Transfer Learning for Faster Region Based Convolutional Neural Networks for Real-time Traffic Classification

Rajinder Singh

M.Sc. in Computing
in Big Data Analytics
2021

Computing Department, Letterkenny Institute of Technology, Port Road, Letterkenny, Co. Donegal, Ireland.

Transfer Learning for Faster Recurrent Convolutional Neural Networks for Real-time Traffic Classification

Author: Rajinder Singh

Supervised by: Dr. Shagufta Henna

A thesis submitted in partial fulfilment of the requirements for the

Master of Science in Computing in Big Data and Analytics

Submitted to Quality and Qualifications Ireland (QQI)
Dearbhú Cáilíochta agus Cáilíochtaí Éireann

September 2021

Declaration

I hereby certify that the material, which l now submit for assessment on the programmes of study leading to the award of Master of Science in Computing in XXXXX, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution. I understand that it is my responsibility to ensure that I have adhered to LYIT's rules and regulations.

I hereby certify that the material on which I have relied on for the purpose of my assessment is not deemed as personal data under the GDPR Regulations. Personal data is any data from living people that can be identified. Any personal data used for the purpose of my assessment has been pseudonymised and the data set and identifiers are not held by LYIT. Alternatively, personal data has been anonymised in line with the Data Protection Commissioners Guidelines on Anonymisation.

I consent that my work will be held for the purposes of education assistance to future students and will be shared on the LYIT Computing website (www.lyitcomputing.com) and Research THEA website (https://research.thea.ie/). I understand that documents once uploaded onto the website can be viewed throughout the world and not just in the Ireland. Consent can be withdrawn for the publishing of material online by emailing Thomas Dowling; Head of Department at thomas.dowling@lyit.ie to remove items from the LYIT Computing website and by email emailing Denise McCaul; Systems Librarian at denise.mccaul@lyit.ie to remove items from the Research THEA website. Material will continue to appear in printed formats once published and as websites are public medium, LYIT cannot guarantee that the material has not been saved or downloaded.

Rajinder Singh

Signature of Candidate Date 31/08/2020

Acknowledgements

My sincerest thanks and appreciation go to my supervisor, Dr. Shagufta Henna, Department of Computing, L.Y.I.T for her sincere efforts to provide excellent guidance and knowledge at every milestone that helped me to find solutions to all the challenges during the dissertation. Her constant efforts to keep me on track with the schedule helped me to submit the dissertation in the allotted time. I would also like to appreciate the support from my family, friends, and colleagues for motivating me throughout my journey.

Abstract

Traffic sign detection is a hotspot of research and development by the onset of semi-automatic and fully automatic cars in the market. However, the low visibility scenarios due to bad weather, reflections, and night time can cause a system failure in these cars. Improving traffic sign detection system vision in real-time can reduce accidents and also regulates drivers to follow traffic rules. FasterRCNN in traffic sign detection has impressive performance in regular conditions however, the performance degrades with objects which are deformed, smaller in size, low visibility, and have obstructions. It requires better training of the model with versatile and large datasets. To address this problem, the thesis proposes an improvement over FasterRCNN. The proposed approach utilizes the benefits of transfer learning for traffic sign detection coupled with FasterRCNN named T-FasterRCNN for real-time traffic classification. T-FasterRCNN helps in distinguishing different classes of traffic signs by implementing image augmentation to simulate the bad weather conditions. The proposed approach demonstrates traffic sign detection using Gaussian blur with three different levels 5%, 10%, and 15%. In contrast to FasterRCNN, data subsets are passed through a Resnet-50 pre-trained model for better accuracy, precision, and real-time inferences. Experiment results reveal an improved accuracy of T-FasterRCNN with a value of 0.95 mAP which is more than the FasterRCNN and RCNN. The proposed approach can be applied by automobile companies to improve their computer vision by enhancing their cameras and tweaking autonomous driving systems.

Acronyms

Acronym	Definition	Page
RCNN	Region Based Convolutional Neural Network	1
YOLO	You Only Look Once	1
DL	Deep Learning	2
TSR	Traffic sign recognition	2
HOG	Histogram of oriented gradients	3
VGG-16	Visual Geometry Group-16	3
GTSRB	German Traffic Sign Recognition Benchmarking	5
DPM	Deformable Part-based Model	5
SPPNet	Spatial Pyramid Pooling Networks	9
SURF	Speeded Up Robust Features	9
SIFT	Scale Invariant Feature Transform	9
GPU	Graphical Processing Unit	9
GAN	Generative Adversarial Networks	9
CIFAR	Canadian Institute of Advanced Research	11
DNN	Deep Neural Network	12
WYDOT	Wyoming Department of Transport	13
BCT	Bilateral Chinese transform	13
VBT	Vertex bisector transform	15
AFF	Attention Feature Fusion	15
GPGPU	General Purpose Graphical User Interface	15
ECM	Electric Chromatic Mirror	17
ACF	Aggregated Channel feature	18

LCD	Liquid Crystal Display	18
KNN	K-nearest Neighbours	19
SCNN	System for Cellular Neural Network	19
NVD	Night Vision Detector	20
RPN	Regional Purpose Network	22
ROI	Region of Interest	29
CSV	Comma Separated Value	29
T-FasterRCNN	Transfer Learning for Region Based Convolutional Neural Network	33

Table of Contents

Declaration **3**

Acknowledgements **4**

Abstract **5**

Acronyms **6**

Table of Contents **8**

Table of Figures **10**

Table of Code Listings **11**

Chapter 1 **1**

Introduction **1**

1.1 Purpose and Research Questions 5

1.2 Approach and Methodology 5

1.3 Contribution 6

1.4 Scope and limitations 7

1.5 Outline 7

Chapter 2 **8**

Literature Review 8

2.1 Object Detection Algorithms 8

2.2 Related Works 9

Chapter 3 **23**

Transfer Learning for Faster Recurrent Convolutional Neural Networks for Real-time Traffic Classification 23

3.1 Design 23

3.2 Datasets 24

3.3 T-FasterRCNN for Traffic Sign Detection 25

3.3.1 Data refinement and multiscale feature mapping using Resnet-50 25

Algorithm 1: Resnet-50 feature extraction 25

3.3.2 T-Faster RCNN Model 27

Algorithm 2 T-Faster RCNN Model: 27

Algorithm 3: Bounded Box Regressor 29

Chapter 4 **31**

Implementation and Results 31

4.1 System Requirements 31

4.1.1 Hardware Requirements 31

4.1.2 Software Requirements 31

4.2 Experiments 33

4.3 Performance Metrics 35

4.3.1 Evaluation Criteria 35

4.4. Analysis of Results 37

4.4.1 Performance of T-faster RCNN Resnet-50 37

4.4.2 Training and Testing time of the T-FasterRCNN-Resnet50 39

4.4.3 Prediction Accuracy of CNN, FasterRCNN, and T-FasterRCNN_RestNet50 40

4.4.4 Effect of Blurriness and Shift levels on the performance of T-FasterRCNN_RestNet50 42

Chapter 5 **46**

Conclusion and Future Works **46**

5.1 Conclusion 46

5.2 Future Work 47

References **48**

Appendix A: Code Listing **53**

Table of Figures

Figure 3.1 T-FasterRCNN stages of image processing	24
Figure 4.1. Correlation matrix for training dataset	35
Figure 4.2. Average height distribution of training images	36
Figure 4.3. Average width distribution of training images.	36
Figure 4.4 Accuracy of T-FasterRCNN as compared to RCNN and Fast RCNN	39
Figure 4.5 Loss of T-fasterRCNN as compared to RCNN and Fast RCNN	40
Figure 4.6 Training time T-FasterRCNN as compared RCNN and Fast RCNN	41
Figure 4.7 Testing time of RCNN,FastRCNN, and T-FasterRCNN	41
Figure 4.8 Accuracy of RCNN,FastRCNN, and T-FasterRCNN	42
Figure 4.9 F1 score of RCNN, FastRCNN, and T-FasterRCNN	43
Figure 4.10 MSE among the different models for testing data	43
Figure 4.11 Accuracy among different blur levels	45
Figure 4.12 F1 Score among the different blur levels	45
Figure 4.13 Precision among different blur levels	46
Figure 4.14 MSE among different blur levels	46

Table of Code Listings

Code Listing 1 Imported Libraries 54
Code Listing 2 DatasetDownloading 54
Code Listing 3 FasterRCNN Model 55
Code Listing 4 RCNN Model 55
Code Listing 5 Image Augmentation 56
Code Listing 6 T-FasterRCNN Model 56
Code Listing 7 Training Results 57
Code Listing 8 Testing Results 57
Code Listing 9 Training PLots 57
Code Listing 10 Testing PLots 58

Chapter 1

Introduction

Since the onset of computer vision, object detection has been a hot topic among researchers and engineers. It is the most fundamental and complicated problem of the time. The past two decades (Zou et al. 2019) are considered the golden period for the development of the object detection discipline. The aim is to develop computational models that provide basic information about the object: What is the object and where it is? It gives the type and position of the object as an output. The domain finds its application (Pathak et al. 2018) in real-world like self-driving cars, path detectors, face detectors, robot vision, and many more. Before 2014, object detection generally followed the traditional approaches based on handcrafted features such as Voila Jones Detectors (Tavallali et al. 2017). This detector was the first real-time human face detector without any constraints. Then other sophisticated approaches are introduced such as HOG detector and Deformable Part-based model (DPM). Around 2012, the world experienced the rebirth of convolutional neural networks with the introduction of deep learning neural networks which were fast, robust, and high-level feature representative. This development continues (Zou et al. 2019) from RCNN to FasterRCNN and now it is "You look only once" or (YOLO) algorithms. The adoption of transfer learning accelerates the growth of object detection as it eliminates the need for high resources and several days of training models. It provides pre-trained models that can retain the features from previous training and can further be fine-tuned to get desired results. This way it strengthened the real-time detection algorithms in domains like face detection and self-driving automobiles. Earlier these technologies were struggling to find their application in the real world.

A few decades ago, self-driving cars were a dream of the creators and inventors but the concept was limited to sci-fi movies. Today, we have Tesla cars (Ajitha and Nagra 2021) running on the roads, winning the heart and trust of people with every passing day. This becomes possible due to the sophisticated machine learning methods and advanced sensors that provide excellent sensitivity and speed to detect objects while driving. The main features in a self-driving car involve paths detection, road signs detections, traffic lights detection, text detections, and many more. Despite having success in the domain, self-driving cars still need to address issues (Gupta et al. 2021) in the Deep Learning (DL) models to improve their performance. The neural networks do not follow the concept of "if and else" programming. They are unable to mimic the intricate complexities of driving. The performance of these cars is tested only in the planned cities with properly built roads and perfect weather conditions. However, there is a possibility of false detection (Ng 2021) on rainy or hazy days. The low light conditions and off-road trips can degrade the system's performance. They also need to know how to act in unfamiliar situations that are socially accepted in context to that country.

Traffic sign recognition (TSR) has been one of the key areas of research for decades. Even when self-driving cars are running on the roads, a few incidents motivate researchers to work in the domain and develop an error-free framework. A 2019 Florida clash (Pearl 2019) due to malfunctioning of the Tesla’s driver assistance system. The car was unable to detect the stop sign and smashed into a parked vehicle and killed a 22-year college student. Such incidents motivate engineers to improve their systems. The deep neural network algorithm Faster RCNN is an efficient image detector that can act in real-time as well because of the faster detection. (Lee et al. 2016).

Other advanced algorithms such as YOLOv4 are also widespread in the domain but they require high-resolution input images and videos. It compromises the performance when dealing with low-resolution input images.

The researchers have been continuously working on object detection and as a result, self-driving cars are already on the roads and people are experiencing the challenges. It is high time to understand the existing methods and make some contribution to the domain. Many recent studies tried to work on the existing issues in detecting signs. The paper (Yang et al. 2016) proposed a method that uses a quick detection model that claimed to be 20 times faster than the existing techniques mentioned in the paper. It uses classification and extraction of proposals that are developed on a color probability model and HOG. It aims to train the model for real-time traffic sign detection. Also, the authors in (Palavanchu 2021) applied transfer learning on the best object detection algorithms and did the comparison. They picked Residual Networks, ResNet50, Xceptionnetwork, VGG-16, and InceptionV3 networks to showcase the impact of transfer learning on the existing techniques. Another interesting study by authors (Rezapour and Ksaibati 2021) proposed a novel method to identify the roadside barriers by using pre-trained models InceptionV3, denseNet121, and VGG 19. They removed the top two layers and froze the rest and shown considerable performance improvement. TS-Yolo, a model based on the YOLO concept in the study (Wan et al. 2021), is employed to acquire good accuracy and time even in low light and dark conditions. Another approach to address the same issue of low visibility is formulated by authors in (Dogancan et al. 2020). They proposed a dataset CURE-TSD rather than a model. The dataset is a real-world dataset with proper labeling for different low visibility scenarios. It also overcomes the shortcomings of the realistic dataset which is appropriate to demonstrate harsh weather conditions and low visibility scenarios. Another study in an attempt to contribute to low visibility issues in self-driving cars by (Xiao et al., 2020) shows the importance of illumination. It suggests the need to adjust the illumination of images to detect the markings on the roads in a more precise way at night. A study (Liu and Xiong 2020) used the YOLOv3 algorithm to detect very small objects in the input images and videos. It replaces the traditional 2D layer with a dilated convolutional layer that expands at the receiving end. It requires very high-resolution images that aim to detect multiple objects of different sizes (smaller and bigger) in the image. This makes the model more robust and fast.

Recently a work (Xiao et al. 2020) aims to improve the performance of FasterRCNN by introducing amendments at the three stages. They implement a context information model, skip pooling, and guided anchors RPN. This shows 6.875% mAP improvement when compared to the state-of-the-art models. This work highlights the limitations of FasterRCNN of being unable to detect the tedious and nonideal cases of object detection. The use cases where lighting is bad, objects are rotating, or deformed show poor performance with FasterRCNN. There is a problem with generating appropriate anchors in Faster RCNN. As the anchors are of fixed size thus anchoring methods unable to cover the entire area of the object. This leads to a low recall value and causes more number anchors. As per calculation, more anchors mean a higher recall value. This results in many negative samples in the fixed range of anchors and it degrades the performance. This thesis implemented T-FasterRCNN to overcome these issues by introducing transfer learning that uses a pre-trained model that is trained with a vast variety of datasets. Moreover, to further enhance the performance, it uses Resnet 50 mode that introduces 50 more layers to reduce the error percentage. Together this approach aims to detect objects better in low light and improve the accuracy score for traffic signs in bad weather.

1.1 Purpose and Research Questions

This thesis aims to formulate a real-time traffic signs detection system using deep learning convolutional networks with transfer learning to elevate the performance from the existing state-of-the-art object detection models in the domain. It will use a proposed model named transfer learning-Faster Region-Based Convolutional Neural Network or (T-Faster RCNN). The focus is to get better accuracy and fast execution even in the bad light and low visibility scenarios. The research will address the issues with the existing published work such as low accuracy in bad weather, improperly trained models, and real-time detection of the signs.

- Can transfer learning when applied to real-time traffic sign classification accelerate the prediction of FasterRCNN and RCNN?
- Can transfer learned FasterRCNN achieve improved performance in terms of accuracy and F1 score for real-time traffic sign classifications?
- Can image/traffic sign blurriness affect the performance of transfer learned FasterRCNN for real-time traffic sign classification?

1.2 Approach and Methodology

This thesis proposed the T-Faster RCNN model as an object detection system that is trained by the GTSRB and GTSDB datasets. The model is pre-trained with ResNet-50 (Rajpal et al. 2021) layers and fine-tuned with dataset labels in the last layers. This work will scrutinize and compare the performance of T-Faster RCNN with RCNN, Fast RCNN, and Faster RCNN based on evaluation metrics like precision, Recall value, accuracy, F1 Score, and average time execution per image. The results are showcased with graph plots and tables. The methodology for the proposed approach involves different stages as explained below.

After the detailed analysis of the published works in the domain, the final approach starts by choosing a dataset with enough records to train the model. T-FasterRCNN uses the GTSRB dataset as it has low resolution and a wide variety of traffic sign images. After that, images are sorted into training and testing datasets as per metadata files. The standard dimensions are also calculated from the images and will be used to resize the dataset accordingly. The next step after data refinement is multi-scale feature mapping using Resnet-50 architecture. It provides the scaled features with the shift and blur levels variants of the same dataset. The different levels of Gaussian blurring (Ibrahim et al. 2021) are used which are 5%, 10%, and 15% blurred images. The output of the previous module goes to Region of Interest (ROI) and Regional Purpose Network (RPN) that replaces the featured map with proposals. The returned proposals go to softmax and regressor (Malini et al. 2021) layers that provide the class id and coordinates of the proposed object. These values will be tested among actual values and compared with other traditional models. The results of the experiments are shared for each blur level and one with the sunny day scenario as well. The experiment uses 32 epochs each with a batch of 30 for better training models within a confined time.

1.3 Contribution

The thesis has the following contributions in the domain of Object detection networks and transfer learning.

- The thesis critically analyses different approaches of transfer learning for object detections algorithms with a primary focus on real-time detection of traffic signs under low visibility scenarios.
- The work proposes a transfer learning approach coupled with the FasterRCNN for real-time traffic sign classification.
-
- The performance evaluation in this work attempts to compare the prediction time, accuracy, and F1 score of T-FasterRCNN in contrast to FasterRCNN and RCNN.
- Further, the work also compares T-FasterRCNN with traditional machine learning approaches.
- It scrutinizes the performance of T-Faster RCNN by using image augmentation to increase the blurriness of the images to simulate the low visibility

1.4 Scope and limitations

Due to the limited time frame and resources, the thesis has the following constraints and assumptions during execution:

- GTSRB dataset is a low-resolution image dataset as per the requirement of the thesis.
- Due to the limitation of resources, the dataset is not very large thus the experiment implements hyper tuning of the dataset to achieve the level of accuracy and effectiveness for the model.
- The thesis does not include advanced object detection algorithms such as YOLO as they require high-definition images and higher processing power.
- The scope of the model training is to focus on traffic signs only. it does not exclude the unwanted objects in the given scenario.
- The trained model cannot detect paths, humans, dogs, or any other object other than the traffic signs as it requires a more generalized dataset and varied labels and classes.

1.5 Outline

The following Chapter 2 focuses on the critical analysis of some of the techniques already published in the domain of object detection and transfer learning. It showcases the approaches they use, their limitations, and their scope of improvement. In chapter 3, the methodology is explained in detail. The algorithms used in T-Faster RCNN give an insight into all the technical details required to understand the implementation. Chapter 4 discusses the results of the experiments by using different graphs and tables. It involves some data analysis and the answers to the research questions this thesis targets highlighted in section 1.3. Finally, chapter 5 summarizes the experiment and gives some insights into the possible improvements and scope of the T-Faster RCNN in the future.

Chapter 2

Literature Review

The chapter discusses the various published works in the area of real-time object detection and analyses the approaches used in the papers. Most of the research focuses on traffic sign detections that use state-of-the-art models and improve them by applying various architectures to elevate the performance. The aim is to highlight the existing issues in the object detection algorithm and find the solutions to these problems. Later employ these solutions in the proposed approach by this thesis and contribute to the domain.

2.1 Object Detection Algorithms

Traffic sign detection uses object detection algorithms that help in the manufacturing of self-driving cars. People are adopting these cars that motivate researchers to put efforts to make them more robust and reliable. There are many ongoing challenges such as creating maps for these self-driven cars, bad weather conditions, cyber-attacks, social behavior, and many more. Researchers not only improved the existing methods but also invented innovative technology to improve the performance of these cars. One of the key technologies is object detection. Object detection is recognizing any road sign, road markings, hurdles, animals, people, and other vehicles on the road and acting accordingly. The evolution of object detection is itself a long way from success.

Twenty years back, Paul Viola and Micheal Jones, developed a framework to detect human faces in real-time by using the sliding windows technique. In 2005, (Dalal and Triggs, 2005) invented the HOG Detector which works on blocks. HOG resize the images multiple times without changing the size of the detection window. Later in the 2010s, a new technology Deformable Part-based Model (DPM) was invented which used divide and conquer techniques to identify the objects in parts. But after 2010, the performance of object detection models reached a saturation point and there was no significant development in the domain. It motivates engineers and researchers to bring reform in the area and make self-driving cars a reality by beginning the era of deep learning.

Recent advancements with convolutional neural networks, the deadlocks of object detection are broken. The technologies like RCNN, SPPNet, Faster RCNN, and YOLO provide better and improved performance in terms of accuracy, time, and precision.

2.2 Related Works

This section will discuss some recently published works in context with traffic sign detection and will scrutinize the approaches they adopted. It will also highlight the scope of improvement by adopting a sophisticated approach.

The authors in (H and James 2019) proposed a method to detect traffic signs in bad weather conditions that uses a capsule neural network that aims to reduce manual efforts. This approach is different from other convolutional models and it provides high efficiency in detecting the traffic signs. The paper formulates a framework for recognizing and detecting the traffic signs using color-based segmentation and performs detection by Hough transform. It is followed by the classification using the capsule neural network as discussed earlier. The images are trained using an Indian traffic dataset with a test dataset of 10350 images. The proposed method shows 15% percent higher accuracy as compared to CNN and 20-30% higher than other networks like SURF, SIFT (Kaun and Mittal 2021), and the BRISK classifiers. However, the comparisons with other convolutional models are limited to CNN. The experiment does not showcase the processing time per image for the proposed method.

Faster RCNN and Yolo are the real-time object detection models that are crucial in traffic sign detection.

A work in (Cao et al. 2019) experimented with the LeNet-5 CNN model by utilizing Gabor Kernel as the first convolutional layer. Later, they performed batch normalization right after the pooling layer. They also used the Adam method as an optimizer. The experiment is executed on a German Traffic Sign dataset and the performance is measured in terms of accuracy and time to detect a single image during the process. The results are quite impressive with correct recognition of traffic sign classes with an accuracy of 99.75% and the execution time per frame is 5.4ms. The performance is better when compared to other object detection algorithms. The time to recognize a single image is also short enough to consider it as a real-time object detection method. Though there is a lack of anti-error detection and comprehensiveness in the proposed method which can reduce the false positives and improve the overall performance of the method.

The experiment (Yi Yang et al. 2016) proposes a quick object detection model that is claimed to be twenty times quicker than the existing techniques. The detection module consists of traffic sign classification and extraction of proposals that are developed on a color probability model and HOG. The next step is to classify the detected signs into subclasses. The German and Chinese road signs datasets are used in the experiment and the performance is measured individually. The accuracy for GTSDB is 98.24% while CTSD gets 98.77%. Also, the proposed method took an average of 165ms approximately to detect a single sign. The results make it a good candidate for real-time traffic sign detection. The proposed method performed well on the evaluation criteria, yet the process could be faster by using GPU to reduce the execution time.

Authors in (R. Gavrilescu's et al 2018) implemented Faster R-CNN basic version on traffic sign detection to measure its performance in terms of classification, accuracy, and time. They used Graphical User Interface (GPU) for training and testing the images at the speed of 15 frames per second. The dataset is limited to 3000 images and mainly contains four classes of traffic signs. In addition to STOP signs, the experiment also detects the traffic lights such as Red, Green, and Yellow. The results clearly show that the model acquired 98% accuracy in recognizing the STOP signs in the morning and afternoon, but the accuracy reduced in the evening to 95% from a 4k dashcam. On the other hand, for an HD dashcam, accuracy in the morning is around 86% and reduced to 70% in the evening. The maximum accuracy is in the afternoon. The time is 15 frames per second. However, the camera used in the experiment did not have enough stability to click clear and robust images. Thus, the performance of the model highly depends on the hardware apparatus used.

The experiment in (Huang, W. et al. 2018) focuses on detecting small objects in traffic sign input frames. The work combines CNN and Generative Adversarial Networks (GAN) to detect and classify the tiny objects in the input images. They used generative network super-resolution features that help to identify items that are considerably smaller in size with low visibility. The Tsinghua-Tencent dataset comprises around 30,000 high-resolution images. Out of 100 classes, they considered only 45 classes for the experiment and divided the dataset into images with different sizes such as small, medium, and large objects. The paper measures the performance based on accuracy and recall values. The proposed method got an accuracy of 89.5 percent which is higher when compared to Fast-RCNN and Faster-RCNN with 60.91 and 70.18 percentages. Furthermore, the recall value is also higher at 87.27 for the proposed method when compared to traditional ones. The combination of GAN with Faster RCNN gives promising results while at the same time, it makes the network complex. The two-stage training process of the model leads to an increase in the execution time and makes calculations more complex. As a result, the detection gets slower and the model is unsuitable for real-time traffic sign detection. Moreover, the complex calculations also require more resources, higher CPU and GPU power, more memory, and other resource issues.

The experiment performed by (LU Runge et al. 2018) introduced transfer learning to the RCNN algorithm. The RCNN has been trained by a large-scale dataset CIFAR-10 that contains at least 50 thousand images of different types of objects. The performance is measured using evaluation metrics like false detections, recall percentage, and precision. The model calculated exactly 95 classes out of 100. The precision and recall percentages are 99 and 95 percent respectively. However, the paper does not compare the proposed model to any state of art model or with the model without transfer learning to showcase the impact of the proposed approach. There are no metrics to measure the speed at which the model detects the images. The dataset is also not large enough to train the labels for pre-trained models. The proposed model just detects the stop sign and needs to implement similar steps for other categories as well.

Authors in (Palavanchu 2021) compared the five most advanced and successful object detection algorithms of the time by implementing transfer learning. They used Xceptionnetwork, InceptionV3 Networks, Residual Networks, ResNet50, VGG-16, and EfficientNetB0 models for the detection. All these models are trained with ImageNet dataset weights from the Keras libraries. The weights of the pre-trained layers are retained by lower layers and act as a base model which can be fine-tuned for the German traffic Sign recognition dataset by further layers. The results are promising enough to discuss. Xception has the highest accuracy that is more than 95 percent whereas VGG-16 underperformed with the lowest value. Likewise, the loss is highest in VGG-16 and lowest in Xception. All the models took around 1hr 50 minutes of training time except InceptionV3 which took only 58 mins to get trained. Multi-Column DNN took several days to get trained and gave 99.4% of accuracy. These discussed approaches are not appropriate for real-time traffic sign recognition as the execution time is relatively high. The technique could be used in a video experiment that could simulate the actual scenario of self-driving cars and reevaluate the results.

The study (Rezapour et al. 2021) proposed a novel approach to roadside barriers identification. They have used pre-trained inceptionV3, denseNet 121, and VGG 19 models for this purpose. They removed the top layer and froze the remaining layers. In addition to it, two layers are added to the stack. In this way, only a few last layers will be trained while the remaining will be locked. The dataset was recorded in 2016-2017 as a part of the WYDOT traffic inventory project. It contains at least 7700 high-resolution images. Notably, among the transfer learning modules, VGG-19 shows 97 percent accuracy when compared to other models. Non-transfer learning models had a considerably good accuracy score of 85%. Furthermore, using VGG-19 with transfer learning architecture, accuracy for most of the classes is 100%. However, the frequency of images is low. The VGG-19 outperformed all other algorithms used as baseline in the experiment. Also, in the non-transfer learning models, there is a poor prediction between hybrid barriers and box beams. This could be due to the lesser amount of images for these classes. This problem can be solved by deploying a larger and more versatile dataset that can offer variety to the training and testing sets. Along with accuracy, pre-trained models take a longer training time to predict the values as compared to self-supervised traditional models.

The author in (Rezapour et.al 2021)attempted to detect bubbles with a proposed method named BubCNN. It used a Faster RCNN detector to locate bubbles and CNN as a shape regressor to detect the shape, size, and other attributes. The amalgamation of these two algorithms gave birth to BubCNN. Stage 1 of BubCNN is shaped regression CNN and will be employed in later stages. Also, the Faster RCNN module relies on pre-trained weights of CNN regression. The experiment required CPU and GPU to accelerate the image processing during the process. As compared to state-of-the-art image processing procedures, BubCNN shows a better ability to generalize. It acts as a tool that can be used in many experimental setups and helps save researchers from programming new evaluations from scratch. However, this approach does not account for time evaluation. They used GPU to accelerate the processing but no efforts on measuring and improving the performance.

The study (JH and HY 2017) used methods such as vertex, bilateral Chinese, and bisector transforms for image processing. The dataset consists of images clicked by the dashcam and processed with the histogram to extract the feature vectors. At the last stage, a neural network is employed to detect traffic signs. The experiment involves an SVM classifier trained by the HOG feature. It is quite common in object detection experiments. To remove the wrong detection from the parts of images that look similar to traffic signs, the authors used Bilateral Chinese transform (BCT) and vertex bisector transform (VBT) to correct it. The training data for triangular signs, circular signs, and no signs are 1200, 1300, and 1500 respectively. The results for HOG+SVM are quite impressive with a recall value of 92% and true positives are 2631. Using BCT and VBT, there is not much improvement in the recall value and ends up at 90%. However, the precision is 88% with 2578 correctly detected images. The experiment has a very small and limited dataset to train the models that affect the performance of the models. Also, the measurement of time execution is not considered as a metric while analyzing the results. It is uncertain whether the proposed approach is appropriate for real-time traffic sign detection. The VBT and BCT do not show considerable improvement in the results for this small dataset. It would be an improvement if the same techniques are applied to the German Traffic Sign dataset or another similar but large dataset.

The authors in (Wan et al. 2021) proposed a novel method called TS-Yolo called Traffic Sign You Only Look Once that aims to improve the accuracy and performance of the convolutional neural networks, especially under low light conditions. The experiment used the copy and paste augmentation technique to reproduce a large dataset from the existing traffic signs. The MixConv technique is used to blend the different kernel sizes in a single operation. Further, AFF or Attention Feature Fusion technique is integrated to speed up the process. The dataset deployed is YoloV5 and the precision value comes out as 71.92. Without augmentation of YoloV5, the precision dropped by 34.56. Also, the mAP value is 80.05 which is improved by 33.11 when compared to the experiment without augmentation. With the implementation of MixConv and AFF. TS-Yolo model precision is 74.53 and the mAP value is 8.73. The overall performance of the proposed model is satisfactory, but it does not stand out against the other efficient models. Also, the author needs to try different levels and combinations of augmentation for the same technique to find better results. Also, the bad weather conditions such as haze, pollution, and tunnel environments should be more realistic to get the actual behavior of the model in such scenarios. There is a scope of using better CNN object detection models such as CenterNet and EfficientDiet with the same approach and analyzing the results.

Another recent work in (Dogancan et al. 2020) proposed a dataset named CURE-TSD Real dataset. It overcomes the shortcomings of the realistic dataset which is appropriate to demonstrate harsh weather conditions and low visibility scenarios. They also observe the visibility changes through spectral analysis and show that the challenging conditions can lead to distinct magnitude spectrum characteristics. It is observed that the performance of two benchmarks algorithms degraded by 29% for precision and 68% for recall. The performance metrics recorded for experiment execution are true positives, false positives, precision, recalls, and F1 Score. The testing contains not only images but few videos with more than 300 frames per video. The experiment is divided into 6 challenge levels, where the base level is a sunny scenario but as the level increases the performance of the model decreases. The experiment also calculated the performance degradation for each of the challenge categories. The maximum degradation of the model's performance is during rain which is 74.5%. Whereas the minimum is 16.2%. This implies the need for engineers to focus more on rainy scenarios rather than shadows, noise, or hazy situations in the future.

The authors (Shustanov and Yakimov, 2017) investigate the possibility of using convolutional neural networks (CNN) for real-time traffic sign detection. It evaluates different approaches based on performance metrics like accuracy, recall value and precision. Training is performed by the Tensorflow library and CUDA (a parallel architecture for multi-threaded programming techniques). The experiment is performed in a real-world scenario with the help of the mobile GPU. The accuracy comes out to be 99.94% which states that the model correctly classified 99.94 percent of the total testing images. The model is able to detect the traffic sign at a distance of 50m as well. Tensorflow libraries help to process videos of high quality during the execution of the experiment thus helping in real-time detection of the signs. The best performance is given by sliding window and SVN together with an accuracy of 100%. The worst performance is given by HOG architecture with an accuracy of 70.33%. However, it is essential to see how many frames per second a particular architecture is while experimenting. Sliding Window + SVN processed one frame per second whereas the model with the lowest performance processed 20 frames per second. Best performance in terms of accuracy and speed is given by Modified GHT with preprocessing + CNN, that is 50 frames per second with an accuracy of 99.94% as discussed above. The authors must consider the real-time situational images which involve rain, haze, fog, dark, cloudy, and many more.
The authors in (Lim et al. 2017) showcase a novel approach named General Purpose Graphical Processing Unit (GPGPU) for traffic sign recognition. The proposed method is robust against illuminating changes and uses hierarchical models to detect and classify the

objects. Firstly, they derive the Byte-MCT feature from the input image to identify the area of interest. It uses landmark-based parallel windows to detect multiple regions of interest with decent performance even in low light. Moreover, GPGPU helps to perform real-time detection using parallel window searching methods rather than traditional ones. For every grid, the processing takes place parallelly thus making the process exponentially fast. The dataset used here is the LISA US traffic sign with the combination of real-world driving videos recorded from the dashcam of the cars. The Vienna traffic sign dataset is also recorded using an HDR front-mounted camera with Electric Chromatic Mirror or ECM. The data is divided into three subsets, the Korean Daytime dataset, Korean Nighttime, and German Daytime.

The performance of the ACF (Aggregated Channel feature) is the highest among the other models on the LISA dataset. The ACF's F1 Score with LISA is 0.84 whereas Korean Nighttime is as low as 01577. The performance of the proposed method for the given data sets drew an interesting picture. The performance on the Korean Nighttime dataset is improved to a 0.98 F1 score. Whereas the F1 score for other datasets is as high as 0.90 and more. The Byte-MCT Detector with CNN also shows better results than the ACF one. The results are promising, and the real-time evaluation is also tested against the model. However, in terms of traffic signs, the average accuracy achieved i.e. 89.5 is not so great as it is a matter of life and death. Higher the accuracy, the more robust the system is.

The study by (Lim et al. 2017; Liu and Xiong 2020) also focuses on the improvement in real time object detection methodology. The authors used the YOLOv3 algorithm for target detection. The main purpose of the paper is to detect the small objects in the image or videos. This study replaces the traditional 2-dimensional convolutional layer with the dilated convolutional layer which expands the receiving end. At the end of the model, they implement tri-layered pyramid structure to make multi-scale feature map deep fusion. The dataset used here is GTSDB. The performance is evaluated by using other competitive algorithms in the state of art like Faster RCNN and Article method. The results showed that the YOLOV3 performed well with accuracy of 80.44 higher than Faster RCNN which is 78% but lower than the Article method which achieved the accuracy of 85.26%. But, as per speed concerns, YOLOv3 outperforms with 46 frames per second whereas the article method processed 38.52 frames for the same time. The cascading made the model bulky and dilated convolution made it complex to calculate efficiently. The final results showcase that when combined with dilated convolution and multilevel spatial pyramid, the performance of the model is 4.82% higher than YOLOv3 and 36 frames faster than the Faster RCNN.

The authors in (A. Santos et al. 2020) made a real-time traffic sign detection algorithm with voice alerts. For efficient, accurate, and faster segmentation, four preprocessed methods are used. In the recognition stage, 10 algorithms are applied to determine which approach will provide the accuracy and fast execution. In the data preprocessing stage, the shadow and highlights are adjusted, and the results are observed. It requires an Nvidia graphics card, an LCD screen, speaker and python code, and an environment to run that code. The apparatus is able to operate on the average frame rate of 8-12 frames per second. The various stages of processing involve data acquisitions, pre-processing and detection, feature extraction, and evaluation. The dataset in the experiment has a total of 20194 images in the testing set. The images are of high resolution and from different angles. The data is augmented with variable image manipulation like hue, saturation, color corrections, brightness, and other attributes. KNN and CNN are the techniques with the highest accuracy with 91.13% and 92.97% respectively. Similarly, QDA and ADAB are the worst-performing algorithms among others. In terms of time execution, CNN is the fastest with 7.81ms, and after that its MLPC with 11.98ms. In contrast to CNN, GPC performed badly at execution time by taking 750ms which is higher than any other algorithm compared. The TDSR system average time taken for the image after going through multiple layers and methods comes out to be 1.095 seconds. The accuracy could be improved in the experiment by deploying an HDR camera with better image stabilization techniques. Also, there is an advanced masked RCNN technique that can help in increasing the accuracy of the results.

This experiment (Chen Z. et al 2020) targets lane detection on the roads while a car is driving in low light conditions. It is a crucial task to perform because any fractional error can lead to loss of life or property. The authors used GANs (Generative Adversarial Networks) to transform the images into low light conditions. The proposed approach consists of three parts, SIM-CycleGAN, lane detection network and light conditions style transfer. The lane detection methods are validated using ERFNet. The modes are trained with the dataset of 3200 images in suitable light and an equal number of images in a bad light. The testing dataset contains 13000 images all together to get the results. The final figures are shown according to the different lighting conditions. In the end, the total accuracy among all the scenarios is shared for different approaches employed during the experiment. Apart from the situation where no line exists, curved roads have the lower performance where SCNN is the underperformer among its fellow methods. CycleGAN+ERFNet performs better in this situation as compared to others as its accuracy is 69. Similarly, in the normal scenario, SIM-CycleGAN+ERFNet performed very well with 91.8 accuracies and ENet-SAD underperformed with an accuracy of 90.1. The authors didn’t train SIM-CycleGAN and ERFNet together which can improve the performance.

Another work in (B. Liu et al 2017) aims to improve the performance of FasterRCNN with various architectures. It will check the feasibility of Faster RCNN with ResNet101, VGG16, and PVANET. The performance is measured using mean average precision commonly known as mAP. The aim is to obtain a better model by comparing the feasible combination of networks with the Faster RCNN and choose the best one. The dataset contains multiple images of different categories like horses, airplanes, bottles, cats, buses, motorbikes, and many more. Each category is scrutinized for three listed approaches. The Faster RCNN + PVANET obtained the higher precision value among the other models which is 84.9 percent. Faster RCNN + ResNet101 is a bit lower than the first one by 72.5%. Also, regarding the performance according to categories, airplanes are detected more accurately by PVANET. The case is similar for most of the other distinct categories such as bicycle, bottle, horse, and others. Though results are satisfactory, there is a scope of improvement in terms of accuracy. To improve the accuracy, the authors can use transfer learning for the same models by using a TensorFlow pre-trained FasterRCNN RESNET-101 model and test against the same dataset.

The paper (Jiang et al. 2020) implemented the latest YOLOv4 model which aims to simplify the network and reduce the parameters. They proposed YOLOv4-tiny which is suitable for mobile or other portable devices. It uses two modules of ResBlock-D instead of CSPBlock modules in YOLOv4-tiny. This technique reduces computational complexity. Moreover, to reduce the error while detecting, an auxiliary network block is used. In the end phase, the auxiliary network is merged with the backbone network to create the basic and improved architecture of YOLOv4-tiny. YOLOv4-tiny and YOLOv3-tiny will be compared on different parameters like mAP and execution time to evaluate the performances. To reduce the processing time, the experiment implemented GPU processors from Nvidia. The YOLOv3-tiny used 1123 MB of GPU and v4 used 1055MB. The proposed method used 1003MB for the same processing images.
The average precision value of v3 is 52.5 whereas v3-tiny is 30.1. Similarly, for v4, mAP is 64.9 and for the tiny version the same ended up with 38.1%. The proposed method is the fastest with 297 frames per second but its mAP value is 38% which is low. This explains that processing speeds up due to the simple architecture but it compromises the accuracy. One has to deploy a more sophisticated approach that can perform well in both scenarios.

The authors (Xiao et al., 2020) aim to improve the object detection for the low illuminated images. Most of the time in real-time implementations, objects with low illumination are ignored by the systems. This paper helped in getting a clear picture of the need for illumination detection and enhancement to get better results. The paper proposed NVD (Night Vision Detector) with a cocktail of customized pyramid networks and context fusion networks for detection. The dataset used here is ExDARK and COCO to serve the purpose. The experiment helped to visualize the impacts of illumination on feature preservations. In the end, NVD performed slightly well as compared to other techniques used in the experiments. It achieved 0.5% to 2.8% higher accuracy than the basic RFB-Net on all standard COCO evaluation criteria. Also, if the image dataset is of high-quality images, it could be handled more gracefully, and illumination detection could be improved.

Another work (Xiao et al. 2020) highlighted the limitations of FasterRCNN when dealing with the specialized cases of low visibility, partial object displays, obstruction in vision, and deformations. The paper proposes an improvement in Faster RCNN by skip pooling and fusion of contextual information. It involves three parts, the first is to add a context information model for feature extraction. The second is to add skip pooling and the third is to replace RPN with guided anchor RPN. The dataset is a collection of generalized images which has images of people and random daily life objects. The algorithm performed better on an average percentage of 6.875% on mAP when compared to Faster RCNN and Yolov3. It shows that the proposed architecture is good at detecting the objects in special conditions which are not ideal. The technique shows positive results however, it still cannot detect the deformed, rotating, and camouflaged objects. Another problem with the model is the lack of training with the vast and generalized dataset and complex computations due to novel architecture.

Based on the in-depth research done on the above-discussed published work in the field of real-time object detection, my proposed model T-FasterRCNN with German Traffic sign dataset is a unique work carried out by me. T-Faster RCNN incorporates transfer learning techniques to improve the performance of the model. This makes T-Faster RCNN a good candidate for real-time traffic sign detection. The papers discussed showcase good approaches to enhance the performance of the detection algorithms but the model's complexity and resources are always an issue. Sometimes a complex model underperformance due to unwanted operations and stages causes loss of accuracy and slow processing. Also, the unavailability of realistic datasets hindered the development efforts in the domain. Another major issue highlighted is the poor performance of the models in low visibility images especially on hazy or rainy days. Keeping this in mind, the T-Faster RCNN model is tested with images with low visibility scenarios with three levels of blurriness and shifting. The performance of the model is evaluated for each level to see how well it performs with such real-time situations.

Chapter 3

Transfer Learning for Faster Recurrent Convolutional Neural Networks for Real-time Traffic Classification

3.1 Design

Traffic sign detection in real-time is a challenge that demands quick detection and accuracy at the same time. Both the metrics are equally important to facilitate automatically driven cars. The hybrid model T-FasterRCNN uses the Faster RCNN object detection algorithm along with Resnet layers and transfer learning aiming to detect the traffic signs in different weather conditions. The methodology includes the following stages in Figure 3.1.

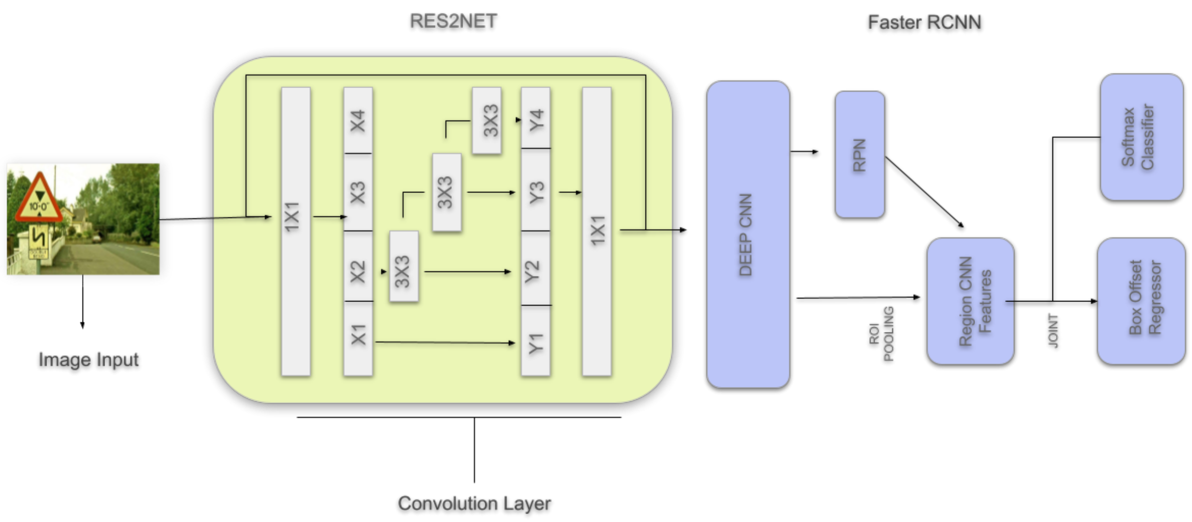


Figure 3.1 T-FasterRCNN stages of image processing.

The methodology includes various stages of data processing, feature engineering, training T-FasterRCNN, and prediction of signs. The output will be the box plots and class id for each of the images to be given as input to the model. The workflow shown above in Figure 3.1 is a basic idea of how the hybrid T-Faster RCNN works in detecting the signs. The T-FasterRCNN has the following steps of processing:

- Deep CNN: Input image in Deep CNN to get feature map.
- RPN (Regional Proposal Network): identifies interesting boxes and regions
- RPN use further connected layers to provide bounded boxes and classes

The dataset as given in Section 3.2 is analyzed with graphs and plots to determine the different attributes and perform operations to modify and split the data into training, validation, and testing subsets. T-Faster RCNN involves training the labels to pre-train models with Resnet-50 layers. The technique not only improves the accuracy but also improves the processing time per image. T-Faster RCNN does not use selective search algorithms but uses separate networks to predict the local proposals. These predicted proposals are fed into the ROI pooling layer to predict the bounding boxes coordinates. This makes the image processing effective in T-Faster RCNN.

3.2 Datasets

The datasets used in the thesis are the German Traffic Sign Benchmark and German Traffic Sign Detection Benchmark which contains 43 different classes of traffic signs that can be used to train the model. The number of images is approximately 50000 that involve signs in low light, bad weather, low visibility, and many more. The dataset is split into 12630 training images and 39209 testing images. The test.csv and train.csv have metadata about the images with attributes like:

- Class ID
- Height of the image
- Width of the image
- ROI.X1 , ROI.X2 coordinates
- ROI.Y1 , ROI.Y2 coordinates
- Path of the images

These attributes are used to validate the results to calculate accuracy and F1 scores. ROI.X and ROI.Y values are the x and y coordinates of the bounding boxes which contain the traffic sign. Height and width are the dimensions of the images in the database. The images in the dataset are resized by finding the average height and weight for images.

3.3 T-FasterRCNN for Traffic Sign Detection

3.3.1 Data refinement and multiscale feature mapping using Resnet-50

Resnet50 has a 4 stage layering process. The network can accept input images of dimensions multiple of 32 with 3 as the channel width. It performs initial convolution and max-pooling using 7X7 and 3X3 kernel sizes. The first stage of the network has 3 Residual blocks with 3 layers in each block. This reduces the size of the input to half and doubles the channel width. As the process moves to the next stages, the dimensions are going to be half and the channel width gets doubled. This is illustrated in Algorithm 1 given below.

Algorithm 1: Resnet-50 feature extraction

```
1. Input: data0 ▷ traffic signals dataset
2. Output: return(scaled_features) ▷ shifted subsets
3. scaled_features ← ∮imgProcessor(data0)

                    (data$\alpha$, data$\beta$, data$\gamma$, data$\theta$) ⊂ scaled_features

                    ▷ where data$\theta$ is data clear visibility no modifications

                    ▷ data$\alpha$ is 5% shifted

                    ▷ data$\beta$ is 10% shifted

                    ▷ data$\gamma$ is 15% shifted

                 ∮imgProcessor is a Image Data Generator
4. for each data ∈ (scaled_features) do

     for each img ∈ data do
```

```
            feature_maps ← ∮convSplit1X1(img) ▷ 1X1 Convolution layers splitter
            Input: feature_maps
            Output: return(scaled_features_data)
            scaled_features_data ← ∮featureMapSubsets(feature_maps) ▷ i ∈ {1, 2, ...,
        s}
                ▷ xi when i = 1;
                ▷ Kθi(xi) when i = 2;
                ▷ Kθi(xi + yi—1) when 2 < i < s.
                ▷ (dataα̂, dataβ, dataγ̂, dataθ̂) ⊂ scaled_features_data
                ▷ 3 × 3 convolution, denoted by Kθ() ▷ for xi
          end
        end
```

The output of the algorithm will be used as input to the next module.

Algorithm 1 takes Traffic sign dataset data0 as an input and provides scale features with corresponding shift levels as output. This algorithm then uses imgProcessor as a data generator that provides different copies of training and testing datasets. These images are shifted to a different level of blurriness such as Level 1 for 5% blurriness. Level2 is for 10% blurriness and shift and Level3 is for 15% blurriness.

The modified training data is then iterated over the feature mapping methods to get the scaled feature data. In the feature mapping method, the image will be sent to convSplit layer and get featured maps as the returned values. These feature maps are further sent to the ∮featureMapSubsets method which processes 50 layered Deep convolutional networks and provides us the scaled features which can be used as an input to the T-Faster RCNN model. This method is also called the regional proposal network. It is a crucial stage in the

FasterRCNN model which distinguishes it from the object detection methods. It is responsible for calculating the object scores and their boundaries for each detection. In this way, RPN is responsible for making the T-Faster RCNN a real-time object detection model.

3.3.2 T-Faster RCNN Model

This algorithm will take featuredMap as input and apply ROI and RPN on each of the images to replace the featured map with the proposal. These proposals are input. to the softmax and boundedRegressor that will return the proposed featured id and box coordinates.

Algorithm 2 T-Faster RCNN Model:

```
1. Input: scaled_features_data
2. Output: return(bounded_images)
3. for each data̕ ∈ (data$\hat{\alpha}$, data$\beta$, data$\hat{\gamma}$, data$\hat{\theta}$) do
       for each featured_map ∈ data˜ do
           obj_proposal ← ∮rpn(featured_map) ▷ (returns object proposals along with
             objectness score)
           obj_proposal ← ∮roi(obj_proposal) ▷ (resize object proposals to same size)
           data̕[featured_map] ← obj_proposal ▷ (replace the featured map with
   proposal)
     end
   end
4. for each featured_map ∈ data˜ do
       ● obj_proposal ← ∮rpn(featured_map) ▷ (returns object proposals along
         with objectness score)
       ● obj_proposal ← ∮roi(obj_proposal) ▷ (resize object proposals to same
         size)
       ● data̕[featured_map] ← obj_proposal ▷ (replace the featured map with
         proposal)
5. for each proposal ∈ data˜ do
     ● object ← ∮softmax(proposal) ▷ (classifier
```

- bounded_image ⟵ ∱boundedRegression(object) ▷ (*create bounded boxes for objects*)
- data[proposal] ⟵ bounded_image ▷ (*replace the featured map with proposal*)

  end

At this point it is very important to discuss the role of key components of this algorithm.

- **∱rpn:** This method is also called the regional proposal network(RPN). It is a crucial stage in the Faster R-CNN model which distinguishes it from the rest of the CNN models. It is responsible for calculating the object scores and their boundaries for each detection. RPN (Fan et al. 2020) has a classifier and a regressor. T-FasterRCNN represents the concept of anchors. It is the central point of the sliding window. The role of classifiers is to find the probability of a proposal while having the target object. Regression, on the other hand, regresses the coordinates of the proposals. Thus, with the presence of multi-scale anchors in the algorithm, the model produces a pyramid of anchors instead of a pyramid of filters. It makes its execution faster and requires less processing. In this way, RPN is responsible for making the T-Faster RCNN a real-time object detection model.

- **∱roi**: ROI (Kumari and Kr 2021)stands for Region of Interest. It is a type of pooling layer which performs maximum pooling on the input of variable sizes into the fixed-size feature map. It first divides the regional proposals into uniformly sized sections. Then it aims to find the largest value in each section and copy these values to the output buffer. This ends up with a list of feature maps of fixed dimensions. This leads to the reduced processing time as calculating the convolutions at the early stages is highly expensive and slow . Thus, the main purpose of ROI pooling is to speed up the training and testing time.

- **Softmax classifier**: It is also called cross-entropy loss. It is the final layer at the end of the network (Gao et al. 2021) which is responsible for giving the actual score of probability for each label. It is generally used to normalize the output. Softmax classifier uses the One-Hot Encoding (Hussein et al. 2021) technique to categorize the data. For example, if the softmax predicts that the object is a stop sign, then One-Hot Encoding will be [1 0 0] for speed limit sign, [0 1 0] for a stop sign and [0 01] for a school ahead.

These predictions are not understandable, and the system needs to identify the objects correctly. Thus, the classifier uses a cross-entropy function here. It calculates the loss for all the classifiers. The low value of loss means the high accuracy of the prediction. The loss could be calculated by the mathematical formula::

LOSS=np.sum(-Y*np.log(Y_pred)) (3.1)

- **Bounded Regression**: It is used to refine the localization boxes in the object detection algorithms (Fu et al. 2020). It helps us to find the continuous coordinates of X and Y that can draw a more precise bounded box over the detected object which is not possible with traditional object detection algorithms.

The algorithm 3 is for bounded box regressor used in T-Faster RCNN is as follows:

Algorithm 3: Bounded Box Regressor

```
1. Input: object {(pi, Gi)}i = 1….N
            ▷ pi ←pixel coordinates of center of proposal
            ▷ Gi←ground-truth bounded box
2. Output: return(bounded_boxes)
3. d(p̀) ← ∮parametrize(object) ▷ d(p̀) is (dk(p), dl(p)ḑm(p), dn(p))
4. G̒← ∮transformation(d(p̀)) ▷ G̒ is (G̒k, G̒l, G̒m, G̒n)
5. ti ← ∮findTargets(pi, Gi)
            ▷ where ti ← (tk, tl, tm, tn)
            tk ←(Gk̀ - Pk̀) / pm̀
            tl ←(Gl̀ - Pl̀) / pǹ
```

```
        tm ←log(Gm̀ / pm̀)

        tn ←log(Gǹ / pǹ)
6.  Lr←regularization(ti, di(p) ) ▷ minimizing sum of squared errors loss

        ▷ where regularization = ∑ i ∈ {k,l,m,n} (ti− di(p̀))^2  + ƛ|w|^2)
7.  return(bounded_boxes)
```

The pixel coordinates represented by pi and Gi (ground truth bounded box) act as input to this projection. It will parametrize the pi coordinates and apply transformations of Gi coordinates. After these two stages, the returned values are passed to the findTargets methods by performing calculations of different coordinates. The final step is regularization in which aims to reduce the squared error loss. The bounded box coordinates for each image are returned as output in the end.

Chapter 4

Implementation and Results

4.1 System Requirements

4.1.1 Hardware Requirements

The experiments in this thesis are carried out by Google Co-Lab. It is a cloud-based platform that is available in the public domain and allows the design of a Jupyter notebook by pre-configured servers. It offers CPU and GPU to accelerate the processing and reduce the execution time for the Machine learning and Deep Learning Algorithms. Google Colab lets the users write the code in the browser interface and processing held in their backend servers. In this way, a user does not need to own high configuration systems. Also, code is easily available on multiple devices and easy to share and deploy

For very extensive experiments with high configuration requirements, Google Co-Lab or any service may charge. In such cases, installation of Anaconda to execute the Jupyter notebook on the local system with local configuration.

The basics requirement of this experiment is

- Intel Xeon Processor (Dual-Core)
- 2.30 Ghz
- 13 GB RAM
- GPU

4.1.2 Software Requirements

The first module in the python notebook for this thesis is to import all the important and required libraries which will be used in the different experiments. This section will discuss the important libraries used in the notebook and their usage and significance. These libraries act as plugins or gems which provide some predefined code to fulfill a specific purpose. For example, library matplotlib provides modules that can be imported into the code to plot the graphs and figures. Each imported module has its usage and the documentation is available on the python libraries. Following is the list of all the important libraries in the thesis.

Pandas: It is used for nearly all data-related projects. Python's Pandas library is widely used for managing and analyzing data. With pandas, you can transform, analyze, and clean your data by extracting it from a CSV file and transforming it into data frames in easy and convenient ways. It also supports the grouping, slicing, subsetting, and indexing of large datasets.

Keras: It is a high-level library built on the base of TensorFlow. Basically, it acts as an interface for Tensorflow. It is designed to perform fast execution of experiments for the deep neural networks. This tool also allows the training of deep learning models in clusters of graphics processing units (GPUs) and tensor processing units (TPUs).

matplotlib:It is a python library to plot the graphs in the code. It is an extension of NumPy. One can plot bar charts, histograms, heatmaps, correlation matrix, bubble charts, line charts, and many more. It provides flexibility to customize the graphs by providing features like gradients, colors, markers, opacity, grouping, and stacking.

tqdm: tqdm is a library used to show the progress of the loops in a progress bar while the code is being executed. Wrapping the iterator in the tqdm method while looping will provide the time being spent in each iteration of the loop and the overall completion percentage of the loop at runtime.

NumPy: NumPy is a very basic library of the Python programming language which provides a robust N-dimensional array object useful for linear algebra and many more. It can also act as a multi-dimensional container for the general data.

4.2 Experiments

The primary objective of the research is to detect the traffic signs using the T-FasterRCNN Resnet-50 model which uses Resnet-50 weights and transfer learning to give better and faster results than the other traditional methods of object detection. The study will also compare the accuracy and execution time of RCNN, Fast RCNN, and T-FasterRCNN Resnet-50 model. The dataset being used here is the German Traffic Sign Dataset which contains 43 classes of traffic signs with 39000 training images and 12000 testing images. The thesis performs four experiments each with the number of epochs and batch size of 2000 images. The first step is to make some decisions related to the dataset like:

- Number of epochs and their batch size
- Image augmentations
- Size and quality of the images

The figures represent different aspects of the training datasets such as the height and width of the images and the correlation between the metadata of the images.

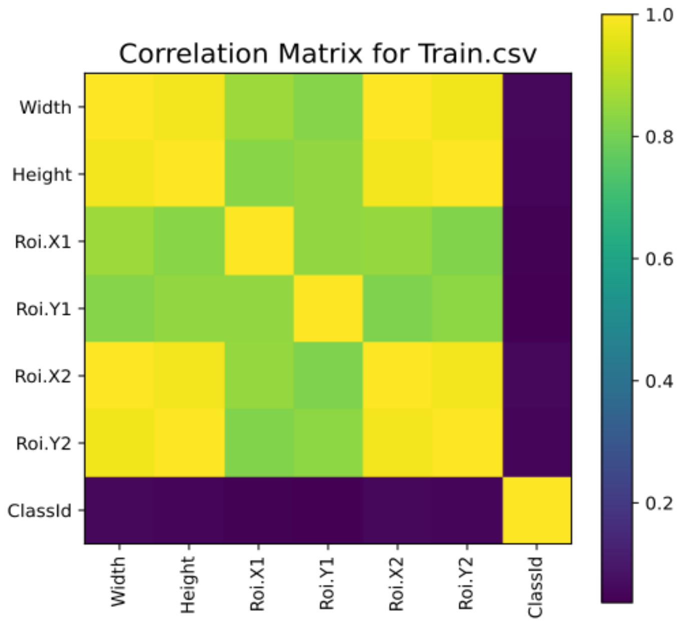


Figure 4.1. Correlation matrix for training dataset.

The aim here is to find the correlation between the different attributes of the metadata of the training dataset. The Train.csv contains the details of each image including attributes like x and y coordinates of boxes, the class id of the object, dimension of the images, and more. Figure 4.1 shows the correlation of the attributes with each other. It is observed that there is a strong correlation between height and width which implies that most of the images are square-shaped but Roi.X1 and Roi.Y1 has a correlation around 0.8 which implies that, unlike images, the bounding boxes are not always square-shaped. There is no correlation between class id and other attributes which implies that the shape and size of the image do not determine the class of the image in any case.

To determine the size to ingest the models for training data, I plot the height and width of the images and then crop those images to the average size that comes up through the graph.

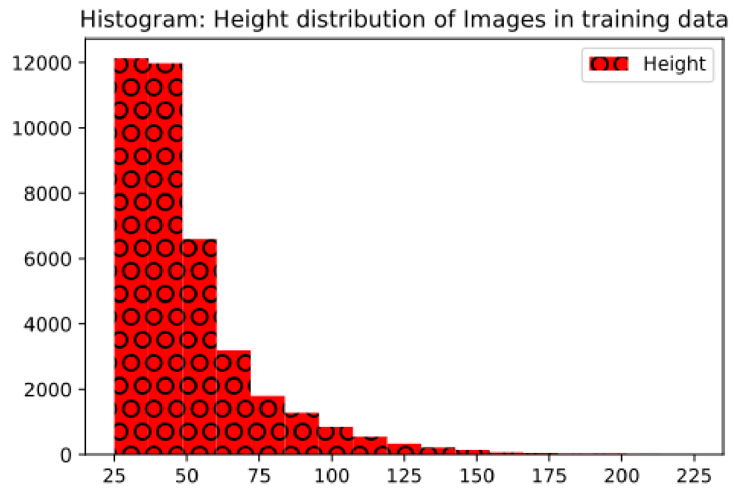


Figure 4.2. Average height distribution of training images.

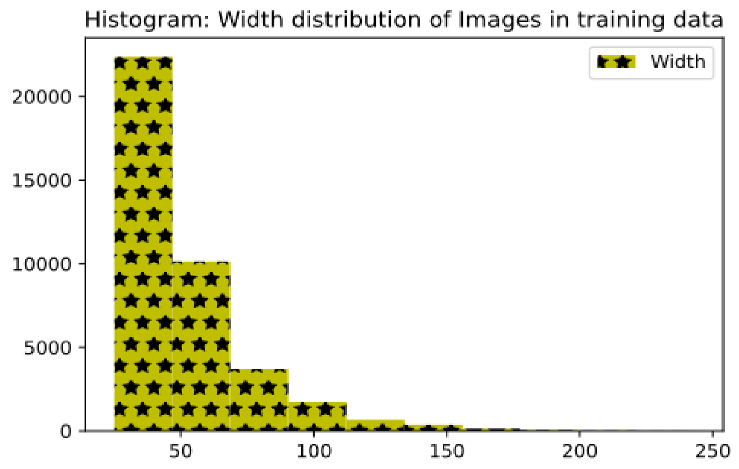


Figure 4.3. Average width distribution of training images.

As observed in Figure 4.2, the average height of the training images is between 25 pixels to 50 pixels. Almost around 25000 images are between 25 to 50 pixels. Similarly, Figure 4.3 shows the width dimensions distribution of training images. So, the average size of the images to ingest into the model must be 30 X 30 pixels. The next step is resizing all the images to the size specified using the Image library.

The ratio of training and testing images for the dataset is 7:3 i.e. 70% of the images are in the training dataset and the remaining 30% are available in the test dataset.

4.3 Performance Metrics

4.3.1 Evaluation Criteria

It's crucial to measure the performance of the trained object detection models for both the training and testing stages. It is important to analyze the behavior of the model while it is being trained by our labels and how effectively it is evaluating the test results. It could be the speed, accuracy, effectiveness, loss, and error in the entire process (Saura, 2021). This thesis analyzes the following performance metrics throughout the experiment's execution.

- ***Accuracy:*** In multilabel classification, the function calculates the accuracy (Yacouby and Axman 2020) per subset, i.e., the predicted and the actual values must be matched. It is one of the most common and important metrics used to evaluate the performances of the models.

- ***F1-Score:*** It could be calculated from the weighted average of the precision and recall as discussed above. The values lie between 0 and 1 where 0 is the worst value and 1 is the best. The mathematical formula to calculate it is as:

F1 Score = 2 x (precision x recall)/(precision + recall) (4.1)

- ***MSE(Mean Squared Value):*** Mean Squared Value (Yacouby and Axman 2020) is common among the regression algorithms. It can be calculated by squaring the difference between the predicted value and target value provided by the regression algorithm. The mathematical expression for the same is:

$$MSE = 1/N \sum^{N}_{(j=1)} (y_j - y'_j) \qquad (4.2)$$

where:

- y_j is the original value
- y'_j is the predicted value
- N is the number of record

- ***Execution Time:*** It is simply the execution time to train the model and predict the images. It is calculated in seconds.

Time = (Number of seconds to process/number of records in process) (4.3)

- ***Recall:*** It is the ratio of the total number of positive predicted values to all the positive predictions possibly made.

Recall = (True Positive/True Positive + False Negative) (4.4)

- ***Precision:*** Precision is the total number of correct positive predictions made for example corrected predicted class and bounded boxes of the images.

Precision: (True Positive/ True Positive + False Positive) (4.5)

4.4. Analysis of Results

This section is dedicated to the discussion of the results in graphical and tabular form and is an attempt to evaluate different neural network models with the help of above-mentioned performance metrics. The results are divided into four experiments and their metrics.

4.4.1 Performance of T-faster RCNN Resnet-50

T-Faster RCNN Resnet-50 is a novel model which uses transfer learning by using the Resnet-50 weights. Resnet-50 is a deep neural network and 50 refers to the number of layers. It is a subclass of Convolutional Neural Network commonly known as CNN and it is mostly used for Image classification as discussed in chapter 3. Faster RCNN is the first real-time object detection model which executes faster than the traditional object detection models. This thesis attempts to compare the performances of RCNN, Fast RCNN, and T-Faster RCNN Resent-50 model performance by measuring the accuracy and loss while training and testing the dataset.

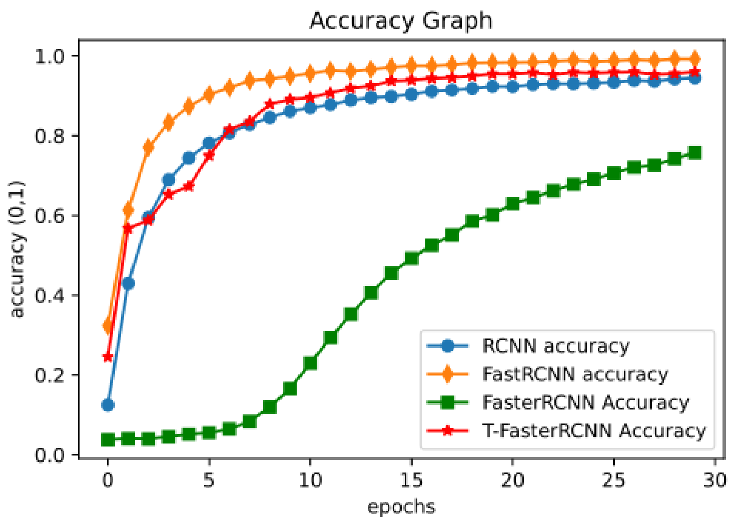


Figure 4.4 Accuracy of T-FasterRCNN as compared to RCNN and Fast RCNN

The training performance of the four models is shown in the line chart above in Figure 4.4. The graph presents the accuracy with an increasing number of epochs. The experiment is performed for 30 epochs with a batch size of 200 images. It is clearly shown that the RCNN gets stable at around 10 echos and its maximum accuracy at epoch 30 which is around 0.9. On the other hand, the Fast-RCNN shows the maximum accuracy while training with labels and shown stability around the 7-8th Epoch. The FasterRCNN accuracy is lower as compared to the other two models and the stability in the epoch attained at around 15 epochs. The accuracy is 0.89 for the training images. This shows that the FastRCNN with VGG16 weights learns faster in this scenario. T-Faster behaves a bit unstable at start but stables around 8 epochs. Training accuracy shoots around 0.9 maximum value.

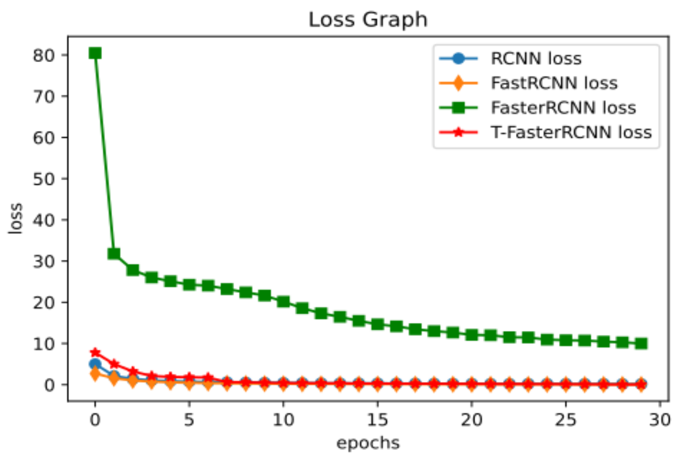


Figure 4.5 Loss of T-fasterRCNN as compared to RCNN and Fast RCNN

Training loss is the error on the training dataset. While training the models in cyclic epochs, we calculate how the models react with the increasing number of epochs. Clearly, in Figure 4.5, the Fast RCNN and RCNN have lesser training loss as compared to the FasterRCNN, this may be possible due to false positives as well. T-FasterRCNN performed well at minimising the training loss due to its ResNet-50 architecture that helps to reduce the loss. At start, the loss is above 10 but as the number of epoch increases the loss diminishes to 0 which shows that number of false positives decreased.

4.4.2 Training and Testing time of the T-FasterRCNN-Resnet50

It is a crucial parameter to consider while comparing the performance of the convolutional neural networks for real-time object detection. Traffic sign detection must be as quick as possible because a small delay in detection can cause serious repercussions. When claiming a real-time object detection model, it becomes everything about time and accuracy. We have observed the accuracy during the training time for three of our models in the above section. This section will showcase the time taken by the models to train the dataset and the time for detecting and predicting a single image. Figure 4.6 shows the training time of the three models with the same hyper tuning parameters.

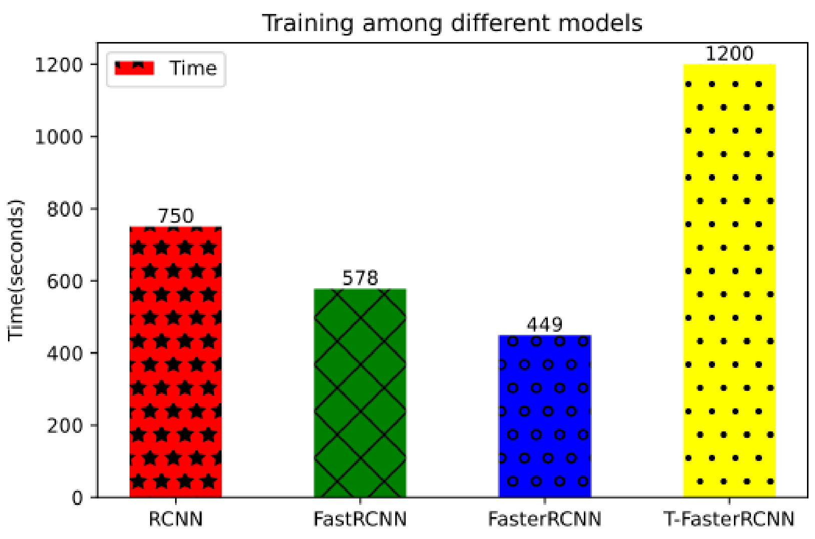


Figure 4.6 Training time T-FasterRCNN as compared RCNN and Fast RCNN.

The graph clearly shows that the RCNN took 750 seconds to train the model while the training time for Fast RCNN comes around 578. Interestingly, the training time of T-Faster RCNN is the highest among the models with 1200 seconds altogether which is higher than the Faster RCNN which is 449. This implies that the FasterRCNN model is training faster than the other three models. The reason behind this is that the Faster-RCNN network relies on region proposal algorithms to predict the object position. T-FasterRCNN takes higher training time due to complexity of the model.

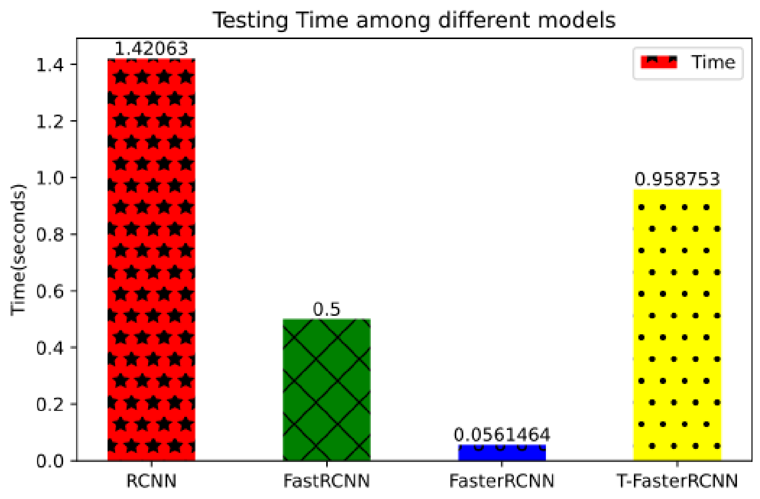


Figure 4.7 Testing time of RCNN,FastRCNN, and T-FasterRCNN

The key factor of object detection is how much time it takes to predict test images. This experiment includes 12361 test images and the graph in figure 4.7 shows the average time taken by the model to predict for a single image in the testing dataset. This graph clearly shows the winner in FasterRCNN as it took 0.032 seconds to predict the labels and bounding boxes of the traffic sign image. After that, FastRCNN took around 1.73 seconds to predict a traffic sign in the given image. T-FasterRCNN takes 0.9 seconds to process one image and in the end, RCNN is 2 times slower than the Fast RCNN with an average of 2.83 seconds to process an image. These results suggest that one can consider T-Faster RCNN and FasterRCNN for real-time Traffic Sign detection purposes though T-Faster RCNN needs improvements.

4.4.3 Prediction Accuracy of CNN, FasterRCNN, and T-FasterRCNN_RestNet50

This section discusses the predictions of the test dataset for all three models which include various performance metrics like Accuracy, F1 Score, and MSE. We have discussed these metrics and their significance in the above section.

Accuracy

Figure 4.8 shows the accuracy of the labels and box coordinates predicted by the models.

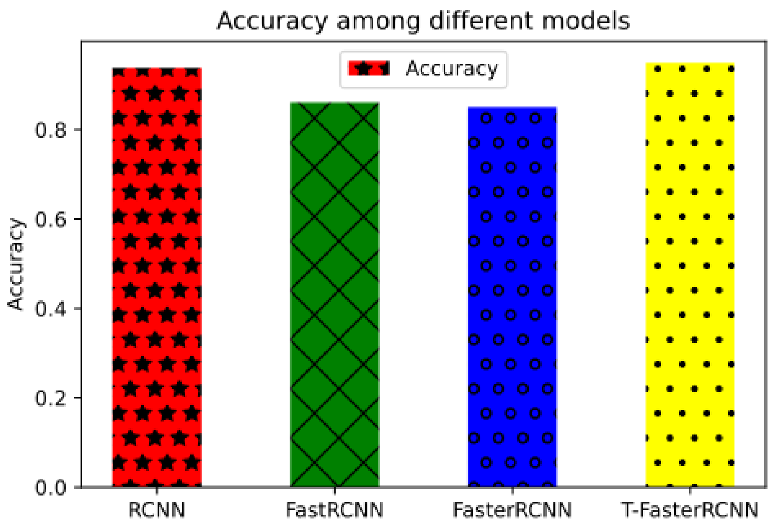


Figure 4.8 Accuracy of RCNN,FastRCNN, and T-FasterRCNN

I have observed that RCNN, FasterRCNN, and T-FasterRCNN have competitive accuracy for the testing dataset. The accuracy of RCNN is 0.933 and that of T-Faster RCNN is 0.958. Faster RCNN has 0.87 which is less than the other two. On the other hand, Fast RCNN feels short in terms of accuracy by almost around 10% with an exact figure of 0.83. This implies that both RCNN and T-FasterRCNN are more than 93 percent correct in their predictions and FasterRCNN is 87% correct.

F1 Score

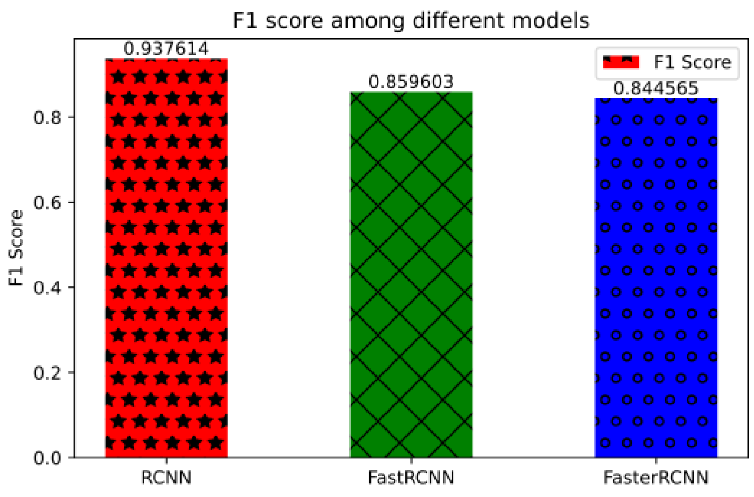


Figure 4.9 F1 score of RCNN, FastRCNN, and T-FasterRCNN

The trend is similar to accuracy for the F1 Score with an F1 Score of 0.93 for RCNN, 0.92 for T-FasterRCNN, and 0.83 for Fast RCNN.

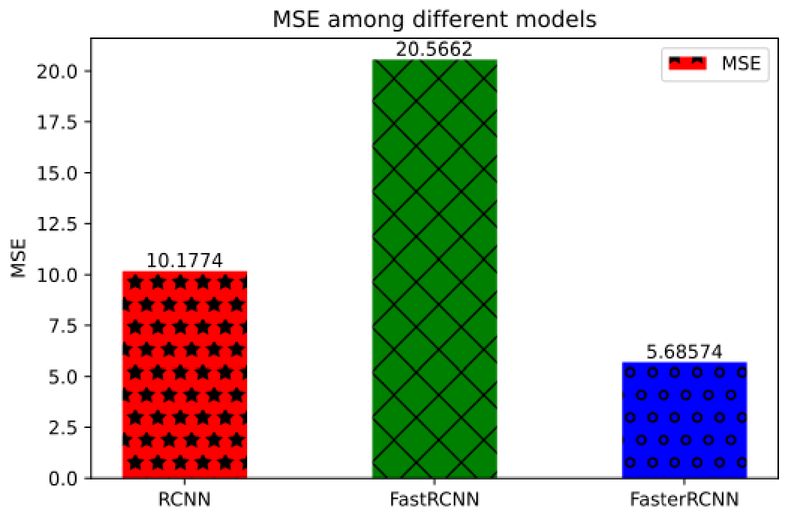


Figure 4.10 MSE among the different models for testing data.

MSE shows an improved performance of T-Faster RCNN as we can observe that the MSE or mean squared error is much smaller in comparison to RCNN and Faster RCNN. T-Faster RCNN has 1.9 MSE whereas RCNN and Fast RCNN have 14.74 and 23.43 respectively. This makes T-Faster RCNN better despite having almost the same level of accuracy for RCNN and T-Faster RCNN Resnet-50.

4.4.4 Effect of Blurriness and Shift levels on the performance of T-FasterRCNN_RestNet50

This section involves the augmentation of images to simulate the bad weather conditions by shifting the images and increasing their blurriness. The experiment has 3 levels of Blur and Shift and it will observe the impact on the different performance metrics for the T-FasterRCNN model.

Figure 4.11 shows the change of accuracy with the increase in the level of blur and shift for the testing images. At level 5, the accuracy is almost half the normal accuracy and set to

0.59. With 10 percent shifting, the same trend could be seen in the accuracy. It plunged to 0.28 which is almost a 40 percent reduction from the previous one. However, with the level of 15%, the accuracy drastically reduced to 0.11 which makes the model insufficient to detect the traffic sign in the scenario.

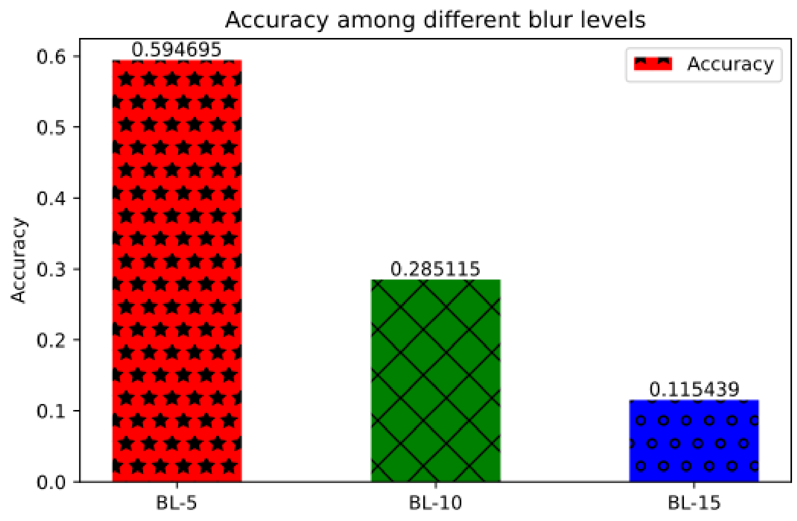


Figure 4.11 Accuracy among different blur levels

Similarly, for Precision, the same trend could be observed. With the increasing level of Blurriness, the F1 score reduced to 40% every time.

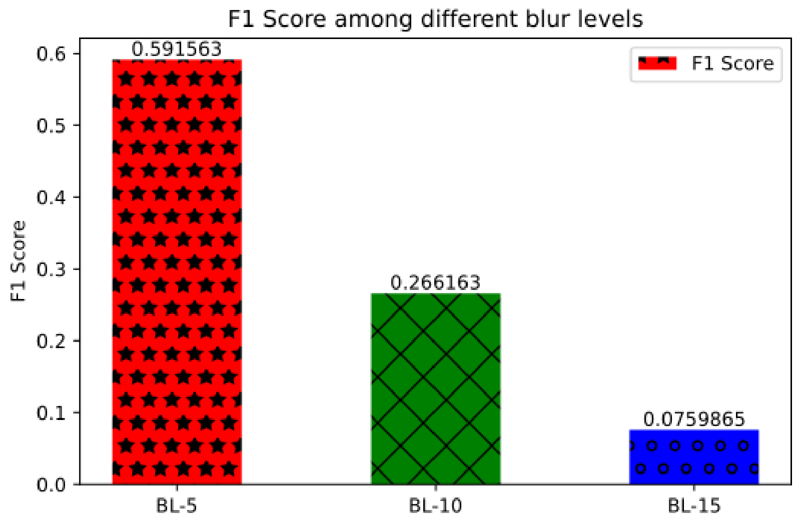


Figure 4.12 F1 Score among the different blur levels.

The same experiment with blurriness has some similar and significant results on Precision. With the increase in blur level, the value of precision degrades as shown in Figure 4.13.

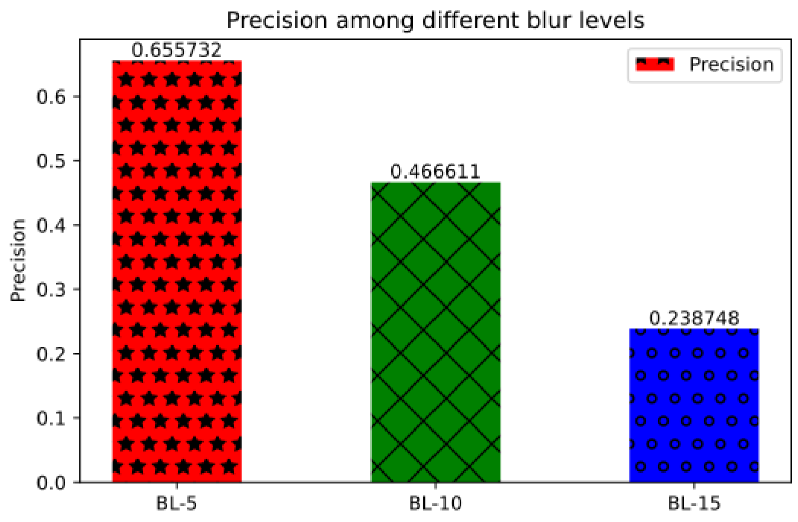


Figure 4.13 Precision among different blur levels.

On BL-5, the precision is 0.65 which is reduced to 0.46 at BL-10 and reduced to 0.23 at BL-15.

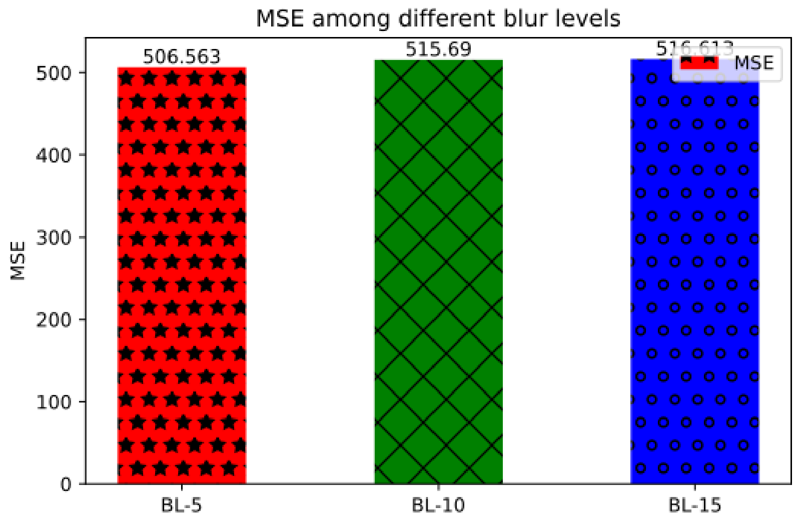


Figure 4.14 MSE among different blur levels.

It is interesting to see that as soon as the experiment touches BL-5, MSE drastically increases from 1.9 to 506. The change is huge and shows the average performance of the model in bad weather conditions. As the experiment moved to BL-10 and BL-15, the MSE level increased by almost 10 points respectively. This shows that there is a significant decrease in the performance of the model with the reduction of visibility due to various bad weather conditions.

In summary the following research questions have been answered by chapter 4:

- Transfer learning when applied to real-time traffic sign classification accelerates the prediction of FasterRCNN and RCNN.
- Transfer learned FasterRCNN achieves improved performance in terms of accuracy and F1 score for real-time traffic sign classifications.
- Image/traffic sign blurriness affects the performance of transfer learned FasterRCNN for real-time traffic sign classification.

Chapter 5
Conclusion and Future Works

5.1 Conclusion

Object detection has been a hot field of research these days and there is a wide scope of development in the domain. Self-driving cars have become the key area of research in object detection after the launch of these cars in commercial markets. Despite successful trials in many planned cities, these cars have a long way to become widespread among people across the globe. Right now these cars experience issues in low visibility and can cause failed detections. Faster RCNN performance is satisfactory in the ideal cases but in real-time scenarios where objects are not visible, rotating or deformed degrades its performance. This is due to the fixed size anchors that are unable to detect the object at once. The problem can be resolved by adding more sophisticated layers to the model. Also, transfer learning can help to train models better to detect the object in the fixed anchors. The thesis proposes the T-FasterRCNN method that improves the performance of the traditional Faster RCNN. It introduces transfer learning in the model by adding the TensorFlow pre-trained model. T-faster RCNN adopted Resnet50 architecture that comprises 4 stages with up to 50 layers to extract the features from the input image. This approach shows 3% better accuracy because it uses Regional Purpose Network and pre-trained weights to provide faster processing. The T-Faster RCNN outperforms RCNN and Fast RCNN in terms of accuracy. The execution time is also less as compared to RCNN and Fast RCNN. It takes 0.9 seconds to detect a single image that can be reduced by accelerating GPU and by removing some complexity from the model. The execution time of FasterRCNN is 0.03 seconds as compared to 1.73 seconds for Fast RCNN and 2.893 for RCNN respectively. The results suggest that the proposed model is a good candidate for real-time traffic sign detections with hardware improvements. Transfer learning improves the performance of the model and reduces errors. The low visibility scenarios with different blur levels show a significant impact on the performance of the models. There is a 35% decrease in the accuracy with the 5% of blurriness. The thesis showcases the effect of blurriness on the accuracy, mse, and precision values as well.

5.2 Future Work

The scope of improvement is the key to development. The T-Faster RCNN can be extended in future research.

- The experiment performs with 30 epochs with a batch size of 32 to reduce the experimental time. However, if we raise the number of epochs to 1000, the model training will improve. This will increase the accuracy of the model.
- The experiment scrutinizes the proposed model against one dataset only. It is interesting to see how the model will perform for a more robust, versatile, and large dataset that has more classes. This will test the scalability of the model.
- Implementing a two-staged detection in the positioning stage of Faster RCNN can improve the loss function based on intersection over Union (IoU) for the regression layer and using bilinear interpolation to improve the Region of Interest (ROI) polling operation can help in the better performance on the model.
- Using high-definition videos instead of image data for the YOLO approach can also be an alternative path to our work. It can formulate a model which is more practically ready.
- GPU acceleration for T-Faster RCNN can reduce the processing time.

References

Zou, Z., Shi, Z., Guo, Y. and Ye, J., 2019. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*.

Pathak, A.R., Pandey, M. and Rautaray, S., 2018. Application of deep learning for object detection. *Procedia computer science*, *132*, pp.1706-1717.

Tavallali, P., Yazdi, M. and Khosravi, M.R., 2017, December. An efficient training procedure for viola-jones face detector. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 828-831

Ajitha, P.V. and Nagra, A., 2021. An Overview of Artificial Intelligence in Automobile Industry–A Case Study on Tesla Cars. *Solid State Technology*, *64*(2), pp.503-512.

Gupta, A., Anpalagan, A., Guan, L. and Khwaja, A.S., 2021. Deep Learning for Object Detection and Scene Perception in Self-Driving Cars: Survey, Challenges, and Open Issues. *Array*

Ng, T.S., 2021. ADAS in Autonomous Driving. In *Robotic Vehicles: Systems and Technology* Springer, Singapore, pp. 87-93

Pearl, T.H., 2017. Fast & furious: the misregulation of driverless cars. *NYU Ann. Surv. Am. L.*73, pp.19-21.

Lee, C., Kim, H.J. and Oh, K.W., 2016, October. Comparison of faster R-CNN models for object detection. In *2016 16th international conference on control, automation and systems (iccas)*, pp. 107-110

Rajpal, S., Lakhyani, N., Singh, A.K., Kohli, R. and Kumar, N., 2021. Using handpicked features in conjunction with ResNet-50 for improved detection of COVID-19 from chest X-ray images. *Chaos, Solitons & Fractals*, *145*

Ibrahim, N.M., Abou ElFarag, A. and Kadry, R., 2021. Gaussian Blur through Parallel Computing. In *IMPROVE*, pp. 175-179

Malini, A., Priyadharshini, P. and Sabeena, S., An automatic assessment of road condition from aerial imagery using modified VGG architecture in faster-RCNN framework. *Journal of Intelligent & Fuzzy Systems*, (Preprint), pp.1-12.

Kaur, P. and Mittal, P., 2021. A Comparative Study of LBPH, SIFT and SURF Algorithms for Face Recognition Task.

Koresh, M.H.J.D. and Deva, J., 2019. Computer vision based traffic sign sensing for smart transport. Journal of Innovative Image Processing (JIIP), 1(01), pp.11-19.

Cao, J., Song, C., Peng, S., Xiao, F. and Song, S. (2019). Improved Traffic Sign Detection and Recognition Algorithm for Intelligent Vehicles. Sensors, 19(18)

Yang, Y., Luo, H., Xu, H. and Wu, F., 2015. Towards real-time traffic sign detection and classification. IEEE Transactions on Intelligent transportation systems, 17(7), pp.2022-2031.

Gavrilescu, R., Zet, C., Foșalău, C., Skoczylas, M. and Cotovanu, D., 2018, October. Faster R-CNN: an approach to real-time object detection. In 2018 International Conference and Exposition on Electrical And Power Engineering (EPE), pp. 0165-0168

Huang, W., Huang, M. and Zhang, Y., 2018, August. Detection of traffic signs based on a combination of GAN and faster-RCNN. In Journal of Physics: Conference Series (Vol. 1069, No. 1, p. 012159)

Y. Yang, H. Luo, H. Xu and F. Wu, "Towards Real-Time Traffic Sign Detection and Classification," in IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 7, pp. 2022-2031, July 2016, doi: 10.1109/TITS.2015.2482461.

Sowjanya Palavanchu (2021) "Transfer Learning Models for Traffic Sign Recognition System", Annals of the Romanian Society for Cell Biology, 25(2), pp. 3477–3489.

Rezapour, M. & Ksaibati, K., 2021. Convolutional Neural Network for Roadside Barriers Detection: Transfer Learning versus Non-Transfer Learning. Signals, 2(1), pp.72–86.

Shi, J.H. and Lin, H.Y., 2017, June. A vision system for traffic sign detection and recognition. In 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE), pp. 1596-1601

Haifeng Wan, Lei Gao, Manman Su, Qinglong You, Hui Qu, Qirun Sun, "A Novel Neural Network Model for Traffic Sign Detection and Recognition under Extreme Conditions", Journal of Sensors, vol. 2021, Article ID 9984787, 16 pages, 2021.

Shustanov, A. and Yakimov, P. (2017). CNN Design for Real-Time Traffic Sign Recognition. Procedia Engineering, 201, pp.718–725.

Lim, K., Hong, Y., Choi, Y. and Byun, H., 2017. Real-time traffic sign recognition based on a general purpose GPU and deep-learning. PLoS one, 12(3)

A. Santos, P.A. Abu, C. Oppus, R. Reyes, 2020. "Real-Time Traffic Sign Detection and Recognition System for Assistive Driving", Advances in Science, Technology and Engineering Systems Journal, vol. 5, no. 4, pp. 600-611.

Liu, T., Chen, Z., Yang, Y., Wu, Z. and Li, H., 2020, May. Lane detection in low-light conditions using an efficient data enhancement: Light conditions style transfer. In 2020 IEEE Intelligent Vehicles Symposium (IV) (pp. 1394-1399)

B. Liu, W. Zhao and Q. Sun, "Study of object detection based on Faster R-CNN," 2017 Chinese Automation Congress (CAC), 2017, pp. 6233-6236.

Jiang, Z., Zhao, L., Li, S. and Jia, Y. 2020. Real-time object detection method based on improved YOLOv4-tiny. arXiv:2011.04244 [cs].

Y. Xiao, A. Jiang, J. Ye and M. Wang, 2020. "Making of Night Vision: Object Detection Under Low-Illumination," in IEEE Access, vol. 8, pp. 123075-123086.

Fan, Q., Zhuo, W., Tang, C.K. and Tai, Y.W., 2020. Few-shot object detection with attention-RPN and multi-relation detector. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4013-4022).

Kumari, P. and KR, S., 2021. An optimal feature enriched region of interest (ROI) extraction for periocular biometric system. Multimedia Tools and Applications, pp.1-19.

Gao, F., Li, B., Chen, L., Shang, Z., Wei, X. and He, C., 2021. A softmax classifier for high-precision classification of ultrasonic similar signals. Ultrasonics, 112, p.106344.

Hussein, A.Y., Falcarin, P. and Sadiq, A.T., 2021. Enhancement performance of random forest algorithm via one hot encoding for IoT IDS. Periodicals of Engineering and Natural Sciences (PEN), 9(3), pp.579-591.

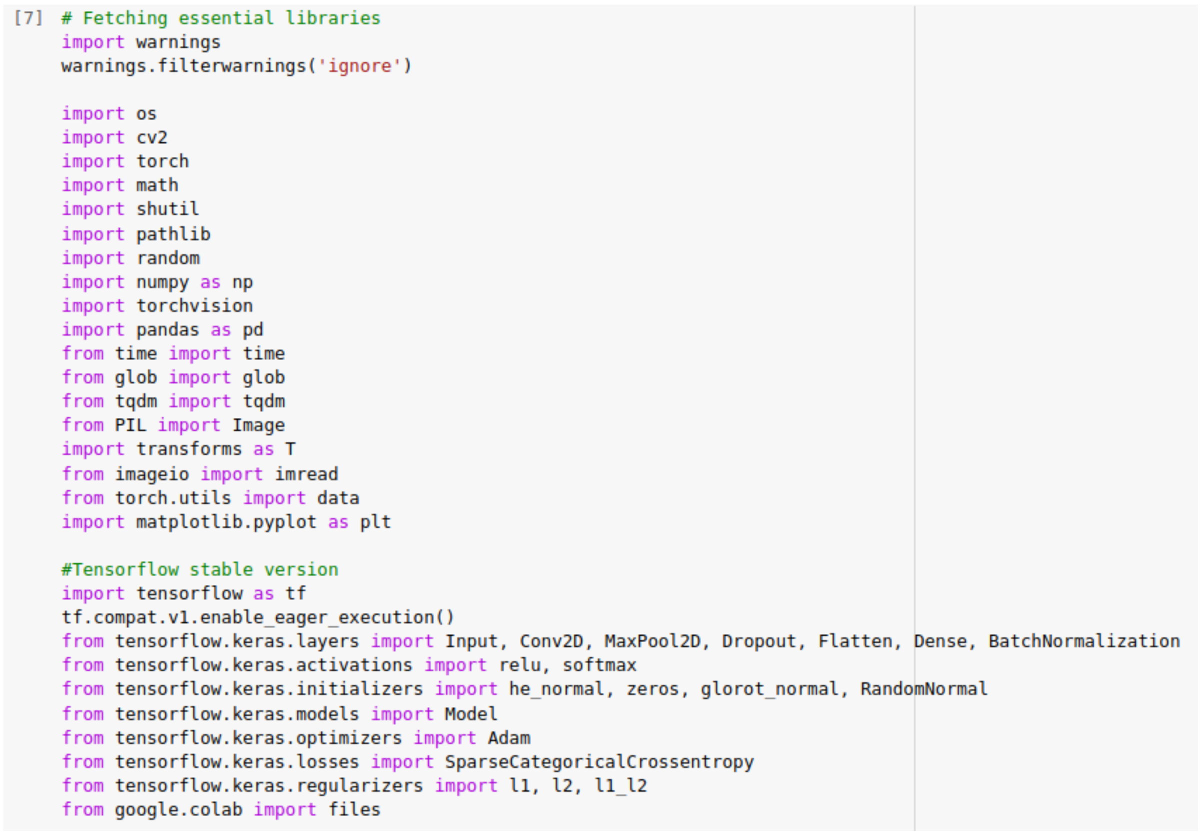
Fu, L., Majeed, Y., Zhang, X., Karkee, M. and Zhang, Q., 2020. Faster R–CNN–based apple detection in dense-foliage fruiting-wall trees using RGB and depth features for robotic harvesting. Biosystems Engineering, 197, pp.245-256.

Saura, J.R., 2021. Using data sciences in digital marketing: Framework, methods, and performance metrics. Journal of Innovation & Knowledge, 6(2), pp.92-102.

Yacouby, R. and Axman, D., 2020, November. Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models. In Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems (pp. 79-91).

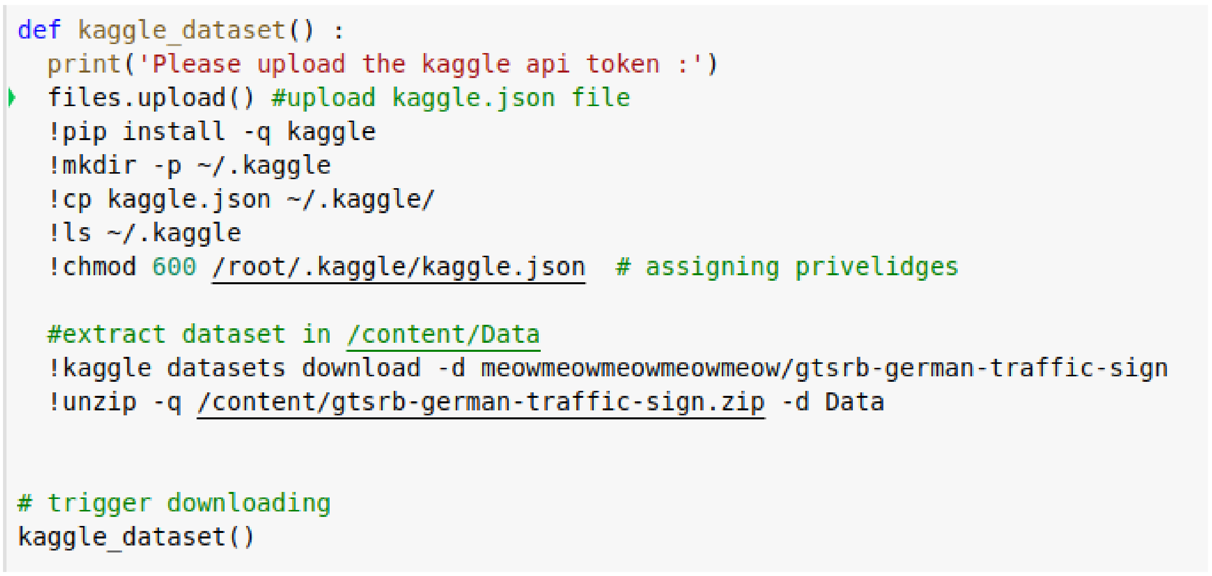
Appendix A: Code Listing

Imported Libraries



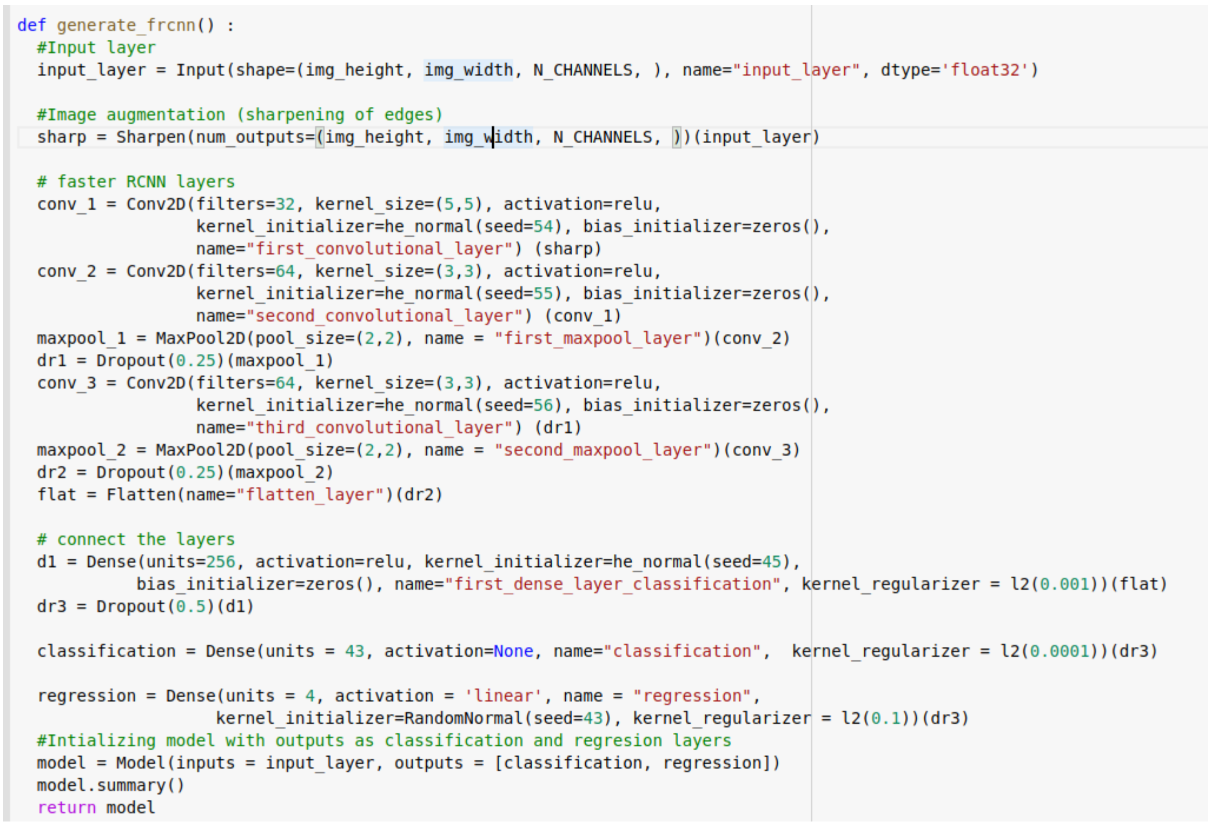
```
# Fetching essential libraries
import warnings
warnings.filterwarnings('ignore')

import os
import cv2
import torch
import math
import shutil
import pathlib
import random
import numpy as np
import torchvision
import pandas as pd
from time import time
from glob import glob
from tqdm import tqdm
from PIL import Image
import transforms as T
from imageio import imread
from torch.utils import data
import matplotlib.pyplot as plt

#Tensorflow stable version
import tensorflow as tf
tf.compat.v1.enable_eager_execution()
from tensorflow.keras.layers import Input, Conv2D, MaxPool2D, Dropout, Flatten, Dense, BatchNormalization
from tensorflow.keras.activations import relu, softmax
from tensorflow.keras.initializers import he_normal, zeros, glorot_normal, RandomNormal
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.regularizers import l1, l2, l1_l2
from google.colab import files
```

Dataset Downloading



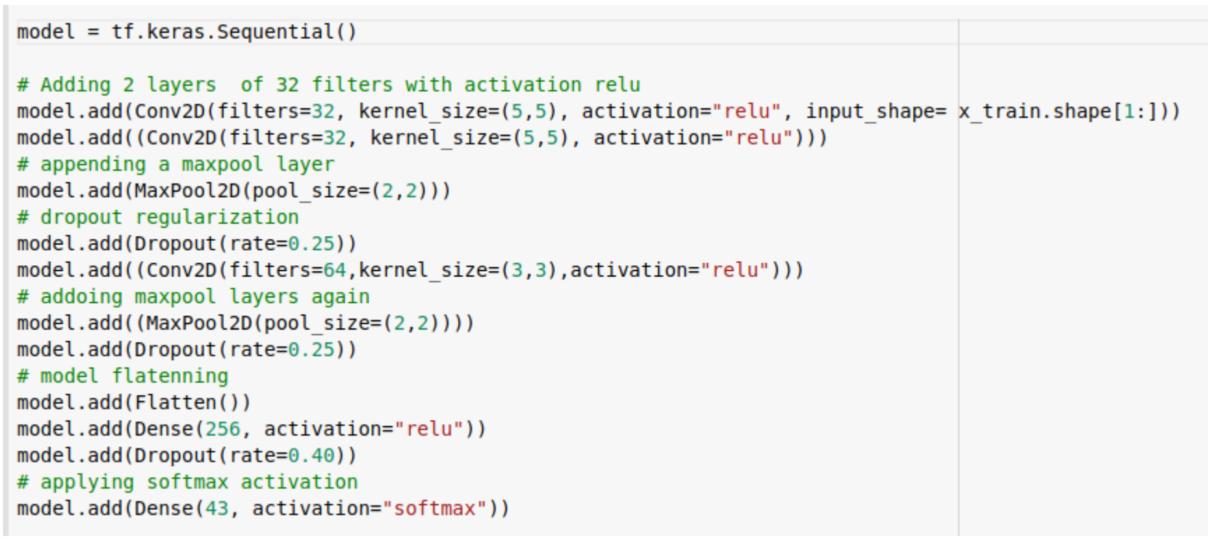
```
def kaggle_dataset() :
  print('Please upload the kaggle api token :')
  files.upload() #upload kaggle.json file
  !pip install -q kaggle
  !mkdir -p ~/.kaggle
  !cp kaggle.json ~/.kaggle/
  !ls ~/.kaggle
  !chmod 600 /root/.kaggle/kaggle.json  # assigning privelidges

  #extract dataset in /content/Data
  !kaggle datasets download -d meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
  !unzip -q /content/gtsrb-german-traffic-sign.zip -d Data


# trigger downloading
kaggle_dataset()
```

FasterRCNN Model



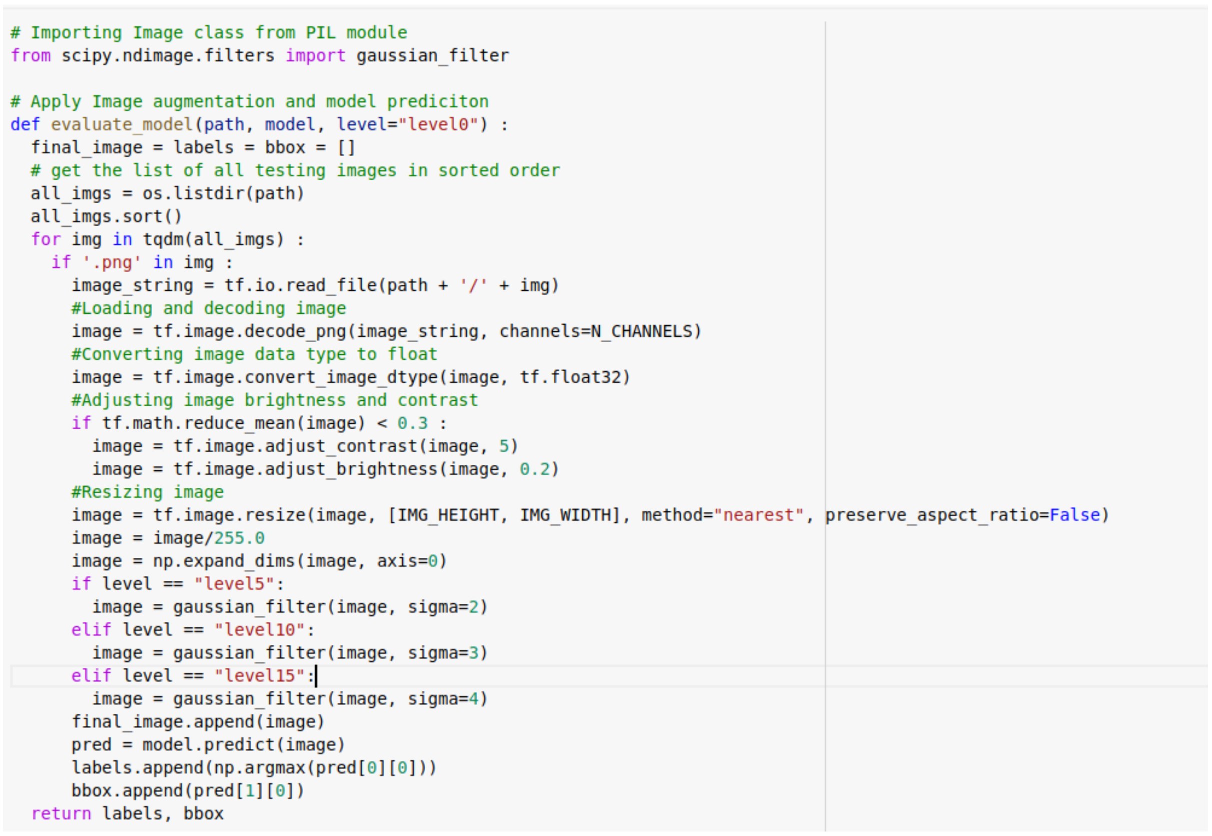
```
def generate_frcnn() :
  #Input layer
  input_layer = Input(shape=(img_height, img_width, N_CHANNELS, ), name="input_layer", dtype='float32')

  #Image augmentation (sharpening of edges)
  sharp = Sharpen(num_outputs=(img_height, img_width, N_CHANNELS, ))(input_layer)

  # faster RCNN layers
  conv_1 = Conv2D(filters=32, kernel_size=(5,5), activation=relu,
                  kernel_initializer=he_normal(seed=54), bias_initializer=zeros(),
                  name="first_convolutional_layer") (sharp)
  conv_2 = Conv2D(filters=64, kernel_size=(3,3), activation=relu,
                  kernel_initializer=he_normal(seed=55), bias_initializer=zeros(),
                  name="second_convolutional_layer") (conv_1)
  maxpool_1 = MaxPool2D(pool_size=(2,2), name = "first_maxpool_layer")(conv_2)
  dr1 = Dropout(0.25)(maxpool_1)
  conv_3 = Conv2D(filters=64, kernel_size=(3,3), activation=relu,
                  kernel_initializer=he_normal(seed=56), bias_initializer=zeros(),
                  name="third_convolutional_layer") (dr1)
  maxpool_2 = MaxPool2D(pool_size=(2,2), name = "second_maxpool_layer")(conv_3)
  dr2 = Dropout(0.25)(maxpool_2)
  flat = Flatten(name="flatten_layer")(dr2)

  # connect the layers
  d1 = Dense(units=256, activation=relu, kernel_initializer=he_normal(seed=45),
            bias_initializer=zeros(), name="first_dense_layer_classification", kernel_regularizer = l2(0.001))(flat)
  dr3 = Dropout(0.5)(d1)

  classification = Dense(units = 43, activation=None, name="classification",  kernel_regularizer = l2(0.0001))(dr3)

  regression = Dense(units = 4, activation = 'linear', name = "regression",
                    kernel_initializer=RandomNormal(seed=43), kernel_regularizer = l2(0.1))(dr3)
  #Intializing model with outputs as classification and regresion layers
  model = Model(inputs = input_layer, outputs = [classification, regression])
  model.summary()
  return model
```

RCNN Model



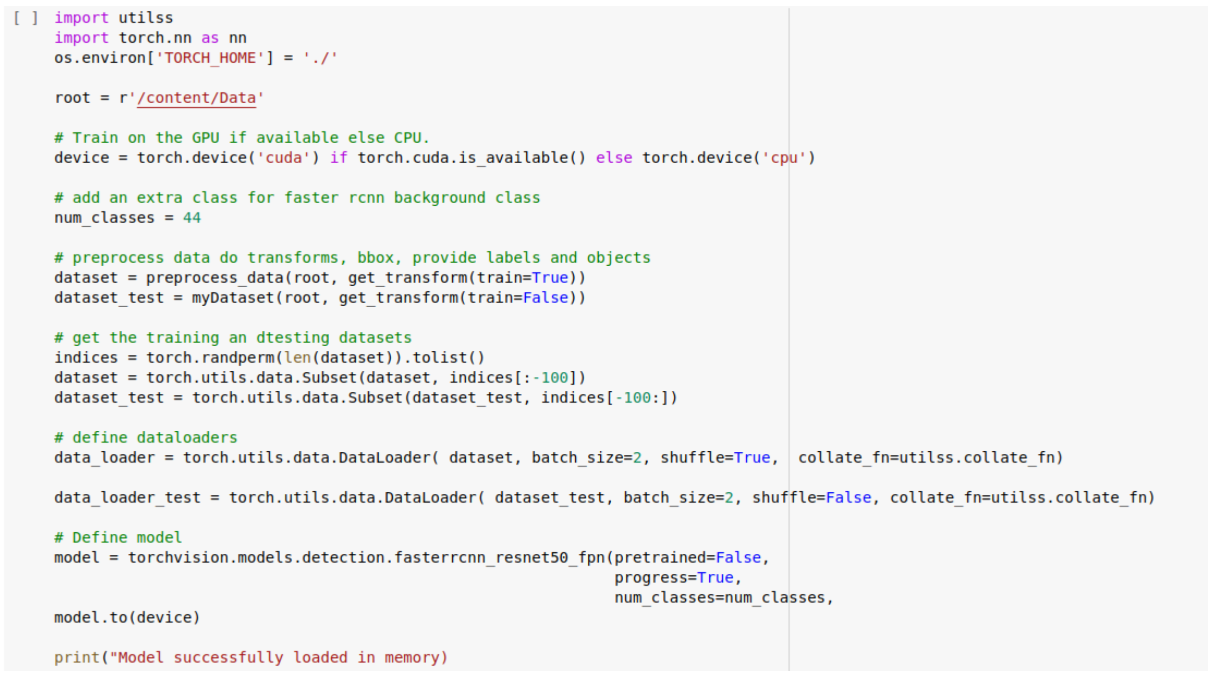
```
model = tf.keras.Sequential()

# Adding 2 layers  of 32 filters with activation relu
model.add(Conv2D(filters=32, kernel_size=(5,5), activation="relu", input_shape= x_train.shape[1:]))
model.add((Conv2D(filters=32, kernel_size=(5,5), activation="relu")))
# appending a maxpool layer
model.add(MaxPool2D(pool_size=(2,2)))
# dropout regularization
model.add(Dropout(rate=0.25))
model.add((Conv2D(filters=64,kernel_size=(3,3),activation="relu")))
# addoing maxpool layers again
model.add((MaxPool2D(pool_size=(2,2))))
model.add(Dropout(rate=0.25))
# model flatenning
model.add(Flatten())
model.add(Dense(256, activation="relu"))
model.add(Dropout(rate=0.40))
# applying softmax activation
model.add(Dense(43, activation="softmax"))
```

Image Augmentation



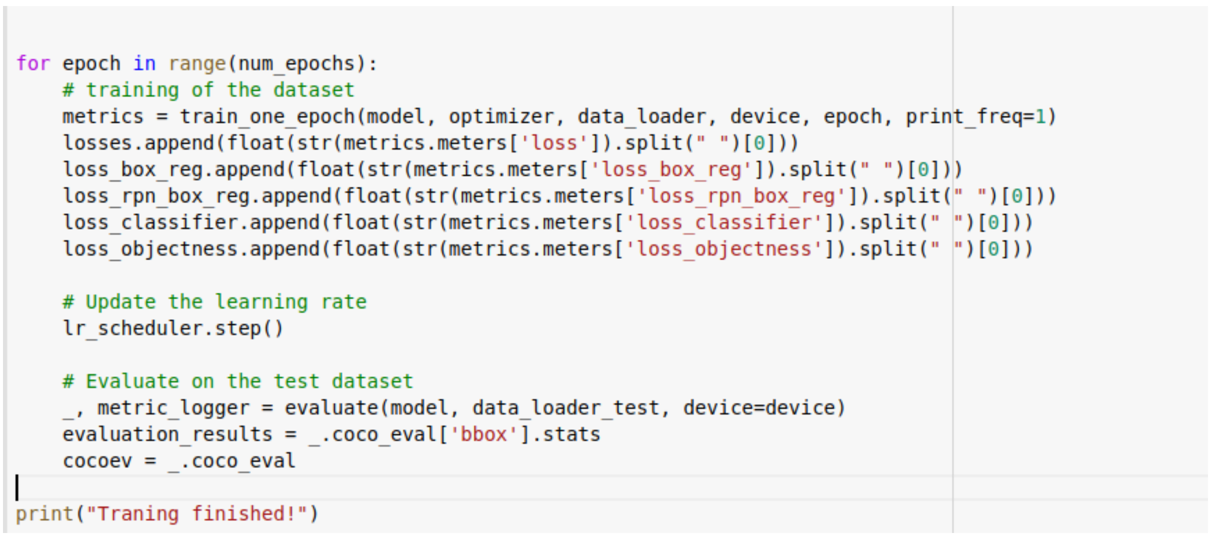
```
# Importing Image class from PIL module
from scipy.ndimage.filters import gaussian_filter

# Apply Image augmentation and model prediciton
def evaluate_model(path, model, level="level0") :
  final_image = labels = bbox = []
  # get the list of all testing images in sorted order
  all_imgs = os.listdir(path)
  all_imgs.sort()
  for img in tqdm(all_imgs) :
    if '.png' in img :
      image_string = tf.io.read_file(path + '/' + img)
      #Loading and decoding image
      image = tf.image.decode_png(image_string, channels=N_CHANNELS)
      #Converting image data type to float
      image = tf.image.convert_image_dtype(image, tf.float32)
      #Adjusting image brightness and contrast
      if tf.math.reduce_mean(image) < 0.3 :
        image = tf.image.adjust_contrast(image, 5)
        image = tf.image.adjust_brightness(image, 0.2)
      #Resizing image
      image = tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH], method="nearest", preserve_aspect_ratio=False)
      image = image/255.0
      image = np.expand_dims(image, axis=0)
      if level == "level5":
        image = gaussian_filter(image, sigma=2)
      elif level == "level10":
        image = gaussian_filter(image, sigma=3)
      elif level == "level15":
        image = gaussian_filter(image, sigma=4)
      final_image.append(image)
      pred = model.predict(image)
      labels.append(np.argmax(pred[0][0]))
      bbox.append(pred[1][0])
  return labels, bbox
```

T-FasterRCNN Model



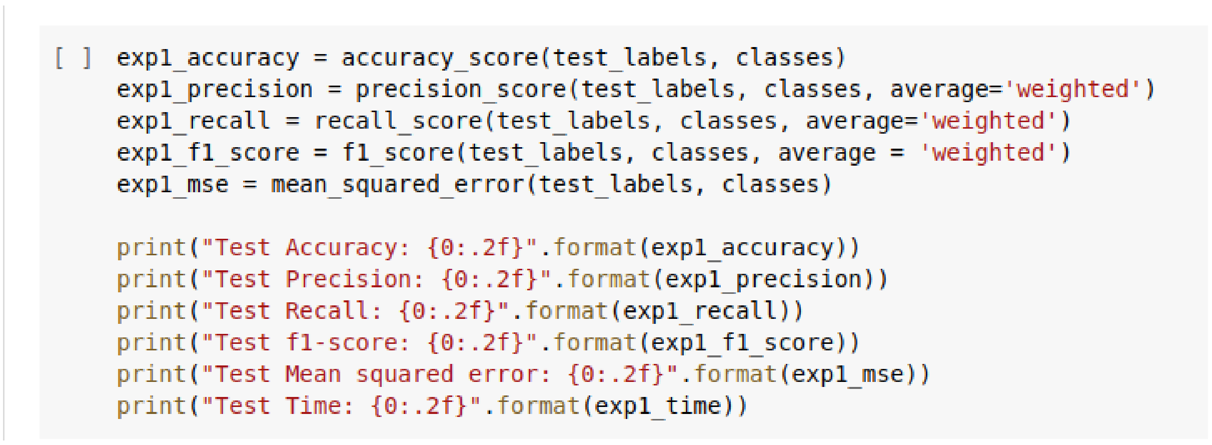
```
[ ]  import utilss
     import torch.nn as nn
     os.environ['TORCH_HOME'] = './'

     root = r'/content/Data'

     # Train on the GPU if available else CPU.
     device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

     # add an extra class for faster rcnn background class
     num_classes = 44

     # preprocess data do transforms, bbox, provide labels and objects
     dataset = preprocess_data(root, get_transform(train=True))
     dataset_test = myDataset(root, get_transform(train=False))

     # get the training an dtesting datasets
     indices = torch.randperm(len(dataset)).tolist()
     dataset = torch.utils.data.Subset(dataset, indices[:-100])
     dataset_test = torch.utils.data.Subset(dataset_test, indices[-100:])

     # define dataloaders
     data_loader = torch.utils.data.DataLoader( dataset, batch_size=2, shuffle=True,  collate_fn=utilss.collate_fn)

     data_loader_test = torch.utils.data.DataLoader( dataset_test, batch_size=2, shuffle=False, collate_fn=utilss.collate_fn)

     # Define model
     model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False,
                                                                  progress=True,
                                                                  num_classes=num_classes,

     model.to(device)

     print("Model successfully loaded in memory)
```

Training Evaluation T-FasterRCNN



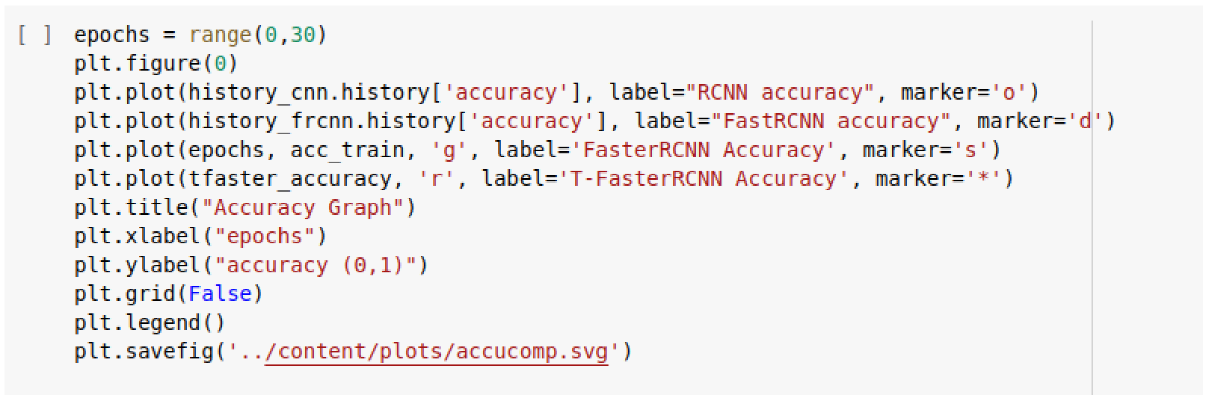
```
for epoch in range(num_epochs):
    # training of the dataset
    metrics = train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=1)
    losses.append(float(str(metrics.meters['loss']).split(" ")[0]))
    loss_box_reg.append(float(str(metrics.meters['loss_box_reg']).split(" ")[0]))
    loss_rpn_box_reg.append(float(str(metrics.meters['loss_rpn_box_reg']).split(" ")[0]))
    loss_classifier.append(float(str(metrics.meters['loss_classifier']).split(" ")[0]))
    loss_objectness.append(float(str(metrics.meters['loss_objectness']).split(" ")[0]))

    # Update the learning rate
    lr_scheduler.step()

    # Evaluate on the test dataset
    _, metric_logger = evaluate(model, data_loader_test, device=device)
    evaluation_results = _.coco_eval['bbox'].stats
    cocoev = _.coco_eval

print("Traning finished!")
```

Testing Results



```
[ ] exp1_accuracy = accuracy_score(test_labels, classes)
    exp1_precision = precision_score(test_labels, classes, average='weighted')
    exp1_recall = recall_score(test_labels, classes, average='weighted')
    exp1_f1_score = f1_score(test_labels, classes, average = 'weighted')
    exp1_mse = mean_squared_error(test_labels, classes)

    print("Test Accuracy: {0:.2f}".format(exp1_accuracy))
    print("Test Precision: {0:.2f}".format(exp1_precision))
    print("Test Recall: {0:.2f}".format(exp1_recall))
    print("Test f1-score: {0:.2f}".format(exp1_f1_score))
    print("Test Mean squared error: {0:.2f}".format(exp1_mse))
    print("Test Time: {0:.2f}".format(exp1_time))
```

Training Plots



```
[ ] epochs = range(0,30)
    plt.figure(0)
    plt.plot(history_cnn.history['accuracy'], label="RCNN accuracy", marker='o')
    plt.plot(history_frcnn.history['accuracy'], label="FastRCNN accuracy", marker='d')
    plt.plot(epochs, acc_train, 'g', label='FasterRCNN Accuracy', marker='s')
    plt.plot(tfaster_accuracy, 'r', label='T-FasterRCNN Accuracy', marker='*')
    plt.title("Accuracy Graph")
    plt.xlabel("epochs")
    plt.ylabel("accuracy (0,1)")
    plt.grid(False)
    plt.legend()
    plt.savefig('../content/plots/accucomp.svg')
```

Testing Plots

```
[ ] labels = ['RCNN', 'FastRCNN', 'FasterRCNN', 'T-FasterRCNN']
    precision =  [exp1_precision, exp2_precision, exp3_precision, tfaster_precision ]


    x = np.arange(len(labels))  # the label locations
    width = 0.50  # the width of the bars

    fig, ax = plt.subplots()
    rects1 = ax.bar(x, precision, width, label='Precision', color=[ 'red', 'green', 'blue', 'yellow'], hatch=['*', 'x', 'o', '.'])

    # Add some text for labels, title and custom x-axis tick labels, etc.
    ax.set_ylabel('Precision')
    ax.set_title('Precision among different models')
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.legend()

    ax.bar_label(rects1, padding=0)

    fig.tight_layout()
    plt.savefig('/content/plots/comparison2.svg')
    plt.show()
```