Subject:

Predict traffic signs labels using Convolutional Networks.

Authors:

Zahra MohammadBeigi

Student no. : 97222079

Ghazal Rafiei

Student no. : 97222044

We have 43 Label_names(0-42):

0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
4	Speed limit (70km/h)
5	Speed limit (80km/h)
6	End of speed limit (80km/h)
7	Speed limit (100km/h)
8	Speed limit (120km/h)
9	No passing
10	No passing for vehicles over 3.5 metric tons
11	Right-of-way at the next intersection
12	Priority road
13	Yield
14	Stop
15	No vehicles
16	Vehicles over 3.5 metric tons prohibited
17	No entry
18	General caution
19	Dangerous curve to the left
20	Dangerous curve to the right
21	Double curve
22	Bumpy road
23	Slippery road
24	Road narrows on the right
25	Road work
26	Traffic signals
27	Pedestrians
28	Children crossing
29	Bicycles crossing
30	Beware of ice/snow
31	Wild animals crossing
32	End of all speed and passing limits
33	Turn right ahead
34	Turn left ahead
35	Ahead only
36	Go straight or right
37	Go straight or left
38	Keep right
39	Keep left
40	Roundabout mandatory
41	End of no passing
42	End of no passing by vehicles over 3.5 metric ...>

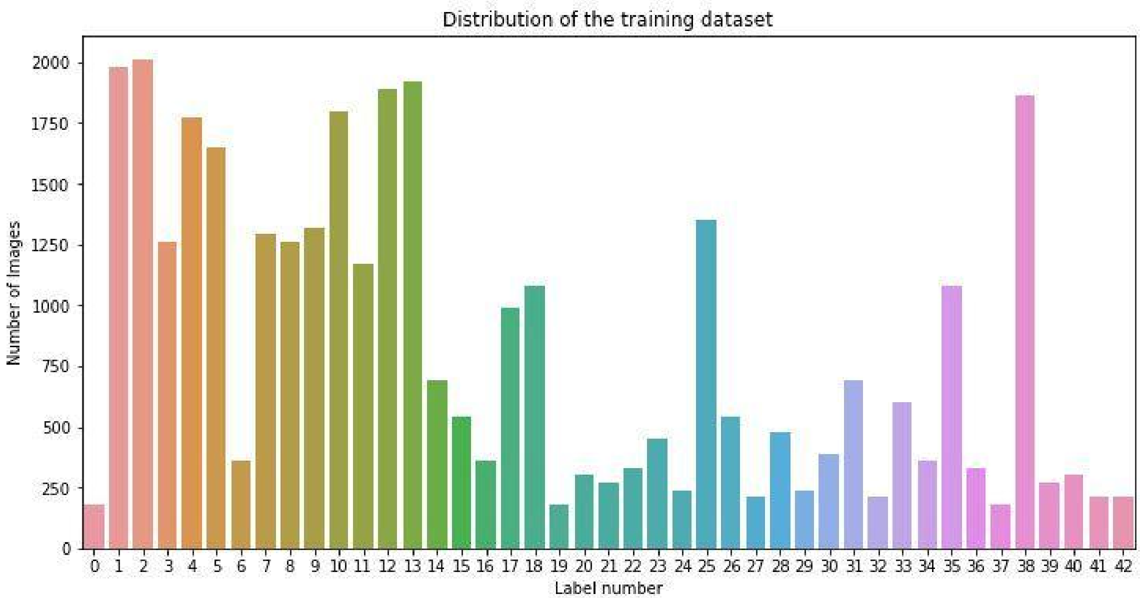
There was also size and coords data of each image but we guessed they wouldn't be effective so we did not consider them in this report article.

For the next step we will check them out.

Label Balancing:

By looking at the plot below which shows us the distribution of the training dataset, we recognize that the images of each label_name are not distributed evenly.

For example the label_name with id 0 has almost 180 images while label_name with id 2 has 2000 images.

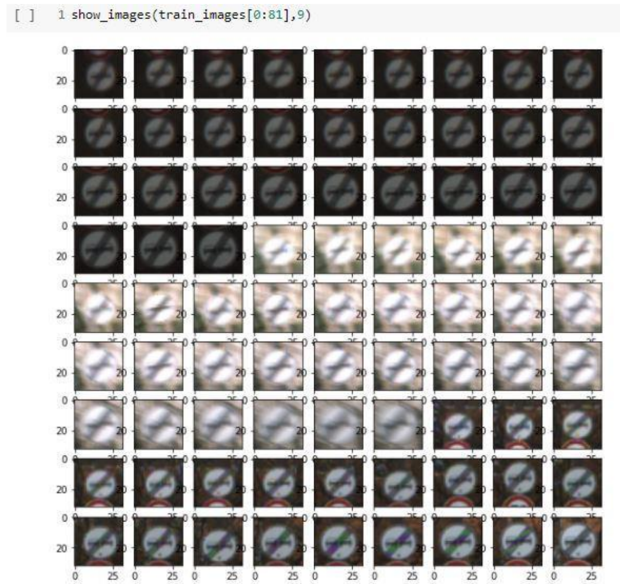


The Label names are not balanced so we balanced the label names by assigning each of them a weight which comes in an array named class_weights_dict.

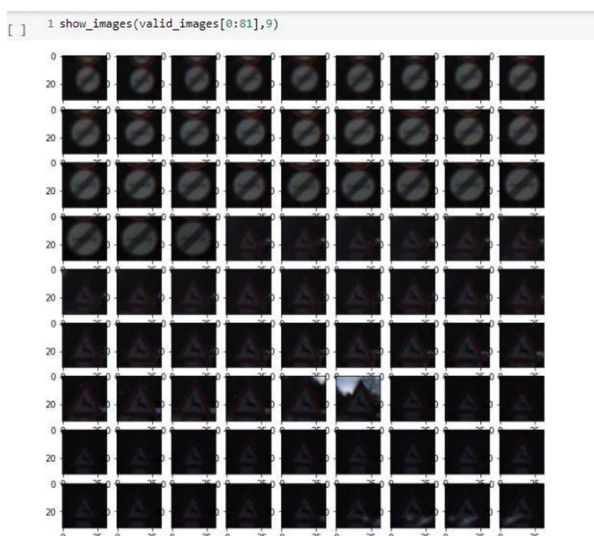
Since In image generating, each label name must come with the same probability.By using this weight dictionary the more each label name has come in the train set, the less weight it will get.

After visualizing our train and valid images we recognize that they are not shuffled well despite the test images.

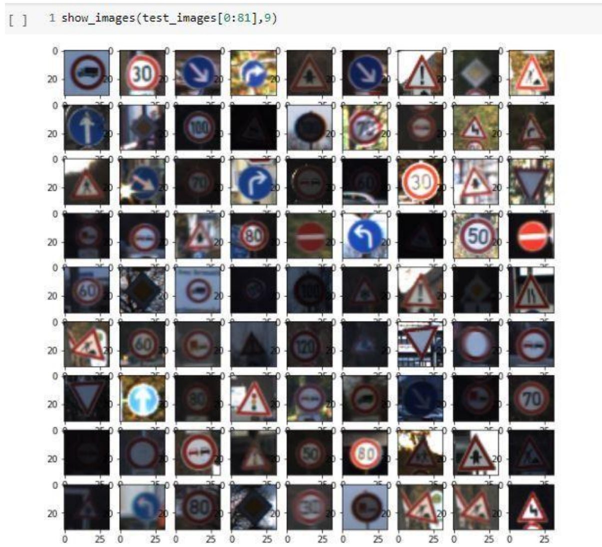
Train set Images:



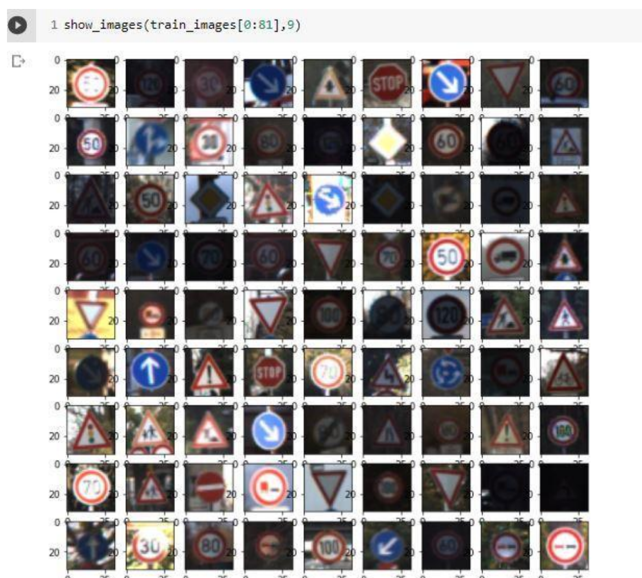
Validation Set Images:



Test Set Images:



Train images after shuffling:



Data Augmentation:

Convolutional neural networks are heavily reliant on big data to avoid overfitting. Overfitting refers to the phenomenon when a network learns a function with very high variance such as to perfectly model the training data. Unfortunately, many application domains do not have access to big data, such as this project. So we use Data Augmentation, a data-space solution to the problem of limited data. Data Augmentation encompasses a suite of techniques that enhance the size and quality of training datasets such that better Deep Learning models can be built using them.

The image augmentation algorithms include geometric transformations, color space augmentations, kernel filters, mixing images, random erasing, feature space augmentation, and etc.

By using the method ImageDataGenerator we implement Data Augmentation with these transformations: Rotate, zoom, horizontal flip.

Models Analysis:

We use the function eval_visual_model to train our models and the function's options are drawing plots and early stopping.

Early stoppings include 2 method:

1- It forces the network to stop if there is no improvement in accuracy after 10 epochs.

2- It forces the network to stop if there is no lessening in validation loss after 10 epochs.

We tried so many networks(maybe more than 50) and did not mention all of them in this article. We only chose a few of them.

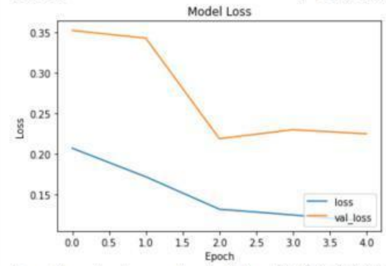
Unfortunately one of the best models got 95 percent accuracy on validation but we lost that and this is all we have!

We also could not do analytics on that.

```
2174/2174 [==============================] - 20s 9ms/step - loss: 0.1722 - accuracy: 0.9361 - val_loss: 0.3424 - val_accuracy: 0.9149
Epoch 3/10
2174/2174 [==============================] - 20s 9ms/step - loss: 0.1323 - accuracy: 0.9500 - val_loss: 0.2189 - val_accuracy: 0.9436
Epoch 4/10
2174/2174 [==============================] - 20s 9ms/step - loss: 0.1253 - accuracy: 0.9511 - val_loss: 0.2300 - val_accuracy: 0.9393
Epoch 5/10
2174/2174 [==============================] - 21s 9ms/step - loss: 0.1175 - accuracy: 0.9563 - val_loss: 0.2250 - val_accuracy: 0.9389
Epoch 00005: early stopping


Result on Test Set:
395/395 [==============================] - 2s 4ms/step - loss: 0.3024 - accuracy: 0.9198

Result on Validation Set:
138/138 [==============================] - 1s 4ms/step - loss: 0.1526 - accuracy: 0.9565
```



model_0:

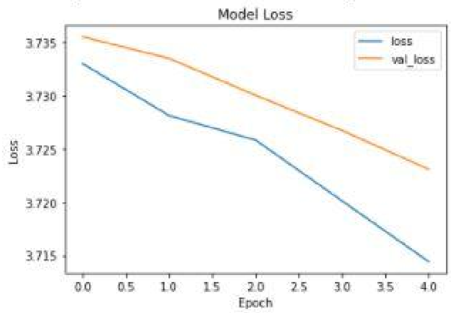
Optimizer = adadelta

```
[ ]  1 eval_visual_model([model_0], batch_sizes = 32, epoch = 5, draw_plot = True, early_stopping = True)
```

```
Epoch 1/5
1087/1087 [==============================] - 21s 20ms/step - loss: 3.7330 - accuracy: 0.0303 - val_loss: 3.7355 - val_accuracy: 0.0495
Epoch 2/5
1087/1087 [==============================] - 21s 20ms/step - loss: 3.7281 - accuracy: 0.0312 - val_loss: 3.7335 - val_accuracy: 0.0458
Epoch 3/5
1087/1087 [==============================] - 22s 20ms/step - loss: 3.7258 - accuracy: 0.0327 - val_loss: 3.7300 - val_accuracy: 0.0472
Epoch 4/5
1087/1087 [==============================] - 21s 20ms/step - loss: 3.7201 - accuracy: 0.0339 - val_loss: 3.7267 - val_accuracy: 0.0500
Epoch 5/5
1087/1087 [==============================] - 21s 20ms/step - loss: 3.7144 - accuracy: 0.0370 - val_loss: 3.7231 - val_accuracy: 0.0495


Result on Test Set:
395/395 [==============================] - 1s 4ms/step - loss: 3.7140 - accuracy: 0.0501

Result on Validation Set:
138/138 [==============================] - 0s 4ms/step - loss: 3.7165 - accuracy: 0.0549
```




```
<tensorflow.python.keras.callbacks.History at 0x7fed470df320>
```

```
[ ]  1 eval_visual_model([model_0], batch_sizes = 20, epoch = 10)
```

```
Epoch 1/10
1739/1739 [==============================] - 23s 13ms/step - loss: 3.7682 - accuracy: 0.0180 - val_loss: 3.7599 - val_accuracy: 0.0093
Epoch 2/10
1739/1739 [==============================] - 23s 13ms/step - loss: 3.7624 - accuracy: 0.0192 - val_loss: 3.7567 - val_accuracy: 0.0084
Epoch 3/10
1739/1739 [==============================] - 23s 13ms/step - loss: 3.7592 - accuracy: 0.0192 - val_loss: 3.7545 - val_accuracy: 0.0100
Epoch 4/10
1739/1739 [==============================] - 23s 13ms/step - loss: 3.7534 - accuracy: 0.0210 - val_loss: 3.7522 - val_accuracy: 0.0170
Epoch 5/10
1739/1739 [==============================] - 23s 13ms/step - loss: 3.7513 - accuracy: 0.0215 - val_loss: 3.7506 - val_accuracy: 0.0291
Epoch 6/10
1739/1739 [==============================] - 23s 13ms/step - loss: 3.7481 - accuracy: 0.0251 - val_loss: 3.7485 - val_accuracy: 0.0355
Epoch 7/10
1739/1739 [==============================] - 24s 14ms/step - loss: 3.7458 - accuracy: 0.0244 - val_loss: 3.7465 - val_accuracy: 0.0414
Epoch 8/10
1739/1739 [==============================] - 23s 13ms/step - loss: 3.7421 - accuracy: 0.0252 - val_loss: 3.7436 - val_accuracy: 0.0405
Epoch 9/10
1739/1739 [==============================] - 23s 13ms/step - loss: 3.7389 - accuracy: 0.0286 - val_loss: 3.7415 - val_accuracy: 0.0457
Epoch 10/10
1739/1739 [==============================] - 23s 14ms/step - loss: 3.7349 - accuracy: 0.0294 - val_loss: 3.7386 - val_accuracy: 0.0459


Result on Test Set:
395/395 [==============================] - 1s 4ms/step - loss: 3.7325 - accuracy: 0.0442

Result on Validation Set:
138/138 [==============================] - 1s 4ms/step - loss: 3.7344 - accuracy: 0.0490
<tensorflow.python.keras.callbacks.History at 0x7fed4735a198>
```

The speed is so low. We need a more rich network.

model_1:

Optimizer = adam

```
[ ]  1 eval_visual_model([model_1],batch_sizes =  20, epoch = 10 , draw_plot=True)
```

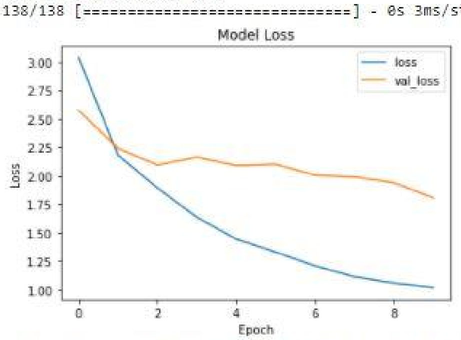
```
Epoch 1/10
1739/1739 [==============================] - 22s 13ms/step - loss: 3.0367 - accuracy: 0.0974 - val_loss: 2.5745 - val_accuracy: 0.1936
Epoch 2/10
1739/1739 [==============================] - 22s 13ms/step - loss: 2.1800 - accuracy: 0.2623 - val_loss: 2.2400 - val_accuracy: 0.2407
Epoch 3/10
1739/1739 [==============================] - 22s 13ms/step - loss: 1.8931 - accuracy: 0.3446 - val_loss: 2.0945 - val_accuracy: 0.3095
Epoch 4/10
1739/1739 [==============================] - 22s 13ms/step - loss: 1.6361 - accuracy: 0.4189 - val_loss: 2.1632 - val_accuracy: 0.3466
Epoch 5/10
1739/1739 [==============================] - 22s 13ms/step - loss: 1.4447 - accuracy: 0.4671 - val_loss: 2.0904 - val_accuracy: 0.3730
Epoch 6/10
1739/1739 [==============================] - 22s 13ms/step - loss: 1.3306 - accuracy: 0.5026 - val_loss: 2.1011 - val_accuracy: 0.3982
Epoch 7/10
1739/1739 [==============================] - 22s 13ms/step - loss: 1.2095 - accuracy: 0.5449 - val_loss: 2.0045 - val_accuracy: 0.4252
Epoch 8/10
1739/1739 [==============================] - 22s 13ms/step - loss: 1.1149 - accuracy: 0.5734 - val_loss: 1.9915 - val_accuracy: 0.4745
Epoch 9/10
1739/1739 [==============================] - 22s 13ms/step - loss: 1.0588 - accuracy: 0.5918 - val_loss: 1.9390 - val_accuracy: 0.4816
Epoch 10/10
1739/1739 [==============================] - 22s 13ms/step - loss: 1.0197 - accuracy: 0.6098 - val_loss: 1.8070 - val_accuracy: 0.4761


Result on Test Set:
395/395 [==============================] - 1s 3ms/step - loss: 1.5986 - accuracy: 0.5928

Result on Validation Set:
138/138 [==============================] - 0s 3ms/step - loss: 1.5466 - accuracy: 0.5306
```


```
Result on Test Set:
395/395 [==============================] - 1s 3ms/step - loss: 1.5986 - accuracy: 0.5928

Result on Validation Set:
138/138 [==============================] - 0s 3ms/step - loss: 1.5466 - accuracy: 0.5306
```


```
<tensorflow.python.keras.callbacks.History at 0x7fed470101d0>
```

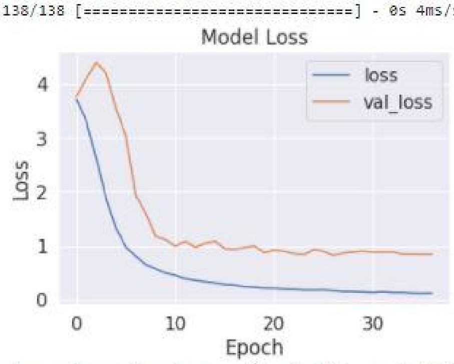
model_2:

Optimizer = adam

```
[ ]  1 eval_visual_model([model_2],batch_sizes =  500, epoch = 250,draw_plot=True, early_stopping=True)
Epoch 21/250
69/69 [==============================] - 16s 231ms/step - loss: 0.2057 - accuracy: 0.9018 - val_loss: 0.9138 - val_accuracy: 0.7678
Epoch 22/250
69/69 [==============================] - 16s 232ms/step - loss: 0.1945 - accuracy: 0.9063 - val_loss: 0.8960 - val_accuracy: 0.7690
Epoch 23/250
69/69 [==============================] - 16s 233ms/step - loss: 0.1868 - accuracy: 0.9121 - val_loss: 0.8505 - val_accuracy: 0.7815
Epoch 24/250
69/69 [==============================] - 16s 230ms/step - loss: 0.1712 - accuracy: 0.9191 - val_loss: 0.8267 - val_accuracy: 0.8012
Epoch 25/250
69/69 [==============================] - 16s 233ms/step - loss: 0.1717 - accuracy: 0.9194 - val_loss: 0.9279 - val_accuracy: 0.7955
Epoch 26/250
69/69 [==============================] - 16s 232ms/step - loss: 0.1744 - accuracy: 0.9200 - val_loss: 0.8936 - val_accuracy: 0.7832
Epoch 27/250
69/69 [==============================] - 16s 234ms/step - loss: 0.1613 - accuracy: 0.9230 - val_loss: 0.8151 - val_accuracy: 0.8092
Epoch 28/250
69/69 [==============================] - 16s 231ms/step - loss: 0.1455 - accuracy: 0.9300 - val_loss: 0.8613 - val_accuracy: 0.7955
Epoch 29/250
69/69 [==============================] - 16s 226ms/step - loss: 0.1438 - accuracy: 0.9327 - val_loss: 0.8841 - val_accuracy: 0.8133
Epoch 30/250
69/69 [==============================] - 16s 228ms/step - loss: 0.1359 - accuracy: 0.9359 - val_loss: 0.8977 - val_accuracy: 0.7970
Epoch 31/250
69/69 [==============================] - 16s 233ms/step - loss: 0.1282 - accuracy: 0.9384 - val_loss: 0.8774 - val_accuracy: 0.8095
Epoch 32/250
69/69 [==============================] - 17s 241ms/step - loss: 0.1389 - accuracy: 0.9365 - val_loss: 0.8809 - val_accuracy: 0.7980
Epoch 33/250
69/69 [==============================] - 17s 244ms/step - loss: 0.1261 - accuracy: 0.9411 - val_loss: 0.8832 - val_accuracy: 0.8077
Epoch 34/250
69/69 [==============================] - 17s 241ms/step - loss: 0.1271 - accuracy: 0.9394 - val_loss: 0.8417 - val_accuracy: 0.8183
Epoch 35/250
69/69 [==============================] - 16s 238ms/step - loss: 0.1151 - accuracy: 0.9457 - val_loss: 0.8441 - val_accuracy: 0.8202
Epoch 36/250
69/69 [==============================] - 16s 238ms/step - loss: 0.1131 - accuracy: 0.9459 - val_loss: 0.8321 - val_accuracy: 0.8213
Epoch 37/250
69/69 [==============================] - 17s 240ms/step - loss: 0.1156 - accuracy: 0.9454 - val_loss: 0.8325 - val_accuracy: 0.8285
Epoch 00037: early stopping


Result on Test Set:
395/395 [==============================] - 1s 3ms/step - loss: 0.5377 - accuracy: 0.8733

Result on Validation Set:
138/138 [==============================] - 0s 4ms/step - loss: 0.7274 - accuracy: 0.8562
```




```
<tensorflow.python.keras.callbacks.History at 0x7ff0f3afc6d8>
```

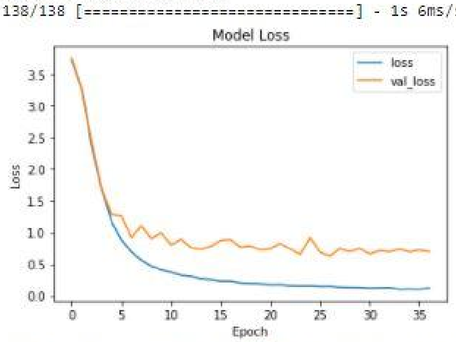
model_3:

Optimizer = adam

```
[ ]   1 eval_visual_model([model_3],batch_sizes =  120, epoch = 500,draw_plot=True,early_stopping=True)
289/289 [==============================] - 20s 70ms/step - loss: 0.1893 - accuracy: 0.9185 - val_loss: 0.7282 - val_accuracy: 0.8174
Epoch 21/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1746 - accuracy: 0.9226 - val_loss: 0.7452 - val_accuracy: 0.8097
Epoch 22/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1771 - accuracy: 0.9253 - val_loss: 0.8252 - val_accuracy: 0.7979
Epoch 23/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1599 - accuracy: 0.9304 - val_loss: 0.7418 - val_accuracy: 0.8306
Epoch 24/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1567 - accuracy: 0.9328 - val_loss: 0.6516 - val_accuracy: 0.8414
Epoch 25/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1585 - accuracy: 0.9330 - val_loss: 0.9192 - val_accuracy: 0.7889
Epoch 26/500
289/289 [==============================] - 21s 71ms/step - loss: 0.1483 - accuracy: 0.9375 - val_loss: 0.6893 - val_accuracy: 0.8363
Epoch 27/500
289/289 [==============================] - 20s 71ms/step - loss: 0.1504 - accuracy: 0.9373 - val_loss: 0.6259 - val_accuracy: 0.8468
Epoch 28/500
289/289 [==============================] - 20s 71ms/step - loss: 0.1358 - accuracy: 0.9411 - val_loss: 0.7498 - val_accuracy: 0.8162
Epoch 29/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1328 - accuracy: 0.9455 - val_loss: 0.7003 - val_accuracy: 0.8356
Epoch 30/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1304 - accuracy: 0.9451 - val_loss: 0.7497 - val_accuracy: 0.8294
Epoch 31/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1196 - accuracy: 0.9484 - val_loss: 0.6608 - val_accuracy: 0.8440
Epoch 32/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1226 - accuracy: 0.9478 - val_loss: 0.7150 - val_accuracy: 0.8208
Epoch 33/500
289/289 [==============================] - 20s 69ms/step - loss: 0.1263 - accuracy: 0.9474 - val_loss: 0.6944 - val_accuracy: 0.8308
Epoch 34/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1054 - accuracy: 0.9539 - val_loss: 0.7417 - val_accuracy: 0.8350
Epoch 35/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1097 - accuracy: 0.9535 - val_loss: 0.6925 - val_accuracy: 0.8528
Epoch 36/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1057 - accuracy: 0.9546 - val_loss: 0.7271 - val_accuracy: 0.8567
Epoch 37/500
289/289 [==============================] - 20s 70ms/step - loss: 0.1212 - accuracy: 0.9512 - val_loss: 0.6978 - val_accuracy: 0.8484
Epoch 00037: early stopping
```

```
Result on Test Set:
395/395 [==============================] - 3s 6ms/step - loss: 0.5088 - accuracy: 0.8836

Result on Validation Set:
138/138 [==============================] - 1s 6ms/step - loss: 0.6308 - accuracy: 0.8678
```




```
<tensorflow.python.keras.callbacks.History at 0x7fed0dab5a20>
```

We reached at accuracy %88. :D

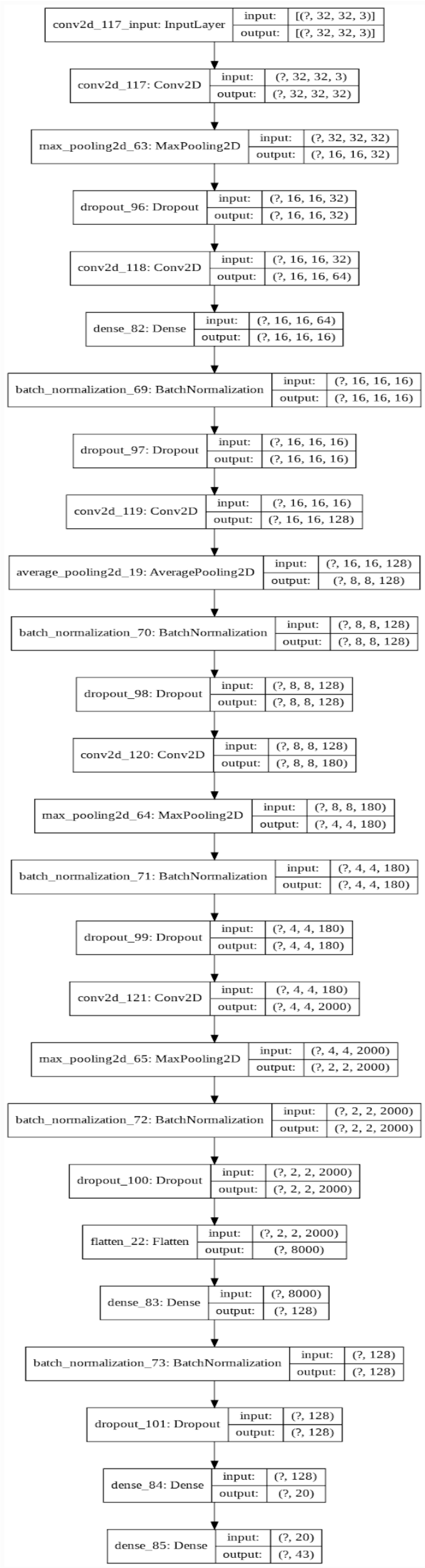
This is our best model up to here.

Model_3 Results:

Model Summary:

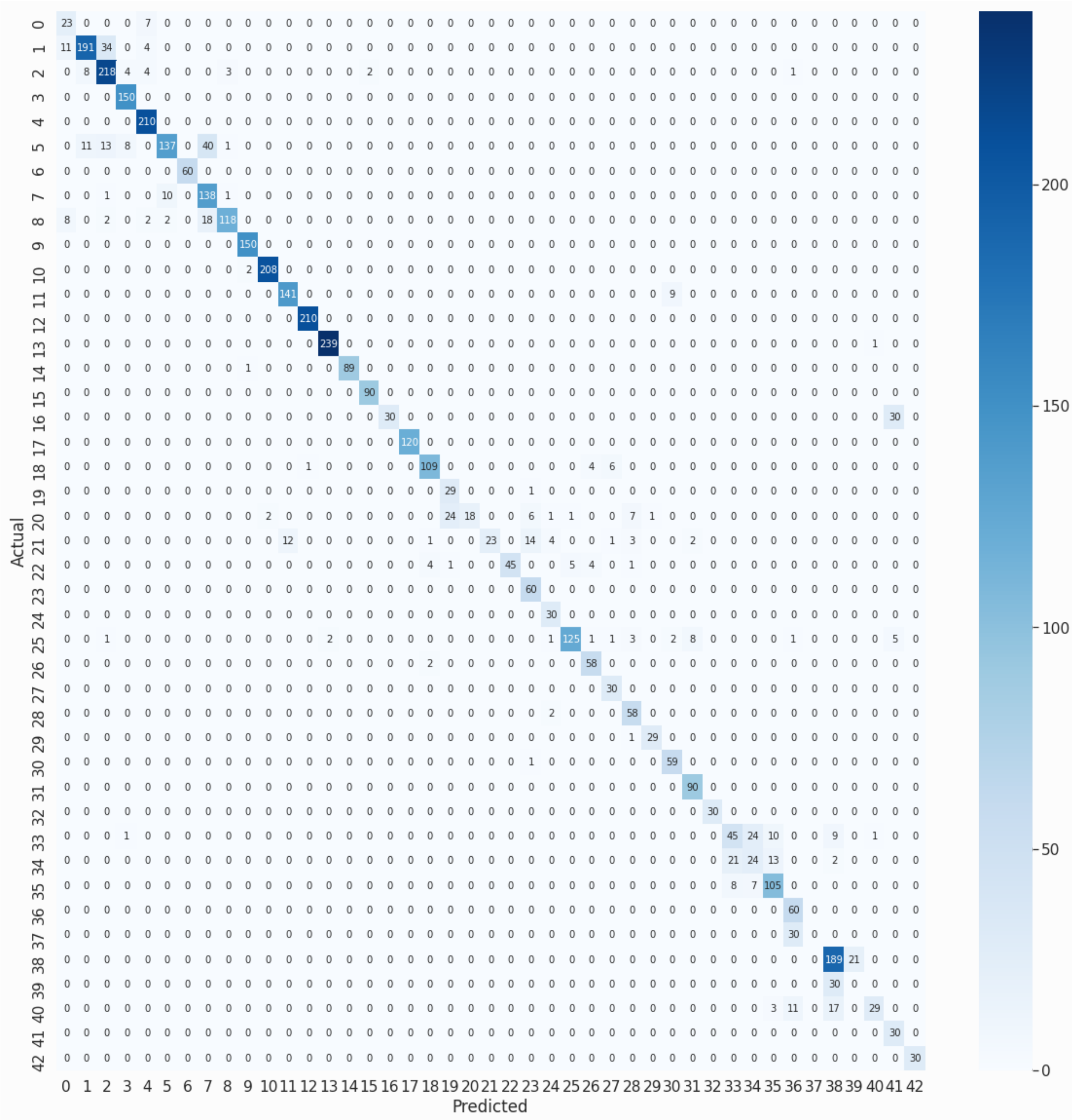
```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_117 (Conv2D)          (None, 32, 32, 32)        13856
_________________________________________________________________
max_pooling2d_63 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_96 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_118 (Conv2D)          (None, 16, 16, 64)        165952
_________________________________________________________________
dense_82 (Dense)             (None, 16, 16, 16)        1040
_________________________________________________________________
batch_normalization_69 (Batc (None, 16, 16, 16)        64
_________________________________________________________________
dropout_97 (Dropout)         (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_119 (Conv2D)          (None, 16, 16, 128)       73856
_________________________________________________________________
average_pooling2d_19 (Averag (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_70 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
dropout_98 (Dropout)         (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_120 (Conv2D)          (None, 8, 8, 180)         1866420
_________________________________________________________________
max_pooling2d_64 (MaxPooling (None, 4, 4, 180)         0
_________________________________________________________________
batch_normalization_71 (Batc (None, 4, 4, 180)         720
_________________________________________________________________
dropout_99 (Dropout)         (None, 4, 4, 180)         0
_________________________________________________________________
conv2d_121 (Conv2D)          (None, 4, 4, 2000)        3242000
_________________________________________________________________
max_pooling2d_65 (MaxPooling (None, 2, 2, 2000)        0
_________________________________________________________________
batch_normalization_72 (Batc (None, 2, 2, 2000)        8000
_________________________________________________________________
dropout_100 (Dropout)        (None, 2, 2, 2000)        0
_________________________________________________________________
flatten_22 (Flatten)         (None, 8000)              0
_________________________________________________________________
_________________________________________________________________
dense_83 (Dense)             (None, 128)               1024128
_________________________________________________________________
batch_normalization_73 (Batc (None, 128)               512
_________________________________________________________________
dropout_101 (Dropout)        (None, 128)               0
_________________________________________________________________
dense_84 (Dense)             (None, 20)                2580
_________________________________________________________________
dense_85 (Dense)             (None, 43)                903
=================================================================
Total params: 6,400,543
Trainable params: 6,395,639
Non-trainable params: 4,904
_________________________________________________________________
```

Model Plot:



conv2d_117_input: InputLayer
input: [(?, 32, 32, 3)]
output: [(?, 32, 32, 3)]
conv2d_117: Conv2D
input: (?, 32, 32, 3)
output: (?, 32, 32, 32)
max_pooling2d_63: MaxPooling2D
input: (?, 32, 32, 32)
output: (?, 16, 16, 32)
dropout_96: Dropout
input: (?, 16, 16, 32)
output: (?, 16, 16, 32)
conv2d_118: Conv2D
input: (?, 16, 16, 32)
output: (?, 16, 16, 64)
dense_82: Dense
input: (?, 16, 16, 64)
output: (?, 16, 16, 16)
batch_normalization_69: BatchNormalization
input: (?, 16, 16, 16)
output: (?, 16, 16, 16)
dropout_97: Dropout
input: (?, 16, 16, 16)
output: (?, 16, 16, 16)
conv2d_119: Conv2D
input: (?, 16, 16, 16)
output: (?, 16, 16, 128)
average_pooling2d_19: AveragePooling2D
input: (?, 16, 16, 128)
output: (?, 8, 8, 128)
batch_normalization_70: BatchNormalization
input: (?, 8, 8, 128)
output: (?, 8, 8, 128)
dropout_98: Dropout
input: (?, 8, 8, 128)
output: (?, 8, 8, 128)
conv2d_120: Conv2D
input: (?, 8, 8, 128)
output: (?, 8, 8, 180)
max_pooling2d_64: MaxPooling2D
input: (?, 8, 8, 180)
output: (?, 4, 4, 180)
batch_normalization_71: BatchNormalization
input: (?, 4, 4, 180)
output: (?, 4, 4, 180)
dropout_99: Dropout
input: (?, 4, 4, 180)
output: (?, 4, 4, 180)
conv2d_121: Conv2D
input: (?, 4, 4, 180)
output: (?, 4, 4, 2000)
max_pooling2d_65: MaxPooling2D
input: (?, 4, 4, 2000)
output: (?, 2, 2, 2000)
batch_normalization_72: BatchNormalization
input: (?, 2, 2, 2000)
output: (?, 2, 2, 2000)
dropout_100: Dropout
input: (?, 2, 2, 2000)
output: (?, 2, 2, 2000)
flatten_22: Flatten
input: (?, 2, 2, 2000)
output: (?, 8000)
dense_83: Dense
input: (?, 8000)
output: (?, 128)
batch_normalization_73: BatchNormalization
input: (?, 128)
output: (?, 128)
dropout_101: Dropout
input: (?, 128)
output: (?, 128)
dense_84: Dense
input: (?, 128)
output: (?, 20)
dense_85: Dense
input: (?, 20)
output: (?, 43)

The heat map in below shows us which signs are mislabeled more:



For example 16 and 40 are labeled vice versa which are ‘Vehicles over 3.5 metric tons prohibited’ and ‘Roundabout mandatory’ respectively.

Or 33 and 34 which are ‘Turn right ahead’ and ‘Turn left ahead’

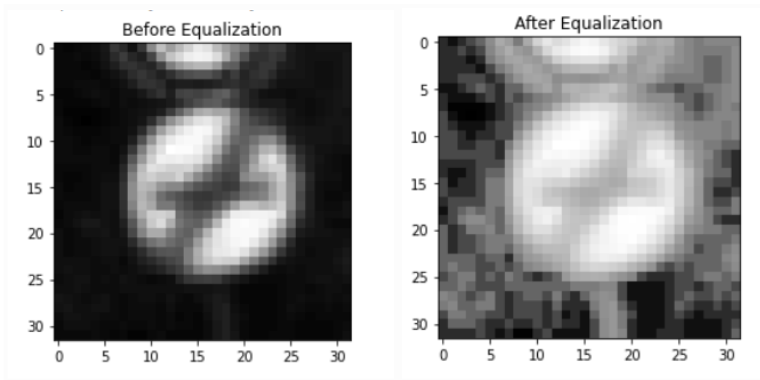
Or 5 and 7 which are ‘Speed limit (80km/h)’ and ’Speed limit (100km/h)’

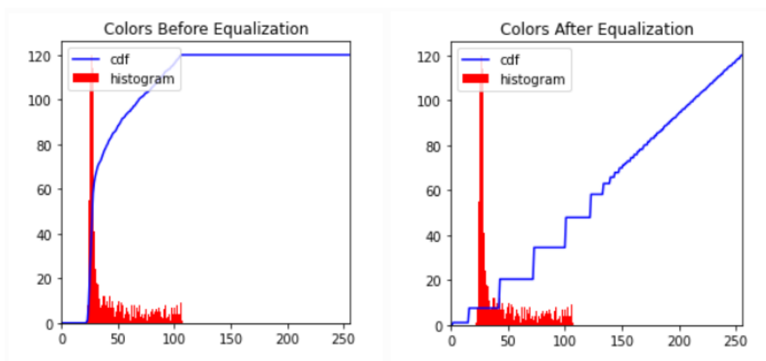
Trying grayscale images instead of RGB:

After changing all our images to grayscale and also Equalizing them all we Test our best model on new images.

Equalization:

It enhances image brightness by equalizing the color histogram of that image. This works only on grayscale images.





model_gr:

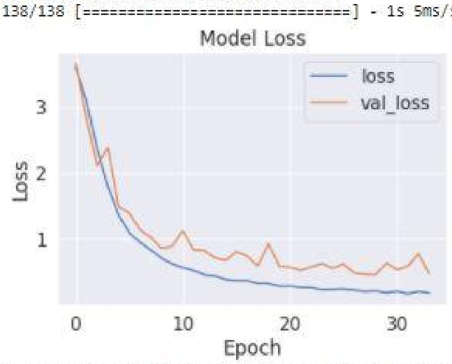
Optimizer = adam

```
[ ]  1 eval_visual_model_gs([model_gr],50,300,draw_plot = True , early_stopping=True)
```

```
695/695 [==============================] - 17s 24ms/step - loss: 0.3290 - accuracy: 0.8762 - val_loss: 0.9322 - val_accuracy: 0.7343
Epoch 20/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2872 - accuracy: 0.8896 - val_loss: 0.5911 - val_accuracy: 0.8360
Epoch 21/300
695/695 [==============================] - 16s 24ms/step - loss: 0.2904 - accuracy: 0.8897 - val_loss: 0.5706 - val_accuracy: 0.8444
Epoch 22/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2679 - accuracy: 0.8969 - val_loss: 0.5301 - val_accuracy: 0.8519
Epoch 23/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2641 - accuracy: 0.9029 - val_loss: 0.5795 - val_accuracy: 0.8321
Epoch 24/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2350 - accuracy: 0.9077 - val_loss: 0.6276 - val_accuracy: 0.8302
Epoch 25/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2402 - accuracy: 0.9110 - val_loss: 0.5576 - val_accuracy: 0.8477
Epoch 26/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2459 - accuracy: 0.9075 - val_loss: 0.6229 - val_accuracy: 0.8243
Epoch 27/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2265 - accuracy: 0.9169 - val_loss: 0.4872 - val_accuracy: 0.8715
Epoch 28/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2068 - accuracy: 0.9192 - val_loss: 0.4667 - val_accuracy: 0.8656
Epoch 29/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2160 - accuracy: 0.9176 - val_loss: 0.4615 - val_accuracy: 0.8700
Epoch 30/300
695/695 [==============================] - 16s 23ms/step - loss: 0.1871 - accuracy: 0.9276 - val_loss: 0.6375 - val_accuracy: 0.8294
Epoch 31/300
695/695 [==============================] - 16s 23ms/step - loss: 0.2106 - accuracy: 0.9221 - val_loss: 0.5383 - val_accuracy: 0.8561
Epoch 32/300
695/695 [==============================] - 16s 23ms/step - loss: 0.1740 - accuracy: 0.9326 - val_loss: 0.5877 - val_accuracy: 0.8438
Epoch 33/300
695/695 [==============================] - 16s 24ms/step - loss: 0.2072 - accuracy: 0.9243 - val_loss: 0.7788 - val_accuracy: 0.7981
Epoch 34/300
695/695 [==============================] - 16s 23ms/step - loss: 0.1837 - accuracy: 0.9318 - val_loss: 0.4783 - val_accuracy: 0.8717
Epoch 00034: early stopping
```

```
Result on Test Set in Gray Scale:
395/395 [==============================] - 2s 5ms/step - loss: 0.4140 - accuracy: 0.8905

Result on Validation Setin Gray Scale:
138/138 [==============================] - 1s 5ms/step - loss: 0.5038 - accuracy: 0.8891
```




```
<tensorflow.python.keras.callbacks.History at 0x7ff0f2e38e10>
```

The result was better than all last networks .

There is a guess that maybe because most traffic signs have blue and red colors, removing the green part and working with only red and blue colors can help to improve the result.

model_rb:

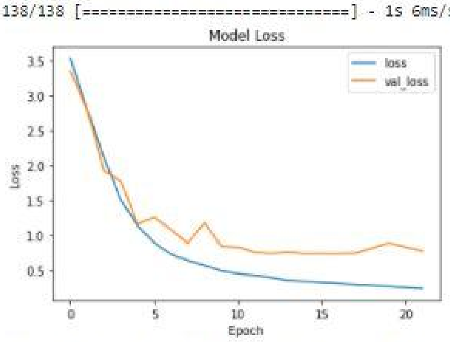
Optimizer = adam

```
[ ]  1 eval_visual_model_rb([model_rb],batch_sizes =  120, epoch = 500,draw_plot=True,early_stopping=True)

Epoch 7/500
289/289 [==============================] - 21s 73ms/step - loss: 0.7295 - accuracy: 0.6990 - val_loss: 1.0856 - val_accuracy: 0.6507
Epoch 8/500
289/289 [==============================] - 21s 73ms/step - loss: 0.6420 - accuracy: 0.7325 - val_loss: 0.8883 - val_accuracy: 0.6870
Epoch 9/500
289/289 [==============================] - 21s 72ms/step - loss: 0.5710 - accuracy: 0.7587 - val_loss: 1.1826 - val_accuracy: 0.6579
Epoch 10/500
289/289 [==============================] - 21s 72ms/step - loss: 0.4964 - accuracy: 0.7847 - val_loss: 0.8439 - val_accuracy: 0.7280
Epoch 11/500
289/289 [==============================] - 21s 73ms/step - loss: 0.4517 - accuracy: 0.8005 - val_loss: 0.8324 - val_accuracy: 0.7472
Epoch 12/500
289/289 [==============================] - 21s 73ms/step - loss: 0.4295 - accuracy: 0.8153 - val_loss: 0.7601 - val_accuracy: 0.7528
Epoch 13/500
289/289 [==============================] - 21s 73ms/step - loss: 0.3955 - accuracy: 0.8290 - val_loss: 0.7408 - val_accuracy: 0.7785
Epoch 14/500
289/289 [==============================] - 21s 73ms/step - loss: 0.3562 - accuracy: 0.8415 - val_loss: 0.7608 - val_accuracy: 0.7817
Epoch 15/500
289/289 [==============================] - 21s 72ms/step - loss: 0.3414 - accuracy: 0.8462 - val_loss: 0.7382 - val_accuracy: 0.7694
Epoch 16/500
289/289 [==============================] - 21s 73ms/step - loss: 0.3267 - accuracy: 0.8569 - val_loss: 0.7405 - val_accuracy: 0.7799
Epoch 17/500
289/289 [==============================] - 21s 74ms/step - loss: 0.3130 - accuracy: 0.8625 - val_loss: 0.7377 - val_accuracy: 0.7859
Epoch 18/500
289/289 [==============================] - 21s 74ms/step - loss: 0.2965 - accuracy: 0.8656 - val_loss: 0.7452 - val_accuracy: 0.7859
Epoch 19/500
289/289 [==============================] - 21s 74ms/step - loss: 0.2833 - accuracy: 0.8753 - val_loss: 0.8182 - val_accuracy: 0.7910
Epoch 20/500
289/289 [==============================] - 21s 73ms/step - loss: 0.2699 - accuracy: 0.8822 - val_loss: 0.8898 - val_accuracy: 0.7602
Epoch 21/500
289/289 [==============================] - 21s 72ms/step - loss: 0.2544 - accuracy: 0.8868 - val_loss: 0.8341 - val_accuracy: 0.7743
Epoch 22/500
289/289 [==============================] - 21s 72ms/step - loss: 0.2415 - accuracy: 0.8898 - val_loss: 0.7764 - val_accuracy: 0.8065
Epoch 00022: early stopping

Result on Test Set:
395/395 [==============================] - 2s 6ms/step - loss: 0.5896 - accuracy: 0.8430

Result on Validation Set:
138/138 [==============================] - 1s 6ms/step - loss: 0.6744 - accuracy: 0.8236
```




```
<tensorflow.python.keras.callbacks.History at 0x7ff1c34056a0>
```

It did not change dramatically compared to colorful images.

Result:

The most accurate was on grayscale model with equalization.

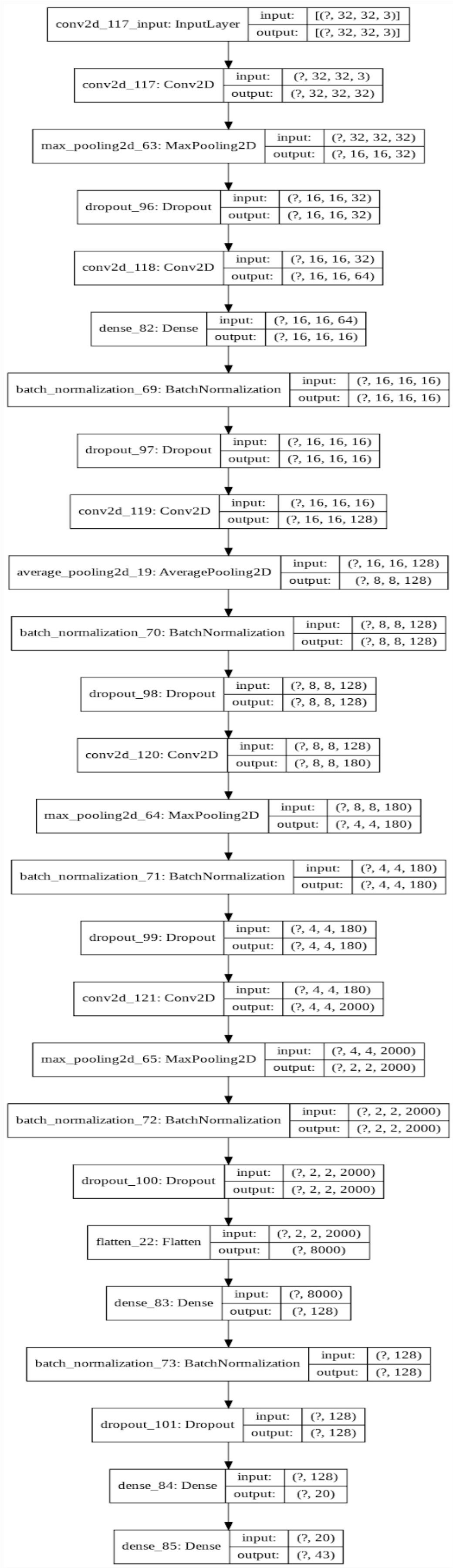
Model summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        7232
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        294976
_________________________________________________________________
dense (Dense)                (None, 16, 16, 64)        4160
_________________________________________________________________
batch_normalization (BatchNo (None, 16, 16, 64)        256
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 81)        419985
_________________________________________________________________
average_pooling2d (AveragePo (None, 8, 8, 81)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 81)          324
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 81)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 100)         291700
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 100)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 80)          288080
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 80)          320
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          46144
```

conv2d_5 (Conv2D)   (None, 4, 4, 64)   46144

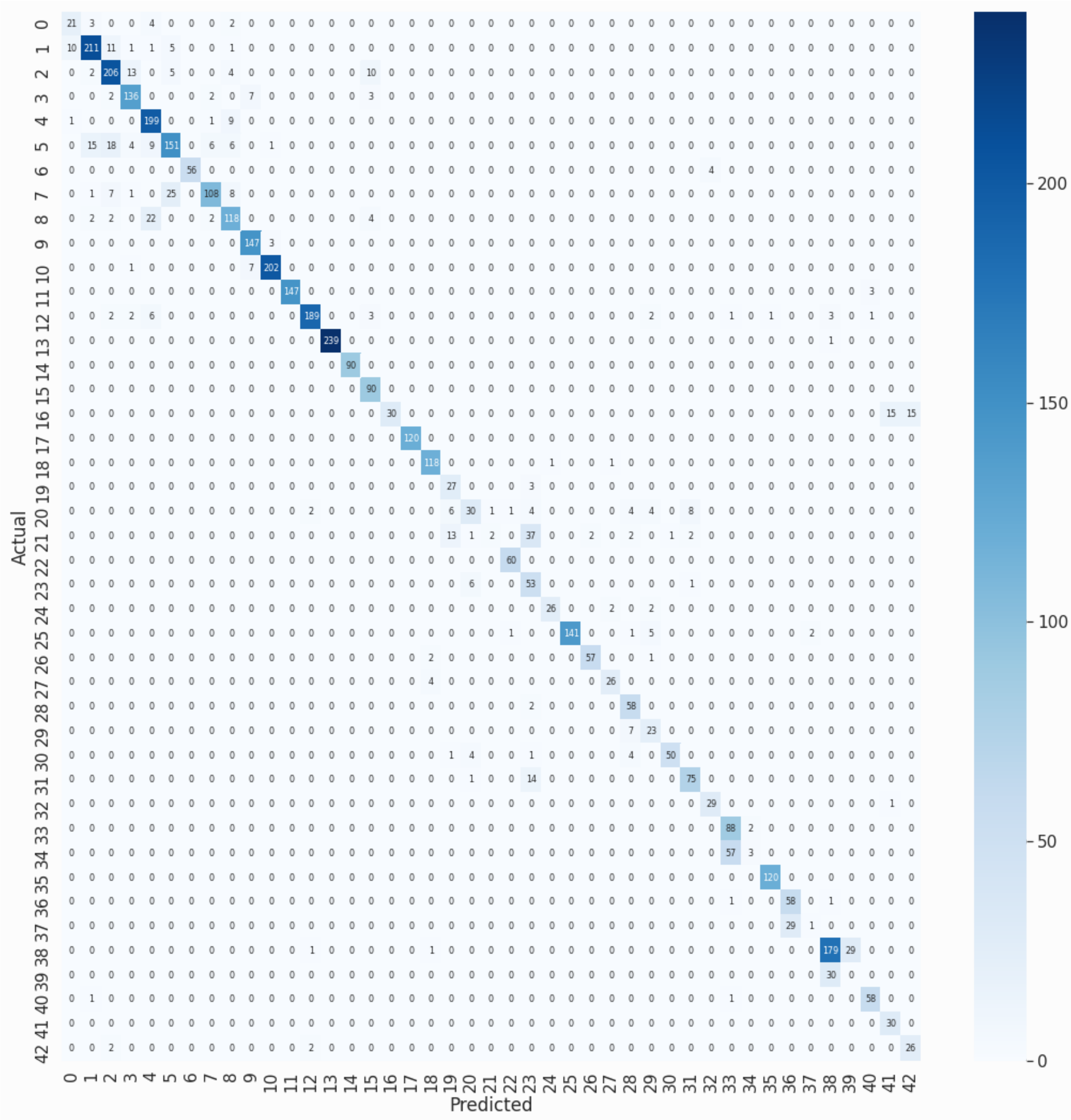
```
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                16448
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                3250
_________________________________________________________________
dense_3 (Dense)              (None, 43)                2193
=================================================================
Total params: 1,375,068
Trainable params: 1,374,618
Non-trainable params: 450
```

Model Plot:



conv2d_117_input: InputLayer | input: [(?, 32, 32, 3)] | output: [(?, 32, 32, 3)]
conv2d_117: Conv2D | input: (?, 32, 32, 3) | output: (?, 32, 32, 32)
max_pooling2d_63: MaxPooling2D | input: (?, 32, 32, 32) | output: (?, 16, 16, 32)
dropout_96: Dropout | input: (?, 16, 16, 32) | output: (?, 16, 16, 32)
conv2d_118: Conv2D | input: (?, 16, 16, 32) | output: (?, 16, 16, 64)
dense_82: Dense | input: (?, 16, 16, 64) | output: (?, 16, 16, 16)
batch_normalization_69: BatchNormalization | input: (?, 16, 16, 16) | output: (?, 16, 16, 16)
dropout_97: Dropout | input: (?, 16, 16, 16) | output: (?, 16, 16, 16)
conv2d_119: Conv2D | input: (?, 16, 16, 16) | output: (?, 16, 16, 128)
average_pooling2d_19: AveragePooling2D | input: (?, 16, 16, 128) | output: (?, 8, 8, 128)
batch_normalization_70: BatchNormalization | input: (?, 8, 8, 128) | output: (?, 8, 8, 128)
dropout_98: Dropout | input: (?, 8, 8, 128) | output: (?, 8, 8, 128)
conv2d_120: Conv2D | input: (?, 8, 8, 128) | output: (?, 8, 8, 180)
max_pooling2d_64: MaxPooling2D | input: (?, 8, 8, 180) | output: (?, 4, 4, 180)
batch_normalization_71: BatchNormalization | input: (?, 4, 4, 180) | output: (?, 4, 4, 180)
dropout_99: Dropout | input: (?, 4, 4, 180) | output: (?, 4, 4, 180)
conv2d_121: Conv2D | input: (?, 4, 4, 180) | output: (?, 4, 4, 2000)
max_pooling2d_65: MaxPooling2D | input: (?, 4, 4, 2000) | output: (?, 2, 2, 2000)
batch_normalization_72: BatchNormalization | input: (?, 2, 2, 2000) | output: (?, 2, 2, 2000)
dropout_100: Dropout | input: (?, 2, 2, 2000) | output: (?, 2, 2, 2000)
flatten_22: Flatten | input: (?, 2, 2, 2000) | output: (?, 8000)
dense_83: Dense | input: (?, 8000) | output: (?, 128)
batch_normalization_73: BatchNormalization | input: (?, 128) | output: (?, 128)
dropout_101: Dropout | input: (?, 128) | output: (?, 128)
dense_84: Dense | input: (?, 128) | output: (?, 20)
dense_85: Dense | input: (?, 20) | output: (?, 43)

Labels Heat Map:



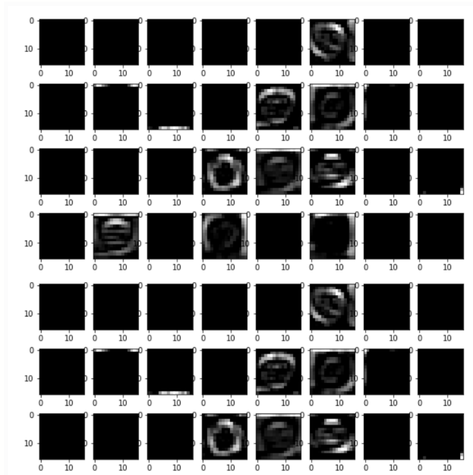
37 and 23 are 'Go straight or left' and 'Slippery road' respectively.

5 and 7 are Speed limit (80km/h)' and 'Speed limit (100km/h)' respectively which was mislabeled more in other models.

Taking a Look at Feature Map:

It shows which features of the image are detected and is more useful to recognize the label.

For example this map shows the network detected the circle shape in the image.



Smaller features: