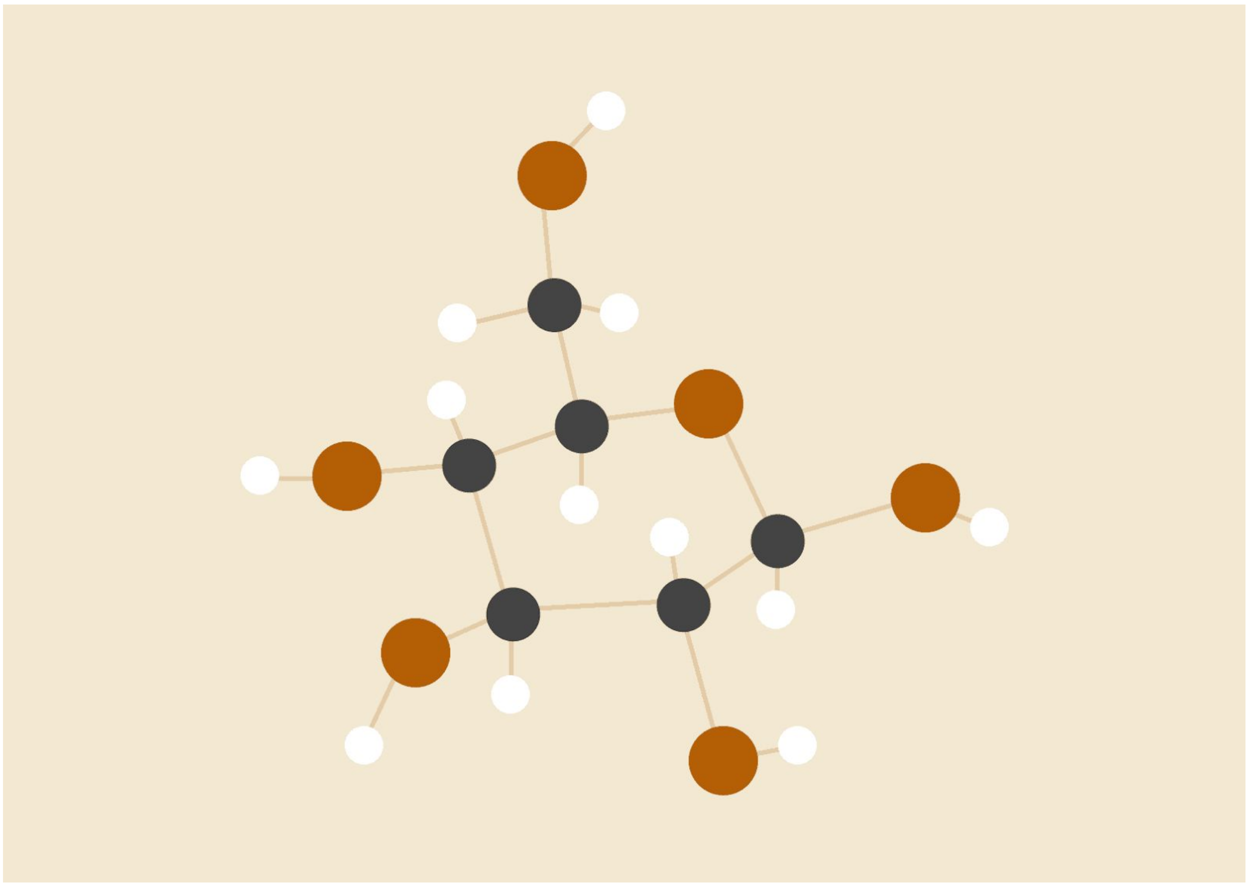
Traffic Sign Recognition

Machine learning - Homework 2



Syed Muhammad Zuhair Abbas

20.10.2019

B17-DS-02

INTRODUCTION

Following traffic signs is an important task for self-driving cars. It captures the video from the front camera and classifies the sign. In this project, we developed a traffic sign recognition system. It takes a traffic sign image with a small context around it and classifies it.

DATASET

We used a benchmark dataset, The German Traffic Sign Recognition which contains more than 40 classes and 50,000 images in total. It is a reliable ground-truth data due to semi-automatic annotation. Training images are grouped by tracks. Each track contains 30 images of one single physical traffic sign.

Image format

- Images contain a border of 10 % around the actual traffic sign (at least 5 pixels) to allow for edge-based approaches
- Images are stored in PPM format (Portable Pixmap, P6)
- Image sizes vary between 15x15 to 250x250 pixels
- Images are not necessarily squared
- The actual traffic sign is not necessarily centered within the image. This is true for images that were close to the image border in the full camera image

PREPROCESSING

Data preprocessing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it, directly affects the ability of our model to learn; therefore, it is extremely important that we pre-process our data before feeding it into our model.

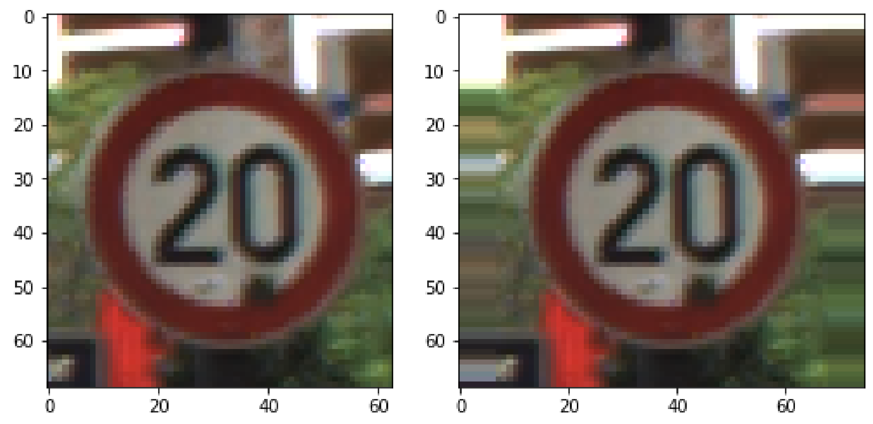
We divided the data preprocessing into the following steps:

- ❖ **Image Transformation**

  Our train and test images differ in size and shape: some of them are square, others are rectangular. We transformed every image to the same size so that they produce the same number and kind of features. The images were transformed by the following steps.

  - ➢ **Padding**

    We converted the rectangular images into squared images by duplicating the border pixels of the shorter side. Below is an example of an image before and after padding.



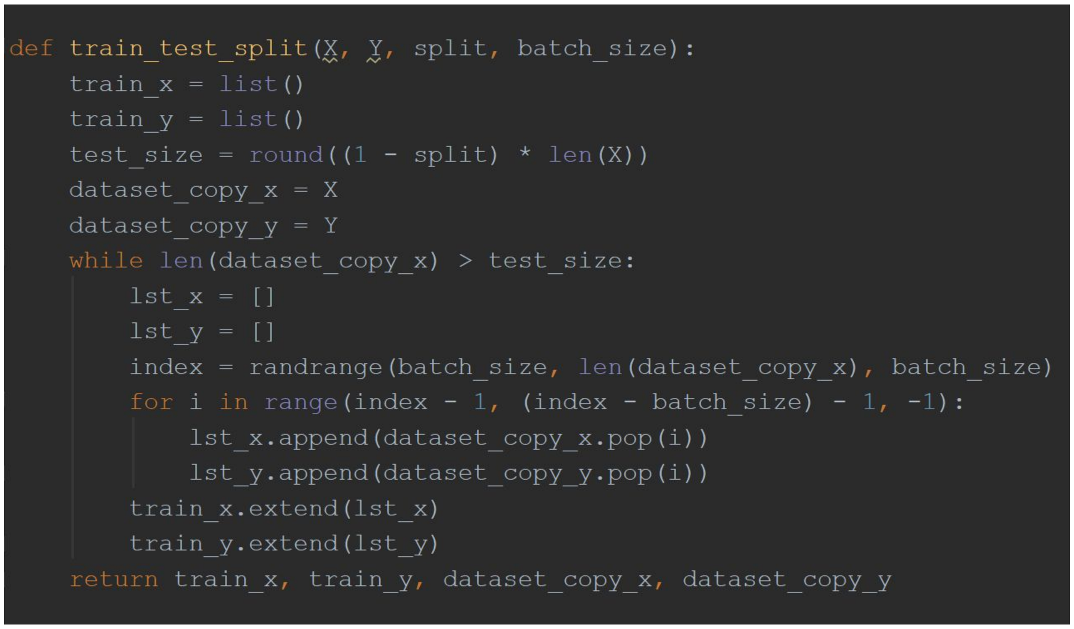
  - ➢ **Resizing**

    After changing all the images into their squared form, we resized them to the same dimensionality - 30x30 pixels.

- ❖ **Splitting data**

  We implemented our own method to split the data into training and validation sets. As the images were organized in tracks (30 images each taken from different distances and angles), it was made sure that that images from the particular track end up being in either training or validation split to avoid performance issues.

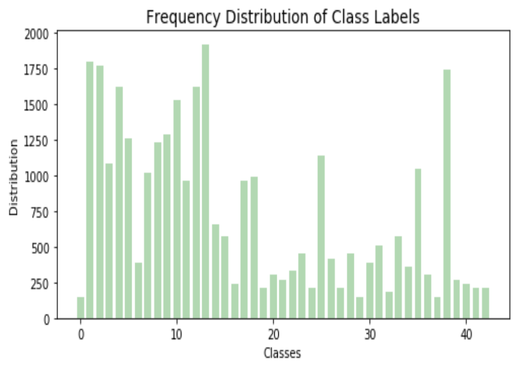
The data was being **Shuffled** while splitting because the tracks were **randomly** chosen to be put into the training set. Here is the code for our ***train_test_split*** function.

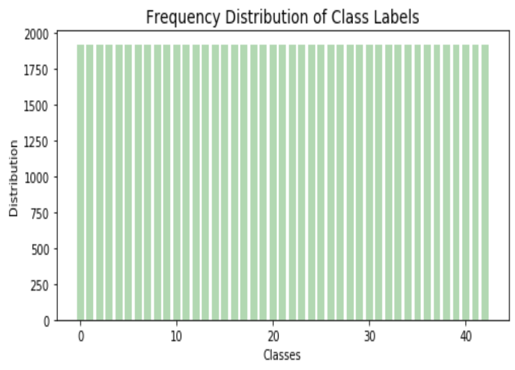


```
def train_test_split(X, Y, split, batch_size):
    train_x = list()
    train_y = list()
    test_size = round((1 - split) * len(X))
    dataset_copy_x = X
    dataset_copy_y = Y
    while len(dataset_copy_x) > test_size:
        lst_x = []
        lst_y = []
        index = randrange(batch_size, len(dataset_copy_x), batch_size)
        for i in range(index - 1, (index - batch_size) - 1, -1):
            lst_x.append(dataset_copy_x.pop(i))
            lst_y.append(dataset_copy_y.pop(i))
        train_x.extend(lst_x)
        train_y.extend(lst_y)
    return train_x, train_y, dataset_copy_x, dataset_copy_y
```

❖ Augmentation

The classes in our training set were highly unbalanced. We used data augmentation to overcome this problem. Here is the bar-chart before and after augmenting the training data, respectively.



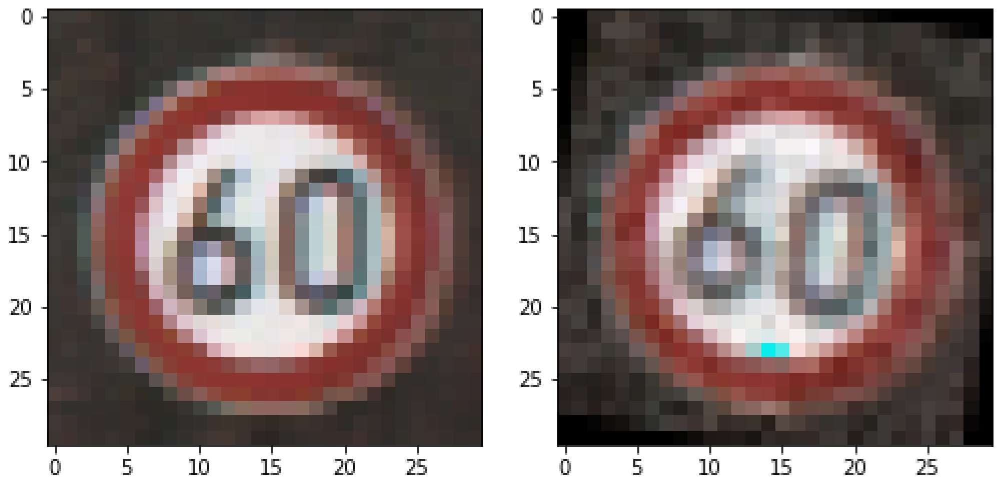


Although there are several approached for producing augmented images such as flipping, rotating, increasing contrast, adding noise, etc.

We preferred to use rotation along with some noise. In our case, some randomly chosen images were added to the dataset after performing a **rotation of 0 up to 25 degrees** and the addition of random noise of **temp ranging from 15 up to 25.**

This augmentation technique was more feasible for our data than the other techniques as it doesn't change the alignment or shape of the traffic sign, while high degree rotations and flips can totally change the shape and meaning of a traffic sign making it more prone to the classification mistakes.

Here is an example showing the original image and the image generated after augmentation.



Normalization

Normalized each image in training and validation set so that it's pixel values fall in the range of 0 to 1.

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

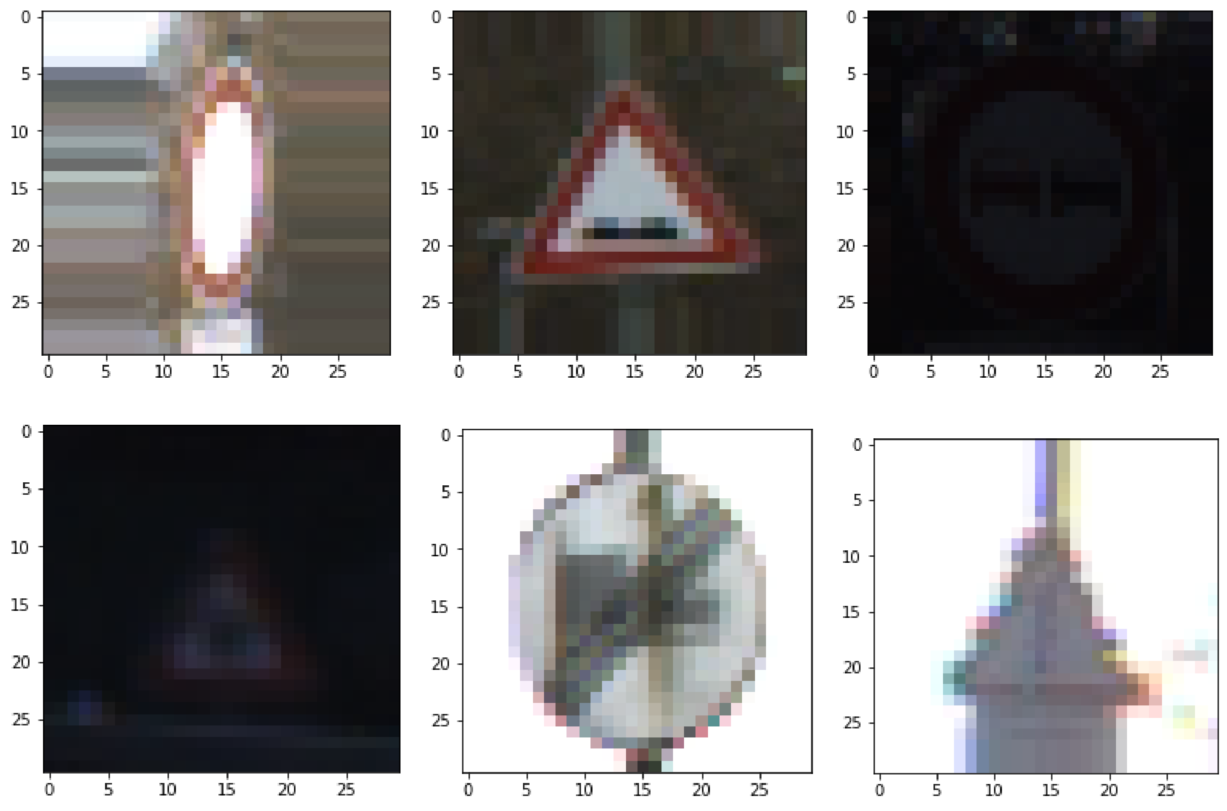
MODEL TRAINING

We used sklearn to train a Random Forest classifier with the following hyper-parameters:

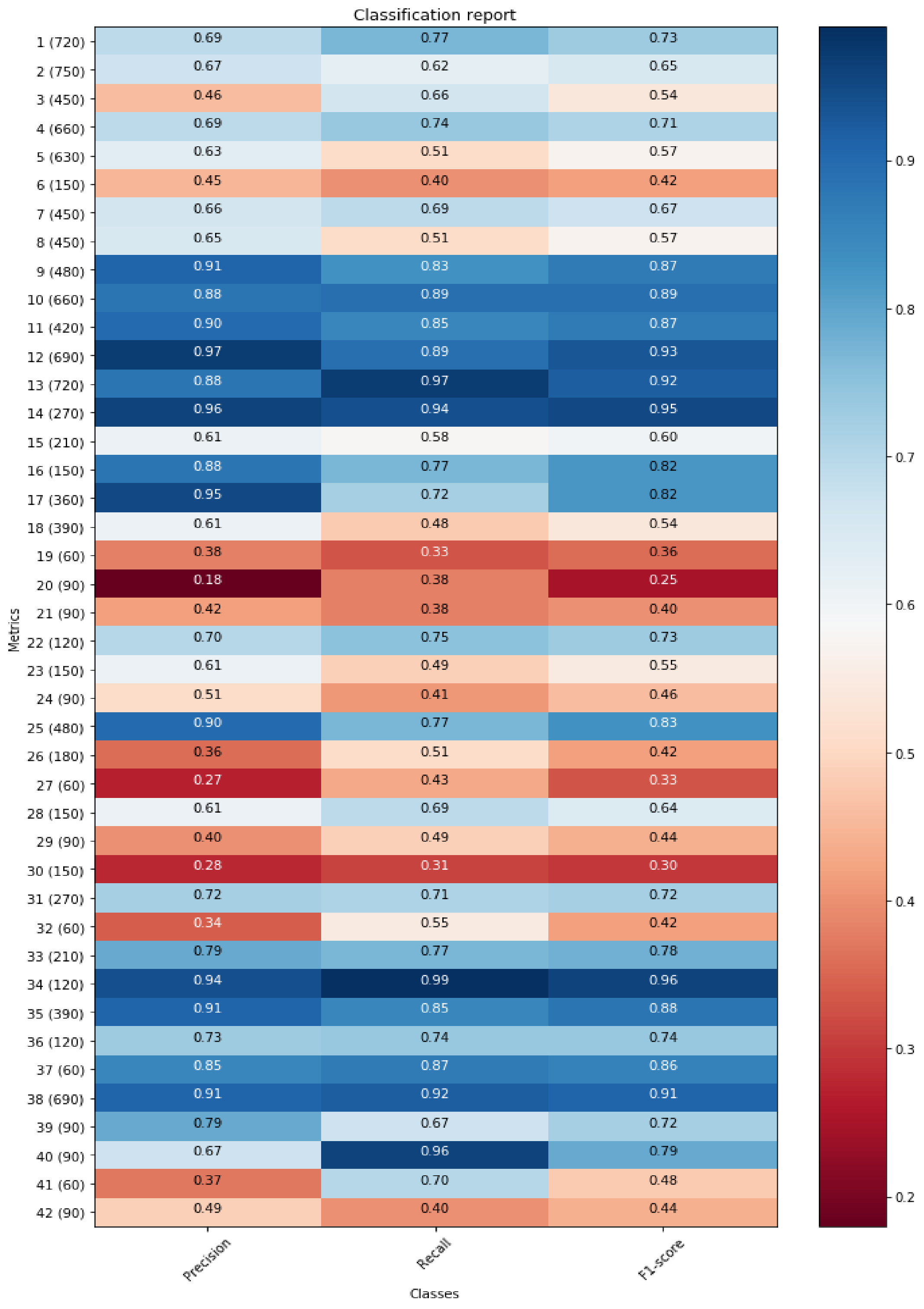
- ➔ *n_estimators* = 100
- ➔ *criterion*= 'entropy'
- ➔ *Min_samples_split* = 30
- ➔ *Max_depth* = 30

EVALUATION

We got a resulting overall accuracy of **76.52 %** for the validation set and 73.71 **%** for the test set. The following are some samples that were misclassified.



However, this estimate didn’t give any idea about the individual classes thus we measured precision and recall for each class separately to analyze problem areas.



	Precision	Recall	F1-score
1 (720)	0.69	0.77	0.73
2 (750)	0.67	0.62	0.65
3 (450)	0.46	0.66	0.54
4 (660)	0.69	0.74	0.71
5 (630)	0.63	0.51	0.57
6 (150)	0.45	0.40	0.42
7 (450)	0.66	0.69	0.67
8 (450)	0.65	0.51	0.57
9 (480)	0.91	0.83	0.87
10 (660)	0.88	0.89	0.89
11 (420)	0.90	0.85	0.87
12 (690)	0.97	0.89	0.93
13 (720)	0.88	0.97	0.92
14 (270)	0.96	0.94	0.95
15 (210)	0.61	0.58	0.60
16 (150)	0.88	0.77	0.82
17 (360)	0.95	0.72	0.82
18 (390)	0.61	0.48	0.54
19 (60)	0.38	0.33	0.36
20 (90)	0.18	0.38	0.25
21 (90)	0.42	0.38	0.40
22 (120)	0.70	0.75	0.73
23 (150)	0.61	0.49	0.55
24 (90)	0.51	0.41	0.46
25 (480)	0.90	0.77	0.83
26 (180)	0.36	0.51	0.42
27 (60)	0.27	0.43	0.33
28 (150)	0.61	0.69	0.64
29 (90)	0.40	0.49	0.44
30 (150)	0.28	0.31	0.30
31 (270)	0.72	0.71	0.72
32 (60)	0.34	0.55	0.42
33 (210)	0.79	0.77	0.78
34 (120)	0.94	0.99	0.96
35 (390)	0.91	0.85	0.88
36 (120)	0.73	0.74	0.74
37 (60)	0.85	0.87	0.86
38 (690)	0.91	0.92	0.91
39 (90)	0.79	0.67	0.72
40 (90)	0.67	0.96	0.79
41 (60)	0.37	0.70	0.48
42 (90)	0.49	0.40	0.44

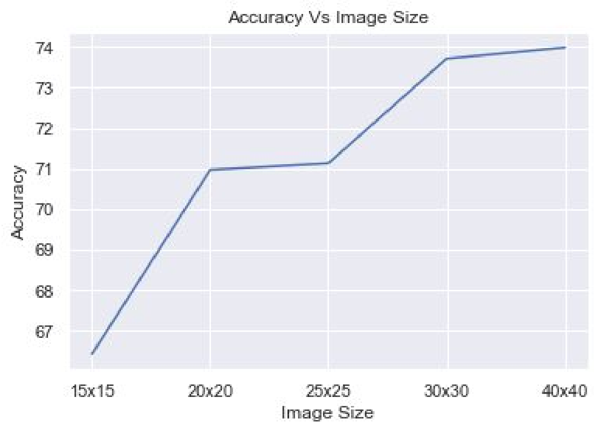
RESULTS AND ANALYSIS

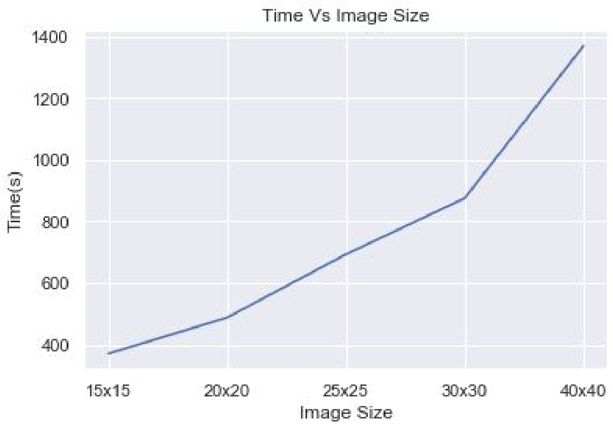
After building our base model we performed the following tests and analysis:

1. **Trained the model on non-augmented data**: Without augmentation, we got **72.72 %** accuracy for the test data set as compared to **73.72 %** accuracy with augmented data. It showed that the augmentation indeed improved the model's performance

2. **Changed the size of the images**: We evaluated the training time and test accuracy for 5 different image sizes and found out that as the image size increases, the training time and accuracy also ***increases***. Below are the results obtained using five different image sizes.

IMAGE SIZE	TEST ACCURACY %	TIME (s)
15 X 15	66.43	370.493
20 X 20	70.97	487.587
25 X 25	71.13	692.649
30 x 30	73.71	874.80
40 x 40	73.98	1368.674





These plots show how the increase in image size affects the accuracy and time respectively.

REFERENCES

1. The German Traffic Sign Recognition Benchmark: A multi-class classification competition
2. Preprocessed images