**Phase 2 Weekly Report**

**Course Initial and name: Senior Project Design II ( 499B)**

**Project Name: Deepfake video detection**



Project Supervisor: Md. Ashrafuzzaman Khan (Azk)

Section: 07

Group Members:

Shayekh Mohiuddin Ahmed Navid-153124702

[Shamima Haque Priya-1620069042](https://github.com/fatema13/CSE499-04-FakeVideoDetection/tree/master/Shamima%20Haque%20Priya-1620069042)

[Umme Kulsum Ritu- 1511619042](https://github.com/fatema13/CSE499-04-FakeVideoDetection/tree/master/Umme%20Kulsum%20Ritu-%201511619042)

[Fatema Tuz Zohra- 1620191042](https://github.com/fatema13/CSE499-04-FakeVideoDetection/tree/master/Fatema%20Tuz%20Zohra-%201620191042)

Department of Electrical & Computer Engineering

North South University

**To be continued after phase 1**

**Week 5 (16/02/2020-22/02/2020)**

After phase 1, there was a group study of different papers that was provided by the facebook deepfake detection challenge. The papers were studied and different methods were implemented. Two of the papers that were chosen for implementation were Mesonet and Deepfake face extraction using simple features. The code that was provided along with the ‘Deepfake face extraction using simple features’ paper was preprocessed and was used for model training. Different methods of machine learning were applied here. During the initial stage, SVM was used for training the model, followed by random forest. After the implementation of these methods, a simple neural network was used to train the model to check and observe a better output.

In **simple features**, two different methods are being implemented. Fourier transformation and Radial profile. After the implementation of these methods, the classification layers are adjusted. The classifiers can be SVM, random forests or a neural network. For all of the classification layers, a small amount of datas was used. Considering there were some problems with the code during preprocessing or in the neural network architecture, the model gave a validation accuracy of 100% for all the different classifiers. However, when the model was run again using the Kaggle dataset, the model couldn’t identify or define any functions.Hence, it couldn’t predict whether the provided inputs were real or fake.

Conclusively, 20,000 datas were used and the accuracy achieved after model training was 17%. The accuracy was considered to be reliable as during the preprocessing stage, it was stated that the model was not able to differentiate whether the provided input was fake or real.

**Week 6 (23/02/2020-29/02/2020)**

This week, methods of ‘**Mesonet’** paper were used for implementation, along with the extraction of the entire dataset from the ‘facebook deepfake detection challenge’. During the initial stages, only 20% of the dataset was used. However, as different methods from different papers were implemented, it was necessary to use the entire dataset to see the peak difference that is observed during different architectures along with different classifiers being implemented. After the entire dataset was extracted, methods of ‘Mesonet’ paper was implemented. However, the top of the keras layer didn’t really work as stated on the paper and it was becoming difficult to call the object in order to train the model. The implementations didn’t work as predicted. The model was able to predict whether input was real or fake. However, it wasn’t possible to train the model overall. Later on, a visit to Kaggle was committed to check the different methods of Mesonet that were implemented. One of the implementations was Mesonet using inception layers. Using this layer, a model was built. The code for the model was taken from Kaggle and then it was used in the model used for the deep fake detection challenge. The weight provided by the mesonet inception layer was taken and later the classification layer was taken for the built-in model. After this, the entire Mesonet layer was trained using a neural network architecture. But the training didn’t take place. This is because, it was taking a huge amount of execution time. 9 hours to be exact. It was very important to check the execution time in order to see the running time complexity. After this, steps from the Keras layers were reduced. Eventually, 20% of the datasets were used after this observation. And for each epoch it took 1 hour. There were 50 epochs. And the accuracy reached 88%. The most interesting part was that, as soon as the epochs started the accuracy started to increase. Therefore, the assumption is, even though the execution time, will increase at the end of the experiment, by increasing the dataset, it is possible that the accuracy will increase further than the current stage.

**Week 7 (01/03/2020-07/03/2020)**

Another experiment that was conducted was the implementation of the **Dense121** network. This network is also known as Dense121 network. 20% of the full dataset was extracted and used for training the model, with the Dense121 network. The accuracy that was conceived was not satisfactory. It was below 80%. Also, the densenet architecture consists of many parameters. As a result, it took some time to converge. Also, it was observed that the densenet architecture was biased towards the minority class, i.e. the real class. However, there was no conclusive results about how the biasness occurred.

Another extension of the project, is the plug-in for the web browser. The plug-in consisted of a few parts. One of them was to take an image, using the plug-in and save it in a designated location. The next step was to take the saved image, and pass it to tensorflow.js model. However, the tensorflow.js model was very complicated which required some fixation. Overall, there was no tensorflow.js model. Instead, there was keras.h5 model. This model was later then converted to a .json model. In order to change the model, at first, tensorflow.js was provided as an input, followed by tensorflow.tf. Then, the model was loaded. The model was loaded by disabling the eager execution, which caused several problems during the initial stages. Secondly, after the model was loaded, in a variable called “model”,then **tensorflow.js** was imported. From the tensorflow.js library, which is a python library (and not a javascript library), the model was executed with javascript. However, as it is a python library, it needed some changes. In google collaborator, the easiest way to use javascript is to use in-line command.

Finally, tensorflow.js was imported using the python kernel. After importing tensorflow.js, a converter was used from the library, and the converted has a function which helped to save the built-in model as the model which was provided as an input. The path was set for the model to be saved. Once the model was saved and executed, several outputs were found which turned out to be tensorfow.js models and along with a model of model.json. The latter model was used to predict and differentiate whether the input image was real or fake.

**Week 8 (08/03/2020-14/03/2020)**

It was important to check the problems that were present inside the TensorFlow library. The main problem that was occuring was the fact that during the implementation of Mesonet, the tensorflow was accidentally downgraded. For which, the old tensorflow cpu was installed and tensorflow gpu was not. For which, the model took more time for each epoch. After this, the tensorflow cpu was uninstalled and the tensorflow gpu was installed. The version of the library was 1.13 for tensorflow gpu.

The datas from face forensics, c40 level videos for the real class, was then extracted to images and was later merged with the existing facebook deepfake detection challenge dataset (only the real class). This is because there was a shortage of the adequate amount of data for the real class. Right now, there is approximately 2,50,000 data for the real class from 1,50,000 which is a substantial improvement. After the fame extraction, **Resnext** architecture was used for model training. The architecture was not compatible with keras for which the architecture was later replaced with **EfficientNet.** The accuracy achieved from this architecture is very disappointing, which is 67% to be very precise, for the first 10 epochs. However, there is a chance of the accuracy to improve as the epochs were taking only 35 minutes.

In conclusion, EfficientNet didn’t provide the expected accuracy.