R	P	R	P	R
LeNet [10]			96.20%	77.50%	92.10%	87.10%	97.00%	95.50%
CovNet	97.30%	99.0%	97.30%	94.50%	75.60%	93.40%	91.50%	70.6
MobileNet+ fine-tuned	97.6%	98.2%	97.30%	94.90%	98.60%	96.10%	94.90%	99.0%

Table 4.2: Comparison of results of experiments with other methods

From the results shown in our first experiment line charts had a bad precision score as compared to the other charts, also, from the confusion matrix of the first experiment we saw that this was as a result of scatter plots being misclassified as line chart and vice versa. We believe this was because of the similarity between the visual appearance of some variants of line charts and scatter plots. Our dataset contained quite a number of scatter plots which had a regression line and these had a striking resemblance to a type of line chart with markers and a line connecting these markers to show their relationship. Also, there were some scatter plots which were positive and negatively correlated with all the markers appearing in a straight line and this could have been as well considered as a line chart by our classifier. Figure 4.4 shows sample scatter plots which resemble line charts and would be difficult for a computer to different. There also quiet a number of box plots misclassified as line charts this can be attributed to the fact that box plots are made of lines and markers (outliers) which resemble a type of line chart with markers and a line connecting these markers. A number of box plots were misclassified as bar charts, we believe this also because box plots and bar charts are somehow similar in their appearance. Figure 4.5 shows 9 randomly chosen misclassified charts after our model was tested.



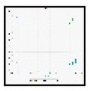
(a) scatter plot

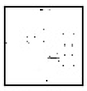


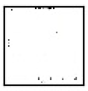
(b) scatter plot

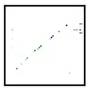
Figure 4.4: Sample scatter plots with a best fit line created in python with the Plotly library

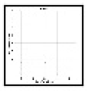
In the second experiment results show that line charts were mostly misclassified as scatter plots as in the case of the first experiment. Also we had a relatively high number of box plots misclassified as scatter plots. Figure 4.6 shows a few samples and what they were misclassified in the second experiment.

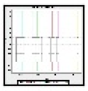

Tr: scatter plot, Pr: bar chart

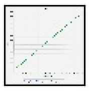

Tr: box plot, Pr: bar chart

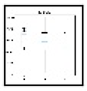

Tr: scatter plot, Pr: box plot


Tr: line plot, Pr: bar chart


Tr: scatter plot, Pr: bar chart

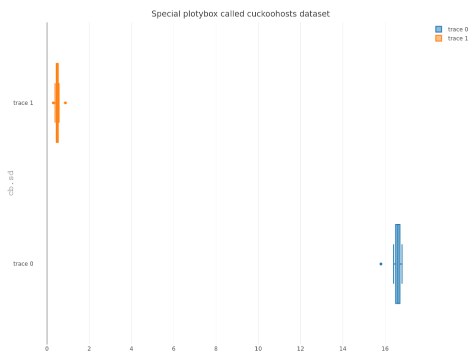

Tr: box plot, Pr: bar chart


Tr: line plot, Pr: bar chart

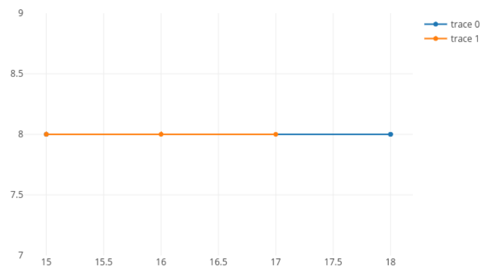

Tr: box plot, Pr: line plot

Tr: box plot, Pr: bar chart

Figure 4.5: Sample plots that were misclassified in the first experiment

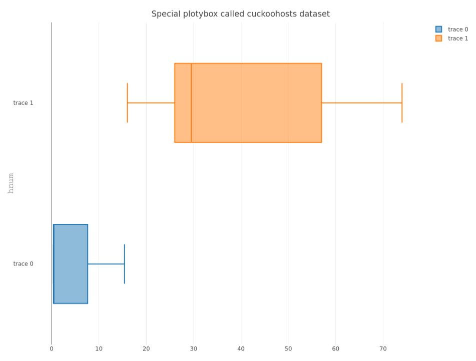


(a) Box plot misclassified as scatter plot

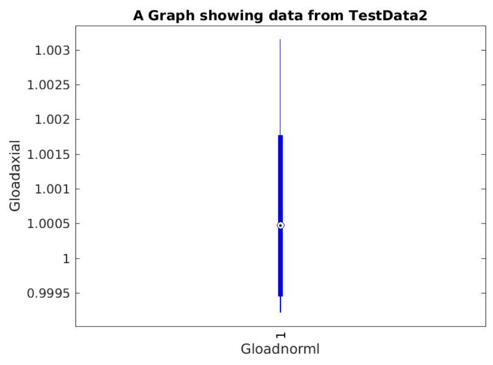


(b) Line chart misclassified as scatter plot

Figure 4.6: Sample plots misclassified in the second experiment



(a) Box plot misclassified as bar chart



(b) Box plot misclassified as a Line chart

Figure 4.7: Sample plots misclassified in the second experiment

4.5 Summary

5 Conclusion

6 Bibliography

[1] Matange, S. Box plot legend. https://blogs.sas.com/content/graphicallyspeaking/2017/04/13/box-plot-legend/, 2018. [Graphically Speaking; accessed 18-Jun-2018].

[2] karpathy. Cs231n convolutional neural networks for visual recognition. http://cs231n.github.io/linear-classify/#softmax, 2018. [Online; accessed 11-Jun-2018].

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[5] Prasad Pai. Data augmentation techniques in cnn using tensorflow. https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9, 2017. [Medium; accessed 26-Jun-2018].

[6] Xiao Liu, Binbin Tang, Zhenyang Wang, Xianghua Xu, Shiliang Pu, Dapeng Tao, and Mingli Song. Chart classification by combining deep convolutional networks and deep belief networks. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 801–805. IEEE, 2015.

[7] V Karthikeyani and S Nagarajan. Machine learning classification algorithms to recognize chart types in portable document format (pdf) files. *International Journal of Computer Applications*, 39(2), 2012.

[8] Thermopylae Sciences + Technology. Humans process visual data better. http://www.t-sciences.com/news/

humans-process-visual-data-better, 2018. [Thermopylae Sciences + Technology. (2018).; accessed 18-Jun-2018].

[9] Few, S. Visual business intelligence – abela's folly – a thought confuser. https://www.perceptualedge.com/blog/?p=2080, 2018. [Perceptualedge.com.; accessed 17-Jun-2018].

[10] Jihen Amara, Pawandeep Kaur, Michael Owonibi, and Bassem Bouaziz. Convolutional neural network based chart image classification. 2017.

[11] Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. Revision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 393–402. ACM, 2011.

[12] V Shiv Naga Prasad, Behjat Siddiquie, Jennifer Golbeck, and Larry S Davis. Classifying computer generated charts. In *Content-Based Multimedia Indexing, 2007. CBMI'07. International Workshop on*, pages 85–92. IEEE, 2007.

[13] Richard Boyle, Bahram Parvin, Darko Koracin, Nikos Paragios, and Syeda-Mahmood Tanveer. *Advances in visual computing*. Springer, 2007.

[14] Yan Ping Zhou and Chew Lim Tan. Bar charts recognition using hough based syntactic segmentation. In *International Conference on Theory and Application of Diagrams*, pages 494–497. Springer, 2000.

[15] Mingyan Shao and Robert P Futrelle. Recognition and classification of figures in pdf documents. In *International Workshop on Graphics Recognition*, pages 231–242. Springer, 2005.

[16] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 13–23. Springer, 2000.

[17] DANIEL SHIFFMAN. *THE NATURE OF CODE*. Free Software Foundation, 2012.

[18] Balwally, A. What is the role of rectified li. https://www.quora.com/What-is-the-role-of-rectified-linear-ReLU-activation-function-in-CNN, 2018. [Online; accessed 9-Jun-2018].

[19] Rodriguez, J. Convolutional neural networks for the rest of us part iii: Benefits and motivation. https://medium.com/@jrodthoughts/

convolutional-neural-networks-for-the-rest-of-us-part-iii-benefits-and-motiv
2018. [Online; accessed 10-Jun-2018].

[20] Adam Geitgey. Machine learning is fun! part 3: Deep learning and convolutional neural networks. https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networ
2016. [Medium ; accessed 14-Jul-2018].

[21] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[22] Nanne van Noord and Eric Postma. Learning scale-variant and scale-invariant features for deep image classification. *Pattern Recognition*, 61:583–592, 2017.

[23] Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation. *arXiv preprint arXiv:1708.06020*, 2017.

[24] pietz. Images.

[25] Akshay Balwally,. What is the difference between validation set and test set? https://www.quora.com/What-is-the-difference-between-validation-set-and-test-set, 2016. [Online; accessed 7-Jun-2018].

[26] Ricardo Gutierrez-Osuna. Validation. http://research.cs.tamu.edu/prism/lectures/iss/iss_l13.pdf, 2017. [Wright State University ; accessed 27-Jun-2018].

[27] Anton Shaleynikov. 10 best frameworks and libraries for ai. https://dzone.com/articles/progressive-tools10-best-frameworks-and-libraries, 2018. [Dzone; accessed 14-Jul-2018].

[28] ujjwalkarn. An intuitive explanation of convolutional neural networks – the data science blog. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/, August 2016. (Accessed on 05/16/2018).

[29] Paulo Roberto Silva Chagas Junior, Alexandre Abreu De Freitas, Rafael Daisuke Akiyama, Brunelli Pinto Miranda, Tiago Davi Oliveira De Araújo, Carlos Gustavo Resque Dos Santos, Bianchi Serique Meiguins, and Jefferson Magalhães De Morais. Architecture proposal for data extraction of chart images using convolutional neural network. In *In-*

formation Visualisation (IV), 2017 21st International Conference, pages 318–323. IEEE, 2017.

[30] Editor. what-to-look-for-in-training-dataset. https://medium.com/@jrodthoughts/convolutional-neural-networks-for-the-rest-of-us-part-iii-bene 2018. [Online; accessed 5-Jun-2018].

[31] James Eagan. Project datasets. https://perso.telecom-paristech.fr/eagan/class/igr204/datasets. [Online; accessed 20-April-2018].

[32] Michael Vogt. Datplot. https://vincentarelbundock.github.io/Rdatasets/datasets.html, 2011. [Online; accessed 20-April-2018].

[33] plotly/datasets. Latex — Wikipedia, the free encyclopedia. https://github.com/plotly/datasets, 2011. [Online; accessed 20-April-2018].

[34] vincentarel bundock. Rdatasets. https://vincentarelbundock.github.io/Rdatasets/datasets.html, 2011. [Online; accessed 20-April-2018].

[35] Po-shen Lee, Jevin D West, and Bill Howe. Viziometrics: Analyzing visual information in the scientific literature. *IEEE Transactions on Big Data*, 4(1):117–129, 2018.

[36] Po-shen Lee, Jevin D West, and Bill Howe. Viziometrix: A platform for analyzing the visual information in big scholarly data. In *Proceedings of the 25th international conference companion on world wide web*, pages 413–418. International World Wide Web Conferences Steering Committee, 2016.

[37] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[38] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.