REPORT

Part – II

Ravi Sankar Gogineni – R11788968

A method to automatically and efficiently detect face tampering in images particularly focuses on two recent techniques used to generate hyper-realistic forged/counterfeit videos: Deepfake and Face2Face. The popularity of smartphones and the growth of social networks have made digital images and videos common digital objects. The tremendous use of digital images has been followed by a rise of techniques to alter image content, using editing software. The field of digital image forensics research is dedicated to the detection of image forgeries in order to regulate the circulation of such falsified contents.

Deep learning performs very well in digital forensics and disrupts traditional signal-processing approaches. Addressing the problem of detecting these two video editing processes, Deepfake follows Face2Face. Up to today, there is no other method dedicated to the detection of the Deepfake video falsification technique.

• **Deepfake** is a technique that aims to replace the face of a targeted person with the face of someone else in a video. The core idea lies in the parallel training of two autoencoders.

• **Face2Face** Reenactment method is designed to transfer image facial expression from a source to a target person. Face2Face. The final image synthesis is rendered by overlaying the target face with a morphed facial blend shape to fit the source facial expression.

We propose to detect forged videos of faces by placing our method at a mesoscopic level of analysis. At a higher semantic level, the human eye struggles to distinguish forged images, especially when the image depicts a human face. That is why we propose to adopt an intermediate approach using a deep neural network with a

small number of layers. The two following architectures have achieved the best classification scores among all our tests, with a low level of representation and a surprisingly low number of parameters:

- **Meso-4**
- **MesoInception-4**

Deepfake is a method that seeks to replace the face of a targeted individual in a video with the face of someone else.

Following that, a tiny community expanded on this technology to produce a user-friendly program known as FakeApp. The parallel training of two autoencoders is the central concept. Depending on the output size, intended training duration, anticipated quality, and resource availability, their design might change. An encoder network and a decoder network are typically chained together to form an auto-encoder. By encoding the data from the input layer into a smaller number of variables, the encoder aims to execute a dimension reduction. The decoder's objective is to use those variables to produce an output that is close to the original input. In the optimization phase, the input and the produced approximation are compared, and the difference between the two is penalized. typically using a L 2 distance. In the case of the Deepfake technique, the original auto-encoder is fed with images of resolution $64 \times 64 \times 3 = 12,288$ variables, encodes those images on 1024 variables and then generates images with the same size as the input.

An auto-encoder EA is trained to rebuild the faces of A from the dataset of facial pictures of A, and an auto-encoder EB is trained to reconstruct the faces of B from the dataset of facial photographs of B, as part of the process to create Deepfake images. The secret is to combine the encoding weights of the two auto-encoders EA and EB while keeping the weights for each decoder distinct. After optimization, any picture with a face of A may be encoded using this common encoder and decoded using the EB decoder.

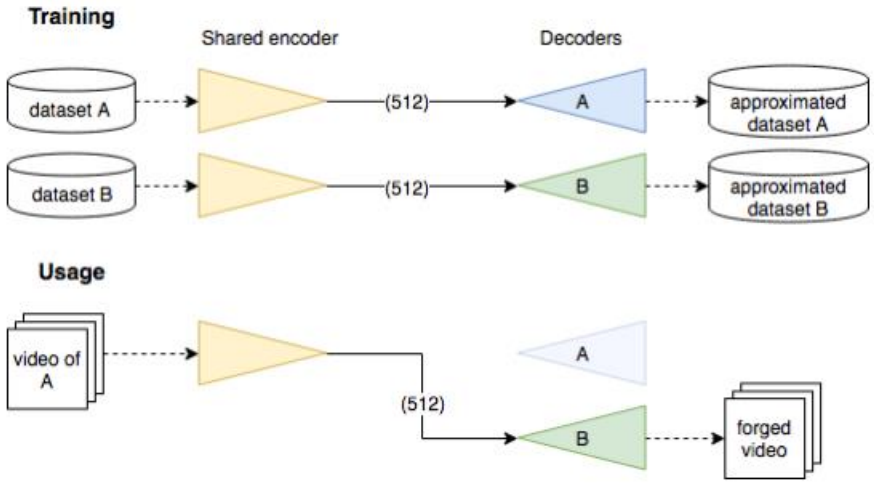


Figure 1. *Deepfake* principle. Top: the training parts with the shared encoder in yellow. Bottom: the usage part where images of **A** are decoded with the decoder of **B**.

The basic idea behind this method is to have a specialized encoder with the ability to encode broad information about the illumination, location, and expression of the face, and a decoder specifically designed to reconstruct the individual's face's consistent distinguishing features and details. This might so divide the morphological information from the contextual information on one side.

The approach is widely used since the findings are effective in practice. The target video must then be used to extract and align the target face from each frame, after which a new face with the same lighting and expression must be created using the updated auto-encoder and merged back into the original video.

More deeply, and this is true for other applications, autoencoders tend to poorly reconstruct fine details because of the compression of the input data on a limited encoding space, the result thus often appears a bit blurry. A larger encoding space does not work properly since while the fine details are certainly better approximated, on the other hand, the resulting face loses realism as it tends to resemble the input face, i.e. morphological data are passed to the decoder, which is a undesired effect.

However, thanks to the similar nature of the falsifications, identical network structures for both problems can yield good results. We propose to detect forged videos of faces by placing our method at a mesoscopic level of analysis. Indeed,

microscopic analyses based on image noise cannot be applied in a compressed video context where the image noise is strongly degraded. Similarly, at a higher semantic level, human eye struggles to distinguish forged images [21], especially when the image depicts a human face [1, 7]. That is why we propose to adopt an intermediate approach using a deep neural network with a small number of layers.

Meso-4 : This network begins with a sequence of four layers of successive convolutions and pooling, and is followed by a dense network with one hidden layer.

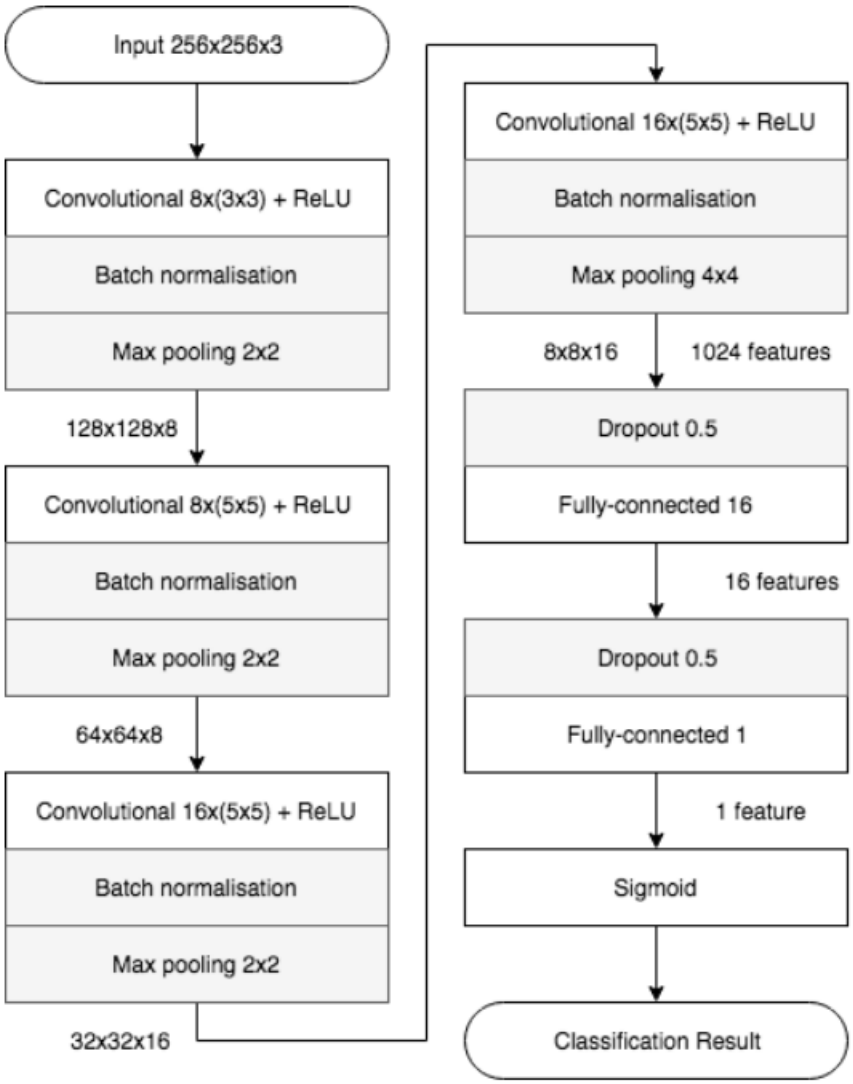


Figure 4. The network architecture of Meso-4. Layers and parameters are displayed in the boxes, output sizes next to the arrows.

MesoInception-4: An alternative structure consists in replacing the first two convolutional layers of Meso4 by a variant of the inception module introduced by Szegedy.

The idea of the module is to stack the output of several convolutional layers with different kernel shapes and thus increase the function space in which the model is optimized.

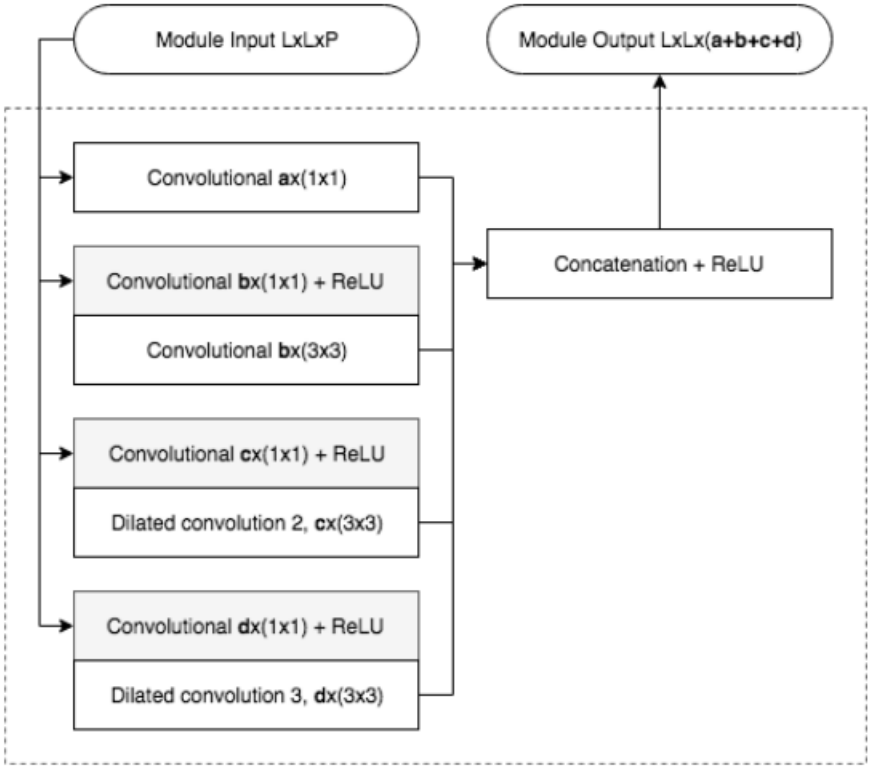


Figure 5. Architecture of the inception modules used in MesoInception-4. The module is parameterized using $a, b, c, d \in \mathbb{N}$. The dilated convolutions are computed without stride.

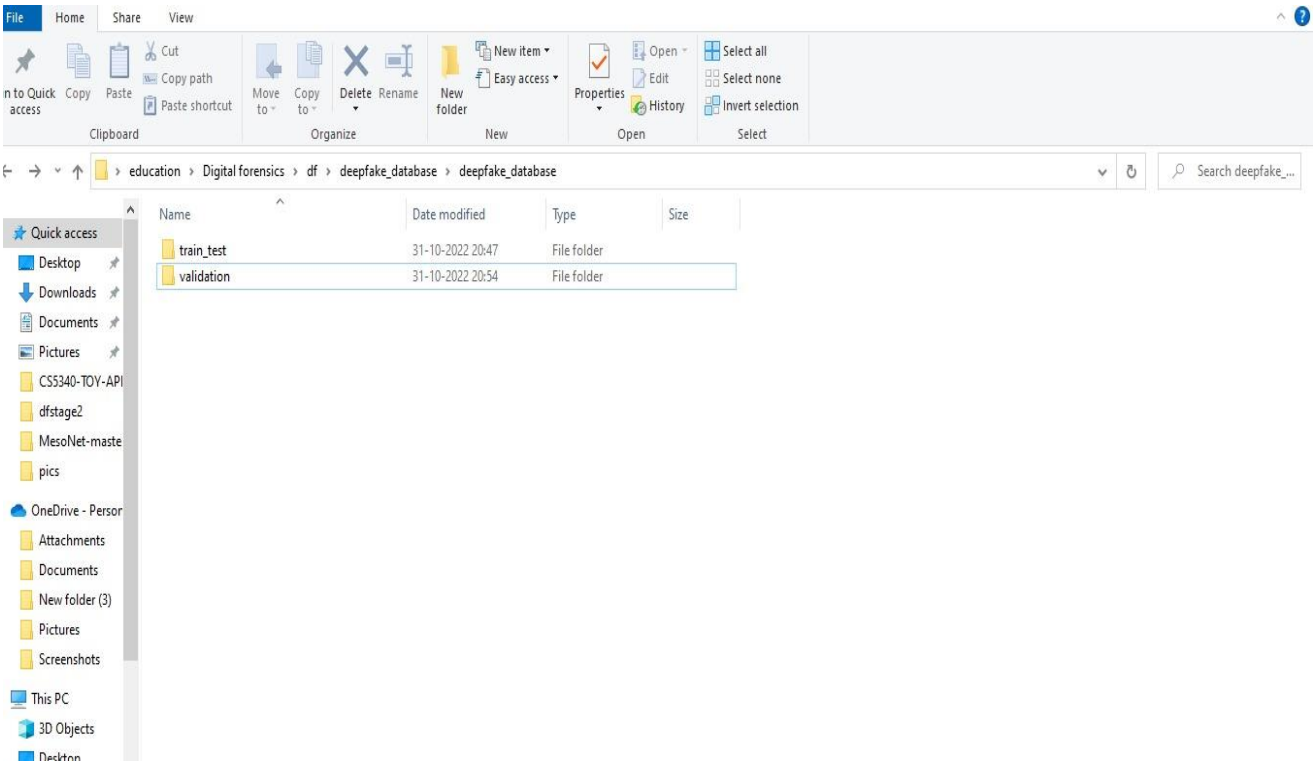
Replacing more than two layers with inception modules did not offer better results for the classification. The chosen parameters (ai, bi, ci, di) for the module at layer i can be found. With those hyper-parameters, this network has 28,615 trainable parameters overall.

Experiment:

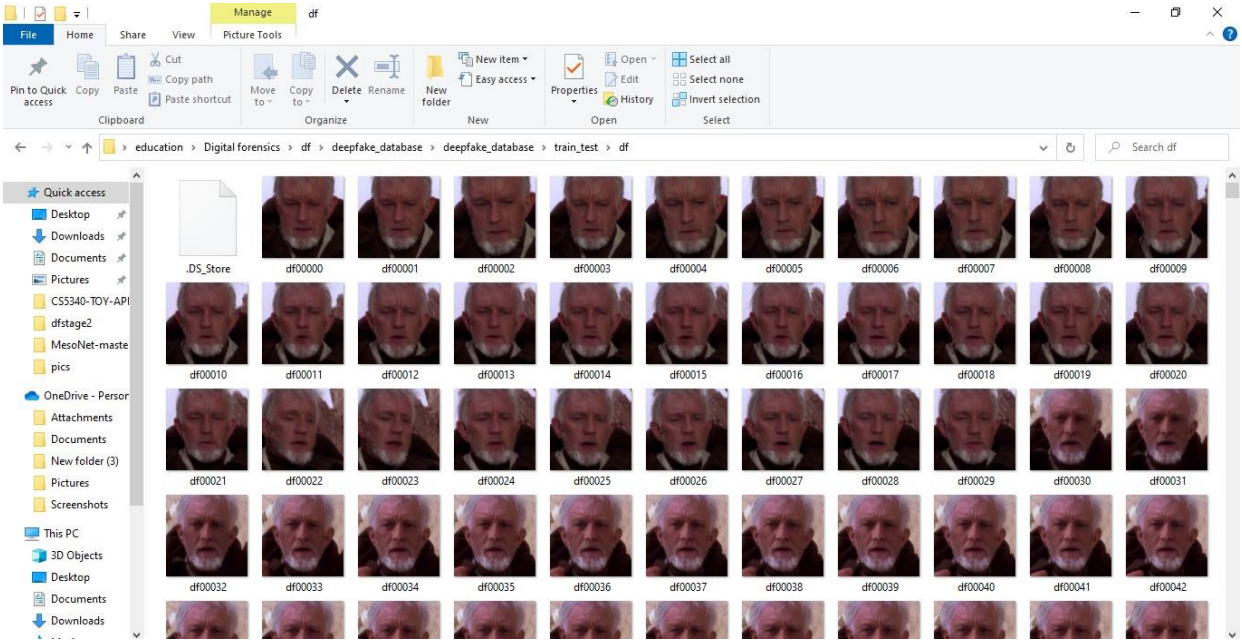
Dataset:

Downloaded the dataset : deepfake_dataset

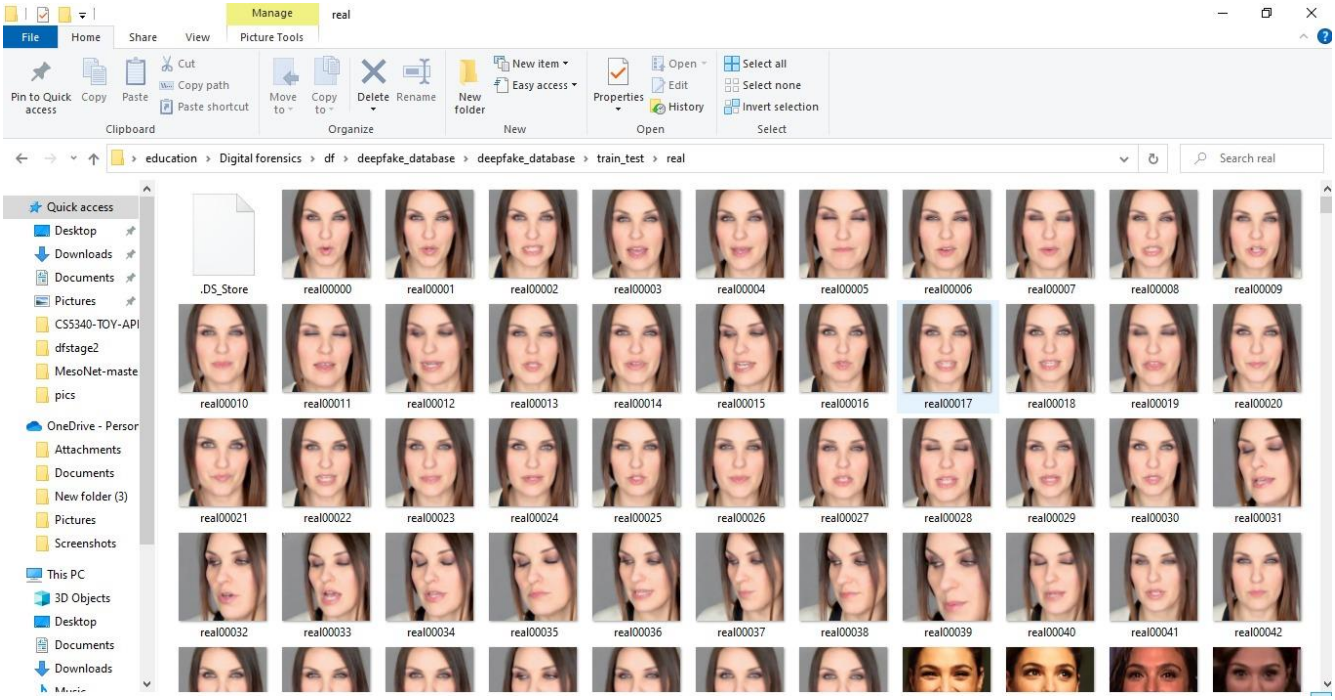
Consisting of both training dataset (12361 classes) and validation dataset (7148 classes) .



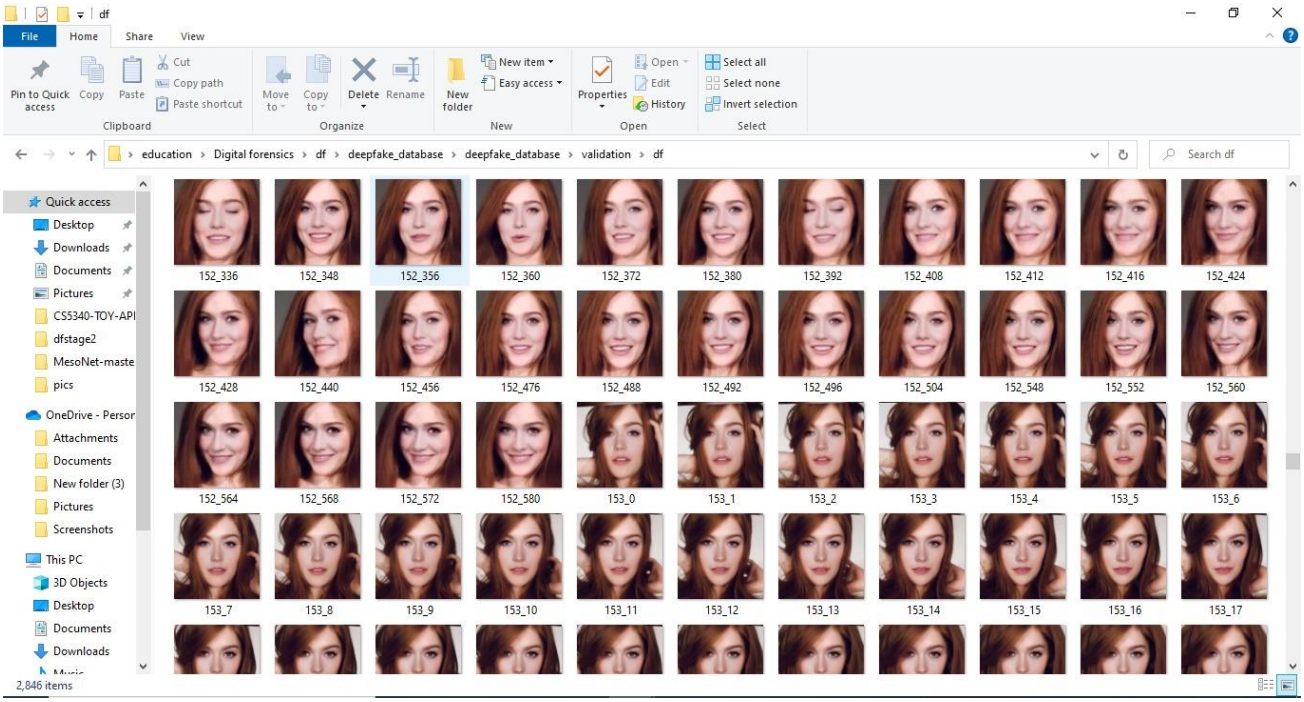
The training dataset consisted of 5111 deepfake (df) images.



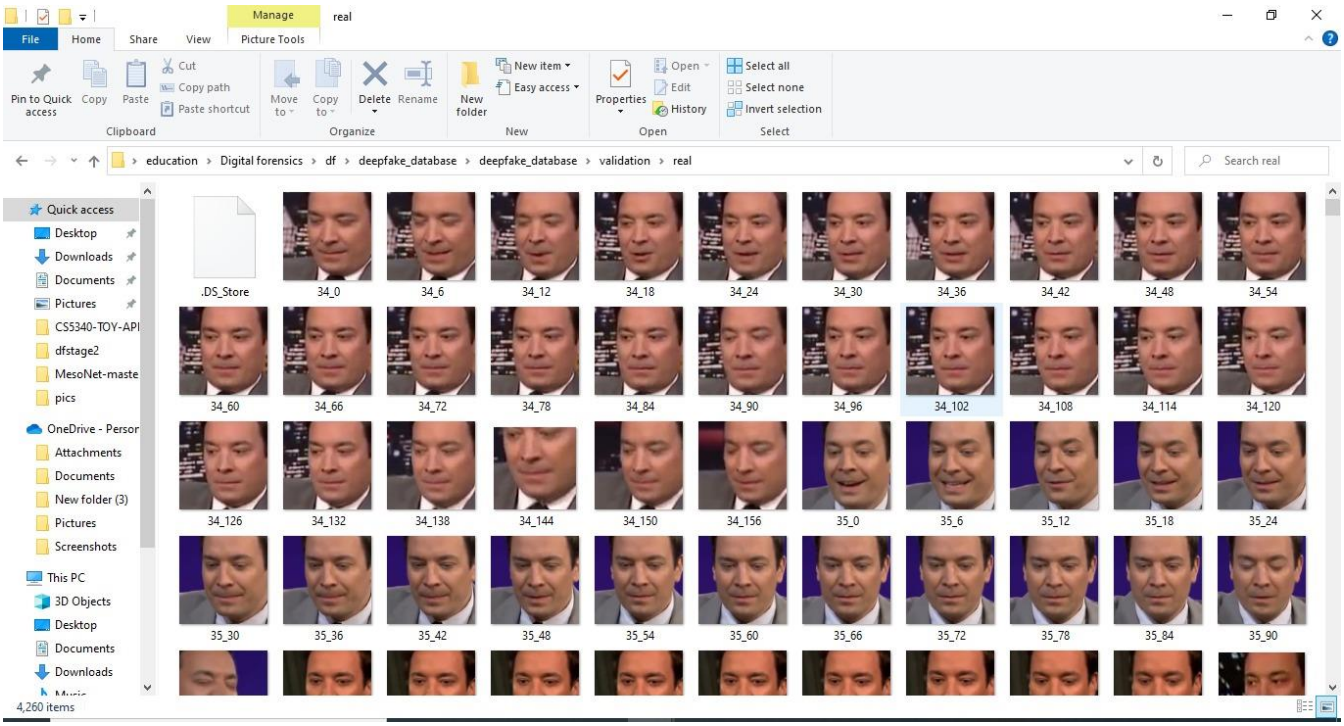
The training dataset consisted of 7250 real images.



The validation dataset consisted of 2889 deepfake (df) images.



The validation dataset consisted of 4259 real images.



The two recent approaches are used to generate hyper-realistic forged videos: deepfake and face2face. Traditional image forensics techniques are usually not well suited to videos due to their compression which strongly degrades the data. Thus, following a deep learning approach and building two networks, both with a low number of layers to focus on the mesoscopic properties of the image.

Set	**forged** class	**real** class
Deepfake training	5111	7250
Deepfake testing	2889	4259

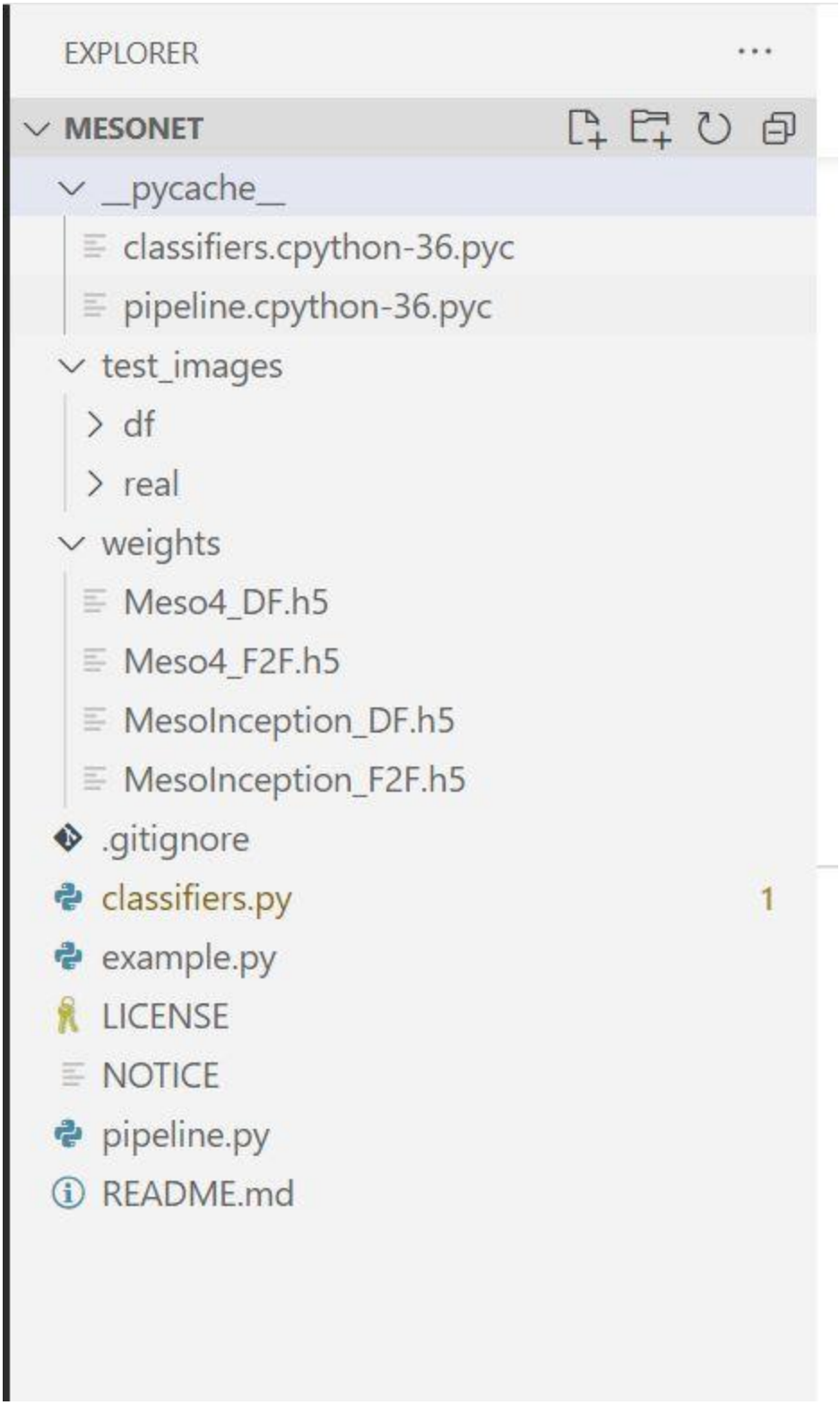
For this we need to initially install:

- Python

- Numpy

- Keras

The packages used: tensorflow, scipy, imageio and face_recognization.



Where Meso4_DF.h5, Meso4_F2F.h5, MesoInception_DF.h5 and MesoInception_F2F.h5 are the pre-trained models.

Classifiers.py

Under the class Classifiers, defining the functions _init_, predict, fit, get_accuracy, Load.

classifiers.py > ...

```
class Classifier:
    def __init__():
        self.model = 0

    def predict(self, x):
        if x.size == 0:
            return []
        return self.model.predict(x)

    def fit(self, x, y):
        return self.model.train_on_batch(x, y)

    def get_accuracy(self, x, y):
        return self.model.test_on_batch(x, y)

    def load(self, path):
        self.model.load_weights(path)


class Meso1(Classifier):
    """
    Feature extraction + Classification
    """
    def __init__(self, learning_rate = 0.001, dl_rate = 1):
        self.model = self.init_model(dl_rate)
        optimizer = Adam(lr = learning_rate)
        self.model.compile(optimizer = optimizer, loss = 'mean_squared_error', metrics = ['accuracy']

    def init_model(self, dl_rate):
        x = Input(shape = (IMGWIDTH, IMGWIDTH, 3))
```

Considering the Meso4 class we need to load the model and its pre-trained weights, With the following parameters.

```
dataGenerator = ImageDataGenerator(rescale=1./255)
generator = dataGenerator.flow_from_directory(
        r'C:\Users\MEGHANA\Downloads\MesoNet\test_images',
        target_size=(256, 256),
        batch_size=1,
        class_mode='binary',
        subset='training')
```

Consider pipeline.py

```
## Face extraction

class Image:
    def __init__(self, path):
        self.path = path
        self.container = imageio.get_reader(path, 'ffmpeg')
        self.length = self.container.count_frames()
        self.fps = self.container.get_meta_data()['fps']

    def init_head(self):
        self.container.set_image_index(0)

    def next_frame(self):
        self.container.get_next_data()

    def get(self, key):
```

FaceFinder()

```
class FaceFinder(Image):
    def __init__(self, path, load_first_face = True):
        super().__init__(path)
        self.faces = {}
        self.coordinates = {}  # stores the face (locations center, rotation, length)
        self.last_frame = self.get(0)
        self.frame_shape = self.last_frame.shape[:2]
        self.last_location = (0, 200, 200, 0)
        if (load_first_face):
            face_positions = face_recognition.face_locations(self.last_frame, number_of_times_to_
            if len(face_positions) > 0:
                self.last_location = face_positions[0]

    def load_coordinates(self, filename):
        np_coords = np.load(filename)
        self.coordinates = np_coords.item()

    def expand_location_zone(self, loc, margin = 0.2):
        ''' Adds a margin around a frame slice '''
        offset = round(margin * (loc[2] - loc[0]))
        y0 = max(loc[0] - offset, 0)
        x1 = min(loc[1] + offset, self.frame_shape[1])
        y1 = min(loc[2] + offset, self.frame_shape[0])
        x0 = max(loc[3] - offset, 0)
        return (y0, x1, y1, x0)
```

Running the training dataset the following are the outputs,

```
LIR Optimization Passes are enabled (registered 2)
Predicted : [[0.99770164]]
Real class : [1.]
PS C:\Users\MEGHANA\Downloads\MesoNet-master>
```

```
Found 4 images belonging to 2 classes.
with meso_df classifier:

2022-10-31 21:56:37.912504: I tensorflow/compiler/m
 Passes are enabled (registered 2)
Predicted : [[0.04869372]]
Real class :  [0.]

with meso_f2f classifier:

Predicted : [[0.9524388]]
Real class : [1.]
```

Currently we have tested with the dense layers = 2

And found 4 images belonging to 2 classes.

Contribution:

Ravi Sankar Gogineni:

Downloaded the dataset and installed the required pipelines and packages.

Sai Meghana Akula and Veera Venkata Siva Dasari:

Research on the various pseudo codes regarding the training and validation dataset and altering the parameters to obtain a good accuracy rate. Changing the weights and creating new models with a different number of dense layers, and various image classes.

Future Work:

By the end of November:

Evaluate the results on the tested dataset and verify the detection rate on both Deepfake and Face2Face. Currently, we have tested the dataset only for Deepfake but not Face2Face. In the next round, we will try to generate Face2Face forged images as well and run the pipeline to check the accuracy using both the Meso4 and MesoInception models. Create model pre-trained models with different parameters. If time permits will run the experiment with a different dataset with more images.