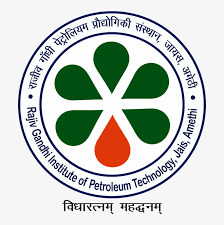


****

**BTP Report**

**on**

**Multimodal approach on deepfake detection**

**Submitted by:**

**Anubhav Ladha (20CS3009)**

**Hitarth (20CS3074)**

**Submitted to:**

**Dr. Pallabhi Saikia**

**Assistant Professor**

**Department of Computer Science and Engineering**

# Contents

| **Name of Topic** | **Page No** |
| --- | --- |
| **Abstract** | **3** |
| **Introduction** | **3** |
| **Approach** | **5** |
| **Tools and Technologies** | **6** |
| **Experiment and Result** | **8** |
| **My Contributions** | **19** |
| **Conclusion** | **19** |
| **Future Developments** | **20** |
| **References** | **20** |
|  |  |

# ABSTRACT

Recent years have witnessed the rapid development of Deepfake techniques which enable attackers to manipulate the facial area of an image and generate a forged image. As synthesized images are becoming more photorealistic, it is extremely difficult whether the given image or video has been manipulated even for human eyes. The above challenges have driven the development of Deepfake forensics using deep neural networks.

# INTRODUCTION

For the above challenges to handle we introduce M2TR , a Multi-modal Multi-Scale transformer for deep fake detection consisting of a Multi-Scale Transformer module and a Cross Modality Fusion module.

Let’s talk about the working of a M2TR working:-

M2TR first extracts features of an input image with a few convolutional layers. We then generate patches of different sizes from the feature map, which are used as inputs to different heads of the transformer. Similarities of spatial patching across different scales are calculated to capture the inconsistency among different regions at multiple scales. This benefits the discovery of forgery artifacts, since certain subtle forgery clues e.g., blurring and color inconsistency are often hidden in small local patches. The outputs from the multiscale transformer are further augmented with the frequency information to derive fused feature representation using a cross modality fusion module. Finally, the integrated features are used as inputs to several convolutional layers to generate prediction results. In addition to binary classification, we also predict the manipulated regions of the face image in a multi-task manner. The rationale behind is that binary classification tends to result in easily overfitted models. Therefore, we use face masks as additional supervisory signals to mitigate overfitting.

Behind all these approaches the availability of large-scale training data is an essential factor in the development of DeepFake detection methods. Some of the available datasets for the Deepfake Detection are UADFV dataset, the FaceForensics++ dataset (FF++), the Google DeepFake Detection dataset (DFD), the WildDeep- fake dataset and the Celeb-DF dataset. However, the quality of visual samples in the current Deepfake datasets is limited, containing clear artifacts like color mismatches, shape distortion and facial blurring. Therefore, there is still a huge gap between the images in the existing datasets and the forged images in the wild which are circulated on the Internet.

In this project I used a large scale and high quality Deepfake dataset, Swapping and Reenactment Deepfake(SR-DF) dataset, which is generated using the state-of-the-art face swapping and facial enactment methods for the development and evaluation of Deepfake Detection methods.

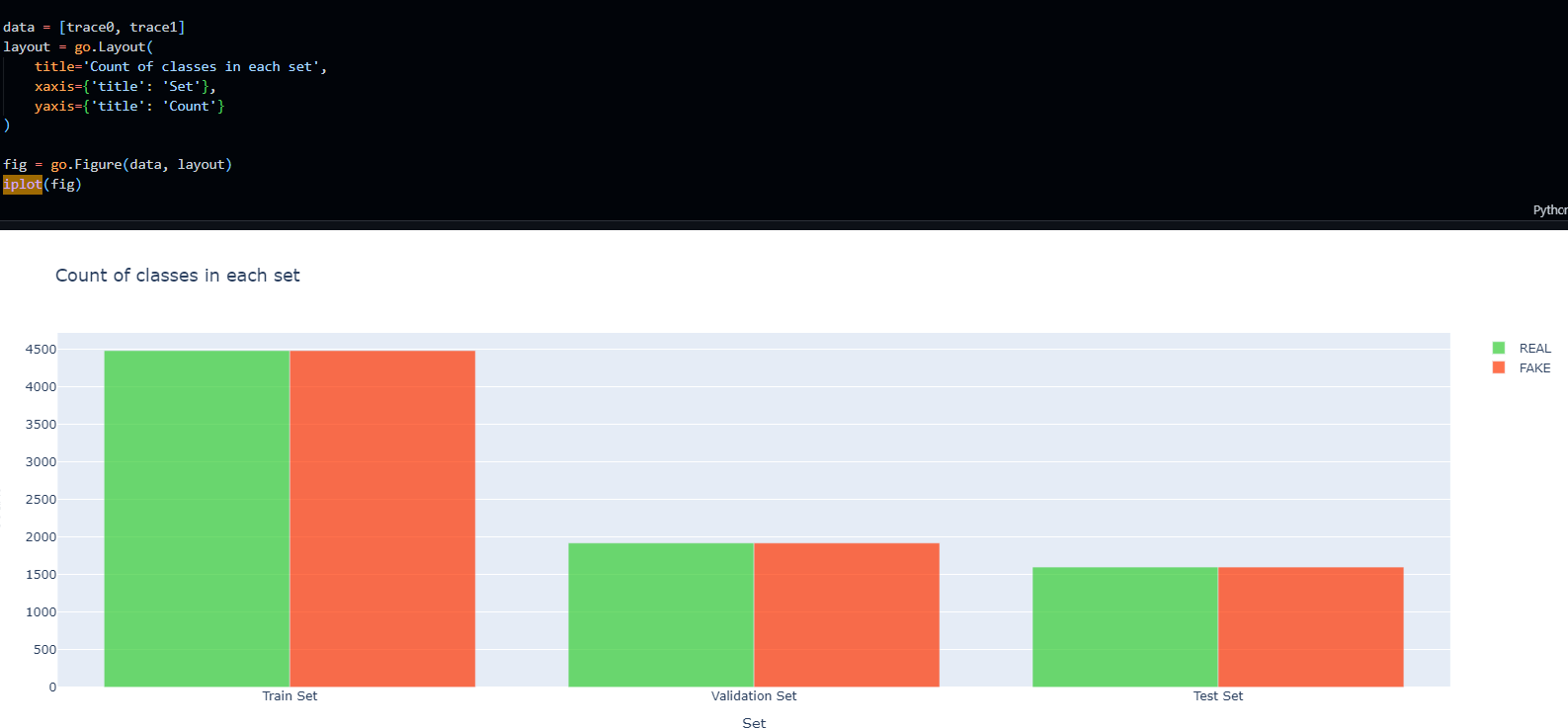
# APPROACH

Our goal is to detect the subtle forgery artifacts that are hidden in the inconsistency of local patches and improve the robustness of image compression with frequency features. We wish to locate regions that contain manipulation artifacts and thus are inconsistent with their other regions in the image. This requires modeling long-range relationships in images, i.e., calculating the similarity of regions not only in a local neighborhood but also lie far apart. Inspired by the great success of transformer models in capturing long-term context information, we use transformers for Deepfake detection. Unlike recent approaches that directly split an input image into multiple patches of the same size as inputs to transformers , we introduce a multi-scale transformer, which generates patches of different scales. The intuition behind is to cover regions with different sizes to identify artifacts generated by manipulation methods. It has been shown that artifacts in manipulated images and videos are no longer perceptible with compression approaches like JPEG compression. We also compute features from the frequency domain to complement RGB features. The resulting frequency features are combined with RGB features with a cross modality fusion module.

# TOOLS AND TECHNOLOGIES

We have used basically the following libraries for the following purposes:-

1. Tensorflow : This is the base for every library. My entire model is based on that library. like whether my gpu is available or not or when we create our CNN architecture and when we create our Xception model for fine tuning.
2. Keras: It is used for various purposes. The Keras Application module provides a number of the state of the art models pre-trained on the ImageNet-1k dataset.We will be using InceptionV3 model for this purpose with the help of keras library.
3. pandas: for importing and reading the dataset. It has features which are used for exploring, cleaning,transforming and visualising from data.
4. numpy:NumPy stands for ‘Numerical Python’. It is an open-source Python library used to perform various mathematical and scientific tasks. It contains multi-dimensional arrays and matrices, along with many high-level mathematical functions that operate on these arrays and matrices.
5. plotly: The plotly Python library is an interactive, open-source plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3-dimensional use-cases.Built on top of the Plotly JavaScript library (plotly.js), plotly enables Python users to create beautiful interactive web-based visualizations that can be displayed in Jupyter notebooks, saved to standalone HTML files, or served as part of pure Python-built web applications using Dash. The plotly Python library is sometimes referred to as "plotly.py" to differentiate it from the JavaScript library. I will show you some parts where I have used plotly for creating better visualization.



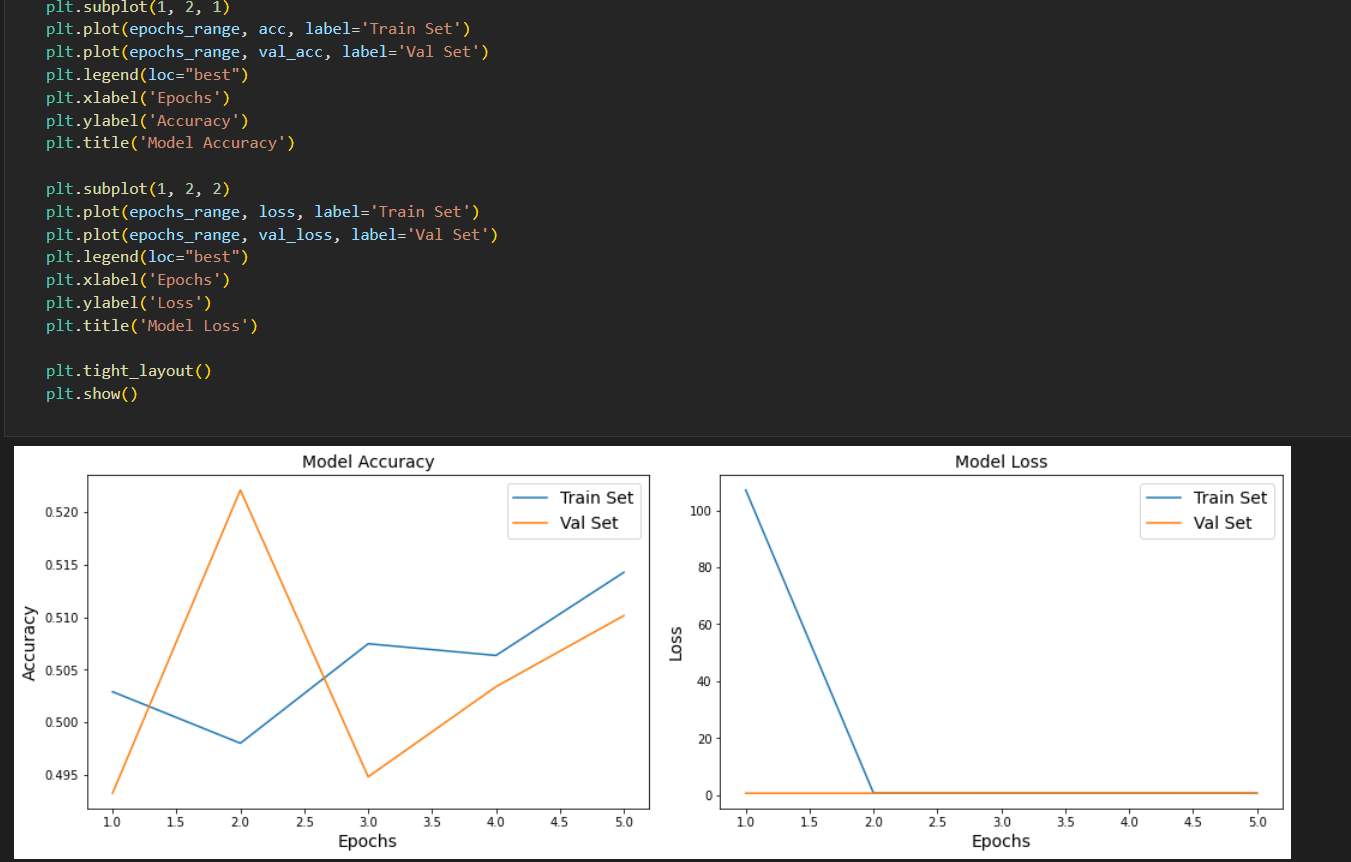
1. matplotlib: Matplotlib is one of the most popular and oldest plotting libraries in Python which is used in Machine Learning. In Machine learning, it helps to understand the huge amount of data through different visualisations.

# EXPERIMENT AND RESULT

We first apply the Discrete Cosine Transform (DCT) to transform the input image X from the RGB domain to the frequency domain and obtain DCT (X) ∈ R 𝐻×𝑊 ×1 . Benefiting from the properties of DCT, low-frequency responses are placed in the top-left corner of DCT (X), while high-frequency responses are in the bottom-right corner. we separate the frequency domain into low, middle, and high frequency bands with three hand-crafted binary base filters {t 𝑖 𝑏𝑎𝑠𝑒 } 3 𝑖=1 and obtain the decomposed frequency components: 𝒅𝑖 = DCT (X) ⊙ 𝒕 𝑖 𝑏𝑎𝑠𝑒,𝑖 = {1, 2, 3}, (3) where ⊙ denotes the element-wise dot-product. Empirically, we manually design the base filters with the following pattern: the low frequency band 𝒕 1 𝑏𝑎𝑠𝑒 is the first 1/16 of the entire spectrum, the middle frequency band 𝒕 2 𝑏𝑎𝑠𝑒 is between 1/16 and 1/8 of the spectrum, and the high frequency band 𝒕 3 𝑏𝑎𝑠𝑒 is the last 7/8 of the spectrum. To preserve the shift invariance and local consistency of natural images and explore the representative capability of CNNs, we then invert di back into the RGB domain via IDCT: 𝒃𝑖 = DCT−1 (𝒅𝑖),𝑖 = {1, 2, 3}. Finally, we re-assemble {𝒃𝑖 } 3 𝑖=1 along the channel axis to obtain the frequency-aware spatial map B ∈ R 𝐻×𝑊 ×3, and input it to several stacked convolution layers to extract frequency features 𝒇𝑓 𝑞, the size of which is the same as 𝒇𝑠 .The feature maps 𝒇𝑐𝑚?? are then passed through several layers, followed by a single-scale Transformer (patch size equal to 2 × 2) to obtain global semantic features, which are finally used to predict whether the input image is real or fake using a cross-entropy loss

L𝑐𝑙𝑠 : L𝑐𝑙𝑠 = 𝑦𝑙𝑜𝑔𝑦ˆ + (1 − 𝑦)𝑙𝑜𝑔(1 − 𝑦ˆ),

where 𝑦 is set to 1 if the face image has been manipulated, otherwise it is set to 0; 𝑦ˆ denotes the predicted label by our network. There are other segmentation losses as well as contrastive losses which also need to be computed. Before jumping to using a pretrained model let’s develop some baseline model to test how our pretrained model outperforms. This will be done using custom CNN and RNN architecture.

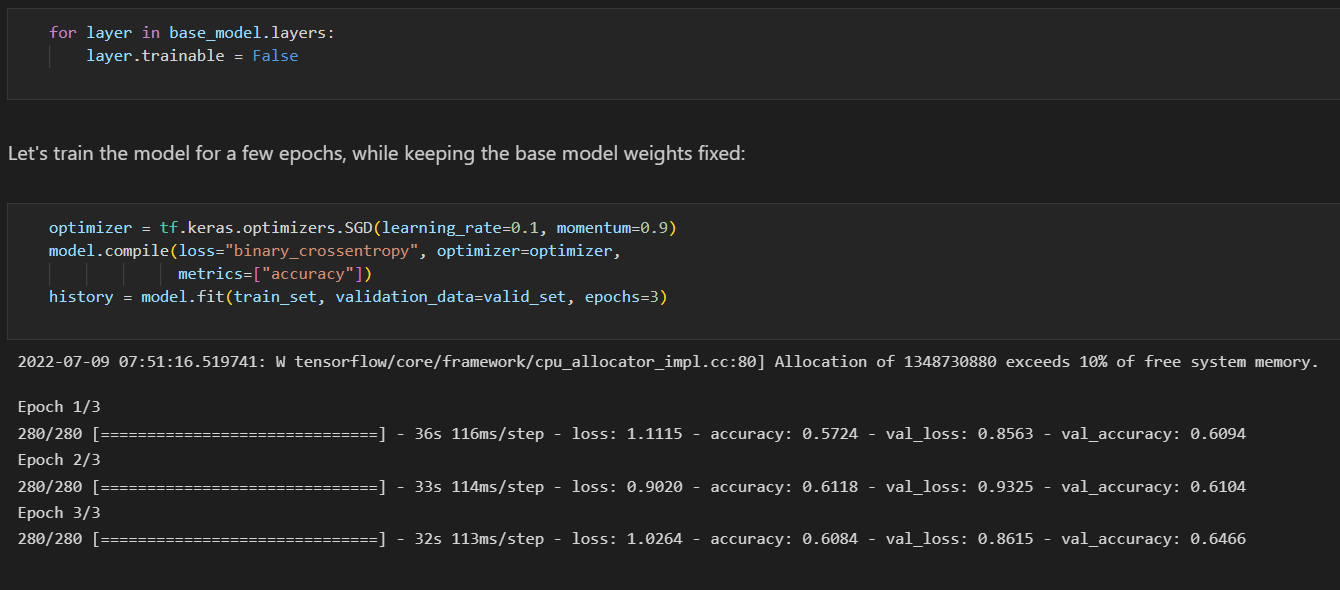


This figure shows the model’s accuracy. A baseline score of 50.06% is good to go, let's finetune some pretrained model. Here I used the Xception model for fine tuning. All three datasets contain individual images. We need to batch them, but for this we first need to ensure they all have the same size or else batching will not be done . We can use a resizing layer for this . We must also call this tf.kears.applications.xception.preprocess\_input() function to preprocess the images appropriately for the Xception model. We also add shuffling and prefetching to the training datasets. Let’s look at the first 9 images from the validation set: they’re all values ranging from -1 to 1 :

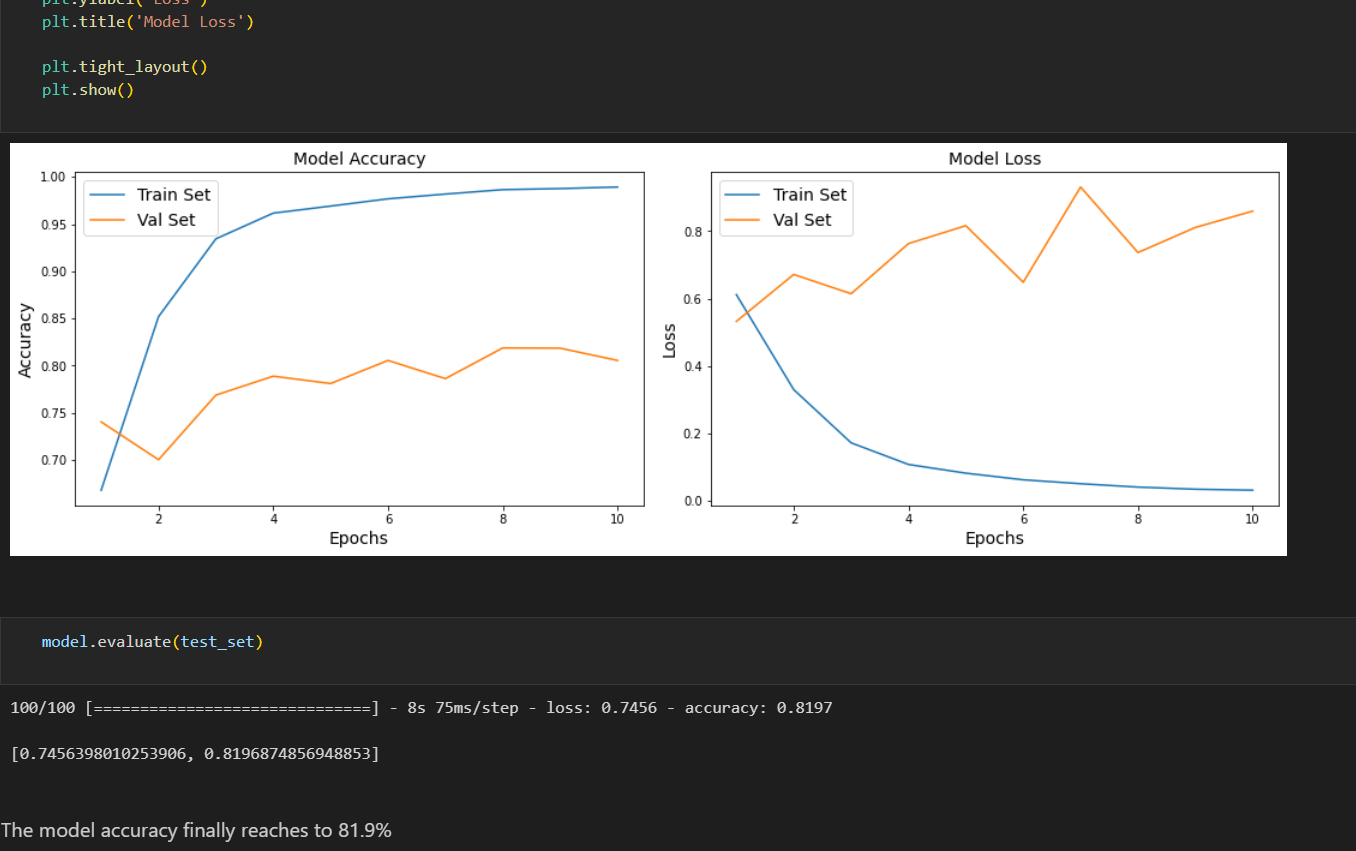


Now with the finetuning of the top layers of the Xception model the model performance jumps to 63.8%. Now that the weights of the top layers are not too bad, we can make the top part of the base model trainable again and

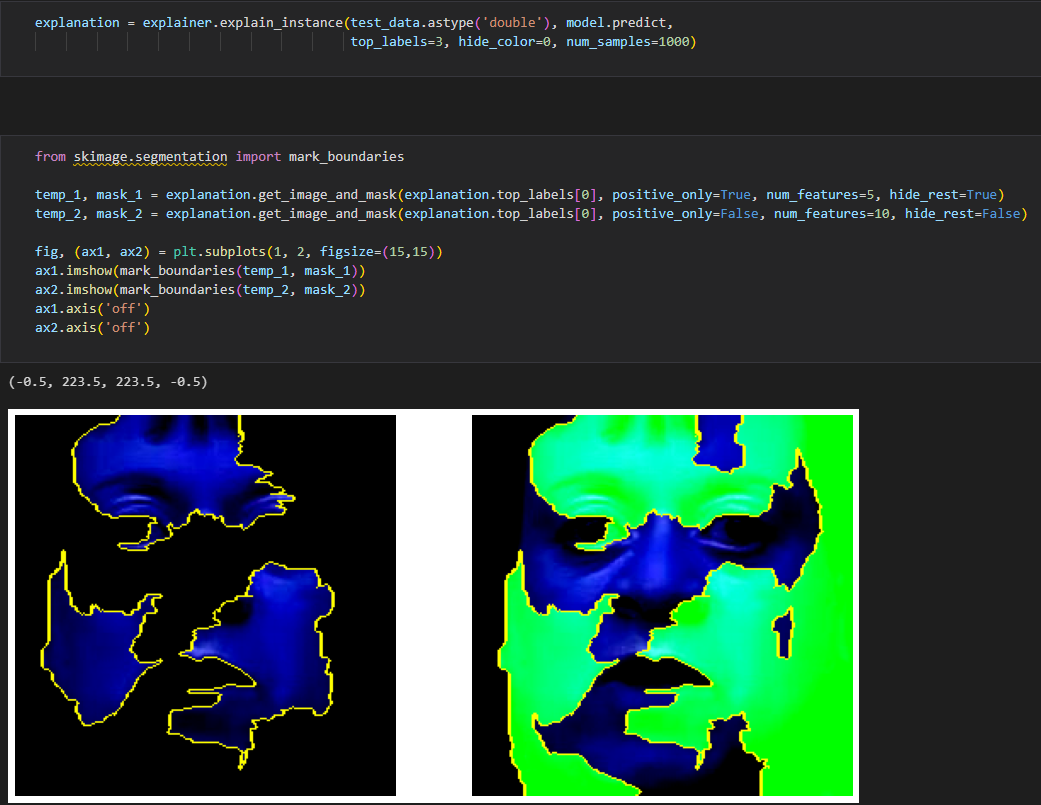
continue training but with a lower learning rate.



After doing this work again and again the model accuracy finally reaches 81.9%.

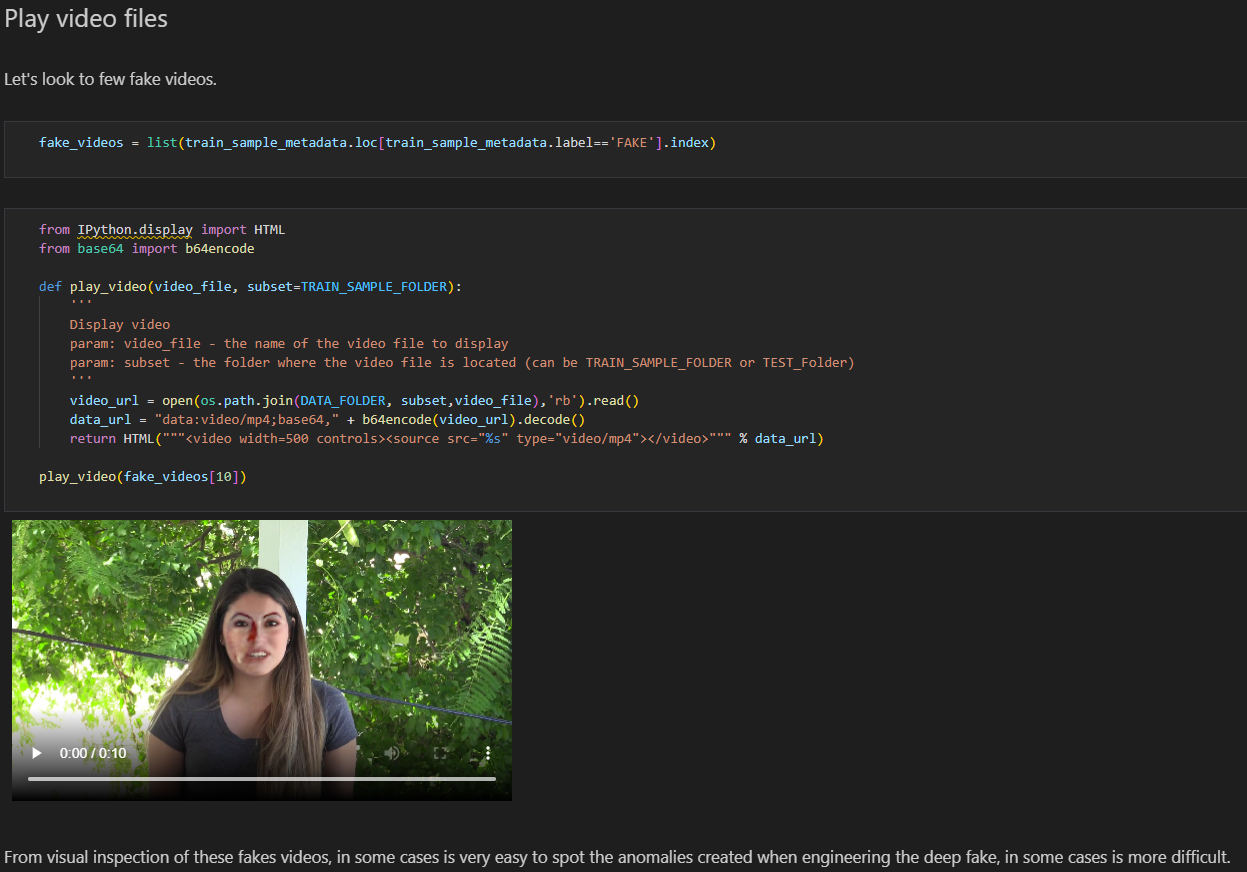


Let’s try to interpret the trained model on how it finds a fake image. For that purpose, we will use the lime library and then we import lime\_image from that . I will show in the code how we can interpret the following part .



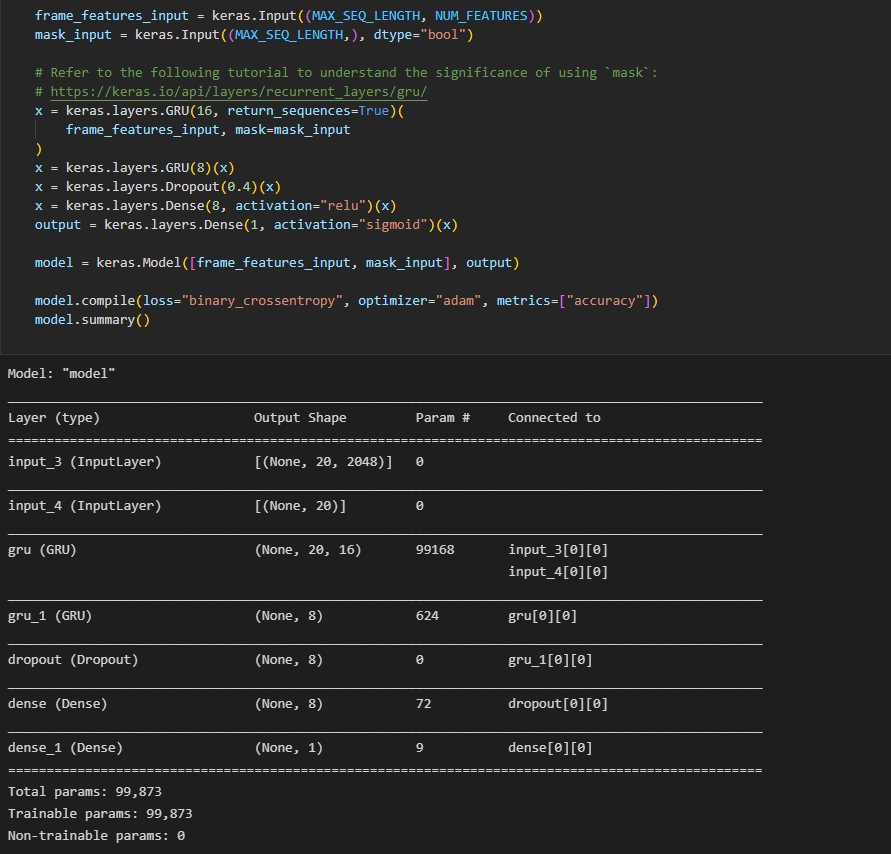
Now, let’s try to interpret the trained model on how it finds a fake video. For that purpose, we will use the IPython.display library and then we import b64encode

from that . I will show in the code how we can interpret the following part .



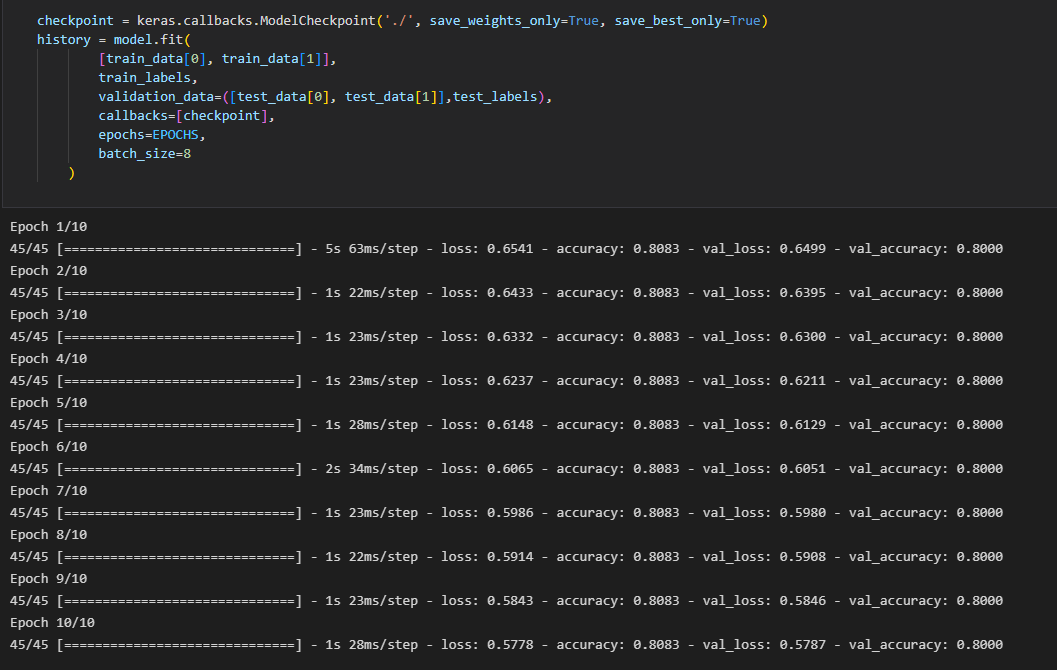
Now, we model a CNN-RNN architecture in which we :

1. Capture the frames of a video.
2. Extract the frames from the videos until a maximum frame count is reached.
3. In the case where a video’s frame count is lesser than the maximum frame count, we will pad the video with zeroes.

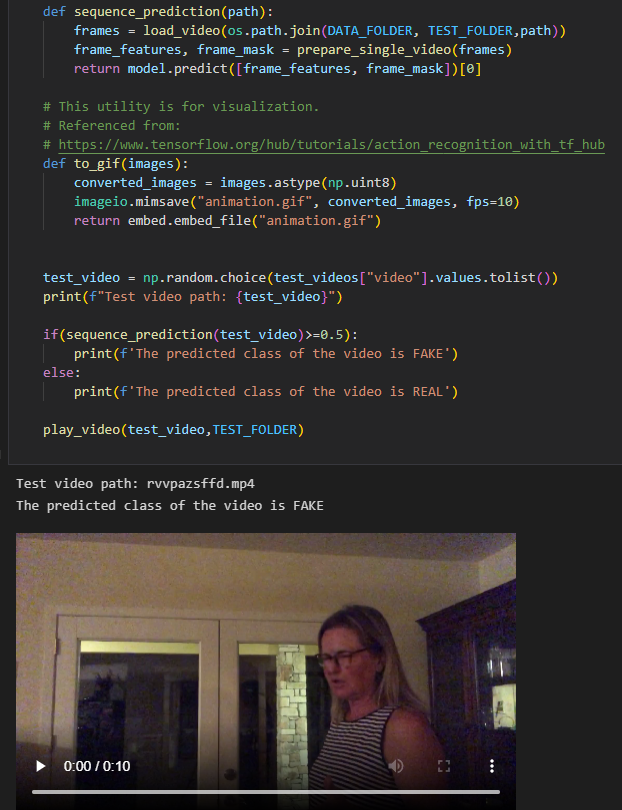


We can use a pre-trained network to extract meaningful features from the extracted frames. The Keras Applications module provides a number of state-of-the-art models pre-trained on the imageNet-1k dataset. We will be using the InceptionV3 model for this purpose.

Now, we can feed this data to a sequence model consisting of recurrent layers like GRUs.



Final Result for the fake video:



# MY CONTRIBUTIONS

I have created a baseline model to test how our pretrained model outperforms. This will result in 50.06% of baseline score which is good to go. Let's fine tune some pretrained models. Here I used an Xception model for fine tuning. All the three datasets contain individual images as I have used the dataset from Kaggle, so we need to batch them but for this we need to ensure they all have the same size or else batching will not work. We can use a Resizing layer for this. We also add shuffling and prefetching to the training dataset. For the Deepfake video classification we have used a CNN-RNN architecture. For this we will do the following operations capturing the video frame,extracting the frames from the video until the maximum frame count is reached. If the frame count is lesser than the maximum count then I will pad it with zeros. We will be using the InceptionV3 module for that purpose. Finally, we can put all the pieces together to create our data processing utility. Now we can feed this data to a sequence model consisting of recurrent layers like GRU.

# CONCLUSIONS

The deep fake image classifier and deep fake video classifier model gave a generalisation error of around 80%.

# FUTURE DEVELOPMENTS

In the future we will try to improve the result using more advanced transformers like LSTM or GPT-3 and along with that I will try to get video as well as text also for the Deepfake detection .

# REFERENCES

• Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Qureshi, and Mehran Ebrahimi. 2019. EdgeConnect: Structure

Guided Image Inpainting using Edge Prediction. In ICCVW

• Huy H Nguyen, Fuming Fang, Junichi Yamagishi, and Isao Echizen. 2019. Multitask learning for detecting and segmenting manipulated facial images and videos. In BTAS.

• Huy H Nguyen, Junichi Yamagishi, and Isao Echizen. 2019. Use of a capsule network to detect fake images and videos. arXiv preprint arXiv:1910.12467 (2019).

• Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. 2019.

Faceforensics++: Learning to detect manipulated facial images. In ICCV

• Yuezun Li and Siwei Lyu. 2018. Exposing deepfake videos by detecting face warping artifacts. arXiv preprint arXiv:1811.00656 (2018).

• Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. 2020. Celeb-df: A large-scale challenging dataset for deepfake forensics. In CVPR.

• Deressa Wodajo and Solomon Atnafu. 2021. Deepfake Video Detection Using Convolutional Vision Transformer. arXiv preprint arXiv:2102.11126 (2021)

• Xin Yang, Yuezun Li, and Siwei Lyu. 2019. Exposing deep fakes using inconsistent head poses. In ICASSP.

• Yang Yang and Xiaojie Guo. 2020. Generative Landmark Guided Face Inpainting. In PRCV

• Pavel Korshunov and Sébastien Marcel. 2018. Deepfakes: a new threat to face recognition? assessment and detection. arXiv preprint arXiv:1812.08685 (2018).

• Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies,

and Matthias Nießner. 2019. Faceforensics++: Learning to detect manipulated facial images. In ICCV