CAPSTONE PROJECT FINAL REPORT

CANADIAN TRAFFIC SIGN DETECTION & RECOGNITION USING OPENCV AND KERAS

A Capstone Project

Submitted to the Faculty through the Zekelman School of Business & Information Technology in Partial Fulfillment of the Requirements for the Ontario College Graduate Certificate at the St. Clair College of Applied Arts & Technology

Windsor, Ontario, Canada

Guided By: Prof. Osama Hamzeh; B.Sc. (Comp. Science), M.Sc. (Comp. Science), PhD

Submitted By:

Meghaben Bhavik Patel: 0749147

Vipin Malik: 0755900

Bishaw Mitra Barua: 0753841

DISCLAIMER

We declare that, to the best of our knowledge, our project does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in our project, published or otherwise, are fully acknowledged in accordance with the standard referencing practices.

We hereby declare that this capstone project reflects our original work and has not been submitted before to any institution or organisation for assessment or commercial purposes. Further, we have acknowledged all sources used and have cited these in the reference section. We understand that our project may be made electronically available to the public.

Meghaben Bhavik Patel	: 0749147
Vipin Malik	: 0755900
Bishaw Mitra Barua	: 0753841

Date: 23rd April 2021

Abstract

There has been an increasing demand for autonomous vehicles in recent years. As a result, the inclusion of a Traffic Sign Detection and Recognition (TSDR) device was highlighted. Unintentionally and often deliberately, drivers fail to detect and observe traffic signals, resulting in traffic accidents. It is important for autonomous vehicles to identify and recognise traffic signs because the intelligent device protects the driver from collisions, provides directions, and reduces travel time by rerouting, among other things. When traffic laws are broken, current road monitoring cameras and their viewing software will simply capture photographs. These images are then used to obtain licence number plates and driver identification. Traffic sign recognition is a form of object detection in which computer vision detects and recognises road signs and alerts the driver to make timely driving decisions. Traffic signals play an important role in our daily lives. Drivers and other road users benefit from the information provided by traffic signs. They reflect the broad guidelines that govern safe and informed driving. Without traffic signals, drivers will have no idea what lies ahead of them, making road travel a nightmare and potentially dangerous. The goal of the onboard traffic sign recognition system is to correctly identify signs over time, allowing the driver to respond and manoeuvre much more quickly to incoming traffic situations, or to slow down their vehicle when there is a possible danger ahead. Recognition errors, judgement errors, and performance errors are the most common types of driving errors that result in road accidents. As a result, Advanced Driver Assistance Systems (ADAS) were required to provide drivers with essential information about road and traffic conditions, monitor repetitive and complex activities, and ensure the overall safety of drivers and pedestrians. As a result, automated vehicles began to use Traffic Sign Detection and Recognition (TSDR) systems. The signals in human understandable perception of recognised signs are the performance of Traffic sign Recognition, which takes either a live video stream or a series of sign images as data.

INTRODUCTION

Initially while conducting our research, we have tried to use the German Traffic Sign Dataset which contained more than 50,000 images spread over 40 different traffic/road signs. The images were all in .png format and coloured. Hence, it is a RGB character (3 channel) dataset. Through our research we have found that if the input data images are in Gray scale (1 channel) image format then then accuracy of classification get enhanced significantly. This conversion to Gray scale image can be performed by using ImageDataGenerator from the Keras library or can also be manually converted using freely available online web tools. In our dataset we have used the web tools for the colour conversion. Our project charter as guided by our instructor required Canadian dataset exclusively. To implement our developed models, it was needed that the input images relate to the Canadian Traffic and road signs. Hence, we started looking for Canadian traffic sign images. We came across one dataset from Kaggle which had around 42 classes of signs claiming to be Canadian road signs. But, on detailed examination of the input images we found that most of the traffic/road signs has been taken from the German traffic sign dataset. Hence, we had to change our dataset again. Finally, with our team effort we have taken 1969 RGB images over 36 different classes by manually capturing photos from the streets of Windsor, Ontario. All these images were matching the traffic/road sign images as published in Ontario (MTO) Driver’s Handbook. As mentioned earlier it was needed that the RGB images be converted into Gray scale images. Hence, we have transformed these 1969 input images to 1874 Gray scale images using free online photo converter. Also, to ensure that all the input image format be the same, we have used the online tool available from https://convertio.co/files to convert them into jpg/jpeg format.

We have used Convolutional Neural Network (CNN) and ResNet-50 model from deep learning to classify these images. Any CNN model needs huge number of input images to train the model appropriately. Hence there was a need to augment the images and create as many as 10,000 images for a comprehensive and generalized model. In such cases the ImageDataGenerator from the Keras library provides support. Image augmentation is a method of altering original images by adding various transformations to them, resulting in several transformed copies of the same image. However, depending on the augmentation techniques you use, such as moving, spinning, and flipping, each copy is unique in certain ways. Applying these minor changes to the original picture does not alter its target class; rather, it offers a fresh perspective on capturing the object in real life. As a result, we often employ it in the creation of deep learning models. These image augmentation techniques not only increase the size of your dataset, but they also add a degree of variation to it, allowing the model to generalise better on data that has not been seen before. When the model is trained on fresh, slightly altered images, it becomes more robust. Next, is we train our models based on the augmented images. We have built three CNN models, each varying by tweaking the model hyperparameters. For example, by changing the number of filters, kernel size, activation functions, input shape, changing the number of epochs, etc. A model hyperparameter is a configuration that is not part of the model and cannot be estimated from data. On a given problem, we cannot know what the best value for a model hyperparameter is. We may use rules of thumb, copy values from other problems, or use trial and error to find the best value. As our second and final model we have used ResNet-50 model for a comparative analysis with that to the CNN model. CNN model is very popular in object detection technique with high accuracy in classification projects such as image recognition, video analysis, Natural Language Processing (NLP), Anomaly Detection, etc. Hence, quite naturally we have chosen the CNN model to be used first in our project. Thereafter we studied about the ResNet-50 model and evaluated its usage convenience for our project. As per our research, ResNet-50 is quite often more accurate in classifying images compared to the CNN model. Later, in this report we have highlighted its functions and its advantages.

The nature of this project called for a Graphical User Interface (GUI) to implement our best model. Hence, we have used Tkinter and Streamlit as our GUIs to showcase the accuracy of our best performing model, wherein we upload any Canadian traffic/road sign image, and the saved model will detect and recognise the symbol and will print its accuracy.

Related Work

We have made the CNN model as our benchmark model as for image classification and object detection problems CNN has been historically recognised as a proven neural network with gives high accuracy. Also, we have set the benchmark accuracy at 85% since our dataset are real time datasets.

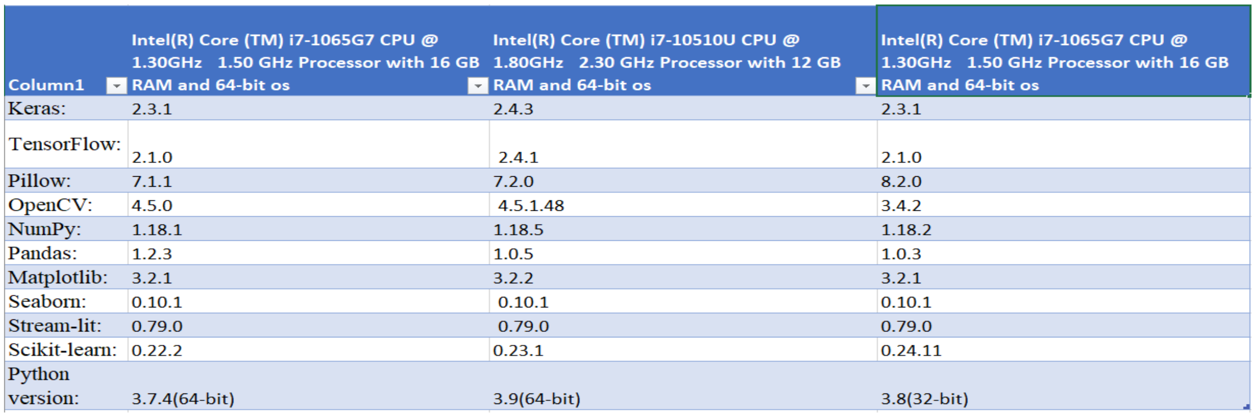
Traditional machine learning methods (such as multilayer perception machines, support vector machines, etc.) mostly use shallow structures to deal with a limited number of samples and computing units. When the target objects have rich meanings, the performance and generalization ability of complex classification problems are obviously insufficient. The convolution neural network (CNN) developed in recent years has been widely used in the field of image processing because it is good at dealing with image classification and recognition problems and has brought great improvement in the accuracy of many machine learning tasks. It has become a powerful and universal deep learning model.

Previous traditional machine learning models like multilayer perception machines, Support Vector Machines (SVM), etc. in most cases use very simple structures to work with less samples and computing units. When the input data carries more information, the performance and the generalization capacity of complex classification problems are inadequate. In the wide field of image processing the Convolutional Neural Network (CNN), developed in recent times, has brought significant changes in improving the accuracy of the machine learning models. CNN has become a powerful model in the field of deep learning.

CNN's analysis is still ongoing, and there is a lot of room for progress. The major changes in CNN results are typically found to have occurred between 2015 and 2019. The depth of a CNN determines its representational power, so an enriched feature set ranging from simple to complex abstractions will aid in learning complex problems. Deep architectures, on the other hand, face a major challenge in the form of diminishing gradients. Researchers initially attempted to alleviate the problem by connecting intermediate layers to auxiliary learners. The creation of new connections to increase the convergence rate of deep CNN architectures was the most prominent field of research in 2015. Various concepts such as information gating through multiple layers, skip connections, and cross-layer channel communication were implemented in this regard. Various experimental studies have shown that state-of-the-art deep architectures such as VGG, ResNet, ResNext, and others can solve difficult recognition and localization problems such as semantic and instance-based object segmentation, scene parsing, and scene position. Most well-known object detection and segmentation architectures, including Single Shot Multibox Detector (SSD), Region-based CNN (R-CNN), Faster R-CNN, Mask R-CNN, and Fully Convolutional Neural Network (FCN), are based on ResNet, VGG, and Inception, among others. Similarly, several interesting detection algorithms, such as Feature Pyramid Networks, Cascade R-CNN, Libra R-CNN, and others, improved performance by modifying the architectures described earlier. By merging deep CNN with recurrent neural networks (RNN), deep CNN applications were applied to image captioning, yielding state-of-the-art results on the MS COCO-2015 image captioning challenge. In 2016, it was discovered that stacking multiple transformations not only in depth but also in parallel resulted in strong learning for complex problems. To improve deep CNN performance, different researchers used a combination of the previously proposed architectures. Researchers in 2017 concentrated their efforts on developing standardised blocks that can be inserted at any learning stage in CNN architecture to increase network representation. One of the fastest-growing areas of research in CNN is the development of new blocks, which are used to assign attention to spatial and feature-map (channel) information. Khan et al. (Khan et al 2018a) proposed a new concept of channel boosting in 2018 to improve the efficiency of a CNN by learning distinct features and leveraging the already learned features through the concept of TL. The high computing cost and memory demand are two major issues with deep and wide architectures. As a result, deploying state-of-the-art large and deep CNN models in resource-constrained environments is extremely difficult. Conventional convolution operations necessitate a large number of multiplications, which increases inference time and limits the applicability of CNN to applications with limited memory and time. Many real-world technologies, such as autonomous vehicles, robots, healthcare, and mobile apps, perform tasks that must be completed in a timely manner on computationally constrained platforms. As a result, various changes to CNN are made to make them suitable for resource-constrained environments. Information distillation, training of small networks, and squeezing of pre-trained networks are all notable modifications (such as pruning, quantization, hashing, Huffman coding, etc.). GoogleNet took advantage of the concept of small networks, which replaces traditional convolution with a computationally efficient point-wise group convolution process. ShuffleNet, too, used point-wise group convolution, but with a modern concept of channel shuffle that drastically reduces the number of operations while maintaining accuracy. ANTNet proposed a novel architectural block known as ANTBlock, which achieved good performance on benchmark datasets at a low computational cost. Furthermore, the trend is toward the creation of lightweight architectures that do not compromise performance, allowing CNN to be used on hardware with limited resources.

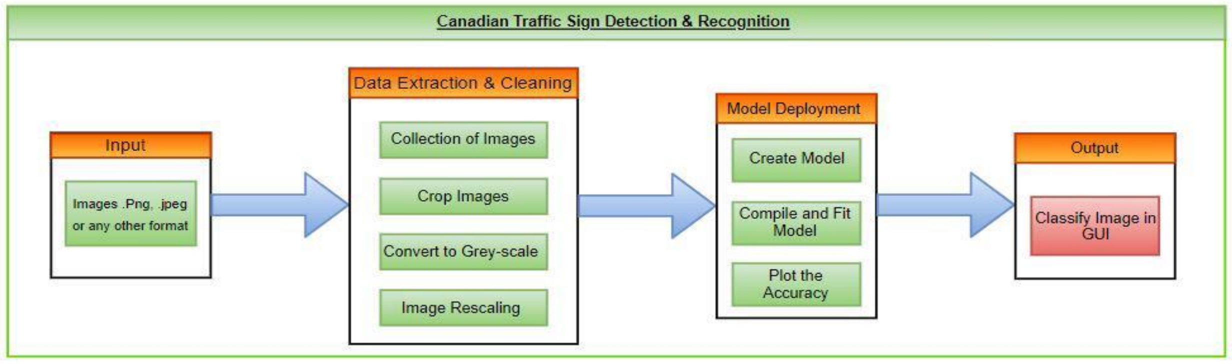
System and package Specifications:

In our project the Training, Testing and all the implementation of the model were performed in three Systems having a specification of Intel®Core™ i5-10210U CPU @ 1.60GHz 2.11 GHz processor with 8 GB RAM and 64-bit operating system for the first system and Intel(R) Core (TM) i7-10510U CPU @ 1.80GHz 2.30 GHz Processor with 12 GB RAM and 64-bit operating system specifications for the second system and last system have Intel(R) Core (TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz Processor with 16 GB RAM and 64-bit operating system.



Column1	Intel(R) Core (TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz Processor with 16 GB RAM and 64-bit os	Intel(R) Core (TM) i7-10510U CPU @ 1.80GHz 2.30 GHz Processor with 12 GB RAM and 64-bit os	Intel(R) Core (TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz Processor with 16 GB RAM and 64-bit os
Keras:	2.3.1	2.4.3	2.3.1
TensorFlow:	2.1.0	2.4.1	2.1.0
Pillow:	7.1.1	7.2.0	8.2.0
OpenCV:	4.5.0	4.5.1.48	3.4.2
NumPy:	1.18.1	1.18.5	1.18.2
Pandas:	1.2.3	1.0.5	1.0.3
Matplotlib:	3.2.1	3.2.2	3.2.1
Seaborn:	0.10.1	0.10.1	0.10.1
Stream-lit:	0.79.0	0.79.0	0.79.0
Scikit-learn:	0.22.2	0.23.1	0.24.11
Python version:	3.7.4(64-bit)	3.9(64-bit)	3.8(32-bit)

Work-Flow Chart:



METHODOLOGY:

Data Preparation:

For our training set we finalized our project on 36 traffic sign classes. All these road signs have been manually photographed form different angles and distances to incorporate variability. But to train the model we required more data, without noise and to overcome the issue of input image format we have cropped all 1874 images to accommodate only the traffic/road sign with as little noise as possible. Thereafter, we transformed all the images to jpg/jpeg format to incorporate homogeneity. Further, we have deleted those images which could not be transformed into jpg/jpeg format. To decrease the training time without sacrificing detection and classification accuracy, we plan to train our network only on grey-scale images. So as a result, we converted the balance of input images to grey-scale images for better accuracy of our intended models i.e., CNN and Resnet50. Then, we resize all the images to 30* 30 pixels, which is 1/9-th of the original size of the image.

Build three different model of CNN:

For training all three models, we used Adam as the optimizer, learning rate scheduler, and model weight save checkpoint.

1st Model:

In our first model, we use two different Activation functions one is '**Relu'** and second is '**SoftMax'** The number of **Dense layers** (feeds all outputs from the previous layer to all its neurons, each neuron providing one output to the next layer) used in our first model is 512 and 36 with a **Dropout rate** of 0.25. Then we also use **Maxpooling** operation with a pool size of (2,2). The input shape is the total number of images which is in our train data. This model has a **Batch_size** = 32and the number of **epochs** is 11 which is quite less but because we got the good result that is why we stopped this model on 11 epochs. Then, we used two different **Kernel size** (that is used to extract the features from the images) (3,3) and (5,5) and it is the most common choice of kernel size that is the reason we choose this size. Moreover, we used different values for our filter which is 16,32,64 and 128.At last, we use **loss= categorical_crossentropy** (which is a loss function used in multi class classification tasks where an example belongs to one out of many possible categories and the model needs to decide which one).

2nd Model:

In our second model, we used the same activation function which was **'Relu'** and **'SoftMax'** but with different **input shape** = (height, width,1). In this model we only use 512 **Dense layers** with 0.5 **Dropout rate**. Then we applied Maxpooling operation with a pool size of (2,2). The second model has a **Batch_size** of 100. We also increase the number of **epochs** from 11 to 30 in this model to get better result. In addition to this, we used only one **kernel size** (3,3) with **filter value** 16 and 32.

In this model we used different augmentation technique like **Rotation, Zoom, Horizontal, Vertical flip** and **Shear** to get the high accuracy. This model uses 10 as rotation_range means it will rotate the image 10 degree in any direction. Moreover, we use **Zoom** setting with a value of 0.15 means it will zoom up to this proportion. The input for Horizontal and Vertical Flip was False which means the image will not flip on any side. Then we use **Shear** (Randomly shear up to this angle) with value 0.15. Last but not the least we also used **Fill mode** with value 'nearest'. As a result of this value in Fill mode it will add the value of the nearest pixel of the original image. At last, for compiling the model we used **loss= categorical_crossentropy** and optimized = **'Adam'.**

3rd Model:

In our third model, we used the same Activation functions one is '**Relu'** and the second one is **'SoftMax'** because they perform well but we used different input shape. In this model we use the input_shape = (X_train. shape [1:]) with 512 and 36 **Dense layers** with 0.5 **Dropout rate**. Then we use **Maxpooling** operation with a pool size of (2,2) and a **Batch_size** of 64. We use different Batch size in all three models to check their accuracy. The number of epochs used in this model is 20 which is less than the number of epochs used in model 2nd. Last but not the least in this model we use two different **Kernel size** which is (5,5) and (3,3). For compiling the model, we use **'Adam'** as an optimizer and **'Categorical_crossentropy'** as a Loss metric.

Build ResNet50:

In resent50 model, we use Learning rate as 0.001 and Optimizer is Adam. In this model we use a batch size of 32 with 50 number of epochs. Loss metric is Categorical Cross Entropy. To train the model we call CSVLogger, ReduceLROnPlateau, Early Stopping (method that allow us to specify an arbitrary large number of training epochs and stop training once the model performance stop improving), Callback ModelCheckpoint. We used Early stopping for this model which give the best accuracy at 19 epochs, so it automatically stopes at that point.

We save the model using **model.save()**

GUI using Tkinter:

Tkinter is Python's basic GUI library. Python's de-facto basic GUI (Graphical User Interface) kit is Tkinter or Tk interface. This is an open-source framework that runs on Unix and Windows operating systems. It's one of the most straightforward and widely used methods for creating a Graphical user interface Python program.

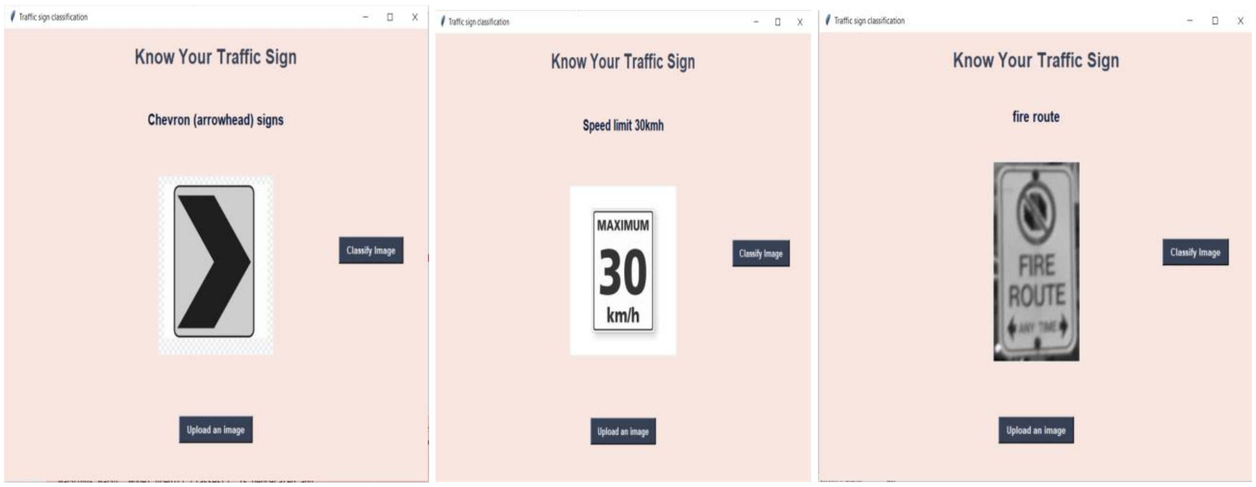
GUI using Streamlit:

Streamlit is a free and open-source app framework for Machine Learning and Data Science. It helps to create and share beautiful web applications for Machine Learning and Within few minutes we can build and deploy powerful data apps. It is not just only easy to learn but with the help of this framework, we do not need to worry about front end development.

Results:

The accuracy of each of the models differed by changing the parameters for each model. We can see in the table below that our best model is the 2nd CNN model which has not only good train accuracy score, but it has good validation score as compared to the other CNN and Resnet50 models. For the testing of our project, we chose the best model among the 4 models that we have built in our project and at the end we finalized to use a second CNN model for evaluating the testing data which we collected. We have also tried to use Streamlit library to test our model but we faced errors while loading the images from the test folder so we have used Tkinter to create the GUI.

Model	Train accuracy	Validation accuracy
1st CNN	96.28	93
2nd CNN	97.44	96.5
3rd CNN	92.62	91.05
ResNet50	94.13	92.61



Classification of traffic/road signs using Graphical User Interface (GUI)

Discussion:

At the start of the project we have decided on using the German Traffic Sign dataset containing over 50,000 images over 40 signs but we hit a roadblock since to implement a GUI, we needed to test our best model using a video with traffic/road signs. But since we are residing in Canada hence this was not possible. We are aware that if we would had used the German dataset then our model could had performed better and could had been more generalized but due to the problem mentioned above we had to rely on the Canadian Traffic/road signs. Thus we proceeded with collecting the Canadian dataset containing 36 classes over 1,874 images.

We tried to implement our CNN model based on the RGB images without any data cropping, formatting etc. but in doing so we achieved accuracy of a mere 4%. So we had to format, crop, and transform our input data images into grey scale images which after implementation of our models resulted in 96.5% accuracy in CNN model-2.

In CNN model-1 we got accuracy of 93% which was satisfactory but we explored more to get more accuracy. In totality we had implemented three CNN models and model-2 of CNN achieved the highest accuracy. In CNN model-1 the number of filters taken was 16, kernel size (3,3), activation function as Relu, optimizer as adam, number of epochs 11, batch size, 32. This model achieved validation accuracy of 93.0%. For CNN model-2 the number of filters taken was 32, kernel size (3,3), activation function as Relu, optimizer as adam, number of epochs 30, batch size taken as 100. This model achieved validation accuracy of 96.5%. For CNN model-3 the number of filters taken was 64, kernel size (3,3), activation function as Relu, optimizer as adam, number of epochs 20, batch size taken as 64. This model achieved validation accuracy of 91.05%. In CNN model-1 we did not allocate the input images in train, validation and test folder as was taken in model-2. In model-1 we kept the images in further sub-folders for all the classes. In model-2 we kept the images in train folder as per the images class and we kept all the images of each class in validation and test folder without any sub folders. On executing our model, it was observed that by making this small change our accuracy was increased. We have also used the ImageDataGenerator from the keras library to shift, rotate, zoom, shear, flip and fill the input data images to make our model more generalized. Interestingly, in our other CNN models we did not try to augment the input data images, yet, the highest validation accuracy of 96.5% was achieved. In ResNet-50 model we used early stopping method, epochs 50, activation Softmax function, optimizer Adam, batch size 32. At epochs of 19 the model stopped and printed accuracy of 92.61%. All these accuracies were based on validation sets.

Conclusion:

After implementing all the models, the highest accuracy was achieved in CNN model 2. Typically, the ResNet-50 models should had given the best accuracy, and it is more preferable as it takes care of the data degradation problem that occurs in the CNN models but since our input data images was small hence ResNet-50 model could not give better accuracy than the CNN model-2 in this project. Also, ResNet-50 requires a greater number of convolutions which in our case was too shallow but on the other hand the number of input images in our dataset is also very limited hence ResNet-50 was ideally not a good model to fit our data. In future we can explore with more models like YOLO, R-CNN, Faster R-CNN and Mask R-CNN. Due to limited resources, we could not implement those models. But we aspire to use those models and further enrich our knowledge in future.

Contribution:

Bishaw Mitra Barua: Worked on several documents like problem statement, data assessment, minutes of meetings, final report and final ppt and has also collected dataset, Created CNN model and data preprocessing like converting the RGB images to Greyscale and formatting of images.

Meghaben Bhavik Patel: Worked on various documents like problem statement, data assessment, minutes of meetings, final report and final ppt. Collected dataset, divided dataset, Created CNN models and GUI using Streamlit and Tkinter.

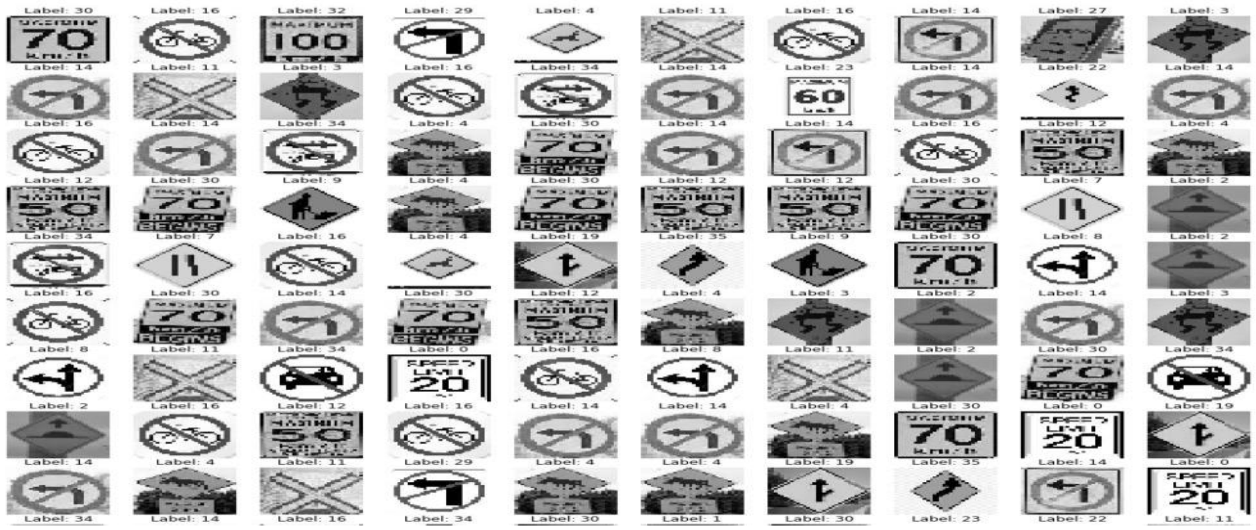
Vipin Malik: Worked on different documents like problem statement, data assessment, minutes of meetings, final report and final ppt. Collect dataset, remove noise from images, Created CNN Model, ResNet50 and GUI using Tkinter.

References:

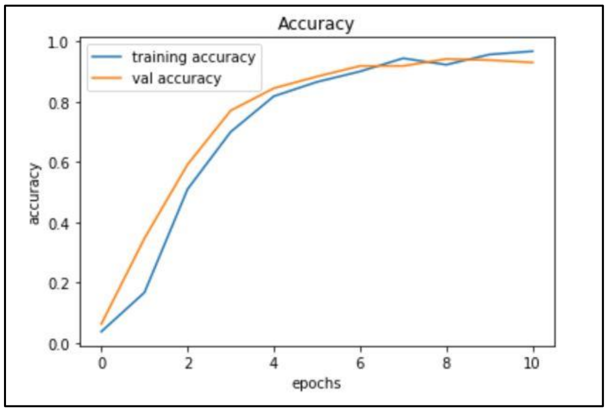
- https://www.google.com/search?q=what+is+streamlit+python&rlz=1C1CHBF_enUS886US886&oq=what+is+stramlit&aqs=chrome.3.69i57j0i13l2j0i13i30.7169j0j15&sourceid=chrome&ie=UTF-8
- https://www.kaggle.com/valentynsichkar/traffic-signs-detection-by-yolo-v3-opencv-keras
- https://arxiv.org/ftp/arxiv/papers/1901/1901.06032.pdf
- https://sci-hub.mksa.top/10.1109/ivs.2017.7995781
- https://www.youtube.com/watch?v=Etksi-F5ug8
- https://www.youtube.com/watch?v=YRhxdVk_sIs&t=31s
- https://www.youtube.com/watch?v=SXrXUqDjICA
- https://debuggercafe.com/visualizing-filters-and-feature-maps-in-convolutional-neural-networks-using-pytorch/
- https://en.wikipedia.org/wiki/Gradient_descent
- https://sci-hub.mksa.top/10.3141/2424-07
- https://sci-hub.mksa.top/10.1016/j.neucom.2018.08.009
- https://one.nhtsa.gov/nhtsa/whatis/planning/2020Report/2020report.html
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6183771/
- https://www.brunel.ac.uk/~csstyyl/papers/tmp/thesis.pdf
- https://www.google.com/search?q=first+conceive+of+self+driven+cars&rlz=1C1CHBF_enCA881CA881&oq=first+conceive+of+self+driven+cars&aqs=chrome..69i57j33i22i29i30.9392j1j9&sourceid=chrome&ie=UTF-8
- https://www.researchgate.net/publication/331807618_Road_sign_detection_and_localization_based_on_camera_and_lidar_data
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6539654/
- https://scihub.wikicn.top/10.1016/j.neucom.2018.08.009
- https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2#:~:text=The%20mean%20Average%20Precision%20or,an%20IoU%20threshold%20of%200.5.
- https://convertio.co/files.
- https://online-photo-converter.com/black-and-white-image

Appendices

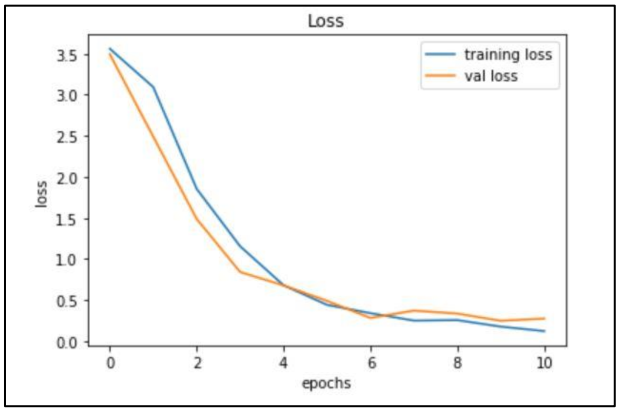
Glimpse of Our Final Dataset



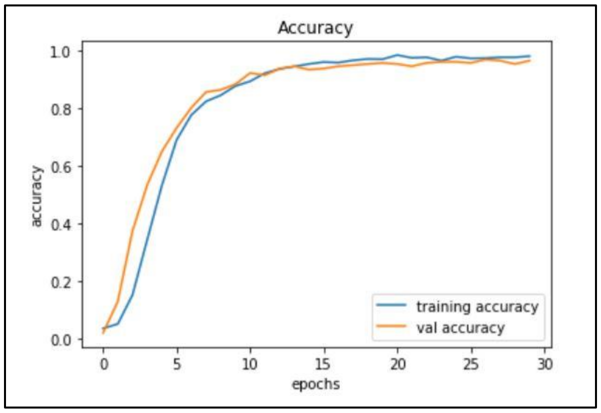
CNN Model -1(Accuracy)



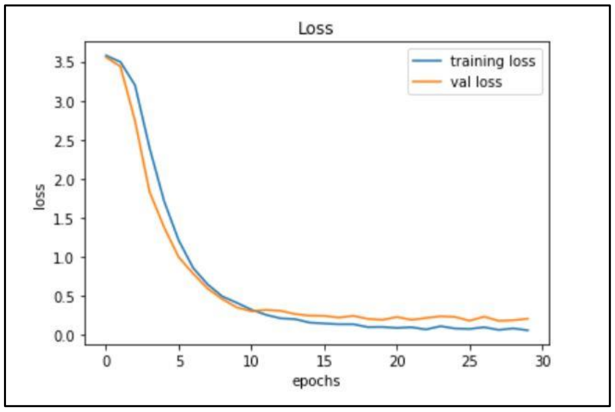
CNN Model -1(Loss)



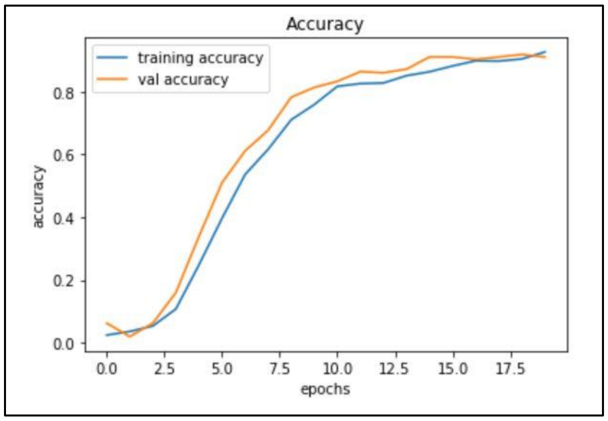
CNN Model -2(Accuracy)



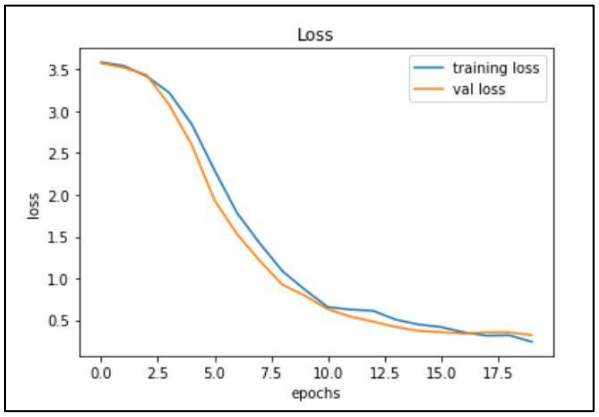
CNN Model -2(Loss)



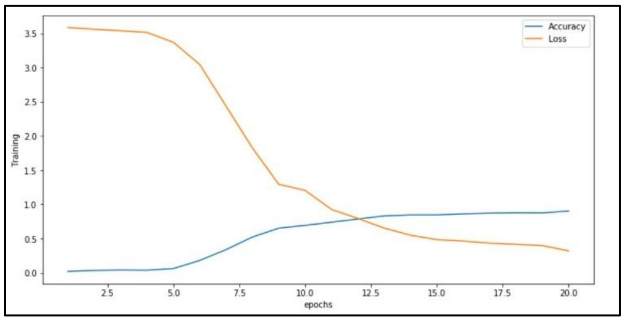
CNN Model -3(Accuracy)



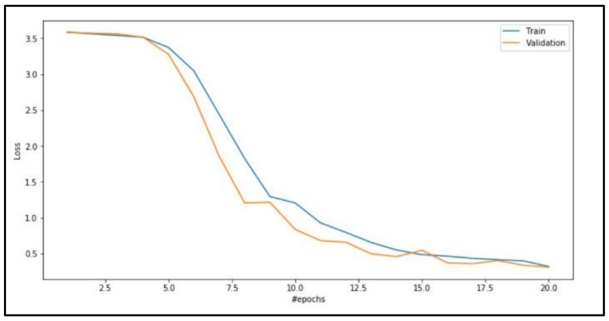
CNN Model -3(Loss)



Resnet-50 Model (Accuracy)



Resnet-50 Model (Loss)


Train
Validation
Loss
#epochs