Facial Expressions Detector

Computer Vision, Dr. Nourhan Zayed

Bassant Abdelaal Nada Ayman Youssef Hussien 900183162

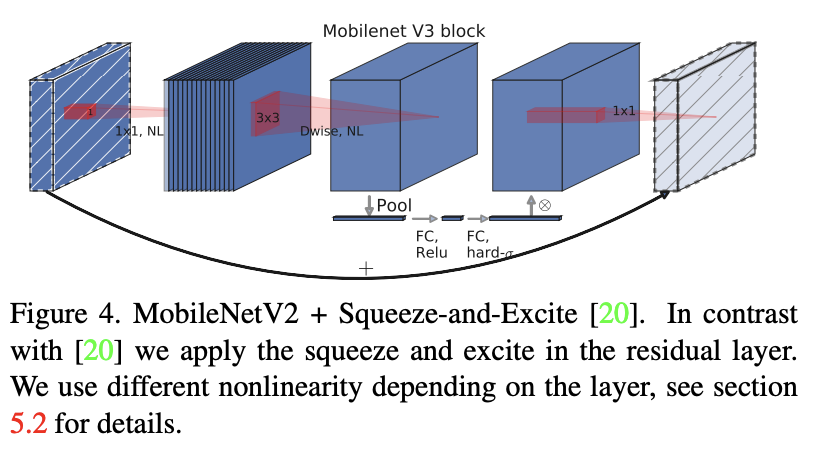
Spring 2022, The american univsersity in cairo

**Solution 2 Using CNNs and MobileNetV3**

There are two versions of this solution, one using the pretrained model MobileNetV3 and the other one is us building the full CNN architecture. Because MobileNetV3 has many weights it wasn’t that suitable to our problem as we ended with an average accuracy of around 47%. However, using our fully CNN architecture we could reach an average accuracy of 78%, all details of both solutions are explained below.

In this solution we used Keras and CNNs to build a model that is trained on the same dataset explained above. Then, we used MobileNetV3 architecture to have a pretrained model and then we tried different combinations of this, by using different optimizers, adding or deleting Dense layers or adding and deleting CNN layers. In the experimentation part below, we explain the different models alongside their results.

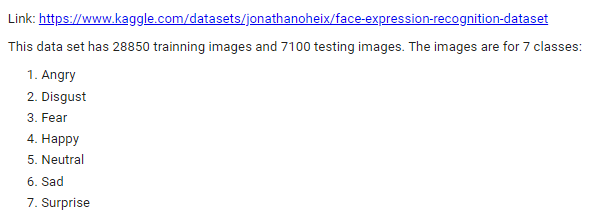
In this solution there are three components, the model that we are training on the data set using CNNs (FacialExpressionDetector.ipnyb) which has the code of the model and importing the dataset. This file exports to us the weights corresponding to the best model (bestModel.h5). The third and last part of this solution is the (videoDetector.py) which is a python file that uses OpenCV to take a real time video stream of the user and draw a box around his/her face. Then we use this to have a continuous stream of images which are then classified into one of the seven classes we are having in our data set. This classification is using the best model (bestModel.h5) that was trained earlier. Then videoDetector.py outputs to the user the label or the classification that is changing as the emotions of the person are changed overtime.

MobileNetV3 is a convolutional neural network that is tuned to mobile phone CPUs through a combination of hardware-aware network architecture search (NAS) complemented by the NetAdapt algorithm, and then subsequently improved through novel architecture advances. Advances include (1) complementary search techniques, (2) new efficient versions of nonlinearities practical for the mobile setting, (3) new efficient network design.

Explaining each part of the solution in detail

# First Explaining FacialExpressionDetector.ipnyb and FacialExpressionDetector(MobileNet).ipnyb

The first part of the model definition is getting the dataset

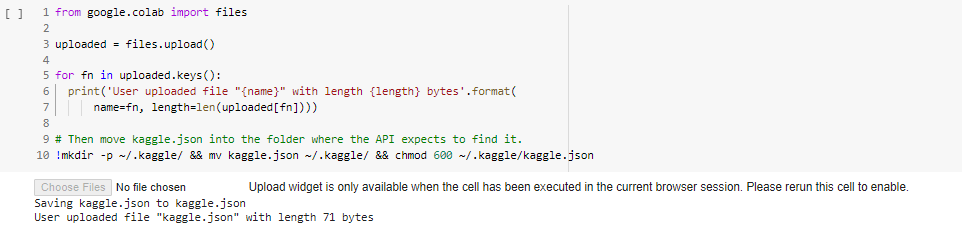


Then we connect to Kaggle API to retrieve the dataset directly and quickly.

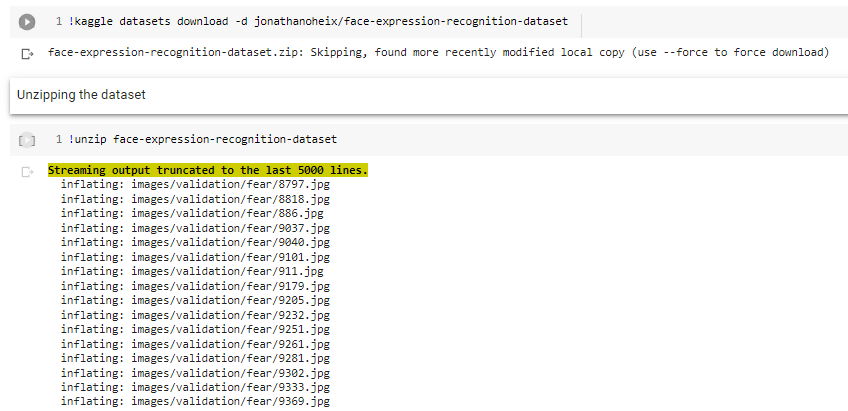
The first step in doing so is installing Kaggle API using



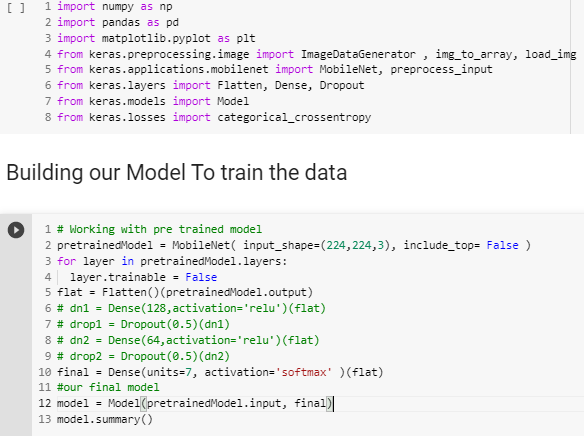
Then we enter the API key associated to my account on Kaggle that is saved in a kaggle.json file, after this authorization step we are connected directly to the platform and can get datasets directly from there.



Then we get our desired dataset and unzip it



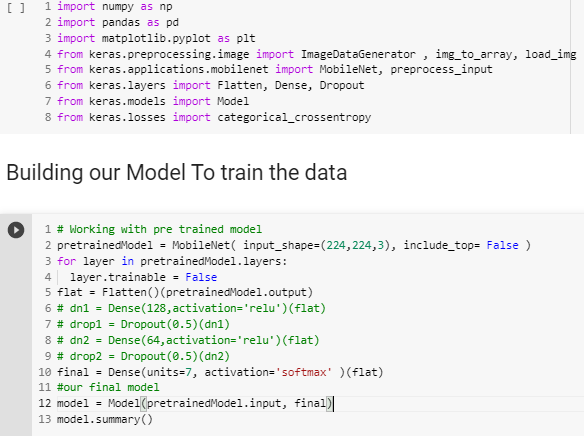
After that we import the needed libraries and build our model, which is a pretrained model importing MobileNetV3 model and adding a final 7-nodes output layer with activation softmax to detect the seven classes we are having.



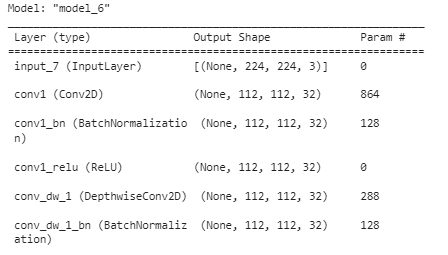
Then we display a bunch of images to make sure everything is correct

Afterwards, we start building our model, in the case of the fully CNN architecture we build our own architecture as shown below 

Afterwards in the case of MobileNetV3 we call the pretrained model and then add layers to it as shown in the following code

The model summary is printed and is shown in the following figure

The input to our model is a 224x224x3 set of images and goes through a series of convolution (CNN) layers which are 13 convolution layers with different activation functions and at the end we have a seven-node output layer with activation softmax. The total number of trainable parameters is 351239 which is quite sufficient to the amount of data we are having.

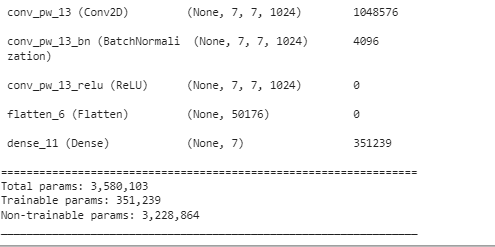


**.**

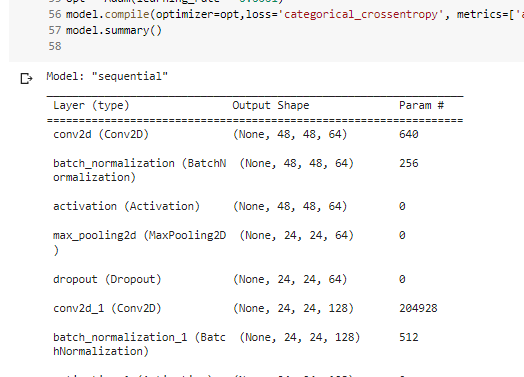
**.**

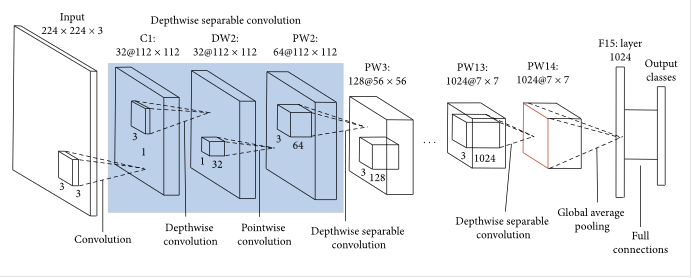
**.**

**.**



However in our fully connected CNN we changed the size of the pictures and made it 48x48x3 to not stuck in an overfitting hole.



An illustrative image of the architecture of MobileNetV3 can be shown below that was implemented in a second version of the CNN for testing purposes

After building the model we compile it with “Adam” optimizer

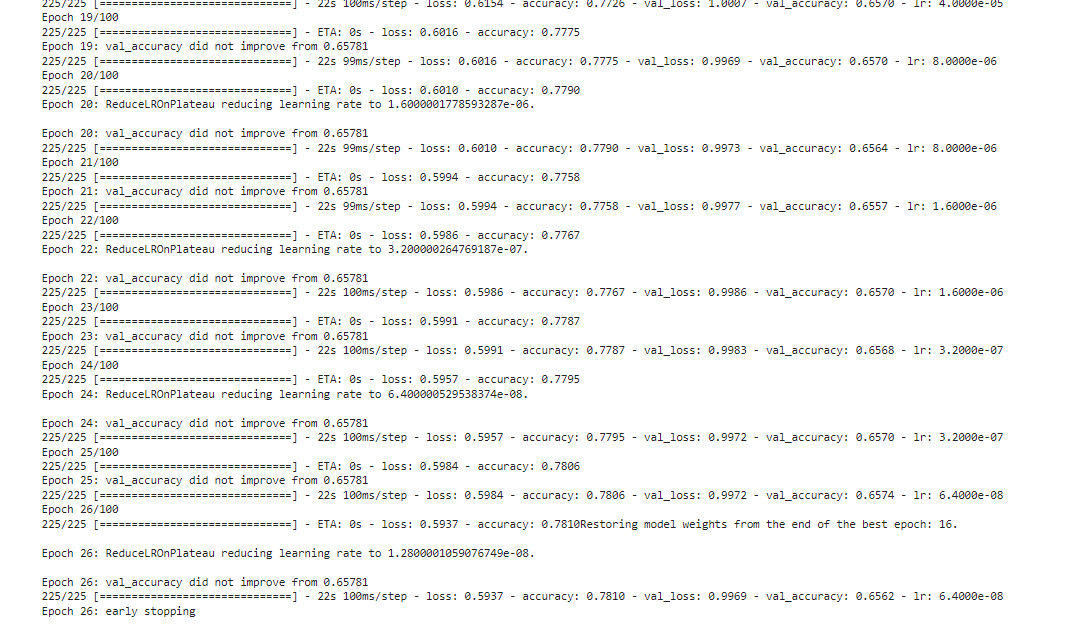


We tried other optimizers as shown in the experimentation section; however, Adam optimizer showed the best results.

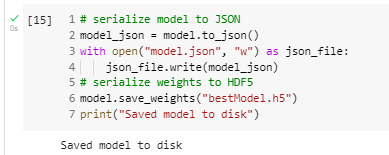
The next step was Data Augmentation as we didn’t have a large data set, so it was crucial to augment more images both to the training and the validation data. We used ImageDataGenerator of Keras as shown and we augmented the images to each of the seven classes 

**Important Checkpointing and Early Stopping:** We then train our model for 30 epochs while enabling checkpointing and early stopping. Checkpointing allows us to save the state of the model for every specific number of epochs so that we can return to it if we want. Early stopping which is also available in Keras allows us to stop training if there is no change in our desired metric. We measure the change in the validation accuracy and if there is no increase in the validation accuracy with a minimum of 0.01 for 8 consecutive epochs then the model stops the training and at each checkpoint it saves its weights to bestModel.h5 if the model at this epoch had an increase in the validation accuracy.

At the beginning we tried to run the model for 100 epochs but we found that this is not needed at all, because the Early stopping used to stop after 15-20 epochs, so we used 30 epochs as the standard for all the models and experiments afterwards. However, this is the result of training the model for 100 epochs.

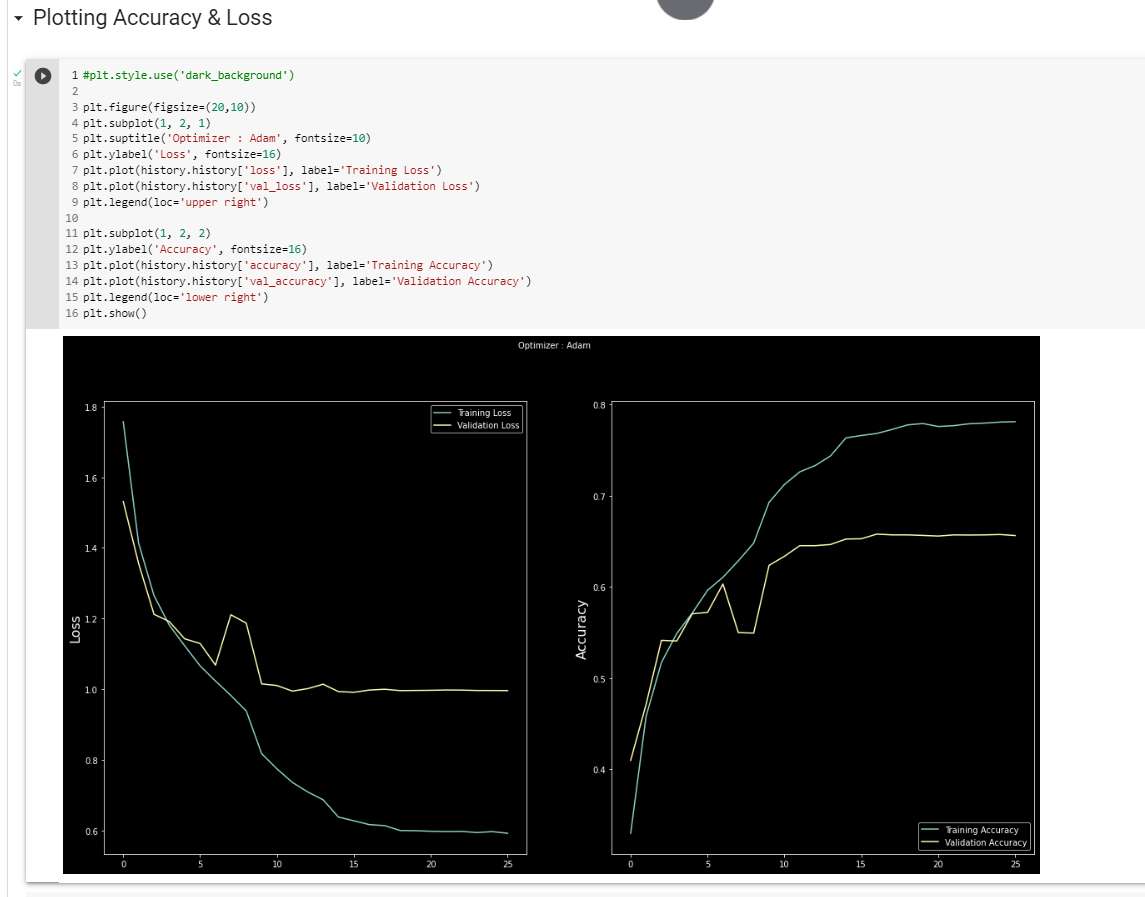


After the training we save the weights of the bestModel.h5 so that it can be used in the video detector to extract emotions from realtime video of the user.



Afterwards, we plot the accuracy vs the validation accuracy and the loss vs the validation loss to see the performance of the model:

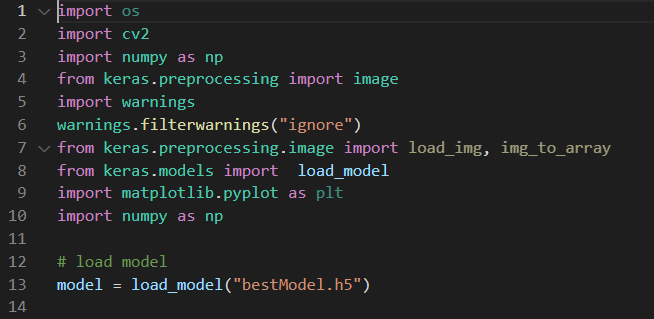
At each time after we train the model we test it on the testing data and see

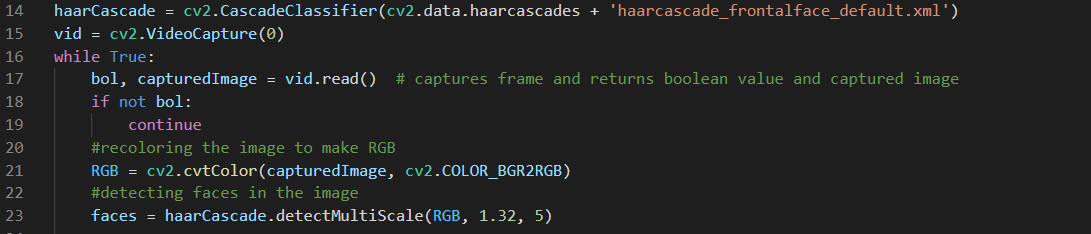


# Explaining VideoDetector.py

This python script is responsible for opening a video stream of the user and extracting facial expressions and detecting his/her emotions out of the seven classes we are having. This script uses the bestModel that is saved after training the CNN model.

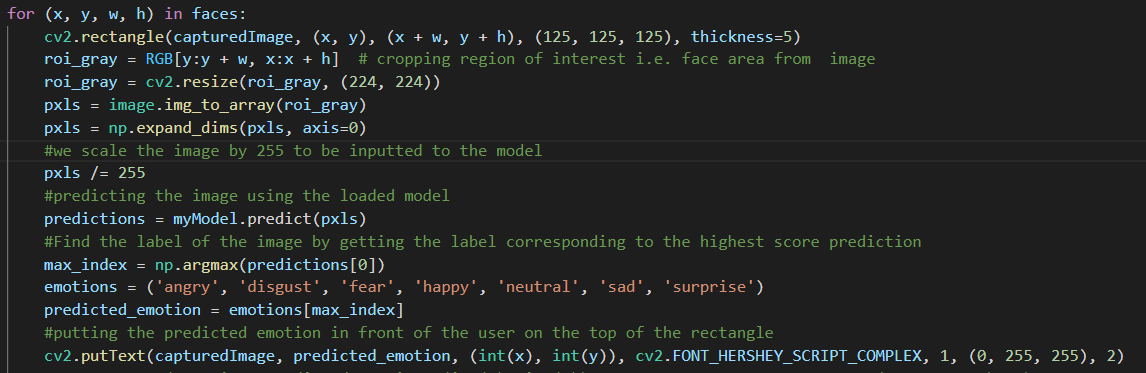
The first thing is getting the right libraries and loading the bestModel.h5

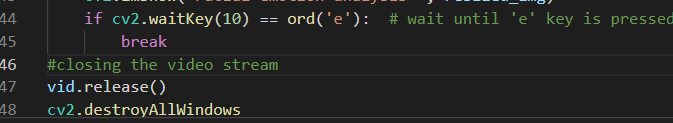
Then we create an instance of the HaarCascade classifier which is found in openCV. Haar Cascade classifiers are an effective way for object detection. This method was proposed by Paul Viola and Michael Jones in their paper Rapid Object Detection using a Boosted Cascade of Simple Features. Then we capture a continuous stream of images and for each image we convert it from BGR to RGB. When the image file is read with the OpenCV function imread(), the order of colors is BGR (blue, green, red). On the other hand, in Pillow, the order of colors is assumed to be RGB (red, green, blue).

Therefore, to use both the Pillow function and the OpenCV function, we need to convert BGR and RGB. Afterwards, for each image we use detectMultiScale() function, knowing that detectMultiScale function is used to detect the faces. This function will return a rectangle with coordinates(x,y,w,h) around the detected face.

Then we loop over each face in the image in case there are more than one face, and for each face we do the following:

1. We draw a rectangle around it using the coordinates that are returned from detectMultiScale()
2. Then we crop the region of interest i.e the face area in the image
3. Then we rescale the region of interest i.e the face by 255 which is a preprocessing step to do a normalization or regularization of all images by dividing each image by the RGB value 255.
4. Then we use our trained model to predict the label of the image and thus classify the emotion in the image
5. Then we write this emotion to the user on the top of the rectangle





If the user presses ‘e’ then it exits from the video stream:

Experimentation and Results of Solution 2

# As mentioned above we have two solutions one of them is the fully connected CNN which is our architecture.

# Results of CNN architecture

# Model 1 (Fully CNN without MobileNet)

# 

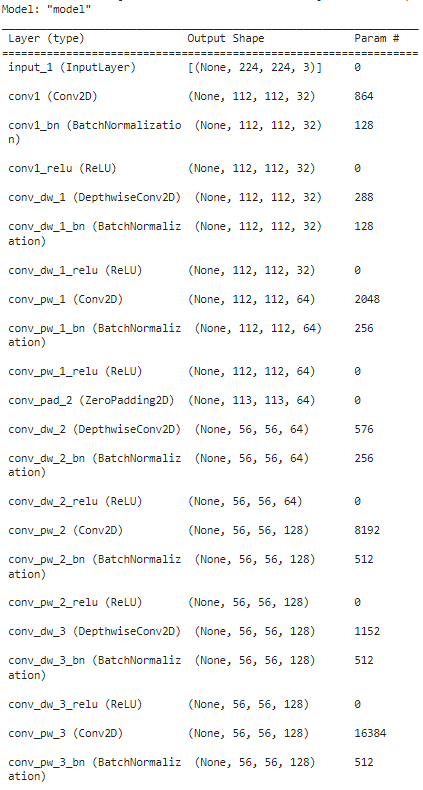
# 

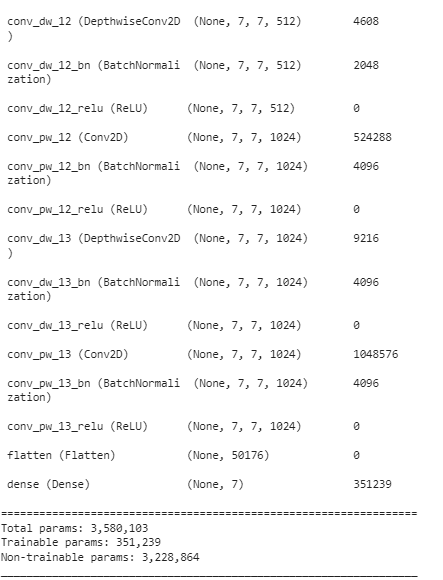
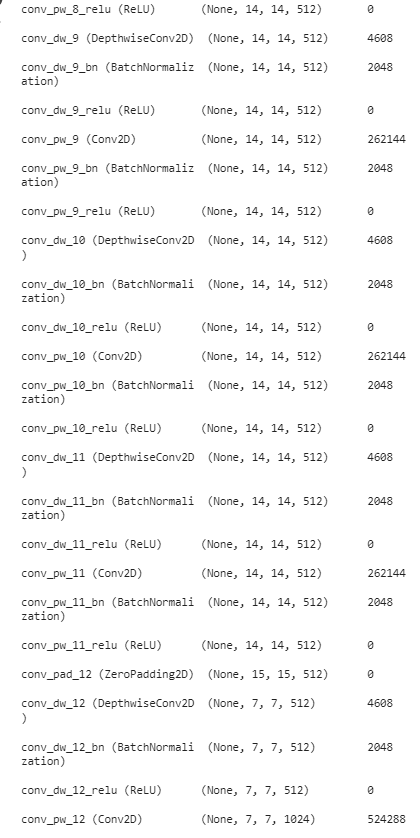
**Results of this part was that we reached an average accuracy of around 78% as can be seen from the below figure and this was the best model out of the different CNN models followed below.**

# 

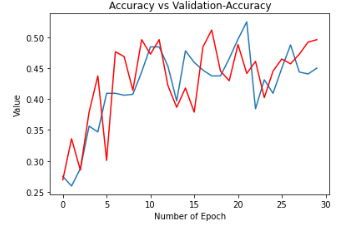
# Model 2 Using MobileNet

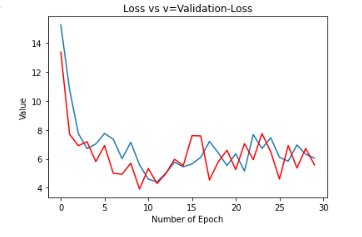
Original MobileNetV3 followed by a seven nodes output layer





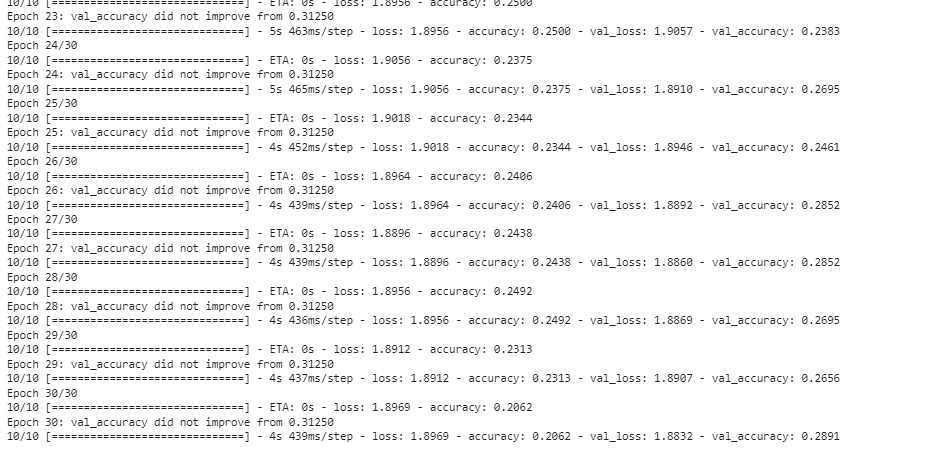
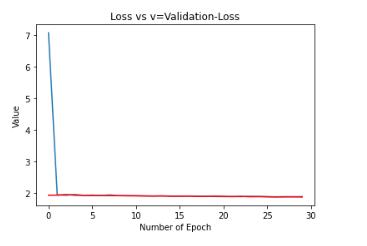
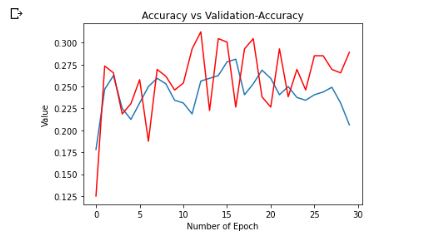
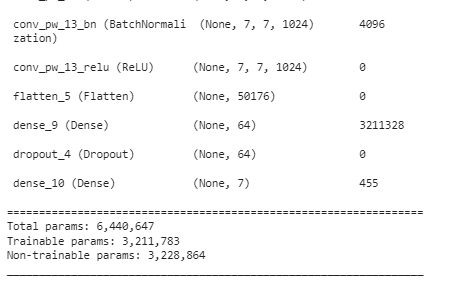
This model reached an accuracy of around 45~47%





# Model 2:

We added a dense layer with 64 nodes however, the number of trainable parameters increased tremendously and thus this led to overfitting due to the scarce of the data points we are having.

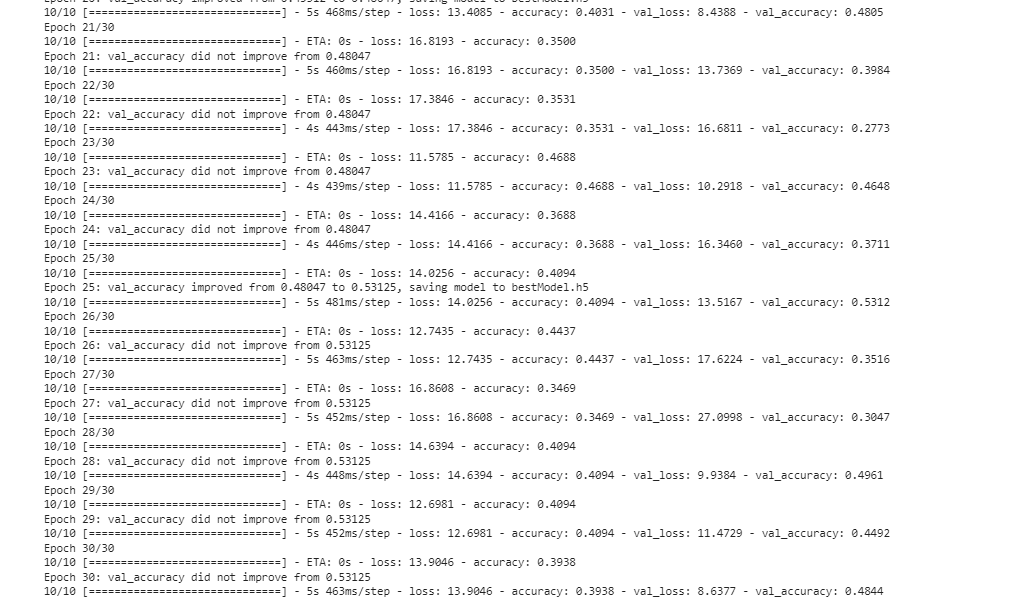


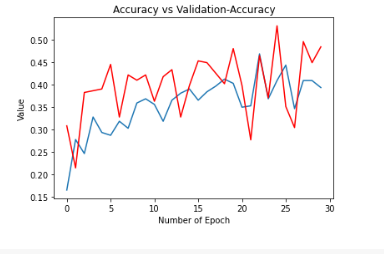
# Model 3:

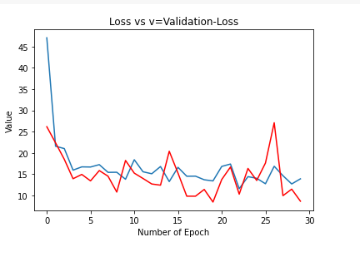
The same model as model one however, we used RMSProb instead of Adam as the optimized of our model.

This model was not better than model 1, rather it had an accuracy between 41~44%

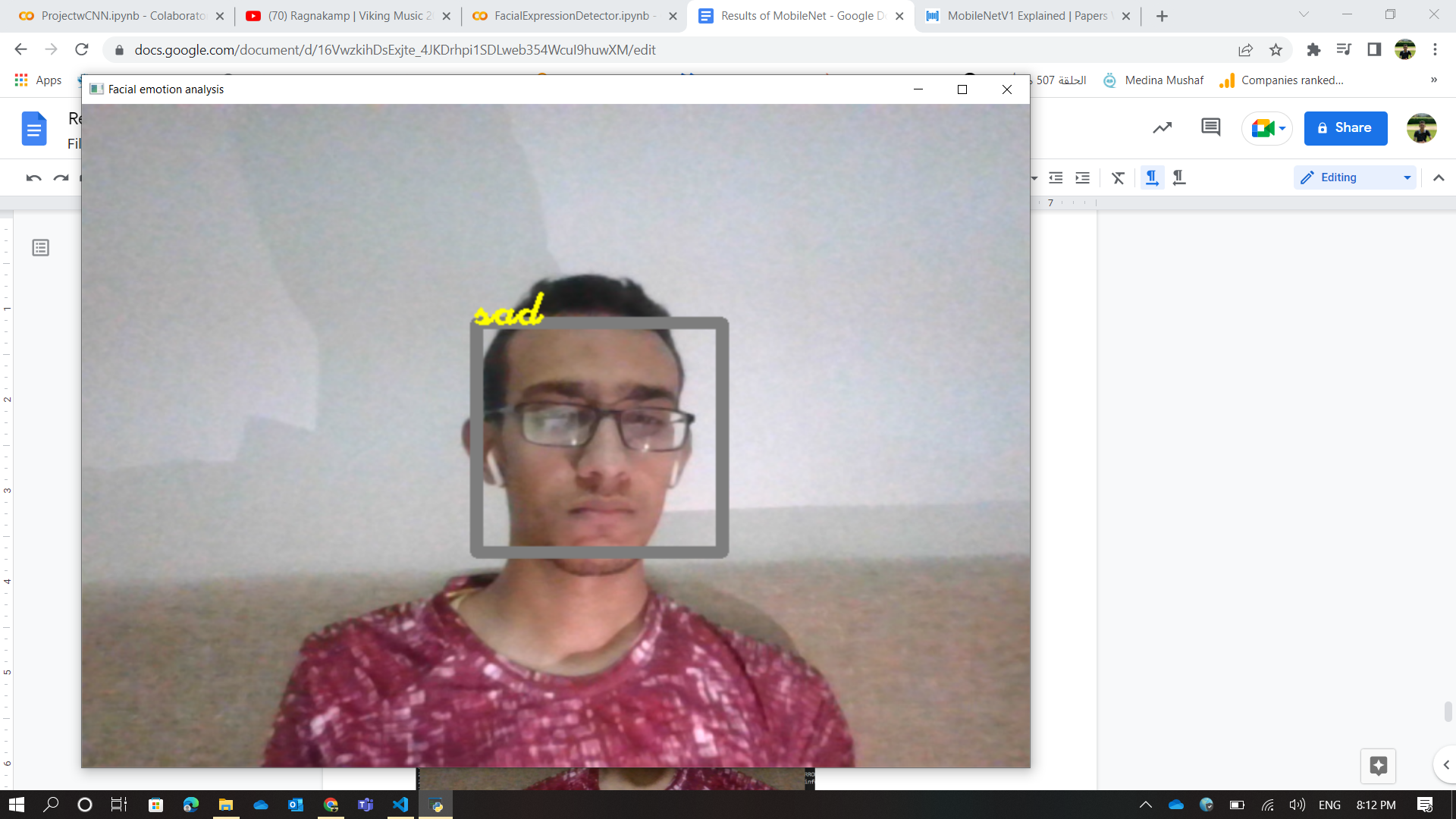








# VideoDetector Results

**

