Postman API

Round 3 AIML Task

Akshay Shukla

Data Preparation

From the roboflow website, I downloaded the Tensorflow Object Detection CSV format dataset. The dataset was divided into test, train and valid folders already. However within these folders the images for rock, paper and scissors were not divided. Each folder contains a csv file called annotations.csv which contains labels for classifying each image in the respective folder. Using this, I wrote a Python code to parse through the dataset and divide each of test, train and valid further into rock, paper and scissors folder, making it easier to train and test the model. This was done using the split_RPS_dataset_into_segmented_folders.ipynb file.

Initial Model

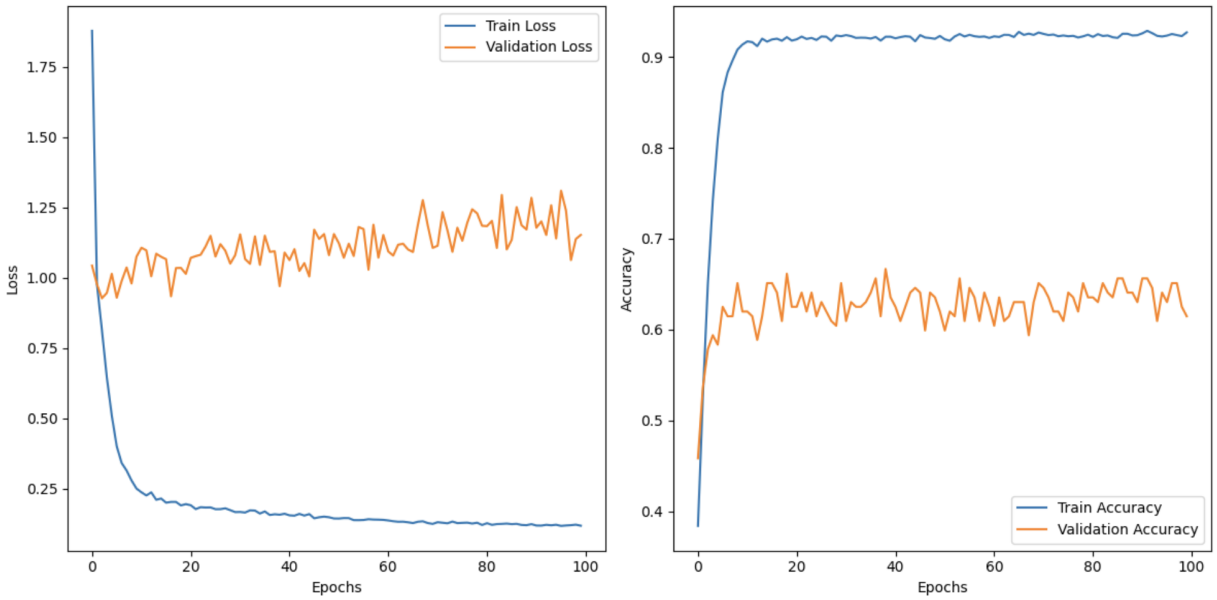
In order to start the training, I created a Conv2D layer with 32 features using a kernel size of 3*3 and a relu activation layer, followed by a flatten layer that would flatten the 3 dimension output from the first layer, a drop out layer to reduce overfitting and finally a softmax layer with 3 output as we needed three outputs corresponding to Paper, Rock and Scissor.

I decided to use Adam optimiser. Since I am using Mac Book Pro M2, I got a warning that Adam optimiser is not optimized for the device. I tried with legacy optimiser but got the same result. Since the warning was for optimization, I decided to continue using the new Adam optimizer. For the first time, I tried Adam optimizer with a learning schedule but realized that the optimizer works well without any explicitly defined learning schedule. With the learning schedule, the accuracy hovered around 40%. Without it, the accuracy jumped to above 90%.

Here's what I got on the 100th epoch -
Epoch 100/100

111/111 [==============================] - 29s 258ms/step - loss: 0.1185 - accuracy: 0.9270 - val_loss: 1.1522 - val_accuracy: 0.6146

So an accuracy of 92.7% and a validation accuracy of 61%

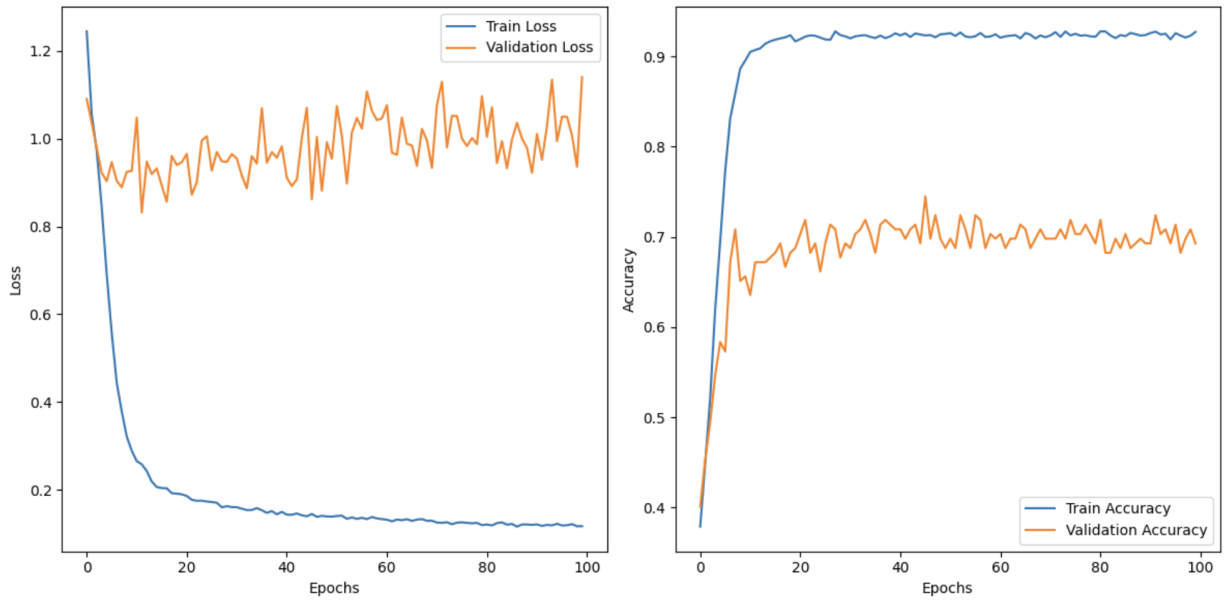


Add additional layers

I added a couple more dense and dropout layers to the above model and retried with 100 epochs.

Epoch 100/100
111/111 [==============================] - 24s 213ms/step - loss: 0.1169 - accuracy: 0.9272 - val_loss: 1.1403 - val_accuracy: 0.6927

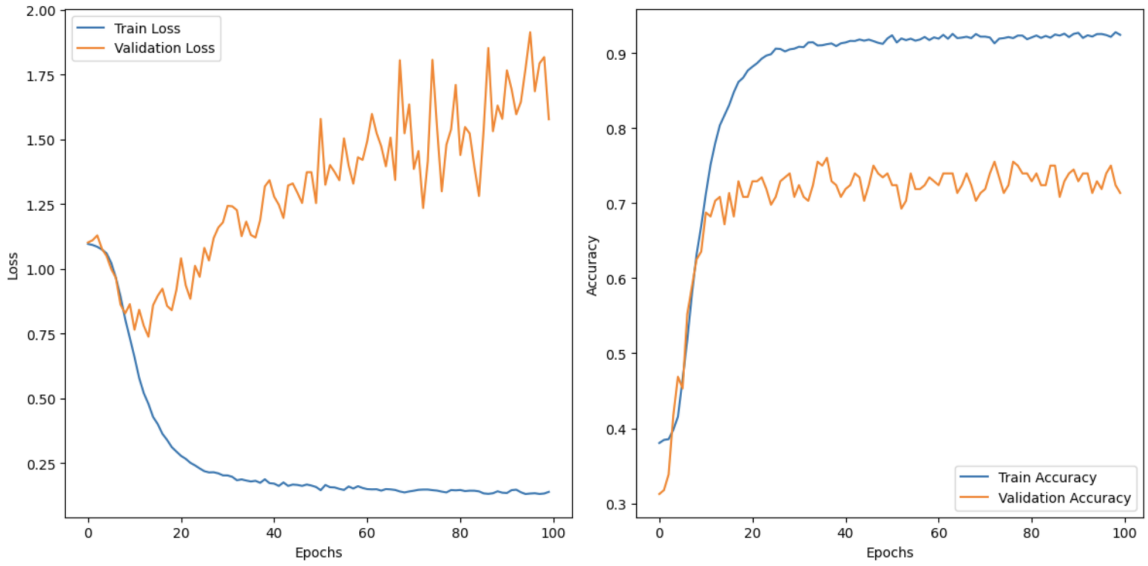
Accuracy remained at 92.72% but the validation accuracy increased fo 69.27%



Add even more layers

Encouraged by the results, I added a few more convolution layers and pooling layers.

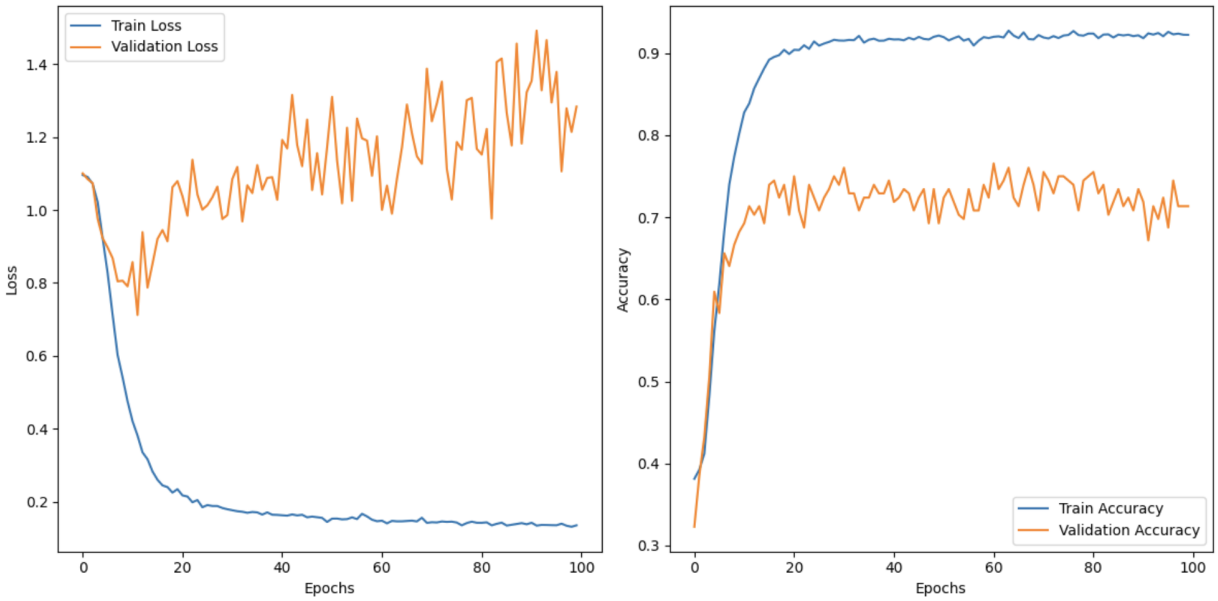
Epoch 100/100
111/111 [==============================] - 43s 388ms/step - loss: 0.1398 - accuracy: 0.9242 - val_loss: 1.5770 - val_accuracy: 0.7135
The accuracy remained more or less the same at 92.42% but the validation accuracy increased to 71.35%



Change Relu to Leaky Relu

Since Relu ignores output less than 0,, I thought I will try leaky relu. The accuracy as well as the validation accuracy remained close to before but the loss reduced significantly. I decided to use this as my final output.

Epoch 100/100
111/111 [==============================] - 42s 374ms/step - loss: 0.1352 - accuracy: 0.9223 - val_loss: 1.2837 - val_accuracy: 0.7135



The project has been divided into different ipynb files. All the different models tried have been saved in a single folder named “Different Models Tried”. While the training accuracy was similar for all of them, the testing accuracy kept on improving as described above. The best model was the one where leaky relu was used and has been saved in the file RPS_trial_model_leaky_relu.ipynb within this same folder.

Testing Model Accuracy

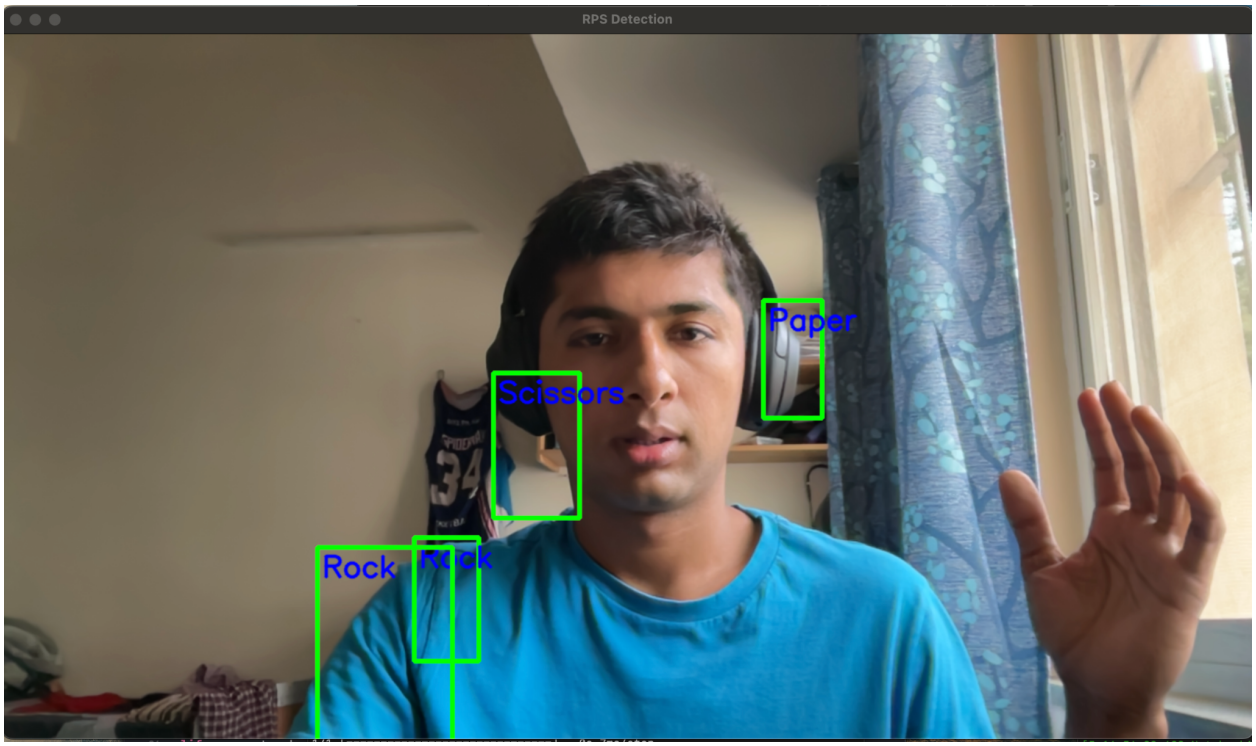
The file, RPS_evaluate.py contains code for retrieving images from the test folder of the dataset and testing it out with the model to check the accuracy. To try out the different models, we just have to change the name of the .h5 file of the model loaded into the code which happens in line 18. The result of the testing is given in the form of

a matrix with the actual values given on one axis and the predicted values given on the other.

Making the model to work with live video input

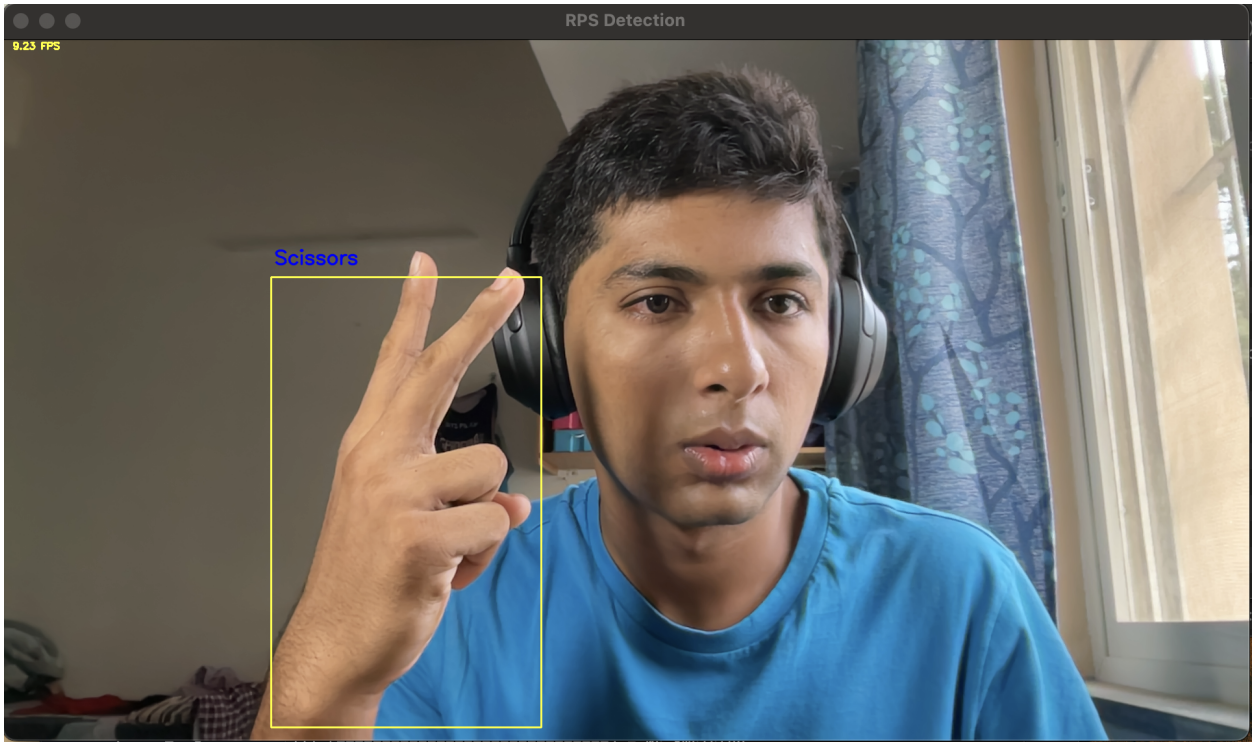
Using Haar Cascade

I implemented live video capture using CV2 which is from the OpenCV library. I tried to use Haar Cascade for hand recognition with video input but could not find a decent xml file that could do that deterministically. Most of the haar cascade files were not able to consistently detect the presence of a hand in the video input. The haar cascade files that I used have been saved in the haarcascades folder and the code for using these can be found in the RPS_test.py file. The names of the xml files to be used can be changed in the lines 36 and 39 and other files can be used as well because the code parses through a list of haar cascades. In order to test the code on pre-loaded video input rather than live input from the webcam, the line 22 should be uncommented and the line 19 should be commented.

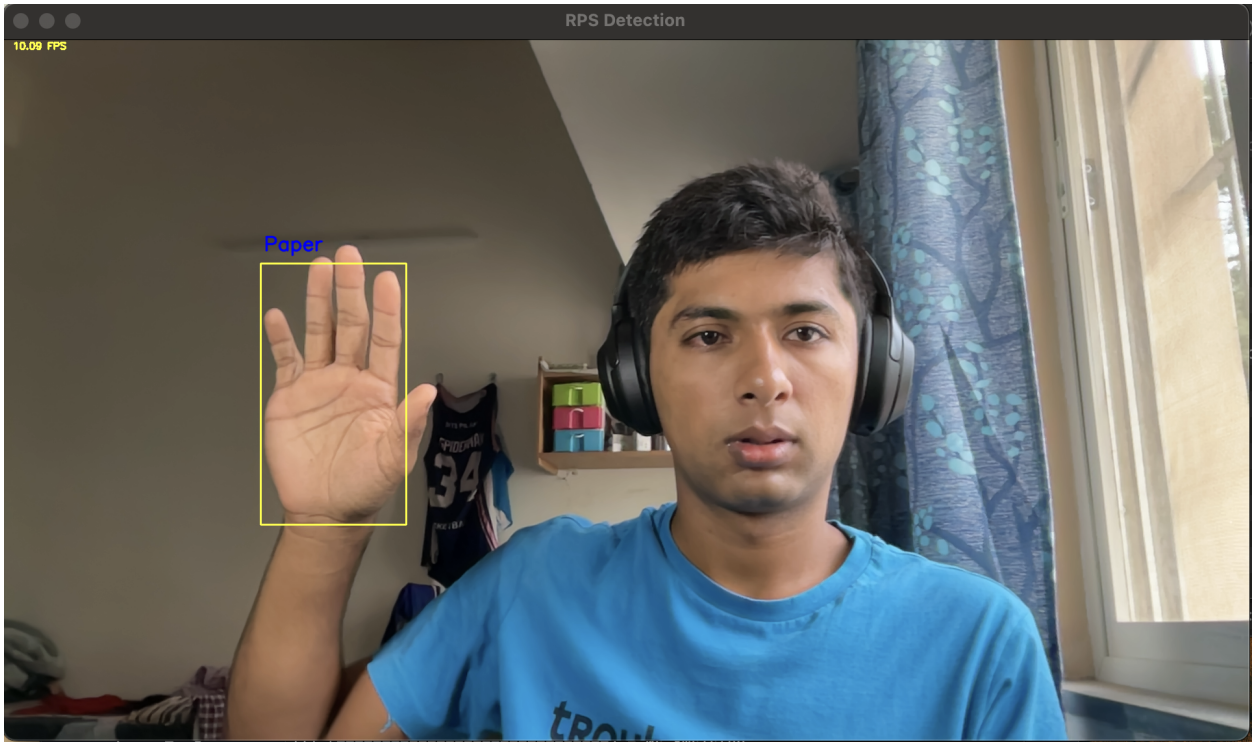


Using YOLO hand detection framework

I then found the YOLO hand detection framework. I used the model generated from Leaky Relu as described above. The code for the same can be found by navigating into the yolo-hand-detection-master folder within which we have to open TestRPSDetectorYolo.py. This gave me much better results.


RPS Detection
9.23 FPS
Scissors


RPS Detection
10.09 FPS
Rock


RPS Detection
10.09 FPS
Paper