Chart type classification and object recognition using deep learning

Anirudh Gokulaprasad

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8RZ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

September 21, 2020

Abstract

Charts are an effective tool to interpret and visualize data. When the underlying data points are not available, it is necessary to extract data from these charts. To perform this, the chart image has to classified into its respective chart type and the objects in the images should be localized and detected to efficiently extract the data. This project aims to perform chart image classification and object recognition steps by using a diverse custom dataset of manually annotated chart images. For this, the concepts of convolution neural network (CNN) and transfer learning has been utilized for classifying the chart images along with a pre-trained model for object recognition. The chart image classifiers can robustly classify 7 types of charts while the object recognition model can localize and detect 3 chart objects. Subsequently, the evaluation performed on the validation set has shown an accuracy of 74.2 percent on the baseline CNN model and 96.8 percent on the transfer learning model, while the mean average precision (mAP) for object recognition was recorded at 1.6 percent.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: ANIRUDH GOKULAPRASAD Signature: ____________________

Acknowledgements

I would like to thank my supervisor Dr.John Williamson for his guidance and valuable inputs which helped me to complete this project.

Next, I would like to thank all my professors of this data science program who helped me acquire the necessary knowledge and skills in order to complete my project.

I would also like to thank each and every developer who have contributed to the open-source community as this project was possible because of some powerful open-source packages

Finally, I would like to thank my parents and friends who provided the moral support which gave me the confidence to continue working on the project through a very difficult worldwide pandemic.

Contents

Chapter 1

Introduction

In the current era of data driven world, accurate representation of data is important to arrive at key decisions. There are various tools to represent data. Among them, Charts are one of the most popular tools. This is because charts provide a visual representation of the data which helps to understand and decipher underlying patterns. For example, in scientific research papers charts are used to represent the result of experiments and analysis which are then used to understand the underlying patterns. However, we do not always have access to the data that was used to produce these chart images. In order to get access to this data, we have to extract the data points and the corresponding chart elements from the chart image. For this, there are 3 major steps to be performed. They are:

1. Chart type classification
2. Chart object recognition
3. Chart data extraction

This project aims to perform the first 2 steps of this process. The first step is to classify the chart images. This is performed as there are different types of charts such as bar chart, pie chart, scatter plot etc with variations in chart style and orientation within each type. Next, I perform object recognition on chart images to localize and identify the various objects in the chart such as title, axis labels and legend. Here, the user only has to input the chart image, as the chart image classification and object recognition is performed automatically without the need for any label or annotation file input from the user.

To achieve these objectives, I implement a custom baseline CNN image classification model, a transfer learning based image classification model and a pre-trained object recognition model to train and perform validations on a custom dataset. The results from these models are analysed and ablation study is performed to understand and explain the performance of these models on the dataset.

In the following chapters, I will explore the background of the project and provide details on the implementation of the models. Then, I present the results of the evaluations that have been conducted. Finally, I conclude by discussing the advantages and disadvantages and provide possible future work that could improve the chart type classification and object recognition.

Chapter 2

Analysis

In this chapter, the background survey of the existing systems has been performed based on which the key objectives of the project are formulated. Further, an overview of the various chart types, techniques and tools that are used to achieve these objectives are presented.

2.1 Background survey

In this section, existing systems that leverage one or both of image classification and object recognition to build a chart data extraction system has been surveyed. A few different approaches have been used for this, which are discussed along with the pitfalls of these systems.

ReVision [13] is one of the early systems to successfully classify, extract data from pie chart and bar charts and redesign the charts. The average classification accuracy across all types of charts is 80 percent. However, this uses a support vector machine (SVM) based approach for classification resulting in this low chart classification accuracy. It can classify 10 types of charts. The dataset consists of 2500 images crawled from the web, which shows a lack of diversity in the source of chart images considered.

The chartsense tool by microsoft [7] builds on the work of ReVision by using the CNN based GoogLeNet model by achieving a 91.3 percent accuracy for classification, which shows the benefit of using CNN based model over other machine learning approaches for image classification. They build on the ReVision corpus by expanding the number of images in the corpus to nearly 7000 for the same 10 chart types. This dataset expansion is performed by using images from the internet and setting an inclusion criteria to include the images in the dataset. However, the issue of lack of diversity of chart image sources in ReVision is not addressed.

A much more recent tool is the chart-text [1]. The aim of this tool is to produce a chart image description by performing image classification and object recognition for pie chart and variants of bar chart such as horizontal, vertical and stacked bar charts. By using transfer learning on a mobilenet model, they achieve a classification accuracy of 99.72 percent. There are 7 objects of interest for object recognition. They claim that faster rcnn has produced the best accuracy in detecting small objects and therefore has been deployed to perform object recognition. The mean average precision (mAP) for intersection over union (IoU) of over 50 percent is approximately 93 percent. The dataset of chart images are generated using matplotlib package in python, which again shows a lack of diversity in the source of chart images. Only the detections that have confidence score greater than 0.5 are recorded, which along with the lack of detailed information on how the images are annotated does not provide the true performance of the system.

Of all the tools surveyed, it is clear to the see the performance of the system has improved by the use of CNN models for classification in chartsense as opposed to the svm approach in ReVision, albeit on a much larger dataset. Further, transfer learning approach was applied to a much smaller variety of charts in the chart-text system with great success.

However, the common thing to note is the lack of diversity in the dataset on the source of chart images. Also, the possibility of including scanned chart images in the dataset has not at all been considered in any of the systems. Further, these systems do not provide any details or evidence to explain the performance of these models. While chart-text performs object recognition, the lack of detailed information on how the images are annotated and the object size statistics makes it difficult to put the results in context.

2.2 Objectives

Based on the learning from background survey, the key objectives of the project is to :

1. Create a diverse, well-defined and controlled dataset that includes images from the internet, chart creation tools (matplotlib, microsoft excel etc) and scans by using a set exclusion criteria
2. Establish a custom CNN model as a baseline
3. Create a transfer learning model and compare its performance to the baseline CNN model
4. Establish well-defined chart image annotation process to perform object recognition
5. Similar to chart-text [1], deploy a faster-rcnn approach to perform object recognition
6. Present the object size statistics for the dataset to give context to the results for object recognition
7. Perform ablation study to understand and explain the performance of these models

2.3 Chart types

This project focuses on the charts that are used to represent data and not the graphs that represent the nodes and edges of a network. It is important to clarify this as the terms graphs and charts are used interchangeably at times but in the world of computer science, graphs and charts and definitively distinguished.

There are several types of charts that are used to represent data. For this project, I have considered the 10 types of charts that were used in ReVision [13] and ChartSense [7] systems. However map, table and venn diagram are not conventional chart types and therefore are not considered for this project. As a result, I use the remaining 7 types of charts, some of which have sub-types that are mentioned below:

1. Area chart
2. Bar chart - sub types: horizontal bar chart, vertical bar chart
3. Pie Chart
4. Scatter plot - sub types: scatter plot with regression line, scatter plot without regression line
5. Line chart
6. Pareto Chart
7. Radar chart

2.4 Techniques used

2.4.1 Convolution neural networks

Classifying chart images can be achieved through a number of techniques including k-nearest neighbours (KNN) and support vector machines (SVM). However, a type of neural network called the convolutional neural networks (CNN) has gained prominence in performing image classification. Between 2012 to 2015, CNN based models have won the ImageNet Large scale visual recognition challenge (ILSVRC) which is the benchmark competition for image classification and object recognition [11]. CNN consists of one or more convolutional layers followed by fully connected dense layers. Each convolutional layer can be configured to use different settings that is best suited to solve a specific problem, since CNN has various window size options, activation functions and internal layer configurations

2.4.2 Transfer learning

Transfer learning is a technique that utilizes a pre-trained model's knowledge to solve a related problem. Training a state-of-the-art image classification model from scratch using a benchmark dataset such as imagenet [4] can take huge computing resources and could take several days to complete training. Instead, by applying transfer learning, we can use a pre-trained model to our problem by utilizing its knowledge. This knowledge is called weights. In tranfer learning, the top layer or a certain small percentage of the pre-trained model's layers can be unfroze and retrained for a specific task. While retraining, the weights are updated so that the model is much more adept at solving the problem at hand.

2.5 Tools used

In order to create maintainable, robust and efficient software systems, it is important to use stable, well-documented and easy-to-use tools.

Here, google colab is used as the development environment as it provides a free NVidia Tesla cloud gpu which accelerates the speed of developing and training the models [5]. The deep learning framework keras (version 2.4.3) is used to build the classification models as it provides a variety of pre-trained models with well organized documentation and quick prototyping capabilities [3]. Detecto (version 1.1.6) is used to build the object recognition model as it is based on pytorch (version 1.4) and torchvision (version 0.5.0) which provides a custom dataloader and data augmentation capabilities [15, 16]. Labelimg tool (version 1.8.2) was used to annotate the images as it provides option to annotate both in yolo and pascal voc format with quick shortcuts to speed up the annotation process [17].

These software tools and packages were used to design and implement this project, which is discussed in following chapter.

Chapter 3

Design and Implementation

This chapter explains the design process that was used to build the dataset, the classification and object recognition models

3.1 Dataset design

I created two custom controlled datasets of chart images, one for chart image classification and another for object recognition. Both these datasets contain the same set of images for training and validation but vary in their directory structure.

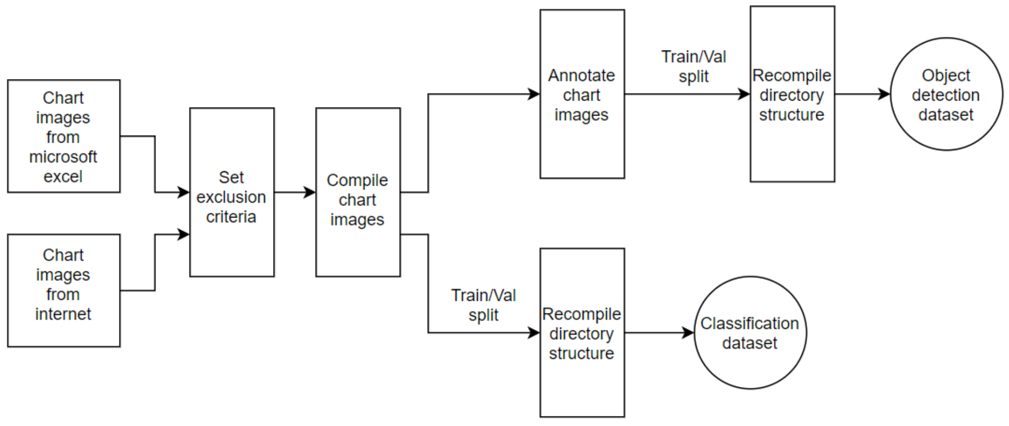


Figure 3.1: Dataset creation process

I expect majority of the source of chart images input by the user to be from the internet. Also, I expect the minor sources of chart image input to be from chart creation tools such as excel and from scans. Therefore, to reflect this diversity in source of the input image, 80 percent of the dataset was built using chart images downloaded from the internet while the remaining 20 percent of dataset was built from chart images created using microsoft excel. Since scanned images tend to have noise, simulated gaussian noise is injected to the dataset as part of the data augmentation, for which the process is explained in the data augmentation subsections.

To build the dataset, I downloaded chart images from google search by using the chart name as keyword (Eg: bar chart images, pie chart jpg etc). The bulk image downloader software was used for this process [6]. The downloaded chart images were then manually removed that contained:

1. 3D chart images
2. Collage of multiple charts in the same image
3. Menu icons
4. Handwritten sketches
5. No axis scale (except pie chart and radar chart)
6. not at least one of title, legend or axis label

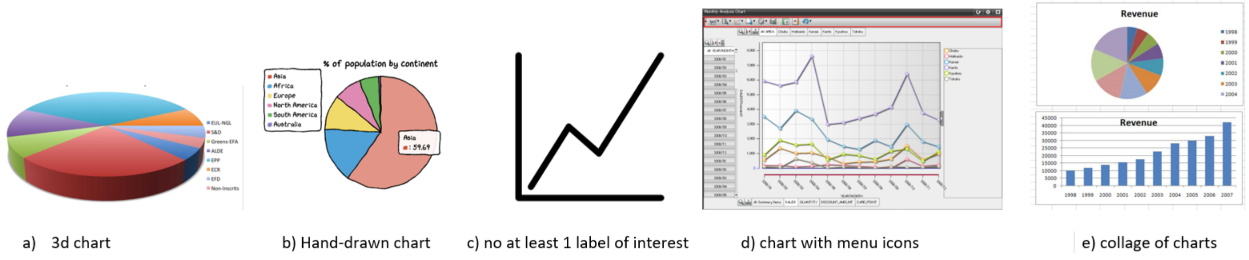


Figure 3.2: Sample of excluded chart images

Using this criteria, I created charts using microsoft excel to represent the charts from chart creation tools. Subsequently, I built a corpus of chart images for the chart types mentioned in chapter 2, the statistics for which is presented in table 3.1.

Total no.of images	No.of training images	No.of validation images	No.of Classes (Chart types)	No.of images per class	Image naming format
700	560 (80%)	140 (20%)	7	100	bar01, line66 etc

Table 3.1: Dataset statistics

3.1.1 Classification dataset

This dataset was created following the recommended structure for image dataset from keras. Here, the train and validation folder consists of subfolders where each subfolder is considered as a class. The name of the subfolder is taken as the class label.

3.1.2 Object recognition dataset

This dataset is built following the recommended structure from detecto, where the images are stored in training and validation folders. Each image has a corresponding pascal voc (xml) annotation file of the same name.

Image annotation process

The images are manually annotated using the labelimg tool by drawing a bounding box around the objects of interest and saving it in the pascal voc format. I have used 3 labels for annotating the images: title, legend and axis label. Here, the title is considered to be the text on top of the image. If there is no recognizable text on top of the image, then title is considered to be at the bottom of the image. The legend can be at any position in the image. Both the x-axis and y-axis labels are commonly annotated as axis label.

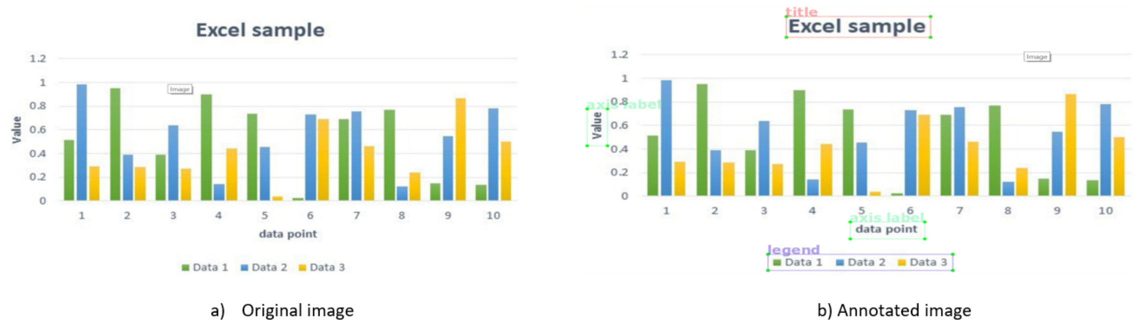


Figure 3.3: Example of annotated image

3.2 Chart image classification

3.2.1 Data Augmentation

Data augmentation is an important part of creating a CNN model as it provides the capability to introduce meaningful augmentations to dataset. This helps us to simulate a better representation of real-world data. Here, I have made augmentations for zoom range, shear angle, brightness and injected variable gaussian noise.

Scanned images can have minor variations in their zoom range as the distance between the viewer and the image changes when scanning using a mobile device. This can also happen by editing or resizing the images. Therefore, zoom range has been set to vary by 20 percent. Likewise, there can be minor variations in the angle of scanning the images. To account for this, the augmentation for shear angle has been set to vary by 20 percent that distorts the image along an axis. Also, the

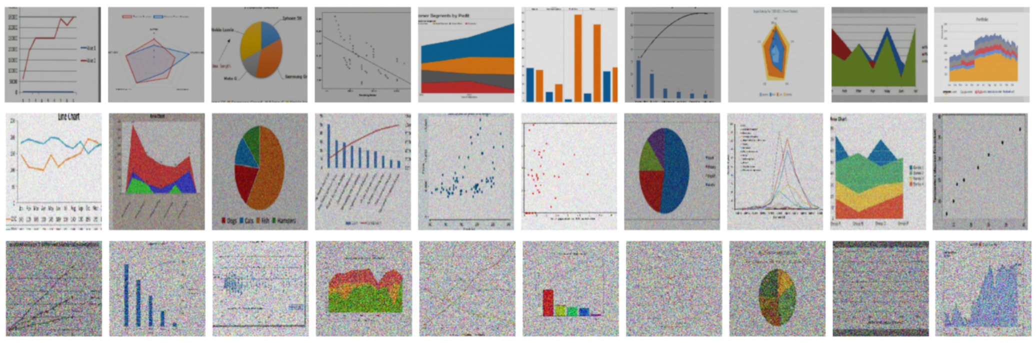


Figure 3.4: Varied levels of gaussian noise in chart images

changes in the lighting conditions can affect the images while using a mobile device. If the image's brightness drops below 50 percent or more from the original image, then the image will be too dark for the model to recognize. Therefore, the image brightness has been set to vary between 60 to 90 percent of the original image.

The gaussian noise has been applied randomly on each batch. The mean is set to 0 while the standard deviation has been set to be a random exponential of 0.15. This ensures difference in noise level throughout the dataset, thereby realistically representing the varied levels of noise that we can expect from scanned images from scanners and mobile devices as shown in figure 3.4.

The image size of 224x244 was found to work well upon empirical evaluation as it gives a good balance between execution speed and model accuracy.

These data augmentations have been applied only on the training set. The images in the validation set have been kept free of any data augmentation to resemble real-world data as close as possible. Both the training and validation set has been re-scaled to the range of 0 to 1 for the model to process the images efficiently.

3.2.2 Baseline CNN model

A simple CNN model is used as the baseline. This model consists of 3 CNN blocks which contains a max pooling layer at each block. On top of this is a fully connected dense layer. All layers have relu activation function with a window size of 3. The optimizer of choice is adam as it is sensitive to the changes in the learning rate and has low memory requirements [8]. This makes it easier to research and test prototypes. Due to memory constraints, a batch size of 16 was used. The learning rate follows an inverse time decay function with a decay rate of 8. As shown in figure 3.5, this decreases the learning rate from 0.001 to 0.0003 over 300 epochs which allows the model to learn at a much slower rate, thereby learning better feature representations to extract from the chart images. Since I perform a multi-class classification, categorical cross-entropy loss is used.

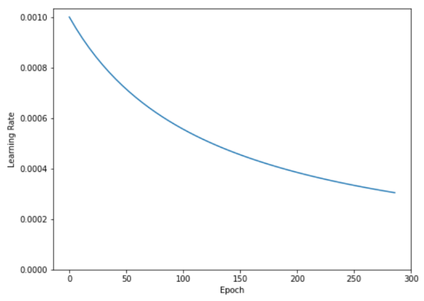
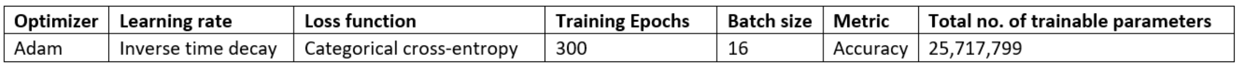


Figure 3.5: Learning rate decay

The performance of the model is measured in terms of its accuracy of chart type classification. The total number of trainable parameters are 25,717,799. A summary of the hyper-parameter settings is provided in table 3.2 along with the visual representation of the network architecture in figure 3.6.



Optimizer	Learning rate	Loss function	Training Epochs	Batch size	Metric	Total no. of trainable parameters
Adam	Inverse time decay	Categorical cross-entropy	300	16	Accuracy	25,717,799

Table 3.2: CNN model hyper-parameter summary

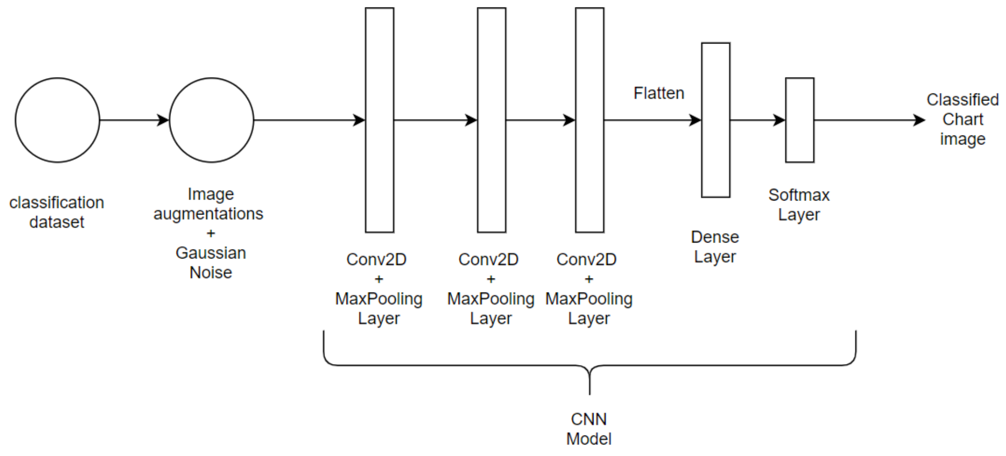

classification dataset
Image augmentations + Gaussian Noise
Conv2D + MaxPooling Layer
Conv2D + MaxPooling Layer
Conv2D + MaxPooling Layer
Flatten
Dense Layer
Softmax Layer
Classified Chart image
CNN Model

Figure 3.6: Architecture of baseline CNN model

3.2.3 Transfer learning model

The transfer learning model used for classifying the chart images is VGG16 [14], from the keras model library.

Initially, the mobilenetV2 model from the keras model library was considered [12]. Since VGG16 produced 7 percent better performance on the validation set, it was used in this project. The VGG16 is a CNN based model pre-trained on the imagenet dataset, which consists of 1000 classes of 1.3 million synthetic images in the training set [4]. As imagenet contains wide variety of classes and the large number of images, it allows the models trained on it to have wide range of image feature extraction capabilities which makes the model suitable to solve wide variety of problems, including chart image classification.

Excluding the default top layer, the VGG16 model has a total of 19 layers. As deep learning models go from bottom to top, the top layer is the classification layer of the model before the output is produced. By using a custom classification layer, the feature extraction can be optimized for our specific problem of classifying the chart images. Here, I created a custom top layer with 4 layers. These 4 layers are the input layer from the model, the global average pooling layer, the dense layer and the output softmax layer

For the first 50 epochs, I trained the model on the top layer with the default learning rate of 0.001 and froze the rest of the layers. For the next 50 epochs, I fine-tuned the model to improve its performance. For this, I unfroze 6 layers from the base model and trained it along with the custom top layer, making a total of 10 trainable layers. For this, the learning rate was set to 0.0001 (1/10th of the default learning rate) which should allow the model to learn and update at a much slower rate and thereby learning better feature representations. Due to memory constraints, a batch size of 16 was used. The VGG16 model was trained for a total of 100 epochs using the adam optimizer and the imagenet weights. Since I perform a multi-class classification, categorical cross-entropy loss is used. A summary of the hyper-parameter settings is provided in table 3.3 along with the visual representation of the network architecture in figure 3.7.

Optimizer	Learning rate	Loss function	Training epochs	Batch size	Weights	Metric	Total no. of trainable layers
Adam	0.001	Categorical cross-entropy	50 (1 to 50)	16	imagenet	Accuracy	4
Adam	0.0001	Categorical cross-entropy	50 (51 to 100)	16	imagenet	Accuracy	10

Table 3.3: Transfer learning model hyper-parameter summary

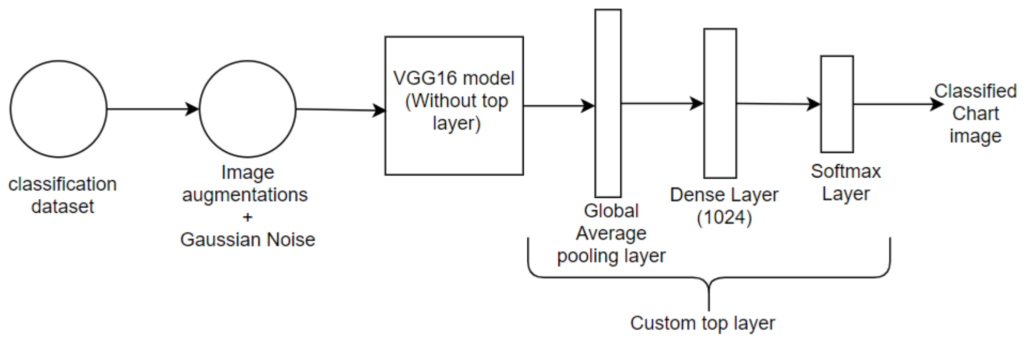

classification dataset
Image augmentations + Gaussian Noise
VGG16 model (Without top layer)
Global Average pooling layer
Dense Layer (1024)
Softmax Layer
Classified Chart image
Custom top layer

Figure 3.7: Architecture of transfer learning model

3.3 Chart object recognition

3.3.1 Data augmentation

Data augmentation in object recognition is essential to increase the the number of data points the model can train on. It is difficult to make the same augmentations with similar effect to the classification models because of the use of a different implementation framework in pytorch. Therefore, the data augmentations that could improve the model performance have been made.

If the augmentations change the size, orientation or angle of the image, it is essential to re-position the annotated bounding boxes of each object corresponding to the augmentation. Here, the images are resized to 800x800 as this size was found to work well on empirical evaluation. For this, detecto can automatically re-size the bounding boxes.

Since variation in zoom range and shear angle are applied randomly on the image by pytorch, it is difficult to make corresponding augmentations to the annotated bounding boxes. If the bounding box co-ordinates are not accurately re-positioned, then the model trains on wrong bounding box co-ordinates which negatively impacts the performance of the model. Therefore, these 2 augmentations have not been used for the object recognition model.

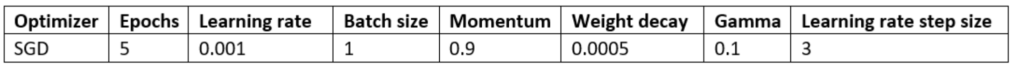
The brightness of the images has been set to vary by 20 percent increase or decrease to the original image. The gaussian noise injection process is similar to the classification model.

These data augmentations have been applied only on the training set. The images in the validation set have been kept free of any data augmentation to resemble the real-world data as close as possible. Both the training and validation sets have been normalised and converted to tensor format to input to the pre-trained model.

3.3.2 Pre-trained object recognition model

The detecto package is built around the pre-trained faster r-cnn model with a resnet50 feature pyramid network (FPN) backbone from the pytorch model zoo. The faster rcnn model provides a good balance between speed and accuracy as it requires very less time to detect objects. According to Lin et al [9], the resnet50 fpn backbone for the faster r-cnn improves the object recognition accuracy by approximately 7 percent when compared to the resnet50 region proposal network (RPN) backbone. During testing, the confidence score threshold has been set at 0.6 (60 percent) to avoid false positive detection being shown to the user.

The model was trained for 5 epochs. As each epoch took approximately 30 minutes to complete, 5 epochs gave the balance between training speed and minimising the validation loss. Due to memory and training time constraints, the batch size of 1 was used. The learning rate of 0.001 was found to work well upon empirical evaluation. Other model parameters such as momentum, weight decay, gamma and learning rate step size have been set to default values, while detecto internally compiles the model using a stochastic gradient descent (SGD) optimizer. Table 3.4 shows the summary of the chosen and default hyper-parameter values.



Optimizer	Epochs	Learning rate	Batch size	Momentum	Weight decay	Gamma	Learning rate step size
SGD	5	0.001	1	0.9	0.0005	0.1	3

Table 3.4: Object recognition hyper-parameter summary

Chapter 4

Evaluation and result

In this chapter, the designed models are evaluated and the result of these evaluations are discussed.

4.1 Evaluation strategy

The classification of chart images is evaluated in terms of its accuracy, as we are trying quantify how accurately the model can classify the input chart image. For this, the time-series graph of the model performance over the epochs in the training and validation sets is used. The per-class performance of the model is explained using the confusion matrix.

The object recognition is evaluated using the mean average precision metric (mAP) for intersection over union (IoU) greater than 0.5, as we are trying to quantify how accurately the model can recognize the objects of interest in the input image. For this, average precision (AP) is calculated which gives the average of precision values for the recall between 0 and 1 for each object of interest. Then, the mAP, which is the mean of the AP across all objects of interest is computed for IoU greater than 0.5. IoU is the measure of overlap between the predicted bounding box and the ground truth bounding box. This is set to be greater than 0.5 to avoid false positive predictions being accounted in the calculation. The statistics for the object size is also provided in order to put the obtained results in perspective. Additionally, the results from the implemented faster-rcnn resnet 50-fpn model is compared with the same model trained with common objects in context (COCO) dataset to give further insights on the performance of the implemented model.

The performance of these models can be affected by the data augmentations and the actual features of the images itself. In order to understand the performance of the models, I performed 2 ablation studies:

1. Remove all data augmentations and gaussian noise
2. Remove gaussian noise but keep all other data augmentations

4.2 Classification results

4.2.1 Baseline CNN model

The baseline CNN model produces a training accuracy of 99.6 percent and a validation accuracy of 74.2 percent after 300 epochs as shown in figure 4.1.

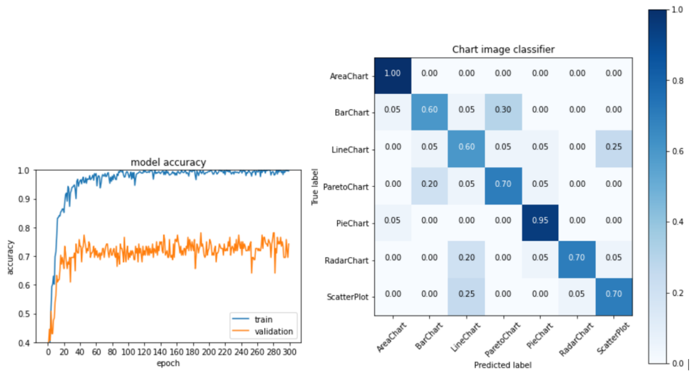
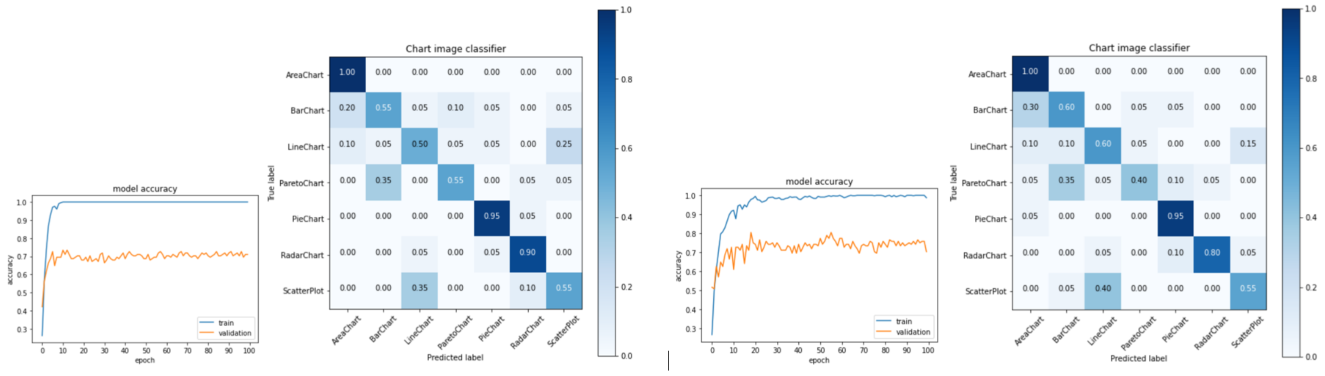


Figure 4.1: Performance of baseline CNN model

Taking a closer look at the confusion matrix reveals the per-class performance of the model on the validation set. Upon closer investigation we can see that the area chart and pie chart have better performance than the other classes of charts while bar chart and line chart have the lowest performance of all the chart classes.

Ablation study

First, I performed ablation study 1 and re-trained the model for 100 epochs. As shown in figure 4.2a, the model overfits after 10 epochs during training. As a result the validation accuracy drops to 67.1 percent making it clear that the model has learned less feature representation compared to the implemented baseline cnn model. As a result of overfitting, the class-wise accuracy from the confusion matrix cannot provide a reasonable inference to compare with the actual model results. Next, I performed ablation study 2 and re-trained the model for 100 epochs. Compared to abla-



a) Remove data augmentation and gaussian noise

b) Keep data augmentation, remove gaussian noise

Figure 4.2: Baseline CNN model ablation study

tion study 1, the training accuracy has increased to 98.7 percent and the validation accuracy has increased to 70.3 percent as shown in figure 4.2b. It is clear that introducing data augmentation has helped to regularise the model, making it robust to learn the appropriate feature representations of all the types of chart images. The introduction of gaussian noise in the implemented baseline cnn model has further regularised the model and made it robust. This has helped the model to learn better feature representations as evidenced by the increase in validation accuracy of the implemented baseline cnn model.

By taking a closer look at the confusion matrix of the implemented baseline model and the ablation study models, a pattern emerges as the bar chart and line chart seems to have the most influence on all the other charts. One possible reason for this could be that some scatter plot images misclassified as line chart are due to the presence of a regression line in them. Likewise, the pareto chart contains both bars and lines to represent data, meaning that both bar chart and line chart will impact the classification of pareto chart. To validate this assumption, I removed the bar chart and line chart images while keeping all the data augmentations including the gaussian noise intact from the implemented baseline model. I re-trained the model for 100 epochs with the same settings as the implemented baseline model. As seen in figure 4.3, there is a significant improvement in the overall model performance as evidenced by the increase in validation accuracy to 91.1 percent. This confirms the earlier assumption that bar chart and line chart have the most influence on the performance of other chart types.

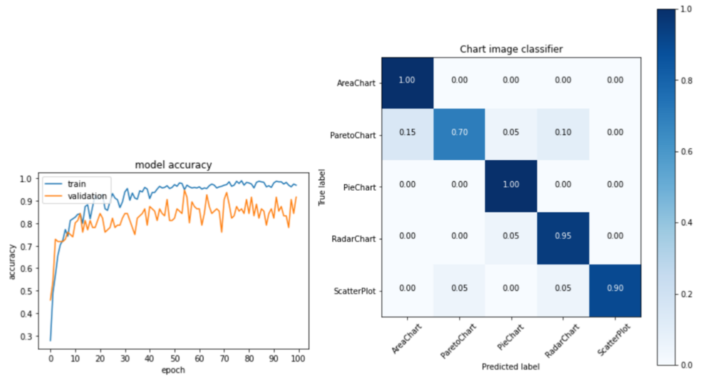


Figure 4.3: Model without bar chart and line chart

By performing the ablation study, we can understand that the data augmentations and the gaussian noise have helped regularise the model by making it robust and thereby improving the overall performance. While the per-class performance has improved, there is no significant increase or decrease in the performance of each class, meaning that the per-class performance is due to the nature of the features extracted from the chart images and is not significantly affected by the data augmentations and gaussian noise.

4.2.2 Transfer learning model

The transfer learning model after 100 epochs, including fine tuning, produces a training accuracy of 97.4 percent and a validation accuracy of is 96.8 percent. As shown in figure 4.4, the close performance of the model on both training and validation accuracy combined with a strong per-class accuracy in the confusion matrix suggests that the model has better understanding of the chart feature representation compared to the baseline cnn model.

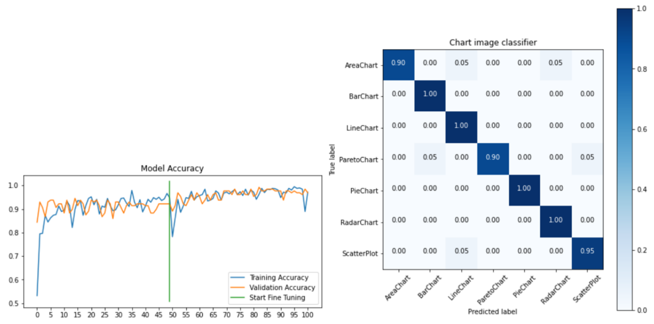


Figure 4.4: Transfer learning model performance

Ablation study

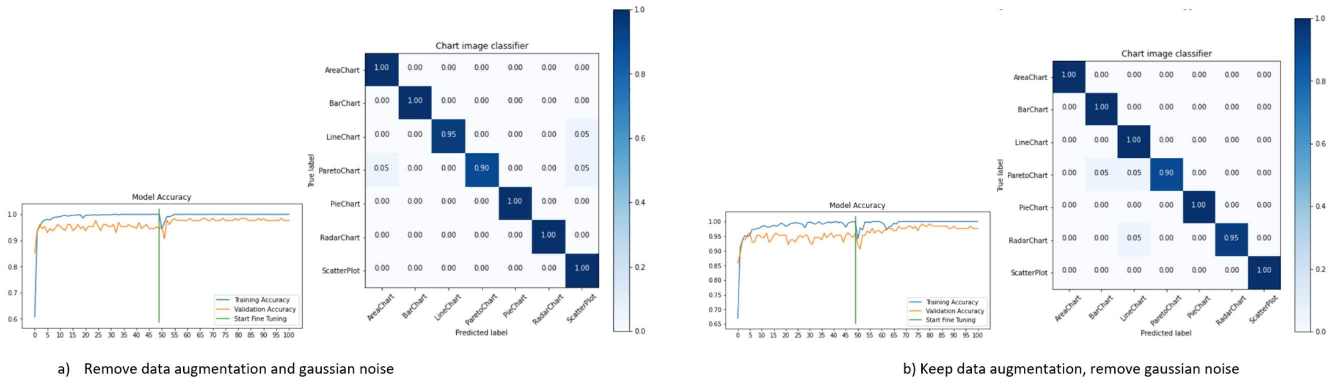


Figure 4.5: Transfer learning model ablation study

First, I performed ablation study 1 and re-trained the model for 100 epochs. As shown in figure 4.5a, the model overfits almost immediately after fine tuning.

Then, I performed ablation study 2 and re-trained the model for 100 epochs. As shown in figure 4.5b, the model overfits 20 epochs after fine tuning. However, despite both these models overfitting, the per-class accuracy is very similar to the implemented transfer learning model.

By this ablation study, we can understand that the applied data augmentations and gaussian noise has helped regularise the model and make it robust. However, it has negligible effect on the per-class accuracy suggesting a strong understanding of the chart feature representation of all the chart types by the model.

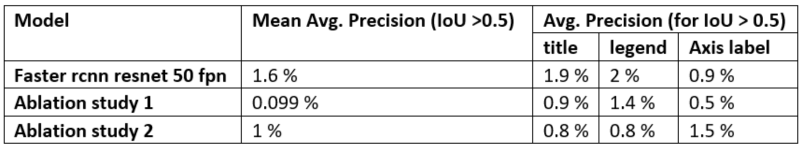
From this evaluation, we can see that the transfer learning model has better overall and per-class performance than the baseline CNN model, suggesting a good understanding of chart image features. This means that the transfer learning model can classify the chart images accurately and consistently across all chart types than the baseline CNN model, for which the performance is affected by the bar chart and line chart features.

4.3 Object recognition results

The model predicts the labels, the confidence score of the labels and their corresponding bounding box co-ordinates (xmin, ymin, xmax and ymax).

The labels are output as a nested list where each inner list corresponds to the predicted labels in an image. The predicted confidence score and bounding box co-ordinates are output as a list of tensors, where each tensor corresponds to an image. The nested list of labels are converted to a list of tensors by replacing the label names with custom label maps. The label maps are 1 for title, 2 for legend, 3 for axis label and 0 for background.

The xml file for each image contains the ground truth labels and the corresponding bounding boxes. This is read using the xml element tree package and converted to a list of tensors. Objects are marked as difficult only when it is not recognizable without substantial context, where 0 is for objects that are not difficult to detect and 1 is for objects that are difficult to detect. I do not include difficulty of the objects in the current implementation and therefore the difficulty for all the objects are marked as 0. However, the mAP calculation includes the difficulties into the computation to support the future possibility of including the difficulty flag in the ground truth annotations.

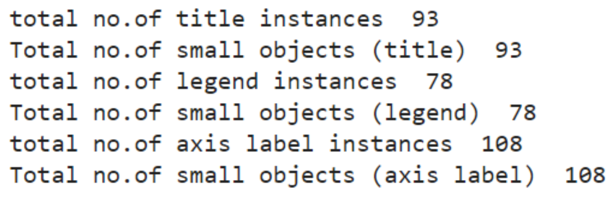


Model	**Mean Avg. Precision (IoU >0.5)**	**Avg. Precision (for IoU > 0.5)**		
		title	**legend**	**Axis label**
Faster rcnn resnet 50 fpn	1.6 %	1.9 %	2 %	0.9 %
Ablation study 1	0.099 %	0.9 %	1.4 %	0.5 %
Ablation study 2	1 %	0.8 %	0.8 %	1.5 %

Table 4.1: Object detection results

The model produces an mAP of 1.6 percent for IoU greater than 0.5. As shown in table 4.1, we understand from the ablation study that by introducing gaussian noise into the dataset, the mAP has improved while the average precision (AP) of the axis label has decreased. Gaussian noise has helped to make the model robust as evidenced by the 0.6 percent increase in mAP between the ablation study 2 (remove noise but keep all other augmentations) and the implemented model

To investigate this result further, taking a look at object size statistics of the ground truth annotations in figure 4.6, reveals all of the instances of our objects are small-objects, which I have defined to be 20 percent or less than the size of actual image.



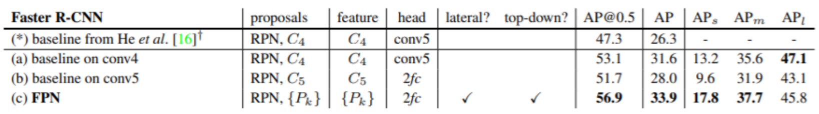
```
total no.of title instances  93
Total no.of small objects (title)  93
total no.of legend instances  78
Total no.of small objects (legend)  78
total no.of axis label instances  108
Total no.of small objects (axis label)  108
```

Figure 4.6: Number of small objects in the dataset

However, COCO dataset defines small-object size to be less than 32x32 pixels irrespective of the actual image size [10]. But, Chen et al [2] states that many small-object instances in COCO dataset occupy a large part of the image, meaning that the actual image is either equal to or slightly bigger than the defined 32x32 pixels. They go on to create a custom small-object dataset, in which the relative median size of the small-object vary between 0.08 to 0.58 percent of the original image for each class. From this, we can understand the issue in defining the size of the small-object as a set resolution. Combined with the novel problem of detecting objects in a chart image rather than the objects in COCO dataset (car, bike etc) means the above definitions of a small-object cannot be applied to our problem as it is.

Therefore, considering the definition of small-object size range (0.08 to 0.58 percent) by Chen et al [2] and the low small-object size threshold (32x32 pixels) of the COCO dataset, I have defined all the objects with bounding box size of less than 20 percent of the original image as small-objects.

From this, it is fair to say that the performance of our model suffers from the small-object detection problem. To put it in context, lets take a look at the small-object detection performance of the faster rcnn resent 50 fpn model from Lin et al [9], trained on the COCO trainval35k dataset, which has 35,000 image instances [10].



Faster R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP$_s$	AP$_m$	AP$_l$
(*) baseline from He *et al.* [16]†	RPN, C_4	C_4	conv5			47.3	26.3	-	-	-
(a) baseline on conv4	RPN, C_4	C_4	conv5			53.1	31.6	13.2	35.6	**47.1**
(b) baseline on conv5	RPN, C_5	C_5	*2fc*			51.7	28.0	9.6	31.9	43.1
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	*2fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	45.8

Figure 4.7: Faster rcnn resnet50 fpn on COCO dataset [9]

As shown in the figure 4.7, the AP for small-objects (APs) using FPN is 17.8 percent. In comparison, our model has been trained with a training set with 560 instances, which reasons the difference in the object detection performance. Though we have defined small-objects differently to the COCO dataset and the object recognition task of detecting chart objects is different to the objects in COCO dataset, we can see that even with a much larger dataset, the faster r-cnn resnet 50-fpn model was only able to achieve 17.8 percent performance. Albeit not being an exact equivalent comparison, this helps to establish, understand and explain the small-object detection problem that affects the model.

Chapter 5

Conclusion

In this chapter, a discussion of the project with regards to the objectives achieved, the recommendations for the future work and the personal reflection on the overall project period is presented.

5.1 Discussion

This project was set out to create a chart image classification and object recognition pipeline which would fix the major pitfalls and provide better context of the results than the existing systems. For this, a diverse dataset that included 3 sources of chart images was created. Next, the baseline cnn model was established and the transfer learning model was created, for both of which ablation study was performed to understand the performance of the models. It was found that the transfer learning model had better performance than the baseline model on the given dataset. Then, object recognition was carried using the pre-trained faster-rcnn model with a resnet50 fpn backbone. The result of this model was analysed for all the obtained predictions and presented with object size statistics and compared with COCO dataset trained model to provide context to the result. It was found that the model suffers from small-object detection problem, which also affects the model trained on the COCO dataset, giving insight into the obtained result.

However, the size of the dataset is a potential pitfall of the project. Due to time and resource constraints, the decision was made to use 700 images which has 100 images per chart type. In comparison, the ReVision dataset has about 2500 chart images and ChartSense has about 7000 images. The small size of the created dataset means that the evaluation was performed using 140 images. If it was performed using more samples, the results would have been far more convincing and representative of the actual model performance.

An another limitation is the small-object detection problem that affects the object recognition model performance. This is an active research area in object recognition field. There are ways such as focal loss and image tiling that could potentially improve the performance. However, since this issue was discovered at the end of the project development cycle, due to time constraints I was not able to apply these techniques to improve the small-object detection performance.

5.2 Future work

One possible future work could be to expand the dataset size by including more number of images. A large dataset can provide a better representation of the actual model performance.

The techniques such as focal loss and image tiling can be used to improve the small-object recognition performance. Using focal loss will punish the network less for misclassifying the classes that it detects well and more for misclassifying the classes that it does not detect well. The weights are also updated to aid the network to recognize the difficult classes. Likewise, by splitting the image into tiles and using the tiles for object detection can also improve the small-object recognition performance.

Furthermore, I was able to run some tests with open-source package tesseract for the data extraction process. Unfortunately, the open source technology is not robust enough to extract the data points in a meaningful way from the chart image. On such example would be the canny edge detection implementation in tesseract. When applying it on a 300x300 pie chart image with 3 slices, the algorithm was able to detect the edges of only 2 slices of the pie chart. This means while extracting the data, the 3rd slice will be completely ignored which leads to inaccurate data being extracted. Likewise, the text within the slice of the pie chart was completely ignored making the object character recognition (OCR) inaccurate as well. For this, a data extraction pipeline can be built using custom software to improve the various limitations of the current open-source implementations.

5.3 Personal reflection

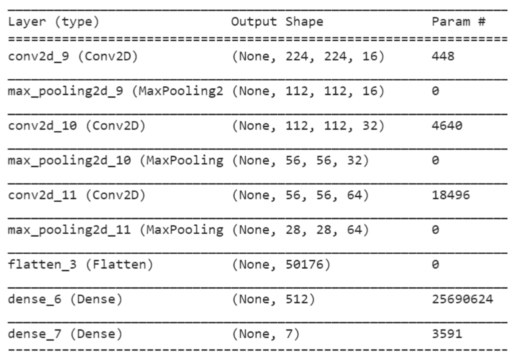
This project has helped me to expand my knowledge and develop a self-starter leadership attitude. I believe that I now have better knowledge of the deep learning frameworks keras and pytorch, to which I intend to contribute more as an open-source developer. I got to build a dataset from scratch by setting appropriate constraints, which I thoroughly enjoyed. I now have a keen understanding of the data pre-processing methods and the effects that it could have on the deep learning model. I learned new techniques such as ablation study and understood the issue of small-object detection, which I intend to explore further.

The past three months has been one of the most unique phases in my life. I went through a roller-coaster of emotions from experiencing joy, learning, questioning and study. The unforeseen circumstance of a global pandemic has helped me gain new perspective in life and develop appreciation for the kindness in people. I hope to take this realization and the experience gained from this project along to my future endeavors.

Appendix A

Network architecture

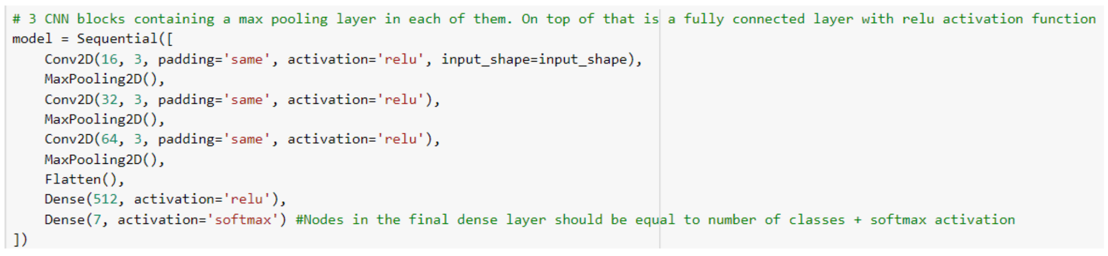
A.1 Baseline CNN network



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 224, 224, 16)      448
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 112, 112, 32)      4640
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 56, 56, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 28, 28, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               25690624
_________________________________________________________________
dense_7 (Dense)              (None, 7)                 3591
-----------------------------------------------------------------
```

Figure A.1: CNN Model summary

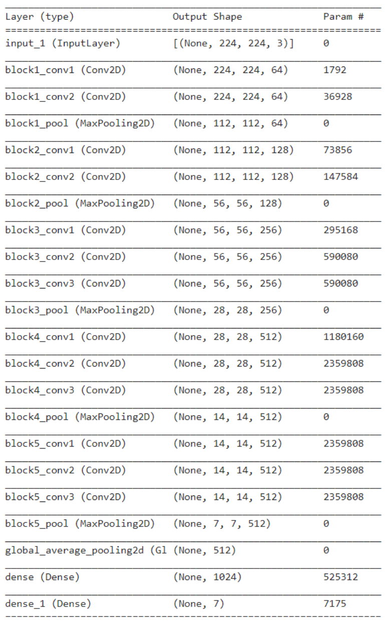
A.2 Baseline CNN model implementation in Keras



```python
# 3 CNN blocks containing a max pooling layer in each of them. On top of that is a fully connected layer with relu activation function
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu', input_shape=input_shape),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(7, activation='softmax') #Nodes in the final dense layer should be equal to number of classes + softmax activation
])
```

Figure A.2: CNN code block

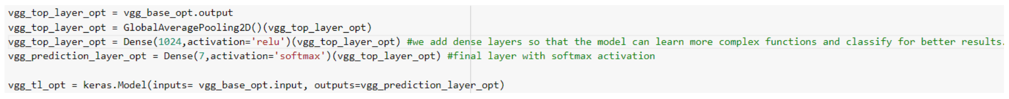
A.3 Transfer learning network



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 1024)              525312
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 7175
-----------------------------------------------------------------
```

Figure A.3: Transfer learning model summary

A.4 Custom top layer implementation in keras



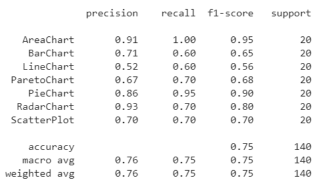
```
vgg_top_layer_opt = vgg_base_opt.output
vgg_top_layer_opt = GlobalAveragePooling2D()(vgg_top_layer_opt)
vgg_top_layer_opt = Dense(1024,activation='relu')(vgg_top_layer_opt) #we add dense layers so that the model can learn more complex functions and classify for better results.
vgg_prediction_layer_opt = Dense(7,activation='softmax')(vgg_top_layer_opt) #final layer with softmax activation

vgg_tl_opt = keras.Model(inputs= vgg_base_opt.input, outputs=vgg_prediction_layer_opt)
```

Figure A.4: Transfer learning code block

Appendix B

Model results

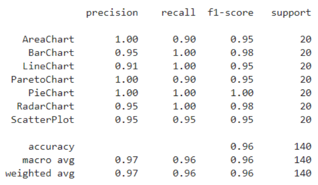
B.1 CNN classification report



```
              precision    recall  f1-score   support

   AreaChart       0.91      1.00      0.95        20
    BarChart       0.71      0.60      0.65        20
   LineChart       0.52      0.60      0.56        20
 ParetoChart       0.67      0.70      0.68        20
    PieChart       0.86      0.95      0.90        20
  RadarChart       0.93      0.70      0.80        20
 ScatterPlot       0.70      0.70      0.70        20

    accuracy                           0.75       140
   macro avg       0.76      0.75      0.75       140
weighted avg       0.76      0.75      0.75       140
```

Figure B.1: Baseline CNN model classification report

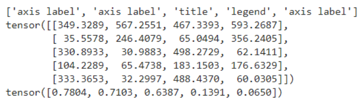
B.2 Transfer learning classification report



```
              precision    recall  f1-score   support

   AreaChart       1.00      0.90      0.95        20
    BarChart       0.95      1.00      0.98        20
   LineChart       0.91      1.00      0.95        20
 ParetoChart       1.00      0.90      0.95        20
    PieChart       1.00      1.00      1.00        20
  RadarChart       0.95      1.00      0.98        20
 ScatterPlot       0.95      0.95      0.95        20

    accuracy                           0.96       140
   macro avg       0.97      0.96      0.96       140
weighted avg       0.97      0.96      0.96       140
```

Figure B.2: Transfer learning classification report

B.3 Object recognition result sample



```
['axis label', 'axis label', 'title', 'legend', 'axis label']
tensor([[349.3289, 567.2551, 467.3393, 593.2687],
        [ 35.5578, 246.4079,  65.0494, 356.2405],
        [330.8933,  30.9883, 498.2729,  62.1411],
        [104.2289,  65.4738, 183.1503, 176.6329],
        [333.3653,  32.2997, 488.4370,  60.0305]])
tensor([0.7804, 0.7103, 0.6387, 0.1391, 0.0650])
```

Figure B.3: Object recognition prediction for 1 image

Bibliography

[1] Abhijit Balaji, Thuvaarakkesh Ramanathan, and Venkateshwarlu Sonathi. Chart-text: A fully automated chart image descriptor. *arXiv preprint arXiv:1812.10636*, 2018.

[2] Chenyi Chen, Ming-Yu Liu, Oncel Tuzel, and Jianxiong Xiao. R-cnn for small object detection. In *Asian conference on computer vision*, pages 214–230. Springer, 2016.

[3] Chollet, Francois, et al. Keras. `https://keras.io`, 2015.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[5] Google. Google colaboratory. `https://colab.research.google.com`, last accessed on 16/09/20.

[6] Bulk image downloader team. Bulk image downloader software. `https://bulkimagedownloader.com/`, last accessed on 16/09/20.

[7] Daekyoung Jung, Wonjae Kim, Hyunjoo Song, Jeong-in Hwang, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. Chartsense: Interactive data extraction from chart images. In *Proceedings of the 2017 chi conference on human factors in computing systems*, pages 6706–6717, 2017.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[10] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[11] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[12] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[13] 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, 2011.

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[15] Detecto team. Detecto. `https://github.com/alankbi/detecto`, last accessed on 15/09/20.

[16] Pytorch team. Pytorch library. `https://github.com/pytorch/pytorch`, last accessed on 15/09/20.

[17] Tzutalin. Labelimg, 2015. `https://github.com/tzutalin/labelImg`, last accessed on 16/09/20.