Classifying Emergency Response Vehicles using Machine Learning Techniques

Created By:
Sumit Dutt Mishra
811123466

Submitted to:
Dr. Amin Gharipour

SUMMARY:

In highly populated countries like India and China, the traffic congestion is a huge problem. Because of which most of the emergency vehicles are not able reach their destination on time. With the help of many Machine Learning techniques we could create an algorithm which could help to detect the emergency vehicles and regulate the traffic smoothly. Here we are discussing about one such technique which is based on **Convolution Neural Networks**.

A Convolution Neural Network is a neural network that has one or more convolution l layers and are used mainly for image processing, classification, segmentation and also for other auto co related data.

For this algorithm we collected a data which have 2357 images of vehicles(emergency and non-emergency). We have differentiated these images into training and test dataset. We have 1646 images in our training data and 711 images in our test dataset. The performance metric is accuracy so we are moving forward with a pre-trained model RestNet101 as it has previous set weights and are trained on millions of images. By using RestNet101 we are able to achieve accuracy of 93.31%.

This model can be used and installed at all the traffic signals. The camera installed at the signal can be used to differentiate between emergency and non-emergency vehicles. The installed camera will be able to identify and track any oncoming any emergency response vehicles up to a certain range. If any emergency vehicle is coming then all the traffic signals in its way should be converted to green and emergency lights should start blinking so that people could make way for the emergency response vehicles.

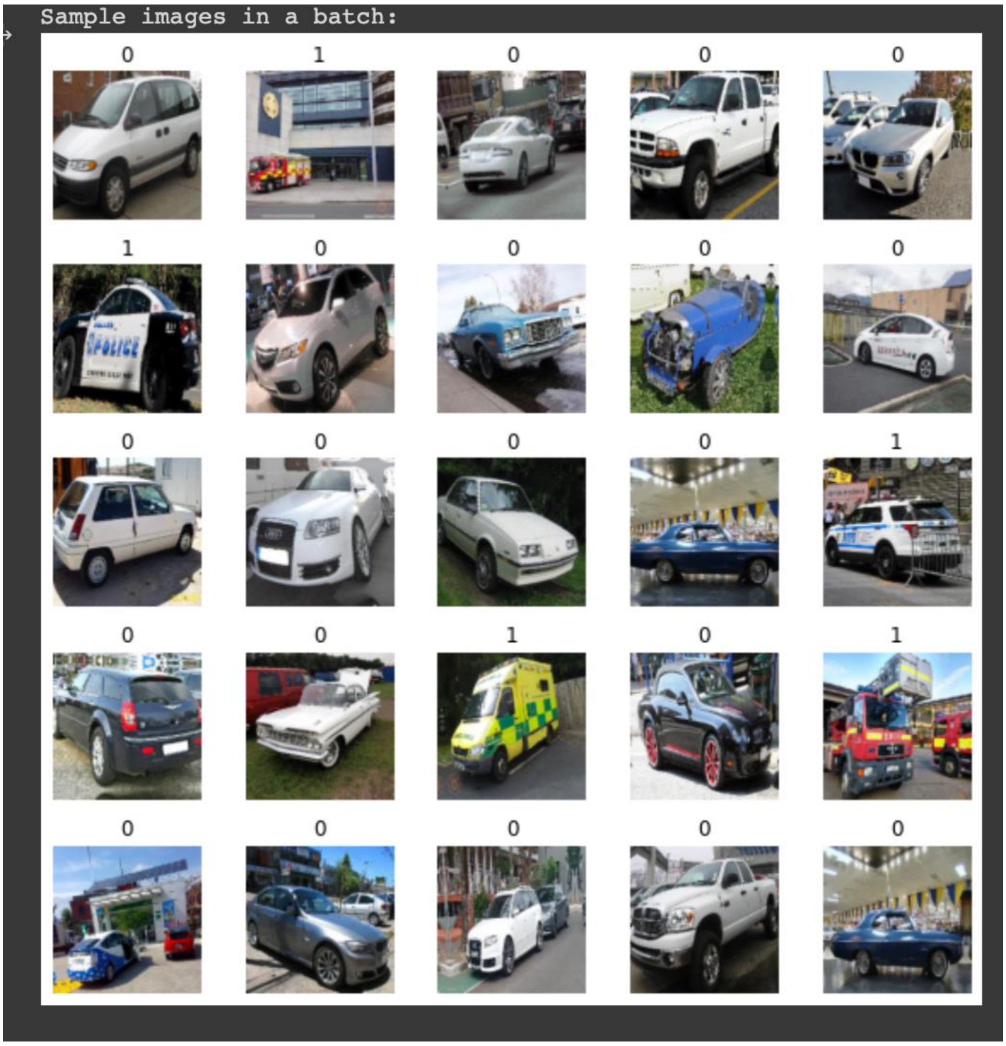
PROBLEM STATEMENT:

Traffic light plays a vital role in any system and the there have been many people who have died because emergency response vehicles were not able to reach their destination on time. In the growing world and in overly populated countries these vehicles find it very difficult to manage and reach their destination on time as the cities are always crowded because of this many people have lost their lives.
People are dying in the back of ambulances and up to 160,000 more a year are coming to harm because the ambulances are stuck in traffic and are unable to be offloaded to the hospitals. In metropolitan cities, the crime rate and traffic congestion is very high and the police is not able to reach the crime scene on time which results in causalities. An increased volume of vehicles not only increases the response time of emergency vehicles but it also increases the chances of them being involved in accidents. The emergency vehicle entering an intersection at high speed on red light poses danger to traffic on other roads and can cause accidents.

Emergency vehicles, such as ambulance, fire engines and police cars should be able to react to emergency calls with minimum delay. The excellence of the emergency service depends on how fast the emergency vehicle can reach the incident location. If the emergency get stuck in the traffic jam and its arrival at the incident location is late it can cause loss of lives and property.

DATA PREPARATION:

We have collected data which have 2357 images of emergency and non-emergency vehicles combined. We have then converted the data into binary form, which means all the emergency vehicles will represent “1” and all the non-emergency vehicles represent “0”. We took a sample of data.



Here we can see that the emergency vehicle is marked as “1” whereas the non-emergency vehicle is marked as “0”.After checking the sample, we have differentiated the data and made new training and test data set. The training dataset consist of 1646 images and the test data set have 711 images for training and testing the model.
After that we have subset the training dataset into two (2) categories i.e. train and validation set. The train set consist of 1317 images and validation set consist of 329 images .

Defining number of classes

```
[ ] print("\n Number of classes: ",training_data1.c)

     Number of classes:  2

[ ] len(training_data1.train_ds), len(training_data1.valid_ds)

    (1317, 329)
```

We have also installed torch-vision for using pretrained model i.e RestNet101 for our algorithm for getting maximum accuracy.

MODEL PLANNING:

For the Convolutional Neural Network , we are going to take three(3) approaches for achieving maximum accuracy.

- Training: 1000 images, Validation: 600 images, Test : 757 images:
  We initiated a small convnet for Emergency vs Non- Emergency vehicles classification . We made a model having 4 conv2D layers and 4 maxpooling layers. Maxpooling is a pooling operation that calculates the maximum, or largest value in each patch of future map.
  We managed the file directory and divided the file into training, validation and test, where training has 1000 images, validation has 600 images and test dataset has 757 images. We started fitting the model on training dataset with 35 epochs(training the neural network with all the training data for one cycle).

```
Epoch 1/35
63/63 [==============================] - 8s 107ms/step - loss: 0.7566 - accuracy: 0.5000 - val_loss: 0.6924 - val_accuracy: 0.5610
Epoch 2/35
63/63 [==============================] - 7s 104ms/step - loss: 0.7388 - accuracy: 0.5255 - val_loss: 0.6893 - val_accuracy: 0.5360
Epoch 3/35
63/63 [==============================] - 7s 105ms/step - loss: 0.6938 - accuracy: 0.5960 - val_loss: 0.6523 - val_accuracy: 0.6060
Epoch 4/35
63/63 [==============================] - 7s 104ms/step - loss: 0.6740 - accuracy: 0.6360 - val_loss: 0.7172 - val_accuracy: 0.5610
Epoch 5/35
63/63 [==============================] - 7s 105ms/step - loss: 0.6243 - accuracy: 0.6710 - val_loss: 0.6157 - val_accuracy: 0.6360
Epoch 6/35
63/63 [==============================] - 7s 104ms/step - loss: 0.5842 - accuracy: 0.6915 - val_loss: 0.5438 - val_accuracy: 0.7170
Epoch 7/35
63/63 [==============================] - 7s 104ms/step - loss: 0.5652 - accuracy: 0.7120 - val_loss: 0.5384 - val_accuracy: 0.7480
Epoch 8/35
63/63 [==============================] - 7s 105ms/step - loss: 0.4938 - accuracy: 0.7650 - val_loss: 1.3275 - val_accuracy: 0.5580
Epoch 9/35
63/63 [==============================] - 7s 106ms/step - loss: 0.4754 - accuracy: 0.7895 - val_loss: 0.5169 - val_accuracy: 0.7480
Epoch 10/35
63/63 [==============================] - 7s 104ms/step - loss: 0.4152 - accuracy: 0.8195 - val_loss: 0.6093 - val_accuracy: 0.7210
Epoch 11/35
63/63 [==============================] - 7s 104ms/step - loss: 0.3769 - accuracy: 0.8365 - val_loss: 0.6382 - val_accuracy: 0.7160
Epoch 12/35
63/63 [==============================] - 7s 104ms/step - loss: 0.3123 - accuracy: 0.8660 - val_loss: 0.7648 - val_accuracy: 0.7180
Epoch 13/35
63/63 [==============================] - 7s 104ms/step - loss: 0.2648 - accuracy: 0.8990 - val_loss: 1.0057 - val_accuracy: 0.6970
Epoch 14/35
63/63 [==============================] - 7s 104ms/step - loss: 0.2264 - accuracy: 0.9100 - val_loss: 0.7703 - val_accuracy: 0.7720
Epoch 15/35
63/63 [==============================] - 7s 105ms/step - loss: 0.1796 - accuracy: 0.9290 - val_loss: 0.8673 - val_accuracy: 0.7470
Epoch 16/35
63/63 [==============================] - 7s 105ms/step - loss: 0.1377 - accuracy: 0.9415 - val_loss: 0.9741 - val_accuracy: 0.7570
Epoch 17/35
63/63 [==============================] - 7s 104ms/step - loss: 0.1180 - accuracy: 0.9555 - val_loss: 0.9153 - val_accuracy: 0.7800
Epoch 18/35
63/63 [==============================] - 7s 106ms/step - loss: 0.1204 - accuracy: 0.9645 - val_loss: 1.0730 - val_accuracy: 0.7710
Epoch 19/35
63/63 [==============================] - 7s 104ms/step - loss: 0.1089 - accuracy: 0.9660 - val_loss: 1.0154 - val_accuracy: 0.7390
Epoch 20/35
63/63 [==============================] - 7s 105ms/step - loss: 0.0605 - accuracy: 0.9760 - val_loss: 1.0701 - val_accuracy: 0.7620
Epoch 21/35
63/63 [==============================] - 7s 105ms/step - loss: 0.1138 - accuracy: 0.9680 - val_loss: 1.1304 - val_accuracy: 0.7580
Epoch 22/35
63/63 [==============================] - 7s 104ms/step - loss: 0.0632 - accuracy: 0.9835 - val_loss: 1.2115 - val_accuracy: 0.7740
Epoch 23/35
63/63 [==============================] - 7s 105ms/step - loss: 0.0656 - accuracy: 0.9795 - val_loss: 1.4243 - val_accuracy: 0.7760
Epoch 24/35
63/63 [==============================] - 7s 103ms/step - loss: 0.0598 - accuracy: 0.9805 - val_loss: 1.4312 - val_accuracy: 0.7420
Epoch 25/35
63/63 [==============================] - 7s 103ms/step - loss: 0.0436 - accuracy: 0.9865 - val_loss: 1.5184 - val_accuracy: 0.7810
Epoch 26/35
63/63 [==============================] - 7s 106ms/step - loss: 0.0705 - accuracy: 0.9780 - val_loss: 1.3087 - val_accuracy: 0.7520
Epoch 27/35
63/63 [==============================] - 7s 105ms/step - loss: 0.0534 - accuracy: 0.9855 - val_loss: 1.7865 - val_accuracy: 0.7340
Epoch 28/35
63/63 [==============================] - 7s 105ms/step - loss: 0.0489 - accuracy: 0.9820 - val_loss: 1.8088 - val_accuracy: 0.7400
Epoch 29/35
63/63 [==============================] - 7s 104ms/step - loss: 0.0358 - accuracy: 0.9880 - val_loss: 1.8951 - val_accuracy: 0.7540
Epoch 30/35
63/63 [==============================] - 7s 103ms/step - loss: 0.0438 - accuracy: 0.9915 - val_loss: 1.8861 - val_accuracy: 0.7680
Epoch 31/35
63/63 [==============================] - 7s 104ms/step - loss: 0.0477 - accuracy: 0.9860 - val_loss: 2.3322 - val_accuracy: 0.7440
Epoch 32/35
63/63 [==============================] - 7s 104ms/step - loss: 0.0317 - accuracy: 0.9920 - val_loss: 3.6170 - val_accuracy: 0.6910
Epoch 33/35
63/63 [==============================] - 7s 107ms/step - loss: 0.0368 - accuracy: 0.9885 - val_loss: 4.9790 - val_accuracy: 0.6520
Epoch 34/35
63/63 [==============================] - 7s 105ms/step - loss: 0.0603 - accuracy: 0.9845 - val_loss: 2.1298 - val_accuracy: 0.7670
Epoch 35/35
63/63 [==============================] - 7s 105ms/step - loss: 0.0468 - accuracy: 0.9860 - val_loss: 2.0142 - val_accuracy: 0.7470
```

We could clearly see that for this model the training accuracy is very high but the validation is very low . This clearly defines that the model is overfitting .
Hence, we cannot move forward with this model.

- <u>Training: 1500 images, Validation: 300 images, Test : 557 images:</u>

  We again initiated a small convnet with training data of 1500 images, validation 300 images and test 557 images. We tried to fit the model and check the accuracy of training and validation data set.

```
Epoch 1/30
125/125 [==============================] - 13s 92ms/step - loss: 0.6809 - accuracy: 0.5437 - val_loss: 0.6844 - val_accuracy: 0.6230
Epoch 2/30
125/125 [==============================] - 12s 91ms/step - loss: 0.6630 - accuracy: 0.5965 - val_loss: 0.6871 - val_accuracy: 0.5280
Epoch 3/30
125/125 [==============================] - 12s 91ms/step - loss: 0.6520 - accuracy: 0.6168 - val_loss: 0.6133 - val_accuracy: 0.6760
Epoch 4/30
125/125 [==============================] - 12s 90ms/step - loss: 0.5995 - accuracy: 0.6768 - val_loss: 0.5488 - val_accuracy: 0.7140
Epoch 5/30
125/125 [==============================] - 12s 91ms/step - loss: 0.5653 - accuracy: 0.7017 - val_loss: 0.5631 - val_accuracy: 0.7050
Epoch 6/30
125/125 [==============================] - 12s 92ms/step - loss: 0.5187 - accuracy: 0.7425 - val_loss: 0.5018 - val_accuracy: 0.7690
Epoch 7/30
125/125 [==============================] - 12s 92ms/step - loss: 0.4807 - accuracy: 0.7660 - val_loss: 0.4814 - val_accuracy: 0.7810
Epoch 8/30
125/125 [==============================] - 12s 92ms/step - loss: 0.4383 - accuracy: 0.7925 - val_loss: 0.4971 - val_accuracy: 0.7800
Epoch 9/30
125/125 [==============================] - 12s 91ms/step - loss: 0.3966 - accuracy: 0.8210 - val_loss: 0.5085 - val_accuracy: 0.7610
Epoch 10/30
125/125 [==============================] - 12s 91ms/step - loss: 0.3212 - accuracy: 0.8602 - val_loss: 0.5722 - val_accuracy: 0.7560
Epoch 11/30
125/125 [==============================] - 12s 92ms/step - loss: 0.2871 - accuracy: 0.8733 - val_loss: 0.5151 - val_accuracy: 0.7960
Epoch 12/30
125/125 [==============================] - 12s 91ms/step - loss: 0.1958 - accuracy: 0.9170 - val_loss: 0.5780 - val_accuracy: 0.8010
Epoch 13/30
125/125 [==============================] - 12s 91ms/step - loss: 0.1619 - accuracy: 0.9350 - val_loss: 0.6761 - val_accuracy: 0.8020
Epoch 14/30
125/125 [==============================] - 12s 91ms/step - loss: 0.1091 - accuracy: 0.9600 - val_loss: 0.6995 - val_accuracy: 0.7990
Epoch 15/30
125/125 [==============================] - 12s 92ms/step - loss: 0.0578 - accuracy: 0.9815 - val_loss: 0.7712 - val_accuracy: 0.8030
Epoch 16/30
125/125 [==============================] - 12s 91ms/step - loss: 0.0638 - accuracy: 0.9780 - val_loss: 0.9803 - val_accuracy: 0.7960
Epoch 17/30
125/125 [==============================] - 12s 91ms/step - loss: 0.0446 - accuracy: 0.9865 - val_loss: 0.9361 - val_accuracy: 0.7810
Epoch 18/30
125/125 [==============================] - 12s 92ms/step - loss: 0.0705 - accuracy: 0.9753 - val_loss: 0.8975 - val_accuracy: 0.7990
Epoch 19/30
125/125 [==============================] - 12s 91ms/step - loss: 0.0297 - accuracy: 0.9895 - val_loss: 0.9067 - val_accuracy: 0.7990
Epoch 20/30
125/125 [==============================] - 12s 92ms/step - loss: 0.0101 - accuracy: 0.9980 - val_loss: 1.0234 - val_accuracy: 0.7930
Epoch 21/30
125/125 [==============================] - 12s 91ms/step - loss: 0.0111 - accuracy: 0.9975 - val_loss: 1.1759 - val_accuracy: 0.8030
Epoch 22/30
125/125 [==============================] - 12s 91ms/step - loss: 0.0426 - accuracy: 0.9872 - val_loss: 1.0098 - val_accuracy: 0.7870
Epoch 23/30
125/125 [==============================] - 12s 92ms/step - loss: 0.0355 - accuracy: 0.9877 - val_loss: 1.0167 - val_accuracy: 0.8080
Epoch 24/30
125/125 [==============================] - 12s 92ms/step - loss: 0.0067 - accuracy: 0.9985 - val_loss: 1.1345 - val_accuracy: 0.7920
Epoch 25/30
125/125 [==============================] - 12s 90ms/step - loss: 0.0087 - accuracy: 0.9975 - val_loss: 1.0628 - val_accuracy: 0.8020
Epoch 26/30
125/125 [==============================] - 12s 91ms/step - loss: 0.0052 - accuracy: 0.9985 - val_loss: 1.1432 - val_accuracy: 0.8120
Epoch 27/30
125/125 [==============================] - 12s 91ms/step - loss: 0.0075 - accuracy: 0.9973 - val_loss: 1.2237 - val_accuracy: 0.7880
Epoch 28/30
125/125 [==============================] - 12s 92ms/step - loss: 0.0053 - accuracy: 0.9987 - val_loss: 1.0887 - val_accuracy: 0.7970
Epoch 29/30
125/125 [==============================] - 12s 91ms/step - loss: 0.0100 - accuracy: 0.9975 - val_loss: 1.1879 - val_accuracy: 0.7800
Epoch 30/30
125/125 [==============================] - 12s 91ms/step - loss: 0.1426 - accuracy: 0.9482 - val_loss: 0.9011 - val_accuracy: 0.7880
```

We can again see that the accuracy for training set is 94.82% and accuracy for validation dataset is 78.80%. We can say that the model is overfitting.

We can again see that the model is overfitting. Now for improving the accuracy of the model we are going to try data augmentation and dropout method. We are randomly flipping, rotating and zooming the images for proper training. Now we can see that Model is converging at a good rate , neither too fast nor too slow and training and validation accuracy are moving at almost similar rates.

```
Epoch 1/55
125/125 [==============================] - 14s 98ms/step - loss: 0.6923 - accuracy: 0.5200 - val_loss: 0.6890 - val_accuracy: 0.5420
Epoch 2/55
125/125 [==============================] - 12s 97ms/step - loss: 0.6900 - accuracy: 0.5225 - val_loss: 0.6904 - val_accuracy: 0.5300
Epoch 3/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6870 - accuracy: 0.5345 - val_loss: 0.6915 - val_accuracy: 0.5150
Epoch 4/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6867 - accuracy: 0.5355 - val_loss: 0.6863 - val_accuracy: 0.5470
Epoch 5/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6843 - accuracy: 0.5425 - val_loss: 0.6762 - val_accuracy: 0.5530
Epoch 6/55
125/125 [==============================] - 12s 97ms/step - loss: 0.6839 - accuracy: 0.5410 - val_loss: 0.6586 - val_accuracy: 0.6020
Epoch 7/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6818 - accuracy: 0.5595 - val_loss: 0.6717 - val_accuracy: 0.5560
Epoch 8/55
125/125 [==============================] - 12s 97ms/step - loss: 0.6724 - accuracy: 0.5900 - val_loss: 0.6317 - val_accuracy: 0.6530
Epoch 9/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6764 - accuracy: 0.5723 - val_loss: 0.6676 - val_accuracy: 0.5950
Epoch 10/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6660 - accuracy: 0.6037 - val_loss: 0.6637 - val_accuracy: 0.5920
Epoch 11/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6546 - accuracy: 0.6085 - val_loss: 0.6600 - val_accuracy: 0.5930
Epoch 12/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6466 - accuracy: 0.6267 - val_loss: 0.6071 - val_accuracy: 0.6750
Epoch 13/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6343 - accuracy: 0.6465 - val_loss: 0.6497 - val_accuracy: 0.6130
Epoch 14/55
125/125 [==============================] - 13s 104ms/step - loss: 0.6220 - accuracy: 0.6553 - val_loss: 0.5792 - val_accuracy: 0.7130
Epoch 15/55
125/125 [==============================] - 12s 96ms/step - loss: 0.6137 - accuracy: 0.6715 - val_loss: 0.5985 - val_accuracy: 0.6770
Epoch 16/55
125/125 [==============================] - 12s 96ms/step - loss: 0.5986 - accuracy: 0.6785 - val_loss: 0.5596 - val_accuracy: 0.7160
Epoch 17/55
125/125 [==============================] - 12s 97ms/step - loss: 0.5787 - accuracy: 0.6935 - val_loss: 0.5569 - val_accuracy: 0.7170
Epoch 18/55
125/125 [==============================] - 12s 97ms/step - loss: 0.5815 - accuracy: 0.6948 - val_loss: 0.6015 - val_accuracy: 0.6710
Epoch 19/55
125/125 [==============================] - 12s 97ms/step - loss: 0.5655 - accuracy: 0.7140 - val_loss: 0.5303 - val_accuracy: 0.7290
Epoch 20/55
125/125 [==============================] - 12s 97ms/step - loss: 0.5618 - accuracy: 0.7097 - val_loss: 0.5450 - val_accuracy: 0.7260
Epoch 21/55
125/125 [==============================] - 12s 96ms/step - loss: 0.5544 - accuracy: 0.7240 - val_loss: 0.5462 - val_accuracy: 0.7140
Epoch 22/55
125/125 [==============================] - 12s 97ms/step - loss: 0.5409 - accuracy: 0.7283 - val_loss: 0.5254 - val_accuracy: 0.7670
Epoch 23/55
125/125 [==============================] - 12s 96ms/step - loss: 0.5269 - accuracy: 0.7368 - val_loss: 0.5513 - val_accuracy: 0.7170
Epoch 24/55
125/125 [==============================] - 12s 97ms/step - loss: 0.5308 - accuracy: 0.7305 - val_loss: 0.4978 - val_accuracy: 0.7630
Epoch 25/55
125/125 [==============================] - 12s 96ms/step - loss: 0.5239 - accuracy: 0.7427 - val_loss: 0.4959 - val_accuracy: 0.7540
Epoch 26/55
125/125 [==============================] - 12s 97ms/step - loss: 0.5196 - accuracy: 0.7435 - val_loss: 0.5295 - val_accuracy: 0.7490
Epoch 27/55
125/125 [==============================] - 12s 96ms/step - loss: 0.5138 - accuracy: 0.7552 - val_loss: 0.4568 - val_accuracy: 0.7900
Epoch 28/55
125/125 [==============================] - 12s 96ms/step - loss: 0.5074 - accuracy: 0.7445 - val_loss: 0.4683 - val_accuracy: 0.7810
Epoch 29/55
125/125 [==============================] - 12s 96ms/step - loss: 0.5168 - accuracy: 0.7398 - val_loss: 0.4932 - val_accuracy: 0.7660
Epoch 30/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4952 - accuracy: 0.7675 - val_loss: 0.5015 - val_accuracy: 0.7590
Epoch 31/55
125/125 [==============================] - 12s 97ms/step - loss: 0.5064 - accuracy: 0.7527 - val_loss: 0.4509 - val_accuracy: 0.7920
Epoch 32/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4735 - accuracy: 0.7760 - val_loss: 0.4548 - val_accuracy: 0.7980
Epoch 33/55
125/125 [==============================] - 12s 96ms/step - loss: 0.4772 - accuracy: 0.7715 - val_loss: 0.4658 - val_accuracy: 0.7890
Epoch 34/55
125/125 [==============================] - 12s 96ms/step - loss: 0.4919 - accuracy: 0.7628 - val_loss: 0.4752 - val_accuracy: 0.7700
Epoch 35/55
125/125 [==============================] - 12s 96ms/step - loss: 0.4764 - accuracy: 0.7703 - val_loss: 0.4396 - val_accuracy: 0.7900
Epoch 36/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4658 - accuracy: 0.7785 - val_loss: 0.4359 - val_accuracy: 0.7940
Epoch 37/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4658 - accuracy: 0.7825 - val_loss: 0.4090 - val_accuracy: 0.8070
Epoch 38/55
125/125 [==============================] - 13s 98ms/step - loss: 0.4595 - accuracy: 0.7793 - val_loss: 0.4437 - val_accuracy: 0.7930
Epoch 39/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4593 - accuracy: 0.7857 - val_loss: 0.4172 - val_accuracy: 0.8070
Epoch 40/55
125/125 [==============================] - 13s 97ms/step - loss: 0.4549 - accuracy: 0.7895 - val_loss: 0.4227 - val_accuracy: 0.8030
Epoch 41/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4603 - accuracy: 0.7797 - val_loss: 0.4293 - val_accuracy: 0.7930
Epoch 42/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4457 - accuracy: 0.7890 - val_loss: 0.3992 - val_accuracy: 0.8230
Epoch 43/55
125/125 [==============================] - 13s 98ms/step - loss: 0.4224 - accuracy: 0.8058 - val_loss: 0.4018 - val_accuracy: 0.8100
Epoch 44/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4389 - accuracy: 0.7993 - val_loss: 0.4225 - val_accuracy: 0.7940
Epoch 45/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4279 - accuracy: 0.8055 - val_loss: 0.4225 - val_accuracy: 0.8040
Epoch 46/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4217 - accuracy: 0.8050 - val_loss: 0.3966 - val_accuracy: 0.8070
Epoch 47/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4230 - accuracy: 0.8098 - val_loss: 0.3899 - val_accuracy: 0.8250
Epoch 48/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4247 - accuracy: 0.8065 - val_loss: 0.4181 - val_accuracy: 0.8160
Epoch 49/55
125/125 [==============================] - 12s 96ms/step - loss: 0.4139 - accuracy: 0.8062 - val_loss: 0.3707 - val_accuracy: 0.8310
Epoch 50/55
125/125 [==============================] - 12s 97ms/step - loss: 0.3949 - accuracy: 0.8280 - val_loss: 0.3664 - val_accuracy: 0.8320
Epoch 51/55
125/125 [==============================] - 12s 97ms/step - loss: 0.3948 - accuracy: 0.8230 - val_loss: 0.3822 - val_accuracy: 0.8300
Epoch 52/55
125/125 [==============================] - 12s 96ms/step - loss: 0.4182 - accuracy: 0.8040 - val_loss: 0.3767 - val_accuracy: 0.8380
Epoch 53/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4028 - accuracy: 0.8165 - val_loss: 0.3807 - val_accuracy: 0.8340
Epoch 54/55
125/125 [==============================] - 13s 99ms/step - loss: 0.4100 - accuracy: 0.8148 - val_loss: 0.3933 - val_accuracy: 0.8200
Epoch 55/55
125/125 [==============================] - 12s 97ms/step - loss: 0.4058 - accuracy: 0.8152 - val_loss: 0.3927 - val_accuracy: 0.8330
```

- <u>Using Pretrained Model: RestNet101</u>

  In this model we have converted the data into binary format i.e. Emergency vehicles will be denoted by “1” and non-emergency vehicles will be denoted by “0”.

	image_names	emergency_or_not
0	1503.jpg	0
1	1420.jpg	0
2	1764.jpg	0
3	1356.jpg	0
4	1117.jpg	0

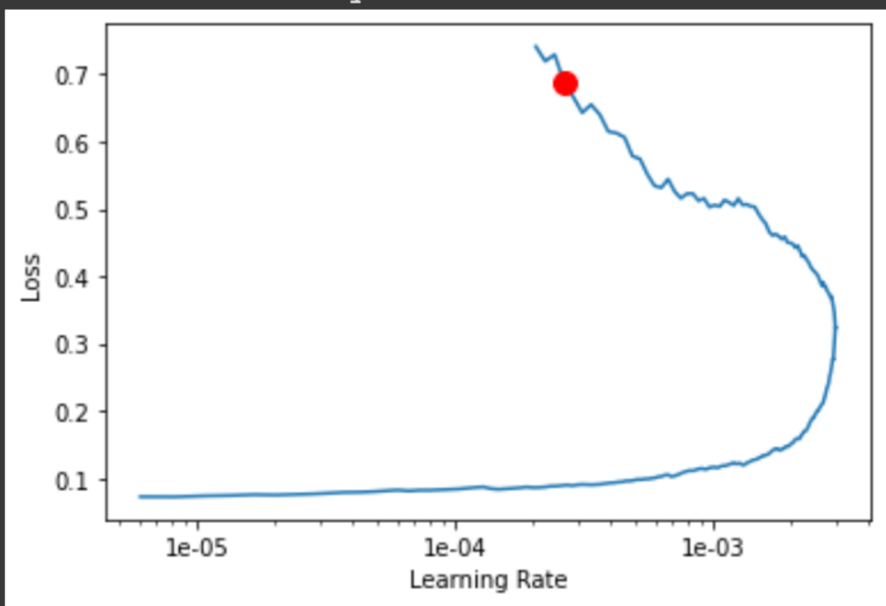
We divided the data set into Training, validation and Test Dataset . We have 1317 images in training dataset and 329 images in validation dataset. We imported the pretrained model from torchvision(RestNet101).RestNet 101 is a convolutional neural network that is 101 layers deep. This pretrained network can classify images into 1000 object categories, such as keyboard , mouse, mirrors, tires, rims etc . The model which will be perfect will be RestNet 101 as its already been trained on millions of images and the weights are set previously. We tried to fit the model and check the accuracy of the model using RestNet 101 by running on 15 epochs.

epoch	train_loss	valid_loss	error_rate	accuracy	time
0	0.612559	0.495688	0.155015	0.844985	00:26
1	0.515480	0.568281	0.124620	0.875380	00:25
2	0.428828	0.479780	0.109422	0.890577	00:25
3	0.370178	0.405741	0.103343	0.896657	00:25
4	0.312689	0.298472	0.085106	0.914894	00:26
5	0.254243	0.243672	0.085106	0.914894	00:25
6	0.205733	0.270777	0.088146	0.911854	00:25
7	0.169888	0.245255	0.088146	0.911854	00:25
8	0.142974	0.203118	0.066869	0.933131	00:26
9	0.125440	0.274988	0.094225	0.905775	00:25
10	0.115558	0.256520	0.100304	0.899696	00:25
11	0.100127	0.226366	0.075988	0.924012	00:25
12	0.090127	0.221575	0.075988	0.924012	00:25
13	0.082397	0.217502	0.079027	0.920973	00:25
14	0.077203	0.222426	0.082067	0.917933	00:26

But as we can see that the accuracy of the model is decreasing after 9 epochs we will stop there as the model is overfitting. Its clearly visible that the accuracy for this model is much more higher as compared to the other two model. The accuracy for this model is 93.31%

The plot between learning rate and loss is as mentioned

```
Min numerical gradient: 2.66E-04
Min loss divided by 10: 8.23E-07
```



We can see clearly that the learning rate is declining as the loss is increasing. We can also see the optimal learning rate for the model is 0.000265

Getting the optimal learning rate

```
[ ] opt_lr = model_rnet.recorder.min_grad_lr
    print("\n Optimum learning rate to be used: ", opt_lr)

 Optimum learning rate to be used:  0.00026573657332919914
```

Learning rate is used to scale the magnitude of parameter updates during gradient descent.
Hence with the best accuracy in all the three models, we are going to go forward with Pretrained Model i.e ***RestNet-101.***

<u>PERFORMANCE METRICS:</u>

The first thing we will see here is the ROC curve and we can determine whether our ROC curve is good or not by looking at the AUC (Area under curve) and other parameters which are also called as confusion metrics. Confusion metrics is used to describe the performance of a classification model on a set of test data for which the true values are known. All the measures except AUC can be calculated by using left most four parameters. Let's understand those four parameters first. True positive and True negative are the observations that are truly predicted whereas we want to minimize the false positives and the false negatives. Let's understand these terms one by one:

True Positives (TP) - These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes.

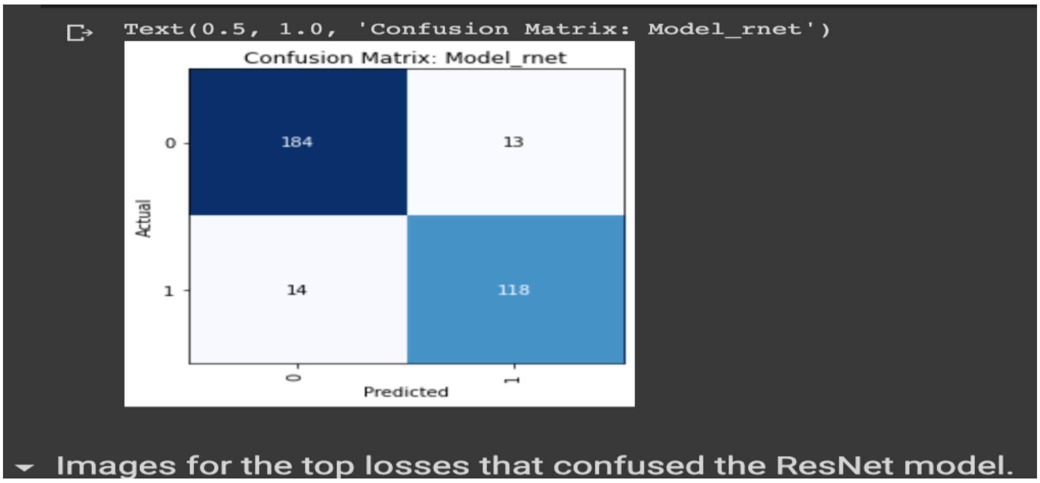
True Negatives (TN) - These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no.

False positives and false negatives, these values occur when actual class contradicts with the predicted class.

False Positives (FP) – When actual class is no and predicted class is yes.

False Negatives (FN) – When actual class is yes but predicted class in no.

The confusion matrix for our model is :



Once we get the basic understanding of these 4 parameters we can calculate the Accuracy, Precision, Recall and F1 score.

Accuracy:

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when we have symmetric datasets where values of false positive and false negatives are almost same. Therefore, we have to look at other parameters to evaluate the performance of your model.

Precision:

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate.

Recall:

Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived.

F1 score:

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if we have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it’s better to look at both Precision and Recall.
These 4 are the important parameters to understand how good our model has performed.

For our model, we have got 0.9331 as accuracy which means our model is approx. 93.31% accurate.

Here we only want ratio of correctly predicted observation to the total observations. Hence we are considering accuracy as our performance metric.

Conclusion:

The purpose of this model was to predict the approach of an emergency vehicle in traffic. To do that we made a model which helps to detect the emergency vehicles through the cameras installed on the traffic signal. We trained over 1646 images in our dataset and we tested over 711 images to use in the model. The accuracy that we got was 0.933131which is the best among all the models trained. The other 2 models that we trained were not as accurate as the pre-trained model. For the first model we trained over 1000 images and validated 600 images and tested 757 images but the results were not accurate whereas for the second model we trained 1500 images and validated 300 images and tested 557 images yet the accuracy was not good as for the first model we maximum accuracy as 74.70% and for the second model we got accuracy as 78.80% . So, we decided to go with the third model where the accuracy rate is 0.933131 i.e. 93.31%. With the help of this model we could detect an emergency vehicle from 50 meters away from the signal(installed camera) and it would activate the sensors installed on the signals which would give us enough time to stop all the signals and make way for the emergency vehicle so there is no chance of colliding with other cars as the signals have already been stopped to make way for the emergency vehicle . We could also install some emergency sound on the signals and as soon as the model detect that an emergency vehicle is approaching that sound will start playing and people could give way for the emergency vehicle before it reaches the signal. The main objective to save as many as lives , reaching at its destination is also being achieved. We could be sure that the model would help us reduce the casualties which were happening due to traffic congestion and due to the delay of emergency vehicle reaching its destination on time. And ultimately we would be able to save millions of lives of people who are facing the crisis just because the emergency response vehicles are not able to reach to them on time.

Reference:

https://www.mathworks.com/help/deeplearning/ref/resnet101.html#:~:text=ResNet%2D101%20is%20a%20convolutional,%2C%20pencil%2C%20and%20many%20animals.

https://radiopaedia.org/articles/epoch-machine-learning?lang=us#:~:text=An%20epoch%20is%20a%20term,of%20data%20is%20very%20large).&text=Many%20models%20are%20created%20with%20more%20than%20one%20epoch.

https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/

******End of Report******