Use of transfer learning and fine-tuning on pre-trained deep learning networks to build a deepfake detector

Program:	MSc Cyber Security
Cohort:	2021/2022
Module Code:	UFCF9Y-60-M
Module Name:	CSCT Masters Project
Student Name:	Lo Lok Yi, Carol
Student ID:	20046885
Date:	11 December 2021
Word count:	12,566 words, including tables and figures (Note: exclude abstract, references and appendices)

Executive Summary

Abstract

Deepfakes are synthetic content which uses ‘deep learning’ to create ‘fake’ images, videos, audios and texts that appears authentic. Deepfakes has gone viral globally after several incredibly realistic videos had been released on the Internet. It is an artificial intelligence and machine learning technique for image creation.

The purpose or aim of this report is to detect deepfake videos by creating deep learning models using transfer learning and fine-tuning techniques. Creating practical tools to detect deepfake videos is paramount, as deepfake technology could be misused for unlawful purposes, leading to misinformation, social injustice, and national security issues.

The task of distinguishing deepfake and real videos is an image classification task. First, an image database containing around 18,000 images was created. Then, ten binary classifiers of different network architectures are built, trained, validated and tested in Google Colab using customised datasets and videos in the wild.

Structure of The Report

This report has four parts.

- The first part introduces deepfake technology, the purpose, scope of this project, cyber security threats and risks, type and techniques behind face synthesis, and current trends of deepfake.
- The second part of this report is a literature review on deepfake detection approaches. It also briefly discusses existing detectors with relatively high performance and problems of these detectors.
- The third part gives the details of the artefacts for the experiment, including the creation of a dataset for model training, validation and testing, data preprocessing and model development.
- The fourth part is the results and analysis of the experiment. Model testing was performed using test datasets and videos captured in the wild.
- The last part is the conclusion of the work performed.

Table of Contents

Chapter 1: Deepfake Technology

1.1 Introduction of Deepfake Technology

Deepfakes are synthetic content which uses ‘deep learning’ to create ‘fake’ images, videos, audios and texts that appears authentic. It gains people’s attention as the technology could create realistic-looking photos and videos of celebrities saying and doing things that they did not say or do, including replicating or altering facial expressions, mannerisms, voice tone and contents. Today, realistic deepfakes can be produced by one source image and a video, as shown in the first-order model developed by Siarohin et al. in 2020 (Figure 1).

Figure 1 This image illustrates the work of the first-order model (Siarohin et al., 2019)

Toews (2020) mentioned that deepfakes has gone viral globally after several incredibly realistic videos had been released on the Internet. For instance, an actor transferred his facial movements and spoke like the former U.S. President Obama using deepfake technology; an artist altered the opinions of Mark Zuckerberg and made the video appear to be official news reporting; an actor acted as Tom Cruise on videos in social media, TikTok.

Deepfakes look realistic. A deep learning algorithm named StyleGAN generated the artefacts in Figure 2. Computer scientists use sophisticated statistical algorithms to enable computer models to learn and extract the patterns from sample data, such as images, video or audio clips. While some algorithms behind deepfakes aim to trick a human into believing what they hear, see and read is authentic.

Figure 2 This image illustrates factitious human faces generated by StyleGAN (Karras et al., 2019)

1.2 Purpose and Objectives of The Project

The purpose or aim of this report is to detect synthetic videos that are created by deepfake technology using transfer learning and fine-tuning techniques during model training. Creating practical tools to detect deepfake videos is paramount, as deepfake technology could be misused for unlawful purposes, leading to misinformation, social injustice, and national security issues.

A successful deepfake detector must achieve an overall high-performance score as a binary image classifier. This project has the following objectives:

1. Create detectors by applying transfer learning and fine-tuning techniques for feature extraction on different pre-trained models;
2. Assess the effectiveness of the detectors by consistently comparing the same sets of performance metrics, including accuracy, precision, recall and F1 score; and
3. Evaluate the characteristics of misclassified images, i.e. reals classified as fakes or fakes classified as reals.

Generalisation is also an essential element of a successful detector. Thus, after reviewing the test datasets, further testing is conducted by using videos captured in the wild to confirm its ability to distinguish between deepfake and real videos. Then, if there is a satisfactory model (≥90% accuracy in the binary classification), it would be deployed as a web service.

1.3 Project Scope

There are different synthetic methods. Some methods create entirely new personas and some target existing persons. Concerning the risk of impersonation on existing persons, this project mainly focuses on detecting deepfake videos or images that applied face swap or identity swap, i.e. replacing one person's face with another person. This project created an image dataset of real faces and synthetic faces extracted from three deepfake detection databases for model training, validation, and testing.

In the recent deepfake research, the following sub-topics are as popular as deepfake detection. However, it is essential to note that these areas are not the main focus or are out-of-scope in this project:

- deepfake prevention mechanism
- deepfake creation process
- legislation of deepfake technology
- adversarial threats to deepfake detection
- creation and detection of face morphing
- creation and detection of voice cloning
- creation and detection of shallow-fakes or cheap fakes, i.e. manipulation without using deep learning

1.4 Cyber Security Threats and Risks on Deepfake Technology

The technology for deepfake creation, such as DeepFaceLab and Faceswap, are open source applications. This technology is also implemented in mobile applications such as the Zao app and REFACE app. This readily available technology attracted many people to use them for entertainment or art creation. For instance, use one picture to swap the face of the artists or singers in a movie clip or music video.

However, deepfake technology is a troublemaker rather than a problem solver. Some of the widespread unethical use of the deepfake technology include modifying political campaign videos to make the politicians express opinions differently, creating fake social media profiles and highly believable messages to support political campaigns, modifying pornographic videos with women not involved in, and undressing the clothes of women in pictures, reported by Wakefield in 2021.

From the perspective of cyber security, in the Private Industry Notification issued in March 2021, the Federal Bureau Of Investigation (F.B.I.) anticipated that foreign and criminal cyber actors would increasingly use synthetic content for spear phishing, social engineering and tradecraft. F.B.I. also estimated that a new form of cyberattack vector, Business Identity Compromise (B.I.C.), would be emerged. B.I.C. uses compromised corporate email accounts and content management software to develop synthetic corporate personas or emulate existing employees.

Moreover, Trend Micro Research (2020) and Europol's European Cybercrime Centre also anticipated that the malicious use or abuse of deepfake technology could bring security threats to society, organisations and individuals when people abuse the tools. Table A summarised the possible malicious and criminal activities that are likely to happen soon:

Table A: Example of Malicious and Criminal Activities Anticipated by Trend Micro

Examples of Misuse	**Impact Level**
• Falsifying or manipulating electronic evidence for criminal justice investigations • Disrupting financial markets by deepfake senior company official • Deepfake high-profile public officials and distributing disinformation and manipulating public opinion • Stoking social unrest and political polarisation, which create mass panic and confusion	Society
• Falsifying online identities and fooling KYC mechanisms • Perpetrating extortion and fraud • Facilitating document fraud	Organisations
• Destroying the image and credibility of an individual • Harassing or humiliating individuals online	Individuals

Up to the date of this report, there are a few notable criminal cases that involved the weaponisation of deepfake technology. For example, as shown in Table B, victims were weaponised by voice cloning technology.

Table B: Notable Criminal Cases

Time	Type	Case Description
03/2019	Voice-spoofing attack	The criminal mimicked the voice and accent of a German C.E.O. in a U.K.-based energy firm. The victim company transferred €220,000 to a fictitious Hungarian supplier after several phone calls with the fake C.E.O. (Stupp, 2019)
01/2020	Voice-spoofing attack	The criminal, who cloned the voice of a company director, spoofed the company's email account and falsified documents related to a fraudulent merger acquisition. The bank manager in Hong Kong, receiving the emails and phone requests, believed it was a legitimate request from authorised persons. The bank manager arranged several bank transfers as per instructions amounting to $35 million. (Brewster, 2021)

As reported by Wiggers (2020) and Palmai (2021), cyber security experts believed a considerable security risk on voice cloning, a subcategory of deepfakes. Our voice is an acoustic fingerprint and a marker of personality. People are less aware that they could be tricked to believe someone else is talking. Talking on the phone with someone who trusts and knows well is one of the most common ways to ensure that someone is familiar. For instance, if an employee recognises the voice of his/ her boss on the phone, the employee would indeed immediately respond and carry out what he/ she was asked to do.

In addition to synthetic voice, another threat of deepfake anticipated by Trend Micro is face morphing attacks on biometric recognition systems. For instance, if the application enrolment process requires submitting a printed photo instead of physical enrolment in person, the criminal could submit a fraudulent renewal application for an I.D. using a morphed photo of individuals who look similar. As the fake photo is small, it would be difficult for bare eyes to detect if it is a morphed photo. If the fake photo is not detected, falsified documents would be created and used as false breeder documents. This type of fraud could undermine the passport issuance/verification process and border security controls, threatening national security.

1.5 Face Synthesis

Deepfakes can be created by different deep learning methods, as shown in Table C. The first two columns of the table showed the synthesis methods and related deepfakes creation techniques (Tolosana et al., Mirkey and Lee). The last column of the table analyses how this type of synthesis might be misused by criminal activities and cause significant cyber security threats.

Table C: Facial synthesis

Synthesis Methods	**Deep Learning Techniques**	**Significance of Cyber Security Threats**
Face synthesis – create faces that do not exist, usually applies to images	GAN (StyleGAN, DiscoFaceGAN)	Creation of fake profile for disseminating misinformation
Identity swap/face swap – replace the face of one person with another person, usually applies on videos	AE, AEI-Net, HEAR-Net (DeepFake FaceSwap, Z.A.O., DeepFaceLab, VGGFace, FaceShifter, FSGAN	Creation of fraudulent renewal application with a morphed video and photo of individuals who look similar
Expression swap/face re-enactment – transfer the facial expression of one person to another person while maintaining the identity of the target person, usually applies to videos	GAN (Face2Face, NeuralTextures, FSGAN, MarioNETte)	Impersonate an identity by creating a video that the target had never expressed or did
Face morphing – create artificial biometric face samples that resemble biometric information of two or more individuals, usually applies to images	GAN (StyleGAN, MorGAN)	Fool the face recognition systems by using new morphed face samples at the image level
Lip-sync/features synthesis (pose, gaze direction, blinking) usually applies to videos	Recurrent neural networks (LSTMs, Neuraltexture), audio-to-video, text-to-video, “Do as I Do” Motion Transfer	Creation of video that the target had never expressed or did
Attribute manipulation – modify face attributes, e.g. skin colour, emotion, age, gender, makeup, beard on the face.	GAN (StarGAN, FaceApp mobile app)	The use of attribute manipulation for criminal activities is not significant.

1.6 Techniques Behind Deepfake Creation

Before building an effective detector for deepfake, it would be better to understand how to create deepfakes. As summarised in the second column of Table C, there are two primary deepfakes creation techniques, including Autoencoders and Generative Adversarial Networks.

Autoencoder (A.E.)

Autoencoder is a data reconstruction task that manipulates images such as swapping faces using two pairs of artificial encoders (compresses images) and decoders (reconstruct images). The two encoders perform the same task of identifying similar features on two faces during the model training process. Then, the decoders reconstruct Face B with the features of Face A.

Generative Adversarial Network (GAN)

GAN is a type of unsupervised machine learning model that consists of two artificial neural networks that compete against each other. The generative network constructs fake images using latent vectors and then passes the images to a discriminative network. The discriminator takes real and fake images as input and then attempts to distinguish whether the media is real or fake. During the model training process, the generator attempts to find out how to map the latent vectors to recognisable media that could fool the discriminator into believing the fakes are reals. Eventually, the generator constructs fake as realistic as possible, and the discriminator is eventually not able to distinguish between fakes and reals.

1.7 Trends in Deepfake Content

The creation of deepfake content does not require expertise in artificial intelligence. Instead, most of the deepfake content is created by non-technical experts for personal or commercial usage. Many factors contributed to this trend.

Firstly, deepfake is not proprietary or protected software possessed by a few hands, such as government agencies or scientific research labs. Source code for the deep learning model used for deepfakes creation is mostly readily available on GitHub or directly downloadable from app stores. Secondly, the general public can easily access hardware that facilitates deep learning, such as physical Graphics Processing Units (GPU) or subscription-based GPU. Moreover, datasets used for machine learning model training are freely available in the public domain.

Quantity of Deepfake Content

According to Deeptrace Lab (Ajder et al., 2019), the number of deepfake content shared by the netizens doubled from 7,964 videos to 14,678 videos in 9 months during 2019. Moreover, 96% of the deepfake videos were pornographic videos. Researchers found over 134 million total views in the top four websites dedicated to deepfake pornography in less than a year. Deepfake technology is a new type of online service. For instance, people could pay the U.S. $10 per word generated by custom voice cloning.

In October 2021, Sensity AI stated that the number of deepfake videos had doubled every six months, as shown in Figure 3. If their projection is correct, the number of deepfake videos as of December 2021 could reach nearly 340,188.

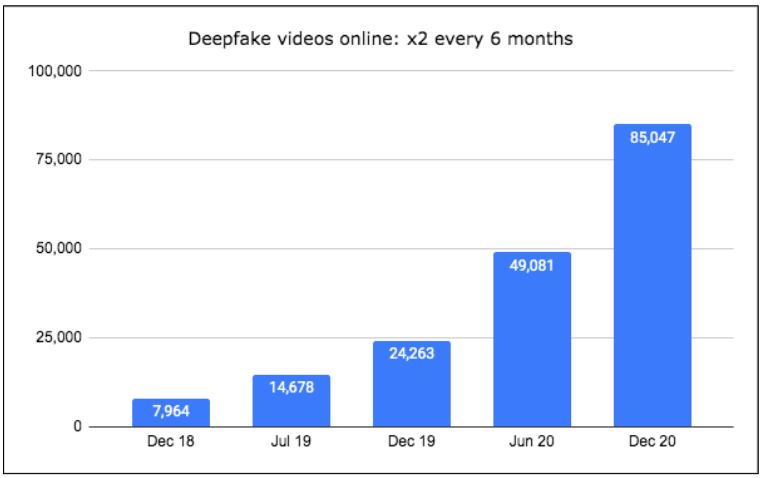


Figure 3 The image illustrates the recent number of fake videos (Sensity AI, 2021)

Quality of Deepfake Content

The quality of the synthetic media significantly improved over the years. Initially, it might still be possible for naked eyes to spot them due to abnormal movement or inconsistencies in the videos. However, the synthetic media's technology improves continuously, the outputs are getting more believable, and it is hard for the public to discern real or fake.

Generally, deepfakes have two generations. The first generation deepfakes are relatively low quality synthesised faces, especially for GAN-generated images before 2018, as shown in Figure 4. Most images or videos contain discolouration or visible boundaries on the target's face. Sometimes, strange artefacts such as weird hairstyles, inconsistent eye iris colours, imbalance ears. Thus, it would be easy to distinguish reals and fakes by human eyes if we look for these clues.

There are fewer discrepancies or discolouration in the second generation deepfakes, and the realism of images and videos significantly improved. In addition, the latest deepfakes have a more significant variety in terms of the surrounding environment of the subject or the subject itself. For instance, the subject is in different lighting conditions, different poses, wider distance from the camera.

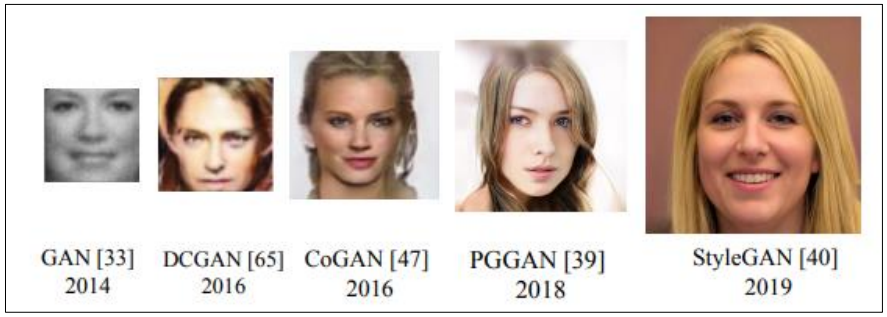


Figure 4 This image illustrates the advancement in GAN-generated images (Pu, 2020)

Chapter 2: Literature Review

2.1 Deepfake Detection Approaches

The research on deepfakes detection has flourished over the past few years due to the significance of cyber security risks related to deepfakes. Different methods are proposed by researchers, usually with a target to detect a particular type of deepfakes. For instance, some detectors detect fictitious images generated by the GAN technique. Mirsky et al. (2020) stated that researchers proposed several detection approaches and types of detectors:

1. Edge detector – focus on the blending boundary of the facial image
2. Environment detector – focus on the foreground or background of the image, such as lighting, residual from face warping processes
3. The forensic detector focuses on traces left on the image, such as the unique fingerprint left by the generative model and the camera's sensor noise.
4. Behavioural detector – focus on the mannerisms or emotions of the person
5. Physiologic detector – focus on biological signal-based information such as heart rate, eye blinking patterns of video
6. Synchronisation detector – focus on consistency of mouth shapes and phonemes
7. Coherence detector – focus on correlation and continuity of adjacent frames of video, such as flickers and jitter of the video
8. Classification detector – focus on the difference between real and fake images using general neural networks
9. Anormal detector – focus on outliners in prediction while only training the models with genuine data. Some researchers also make use of Benford's law to detect GAN-generated images, which is a method to calculate the distribution of the most significant digit for quantised Discrete Cosine Transform coefficients for the image classification task

The surveys of Tolosana et al. (2020) and Yu et al. (2021) suggested that most detectors could successfully detect the kind of deepfakes trained. In addition, however, detectors should distinguish unseen reals and fakes generated by any new generative techniques. Thus, both research teams suggested that generalisation is vital for future detectors.

While researchers have developed many different kinds of detectors, some detectors are already obsolete as the deepfake creation techniques are evolving rapidly. For instance, the detector tracks the eye blinking frequency, or visual artefacts (e.g. inconsistent eye colours, lip movements, and audio speech) became useless because the recent deepfakes creation has already considered these factors.

Another example is detecting deepfakes generated by StyleGAN and ProGAN based on the artefacts presented in the synthetic images. However, such GAN fingerprints are removable. To challenge the detection method focusing on artefacts generated by the GAN technique, some researchers (e.g. Neves et al., 2020) even created a dataset, iFakeFaceDB, containing 87,000 images without GAN fingerprints.

2.2 Deep Learning Detectors With Relatively High Performance

Based on the surveys conducted by Tolosana et al. (2020) and Yu et al. (2021), detectors show relatively high accuracy[1] against their own or publicly available database (e.g. FaceForensic++).

For GAN related face synthesis, Yu et al. (2019) proposed to analyse GAN fingerprints with a convolutional neural network (CNN). Yu found that every GAN technique inserted unique artefacts on the image during the image rendering process. The researcher achieved 99.5% accuracy when detecting features on images generated by ProGAN, SNGAN, CramerGAN and MMDGAN. However, other researchers found that this approach could not be generalised, especially if there is intention image augmentation, such as noise, blur, cropping or compression.

Like Yu, Dang et al. (2020) proposed using CNN classifiers (XceptionNet and VGG16 networks as the backbone networks) and further incorporated attention mechanisms in the detector to process feature maps for the classification. As a result, their detector reached 99% to 100% accuracy on images generated by ProGAN and StyleGAN and on videos/images taken from several deepfake forensic databases, including CelebA FFHQ, FaceForensics++ (Face2Face).

Rossler et al. (2019) also proposed the use of a CNN classifier for detecting identity swap/face swap as well as expression swap/face re-enactment. Again, they used XceptionNet as the backbone network for model training using the FaceForensics++ dataset built by themselves. Again, it achieved around 93% to 100% accuracy when they conducted testing on the FaceForensics dataset, including videos created by Deepfake and FaceSwap techniques.

Another popular deepfake detector for the benchmark is MesoNet. Afchar et al. (2018) proposed the study of mesoscopic features of deepfakes using a CNN classifier. MesoNet4 is capable of detecting identity swap/face swap as well as expression swap/face re-enactment. Their detector achieved around 96% accuracy on the FaceForensics++ dataset for the raw video files generated by Face2Face and NeuralTextures techniques. However, Tolosana noted that this detector could not generalise on a more recent deepfake dataset. Their review found that the MesoNet4 only reached 75.3% on the DFDC Preview dataset and 54.8% on Celeb-DF (version 1) dataset.

Like many other researchers, Wang and Dantcheva (2020) used a 3DCNN classifier and achieved 90% to 95% on FaceForensics++ datasets, including low-quality videos generated by Face2Face. Tarasiou, M. and Zafeiriou, S. (2020) also developed a CNN architecture to extract features. Their binary classifier achieved 91% to 99% accuracy on the FaceForensics++ dataset at different video compression levels.

1. For comparing different models, this project compares the Area Under Curve (AUC) disclosed by the authors. Papers that declared that their models could achieve AUC over 90% are regarded as high accuracy.

In the work proposed by Bonettini et al. (2021), they experimented with two networks, including XceptionNet and EfficientNet and several variants of the EfficientNet. They proposed to ensemble EfficientNetB4 with attention layers and Siamese during model training. Their detector achieved 94.4% accuracy on FaceForensics++. However, Yu argued that such network-based algorithms are prone to overfitting.

Ciftci et al. (2020) developed a heart rate detector to evaluate the subtle changes of colour and motion in videos. Their experiments validated that spatial coherence and temporal consistency of heart rate signals are not well preserved in deepfake videos. Their CNN classifier attained around 93% to 96% accuracy on the FaceForensics++ videos, created by DeepFake, Face2Face and FaceSwap.

2.3 Problems of Existing Detectors

Tolosana et al. (2020) mentioned two significant problems with the existing detectors. Firstly, the detectors were evaluated in the same conditions that they trained. This controlled scenario is unrealistic as deepfake videos or images are usually shared on social networks with high variations on compression level, resizing, noise. Secondly, the generalisation ability of the detectors against unseen conditions requires improvement because facial manipulation techniques are constantly improved.

Tolosana suggested that to make use of fusion techniques on detection. For instance, combine steganalysis and pure deep learning features. It is also suggested not to use deepfake videos for training, such that the detector could better generalise on unseen deepfake videos or images. Also, researchers should collect additional source information from the videos for detection. For instance, analyse the text and audio accompanying the videos when uploading to the social networks.

Chapter 3: The Artefact

3.1 An Overview

Figure 5 shows an overview of the workflow of this project. The ultimate goal of this project is to develop a binary image detector that could effectively classify videos/images into ‘real’ or ‘deepfake’. In particular, this project focus on deepfakes that applied face swap or identity swap techniques. Since deep learning is used for deepfakes creation, it is logical to build a deepfake detector by applying the same method to facilitate such detection. For simplicity, this project uses Google Colab for implementation.

Instead of building neural network layers from scratch, this project aims to apply transfer learning from a pre-trained network for image classification. Transfer learning means transferring the knowledge learned from doing a task to a similar task. This technique could speed up the model building process as the weights learned in the network could be transferred and fine-tuned when applying to a new task. This method was used by some researchers mentioned in Section 2 above.

Precisely, this project assesses ten pre-trained networks trained on the ImageNet dataset, a large dataset consisting of 1.4 million images and 1000 classes such as animals and food. ImageNet does not contain ‘real’ or ‘deepfake’ classes. Thus, it is exciting to experiment and work out the best pre-trained network on this new task.

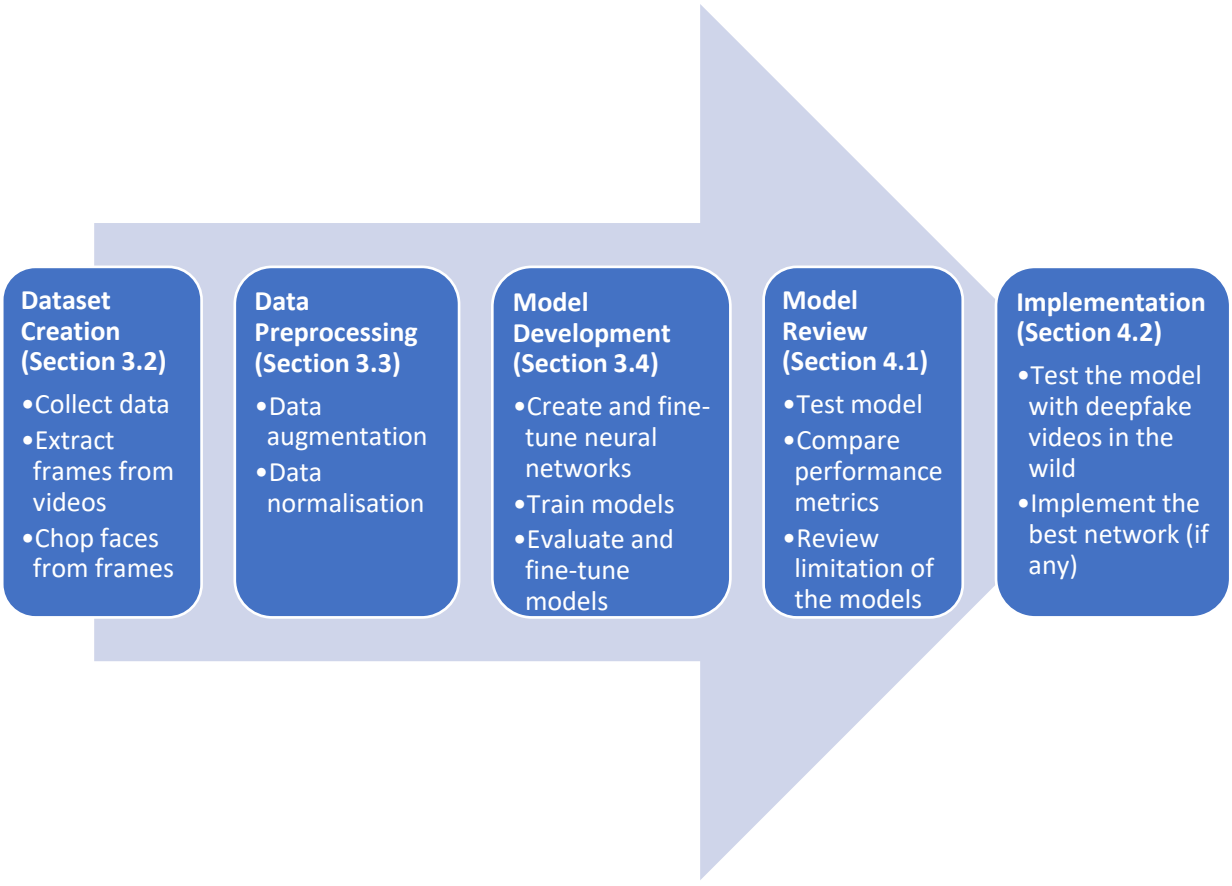


Figure 5 An overview of the project workflow

The first step of this project is to collect videos and images containing the types of deepfake being concerned. Then, to ensure the model focus on the face region of the target, a dataset is created to store only the faces of the videos or images. Section 3.2 walkthrough the tools, process and codes.

Section 3.3 walkthrough the data preprocessing tasks performed before inputting the dataset into the network for training. All the images for training, validation, and testing are normalised. Moreover, the training dataset is augmented such that every picture used for model training would look differently.

The next step is to build the model based on a pre-trained model. Ten state-of-the-art networks are assessed one by one. After training for ten epochs, fine-tune the model and continue training for ten more epochs.

After training, the most crucial step of the project is to test the model performance by using the same set of unseen test datasets on the ten trained models. One hundred fake and one hundred authentic images were used to verify their ability to classify fakes and reals correctly.

The most critical step of the artefact creation is the experiment shown in Section 4.1. All ten models are put into an objective evaluation using the same set of test datasets for prediction. The performance metrics are collected and analysed to determine which model can achieve the highest overall accuracy. Moreover, this project also reviews misclassified images to find out their characteristics.

Generalisation is an essential element of a successful detector. Thus a few real videos and face swap videos are further captured from YouTube for final checking to confirm this model could adequately classify the videos as real or fake.

3.2 Dataset Creation

In most cases, large datasets are necessary for machine learning algorithms to learn the image feature maps before deepfake creation or detection. In the survey paper issued by Tolosana et al. in 2020, the research team looked into the datasets used for different kinds of deepfakes creation and detection. Many researchers assess their detectors against these publicly available databases as a benchmarking process. In contrast, some of the researchers used their datasets to check the accuracy of their detectors. However, it was not easy to compare their detectors with their counterparties.

As the project focuses on addressing the deepfakes related to impersonation, the dataset used for model training should include videos and images related to identity swap/face swap. In addition, there are a few publicly available databases for related research, as shown in Table D.

Table D: Deepfake Databases on Identity Swap / Face Swap

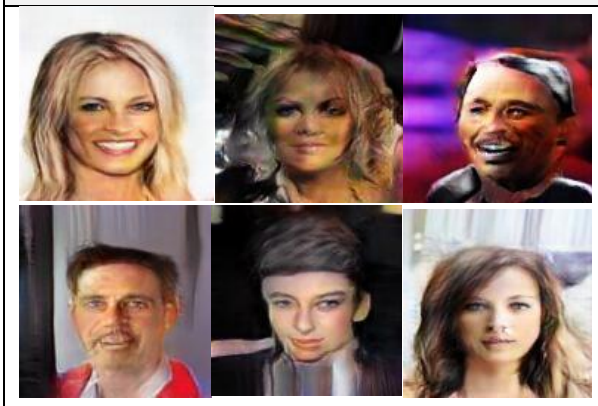
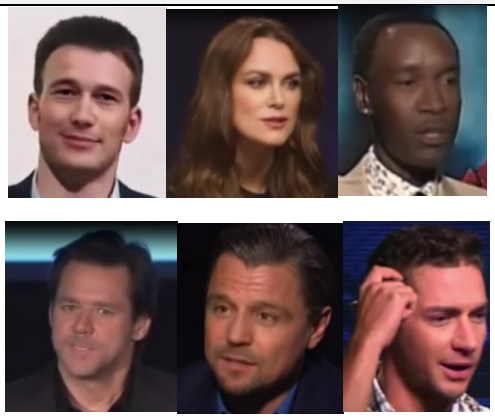
Year	Database Name	Real Video	Fake Video
2018	UADFV	49	49 (from FakeApp)
2018	DeepfakeTIMIT	-	620 (from faceswap-GAN)
2019	FaceForensics++	1000	4000 (from Face2Face, FaceSwap, DeepFakes and NeuralTextures)
2019	Google DFD	363	3068
2019	DFDC Preview	77	323
2019	DFDC	23,654	104,500
2019	Celeb-DF (version 1)	890	5,639
2019	Celeb-DF (version 2)	262	1,189
2020	DeeperForensics	10,000	50,000

However, two major issues were found on the existing databases, as listed below.

1. The number of videos between genuine and deepfake is not balanced. Most of the databases contain much more fake videos than real videos. Each real video is usually used for making multiple fake videos in the dataset. For example, in the DeepfakeTIMIT database, there are no real videos.
2. The quality of some deepfake videos in the datasets is not satisfactory. As shown in Table E, people could easily differentiate deepfakes in version 1 of Celeb-DB due to its low realism, but the deepfakes videos in version 2 are significantly improved. However, it is difficult for naked eyes to differentiate between real and deepfake videos.

Hence, direct use of these datasets without reviewing and filtering out unsatisfactory videos and images might result in dissatisfaction during model creation.

Table E: Images of fake faces from Celeb-DB version 1 and Celeb-DB version 2

Examples of Fake Faces	
Celeb-DF (version 1)	**Celeb-DF (version 2)**
	

Collect Data

This project creates a dataset using deepfake videos and images with very high realism regarding the issues noted above. The datasets sourced from:

- videos from DFDC Preview
- videos from Celeb-DF (version 2)
- images from Deepfake Database used by MesoNet

For the dataset collected from Kaggle, namely DFDC Preview, there are 800 videos downloadable for training and testing. This project mainly used 400 training videos containing the labels of 'real' and 'deepfake' for each video. This dataset contains 77 real videos and 323 deepfake videos. The other 400 testing videos do not have labelling (fake/real). The dataset collected from Github, namely Celeb-DF (version 2), contains 262 real videos and 1,189 deepfake videos.

Conversion from videos to frames and frames to images of faces for the Deepfake Database used by MesoNet is directly downloadable without the need for further processing. The steps below are mainly for processing the video files in the DFDC Preview and Celeb-DF (v2) datasets.

Extract frames from videos

After downloading the videos and images, they are stored in Google Drive. After opening a new notebook at Google Colab, set up a temporary environment and import libraries (Figure 6). In this part of the project, the essential libraries include:

- OpenCV – for computer vision task
- Scikit-learn – for resizing the cropped image
- Matplotlib – for visualisation of images

```
# Set up environment and import libraries
!pip3 install --upgrade pip > /dev/null
!pip3 install scikit-image
!pip3 install opencv-python
!pip3 install opencv-contrib-python

# Import libraries and modules
import cv2
from skimage.color import rgb2gray
from skimage.transform import resize
import matplotlib.pyplot as plt
import math
import os
```

Figure 6 Set up environment and import libraries

Connect to Google drive, input user credentials for access authentication (Figure 7). Upon successful connection, copy all the mp4 files from my video file directory in Google Drive to the temporary directory in Google Colab (INPUT_VIDEOS_PATH) to prepare for image processing (Figure 8).

```
# Directory tree structure in Google drive
#       /DFD ─┐
#             └──────── /sources ──────────┐
#                                          ├─ videos (.mp4)
#                                          └─ /frames (.jpg)

!mkdir -p /content/DFD/frames > /dev/null

# Confirm Google Drive is connected
google = !if [ -d 'GDrive/' ]; then echo "1" ; else echo "0"; fi
if (google[0] is '0' ):
  from google.colab import drive
  drive.mount('/content/GDrive/')
```

Figure 7 Connect Google drive and set up a temporary directory in Google Colab

Define variable one_frame_each as 120 for extracting one frame every second from a video.

```
# Folder path contains the videos
INPUT_VIDEOS_PATH = '/content/DFD/Sources'

# Folder path for storing image frames to be extracted from videos
OUTPUT_FRAMES_PATH = '/content/DFD/Sources/frames'

# Create variables
# frame_name = 'frame'
one_frame_each = 120

!if [ -d {OUTPUT_FRAMES_PATH} ]; then echo "Output to be stored in "{OUTPUT_FRAMES_PATH} ;
    else mkdir {OUTPUT_FRAMES_PATH} && echo "Output directory created"; fi

videofiles = !ls {INPUT_VIDEOS_PATH}/*.mp4
```

Figure 8 Set file paths and variables

For video files in the Google Colab, use OpenCV cv2.VideoCapture method to extract frames from videos. If frame captures have a size greater than 640 pixels, resize it. Then, save each frame into a jpg file under a separate folder in the temporary directory in Google Colab (OUTPUT_FRAMES_PATH) (Figure 9). Keep capturing frames for the video until the end of the video. After capturing the frame of one video, start processing the next video in the INPUT_VIDOES_PATH until all the videos are processed.

```
# Extract frames from videos
for videofile in videofiles:
    count = 0
    success = True
    filename = os.path.basename(videofile)
    vidcap = cv2.VideoCapture(videofile)

    while success:
        if (count%one_frame_each == 0):
            success,image = vidcap.read()
            if image.shape[1]>640:
                tmp = resize(image, (math.floor(640 / image.shape[1] * image.shape[0]), 640))
            plt.imsave("%s/%s%d.jpg" % (OUTPUT_FRAMES_PATH,filename,count), tmp, format='jpg')
            print ('*', end="")
        else:
            success,image = vidcap.read()
        count += 1
```

Figure 9 Extract frames from videos

After completing the frame capturing process, copy all the jpg files back to Google Drive to prepare for face chopping (Figure 10).

```
# Count number of image frames in Google Colab's temporary directory
!ls /content/DFD/Sources/frames | wc -l

# Copy image frames stored in temporary directory to Google Drive
# Reminder - select one of the folders for processing
!cp DFD/Sources/frames/* GDrive/My\ Drive/DFD/data/DFDCpreview/frames/train
!cp DFD/Sources/frames/* GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celereal
!cp DFD/Sources/frames/* GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celesyn

# Check to confirm all image frames stored in Google Drive
!ls GDrive/My\ Drive/DFD/data/DFDCpreview/frames/train | wc -l
!ls GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celereal | wc -l
!ls GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celesyn | wc -l
```

Figure 10 Confirm all the jpg files are copied to Google Drive

Extract faces from frames

Before extracting faces from the frames captured in the videos, in the runtime of Google Colab, it is a must to enable the use of GPU for running face recognition library. Similar to the previous step, set up a temporary environment and folder in Google Colab (Figure 11). Copy the jpg files into the folder.

```
# Directory tree structure in Google drive
#      /DFD ─┐
#            └─── /Sources ──────┐
#                                ├─ images (.jpg)
#                                └─ /Faces (.jpg)

!mkdir -p /content/DFD/Sources/Faces > /dev/null
```

Figure 11 Set up a temporary directory in Google Colab

Import necessary libraries (Figure 12). In this part of the project, the essential libraries include Open CV and face detection with dlib for face chopping from the frames.

```
!apt-get install build-essential cmake
!apt-get install libopenblas-dev liblapack-dev
!pip install dlib
!pip install face_recognition
!pip install opencv-python
!pip install opencv-contrib-python

from PIL import Image
import cv2
import face_recognition
import matplotlib.pyplot as plt
import math
import os
```

Figure 12 Import libraries

Define the folder paths for file processing (Figure 13).

```
# Folder path contains the frames
INPUT_FRAMES_PATH = '/content/DFD/Sources'

# Folder path for storing faces to be extracted from images
OUTPUT_FACES_PATH = '/content/DFD/Sources/Faces'

!if [ -d {OUTPUT_FACES_PATH} ]; then echo "Output to be stored in "{OUTPUT_FACES_PATH} ;
else mkdir {OUTPUT_FACES_PATH} && echo "Output directory created"; fi
imagefiles = !ls {INPUT_FRAMES_PATH}/*.jpg
```

Figure 13 Set file paths and variables

For image files in the Google Colab (INPUT_FRAMES_PATH), use dlib's face recognition to detect the face region, and then use OpenCV cv2.write method to extract face from the frame. Then, save the jpg file under a separate folder in the temporary directory in Google Colab (OUTPUT_FACES_PATH) (Figure 13). After chopping the face of one frame, start processing the next frame in the INPUT_FRAMES_PATH until all the frames are processed.

```
# Extract face from all image files in the given directory
for imagefile in imagefiles:
    filename = os.path.basename(imagefile)
    image = face_recognition.load_image_file(imagefile)
    face_locations = face_recognition.face_locations(image)
    print("Found {} face(s) in this photograph - %s.".format(len(face_locations)),filename)

    %matplotlib inline
    plt.rcParams['figure.figsize'] = [32, 8]

    for face_location in face_locations:
        top, right, bottom, left = face_location
        print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right: \
        {} for %s".format(top, left, bottom, right), filename)
        face_image = image[top:bottom, left:right]

        if(len(face_locations) == 1):
            cv2.imwrite("%s/%sF.jpg" % (OUTPUT_FACES_PATH,filename), face_image)
            print ('*', end="")
```

Figure 14 Extract faces from frames

```
# Check to confirm all image frames stored in Google Drive
!ls DFD/Sources/Faces | wc -l

# Copy image frames stored in temporary directory to Google Drive
# Reminder - select one of the folders for processing
!cp DFD/Sources/Faces/* GDrive/My\ Drive/DFD/data/DFDCpreview/faces/train
!cp DFD/Sources/Faces/* GDrive/My\ Drive/DFD/data/CelebDFv2/faces/celereal
!cp DFD/Sources/Faces/* GDrive/My\ Drive/DFD/data/CelebDFv2/faces/celesyn

# Check number of face images stored in Google drive for completeness check
# Reminder - select one of the folders for processing
!ls GDrive/My\ Drive/DFD/data/C/faces/celetrain | wc -l
!ls GDrive/My\ Drive/DFD/data/C/faces/celereal | wc -l
!ls GDrive/My\ Drive/DFD/data/C/faces/celesyn | wc -l
```

Figure 15 Copy and confirm all the jpg files are duplicated to Google Drive

After the previous steps, download and organise the face images on the local computer. These pictures are selected and arranged to create balanced training, validation or test datasets.

A total of 18,278 pictures were used in this project (Figure 16). The proportion of the datasets is 80% (14,539) for training, 19% (3,539) for validation and 1% (200) for testing. The proportion for the two classes is 49% (9030) for fake faces and 51% (9248) for real faces. Table F shows the proportion of two classes by the data source.

	FAKE	REAL	Grand Total
train	**7162**	**7377**	**14539**
Celeb-DFv2	398	836	1234
deepfakeDB	4102	5850	9952
DFDCpreview	2662	691	3353
validation	**1768**	**1771**	**3539**
Celeb-DFv2	101	200	301
deepfakeDB	1001	1400	2401
DFDCpreview	666	171	837
test	**100**	**100**	**200**
Celeb-DFv2	35	34	69
deepfakeDB	65	66	131
Grand Total	**9030**	**9248**	**18278**

	FAKE	REAL	Grand Total
train	**49%**	**51%**	**100%**
Celeb-DFv2	32%	68%	100%
deepfakeDB	41%	59%	100%
DFDCpreview	79%	21%	100%
validation	**50%**	**50%**	**100%**
Celeb-DFv2	34%	66%	100%
deepfakeDB	42%	58%	100%
DFDCpreview	80%	20%	100%
test	**50%**	**50%**	**100%**
Celeb-DFv2	51%	49%	100%
deepfakeDB	50%	50%	100%
Grand Total	**49%**	**51%**	**100%**

Figure 16 An overview of the dataset

Table F: Proportion of face images by data source

Source	Real Faces	Fake Faces	Total
Deepfake Database	57% (5168)	79% (7316)	68% (12,484)
DFDC Preview	37% (3328)	9% (862)	23% (4,190)
Celeb-DF (version 2)	6% (534)	12% (1070)	9% (1,604)
	100% (9248)	100% (9030)	100% (18,278)

Upload images to Google drive with the folder structure shown in Figure 17. Then, the myDBmeta.json is created based on the CSV lists generated from Google drive and then converted to JSON on an online file converter.

```
# MyDrive/DFD/data ─┐
#                   └──/MyDB──────┐
#                                 ├─ /test
#                                 │        └─ /Fake (images here)
#                                 │        └─ /Real (images here)
#                                 └─ /train ─┐
#                                 │        └─ /Fake (images here)
#                                 │        └─ /Real (images here)
#                                 └─ /validation ─┐
#                                 │               └─ /Fake (images here)
#                                 │               └─ /Real (images here)
#                                 └─ myDBmeta.json
```

Figure 17 File structure in Google Drive

3.3 Data Preprocessing

Several libraries must be inputted before data augmentation and normalisation (Figure 18). The libraries include

- os – for file handling
- NumPy – for array operation
- TensorFlow – for machine learning
- matplotlib – for image visualisation

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

Figure 18 File structure in Google Drive

Set the path of the folders (Figure 19) according to the file structure in Google Drive (Figure 17). Define the visualisation setting on images (Figure 20) and execute the code (Figure 21) to display images (Figure 22).

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

Figure 19 File path set up

```
# Define matplotlib parameters for output images as 4x4 images
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

nrows = 4
ncols = 4

pic_index = 0                                # index for image iteration
```

Figure 20 Define matplotlib parameters for output images

```
# Rerun the cell to fetch a new batch of images

fig = plt.gcf()                              # set up matplotlib fig
fig.set_size_inches(ncols * 4, nrows * 4)    # size matplotlib fig to fit into a 4x4 image

pic_index += 8
next_real_pix = [os.path.join(train_real_dir, fname)
                 for fname in train_real_fnames[pic_index-8:pic_index]]
next_fake_pix = [os.path.join(train_fake_dir, fname)
                 for fname in train_fake_fnames[pic_index-8:pic_index]]

for i, img_path in enumerate(next_fake_pix+next_real_pix):
  sp = plt.subplot(nrows, ncols, i + 1)      # Set up subplot, subplot indices start at 1
  sp.axis('Off')                             # Turn off axis

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```

Figure 21 Use matplotlib to display images

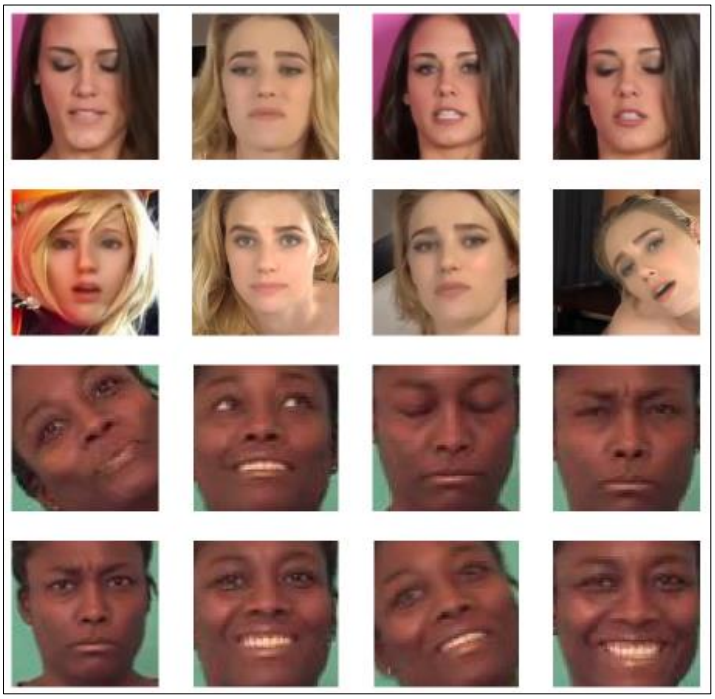


Figure 22 Real and fake images displayed by matplotlib

Data augmentation

Define the image size (IMG_SIZE) as 150 pixel by 150 pixel and batch size for processing 40 images for each epoch, except for NASNetMobile, which require a larger pixel, and it is set to 224 pixel by 224 pixel and MobileNet, which require a slightly larger image input of 160 by 160 pixel. Take the images from the directory defined in Figure 19 to create three datasets using TensorFlow Keras utilities' image_dataset_from_directory'. Shuffle the training and validation datasets during model training, but not for testing datasets (Figure 23).

```
BATCH_SIZE = 40
IMG_SIZE = (224, 224)

train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

Figure 23 Create datasets

Visualise a batch of training images with the label (Figure 24, 25).

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

Figure 24 Visualise images

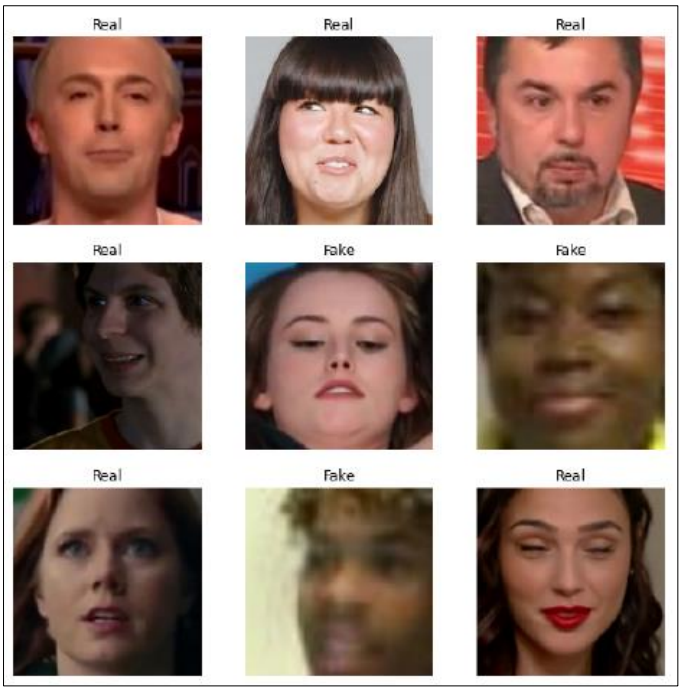


Figure 25 A batch of training images with label

To enhance data performance, use dataset prefetch (Figure 26) to load images from disk to a buffer to prevent I/O blocking.

```
AUTOTUNE = tensorflow.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Figure 26 Enhance data performance

Horizontally flip and rotate the image randomly (Figure 27) to transform an image to resemble multiple images (Figure 28). Augmentation aims to minimise overfitting by introducing sample diversity during model training. Data augmentation is applied only to the training dataset but not to the validation or test dataset.

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])

for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```

Figure 27 Image augmentation

Figure 28 Visualise an augmented image

Data normalisation

The pixel values in the images in the datasets are in [0, 255], but the pre-trained model expects the pixel values in [-1, 1]. Therefore, to normalise the images, use Tensorflow Keras's preprocess_input method of the specific pre-trained network to rescale the images. For instance, Figure 29 shows the preprocess_input method for NSANetMobile.

```
preprocess_input = tensorflow.keras.applications.nasnet.preprocess_input
```

Figure 29 Rescale images

3.4 Model Development

The next step is to create the model based on a pre-trained network. In Keras Applications, there are 26 deep learning models with pre-trained weights. This project selects ten models for assessments, as shown in Table H. The Figures disclosed by Keras, Top-1 accuracy and Top-5 accuracy, refer to the model's performance on the ImageNet validation dataset. Depth refers to the topological depth of the network. It includes activation layers, batch normalisation layers and so on.

Table H: Selected models for model creation

Model Ref	Model	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
M1	Xception	0.79	0.945	20,861,480	126
M2	VGG19	0.713	0.9	20,024,384	26
M3	ResNet152V2	0.78	0.942	58,331,648	-
M4	InceptionV3	0.779	0.937	21,802,784	159
M5	InceptionResNetV2	0.803	0.953	54,336,736	572
M6	MobileNetV2	0.713	0.901	2,257,984	88
M7	DenseNet201	0.773	0.936	18,321,984	201
M8	NASNetMobile	0.744	0.919	4,269,716	-
M9	EfficientNetB3	-	-	10,783,535	-
M10	EfficientNetB7	-	-	64,097,687	-

Create and fine-tune neural networks

For illustration, instantiate the NASNetMobile model as shown in Figure 30. The weights are automatically downloaded when instantiating the model. As a convention for transferring the knowledge on feature extraction, include_top is set to 'False' to exclude the classification layers at the top of the neural network.

```
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.nasnet.NASNetMobile(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')
```

Figure 30 Load pre-trained model

Freeze the convolutional base by setting base_model.trainable to 'False' (Figure 31) to prevent the model's weights from being updated during training. All the parameters become non-trainable after setting up this configuration.

```
base_model.trainable = False
```

Figure 31 Freeze the base model

Apply the GlobalAveragePooling2D layer (Figure 32) and a Dense layer (Figure 33) to convert the features into a prediction value per image. Positive values refer to authentic images (class 1), and negative values refer to fake ones (class 0).

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
```

Figure 32 Add GlobalAveragePooling2D layer

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
```

Figure 33 Add Dense layer

Chain the data augmentation, data preprocessing, base model, global average layer and the dense layer together, build and compile the model as shown in Figure 34. Image size is rescaled to 150 pixels by 150 pixels for all models, except for NASNetMobile, which requires a larger image size (224 pixels by 224 pixels) and MobileNetV2, which requires 160 pixels by 160 pixels as an input. Use BinaryCrossentropy for classifications with only two classes. Set from_logits to True to generate linear values.

```
inputs = tensorflow.keras.Input(shape=(150, 150, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Figure 34 Build and compile the model

Use model.summary() to check the number of trainable and non-trainable parameters in the model and summarise the values in Table H.

Table H: Trainable parameters (before fine-tune)

Model Ref	Model	Trainable Parameters	Non-trainable Parameters	Total
M1	Xception	2,049	20,861,480	20,863,529
M2	VGG19	513	20,024,384	20,024,897
M3	ResNet152V2	2,049	58,331,648	58,333,697
M4	InceptionV3	2,049	21,802,784	21,804,833
M5	InceptionResNetV2	1,537	54,336,736	54,338,273
M6	MobileNetV2	1,281	2,257,984	2,259,265
M7	DenseNet201	1,921	18,321,984	18,323,905
M8	NASNetMobile	1,057	4,269,716	4,270,773
M9	EfficientNetB3	1,537	10,783,535	10,785,072
M10	EfficientNetB7	2,561	64,097,687	64,100,248

Train models

Train the model using the training dataset for ten epochs. At the same time, evaluate the model using the validation dataset. Capture the accuracy and loss during training (Figure 35, Figure 36).

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

Figure 35 Set up the number of epoch, loss and accuracy

```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

Figure 36 Trian the model

Evaluate models

After training for ten epochs, summarise the accuracy and loss in Table I. Visualise the training and loss for the training and validation datasets (Figure 37).

Table I: Training and validation accuracy

Model Ref	Model	Training Accuracy	Validation Accuracy
M1	Xception	0.7343	0.6931
M2	VGG19	0.6876	0.6671
M3	ResNet152V2	0.7342	0.7358
M4	InceptionV3	0.7302	0.6948
M5	InceptionResNetV2	0.6972	0.7203
M6	MobileNetV2	0.7553	0.7115
M7	DenseNet201	0.7378	0.7578
M8	NASNetMobile	0.6904	0.6796
M9	EfficientNetB3	0.7492	0.7299
M10	EfficientNetB7	0.7399	0.6756

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

Figure 37 Use matplotlib to visualise the training and validation result

Figure 38 illustrates the result for NASNetMobile after ten epochs. All other nine selected models show similar trends.

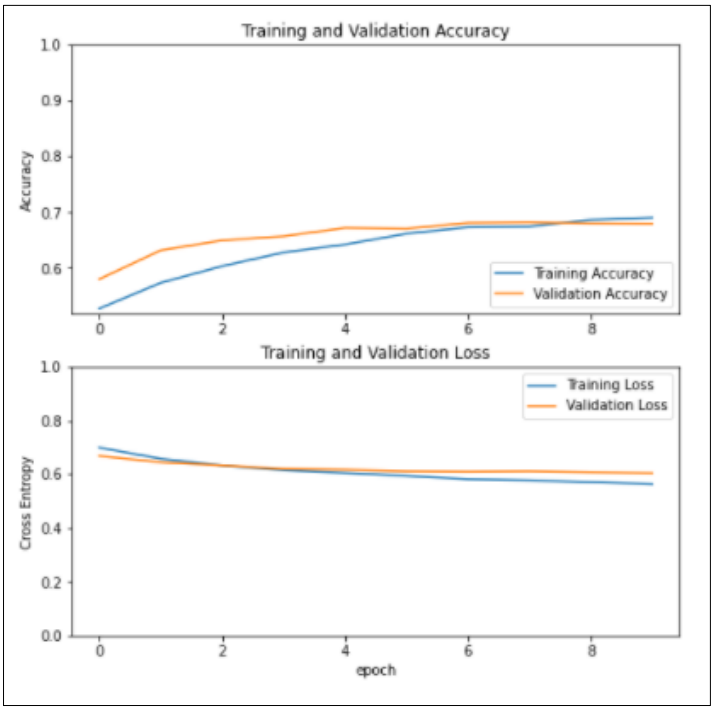


Figure 38 Visualise the loss and accuracy for the training and validation datasets

Fine-tune models

To enhance the performance, fine-tune the model by updating its weights. Unfreeze the base model and train a small number of top layers for ten more epochs (Figure 39). Use an arbitrary number (value 100) to fine-tune the models in this project. The rationale is that the first few layers only learn the generic features of image classification. The deeper layers provide increasingly specific features to the ImageNet dataset.

```
base_model.trainable = True

fine_tune_at = 100

for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

Figure 39 Unfreeze the top layers

Recompile the model with RMSprop with a lower learning rate to avoid overfitting (Figure 40).

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

Figure 40 Recompile the model

Use model.summary() to check the number of trainable and non-trainable parameters in the model and summarise the values in Table J.

Table J: Trainable parameters (after fine-tune)

Model Ref	Model	Number of layers in the base model	Trainable Parameters	Non-trainable Parameters	Total
M1	Xception	132	9,480,393	11,383,136	20,863,529
M2	VGG19	22	513	20,024,384	20,024,897
M3	ResNet152V2	564	56,396,545	1,937,152	58,333,697
M4	InceptionV3	311	19,628,417	2,176,416	21,804,833
M5	InceptionResNetV2	780	53,510,161	828,112	54,338,273
M6	MobileNetV2	154	1,862,721	396,544	2,259,265
M7	DenseNet201	707	17,290,177	1,033,728	18,323,905
M8	NASNetMobile	769	4,210,713	60,060	4,270,773
M9	EfficientNetB3	384	10,577,383	207,689	10,785,072
M10	EfficientNetB7	813	63,663,721	436,527	64,100,248

Note: fine-tune starts at layer 100 (Figure 39). Thus there are more trainable parameters if the number of layers in the base model increase.

Train models

Resume training for ten more epochs (Figure 41). At the same time, evaluate the model using the validation dataset. Meanwhile, capture accuracy and loss.

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

Figure 41 Resume training the model

Evaluate models

After training for ten more epochs, summarise the accuracy and loss at the 10th and 20th epoch in Table K. Correlate to the number of trainable parameters in Table J. The model accuracy improved with more trainable parameters after fine-tune.

Table K: Training and validation accuracy

		Before Fine-tune (10th epoch)		After Fine-tune (20th epoch)		Improvement	
Model Ref	**Model**	**Training Accuracy**	**Validation Accuracy**	**Training Accuracy**	**Validation Accuracy**	**Training Accuracy**	**Validation Accuracy**
M1	Xception	0.7343	0.6931	0.9393	0.8254	21% ↑	13% ↑
M2	VGG19	0.6876	0.6671	0.695	0.6796	1% ↑	1% ↑
M3	ResNet152V2	0.7342	0.7358	0.9831	0.8556	25% ↑	12% ↑
M4	InceptionV3	0.7302	0.6948	0.978	0.8565	25% ↑	16% ↑
M5	InceptionResNet V2	0.6972	0.7203	0.9874	0.8539	29% ↑	13% ↑
M6	MobileNetV2	0.7553	0.7115	0.9409	0.8302	19% ↑	12% ↑
M7	DenseNet201	0.7378	0.7578	0.9842	0.9265	25% ↑	17% ↑
M8	NASNetMobile	0.6904	0.6796	0.9681	0.8533	28% ↑	17% ↑
M9	EfficientNetB3	0.7492	0.7299	0.9834	0.8483	23% ↑	12% ↑
M10	EfficientNetB7	0.7399	0.6756	0.9909	0.8734	25% ↑	20% ↑

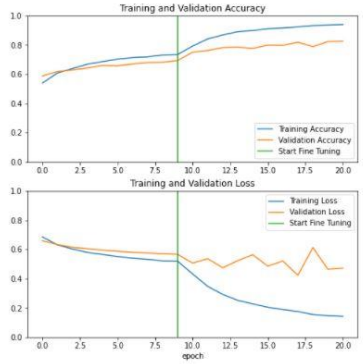
Visualise the training and loss for the training dataset and validation dataset for all the models (Figure 42).

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

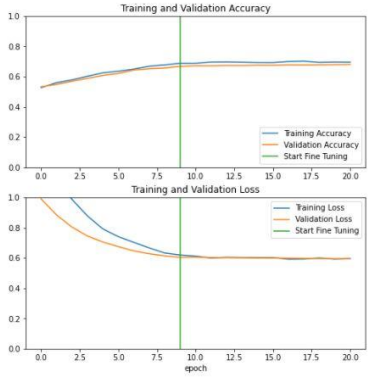
Figure 42 Use matplotlib to visualise the training and validation result

Visualise the results in Figure 43. From these graphs, although there is increasing training accuracy and reducing training loss for all the models, the validation accuracy is increased less than that of training accuracy, and there is increasing validation loss. Therefore, it implies a certain degree of overfitting, and there is a risk of overfitting the model. Therefore, further testing is needed to confirm the accuracy of these models.

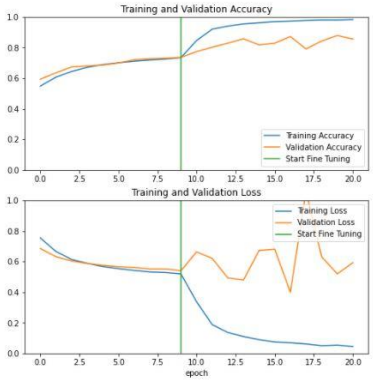
Xception



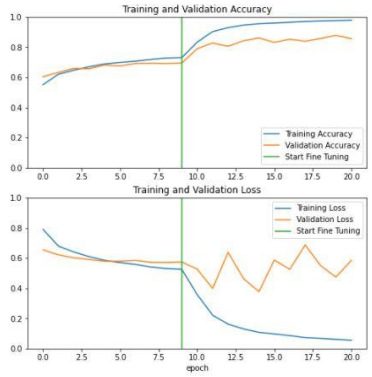
VGG19



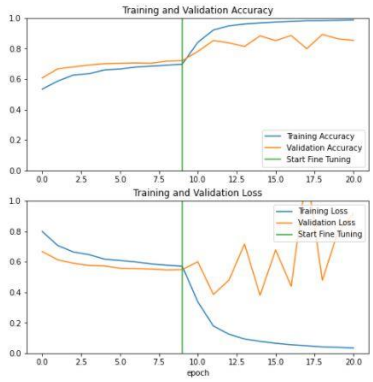
ResNet152V2



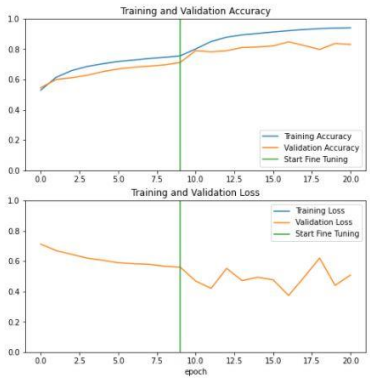
InceptionV3



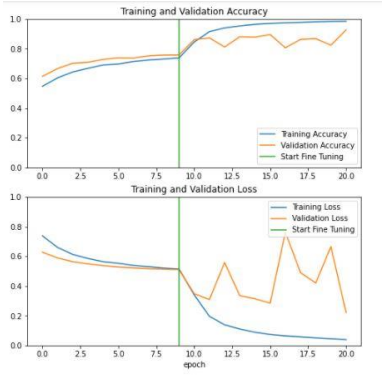
InceptionResNetV2



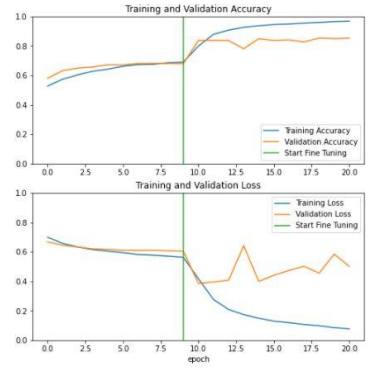
MobileNetV2



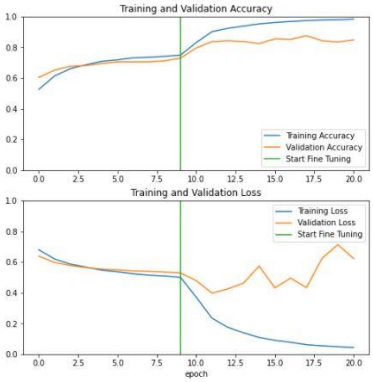
DenseNet201



NASNetMobile



EfficientNetB3



EfficientNetB7

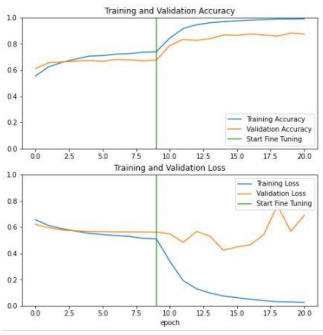


Figure 43 Result of training and validation

Chapter 4: The Experiment

4.1 Model Review (Test Dataset)

Conduct testing using testing datasets to assess the model performance. It consists of 100 authentic and 100 fake images (Figure 44).

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

Figure 44 Check number of images in the test dataset

Test model

During testing, capture the accuracy and loss for each model. Then, summarise the results of testing accuracy for all models in Table L, with the training and validation accuracy. As mentioned in the previous section, there are overfittings on the models, resulting in lower testing accuracy and training accuracy.

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

Figure 45 Evaluate model with the test dataset

Table L: Training, validation and testing accuracy

Model Ref	Model	Training Accuracy	Validation Accuracy	Testing Accuracy
M1	Xception	0.9393	0.8254	0.7250
M2	VGG19	0.695	0.6796	0.6250
M3	ResNet152V2	0.9831	0.8556	0.8400
M4	InceptionV3	0.978	0.8565	0.8200
M5	InceptionResNetV2	0.9874	0.8539	0.7350
M6	MobileNetV2	0.9409	0.8302	0.7450
M7	DenseNet201	0.9842	0.9265	0.8400
M8	NASNetMobile	0.9681	0.8533	0.7450
M9	EfficientNetB3	0.9834	0.8483	0.8250
M10	EfficientNetB7	0.9909	0.8734	0.8150

Compare performance metrics

For detailed analysis on image level, upload the authentic images (Figure 46) and fake images (Figure 47) separately and assess how good the models perform. Then, summarise the results in Table M (real images) and Table N (fake images). The model reference ('M1' to 'M10') refers to the model name shown in Table L.

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(224, 224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Figure 46 Assess real images

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(224, 224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Figure 47 Assess fake images

The results in Table M have the following indicators:

- ‘TP’– true positive, real images correctly predicted as real images
- ‘FN’– false negative, real images incorrectly predicted as deepfake images

Refer to Appendix 5 for real images for testing.

Table M: Predictions on real images

Ref	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	% of TP	% of FN
R1	TP	FN	TP	TP	FN	FN	TP	FN	TP	FN	50%	50%
R2	TP	TP	TP	TP	FN	FN	TP	TP	TP	FN	70%	30%
R3	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R4	FN	FN	TP	TP	TP	TP	TP	TP	TP	TP	80%	20%
R5	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R6	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R7	TP	FN	FN	FN	FN	TP	TP	TP	TP	FN	50%	50%
R8	TP	TP	TP	FN	FN	TP	FN	FN	FN	FN	40%	60%
R9	FN	TP	TP	FN	FN	FN	TP	TP	FN	FN	40%	60%
R10	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R11	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R12	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R13	FN	TP	TP	TP	FN	FN	FN	FN	TP	FN	40%	60%
R14	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R15	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R16	FN	TP	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R17	TP	TP	TP	FN	FN	FN	FN	FN	TP	FN	40%	60%
R18	FN	FN	FN	FN	FN	FN	FN	TP	FN	FN	10%	90%
R19	FN	FN	FN	TP	FN	FN	FN	FN	FN	FN	10%	90%
R20	FN	FN	TP	FN	FN	TP	FN	FN	FN	FN	20%	80%
R21	FN	TP	TP	FN	FN	FN	FN	FN	FN	FN	20%	80%
R22	FN	FN	FN	FN	FN	FN	FN	FN	FN	FN	0%	100%
R23	FN	FN	TP	FN	FN	FN	FN	FN	FN	FN	10%	90%
R24	FN	FN	FN	FN	FN	FN	FN	FN	FN	FN	0%	100%
R25	TP	TP	TP	TP	FN	FN	TP	FN	TP	TP	70%	30%
R26	FN	TP	TP	FN	FN	FN	FN	FN	TP	TP	40%	60%
R27	TP	TP	TP	FN	FN	FN	TP	FN	TP	TP	60%	40%
R28	TP	FN	TP	FN	TP	TP	FN	TP	TP	FN	60%	40%
R29	FN	FN	TP	TP	TP	FN	TP	TP	TP	TP	70%	30%
R30	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R31	FN	TP	TP	FN	FN	FN	TP	FN	FN	FN	30%	70%
R32	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R33	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R34	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R35	FN	FN	TP	TP	TP	TP	TP	FN	TP	TP	70%	30%
R36	TP	FN	TP	FN	TP	TP	TP	TP	TP	TP	80%	20%
R37	FN	TP	TP	TP	FN	FN	TP	FN	TP	FN	50%	50%
R38	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R39	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R40	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R41	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R42	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R43	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R44	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R45	TP	FN	TP	TP	TP	TP	TP	FN	TP	TP	80%	20%
R46	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R47	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R48	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R49	TP	TP	TP	TP	TP	FN	TP	TP	FN	TP	80%	20%
R50	TP	TP	TP	TP	TP	FN	TP	TP	FN	TP	80%	20%
R51	TP	TP	FN	FN	TP	FN	FN	TP	FN	TP	50%	50%
R52	TP	TP	TP	TP	TP	FN	TP	TP	TP	TP	90%	10%
R53	TP	FN	TP	TP	FN	TP	TP	TP	TP	TP	80%	20%
R54	TP	FN	TP	FN	FN	FN	TP	FN	FN	TP	40%	60%
R55	TP	FN	FN	FN	FN	FN	TP	TP	TP	FN	40%	60%
R56	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R57	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%

Ref	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	% of TP	% of FN
R58	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R59	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R60	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R61	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R62	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R63	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
R64	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R65	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
R66	FN	FN	FN	FN	FN	FN	FN	FN	FN	FN	0%	100%
R67	TP	FN	TP	TP	TP	TP	TP	FN	TP	TP	80%	20%
R68	FN	FN	FN	FN	FN	FN	TP	FN	TP	TP	30%	70%
R69	FN	FN	TP	TP	FN	FN	TP	FN	TP	TP	50%	50%
R70	FN	FN	TP	TP	TP	TP	TP	TP	TP	TP	80%	20%
R71	FN	FN	FN	FN	FN	FN	TP	TP	FN	TP	30%	70%
R72	FN	TP	TP	TP	FN	FN	TP	TP	FN	TP	60%	40%
R73	FN	TP	TP	FN	FN	FN	TP	FN	TP	TP	50%	50%
R74	TP	TP	TP	TP	FN	FN	TP	FN	FN	TP	60%	40%
R75	FN	TP	FN	FN	FN	FN	TP	FN	TP	TP	40%	60%
R76	FN	TP	TP	FN	FN	FN	TP	TP	TP	FN	50%	50%
R77	TP	TP	TP	FN	FN	FN	TP	FN	TP	TP	60%	40%
R78	FN	FN	TP	TP	FN	FN	TP	FN	TP	TP	50%	50%
R79	FN	FN	TP	TP	FN	FN	TP	FN	TP	TP	50%	50%
R80	FN	FN	TP	TP	FN	FN	TP	FN	TP	TP	50%	50%
R81	FN	FN	TP	TP	FN	FN	TP	FN	TP	TP	50%	50%
R82	FN	FN	TP	TP	FN	FN	TP	FN	TP	TP	50%	50%
R83	FN	FN	TP	TP	FN	FN	TP	FN	TP	TP	50%	50%
R84	FN	TP	TP	TP	FN	TP	TP	TP	TP	TP	80%	20%
R85	TP	FN	FN	TP	FN	FN	FN	FN	TP	FN	30%	70%
R86	FN	TP	TP	TP	FN	FN	TP	FN	TP	FN	50%	50%
R87	TP	TP	TP	TP	FN	FN	FN	FN	TP	FN	50%	50%
R88	FN	TP	TP	TP	FN	FN	TP	FN	TP	FN	50%	50%
R89	FN	FN	TP	TP	FN	FN	TP	FN	TP	TP	50%	50%
R90	TP	FN	TP	TP	TP	TP	FN	TP	TP	FN	70%	30%
R91	TP	FN	TP	TP	TP	FN	FN	TP	TP	FN	60%	40%
R92	TP	FN	TP	TP	TP	FN	TP	TP	TP	TP	80%	20%
R93	TP	TP	TP	TP	TP	FN	TP	TP	TP	FN	80%	20%
R94	TP	TP	TP	TP	TP	FN	TP	TP	TP	TP	90%	10%
R95	FN	FN	TP	TP	FN	FN	TP	FN	TP	FN	40%	60%
R96	FN	FN	TP	TP	FN	FN	TP	FN	TP	FN	40%	60%
R97	FN	FN	TP	TP	FN	FN	TP	FN	TP	FN	40%	60%
R98	FN	FN	TP	TP	FN	FN	TP	FN	TP	FN	40%	60%
R99	TP	FN	FN	TP	FN	FN	TP	TP	TP	FN	50%	50%
R100	FN	FN	FN	FN	FN	FN	TP	FN	TP	FN	20%	80%

The results in Table N have the following indicators:

- ‘TN’– true negative, deepfake images correctly predicted as fake images
- ‘FP’– false positive, deepfake images incorrectly predicted as real images

Refer to Appendix 6 for the deepfake images for testing.

Table N: Predictions on fake images

Ref	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	% of TN	% of FP
F1	TN	FN	TN	TN	TN	TN	TN	TN	TN	TN	90%	10%
F2	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F3	FN	TN	TN	FN	TN	TN	TN	TN	FN	TN	70%	30%
F4	TN	FN	TN	TN	TN	TN	TN	TN	TN	TN	90%	10%
F5	TN	TN	FN	TN	TN	TN	FN	TN	TN	TN	80%	20%
F6	FN	TN	TN	TN	FN	TN	TN	TN	FN	TN	70%	30%
F7	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F8	TN	TN	TN	TN	TN	TN	TN	TN	FN	FN	80%	20%
F9	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F10	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F11	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%

Ref	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	% of TN	% of FP
F12	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F13	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F14	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F15	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F16	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F17	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F18	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F19	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F20	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F21	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F22	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F23	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F24	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F25	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F26	TN	TN	TN	TN	FN	TN	TN	TN	TN	TN	90%	10%
F27	TN	TN	FN	FN	TN	TN	TN	TN	FN	TN	70%	30%
F28	TN	TN	FN	FN	FN	TN	TN	TN	FN	TN	60%	40%
F29	TN	TN	TN	TN	TN	TN	TN	TN	FN	TN	90%	10%
F30	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F31	FN	FN	FN	TN	TN	TN	TN	TN	TN	FN	60%	40%
F32	TN	TN	TN	TN	TN	TN	TN	TN	TN	FN	90%	10%
F33	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F34	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F35	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F36	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F37	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F38	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F39	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F40	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F41	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F42	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F43	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F44	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F45	TN	TN	TN	TN	TN	TN	TN	TN	FN	TN	90%	10%
F46	TN	TN	TN	TN	TN	TN	TN	TN	TN	FN	90%	10%
F47	TN	TN	TN	TN	TN	FN	TN	TN	TN	FN	80%	20%
F48	FN	TN	TN	TN	TN	TN	TN	TN	TN	TN	90%	10%
F49	FN	TN	TN	FN	TN	TN	TN	TN	TN	TN	80%	20%
F50	FN	TN	FN	FN	TN	TN	TN	FN	TN	TN	60%	40%
F51	FN	TN	FN	FN	TN	TN	TN	FN	TN	TN	60%	40%
F52	FN	TN	FN	FN	TN	FN	TN	TN	TN	TN	60%	40%
F53	FN	FN	FN	FN	TN	FN	TN	TN	TN	TN	50%	50%
F54	FN	TN	FN	TN	TN	FN	TN	TN	TN	TN	70%	30%
F55	FN	FN	FN	FN	TN	TN	TN	TN	TN	TN	60%	40%
F56	FN	FN	TN	TN	TN	TN	TN	TN	TN	TN	80%	20%
F57	FN	FN	TN	TN	TN	TN	TN	TN	TN	TN	80%	20%
F58	TN	TN	TN	TN	TN	FN	TN	TN	TN	TN	90%	10%
F59	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F60	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F61	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F62	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F63	TN	FN	TN	TN	TN	TN	TN	TN	TN	TN	90%	10%
F64	TN	FN	TN	TN	TN	TN	TN	TN	TN	TN	90%	10%
F65	TN	FN	TN	TN	TN	TN	TN	TN	TN	TN	90%	10%
F66	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F67	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F68	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F69	TN	FN	TN	TN	TN	TN	TN	TN	TN	TN	90%	10%
F70	FN	FN	TN	TN	TN	FN	FN	FN	TN	TN	50%	50%
F71	TN	FN	FN	TN	FN	TN	TN	TN	FN	FN	50%	50%
F72	TN	TN	FN	FN	TN	TN	TN	TN	TN	TN	80%	20%
F73	FN	FN	TN	TN	TN	FN	TN	TN	FN	FN	50%	50%
F74	FN	FN	TN	FN	TN	TN	FN	FN	FN	FN	30%	70%
F75	TN	FN	TN	TN	TN	FN	TN	TN	FN	TN	70%	30%
F76	TN	TN	TN	TN	TN	TN	FN	TN	TN	TN	90%	10%
F77	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F78	TN	TN	TN	TN	TN	TN	FN	TN	TN	TN	90%	10%
F79	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F80	TN	TN	TN	TN	TN	TN	TN	TN	FN	TN	90%	10%

Ref	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	% of TN	% of FP
F81	TN	FN	FN	TN	TN	TN	TN	TN	FN	TN	70%	30%
F82	TN	TN	TN	TN	TN	TN	TN	TN	FN	TN	90%	10%
F83	TN	TN	TN	TN	TN	TN	TN	TN	FN	TN	90%	10%
F84	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F85	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F86	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F87	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F88	TN	TN	TN	TN	TN	TN	FN	TN	TN	TN	90%	10%
F89	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F90	TN	FN	TN	TN	TN	TN	TN	TN	TN	TN	90%	10%
F91	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F92	TN	TN	TN	TN	TN	FN	FN	FN	TN	TN	70%	30%
F93	TN	TN	FN	TN	TN	FN	FN	FN	FN	TN	50%	50%
F94	TN	TN	TN	TN	TN	TN	TN	TN	TN	TN	100%	0%
F95	TN	FN	TN	TN	TN	TN	FN	FN	FN	TN	60%	40%
F96	TN	TN	TN	TN	TN	FN	TN	TN	FN	TN	80%	20%
F97	TN	TN	TN	FN	TN	TN	TN	TN	FN	TN	80%	20%
F98	TN	TN	TN	TN	TN	TN	TN	TN	FN	TN	90%	10%
F99	TN	TN	TN	FN	TN	TN	FN	TN	FN	TN	70%	30%
F100	FN	TN	TN	FN	TN	TN	TN	TN	FN	TN	70%	30%

Further summarise the number of TP, TN, FP (Type 1 error) and FN (Type 2 error) for each model in Table O.

Table O: Summary of result for models

Model Ref	Model	TP	TN	FP	FN	Grand Total
M1	Xception	60	83	17	40	**200**
M2	VGG19	47	81	19	53	**200**
M3	ResNet152V2	86	86	14	14	**200**
M4	InceptionV3	74	86	14	26	**200**
M5	InceptionResNetV2	50	96	4	50	**200**
M6	MobileNetV2	46	89	11	54	**200**
M7	DenseNet201	82	90	10	18	**200**
M8	NASNetMobile	58	93	7	42	**200**
M9	EfficientNetB3	82	78	22	18	**200**
M10	EfficientNetB7	67	92	8	33	**200**
	Grand Total	**652 (33%)**	**874 (44%)**	**126 (6%)**	**348 (17%)**	**2000 (100%)**

4.1.1 Compare Performance Metrics using Test Dataset

To compare the models' ability to properly classification on the test dataset. Calculate several metrics in Table P, including

- **Accuracy** – proportion of the correct predictions of real and deepfake images: (TP + TN) / (TP + TN + FP + FN)
- **Recall or True Positive Rate (TPR)** – the proportion of the actual real images was identified correctly as real images: TP / (TP + FN)
- **Specificity or True Negative Rate (TNR)** – the proportion of the actual deepfake images was identified as deepfake images: TN / (TN + FP)
- **Precision or Positive Predictive Value (PPV)** – the proportion of positive identifications (i.e. identified as reals) was actually real images: TP / (TP + FP)
- **Negative Predictive Value (N.P.V.)** – the proportion of negative identifications (i.e. identified as deepfakes) was actually deepfake images: TN / (TN + FN)
- **F1 Score –** the harmonic mean of precision and recall: 2 / [(1 / Recall) + (1 / Precision)]
- **Matthew's correlation coefficient**: [(TP X TN) – (FP X FN)] / [((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5] where a value near 1 means a perfect classifier.

Table P: Performance metrics for models (Test Dataset)

Model Ref	Model	Accuracy	Recall	Specificity	Precision	Negative Predictive Value	F1 Score	Matthew's correlation coefficient
M1	Xception	0.7150	0.6000	0.8300	0.7790	0.6750	0.6780	0.4420
M2	VGG19	0.6400	0.4700	0.8100	0.7120	0.6040	0.5660	0.2980
M3	ResNet152V2	0.8600	0.8600	0.8600	0.8600	0.8600	0.8600	0.7200
M4	InceptionV3	0.8000	0.7400	0.8600	0.8410	0.7680	0.7870	0.6040
M5	InceptionResNetV2	0.7300	0.5000	0.9600	0.9260	0.6580	0.6490	0.5180
M6	MobileNetV2	0.6750	0.4600	0.8900	0.8070	0.6220	0.5860	0.3880
M7	DenseNet201	0.8600	0.8200	0.9000	0.8910	0.8330	0.8540	0.7220
M8	NASNetMobile	0.7550	0.5800	0.9300	0.8920	0.6890	0.7030	0.5440
M9	EfficientNetB3	0.8000	0.8200	0.7800	0.7880	0.8120	0.8040	0.6000
M10	EfficientNetB7	0.7950	0.6700	0.9200	0.8930	0.7360	0.7660	0.6090

Note: Highlight the best cases in orange and worst cases in grey.

If the deepfake detector misclassifies reals as deepfakes, it may cause annoyance and frustration to users. On the other hand, this detector tends to misclassify deepfakes as reals may cause serious security threats to users. Assuming that the costs of misclassifying reals as deepfakes versus the costs of misclassifying deepfakes as reals are the same, it is necessary to strike a balance between TPR and TNR.

ResNet152V2 (Model Ref: M3) is the best model with the highest accuracy, recall, negative predictive value and F1 score. Moreover, **DenseNet201 (Model Ref: M7)** also provides similar results as ResNet152V2. Although **InceptionResNetV2 (Model Ref: M5)** obtained the highest specificity and high precision, it only obtained 0.5 for recall. The worst model is **VGG19 (Model Ref: M2)**, likely caused by the lowest trainable parameters, as shown in Table H.

ResNet152V2 (Model Ref: M3) seems to be performing well on the test dataset. Conduct further testing on all these models using real and deepfake videos in the wild, and please refer to Section 4.2 for assessment.

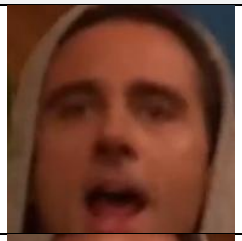
4.1.2 Review Test Dataset at Image Level

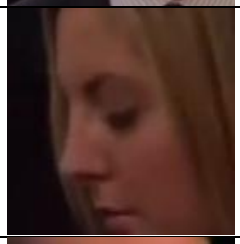
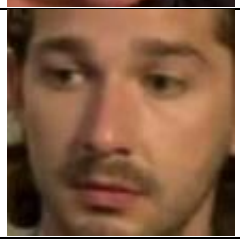
This project also looks at the misclassified images to see if we could find out the reasons for misclassification.

Real images

In the last column of Table M, 9 out of 100 misclassified images by eight or more detectors. Visualise these images in Table Q below.

Table Q: Real images being misclassified as deepfakes by ≥80% detectors

Ref	File Name	Source	% of FN	Misclassified by ResNet152V2?	Image	Image Property
R18	43_192.jpg	Deepfake Database [31]	90%	Yes		188 X 188 96 dpi
R19	43_210.jpg		90%	Yes		182 X 182 96 dpi
R20	44_6.jpg		80%	No		224 X 224 96 dpi
R21	44_90.jpg		80%	No		204 X 204 96 dpi
R22	44_192.jpg		100%	Yes		222 X 222 96 dpi

Ref	File Name	Source	% of FN	Misclassified by ResNet152V2?	Image	Image Property
R23	44_198.jpg		90%	No		240 X 240 96 dpi
R24	45_60.jpg		100%	Yes		134 X 134 96 dpi
R66	102_320.jpg		100%	Yes		148 X 148 96 dpi
R100	id25_0004.mp4 312.jpgFace.jpg	Celeb-DF (version 2) [31]	80%	Yes		107 X 107 96 dpi

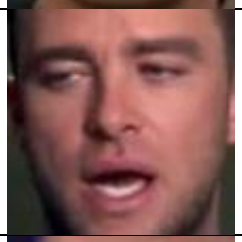
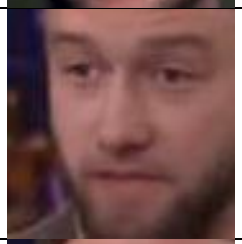
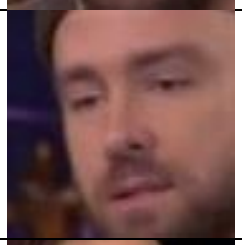
Compared with the authentic images being correctly classified, the image size ranges similar to the misclassified images above. Thus, the images' size is not likely to be the root cause for the misclassification.

In addition, pictures are relatively blurry for R18, R19 and R66 compared to other images. However, for the remaining six misclassified authentic images, there are no apparent reasons for misclassifications.

Fake images

Misclassification of deepfakes as reals is much less than reals' misclassified as deepfakes. For example, in the last column of Table N, 6 out of 100 deepfake images are being misclassified as reals by five or more detectors. Visualise these images in Table R below.

Table R: Fake images being misclassified as reals by ≥50% detectors

Ref	File Name	Source	% of FN	Misclassified by ResNet152V2?	Image	Image Property
F53	143_41.jpg	Deepfake Database [31]	50%	Yes		504 X 504 96 dpi
F70	cele17750.jpgF.jpg	Celeb-DF (version 2) [31]	50%	No		154 X 155 96 dpi
F71	cele17783.jpgF.jpg		50%	Yes		108 X 107 96 dpi
F73	cele18514.jpgF.jpg		50%	No		89 X 90 96 dpi
F74	cele18909.jpgF.jpg		70%	No		90 X 90 96 dpi
F93	cele35320.jpgF.jpg		50%	Yes		129 X 129 96 dpi

Similar to the previous reason, for F70, F73 and F74, these pictures are relatively blurry compared to other images. However, there are no apparent reasons for misclassifications for the remaining three misclassified deepfake images.

4.2 Model Review (Videos in the Wild)

A good model must correctly classify both real and deepfake with high accuracy. This project further tested all the models using five real and five incredibly realistic deepfake videos from YouTube. First, 10 seconds of the videos are used and then preprocessed using the same method as demonstrated previously. Then, after face cropping from video frames, there are 14 authentic faces and 12 images of deepfake faces.

4.2.1 Compare Performance Metrics using Wild Videos

Summarise the results of images in Table S. The model reference ('M1' to 'M10') refers to the model name shown in Table L. Refer to Appendix 7 for the images for final testing. The results in Table S have the following indicators:

- 'TP'– true positive, real images correctly predicted as real images
- 'FN'– false negative, real images incorrectly predicted as deepfake images
- 'TN'– true negative, deepfake images correctly predicted as fake images
- 'FP'– false positive, deepfake images incorrectly predicted as real images

Table S: Predictions on images captured in the videos in the YouTube

Ref	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	% of Correctly Classified	% of Incorrectly Classified
Real												
WR1	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
WR2	FN	FN	TP	FN	FN	FN	TP	TP	TP	TP	50%	50%
WR3	TP	FN	TP	TP	FN	FN	TP	TP	TP	TP	70%	30%
WR4	TP	TP	TP	TP	FN	TP	TP	TP	TP	TP	90%	10%
WR5	TP	TP	TP	TP	FN	TP	TP	TP	TP	TP	90%	10%
WR6	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
WR7	TP	TP	TP	TP	TP	FN	TP	TP	TP	FN	80%	20%
WR8	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
WR9	TP	TP	TP	TP	TP	TP	TP	TP	TP	FN	90%	10%
WR10	TP	TP	TP	TP	TP	FN	TP	TP	TP	TP	90%	10%
WR11	TP	TP	TP	TP	TP	FN	TP	TP	TP	TP	90%	10%
WR12	TP	FN	TP	TP	TP	TP	TP	TP	TP	TP	90%	10%
WR13	FN	FN	TP	TP	TP	TP	TP	TP	TP	TP	80%	20%
WR14	TP	TP	TP	TP	TP	TP	TP	TP	TP	TP	100%	0%
Fake												
WF1	FP	TN	FP	FP	TN	TN	FP	FP	FP	FP	30%	70%
WF2	TN	FP	TN	FP	TN	FP	FP	TN	FP	TN	50%	50%
WF3	TN	TN	FP	TN	TN	TN	FP	FP	FP	TN	60%	40%
WF4	TN	FP	TN	FP	TN	FP	FP	TN	FP	TN	50%	50%
WF5	TN	FP	FP	FP	FP	FP	FP	TN	FP	FP	20%	80%
WF6	FP	FP	FP	FP	FP	TN	FP	FP	FP	FP	10%	90%
WF7	FP	FP	FP	TN	FP	TN	FP	FP	FP	FP	20%	80%
WF8	TN	TN	FP	TN	TN	TN	FP	TN	TN	TN	80%	20%
WF9	TN	TN	FP	TN	TN	TN	FP	FP	TN	FP	60%	40%
WF10	FP	FP	FP	FP	FP	FP	FP	FP	FP	TN	10%	90%
WF11	FP	TN	FP	FP	FP	TN	FP	FP	FP	TN	30%	70%
WF12	FP	FP	FP	FP	FP	TN	FP	TN	FP	FP	20%	80%

Table T shows the performance metrics of prediction on videos on YouTube.

Table T: Performance metrics for models (YouTube videos)

Model Ref	Model	Accuracy	Recall	Specificity	Precision	Negative Predictive Value	F1 Score	Matthew's correlation coefficient
M1	Xception	0.692	0.857	0.500	0.667	0.750	0.750	0.386
M2	VGG19	0.538	0.643	0.417	0.562	0.500	0.600	0.061
M3	ResNet152V2	0.615	1.000	0.167	0.583	1.000	0.737	0.312
M4	InceptionV3	0.654	0.929	0.333	0.619	0.800	0.743	0.331
M5	InceptionResNetV2	0.615	0.714	0.500	0.625	0.600	0.667	0.220
M6	MobileNetV2	0.654	0.643	0.667	0.692	0.615	0.667	0.309
M7	DenseNet201	0.538	1.000	0.000	0.538	0.000	0.700	0.000
M8	NASNetMobile	0.731	1.000	0.417	0.667	1.000	0.800	0.527
M9	EfficientNetB3	0.615	1.000	0.167	0.583	1.000	0.737	0.312
M10	EfficientNetB7	0.692	0.857	0.500	0.667	0.750	0.750	0.386

Note: Highlight the best cases in orange and worst cases in grey.

After verifying all the models with the ten videos from YouTube, it is interesting that the "best" model previously identified, **ResNet152V2 (Model Ref: M3)**, did not perform well in this final test. Although this model correctly classified all the real videos as reals, it cannot correctly identify deepfake videos, resulting in a low specificity percentage.

DenseNet201 (Model Ref: M7), which performs well in the test dataset, cannot identify authentic and deepfake images. Unexpectedly, it classified all the deepfakes as reals and became the worst model in this testing.

NASNetMobile (Model Ref: M8) shows the best performance with the highest accuracy, recall, Negative Predictive Value, and F1 score among the ten models, but it does not perform well in the test dataset.

Overall, the assessments in the test dataset and the wild videos show conflicting results. Therefore, this project cannot conclude based on the two testings conducted.

4.2.2 Review Wild Videos at Image Level

As performed in the previous section, this project looks at the misclassified images to determine the reasons for misclassification.

Real images

Misclassified reals are much less than misclassified deepfakes. In general, the detectors perform well in authentic images, except for one image being misclassified as a deepfake image by five detectors. For example, visualise the images in Table U below. This particular image has the smallest size (75 pixels by 75 pixels) compared with all the other authentic images in the testing. However, as assessed in the previous testing using the test dataset, the image size might not be the main reason causing misclassifications. Thus, this project cannot draw a solid conclusion on reasons for misclassifications.

Table U: Real images being misclassified as deepfakes by ≥50% detectors

Ref	File Name	Source	% of FN	Image	Image Property	Models Classify As Real Correctly
WR2	BO_1.mp4120 .jpgF.jpg	YouTube	50%		75 X 75 96 dpi	M3, M7, M8, M9, M10

Fake images

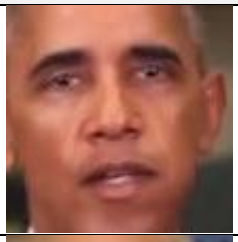
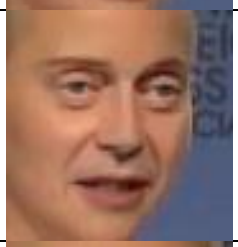
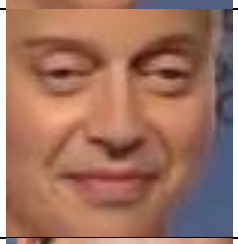
7 out of 12 deepfake images are misclassified as reals by seven or more detectors. Visualise these images in Table V below. Most of these misclassified images are mainly extracted from 2 outstanding deepfake videos.

WF5 – WF7 are extracted from a video posted on YouTube in 2020. The video creator said WF5 – WF7 was trained by faceswap using a custom model two years ago. Although the image size is small and the quality is not very high, there is no awkward boundary on the target's face. Even though it is not tricky for naked eyes to confirm it is not the original face (Jennifer Lawrence), the video shows a natural blending of the original person and the impersonator.

WF10 – WF12 are extracted from a video posted on TikTok in early 2021. It is undoubtedly the most realistic deepfake video as of today. In the interviews with the video creator, he mentioned that the extreme realistic video was produced using Deepfacelab and Machine Editor. He trained his model with a customised dataset, which contains images of Tom Cruise with different expressions and filming angles. Moreover, a professional actor with a highly similar face shape to Tom Cruise was involved in the video filming. He also added additional visual effects to the videos during the creation of the video.

With such a high impersonation level, **MobileNetV2 (Model Ref: M6)**, which could identify four of the six concerned images, seems to provide a good result for more advanced deepfake videos. However, this model only has an average performance on identifying reals and fake videos in the test dataset and the YouTube video in the wild.

Table V: Fake images being misclassified as reals by ≥70% detectors

Ref	File Name	Source	% of FN	Image	Image Property	Models Classify As Deepfake Correctly
WF1	BO_0.mp40 .jpgF.jpg	YouTube	70%		107 X 107 96 dpi	M2, M5, M6
WF5	JL_0.mp40 .jpgF.jpg	YouTube	80%		89 X 90 96 dpi	M1, M8
WF6	JL_0.mp4120 .jpgF.jpg		90%		89 X 90 96 dpi	M6
WF7	JL_0.mp4240 .jpgF.jpg		80%		75 X 75 96 dpi	M4, M6
WF10	TC_0.mp40 .jpgF.jpg	YouTube	90%		129 X 129 96 dpi	M10
WF11	TC_0.mp4120 .jpgF.jpg		70%		108 X 107 96 dpi	M2, M6, M10

Ref	File Name	Source	% of FN	Image	Image Property	Models Classify As Deepfake Correctly
WF12	TC_0.mp4240 .jpgF.jpg		80%		89 X 90 96 dpi	M6, M8

4.3 Overall Test Result

Based on the results in the test dataset with 100 authentic images and 100 fake images, **ResNet152V2 (Model Ref: M3)** provides good predictions on both real videos and deepfake videos from Celeb-DF (version 2) dataset as well as real/fake images from Deepfake Dataset [32]. Furthermore, among the ten models, this model achieved the highest performance in terms of accuracy (86%), recall (86%), negative predictive value (86%) and F1 score (86%). In addition, the model's specificity (86%) and precision (86%) also achieved high values.

Unfortunately, after further testing on several popular deepfake videos downloaded from YouTube, the model above could not effectively distinguish deepfakes. Even though data augmentation is applied during training, inevitably, generalisation issue on unseen data is not effectively prevented. Interestingly, among the ten models, **NASNetMobile (Model Ref: M8)** shows the best performance with the highest accuracy (73%), recall (100%), Negative Predictive Value (100%) as well as F1 score (80%), but it does not perform well in the test dataset.

MobileNetV2 (Model Ref: M6) also perform well in detecting extraordinarily realistic and natural deepfake videos in the wild, especially the deepfake Tom Cruise and deepfake Jennifer Lawrence. Nevertheless, it is difficult to confirm the best detector by testing only a few real and deepfake videos.

Furthermore, by analysing misclassified cases at the image level, it is noted that if the image captured from the video for analysis is blurry, it is more likely for a detector to be misclassified. The image size for testing might also be a reason for misclassification. However, there is no solid evidence to confirm due to contradicting results among the models.

Chapter 5: Conclusion

An effective deepfake detector should efficiently and correctly classify real and deepfakes videos. On the one hand, if the detector could not identify deepfakes (i.e. fake videos being misclassified as real videos), it provides users with a false sense of security, leading to potential security issues. On the other hand, a detector would cause users annoyance and frustration if there are many false alarms (i.e. real videos being misclassified as fake videos).

This project attempted to create a deepfake detector using transfer learning and fine-tuning techniques on models originally pre-trained in ImageNet. The task of distinguishing deepfake and real videos is an image classification task. First, an image database containing around 18,000 images was created. Then, ten binary classifiers of different network architectures are built, trained, validated and tested in Google Colab using customised datasets and videos in the wild.

Overall, the assessments in the test dataset and the wild videos show conflicting results in the ten trained models. There is no single detector that could stand out to be highly effective. Even though this project cannot discover a perfect detector to tackle the issue, it has achieved its original intent and purpose to a large extend.

As a final remark, this project cannot conclude based on merely two testing conducted in the experiment due to limited testing performed. More videos might be needed for better model training and testing. Detecting deepfakes should continuously be a hot topic that researchers should pay attention to. The problem of deepfake is like a cat chasing a naughty and tricky mouse. It would be unrealistic to create a detector that could 100% distinguish real and fake videos without continuous model training. However, training a perfect detector is time-consuming and requires an appropriate dataset. Thus, a more innovative approach might be required.

Appendices

Appendix 1: Bibliography and Reading List

Toews, R. (2020) Deepfakes Are Going To Wreak Havoc On Society. We Are Not Prepared. *Forbes* [online] 25 May. Available from: https://www.forbes.com/sites/robtoews/2020/05/25/deepfakes-are-going-to-wreak-havoc-on-society-we-are-not-prepared [Accessed 25 October 2021].

Nguyen et al. (2021) Deep Learning for Deepfakes Creation and Detection: A Survey. [online]. Version 3. 26 April 2021. Available from: https://arxiv.org/abs/1909.11573 [Accessed 5 November 2021].

Perov et al. (2020) DeepFaceLab: Integrated, flexible and extensible face-swapping framework. [online] Available from: https://github.com/iperov/DeepFaceLab [Accessed 1 November 2021].

Anon. (2021) FaceSwap Github code respositary [online] Available from: https://github.com/deepfakes/faceswap [Accessed 30 October 2021].

Anon. (2019) Zao App official website [online] Available from: https://zaodownload.com [Accessed 30 October 2021].

Anon. (2021) REFACE App official website [online] Available from: https://hey.reface.ai [Accessed 30 October 2021].

Jiang et al. (2020) DeeperForensics-1.0: A Large-Scale Dataset for Real-World Face Forgery Detection. *CVPR 2020: Conference on Computer Vision and Pattern Recognition* [online] virtual conference, 14-20 June 2020. Computer Vision Foundation. Available from: https://openaccess.thecvf.com/content_CVPR_2020/papers/Jiang_DeeperForensics-1.0_A_Large-Scale_Dataset_for_Real-World_Face_Forgery_Detection_CVPR_2020_paper.pdf [Accessed 5 November 2021].

Raja et al. (2021) Morphing Attack Detection-Database, Evaluation Platform, and Benchmarking. *IEEE Transactions on Information Forensics and Security* [online] 16, pp. 4336-4351[Accessed 5 November 2021].

L. Zhang, F. Peng and M. Long (2018) Face Morphing Detection Using Fourier Spectrum of Sensor Pattern Noise. I*CME: 2018 IEEE International Conference on Multimedia and Expo* [online] San Diego, 23-27 July 2018. IEEE Computer Society. Available from: https://ieeexplore.ieee.org/document/8486607 [Accessed 5 November 2021].

Li et al. (2020) Face X-Ray for More General Face Forgery Detection. *CVPR 2020: Conference on Computer Vision and Pattern Recognition* [online] virtual conference, 14-20 June 2020. Computer Vision Foundation. Available from: Available from: https://ieeexplore.ieee.org/document/9157215 [Accessed 5 November 2021].

Nimmo et al. (2020) Facebook Takes Down Small, Recently Created Network Linked to Internet Research Agency [online]. Graphika. Available from: https://public-assets.graphika.com/reports/graphika_report_ira_again_unlucky_thirteen.pdf [Accessed 30 October 2021].

Nimmo et al. (2020) Facebook Takes Down Inauthentic Chinese Network [online] Graphika. Available from: https://graphika.com/reports/operation-naval-gazing/ [Accessed 30 October 2021].

Nimmo et al. (2020) Pro-Chinese Inauthentic Network Debuts English-Language Videos [online] Graphika. Available from: https://graphika.com/reports/spamouflage-dragon-goes-to-america/ [Accessed 30 October 2021].

B.B.C. (2019) How to Obama / Jordan Peele DEEPFAKE actually works | Ian Hislop's Fake News. *YouTube* [video] 16 October. Available from: https://youtu.be/g5wLaJYBAm4 [Accessed 30 October 2021].

Washington Post (2019) Mark Zuckerberg' deepfake' will remain online. *YouTube* [video] 18 June. Available from: https://youtu.be/NbedWhzx1rs [Accessed 30 October 2021].

CNN (2021) No, Tom Cruise isn't on TikTok. It's a deepfake. *CNN Business* [video] 2 March. Available from: https://edition.cnn.com/videos/business/2021/03/02/tom-cruise-tiktok-deepfake-orig.cnn-business/video/playlists/business-social-media/ [Accessed 30 October 2021].

A.B.C. News (2021) Deepfake videos are becoming easier to make but dangerously difficult to identify | Nightline. *YouTube* [video] 19 March. Available from: https://youtu.be/70l8LzBpGWs [Accessed 30 October 2021].

Gralla, P. (2019) Here's How Deepfakes Can Harm Your Enterprise — and What to Do About Them [online] *Symantec Enterprise Blogs / Feature Stories* [blog]. 23 September. Available from: https://symantec-enterprise-blogs.security.com/blogs/feature-stories/heres-how-deepfakes-can-harm-your-enterprise-and-what-do-about-them [Accessed 1 November 2021].

Coleman, A. (2019) 'Deepfake' app causes fraud and privacy fears in China. *BBC News* [online] 4 September. Available from: https://www.bbc.co.uk/news/technology-49570418 [Accessed 29 October 2021].

Branscombe, M. (2021) Deepfakes: Microsoft and others in big tech are working to bring authenticity to videos, photos. *TechRepublic* [online] 26 July. Available from: https://www.techrepublic.com/article/deepfakes-microsoft-and-others-in-big-tech-are-working-to-bring-authenticity-to-videos-photos/ [Accessed 2 November 2021].

Swathi, P. and SK, Saritha. (2021) DeepFake Creation and Detection: A Survey. *ICIRCA: 2021 Third International Conference on Inventive Research in Computing Applications* [online] R.V.S. College of Engineering and Technology, Tamilnadu, 2-4 September 2021. Available from: https://ieeexplore.ieee.org/document/9544522 [Accessed 9 July 2021].

Appendix 2: Reference

Siarohin et al. (2019) First Order Motion Model for Image Animation. *NeurIPS 2019: Advances in Neural Information Processing Systems 32* [online] Available from: https://papers.nips.cc/paper/2019/hash/31c0b36aef265d9221af80872ceb62f9-Abstract.html [Accessed 30 October 2021].

Karras et al. (2019) A Style-Based Generator Architecture for Generative Adversarial Networks. [online] Available from: https://github.com/NVlabs/stylegan [Accessed 30 October 2021].

Toews, R. (2020) Deepfakes Are Going To Wreak Havoc On Society. We Are Not Prepared. Forbes [online] 25 May. Available from: https://www.forbes.com/sites/robtoews/2020/05/25/deepfakes-are-going-to-wreak-havoc-on-society-we-are-not-prepared/?sh=1505aaff7494 [Accessed 1 November 2021].

Wakefield, J. (2021) MP Maria Miller wants A.I. 'nudifying' tool banned. *BBC News* [online] 4 August. Available from: https://www.bbc.co.uk/news/technology-57996910 [Accessed 30 October 2021].

Federal Bureau of Investigation, Internet Crime Complaint Center (2021) *Malicious actors almost certainly will leverage synthetic content for cyber and foreign influence operations* [online]. Available from: https://www.ic3.gov/media/news/2021/210310-2.pdf [Accessed 30 October 2021].

Trend Micro, United Nations Interregional Crime and Justice Research Institute (UNICRI), and Europol (2020) *Exploiting Artificial intelligence: How Cybercriminals Misuse and Abuse Artificial intelligence and Machine Learning.* Available from: https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/exploiting-ai-how-cybercriminals-misuse-abuse-ai-and-ml [Accessed 26 October 2021].

Stupp, C. (2019) Fraudsters Used A.I. to Mimic C.E.O.'s Voice in Unusual Cybercrime Case. *The Wall Street Journal* [online] 30 August. Available from: https://www.wsj.com/articles/fraudsters-use-ai-to-mimic-ceos-voice-in-unusual-cybercrime-case-11567157402 [Accessed 30 October 2021].

Brewster, T. (2021) Fraudsters Cloned Company Director's Voice In $35 Million Bank Heist, Police Find. *Forbes* [online] 14 October. Available from: https://www.forbes.com/sites/thomasbrewster/2021/10/14/huge-bank-fraud-uses-deep-fake-voice-tech-to-steal-millions/?sh=1a976ab37559 [Accessed 30 October 2021].

Wiggers, K. (2020) Voice cloning experts cover crime, positive use cases, and safeguards. *VentureBeat* [online] 29 January. Available from: https://venturebeat.com/2020/01/29/ftc-voice-cloning-seminar-crime-use-cases-safeguards-ai-machine-learning/ [Accessed 29 October 2021].

Palmai, K. (2021) Voice cloning of growing interest to actors and cybercriminals. *BBC News* [online] 12 July. Available from: https://www.bbc.co.uk/news/business-57761873 [Accessed 29 October 2021].

Tolosana et al. (2020) Deepfakes and beyond: A Survey of face manipulation and fake detection. *Journal of Information Fusion* [online]. 64, pp.131-148. Available from: https://www.sciencedirect.com/science/article/abs/pii/S1566253520303110 [Accessed 9 July 2021].

Mirkey, Y. and Lee, M. (2021) The Creation and Detection of Deepfakes: A Survey. *A.C.M. Computing Surveys* [online] 54(1), pp.1-41. Available from: https://dl.acm.org/doi/10.1145/3425780 [Accessed 9 July 2021].

Ajder et al. (2019) *The State of Deepfakes: Landscape, Threats and Impact* [online] Deeptrace Lab. Available from: https://regmedia.co.uk/2019/10/08/deepfake_report.pdf [Accessed 29 October 2021].

Sensity AI (2021) How to Detect a Deepfake Online. *Deepfake Detection* [blog]. 2 August. Available from: https://sensity.ai/blog/deepfake-detection/how-to-detect-a-deepfake [Accessed 25 October 2021].

Pu (2020) NoiseScope: Detecting Deepfake Images in a Blind Setting. *ACSAC 20: Annual Computer Security Applications Conference.* [online] Available from: https://dl.acm.org/doi/10.1145/3427228.3427285 [Accessed 25 October 2021].

Yu et al. (2021) A Survey on Deepfake Video Detection. *The Institution of Engineering and Technology Biometrics* [online] Available from: https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/bme2.12031 [Accessed 12 July 2021].

Neves et al. (2020) GANprintR: improved fakes and evaluation of the state-of-the-art in face manipulation detection. *IEEE Journal of Selected Topics in Signal Processing* 14 (5), pp.1038-1048. [online] [Accessed 5 November 2021].

Yu et al. (2019) Attributing Fake Images to GANs: Learning and Analyzing GAN Fingerprints. *ICCV 2019: International Conference on Computer Vision* [online] COEX Convention Center, Seoul, 27 October – 2 November 2019. Computer Vision Foundation. Available from: https://openaccess.thecvf.com/content_ICCV_2019/papers/Yu_Attributing_Fake_Images_to_GANs_Learning_and_Analyzing_GAN_Fingerprints_ICCV_2019_paper.pdf [Accessed 5 November 2021].

Dang et al. (2020) On the Detection of Digital Face Manipulation. *CVPR 2020: Conference on Computer Vision and Pattern Recognition* [online] virtual conference, 14-20 June 2020. Computer Vision Foundation. Available from: https://openaccess.thecvf.com/content_CVPR_2020/papers/Dang_On_the_Detection_of_Digital_Face_Manipulation_CVPR_2020_paper.pdf [Accessed 5 November 2021].

Rossler et al. (2019) FaceForensics++: Learning to Detect Manipulated Facial Images. *ICCV 2019: International Conference on Computer Vision* [online] COEX Convention Center, Seoul, 27 October – 2 November 2019. Computer Vision Foundation. Available from: https://openaccess.thecvf.com/content_ICCV_2019/papers/Rossler_FaceForensics_Learning_to_Detect_Manipulated_Facial_Images_ICCV_2019_paper.pdf [Accessed 5 November 2021].

Afchar et al. (2018) MesoNet: a compact facial video forgery detection network. *2018 IEEE International Workshop on Information Forensics and Security (WIFS)* [online] Hong Kong, 11-13 December 2018. IEEE Computer Society. Available from: https://ieeexplore.ieee.org/document/8630761 [Accessed 5 November 2021].

Wang, Y. and Dantcheva, A. (2020) A video is worth more than 1000 lies. Comparing 3DCNN approaches for detecting deepfakes. *FG2020: 2020 15th IEEE International Conference on Automatic Face and Gesture Recognition* [online] Buenos Aires, 16-20 November 2020. IEEE Computer Society. Available from: https://ieeexplore.ieee.org/xpl/conhome/9320148/proceeding [Accessed 5 November 2021].

Tarasiou, M. and Zafeiriou, S. (2020) Extracting Deep Local Features to Detect Manipulated Images of Human Faces *ICIP: 2020 IEEE International Conference on Image Processing* [online] United Arab Emirates, 25-28 October 2020. IEEE Computer Society. Available from: https://ieeexplore.ieee.org/document/9190714 [Accessed 5 November 2021].

Bonettini et al. (2021) – Video Face Manipulation Detection Through Ensemble of CNNs. ICPR 2020: 25th International Conference on Pattern Recognition [online] Milano, 10-15 January 2021. IEEE Computer Society. Available from: https://ieeexplore.ieee.org/document/9412711 [Accessed 5 November 2021].

Ciftci et al. (2020) FakeCatcher: Detection of Synthetic Portrait Videos using Biological Signals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Available from: https://ieeexplore.ieee.org/document/9141516 [Accessed 5 November 2021].

Li et al. (2020) Celeb-DF: A Large-scale Challenging Dataset for DeepFake Forensics. *CVPR 2020: Conference on Computer Vision and Pattern Recognition* [online] virtual conference, 14-20 June 2020. Computer Vision Foundation. Available from: https://openaccess.thecvf.com/content_CVPR_2020/papers/Li_Celeb-DF_A_Large-Scale_Challenging_Dataset_for_DeepFake_Forensics_CVPR_2020_paper.pdf [Accessed 5 November 2021].

Kaggle (2019) Deepfake Detection Challenge: Identify videos with facial or voice manipulations. [online] Available from: https://www.kaggle.com/c/deepfake-detection-challenge/data [Accessed 25 October 2021].

Anon. (2021) Convert CSV to JSON [online] Available from: http://www.convertcsv.com/csv-to-json.htm [Accessed 25 October 2021].

Keras (2021) Keras Applications [online] Available from: https://keras.io/api/applications/ [Accessed 25 October 2021].

VICE (2021) The Person Behind the Viral Tom Cruise Deepfake | Super Users. *YouTube* [video] 1 June. Available from: https://www.youtube.com/watch?v=p7-B8S734T4 [Accessed 1 December 2021].

YouTube (2021) President Barack Obama's Keynote Speech | Dear Earth. *YouTube* [video] 1 November. Available from: https://www.youtube.com/watch?v=iaxqTVNHFB0 [Accessed 1 December 2021].

Reuters (2021) Maybe better if Biden infrastructure bill 'doesn't pass,' says Musk. *YouTube* [video] 7 December. Available from: https://www.youtube.com/watch?v=Pxc9e5Cct9A [Accessed 8 December 2021].

The Upcoming (2021) Jennifer Lawrence Don't Look Up interview at premiere. *YouTube* [video] 6 December. Available from: https://www.youtube.com/watch?v=Dfpkw9LTLjY [Accessed 8 December 2021].

The Royal Family (2021) The Queen's speech at the COP26 Evening Reception. *YouTube* [video] 10 November. Available from: https://www.youtube.com/watch?v=eXvfqUe4EFQ [Accessed 8 December 2021].

BBC (2021) Tom Cruise on the INTENSE pressure of flying in Top Gun! *YouTube* [video] 1 May. Available from: https://www.youtube.com/watch?v=MvsO9bNmClw [Accessed 8 December 2021].

BuzzFeedVideo (2019) You Won't Believe What Obama Says In This Video! *YouTube* [video] 1 May. Available from: https://www.youtube.com/watch?v=cQ54GDm1eL0 [Accessed 8 December 2021].

Deepfakery (2019) Elon Musk "I Am Iron Man" (Deepfake) *YouTube* [video] 20 September. Available from: https://www.youtube.com/watch?v=TrDV7dKjJao [Accessed 8 December 2021].

birbfakes (2019) Jennifer Lawrence-Buscemi on her favorite housewives [Deepfake] *YouTube* [video] 10 December. Available from: https://www.youtube.com/watch?v=r1jng79a5xc [Accessed 8 December 2021].

Channel 4 (2020) Deepfake Queen: 2020 Alternative Christmas Message. *YouTube* [video] 1 December. Available from: https://www.youtube.com/watch?v=IvY-Abd2FfM [Accessed 8 December 2021].

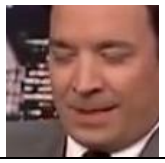
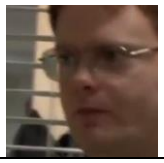
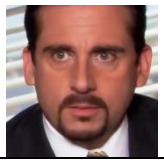
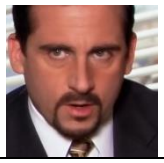
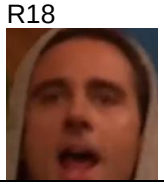
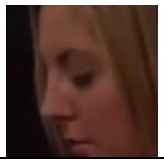
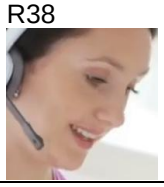
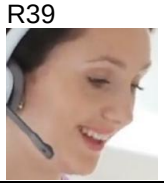
VFXChris Ume (2021) The Chronicles of DeepTomCruise Part3 (Viral Tiktok Videos). *YouTube* [video] 1 April. Available from: https://www.youtube.com/watch?v=Z_2DEcR3O9I [Accessed 8 December 2021].

Appendix 3: Table of Figures

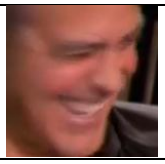
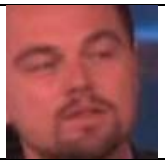
Appendix 4: Table of Tables

Appendix 5: Real Images in Test Dataset

Real images (rescaled for display below)

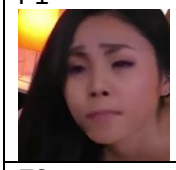
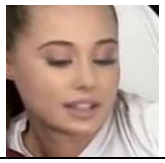
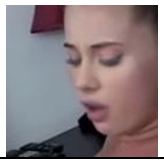
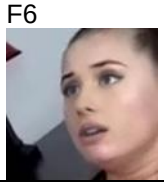
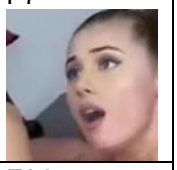
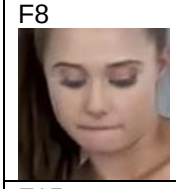
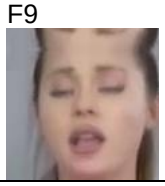
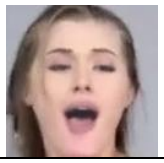
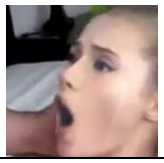
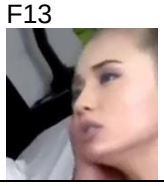
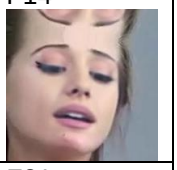
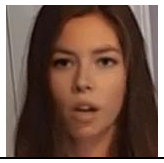
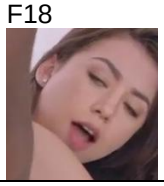
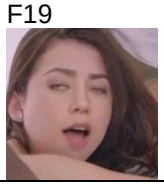
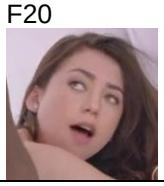
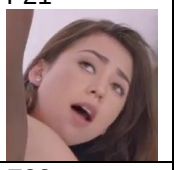
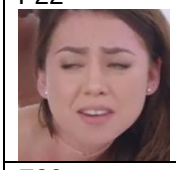
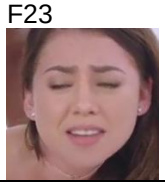
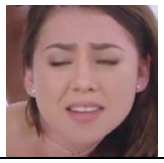
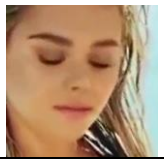
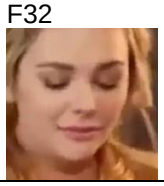
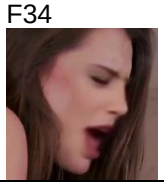
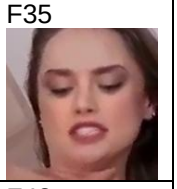
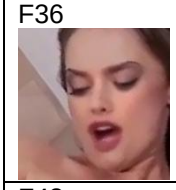
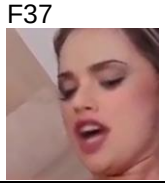
R1 	R2 	R3 	R4 	R5 	R6 	R7 
R8 	R9 	R10 	R11 	R12 	R13 	R14 
R15 	R16 	R17 	R18 	R19 	R20 	R21 
R22 	R23 	R24 	R25 	R26 	R27 	R28 
R29 	R30 	R31 	R32 	R33 	R34 	R35 
R36 	R37 	R38 	R39 	R40 	R41	R42
R43	R44	R45	R46	R47	R48	R49
R50	R51	R52	R53	R54	R55	R56
R57	R58	R59	R60	R61	R62	R63
R64	R65	R66	R67	R68	R69	R70

UFCF9Y-60-M

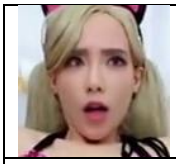
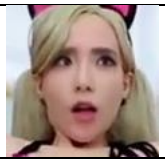
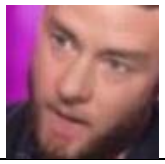
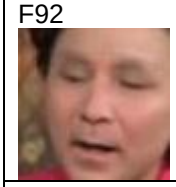
						
R71 	R72 	R73 	R74 	R75 	R76 	R77 
R78 	R79 	R80 	R81 	R82 	R83 	R84 
R85 	R86 	R87 	R88 	R89 	R90 	R91 
R92 	R93 	R94 	R95 	R96 	R97 	R98 
R99 	R100 					

Appendix 6: Deepfake Images in Test Dataset

Deepfake images (rescaled for display below)

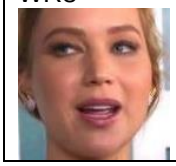
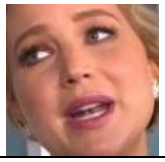
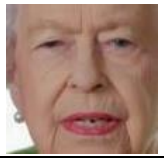
F1 	F2 	F3 	F4 	F5 	F6 	F7 
F8 	F9 	F10 	F11 	F12 	F13 	F14 
F15 	F16 	F17 	F18 	F19 	F20 	F21 
F22 	F23 	F24 	F25 	F26 	F27 	F28 
F29 	F30 	F31 	F32 	F33 	F34 	F35 
F36 	F37 	F38 	F39 	F40 	F41	F42
F43	F44	F45	F46	F47	F48	F49
F50	F51	F52	F53	F54	F55	F56
F57	F58	F59	F60	F61	F62	F63
F64	F65	F66	F67	F68	F69	F70

UFCF9Y-60-M

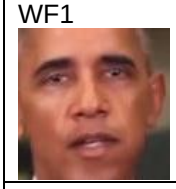
						
F71 	F72 	F73 	F74 	F75 	F76 	F77 
F78 	F79 	F80 	F81 	F82 	F83 	F84 
F85 	F86 	F87 	F88 	F89 	F90 	F91 
F92 	F93 	F94 	F95 	F96 	F97 	F98 
F99 	F100 					

Appendix 7: Images Captured from Videos in the Wild

Real images (rescaled for display below)

WR1 	WR2 	WR3 	WR4 	WR5 	WR6 	WR7 
WR8 	WR9 	WR10 	WR11 	WR12 	WR13 	WR14 

Fake images (rescaled for display below)

WF1 	WF2 	WF3 	WF4 	WF5 	WF6 	WF7 
WF8 	WF9 	WF10 	WF11 	WF12 		

Appendix 8: Codes

Summary of files

1_Dataset_Creation.ipynb - Colaboratory

2_ModelM1_Xception.ipynb - Colaboratory

2_ModelM2_VGG19.ipynb - Colaboratory

2_ModelM3_ResNet152V2.ipynb - Colaboratory

2_ModelM4_InceptionV3.ipynb - Colaboratory

2_ModelM5_InceptionResNet.ipynb - Colaboratory

2_ModelM6_MobileNet.ipynb - Colaboratory

2_ModelM7_DenseNet.ipynb - Colaboratory

2_ModelM8_NASNetMobile.ipynb - Colaboratory

2_ModelM9_EfficientNetB3.ipynb - Colaboratory

2_ModelM10_EfficientNetB7.ipynb - Colaboratory

3_FinalTest_DeepfakesInTheWild.ipynb - Colaboratory

This is a notebook for creating the datasets for training, validation and testing for checking whether a video is a real video or deepfake video

Step 1 - In a new Google Colab notebook, turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Upload the video to Google Drive

Create several folders in Google Drive

Save the videos (.mp4) inside the following folders:

- My Drive/DFD/data/DFDCpreview/trainSampleVideos/xxx.mp4
- My Drive/DFD/data/CelebDFv2/Celeb-real/xxx.mp4
- My Drive/DFD/data/CelebDFv2/Celeb-syn/xxx.mp4

Also create the following folders for backup the frames and chopped faces during image processing.

- My Drive/DFD/data/DFDCpreview/frames/train
- My Drive/DFD/data/CelebDFv2/frames/celereal
- My Drive/DFD/data/CelebDFv2/frames/celesyn
- My Drive/DFD/data/DFDCpreview/faces/train
- My Drive/DFD/data/CelebDFv2/faces/celereal
- My Drive/DFD/data/CelebDFv2/faces/celesyn

Note: Conversion from video to frames for DeepFakeDB is already processed into images of faces from the MesoNet (https://github.com/DariusAf/MesoNet). The steps for converting video to frames are mainly for Celeb-DF (v2) dataset and Deepfake Detection Challenge preview (DFDC preview) dataset. For DFDC preview, I only use training sample videos. I do not use the testing videos downloaded from Kaggle, as I do not know the label (fake / real) of the testing videos provided.

```
# Confirm Google Drive is connected with Google Colab
google = !if [ -d 'GDrive/' ]; then echo "1" ; else echo "0"; fi
if (google[0] is '0' ):
  from google.colab import drive
  drive.mount('/content/GDrive/')

!if [ -d 'GDrive/' ]; then echo "Connection to Google drive successful" ; else echo "Error to connect to Google drive"; fi
```

```
Mounted at /content/GDrive/
Connection to Google drive successful
```

```
# List the video files and model in Google drive which are to be transferred to a temporary directory in Google Colab
!ls GDrive/My\ Drive/DFD/data/DFDCpreview/trainSampleVideos
!ls GDrive/My\ Drive/DFD/data/CelebDFv2/Celeb-real
!ls GDrive/My\ Drive/DFD/data/CelebDFv2/Celeb-syn
```

```
# Count the number of videos in the DFDC preview training sample videos dataset
!ls GDrive/My\ Drive/DFD/data/DFDCpreview/trainSampleVideos | wc -l

# Count the number of videos in the Celeb-DF (v2) dataset
!ls GDrive/My\ Drive/DFD/data/CelebDFv2/Celeb-real | wc -l
!ls GDrive/My\ Drive/DFD/data/CelebDFv2/Celeb-syn | wc -l
```

Step 3 - Create a temporary directory in Google Colab. Copy the mp4 files and the model from your Google Drive to Google Colab

```
# Set up environment and import libraries in Google Colab
!pip3 install --upgrade pip > /dev/null
!pip3 install scikit-image
!pip3 install opencv-python
```

```
# Import libraries and modules in Google Colab
import cv2
```

```
from skimage.transform import resize
import matplotlib.pyplot as plt
import math
import os
```

```
not to use
# Make a temporarty directory in Google colab
!mkdir -p /content/DetectionTest/Sources/Real > /dev/null
!mkdir -p /content/DetectionTest/Sources/Fake > /dev/null
!mkdir -p /content/DetectionTest/Sources/frames/real > /dev/null
!mkdir -p /content/DetectionTest/Sources/frames/fake > /dev/null
!mkdir -p /content/DetectionTest/Sources/faces/real > /dev/null
!mkdir -p /content/DetectionTest/Sources/faces/fake > /dev/null
!mkdir -p /content/DetectionTest/Model > /dev/null

# copy video stored in Google Drive to Google Colab
!cp GDrive/My\ Drive/MyVideo/Real/*.mp4 DetectionTest/Sources/Real/
!cp GDrive/My\ Drive/MyVideo/Fake/*.mp4 DetectionTest/Sources/Fake/

# copy model stored in Google Drive to Google Colab
!cp GDrive/My\ Drive/MyModel/*.h5 DetectionTest/Model/

# List files Google Colab's temporary directory
!ls /content/DetectionTest/Sources/Real
!ls /content/DetectionTest/Sources/Fake
!ls /content/DetectionTest/Model
```

```
# Create temporarty directory in Google Colab
# Directory tree structure in Google drive
#      /DFD ─┐
#            └──────── /Sources ──────────┐
#                                         ├─ videos (.mp4)
#                                         └─ /frames (.jpg)

!mkdir -p /content/DFD/Sources/frames > /dev/null
```

```
# Copy mp4 videos stored in Google Drive to temporary directory in Google Colab
# Reminder - because of long processing time, select one of the folders for processing each time for avoid disconnection
!cp GDrive/My\ Drive/DFD/data/DFDCpreview/trainSampleVideos/*.mp4 DFD/Sources/              # trainVideos with real and fake videos
!cp GDrive/My\ Drive/DFD/data/CelebDFv2/Celeb-real/*.mp4 DFD/Sources/                      # realVideo
!cp GDrive/My\ Drive/DFD/data/CelebDFv2/Celeb-syn/*.mp4 DFD/Sources/                       # fakeVideo
```

```
# List files Google Colab's temporary directory
!ls /content/DFD/Sources
```

```
# Count number of files and folders in Google Colab's temporary directory for completeness check
!ls /content/DFD/Sources | wc -l
```

Step 4 - Extract frames from the video

```
# Folder path contains the videos
INPUT_VIDEOS_PATH = '/content/DFD/Sources'

# Folder path for storing image frames to be extracted from videos
OUTPUT_FRAMES_PATH = '/content/DFD/Sources/frames'

# Create variables
one_frame_each = 120                                                    # Para = 24, Extract 1 frame from a video for every 24 frames

!if [ -d {OUTPUT_FRAMES_PATH} ]; then echo "Output to be stored in "{OUTPUT_FRAMES_PATH} ; else mkdir {OUTPUT_FRAMES_PATH} && echo "Output d

videofiles = !ls {INPUT_VIDEOS_PATH}/*.mp4                              # Video file names in INPUT VIDEOS PATH
```

```
# Extract frames from videos
for videofile in videofiles:
    count = 0
    success = True
    filename = os.path.basename(videofile)
    vidcap = cv2.VideoCapture(videofile)

    while success:
        if (count%one_frame_each == 0):                                 # checks frame number and keeps one_frame_each
            success,image = vidcap.read()                               # reads next frame
            if image.shape[1]>640:                                      # if image width > 640, resize it
                tmp = resize(image, (math.floor(640 / image.shape[1] * image.shape[0]), 640))
            plt.imsave("%s/%s%d.jpg" % (OUTPUT_FRAMES_PATH,filename,count), tmp, format='jpg') # saves images to frame folder
            print ('*', end="")
```

```
        else:
            success,image = vidcap.read()                                   # reads next frame
        count += 1                                                          # loops counter
```

```
# Count number of image frames in Google Colab's temporary directory
!ls /content/DFD/Sources/frames | wc -l
```

```
# Copy image frames stored in temporary directory to Google Drive
# Reminder - because of long processing time, select one of the folders for processing each time for avoid disconnection
!cp DFD/Sources/frames/* GDrive/My\ Drive/DFD/data/DFDCpreview/frames/train
!cp DFD/Sources/frames/* GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celereal
!cp DFD/Sources/frames/* GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celesyn
```

```
# Check to confirm all image frames stored in Google Drive
!ls GDrive/My\ Drive/DFD/data/DFDCpreview/frames/train | wc -l
!ls GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celereal | wc -l
!ls GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celesyn | wc -l
```

```
5200
5200
1681
35388
```

Step 5 - Extract faces from the frames

Note: Conversion from frames to images of faces for DeepFakeDB is already made and downloaded from the MesoNet (https://github.com/DariusAf/MesoNet). The steps for converting frames to images of faces are mainly for Celeb-DF (v2) dataset and Deepfake Detection Challenge preview (DFDC preview) dataset. For DFDC preview, I only use training sample videos. I do not use the testing videos downloaded from Kaggle, as I do not know the label (fake / real) of the testing videos provided.

Because of long processing time, I have disconnected and reconnected to Google Colab and Google Drive after step 4. Temporary folder has to be created again in Google Colab. The frames are copied from the Google Drive to Google Colab for processing. Reconnect to Google Colab with GPU in the run time is required.

```
!apt-get install build-essential cmake
!apt-get install libopenblas-dev liblapack-dev
!pip install dlib
!pip install face_recognition
!pip install opencv-python
!pip install opencv-contrib-python
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
cmake is already the newest version (3.10.2-1ubuntu2.18.04.2).
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
Reading package lists... Done
Building dependency tree
Reading state information... Done
liblapack-dev is already the newest version (3.7.1-4ubuntu1).
libopenblas-dev is already the newest version (0.2.20+ds-4).
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
Requirement already satisfied: dlib in /usr/local/lib/python3.7/dist-packages (19.18.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
Collecting face_recognition
  Downloading face_recognition-1.3.0-py2.py3-none-any.whl (15 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from face_recognition) (1.19.5)
Collecting face-recognition-models>=0.3.0
  Downloading face_recognition_models-0.3.0.tar.gz (100.1 MB)
     |████████████████████████████████| 100.1 MB 27 kB/s
  Preparing metadata (setup.py) ... done
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (from face_recognition) (7.1.2)
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.7/dist-packages (from face_recognition) (7.1.2)
Requirement already satisfied: dlib>=19.7 in /usr/local/lib/python3.7/dist-packages (from face_recognition) (19.18.0)
Building wheels for collected packages: face-recognition-models
  Building wheel for face-recognition-models (setup.py) ... done
  Created wheel for face-recognition-models: filename=face_recognition_models-0.3.0-py2.py3-none-any.whl size=100566185 sha256=671044ba
  Stored in directory: /root/.cache/pip/wheels/d6/81/3c/884bcd5e1c120ff548d57c2ecc9ebf3281c9a6f7c0e7e7947a
Successfully built face-recognition-models
Installing collected packages: face-recognition-models, face-recognition
Successfully installed face-recognition-1.3.0 face-recognition-models-0.3.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.19.5)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
Requirement already satisfied: opencv-contrib-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-contrib-python) (1.19.5)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
```

```
# Create temporarty directory in Google Colab
# Directory tree structure in Google drive
#      /DFD ─┐
#            └──── /Sources ──────┐
#                                 ├─ images (.jpg)
#                                 └─ /Faces (.jpg)

!mkdir -p /content/DFD/Sources/Faces > /dev/null
```

```
# Import OpenCV and face_recognition libraries to extract the faces from frames
from PIL import Image
import cv2
import face_recognition
```

```
# Copy images stored in Google Drive to Colab
# Reminder - select one of the folders for processing
!cp GDrive/My\ Drive/DFD/data/DFDCpreview/frames/train/*.jpg DFD/Sources/
!cp GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celereal/*.jpg DFD/Sources/
!cp GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celefake_tr/*.jpg DFD/Sources/
!cp GDrive/My\ Drive/DFD/data/CelebDFv2/frames/celefake_te/*.jpg DFD/Sources/
```

```
# List files Google Colab's temporary directory
!ls -la /content/DFD/Sources | wc -l
```

```
     1989
```

```
# Folder path contains the frames
INPUT_FRAMES_PATH = '/content/DFD/Sources'

# Folder path for storing faces to be extracted from images
OUTPUT_FACES_PATH = '/content/DFD/Sources/Faces'

!if [ -d {OUTPUT_FACES_PATH} ]; then echo "Output to be stored in "{OUTPUT_FACES_PATH} ; else mkdir {OUTPUT_FACES_PATH} && echo "Output direc
imagefiles = !ls {INPUT_FRAMES_PATH}/*.jpg                                    # Image file names in INPUT FRAMES PATH
```

```
     Output to be stored in /content/D/F
```

```
# Extract face from all image files in the given directory
for imagefile in imagefiles:
    filename = os.path.basename(imagefile)
    image = face_recognition.load_image_file(imagefile)
    face_locations = face_recognition.face_locations(image)
    print("Found {} face(s) in this photograph - %s.".format(len(face_locations)),filename)

    %matplotlib inline
    plt.rcParams['figure.figsize'] = [32, 8]

    for face_location in face_locations:
        top, right, bottom, left = face_location
        print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right: {} for %s".format(top, left, bottom, right), filenam
        face_image = image[top:bottom, left:right]

        if(len(face_locations) == 1):
            cv2.imwrite("%s/%sF.jpg" % (OUTPUT_FACES_PATH,filename), face_image)
            print ('*', end="")
```

```
# Check to confirm all image frames stored in Google Drive
!ls DFD/Sources/Faces | wc -l
```

```
# Copy image frames stored in temporary directory to Google Drive
# Reminder - select one of the folders for processing
!cp DFD/Sources/Faces/* GDrive/My\ Drive/DFD/data/DFDCpreview/faces/train
!cp DFD/Sources/Faces/* GDrive/My\ Drive/DFD/data/CelebDFv2/faces/celereal
!cp DFD/Sources/Faces/* GDrive/My\ Drive/DFD/data/CelebDFv2/faces/celesyn
```

```
# Check number of face images stored in Google drive for completeness check
# Reminder - select one of the folders for processing
!ls GDrive/My\ Drive/DFD/data/C/faces/celetrain | wc -l
!ls GDrive/My\ Drive/DFD/data/C/faces/celereal | wc -l
!ls GDrive/My\ Drive/DFD/data/C/faces/celesyn | wc -l
```

Step 6 - Organzie file structure

After the previous steps, I have organized the images of faces in my local computer. The images of faces are then upload to Google drive with

```
# MyDrive/DFD/data ─┐
#                   └──/MyDB─────┐
#                                ├─ /test
#                                          └─ /Fake (images here)
#                                          └─ /Real (images here)
#                                └─ /train ─┐
#                                          └─ /Fake (images here)
#                                          └─ /Real (images here)
#                                └─ /validation ─┐
#                                                └─ /Fake (images here)
#                                                └─ /Real (images here)
#                                └─ myDBmeta.json


# About the metadata.json, it is created based on the file lists generated from Google drive with the following codes:
from google.colab import files

!ls GDrive/My\ Drive/DFD/data/MyDB/test/Fake > MyDBtestfakelist.csv
files.download('MyDBtestfakelist.csv')

!ls GDrive/My\ Drive/DFD/data/MyDB/test/Real > MyDBtestreallist.csv
files.download('MyDBtestreallist.csv')

!ls GDrive/My\ Drive/DFD/data/MyDB/train/Fake > MyDBtrainfakelist.csv
files.download('MyDBtrainfakelist.csv')

!ls GDrive/My\ Drive/DFD/data/MyDB/train/Real > MyDBtrainreallist.csv
files.download('MyDBtrainreallist.csv')

!ls GDrive/My\ Drive/DFD/data/MyDB/validation/Fake > MyDBvalfakelist.csv
files.download('MyDBvalfakelist.csv')

!ls GDrive/My\ Drive/DFD/data/MyDB/validation/Real > MyDBvalreallist.csv
files.download('MyDBvalreallist.csv')

# The file lists are later converted from csv to json at http://www.convertcsv.com/csv-to-json.htm
```

In sum, the dataset contains information from three sources:

1. DFDC preview - images of faces captured from 400 train sample videos downloaded from Kaggle (Link to Kaggle)
2. Celeb-DF (v2) - images of faces from real videos and deepfake videos. All videos are downloaded from authors of Celeb-DF (v2) (Link to Github, application for access to dataset is needed)
3. DeepfakeDB - images of faces from real videos and deepfake videos (Link to Github, click link to download from the given cloud storage)

Step 7 - Understand the characteristics of datasets and images

```
# Mount to Google drive for this task
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/'
```

```
/content/drive/MyDrive/DFD/data
```

```
# Show the information in metadata file
meta = pd.read_json("myDBmeta.json")
meta
```

The database is purposefully created at a balance of 50:50 real versus fake videos as shown below.

```
# Check overall distribution of real and fake videos
meta.groupby('label')['label'].count().plot(figsize=(15, 5), kind='pie', autopct='%1.2f%%', title='Real versus Fake Videos in MyDB', shadow=
plt.show()
```


```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'aba
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)
```

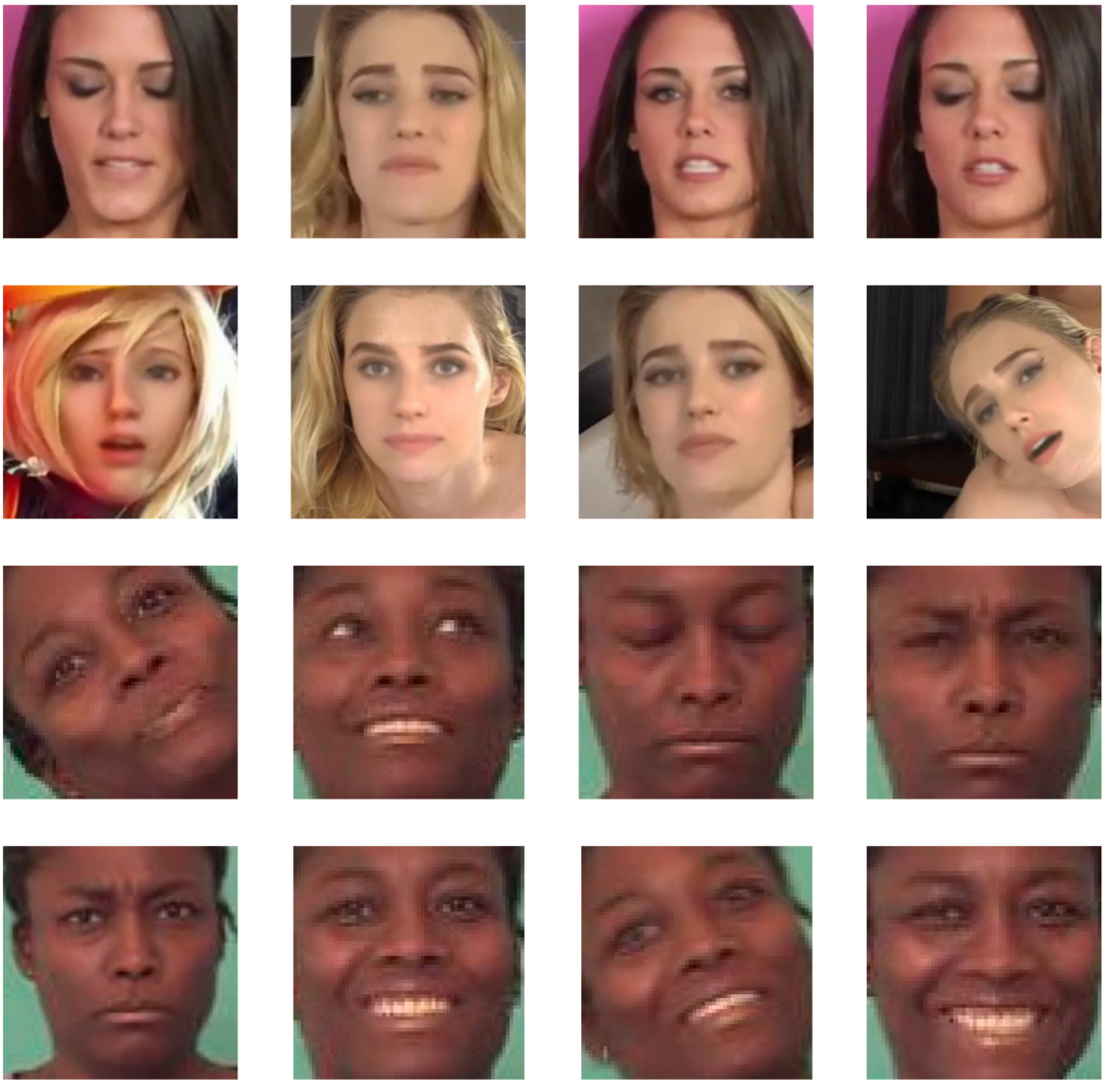
```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
```

```
                                                    image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
```

```
# Define matplotlib parameters for output images as 4x4 images
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

nrows = 4
ncols = 4

pic_index = 0                               # index for image iteration
```

```
# Show a batch of 8 real images and 8 fake images - see how difficult for human eyes could distingish real and fake images
# Rerun the cell to fetch a new batch of images

fig = plt.gcf()                               # set up matplotlib fig
fig.set_size_inches(ncols * 4, nrows * 4)     # size matplotlib fig to fit into a 4x4 image

pic_index += 8
next_real_pix = [os.path.join(train_real_dir, fname)
                 for fname in train_real_fnames[pic_index-8:pic_index]]
next_fake_pix = [os.path.join(train_fake_dir, fname)
                 for fname in train_fake_fnames[pic_index-8:pic_index]]

for i, img_path in enumerate(next_fake_pix+next_real_pix):
  sp = plt.subplot(nrows, ncols, i + 1)       # Set up subplot, subplot indices start at 1
  sp.axis('Off')                              # Turn off axis

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```

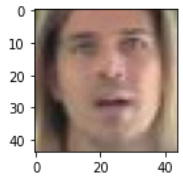


During review of the images of faces captured in the previous precedures, especially for DFDC preview and celeb-db (v2) datasets, it is noted that the same persons recording the real videos are also used for creating the deepfake. Thus, it introduces more challenges for the model to learn the features to determine if the face is real or fake.

The following two code snippets show an example of the same persons appears in the DFDC preview real and fake videos.

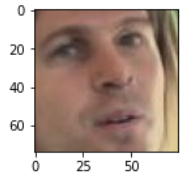
```
# Visualise a real face
plt.figure(figsize = (5,2))
viewreal = plt.imread("/content/drive/MyDrive/DFD/data/MyDB/train/Real/aelfnikyqj.mp472.jpgFace.jpg")
plt.imshow(viewreal)
```

```
<matplotlib.image.AxesImage at 0x7fd5a7bae790>
```




```
# Visualise a fake face
plt.figure(figsize = (5,2))
viewfake = plt.imread("/content/drive/MyDrive/DFD/data/MyDB/train/Fake/aagfhgtpmv.mp40.jpgFace.jpg")
plt.imshow(viewfake)
```

```
<matplotlib.image.AxesImage at 0x7fd5a7316350>
```



End of notebook

This is a notebook for training, validation and testing a binary classifier with Xception as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection is lost
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'ab
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

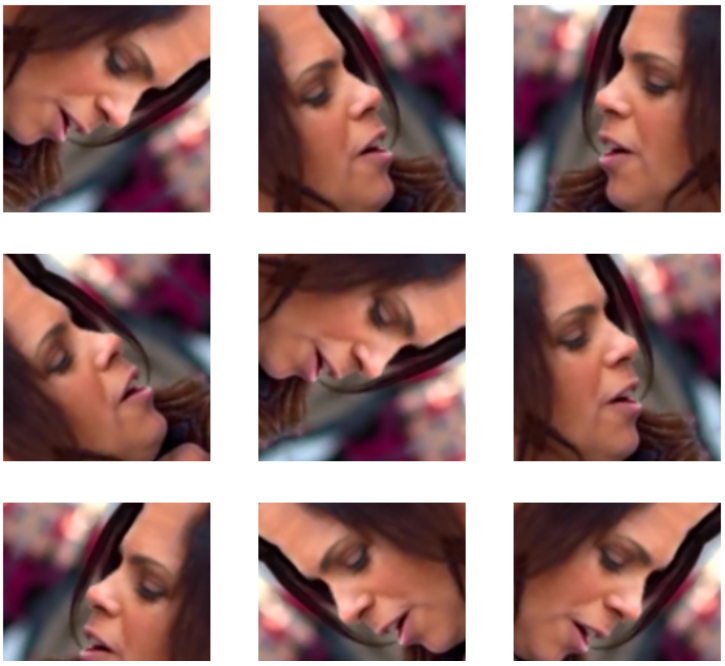
```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```


```
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```



Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.xception.preprocess_input
```

```
# Create the base model from the pre-trained model
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.xception.Xception(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernel
83689472/83683744 [==============================] - 1s 0us/step
83697664/83683744 [==============================] - 1s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 5, 5, 2048)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "xception"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 150, 150, 3  0           []
```

```
                                )]

block1_conv1 (Conv2D)          (None, 74, 74, 32)   864         ['input_1[0][0]']

block1_conv1_bn (BatchNormaliz  (None, 74, 74, 32)  128         ['block1_conv1[0][0]']
ation)

block1_conv1_act (Activation)  (None, 74, 74, 32)   0           ['block1_conv1_bn[0][0]']

block1_conv2 (Conv2D)          (None, 72, 72, 64)   18432       ['block1_conv1_act[0][0]']

block1_conv2_bn (BatchNormaliz  (None, 72, 72, 64)  256         ['block1_conv2[0][0]']
ation)

block1_conv2_act (Activation)  (None, 72, 72, 64)   0           ['block1_conv2_bn[0][0]']

block2_sepconv1 (SeparableConv  (None, 72, 72, 128)  8768       ['block1_conv2_act[0][0]']
2D)

block2_sepconv1_bn (BatchNorma  (None, 72, 72, 128)  512        ['block2_sepconv1[0][0]']
lization)

block2_sepconv2_act (Activatio  (None, 72, 72, 128)  0          ['block2_sepconv1_bn[0][0]']
n)

block2_sepconv2 (SeparableConv  (None, 72, 72, 128)  17536      ['block2_sepconv2_act[0][0]']
2D)

block2_sepconv2_bn (BatchNorma  (None, 72, 72, 128)  512        ['block2_sepconv2[0][0]']
lization)

conv2d (Conv2D)                (None, 36, 36, 128)  8192        ['block1_conv2_act[0][0]']

block2_pool (MaxPooling2D)     (None, 36, 36, 128)  0           ['block2_sepconv2_bn[0][0]']

batch_normalization (BatchNorm  (None, 36, 36, 128)  512        ['conv2d[0][0]']
alization)

add (Add)                      (None, 36, 36, 128)  0           ['block2_pool[0][0]',
                                                                 'batch_normalization[0][0]']

block3_sepconv1_act (Activatio  (None, 36, 36, 128)  0          ['add[0][0]']
n)

block3_sepconv1 (SeparableConv  (None, 36, 36, 256)  33920      ['block3_sepconv1_act[0][0]']
2D)

block3_sepconv1_bn (BatchNorma  (None, 36, 36, 256)  1024       ['block3_sepconv1[0][0]']
lization)

block3_sepconv2_act (Activatio  (None, 36, 36, 256)  0          ['block3_sepconv1_bn[0][0]']
n)
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 2048)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(150, 150, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 xception (Functional)       (None, 5, 5, 2048)        20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 976s 10s/step - loss: 0.7335 - accuracy: 0.4459
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.73
initial accuracy: 0.45
```

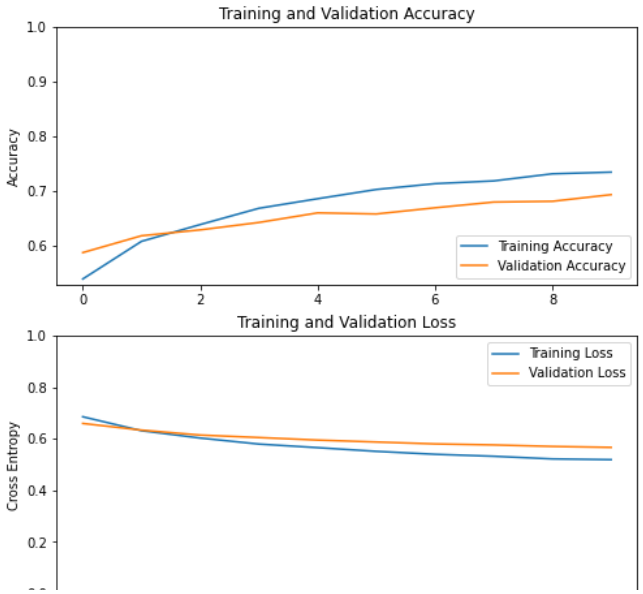
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 3917s 11s/step - loss: 0.6856 - accuracy: 0.5390 - val_loss: 0.6596 - val_accuracy: 0.5872
Epoch 2/10
364/364 [==============================] - 35s 94ms/step - loss: 0.6313 - accuracy: 0.6077 - val_loss: 0.6336 - val_accuracy: 0.6180
Epoch 3/10
364/364 [==============================] - 35s 94ms/step - loss: 0.6028 - accuracy: 0.6384 - val_loss: 0.6144 - val_accuracy: 0.6287
Epoch 4/10
364/364 [==============================] - 35s 94ms/step - loss: 0.5795 - accuracy: 0.6682 - val_loss: 0.6048 - val_accuracy: 0.6423
Epoch 5/10
364/364 [==============================] - 35s 94ms/step - loss: 0.5655 - accuracy: 0.6857 - val_loss: 0.5949 - val_accuracy: 0.6598
Epoch 6/10
364/364 [==============================] - 34s 92ms/step - loss: 0.5511 - accuracy: 0.7026 - val_loss: 0.5874 - val_accuracy: 0.6578
Epoch 7/10
364/364 [==============================] - 36s 96ms/step - loss: 0.5397 - accuracy: 0.7133 - val_loss: 0.5801 - val_accuracy: 0.6691
Epoch 8/10
364/364 [==============================] - 36s 97ms/step - loss: 0.5321 - accuracy: 0.7183 - val_loss: 0.5761 - val_accuracy: 0.6796
Epoch 9/10
364/364 [==============================] - 34s 93ms/step - loss: 0.5218 - accuracy: 0.7313 - val_loss: 0.5703 - val_accuracy: 0.6810
Epoch 10/10
364/364 [==============================] - 35s 95ms/step - loss: 0.5189 - accuracy: 0.7343 - val_loss: 0.5665 - val_accuracy: 0.6931
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
```

```
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  132
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )
```

```
 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 xception (Functional)       (None, 5, 5, 2048)        20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 20,863,529
Trainable params: 9,480,393
Non-trainable params: 11,383,136
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
41
```

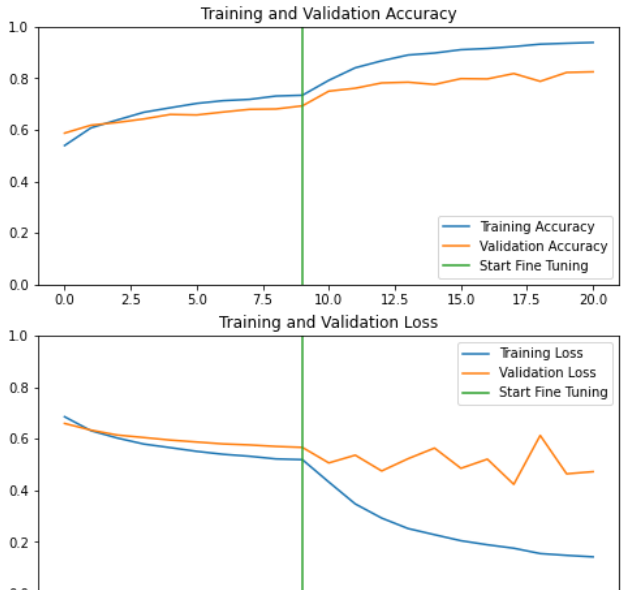
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
364/364 [==============================] - 42s 103ms/step - loss: 0.4320 - accuracy: 0.7924 - val_loss: 0.5062 - val_accuracy: 0.7502
Epoch 11/20
364/364 [==============================] - 36s 96ms/step - loss: 0.3470 - accuracy: 0.8410 - val_loss: 0.5363 - val_accuracy: 0.7615
Epoch 12/20
364/364 [==============================] - 35s 96ms/step - loss: 0.2920 - accuracy: 0.8680 - val_loss: 0.4750 - val_accuracy: 0.7821
Epoch 13/20
364/364 [==============================] - 35s 96ms/step - loss: 0.2513 - accuracy: 0.8906 - val_loss: 0.5226 - val_accuracy: 0.7850
Epoch 14/20
364/364 [==============================] - 35s 96ms/step - loss: 0.2274 - accuracy: 0.8983 - val_loss: 0.5639 - val_accuracy: 0.7762
Epoch 15/20
364/364 [==============================] - 35s 96ms/step - loss: 0.2040 - accuracy: 0.9113 - val_loss: 0.4853 - val_accuracy: 0.7988
Epoch 16/20
364/364 [==============================] - 35s 96ms/step - loss: 0.1883 - accuracy: 0.9158 - val_loss: 0.5208 - val_accuracy: 0.7977
Epoch 17/20
364/364 [==============================] - 35s 96ms/step - loss: 0.1749 - accuracy: 0.9233 - val_loss: 0.4230 - val_accuracy: 0.8186
Epoch 18/20
364/364 [==============================] - 35s 96ms/step - loss: 0.1542 - accuracy: 0.9328 - val_loss: 0.6130 - val_accuracy: 0.7884
Epoch 19/20
364/364 [==============================] - 35s 96ms/step - loss: 0.1471 - accuracy: 0.9362 - val_loss: 0.4638 - val_accuracy: 0.8228
Epoch 20/20
364/364 [==============================] - 35s 96ms/step - loss: 0.1413 - accuracy: 0.9393 - val_loss: 0.4723 - val_accuracy: 0.8254
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

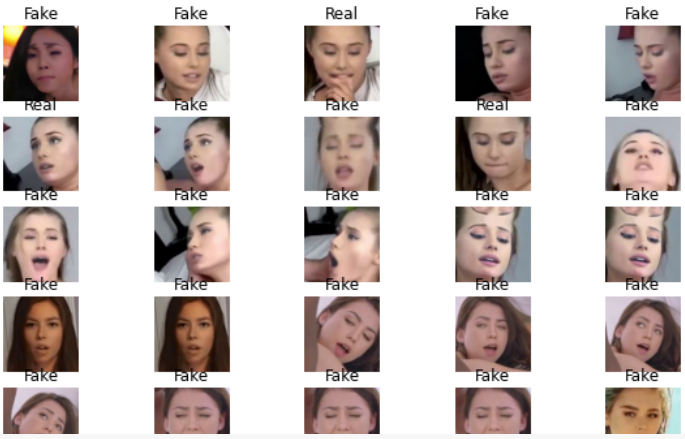
```
5/5 [==============================] - 60s 12s/step - loss: 0.6574 - accuracy: 0.7250
Test accuracy : 0.7250000238418579
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 34_0.jpg to 34_0.jpg
Saving 34_6.jpg to 34_6.jpg
Saving 35_0.jpg to 35_0.jpg
Saving 39_360.jpg to 39_360.jpg
Saving 39_372.jpg to 39_372.jpg
Saving 39_384.jpg to 39_384.jpg
Saving 39_402.jpg to 39_402.jpg
Saving 39_780.jpg to 39_780.jpg
Saving 39_786.jpg to 39_786.jpg
Saving 40_0.jpg to 40_0.jpg
Saving 40_12.jpg to 40_12.jpg
Saving 40_24.jpg to 40_24.jpg
Saving 42_168.jpg to 42_168.jpg
Saving 42_198.jpg to 42_198.jpg
Saving 42_204.jpg to 42_204.jpg
Saving 43_36.jpg to 43_36.jpg
Saving 43_150.jpg to 43_150.jpg
Saving 43_192.jpg to 43_192.jpg
Saving 43_210.jpg to 43_210.jpg
Saving 44_6.jpg to 44_6.jpg
Saving 44_90.jpg to 44_90.jpg
Saving 44_192.jpg to 44_192.jpg
Saving 44_198.jpg to 44_198.jpg
Saving 45_60.jpg to 45_60.jpg
Saving 45_150.jpg to 45_150.jpg
Saving 46_360.jpg to 46_360.jpg
Saving 46_384.jpg to 46_384.jpg
Saving 47_528.jpg to 47_528.jpg
Saving 47_738.jpg to 47_738.jpg
Saving 48_12.jpg to 48_12.jpg
Saving 48_270.jpg to 48_270.jpg
Saving 48_318.jpg to 48_318.jpg
Saving 48_522.jpg to 48_522.jpg
Saving 48_708.jpg to 48_708.jpg
Saving 50_192.jpg to 50_192.jpg
Saving 50_222.jpg to 50_222.jpg
Saving 51_120.jpg to 51_120.jpg
Saving 51_300.jpg to 51_300.jpg
Saving 51_372.jpg to 51_372.jpg
Saving 52_42.jpg to 52_42.jpg
Saving 52_450.jpg to 52_450.jpg
Saving 53_66.jpg to 53_66.jpg
Saving 54_18.jpg to 54_18.jpg
Saving 54_186.jpg to 54_186.jpg
Saving 69_268.jpg to 69_268.jpg
Saving 69_302.jpg to 69_302.jpg
Saving 78_3.jpg to 78_3.jpg
Saving 78_318.jpg to 78_318.jpg
Saving 96_30.jpg to 96_30.jpg
```

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image
```

```
uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 114_666.jpg to 114_666.jpg
Saving 115_306.jpg to 115_306.jpg
Saving 115_816.jpg to 115_816.jpg
Saving 116_24.jpg to 116_24.jpg
Saving 116_1056.jpg to 116_1056.jpg
Saving 117_12.jpg to 117_12.jpg
Saving 117_750.jpg to 117_750.jpg
Saving 118_9.jpg to 118_9.jpg
Saving 118_405.jpg to 118_405.jpg
Saving 119_65.jpg to 119_65.jpg
Saving 119_660.jpg to 119_660.jpg
Saving 120_50.jpg to 120_50.jpg
Saving 120_235.jpg to 120_235.jpg
Saving 121_24.jpg to 121_24.jpg
Saving 121_28.jpg to 121_28.jpg
Saving 122_0.jpg to 122_0.jpg
Saving 122_1.jpg to 122_1.jpg
Saving 124_154.jpg to 124_154.jpg
Saving 124_158.jpg to 124_158.jpg
Saving 124_160.jpg to 124_160.jpg
Saving 124_170.jpg to 124_170.jpg
Saving 125_65.jpg to 125_65.jpg
Saving 125_68.jpg to 125_68.jpg
Saving 125_69.jpg to 125_69.jpg
Saving 130_2.jpg to 130_2.jpg
Saving 130_34.jpg to 130_34.jpg
Saving 130_50.jpg to 130_50.jpg
Saving 130_78.jpg to 130_78.jpg
Saving 131_172.jpg to 131_172.jpg
Saving 131_212.jpg to 131_212.jpg
Saving 132_36.jpg to 132_36.jpg
Saving 132_105.jpg to 132_105.jpg
Saving 132_237.jpg to 132_237.jpg
Saving 133_144.jpg to 133_144.jpg
Saving 134_90.jpg to 134_90.jpg
Saving 134_190.jpg to 134_190.jpg
Saving 134_555.jpg to 134_555.jpg
Saving 135_370.jpg to 135_370.jpg
Saving 135_830.jpg to 135_830.jpg
Saving 135_1080.jpg to 135_1080.jpg
Saving 142_396.jpg to 142_396.jpg
Saving 142_412.jpg to 142_412.jpg
Saving 142_420.jpg to 142_420.jpg
Saving 143_23.jpg to 143_23.jpg
Saving 143_25.jpg to 143_25.jpg
Saving 143_26.jpg to 143_26.jpg
Saving 143_27.jpg to 143_27.jpg
Saving 143_29.jpg to 143_29.jpg
Saving 143_32.jpg to 143_32.jpg
```

Step 10 - Determine the performance of the classifier

```
# Figures for performance calculation
# real = 1, fake = 0
TP = true_pos = 60 # real faces predicted as real
TN = true_neg = 83 # fake faces predicted as fake
FP = false_pos = 17 # fake faces predicted as real
FN = false_neg = 40 # real faces predicted as fake
```

```
results = {}
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
ACC is  0.715
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.600
```

```
# True Negative Rate
```

```
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.830
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.779
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.675
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.678
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.442
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_Xception.h5')
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask layers require a config and mus
  layer_config = serialize_layer_fn(layer)
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 xception (Functional)       (None, 5, 5, 2048)        20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)
```

```
 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 20,863,529
Trainable params: 9,480,393
Non-trainable params: 11,383,136
_________________________________________________________________
```

End of Notebook

This is a notebook for training, validation and testing a binary classifier with VGG19 as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'ab
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

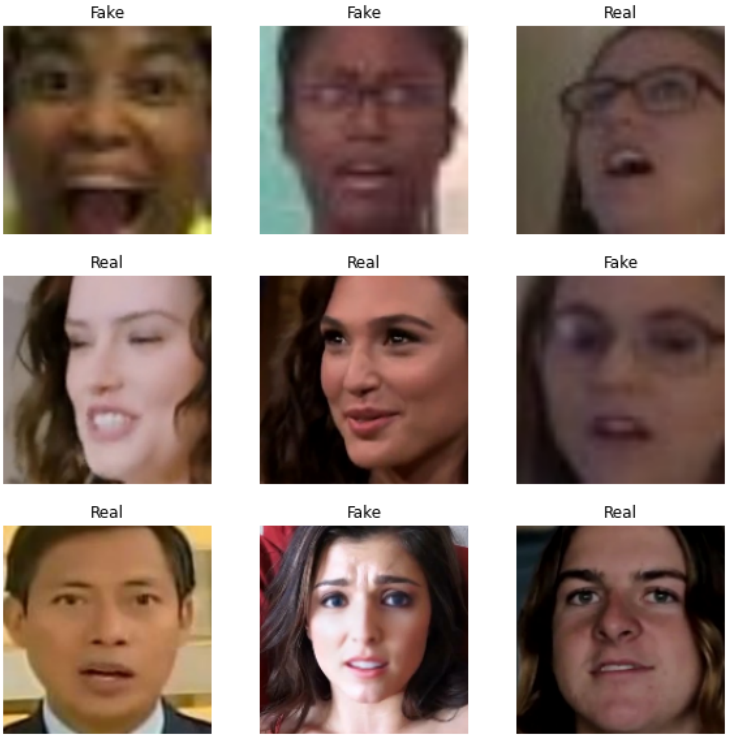
Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

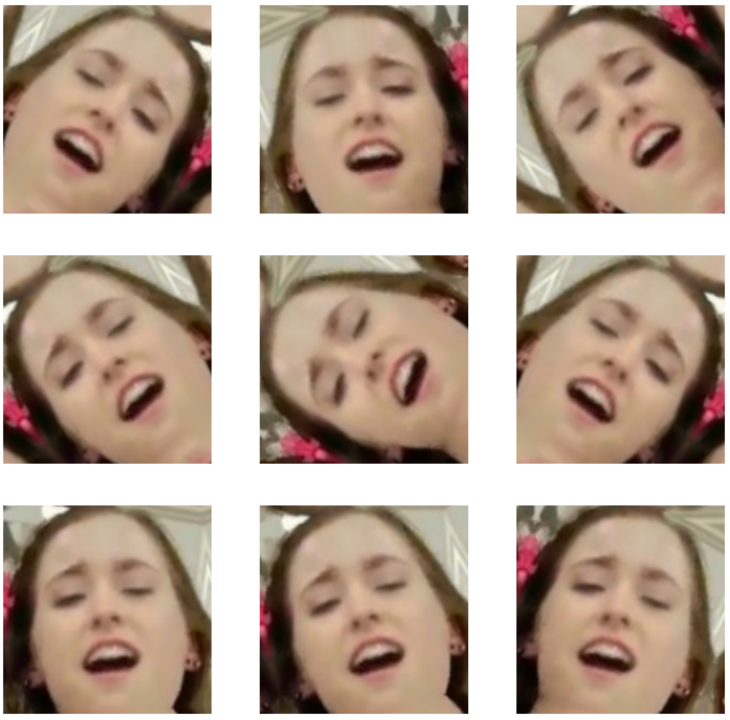



```
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```



Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.vgg19.preprocess_input
```

```
# Create the base model from the pre-trained model
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.vgg19.VGG19(input_shape=IMG_SHAPE,
                                                       include_top=False,
                                                       weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_noto
80142336/80134624 [==============================] - 0s 0us/step
80150528/80134624 [==============================] - 0s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 4, 4, 512)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0
```

```
 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv4 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv4 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
_________________________________________________________________
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 512)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(150, 150, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.__operators__.getitem (S  (None, 150, 150, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 150, 150, 3)      0

 vgg19 (Functional)          (None, 4, 4, 512)         20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 1)                 513

=================================================================
Total params: 20,024,897
Trainable params: 513
Non-trainable params: 20,024,384
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 422s 4s/step - loss: 1.3527 - accuracy: 0.5112
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 1.35
initial accuracy: 0.51
```

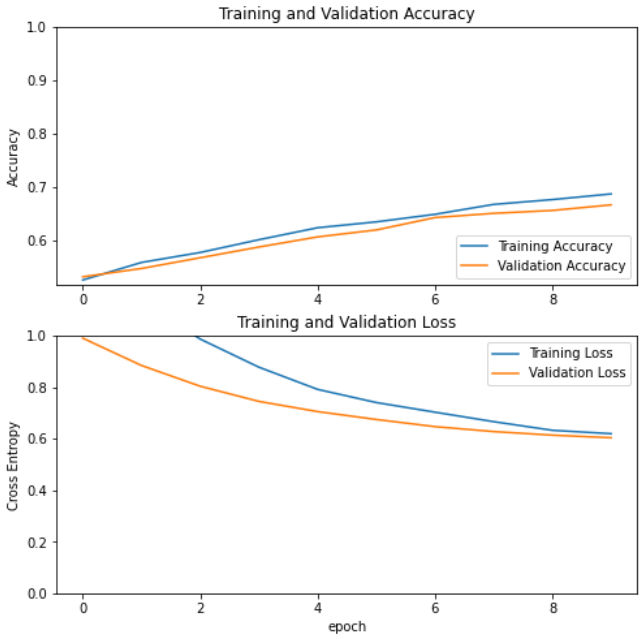
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 1747s 5s/step - loss: 1.3898 - accuracy: 0.5267 - val_loss: 0.9908 - val_accuracy: 0.5326
Epoch 2/10
364/364 [==============================] - 34s 93ms/step - loss: 1.1329 - accuracy: 0.5595 - val_loss: 0.8847 - val_accuracy: 0.5482
Epoch 3/10
364/364 [==============================] - 33s 91ms/step - loss: 0.9873 - accuracy: 0.5781 - val_loss: 0.8042 - val_accuracy: 0.5682
Epoch 4/10
364/364 [==============================] - 33s 91ms/step - loss: 0.8781 - accuracy: 0.6020 - val_loss: 0.7452 - val_accuracy: 0.5883
Epoch 5/10
364/364 [==============================] - 34s 92ms/step - loss: 0.7919 - accuracy: 0.6244 - val_loss: 0.7056 - val_accuracy: 0.6072
Epoch 6/10
364/364 [==============================] - 34s 93ms/step - loss: 0.7408 - accuracy: 0.6353 - val_loss: 0.6749 - val_accuracy: 0.6202
Epoch 7/10
364/364 [==============================] - 34s 91ms/step - loss: 0.7033 - accuracy: 0.6495 - val_loss: 0.6471 - val_accuracy: 0.6434
Epoch 8/10
364/364 [==============================] - 34s 93ms/step - loss: 0.6664 - accuracy: 0.6680 - val_loss: 0.6279 - val_accuracy: 0.6513
Epoch 9/10
364/364 [==============================] - 34s 92ms/step - loss: 0.6329 - accuracy: 0.6770 - val_loss: 0.6141 - val_accuracy: 0.6567
Epoch 10/10
364/364 [==============================] - 34s 91ms/step - loss: 0.6197 - accuracy: 0.6876 - val_loss: 0.6040 - val_accuracy: 0.6671
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
```

```
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  22
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.__operators__.getitem (S  (None, 150, 150, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 150, 150, 3)      0

 vgg19 (Functional)          (None, 4, 4, 512)         20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)
```

```
 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 1)                 513

=================================================================
Total params: 20,024,897
Trainable params: 513
Non-trainable params: 20,024,384
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

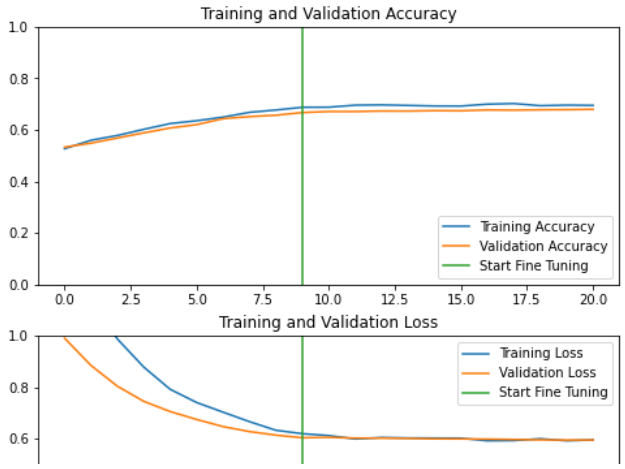
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
364/364 [==============================] - 35s 91ms/step - loss: 0.6124 - accuracy: 0.6879 - val_loss: 0.6057 - val_accuracy: 0.6711
Epoch 11/20
364/364 [==============================] - 34s 92ms/step - loss: 0.5992 - accuracy: 0.6959 - val_loss: 0.6028 - val_accuracy: 0.6711
Epoch 12/20
364/364 [==============================] - 34s 93ms/step - loss: 0.6048 - accuracy: 0.6968 - val_loss: 0.6024 - val_accuracy: 0.6731
Epoch 13/20
364/364 [==============================] - 34s 92ms/step - loss: 0.6032 - accuracy: 0.6948 - val_loss: 0.6015 - val_accuracy: 0.6728
Epoch 14/20
364/364 [==============================] - 35s 94ms/step - loss: 0.6027 - accuracy: 0.6927 - val_loss: 0.5997 - val_accuracy: 0.6745
Epoch 15/20
364/364 [==============================] - 34s 93ms/step - loss: 0.6024 - accuracy: 0.6924 - val_loss: 0.5994 - val_accuracy: 0.6739
Epoch 16/20
364/364 [==============================] - 35s 94ms/step - loss: 0.5922 - accuracy: 0.6998 - val_loss: 0.5986 - val_accuracy: 0.6773
Epoch 17/20
364/364 [==============================] - 33s 90ms/step - loss: 0.5930 - accuracy: 0.7021 - val_loss: 0.5972 - val_accuracy: 0.6765
Epoch 18/20
364/364 [==============================] - 35s 96ms/step - loss: 0.6002 - accuracy: 0.6939 - val_loss: 0.5960 - val_accuracy: 0.6779
Epoch 19/20
364/364 [==============================] - 35s 96ms/step - loss: 0.5923 - accuracy: 0.6960 - val_loss: 0.5952 - val_accuracy: 0.6784
Epoch 20/20
364/364 [==============================] - 32s 85ms/step - loss: 0.5956 - accuracy: 0.6950 - val_loss: 0.5954 - val_accuracy: 0.6796
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

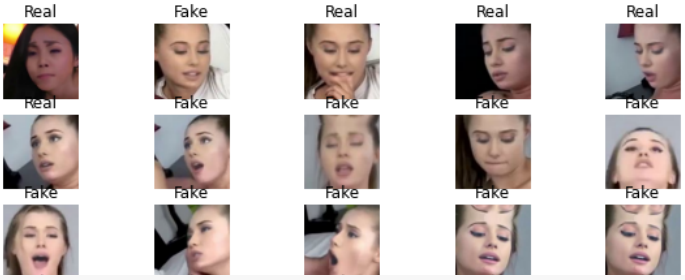
```
5/5 [==============================] - 24s 5s/step - loss: 0.5890 - accuracy: 0.6250
Test accuracy : 0.625
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0
 0 1 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 34_0.jpg to 34_0.jpg
Saving 34_6.jpg to 34_6.jpg
Saving 35_0.jpg to 35_0.jpg
Saving 39_360.jpg to 39_360.jpg
Saving 39_372.jpg to 39_372.jpg
Saving 39_384.jpg to 39_384.jpg
Saving 39_402.jpg to 39_402.jpg
Saving 39_780.jpg to 39_780.jpg
Saving 39_786.jpg to 39_786.jpg
Saving 40_0.jpg to 40_0.jpg
Saving 40_12.jpg to 40_12.jpg
Saving 40_24.jpg to 40_24.jpg
Saving 42_168.jpg to 42_168.jpg
Saving 42_198.jpg to 42_198.jpg
Saving 42_204.jpg to 42_204.jpg
Saving 43_36.jpg to 43_36.jpg
Saving 43_150.jpg to 43_150.jpg
Saving 43_192.jpg to 43_192.jpg
Saving 43_210.jpg to 43_210.jpg
Saving 44_6.jpg to 44_6.jpg
Saving 44_90.jpg to 44_90.jpg
Saving 44_192.jpg to 44_192.jpg
Saving 44_198.jpg to 44_198.jpg
Saving 45_60.jpg to 45_60.jpg
Saving 45_150.jpg to 45_150.jpg
Saving 46_360.jpg to 46_360.jpg
Saving 46_384.jpg to 46_384.jpg
Saving 47_528.jpg to 47_528.jpg
Saving 47_738.jpg to 47_738.jpg
Saving 48_12.jpg to 48_12.jpg
Saving 48_270.jpg to 48_270.jpg
Saving 48_318.jpg to 48_318.jpg
Saving 48_522.jpg to 48_522.jpg
Saving 48_708.jpg to 48_708.jpg
Saving 50_192.jpg to 50_192.jpg
Saving 50_222.jpg to 50_222.jpg
Saving 51_120.jpg to 51_120.jpg
Saving 51_300.jpg to 51_300.jpg
Saving 51_372.jpg to 51_372.jpg
Saving 52_42.jpg to 52_42.jpg
Saving 52_450.jpg to 52_450.jpg
Saving 53_66.jpg to 53_66.jpg
Saving 54_18.jpg to 54_18.jpg
Saving 54_186.jpg to 54_186.jpg
Saving 69_268.jpg to 69_268.jpg
Saving 69_302.jpg to 69_302.jpg
Saving 78_3.jpg to 78_3.jpg
Saving 78_318.jpg to 78_318.jpg
Saving 96_30.jpg to 96_30.jpg
```

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
```

```
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Step 10 - Determine the performance of the classifier

```
# Figures for performance calculation
# real = 1, fake = 0
TP = true_pos = 47 # real faces predicted as real
TN = true_neg = 81 # fake faces predicted as fake
FP = false_pos = 19 # fake faces predicted as real
FN = false_neg = 53  # real faces predicted as fake
```

```
results = {}
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
```

```
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
Saving 114_660.jpg to 114_660.jpg
ACC is  0.620
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.470
```

```
Saving 119_660.jpg to 119_660.jpg
```

```
# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.810
```

```
Saving 124_160.jpg to 124_160.jpg
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.712
```

```
Saving 133_270.jpg to 133_270.jpg
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.604
```

```
Saving 135_830.jpg to 135_830.jpg
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.566
```

```
Saving 143_29.jpg to 143_29.jpg
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.298
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_VGG19.h5')
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0
```

```
 tf.__operators__.getitem (S  (None, 150, 150, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 150, 150, 3)      0

 vgg19 (Functional)           (None, 4, 4, 512)        20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dropout (Dropout)            (None, 512)              0

 dense (Dense)                (None, 1)                513

=================================================================
Total params: 20,024,897
Trainable params: 513
Non-trainable params: 20,024,384
_________________________________________________________________
```

End of Notebook

This is a notebook for training, validation and testing a binary classifier with ResNet152V2 as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection is lost
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'aba
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

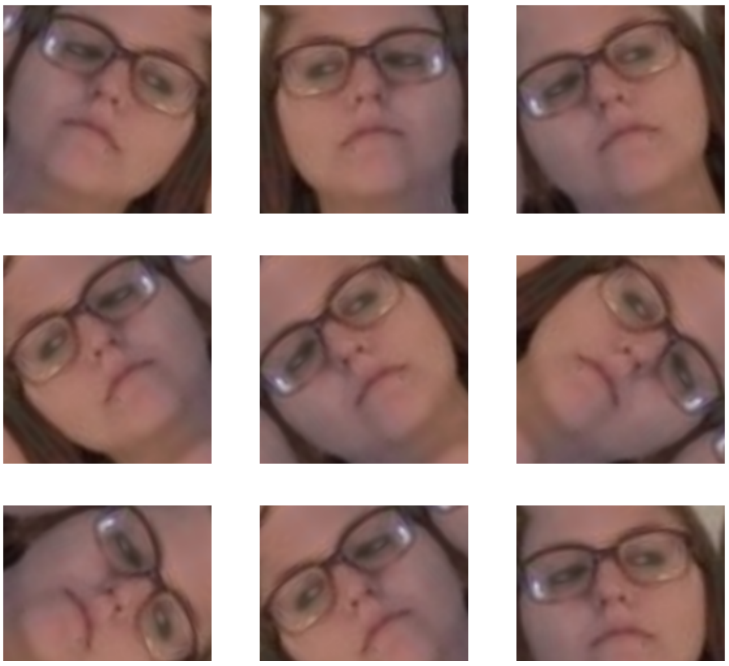
```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```


```
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```



Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.resnet_v2.preprocess_input
```

```
# Create the base model from the pre-trained model DenseNet
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.resnet_v2.ResNet152V2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet152v2_weights_tf_dim_ordering_tf_kernel
234553344/234545216 [==============================] - 3s 0us/step
234561536/234545216 [==============================] - 3s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 5, 5, 2048)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "resnet152v2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 150, 150, 3  0           []
```

```
                                )]

 conv1_pad (ZeroPadding2D)      (None, 156, 156, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 75, 75, 64)   9472        ['conv1_pad[0][0]']

 pool1_pad (ZeroPadding2D)      (None, 77, 77, 64)   0           ['conv1_conv[0][0]']

 pool1_pool (MaxPooling2D)      (None, 38, 38, 64)   0           ['pool1_pad[0][0]']

 conv2_block1_preact_bn (BatchN  (None, 38, 38, 64)  256         ['pool1_pool[0][0]']
 ormalization)

 conv2_block1_preact_relu (Acti  (None, 38, 38, 64)  0           ['conv2_block1_preact_bn[0][0]']
 vation)

 conv2_block1_1_conv (Conv2D)   (None, 38, 38, 64)   4096        ['conv2_block1_preact_relu[0][0]'
                                                                 ]

 conv2_block1_1_bn (BatchNormal  (None, 38, 38, 64)  256         ['conv2_block1_1_conv[0][0]']
 ization)

 conv2_block1_1_relu (Activatio  (None, 38, 38, 64)  0           ['conv2_block1_1_bn[0][0]']
 n)

 conv2_block1_2_pad (ZeroPaddin  (None, 40, 40, 64)  0           ['conv2_block1_1_relu[0][0]']
 g2D)

 conv2_block1_2_conv (Conv2D)   (None, 38, 38, 64)   36864       ['conv2_block1_2_pad[0][0]']

 conv2_block1_2_bn (BatchNormal  (None, 38, 38, 64)  256         ['conv2_block1_2_conv[0][0]']
 ization)

 conv2_block1_2_relu (Activatio  (None, 38, 38, 64)  0           ['conv2_block1_2_bn[0][0]']
 n)

 conv2_block1_0_conv (Conv2D)   (None, 38, 38, 256)  16640       ['conv2_block1_preact_relu[0][0]'
                                                                 ]

 conv2_block1_3_conv (Conv2D)   (None, 38, 38, 256)  16640       ['conv2_block1_2_relu[0][0]']

 conv2_block1_out (Add)         (None, 38, 38, 256)  0           ['conv2_block1_0_conv[0][0]',
                                                                  'conv2_block1_3_conv[0][0]']

 conv2_block2_preact_bn (BatchN  (None, 38, 38, 256)  1024       ['conv2_block1_out[0][0]']
 ormalization)

 conv2_block2_preact_relu (Acti  (None, 38, 38, 256)  0          ['conv2_block2_preact_bn[0][0]']
 vation)

 conv2_block2_1_conv (Conv2D)   (None, 38, 38, 64)   16384       ['conv2_block2_preact_relu[0][0]'
                                                                 ]
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 2048)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(150, 150, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 resnet152v2 (Functional)    (None, 5, 5, 2048)        58331648

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 58,333,697
Trainable params: 2,049
Non-trainable params: 58,331,648
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 859s 9s/step - loss: 1.0206 - accuracy: 0.4747
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 1.02
initial accuracy: 0.47
```

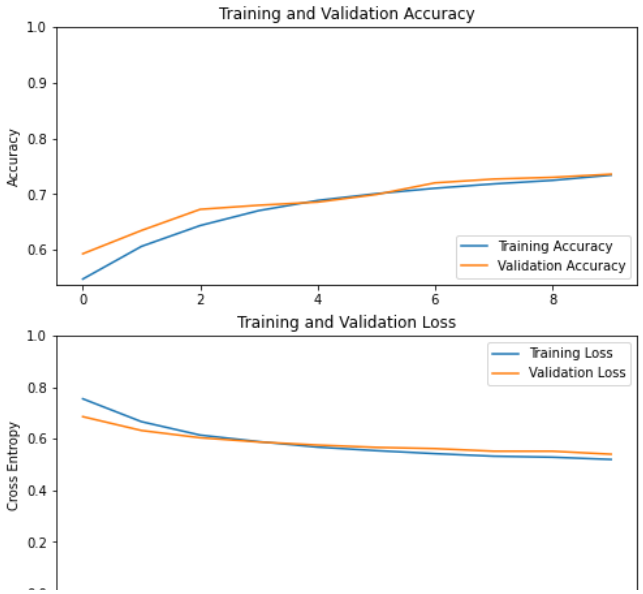
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 3406s 9s/step - loss: 0.7553 - accuracy: 0.5478 - val_loss: 0.6860 - val_accuracy: 0.5931
Epoch 2/10
364/364 [==============================] - 50s 135ms/step - loss: 0.6664 - accuracy: 0.6064 - val_loss: 0.6321 - val_accuracy: 0.6349
Epoch 3/10
364/364 [==============================] - 49s 134ms/step - loss: 0.6143 - accuracy: 0.6437 - val_loss: 0.6040 - val_accuracy: 0.6728
Epoch 4/10
364/364 [==============================] - 49s 134ms/step - loss: 0.5889 - accuracy: 0.6705 - val_loss: 0.5875 - val_accuracy: 0.6801
Epoch 5/10
364/364 [==============================] - 49s 134ms/step - loss: 0.5676 - accuracy: 0.6887 - val_loss: 0.5756 - val_accuracy: 0.6858
Epoch 6/10
364/364 [==============================] - 50s 135ms/step - loss: 0.5539 - accuracy: 0.7009 - val_loss: 0.5666 - val_accuracy: 0.6994
Epoch 7/10
364/364 [==============================] - 49s 134ms/step - loss: 0.5423 - accuracy: 0.7106 - val_loss: 0.5618 - val_accuracy: 0.7203
Epoch 8/10
364/364 [==============================] - 50s 136ms/step - loss: 0.5323 - accuracy: 0.7184 - val_loss: 0.5517 - val_accuracy: 0.7270
Epoch 9/10
364/364 [==============================] - 50s 135ms/step - loss: 0.5284 - accuracy: 0.7247 - val_loss: 0.5517 - val_accuracy: 0.7301
Epoch 10/10
364/364 [==============================] - 50s 135ms/step - loss: 0.5197 - accuracy: 0.7342 - val_loss: 0.5401 - val_accuracy: 0.7358
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
```

```
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  564
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )
```

```
 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 resnet152v2 (Functional)    (None, 5, 5, 2048)        58331648

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 58,333,697
Trainable params: 56,396,545
Non-trainable params: 1,937,152
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
423
```

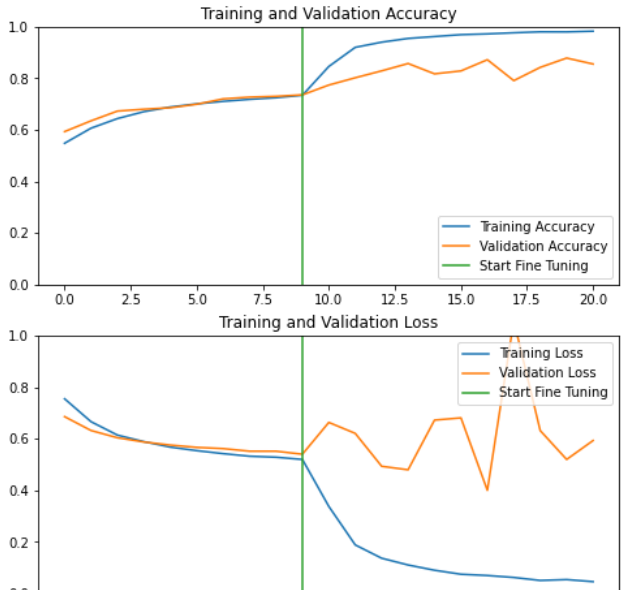
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
364/364 [==============================] - 142s 326ms/step - loss: 0.3373 - accuracy: 0.8461 - val_loss: 0.6636 - val_accuracy: 0.7739
Epoch 11/20
364/364 [==============================] - 116s 318ms/step - loss: 0.1878 - accuracy: 0.9204 - val_loss: 0.6209 - val_accuracy: 0.8022
Epoch 12/20
364/364 [==============================] - 116s 319ms/step - loss: 0.1360 - accuracy: 0.9405 - val_loss: 0.4931 - val_accuracy: 0.8290
Epoch 13/20
364/364 [==============================] - 116s 318ms/step - loss: 0.1098 - accuracy: 0.9549 - val_loss: 0.4795 - val_accuracy: 0.8576
Epoch 14/20
364/364 [==============================] - 116s 318ms/step - loss: 0.0894 - accuracy: 0.9624 - val_loss: 0.6726 - val_accuracy: 0.8175
Epoch 15/20
364/364 [==============================] - 116s 318ms/step - loss: 0.0738 - accuracy: 0.9697 - val_loss: 0.6811 - val_accuracy: 0.8288
Epoch 16/20
364/364 [==============================] - 117s 319ms/step - loss: 0.0689 - accuracy: 0.9726 - val_loss: 0.3998 - val_accuracy: 0.8723
Epoch 17/20
364/364 [==============================] - 117s 319ms/step - loss: 0.0613 - accuracy: 0.9770 - val_loss: 1.0586 - val_accuracy: 0.7912
Epoch 18/20
364/364 [==============================] - 116s 319ms/step - loss: 0.0497 - accuracy: 0.9806 - val_loss: 0.6318 - val_accuracy: 0.8426
Epoch 19/20
364/364 [==============================] - 117s 319ms/step - loss: 0.0531 - accuracy: 0.9804 - val_loss: 0.5197 - val_accuracy: 0.8788
Epoch 20/20
364/364 [==============================] - 117s 319ms/step - loss: 0.0450 - accuracy: 0.9831 - val_loss: 0.5931 - val_accuracy: 0.8556
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

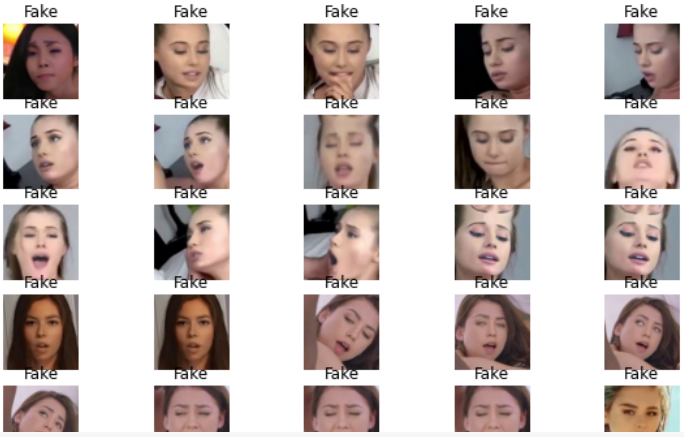
```
5/5 [==============================] - 47s 9s/step - loss: 0.6016 - accuracy: 0.8400
Test accuracy : 0.8399999737739563
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0
 0 0 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving 34_0.jpg to 34_0.jpg
Saving 34_6.jpg to 34_6.jpg
Saving 35_0.jpg to 35_0.jpg
Saving 39_360.jpg to 39_360.jpg
Saving 39_372.jpg to 39_372.jpg
Saving 39_384.jpg to 39_384.jpg
Saving 39_402.jpg to 39_402.jpg
Saving 39_780.jpg to 39_780.jpg
Saving 39_786.jpg to 39_786.jpg
Saving 40_0.jpg to 40_0.jpg
Saving 40_12.jpg to 40_12.jpg
Saving 40_24.jpg to 40_24.jpg
Saving 42_168.jpg to 42_168.jpg
```

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving 114_666.jpg to 114_666.jpg
Saving 115_306.jpg to 115_306.jpg
Saving 115_816.jpg to 115_816.jpg
```

Step 10 - Determine the performance of the classifier

```
Saving 117_750.jpg to 117_750.jpg
```

```
# Figures for performance calculation
# real = 1, fake = 0
TP = true_pos = 86 # real faces predicted as real
TN = true_neg = 86 # fake faces predicted as fake
FP = false_pos = 14 # fake faces predicted as real
FN = false_neg = 14 # real faces predicted as fake
```

```
Saving 122_0.jpg to 122_0.jpg
```

```
results = {}
```

```
Saving 124_158.jpg to 124_158.jpg
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
ACC is  0.860
```

```
Saving 130_78.jpg to 130_78.jpg
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.860
```

```
Saving 134_999.jpg to 134_999.jpg
```

```
# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.860
```

```
Saving 143_27.jpg to 143_27.jpg
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.860
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.860
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.860
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.720
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_ResNet152V2.h5')
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask layers require a config and must
  layer_config = serialize_layer_fn(layer)
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 resnet152v2 (Functional)    (None, 5, 5, 2048)        58331648

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 58,333,697
Trainable params: 56,396,545
Non-trainable params: 1,937,152
_________________________________________________________________
```

End of Notebook

This is a notebook for training, validation and testing a binary classifier with InceptionV3 as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection is lost
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'aba
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

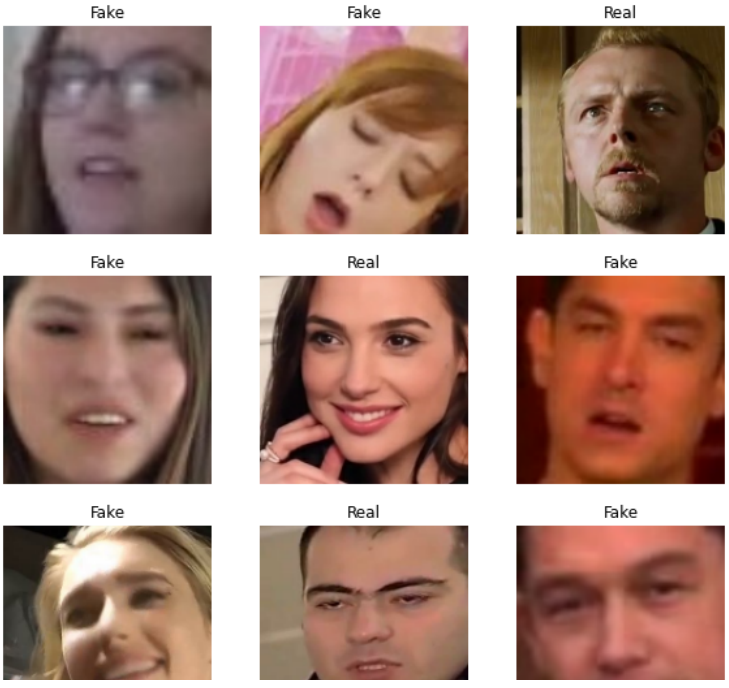
Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```




```
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```

Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.inception_v3.preprocess_input
```

```
# Create the base model from the pre-trained model
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.inception_v3.InceptionV3(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf
87916544/87910968 [==============================] - 1s 0us/step
87924736/87910968 [==============================] - 1s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 3, 3, 2048)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "inception_v3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 150, 150, 3  0           []
```

```
                                )]

conv2d (Conv2D)                 (None, 74, 74, 32)   864         ['input_1[0][0]']

batch_normalization (BatchNorm  (None, 74, 74, 32)  96          ['conv2d[0][0]']
alization)

activation (Activation)         (None, 74, 74, 32)   0           ['batch_normalization[0][0]']

conv2d_1 (Conv2D)               (None, 72, 72, 32)   9216        ['activation[0][0]']

batch_normalization_1 (BatchNo  (None, 72, 72, 32)  96          ['conv2d_1[0][0]']
rmalization)

activation_1 (Activation)       (None, 72, 72, 32)   0           ['batch_normalization_1[0][0]']

conv2d_2 (Conv2D)               (None, 72, 72, 64)   18432       ['activation_1[0][0]']

batch_normalization_2 (BatchNo  (None, 72, 72, 64)  192         ['conv2d_2[0][0]']
rmalization)

activation_2 (Activation)       (None, 72, 72, 64)   0           ['batch_normalization_2[0][0]']

max_pooling2d (MaxPooling2D)    (None, 35, 35, 64)   0           ['activation_2[0][0]']

conv2d_3 (Conv2D)               (None, 35, 35, 80)   5120        ['max_pooling2d[0][0]']

batch_normalization_3 (BatchNo  (None, 35, 35, 80)  240         ['conv2d_3[0][0]']
rmalization)

activation_3 (Activation)       (None, 35, 35, 80)   0           ['batch_normalization_3[0][0]']

conv2d_4 (Conv2D)               (None, 33, 33, 192)  138240      ['activation_3[0][0]']

batch_normalization_4 (BatchNo  (None, 33, 33, 192)  576         ['conv2d_4[0][0]']
rmalization)

activation_4 (Activation)       (None, 33, 33, 192)  0           ['batch_normalization_4[0][0]']

max_pooling2d_1 (MaxPooling2D)  (None, 16, 16, 192)  0          ['activation_4[0][0]']

conv2d_8 (Conv2D)               (None, 16, 16, 64)   12288       ['max_pooling2d_1[0][0]']

batch_normalization_8 (BatchNo  (None, 16, 16, 64)  192         ['conv2d_8[0][0]']
rmalization)

activation_8 (Activation)       (None, 16, 16, 64)   0           ['batch_normalization_8[0][0]']

conv2d_6 (Conv2D)               (None, 16, 16, 48)   9216        ['max_pooling2d_1[0][0]']

conv2d_9 (Conv2D)               (None, 16, 16, 96)   55296       ['activation_8[0][0]']

batch_normalization_6 (BatchNo  (None, 16, 16, 48)  144         ['conv2d_6[0][0]']
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 2048)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(150, 150, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 inception_v3 (Functional)   (None, 3, 3, 2048)        21802784

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 21,804,833
Trainable params: 2,049
Non-trainable params: 21,802,784
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 455s 5s/step - loss: 0.8346 - accuracy: 0.5185
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.83
initial accuracy: 0.52
```

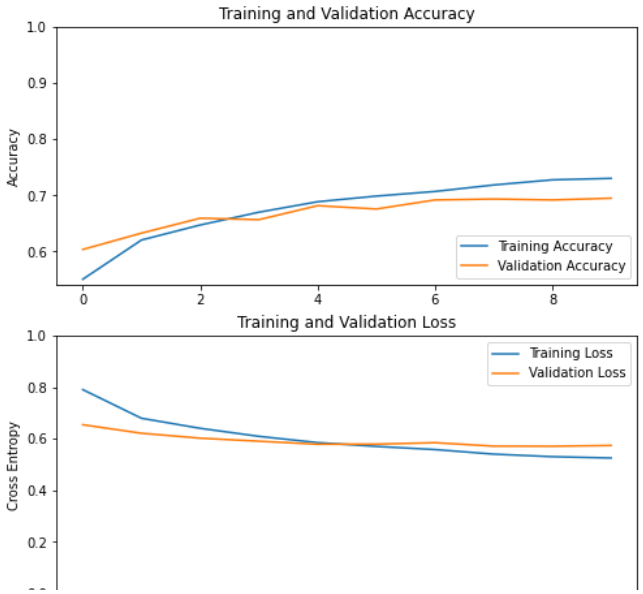
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 1823s 5s/step - loss: 0.7910 - accuracy: 0.5505 - val_loss: 0.6546 - val_accuracy: 0.6036
Epoch 2/10
364/364 [==============================] - 35s 94ms/step - loss: 0.6796 - accuracy: 0.6204 - val_loss: 0.6214 - val_accuracy: 0.6327
Epoch 3/10
364/364 [==============================] - 35s 94ms/step - loss: 0.6406 - accuracy: 0.6472 - val_loss: 0.6023 - val_accuracy: 0.6592
Epoch 4/10
364/364 [==============================] - 33s 89ms/step - loss: 0.6095 - accuracy: 0.6699 - val_loss: 0.5906 - val_accuracy: 0.6567
Epoch 5/10
364/364 [==============================] - 33s 89ms/step - loss: 0.5851 - accuracy: 0.6886 - val_loss: 0.5788 - val_accuracy: 0.6815
Epoch 6/10
364/364 [==============================] - 34s 93ms/step - loss: 0.5703 - accuracy: 0.6986 - val_loss: 0.5792 - val_accuracy: 0.6756
Epoch 7/10
364/364 [==============================] - 34s 91ms/step - loss: 0.5581 - accuracy: 0.7069 - val_loss: 0.5848 - val_accuracy: 0.6917
Epoch 8/10
364/364 [==============================] - 34s 92ms/step - loss: 0.5403 - accuracy: 0.7184 - val_loss: 0.5714 - val_accuracy: 0.6934
Epoch 9/10
364/364 [==============================] - 35s 95ms/step - loss: 0.5305 - accuracy: 0.7276 - val_loss: 0.5710 - val_accuracy: 0.6917
Epoch 10/10
364/364 [==============================] - 35s 96ms/step - loss: 0.5254 - accuracy: 0.7302 - val_loss: 0.5740 - val_accuracy: 0.6948
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
```

```
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  311
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )
```

```
 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 inception_v3 (Functional)   (None, 3, 3, 2048)        21802784

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 21,804,833
Trainable params: 19,628,417
Non-trainable params: 2,176,416
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
130
```

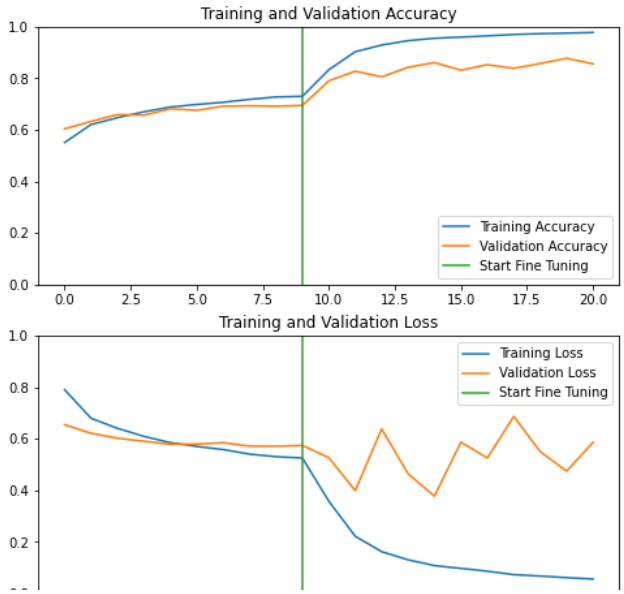
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
364/364 [==============================] - 50s 113ms/step - loss: 0.3575 - accuracy: 0.8333 - val_loss: 0.5267 - val_accuracy: 0.7901
Epoch 11/20
364/364 [==============================] - 40s 110ms/step - loss: 0.2213 - accuracy: 0.9033 - val_loss: 0.3983 - val_accuracy: 0.8276
Epoch 12/20
364/364 [==============================] - 40s 109ms/step - loss: 0.1616 - accuracy: 0.9293 - val_loss: 0.6383 - val_accuracy: 0.8056
Epoch 13/20
364/364 [==============================] - 41s 112ms/step - loss: 0.1298 - accuracy: 0.9463 - val_loss: 0.4640 - val_accuracy: 0.8429
Epoch 14/20
364/364 [==============================] - 41s 112ms/step - loss: 0.1073 - accuracy: 0.9556 - val_loss: 0.3771 - val_accuracy: 0.8613
Epoch 15/20
364/364 [==============================] - 41s 111ms/step - loss: 0.0966 - accuracy: 0.9601 - val_loss: 0.5869 - val_accuracy: 0.8316
Epoch 16/20
364/364 [==============================] - 40s 108ms/step - loss: 0.0855 - accuracy: 0.9650 - val_loss: 0.5253 - val_accuracy: 0.8531
Epoch 17/20
364/364 [==============================] - 41s 111ms/step - loss: 0.0722 - accuracy: 0.9704 - val_loss: 0.6869 - val_accuracy: 0.8389
Epoch 18/20
364/364 [==============================] - 42s 113ms/step - loss: 0.0669 - accuracy: 0.9734 - val_loss: 0.5499 - val_accuracy: 0.8576
Epoch 19/20
364/364 [==============================] - 41s 112ms/step - loss: 0.0604 - accuracy: 0.9757 - val_loss: 0.4742 - val_accuracy: 0.8776
Epoch 20/20
364/364 [==============================] - 41s 110ms/step - loss: 0.0552 - accuracy: 0.9780 - val_loss: 0.5863 - val_accuracy: 0.8565
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

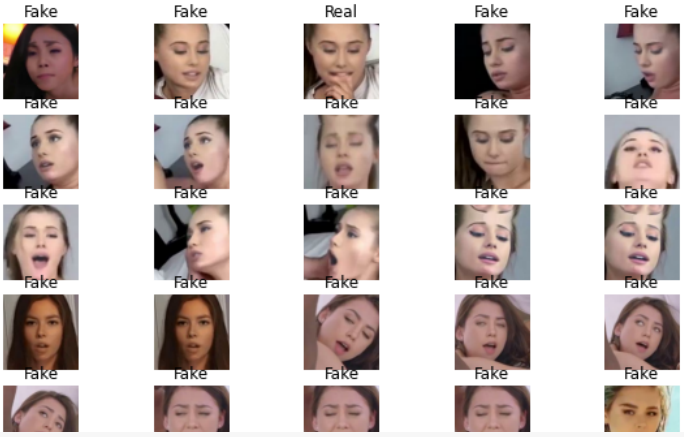
```
5/5 [==============================] - 24s 5s/step - loss: 0.7242 - accuracy: 0.8200
Test accuracy : 0.8199999928474426
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
 0 0 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 34_0.jpg to 34_0.jpg
Saving 34_6.jpg to 34_6.jpg
Saving 35_0.jpg to 35_0.jpg
Saving 39_360.jpg to 39_360.jpg
Saving 39_372.jpg to 39_372.jpg
Saving 39_384.jpg to 39_384.jpg
Saving 39_402.jpg to 39_402.jpg
Saving 39_780.jpg to 39_780.jpg
Saving 39_786.jpg to 39_786.jpg
Saving 40_0.jpg to 40_0.jpg
Saving 40_12.jpg to 40_12.jpg
Saving 40_24.jpg to 40_24.jpg
Saving 42_168.jpg to 42_168.jpg
Saving 42_198.jpg to 42_198.jpg
Saving 42_204.jpg to 42_204.jpg
Saving 43_36.jpg to 43_36.jpg
Saving 43_150.jpg to 43_150.jpg
Saving 43_192.jpg to 43_192.jpg
Saving 43_210.jpg to 43_210.jpg
Saving 44_6.jpg to 44_6.jpg
Saving 44_90.jpg to 44_90.jpg
Saving 44_192.jpg to 44_192.jpg
Saving 44_198.jpg to 44_198.jpg
Saving 45_60.jpg to 45_60.jpg
Saving 45_150.jpg to 45_150.jpg
Saving 46_360.jpg to 46_360.jpg
Saving 46_384.jpg to 46_384.jpg
Saving 47_528.jpg to 47_528.jpg
Saving 47_738.jpg to 47_738.jpg
Saving 48_12.jpg to 48_12.jpg
Saving 48_270.jpg to 48_270.jpg
Saving 48_318.jpg to 48_318.jpg
Saving 48_522.jpg to 48_522.jpg
Saving 48_708.jpg to 48_708.jpg
Saving 50_192.jpg to 50_192.jpg
Saving 50_222.jpg to 50_222.jpg
Saving 51_120.jpg to 51_120.jpg
Saving 51_300.jpg to 51_300.jpg
Saving 51_372.jpg to 51_372.jpg
Saving 52_42.jpg to 52_42.jpg
Saving 52_450.jpg to 52_450.jpg
Saving 53_66.jpg to 53_66.jpg
Saving 54_18.jpg to 54_18.jpg
Saving 54_186.jpg to 54_186.jpg
Saving 69_268.jpg to 69_268.jpg
Saving 69_302.jpg to 69_302.jpg
Saving 78_3.jpg to 78_3.jpg
Saving 78_318.jpg to 78_318.jpg
Saving 96_30.jpg to 96_30.jpg
```

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image
```

```
uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 114_666.jpg to 114_666.jpg
Saving 115_306.jpg to 115_306.jpg
Saving 115_816.jpg to 115_816.jpg
Saving 116_24.jpg to 116_24.jpg
Saving 116_1056.jpg to 116_1056.jpg
Saving 117_12.jpg to 117_12.jpg
Saving 117_750.jpg to 117_750.jpg
Saving 118_9.jpg to 118_9.jpg
Saving 118_405.jpg to 118_405.jpg
Saving 119_65.jpg to 119_65.jpg
Saving 119_660.jpg to 119_660.jpg
Saving 120_50.jpg to 120_50.jpg
Saving 120_235.jpg to 120_235.jpg
Saving 121_24.jpg to 121_24.jpg
Saving 121_28.jpg to 121_28.jpg
Saving 122_0.jpg to 122_0.jpg
Saving 122_1.jpg to 122_1.jpg
Saving 124_154.jpg to 124_154.jpg
Saving 124_158.jpg to 124_158.jpg
Saving 124_160.jpg to 124_160.jpg
Saving 124_170.jpg to 124_170.jpg
Saving 125_65.jpg to 125_65.jpg
Saving 125_68.jpg to 125_68.jpg
Saving 125_69.jpg to 125_69.jpg
Saving 130_2.jpg to 130_2.jpg
Saving 130_34.jpg to 130_34.jpg
Saving 130_50.jpg to 130_50.jpg
Saving 130_78.jpg to 130_78.jpg
Saving 131_172.jpg to 131_172.jpg
Saving 131_212.jpg to 131_212.jpg
Saving 132_36.jpg to 132_36.jpg
Saving 132_105.jpg to 132_105.jpg
Saving 132_237.jpg to 132_237.jpg
Saving 133_144.jpg to 133_144.jpg
Saving 134_90.jpg to 134_90.jpg
Saving 134_190.jpg to 134_190.jpg
Saving 134_555.jpg to 134_555.jpg
Saving 135_370.jpg to 135_370.jpg
Saving 135_830.jpg to 135_830.jpg
Saving 135_1080.jpg to 135_1080.jpg
Saving 142_396.jpg to 142_396.jpg
Saving 142_412.jpg to 142_412.jpg
Saving 142_420.jpg to 142_420.jpg
Saving 143_23.jpg to 143_23.jpg
Saving 143_25.jpg to 143_25.jpg
Saving 143_26.jpg to 143_26.jpg
Saving 143_27.jpg to 143_27.jpg
Saving 143_29.jpg to 143_29.jpg
Saving 143_32.jpg to 143_32.jpg
```

Step 10 - Determine the performance of the classifier

```
# Figures for performance calculation
# real = 1, fake = 0
TP = true_pos = 74 # real faces predicted as real
TN = true_neg = 86 # fake faces predicted as fake
FP = false_pos = 14 # fake faces predicted as real
FN = false_neg = 26 # real faces predicted as fake
```

```
results = {}
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
     ACC is  0.800
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
     TPR is  0.740
```

```
# True Negative Rate
```

```
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.860
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.841
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.768
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.787
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.604
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_InceptionNetV3.h5')
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 inception_v3 (Functional)   (None, 3, 3, 2048)        21802784

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049
```

```
==========================================================================
Total params: 21,804,833
Trainable params: 19,628,417
Non-trainable params: 2,176,416
__________________________________________________________________________
```

End of Notebook

This is a notebook for training, validation and testing a binary classifier with InceptionResNet as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection is lost
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'ab
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

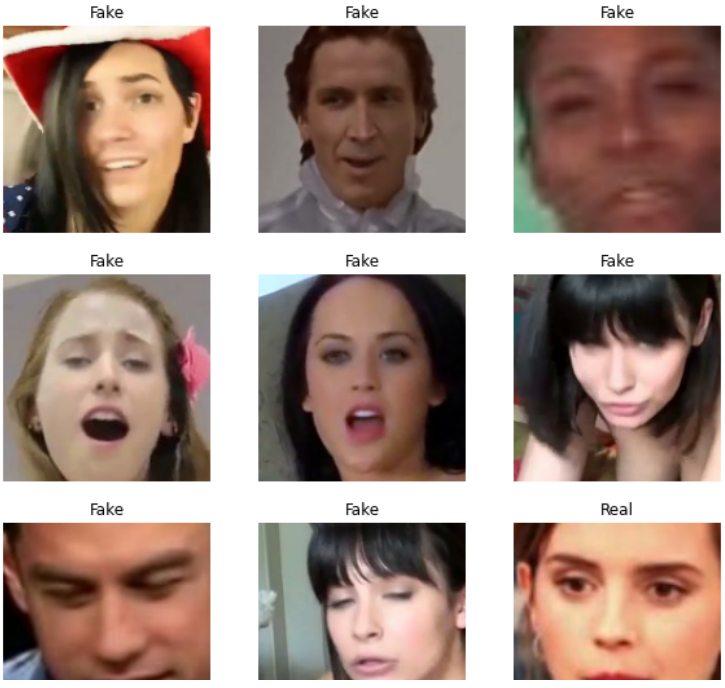
Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

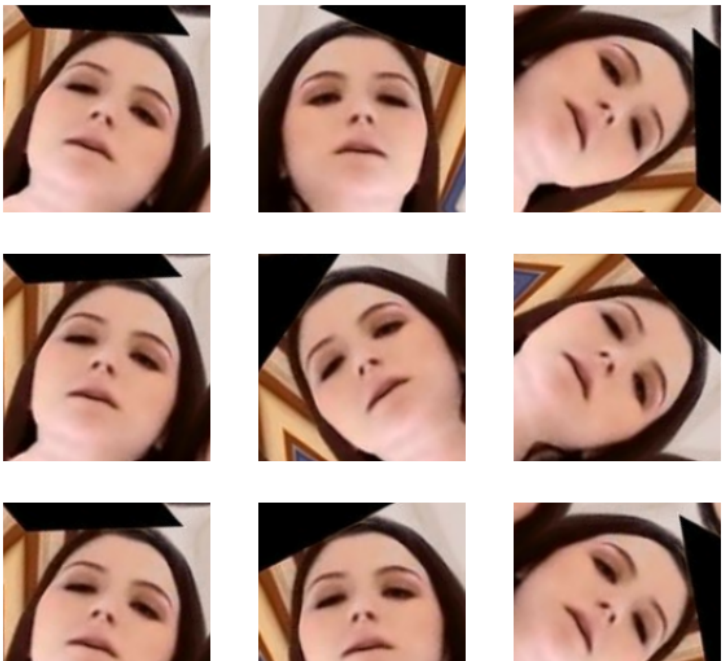



```
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```



Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.inception_resnet_v2.preprocess_input
```

```
# Create the base model from the pre-trained model
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.InceptionResNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_v2_weights_tf_
219062272/219055592 [==============================] - 2s 0us/step
219070464/219055592 [==============================] - 2s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 3, 3, 1536)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "inception_resnet_v2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 150, 150, 3  0           []
```

```
                                )]

 conv2d (Conv2D)                (None, 74, 74, 32)   864         ['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 74, 74, 32)  96          ['conv2d[0][0]']
 alization)

 activation (Activation)        (None, 74, 74, 32)   0           ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)              (None, 72, 72, 32)   9216        ['activation[0][0]']

 batch_normalization_1 (BatchNo  (None, 72, 72, 32)  96          ['conv2d_1[0][0]']
 rmalization)

 activation_1 (Activation)      (None, 72, 72, 32)   0           ['batch_normalization_1[0][0]']

 conv2d_2 (Conv2D)              (None, 72, 72, 64)   18432       ['activation_1[0][0]']

 batch_normalization_2 (BatchNo  (None, 72, 72, 64)  192         ['conv2d_2[0][0]']
 rmalization)

 activation_2 (Activation)      (None, 72, 72, 64)   0           ['batch_normalization_2[0][0]']

 max_pooling2d (MaxPooling2D)   (None, 35, 35, 64)   0           ['activation_2[0][0]']

 conv2d_3 (Conv2D)              (None, 35, 35, 80)   5120        ['max_pooling2d[0][0]']

 batch_normalization_3 (BatchNo  (None, 35, 35, 80)  240         ['conv2d_3[0][0]']
 rmalization)

 activation_3 (Activation)      (None, 35, 35, 80)   0           ['batch_normalization_3[0][0]']

 conv2d_4 (Conv2D)              (None, 33, 33, 192)  138240      ['activation_3[0][0]']

 batch_normalization_4 (BatchNo  (None, 33, 33, 192)  576        ['conv2d_4[0][0]']
 rmalization)

 activation_4 (Activation)      (None, 33, 33, 192)  0           ['batch_normalization_4[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 16, 16, 192)  0         ['activation_4[0][0]']

 conv2d_8 (Conv2D)              (None, 16, 16, 64)   12288       ['max_pooling2d_1[0][0]']

 batch_normalization_8 (BatchNo  (None, 16, 16, 64)  192         ['conv2d_8[0][0]']
 rmalization)

 activation_8 (Activation)      (None, 16, 16, 64)   0           ['batch_normalization_8[0][0]']

 conv2d_6 (Conv2D)              (None, 16, 16, 48)   9216        ['max_pooling2d_1[0][0]']

 conv2d_9 (Conv2D)              (None, 16, 16, 96)   55296       ['activation_8[0][0]']

 batch_normalization_6 (BatchNo  (None, 16, 16, 48)  144         ['conv2d_6[0][0]']
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 1536)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(150, 150, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 inception_resnet_v2 (Functi  (None, 3, 3, 1536)       54336736
 onal)

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1536)              0

 dense (Dense)               (None, 1)                 1537

=================================================================
Total params: 54,338,273
Trainable params: 1,537
Non-trainable params: 54,336,736
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 515s 5s/step - loss: 0.9471 - accuracy: 0.4936
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.95
initial accuracy: 0.49
```

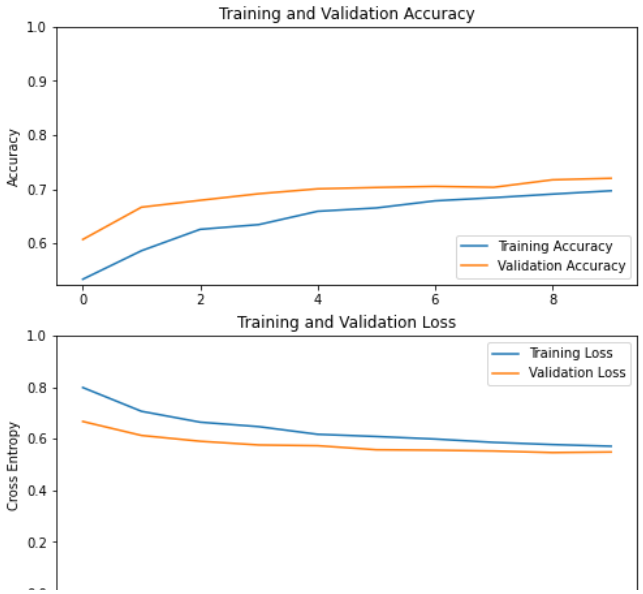
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 2109s 6s/step - loss: 0.7991 - accuracy: 0.5337 - val_loss: 0.6672 - val_accuracy: 0.6072
Epoch 2/10
364/364 [==============================] - 56s 151ms/step - loss: 0.7067 - accuracy: 0.5864 - val_loss: 0.6129 - val_accuracy: 0.6669
Epoch 3/10
364/364 [==============================] - 55s 151ms/step - loss: 0.6645 - accuracy: 0.6260 - val_loss: 0.5903 - val_accuracy: 0.6796
Epoch 4/10
364/364 [==============================] - 55s 149ms/step - loss: 0.6471 - accuracy: 0.6347 - val_loss: 0.5759 - val_accuracy: 0.6917
Epoch 5/10
364/364 [==============================] - 55s 151ms/step - loss: 0.6173 - accuracy: 0.6593 - val_loss: 0.5731 - val_accuracy: 0.7008
Epoch 6/10
364/364 [==============================] - 55s 150ms/step - loss: 0.6088 - accuracy: 0.6655 - val_loss: 0.5573 - val_accuracy: 0.7033
Epoch 7/10
364/364 [==============================] - 56s 152ms/step - loss: 0.5990 - accuracy: 0.6786 - val_loss: 0.5556 - val_accuracy: 0.7053
Epoch 8/10
364/364 [==============================] - 56s 151ms/step - loss: 0.5860 - accuracy: 0.6844 - val_loss: 0.5524 - val_accuracy: 0.7036
Epoch 9/10
364/364 [==============================] - 56s 151ms/step - loss: 0.5775 - accuracy: 0.6910 - val_loss: 0.5463 - val_accuracy: 0.7174
Epoch 10/10
364/364 [==============================] - 55s 150ms/step - loss: 0.5711 - accuracy: 0.6972 - val_loss: 0.5485 - val_accuracy: 0.7203
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  780
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )
```

```
 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 inception_resnet_v2 (Functi  (None, 3, 3, 1536)       54336736
 onal)

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1536)              0

 dense (Dense)               (None, 1)                 1537

=================================================================
Total params: 54,338,273
Trainable params: 53,510,161
Non-trainable params: 828,112
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
426
```

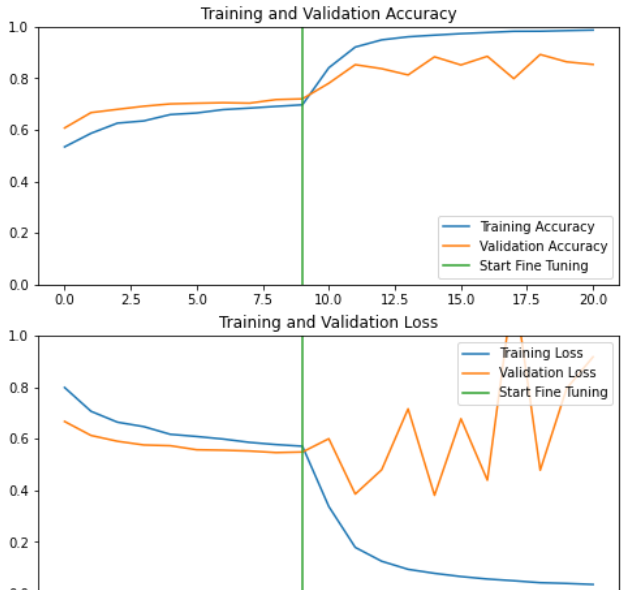
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
364/364 [==============================] - 141s 295ms/step - loss: 0.3369 - accuracy: 0.8411 - val_loss: 0.6002 - val_accuracy: 0.7810
Epoch 11/20
364/364 [==============================] - 104s 284ms/step - loss: 0.1782 - accuracy: 0.9215 - val_loss: 0.3853 - val_accuracy: 0.8531
Epoch 12/20
364/364 [==============================] - 105s 286ms/step - loss: 0.1243 - accuracy: 0.9494 - val_loss: 0.4796 - val_accuracy: 0.8375
Epoch 13/20
364/364 [==============================] - 105s 286ms/step - loss: 0.0929 - accuracy: 0.9614 - val_loss: 0.7163 - val_accuracy: 0.8132
Epoch 14/20
364/364 [==============================] - 104s 285ms/step - loss: 0.0773 - accuracy: 0.9678 - val_loss: 0.3804 - val_accuracy: 0.8836
Epoch 15/20
364/364 [==============================] - 104s 285ms/step - loss: 0.0649 - accuracy: 0.9736 - val_loss: 0.6777 - val_accuracy: 0.8519
Epoch 16/20
364/364 [==============================] - 105s 287ms/step - loss: 0.0553 - accuracy: 0.9781 - val_loss: 0.4391 - val_accuracy: 0.8856
Epoch 17/20
364/364 [==============================] - 105s 286ms/step - loss: 0.0485 - accuracy: 0.9828 - val_loss: 1.1971 - val_accuracy: 0.7988
Epoch 18/20
364/364 [==============================] - 104s 285ms/step - loss: 0.0410 - accuracy: 0.9831 - val_loss: 0.4779 - val_accuracy: 0.8926
Epoch 19/20
364/364 [==============================] - 104s 283ms/step - loss: 0.0382 - accuracy: 0.9855 - val_loss: 0.7953 - val_accuracy: 0.8641
Epoch 20/20
364/364 [==============================] - 104s 283ms/step - loss: 0.0341 - accuracy: 0.9874 - val_loss: 0.9180 - val_accuracy: 0.8539
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

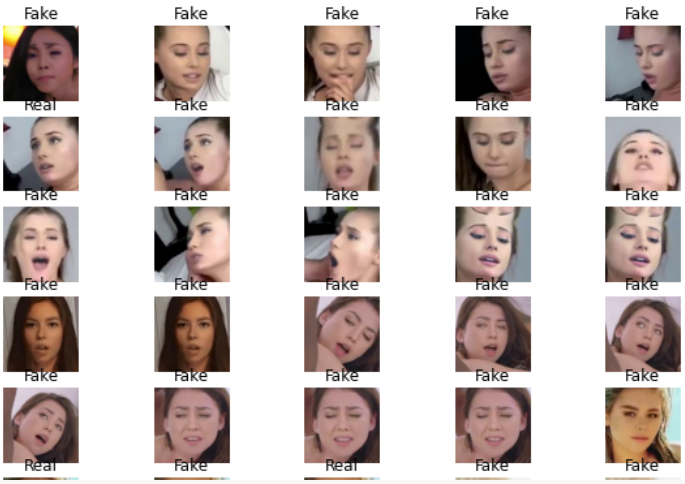
```
5/5 [==============================] - 29s 6s/step - loss: 1.9864 - accuracy: 0.7350
Test accuracy : 0.7350000143051147
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0
 0 0 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 34_0.jpg to 34_0.jpg
Saving 34_6.jpg to 34_6.jpg
Saving 35_0.jpg to 35_0.jpg
Saving 39_360.jpg to 39_360.jpg
Saving 39_372.jpg to 39_372.jpg
Saving 39_384.jpg to 39_384.jpg
Saving 39_402.jpg to 39_402.jpg
Saving 39_780.jpg to 39_780.jpg
Saving 39_786.jpg to 39_786.jpg
Saving 40_0.jpg to 40_0.jpg
Saving 40_12.jpg to 40_12.jpg
Saving 40_24.jpg to 40_24.jpg
Saving 42_168.jpg to 42_168.jpg
Saving 42_198.jpg to 42_198.jpg
Saving 42_204.jpg to 42_204.jpg
Saving 43_36.jpg to 43_36.jpg
Saving 43_150.jpg to 43_150.jpg
Saving 43_192.jpg to 43_192.jpg
Saving 43_210.jpg to 43_210.jpg
Saving 44_6.jpg to 44_6.jpg
Saving 44_90.jpg to 44_90.jpg
Saving 44_192.jpg to 44_192.jpg
Saving 44_198.jpg to 44_198.jpg
Saving 45_60.jpg to 45_60.jpg
Saving 45_150.jpg to 45_150.jpg
Saving 46_360.jpg to 46_360.jpg
Saving 46_384.jpg to 46_384.jpg
Saving 47_528.jpg to 47_528.jpg
Saving 47_738.jpg to 47_738.jpg
Saving 48_12.jpg to 48_12.jpg
Saving 48_270.jpg to 48_270.jpg
Saving 48_318.jpg to 48_318.jpg
Saving 48_522.jpg to 48_522.jpg
Saving 48_708.jpg to 48_708.jpg
Saving 50_192.jpg to 50_192.jpg
Saving 50_222.jpg to 50_222.jpg
Saving 51_120.jpg to 51_120.jpg
Saving 51_300.jpg to 51_300.jpg
Saving 51_372.jpg to 51_372.jpg
Saving 52_42.jpg to 52_42.jpg
Saving 52_450.jpg to 52_450.jpg
Saving 53_66.jpg to 53_66.jpg
Saving 54_18.jpg to 54_18.jpg
Saving 54_186.jpg to 54_186.jpg
Saving 69_268.jpg to 69_268.jpg
Saving 69_302.jpg to 69_302.jpg
Saving 78_3.jpg to 78_3.jpg
Saving 78_318.jpg to 78_318.jpg
Saving 96_30.jpg to 96_30.jpg
Saving 96_32.jpg to 96_32.jpg
Saving 97_137.jpg to 97_137.jpg
Saving 97_146.jpg to 97_146.jpg
Saving 98_62.jpg to 98_62.jpg
Saving 99_108.jpg to 99_108.jpg
Saving 100_168.jpg to 100_168.jpg
Saving 100_956.jpg to 100_956.jpg
Saving 100_972.jpg to 100_972.jpg
Saving 101_6.jpg to 101_6.jpg
Saving 101_8.jpg to 101_8.jpg
Saving 101_10.jpg to 101_10.jpg
Saving 101_40.jpg to 101_40.jpg
Saving 101_488.jpg to 101_488.jpg
Saving 101_490.jpg to 101_490.jpg
Saving 102_14.jpg to 102_14.jpg
Saving 102_16.jpg to 102_16.jpg
Saving 102_320.jpg to 102_320.jpg
Saving id19_0000.mp40.jpgFace.jpg to id19_0000.mp40.jpgFace.jpg
Saving id19_0001.mp4288.jpgFace.jpg to id19_0001.mp4288.jpgFace.jpg
Saving id19_0001.mp4312.jpgFace.jpg to id19_0001.mp4312.jpgFace.jpg
Saving id19_0005.mp4312.jpgFace.jpg to id19_0005.mp4312.jpgFace.jpg
Saving id19_0006.mp40.jpgFace.jpg to id19_0006.mp40.jpgFace.jpg
Saving id19_0006.mp424.jpgFace.jpg to id19_0006.mp424.jpgFace.jpg
Saving id19_0006.mp448.jpgFace.jpg to id19_0006.mp448.jpgFace.jpg
Saving id19_0006.mp472.jpgFace.jpg to id19_0006.mp472.jpgFace.jpg
Saving id19_0006.mp496.jpgFace.jpg to id19_0006.mp496.jpgFace.jpg
Saving id19_0006.mp4120.jpgFace.jpg to id19_0006.mp4120.jpgFace.jpg
Saving id19_0006.mp4144.jpgFace.jpg to id19_0006.mp4144.jpgFace.jpg
Saving id20_0002.mp424.jpgFace.jpg to id20_0002.mp424.jpgFace.jpg
Saving id20_0002.mp448.jpgFace.jpg to id20_0002.mp448.jpgFace.jpg
Saving id20_0002.mp472.jpgFace.jpg to id20_0002.mp472.jpgFace.jpg
Saving id20_0002.mp496.jpgFace.jpg to id20_0002.mp496.jpgFace.jpg
Saving id20_0002.mp4120.jpgFace.jpg to id20_0002.mp4120.jpgFace.jpg
Saving id20_0002.mp4144.jpgFace.jpg to id20_0002.mp4144.jpgFace.jpg
Saving id23_0007.mp4144.jpgFace.jpg to id23_0007.mp4144.jpgFace.jpg
Saving id23_0008.mp4168.jpgFace.jpg to id23_0008.mp4168.jpgFace.jpg
Saving id23_0008.mp4192.jpgFace.jpg to id23_0008.mp4192.jpgFace.jpg
Saving id23_0008.mp4216.jpgFace.jpg to id23_0008.mp4216.jpgFace.jpg
Saving id23_0008.mp4240.jpgFace.jpg to id23_0008.mp4240.jpgFace.jpg
Saving id23_0008.mp4264.jpgFace.jpg to id23_0008.mp4264.jpgFace.jpg
Saving id23_0008.mp4288.jpgFace.jpg to id23_0008.mp4288.jpgFace.jpg
```

```
Saving id23_0009.mp40.jpgFace.jpg to id23_0009.mp40.jpgFace.jpg
Saving id23_0009.mp424.jpgFace.jpg to id23_0009.mp424.jpgFace.jpg
Saving id23_0009.mp448.jpgFace.jpg to id23_0009.mp448.jpgFace.jpg
Saving id23_0009.mp472.jpgFace.jpg to id23_0009.mp472.jpgFace.jpg
Saving id24_0003.mp40.jpgFace.jpg to id24_0003.mp40.jpgFace.jpg
Saving id24_0003.mp472.jpgFace.jpg to id24_0003.mp472.jpgFace.jpg
Saving id24_0003.mp4120.jpgFace.jpg to id24_0003.mp4120.jpgFace.jpg
Saving id24_0003.mp4144.jpgFace.jpg to id24_0003.mp4144.jpgFace.jpg
Saving id25_0004.mp4288.jpgFace.jpg to id25_0004.mp4288.jpgFace.jpg
Saving id25_0004.mp4312.jpgFace.jpg to id25_0004.mp4312.jpgFace.jpg
34_0.jpg is deepfake
34_6.jpg is deepfake
35_0.jpg is real
39_360.jpg is real
39_372.jpg is real
39_384.jpg is real
39_402.jpg is deepfake
39_780.jpg is deepfake
39_786.jpg is deepfake
40_0.jpg is real
40_12.jpg is real
40_24.jpg is real
42_168.jpg is deepfake
42_198.jpg is real
42_204.jpg is real
43_36.jpg is real
43_150.jpg is deepfake
43_192.jpg is deepfake
43_210.jpg is deepfake
44_6.jpg is deepfake
44_90.jpg is deepfake
44_192.jpg is deepfake
44_198.jpg is deepfake
45_60.jpg is deepfake
45_150.jpg is deepfake
46_360.jpg is deepfake
46_384.jpg is deepfake
47_528.jpg is real
47_738.jpg is real
48_12.jpg is real
48_270.jpg is deepfake
48_318.jpg is real
48_522.jpg is real
48_708.jpg is real
50_192.jpg is real
50_222.jpg is real
51_120.jpg is deepfake
51_300.jpg is real
51_372.jpg is real
52_42.jpg is real
52_450.jpg is real
53_66.jpg is real
54_18.jpg is real
54_186.jpg is real
69_268.jpg is real
69_302.jpg is real
78_3.jpg is real
78_318.jpg is real
96_30.jpg is real
96_32.jpg is real
97_137.jpg is real
97_146.jpg is real
98_62.jpg is deepfake
99_108.jpg is deepfake
100_168.jpg is deepfake
100_956.jpg is real
100_972.jpg is real
101_6.jpg is real
101_8.jpg is real
101_10.jpg is real
101_40.jpg is real
101_488.jpg is real
101_490.jpg is real
102_14.jpg is real
102_16.jpg is real
102_320.jpg is deepfake
id19_0000.mp40.jpgFace.jpg is real
id19_0001.mp4288.jpgFace.jpg is deepfake
id19_0001.mp4312.jpgFace.jpg is deepfake
id19_0005.mp4312.jpgFace.jpg is real
id19_0006.mp40.jpgFace.jpg is deepfake
id19_0006.mp424.jpgFace.jpg is deepfake
id19_0006.mp448.jpgFace.jpg is deepfake
id19_0006.mp472.jpgFace.jpg is deepfake
id19_0006.mp496.jpgFace.jpg is deepfake
id19_0006.mp4120.jpgFace.jpg is deepfake
id19_0006.mp4144.jpgFace.jpg is deepfake
id20_0002.mp424.jpgFace.jpg is deepfake
id20_0002.mp448.jpgFace.jpg is deepfake
id20_0002.mp472.jpgFace.jpg is deepfake
id20_0002.mp496.jpgFace.jpg is deepfake
id20_0002.mp4120.jpgFace.jpg is deepfake
id20_0002.mp4144.jpgFace.jpg is deepfake
```

```
id23_0007.mp4144.jpgFace.jpg is deepfake
id23_0008.mp4168.jpgFace.jpg is deepfake
id23_0008.mp4192.jpgFace.jpg is deepfake
id23_0008.mp4216.jpgFace.jpg is deepfake
id23_0008.mp4240.jpgFace.jpg is deepfake
id23_0008.mp4264.jpgFace.jpg is deepfake
id23_0008.mp4288.jpgFace.jpg is real
id23_0009.mp40.jpgFace.jpg is real
id23_0009.mp424.jpgFace.jpg is real
id23_0009.mp448.jpgFace.jpg is real
id23_0009.mp472.jpgFace.jpg is real
id24_0003.mp40.jpgFace.jpg is deepfake
id24_0003.mp472.jpgFace.jpg is deepfake
id24_0003.mp4120.jpgFace.jpg is deepfake
id24_0003.mp4144.jpgFace.jpg is deepfake
id25_0004.mp4288.jpgFace.jpg is deepfake
id25_0004.mp4312.jpgFace.jpg is deepfake
```

```
# Upload the deepfake faces to make prediction
import numpy as np
```

```
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 114_666.jpg to 114_666.jpg
Saving 115_306.jpg to 115_306.jpg
Saving 115_816.jpg to 115_816.jpg
Saving 116_24.jpg to 116_24.jpg
Saving 116_1056.jpg to 116_1056.jpg
Saving 117_12.jpg to 117_12.jpg
Saving 117_750.jpg to 117_750.jpg
Saving 118_9.jpg to 118_9.jpg
Saving 118_405.jpg to 118_405.jpg
Saving 119_65.jpg to 119_65.jpg
Saving 119_660.jpg to 119_660.jpg
Saving 120_50.jpg to 120_50.jpg
Saving 120_235.jpg to 120_235.jpg
Saving 121_24.jpg to 121_24.jpg
Saving 121_28.jpg to 121_28.jpg
Saving 122_0.jpg to 122_0.jpg
Saving 122_1.jpg to 122_1.jpg
Saving 124_154.jpg to 124_154.jpg
Saving 124_158.jpg to 124_158.jpg
Saving 124_160.jpg to 124_160.jpg
Saving 124_170.jpg to 124_170.jpg
Saving 125_65.jpg to 125_65.jpg
Saving 125_68.jpg to 125_68.jpg
Saving 125_69.jpg to 125_69.jpg
Saving 130_2.jpg to 130_2.jpg
Saving 130_34.jpg to 130_34.jpg
Saving 130_50.jpg to 130_50.jpg
Saving 130_78.jpg to 130_78.jpg
Saving 131_172.jpg to 131_172.jpg
Saving 131_212.jpg to 131_212.jpg
Saving 132_36.jpg to 132_36.jpg
Saving 132_105.jpg to 132_105.jpg
Saving 132_237.jpg to 132_237.jpg
Saving 133_144.jpg to 133_144.jpg
Saving 134_90.jpg to 134_90.jpg
Saving 134_190.jpg to 134_190.jpg
Saving 134_555.jpg to 134_555.jpg
Saving 135_370.jpg to 135_370.jpg
Saving 135_830.jpg to 135_830.jpg
Saving 135_1080.jpg to 135_1080.jpg
Saving 142_396.jpg to 142_396.jpg
Saving 142_412.jpg to 142_412.jpg
Saving 142_420.jpg to 142_420.jpg
Saving 143_23.jpg to 143_23.jpg
Saving 143_25.jpg to 143_25.jpg
Saving 143_26.jpg to 143_26.jpg
Saving 143_27.jpg to 143_27.jpg
Saving 143_29.jpg to 143_29.jpg
Saving 143_32.jpg to 143_32.jpg
Saving 143_33.jpg to 143_33.jpg
Saving 143_34.jpg to 143_34.jpg
Saving 143_39.jpg to 143_39.jpg
Saving 143_41.jpg to 143_41.jpg
Saving 143_43.jpg to 143_43.jpg
Saving 143_44.jpg to 143_44.jpg
Saving 143_46.jpg to 143_46.jpg
Saving 143_48.jpg to 143_48.jpg
Saving 143_52.jpg to 143_52.jpg
Saving 166_71.jpg to 166_71.jpg
Saving 166_74.jpg to 166_74.jpg
Saving 166_76.jpg to 166_76.jpg
Saving 167_24.jpg to 167_24.jpg
Saving 172_600.jpg to 172_600.jpg
Saving 172_610.jpg to 172_610.jpg
Saving 172_615.jpg to 172_615.jpg
Saving cele16396.jpgF.jpg to cele16396.jpgF.jpg
Saving cele16748.jpgF.jpg to cele16748.jpgF.jpg
Saving cele17100.jpgF.jpg to cele17100.jpgF.jpg
Saving cele17543.jpgF.jpg to cele17543.jpgF.jpg
Saving cele17750.jpgF.jpg to cele17750.jpgF.jpg
Saving cele17783.jpgF.jpg to cele17783.jpgF.jpg
Saving cele18188.jpgF.jpg to cele18188.jpgF.jpg
Saving cele18514.jpgF.jpg to cele18514.jpgF.jpg
Saving cele18909.jpgF.jpg to cele18909.jpgF.jpg
Saving cele33231.jpgF.jpg to cele33231.jpgF.jpg
Saving cele33349.jpgF.jpg to cele33349.jpgF.jpg
Saving cele33353.jpgF.jpg to cele33353.jpgF.jpg
Saving cele33363.jpgF.jpg to cele33363.jpgF.jpg
Saving cele33385.jpgF.jpg to cele33385.jpgF.jpg
Saving cele33393.jpgF.jpg to cele33393.jpgF.jpg
Saving cele33483.jpgF.jpg to cele33483.jpgF.jpg
Saving cele33493.jpgF.jpg to cele33493.jpgF.jpg
Saving cele33499.jpgF.jpg to cele33499.jpgF.jpg
Saving cele33502.jpgF.jpg to cele33502.jpgF.jpg
Saving cele34056.jpgF.jpg to cele34056.jpgF.jpg
Saving cele34064.jpgF.jpg to cele34064.jpgF.jpg
Saving cele34066.jpgF.jpg to cele34066.jpgF.jpg
Saving cele34070.jpgF.jpg to cele34070.jpgF.jpg
Saving cele34073.jpgF.jpg to cele34073.jpgF.jpg
Saving cele35162.jpgF.jpg to cele35162.jpgF.jpg
```

```
Saving cele35173.jpgF.jpg to cele35173.jpgF.jpg
Saving cele35180.jpgF.jpg to cele35180.jpgF.jpg
Saving cele35320.jpgF.jpg to cele35320.jpgF.jpg
Saving cele35328.jpgF.jpg to cele35328.jpgF.jpg
Saving cele35330.jpgF.jpg to cele35330.jpgF.jpg
Saving cele35347.jpgF.jpg to cele35347.jpgF.jpg
Saving cele35353.jpgF.jpg to cele35353.jpgF.jpg
Saving cele35370.jpgF.jpg to cele35370.jpgF.jpg
Saving cele35374.jpgF.jpg to cele35374.jpgF.jpg
Saving cele35383.jpgF.jpg to cele35383.jpgF.jpg
114_666.jpg is deepfake
115_306.jpg is deepfake
115_816.jpg is deepfake
116_24.jpg is deepfake
116_1056.jpg is deepfake
117_12.jpg is real
117_750.jpg is deepfake
118_9.jpg is deepfake
118_405.jpg is deepfake
119_65.jpg is deepfake
119_660.jpg is deepfake
120_50.jpg is deepfake
120_235.jpg is deepfake
121_24.jpg is deepfake
121_28.jpg is deepfake
122_0.jpg is deepfake
122_1.jpg is deepfake
124_154.jpg is deepfake
124_158.jpg is deepfake
124_160.jpg is deepfake
124_170.jpg is deepfake
125_65.jpg is deepfake
125_68.jpg is deepfake
125_69.jpg is deepfake
130_2.jpg is deepfake
130_34.jpg is real
130_50.jpg is deepfake
130_78.jpg is real
131_172.jpg is deepfake
131_212.jpg is deepfake
132_36.jpg is deepfake
132_105.jpg is deepfake
132_237.jpg is deepfake
133_144.jpg is deepfake
134_90.jpg is deepfake
134_190.jpg is deepfake
134_555.jpg is deepfake
135_370.jpg is deepfake
135_830.jpg is deepfake
135_1080.jpg is deepfake
142_396.jpg is deepfake
142_412.jpg is deepfake
142_420.jpg is deepfake
143_23.jpg is deepfake
143_25.jpg is deepfake
143_26.jpg is deepfake
143_27.jpg is deepfake
143_29.jpg is deepfake
143_32.jpg is deepfake
143_33.jpg is deepfake
143_34.jpg is deepfake
143_39.jpg is deepfake
143_41.jpg is deepfake
143_43.jpg is deepfake
143_44.jpg is deepfake
143_46.jpg is deepfake
143_48.jpg is deepfake
143_52.jpg is deepfake
166_71.jpg is deepfake
166_74.jpg is deepfake
166_76.jpg is deepfake
167_24.jpg is deepfake
172_600.jpg is deepfake
172_610.jpg is deepfake
172_615.jpg is deepfake
cele16396.jpgF.jpg is deepfake
cele16748.jpgF.jpg is deepfake
cele17100.jpgF.jpg is deepfake
cele17543.jpgF.jpg is deepfake
cele17750.jpgF.jpg is deepfake
cele17783.jpgF.jpg is real
cele18188.jpgF.jpg is deepfake
cele18514.jpgF.jpg is deepfake
cele18909.jpgF.jpg is deepfake
cele33231.jpgF.jpg is deepfake
cele33349.jpgF.jpg is deepfake
cele33353.jpgF.jpg is deepfake
cele33363.jpgF.jpg is deepfake
cele33385.jpgF.jpg is deepfake
cele33393.jpgF.jpg is deepfake
cele33483.jpgF.jpg is deepfake
cele33493.jpgF.jpg is deepfake
cele33499.jpgF.jpg is deepfake
```

```
cele33502.jpgF.jpg is deepfake
cele34056.jpgF.jpg is deepfake
cele34064.jpgF.jpg is deepfake
cele34066.jpgF.jpg is deepfake
cele34070.jpgF.jpg is deepfake
cele34073.jpgF.jpg is deepfake
cele35162.jpgF.jpg is deepfake
cele35173.jpgF.jpg is deepfake
cele35180.jpgF.jpg is deepfake
cele35320.jpgF.jpg is deepfake
cele35328.jpgF.jpg is deepfake
cele35330.jpgF.jpg is deepfake
cele35347.jpgF.jpg is deepfake
cele35353.jpgF.jpg is deepfake
cele35370.jpgF.jpg is deepfake
cele35374.jpgF.jpg is deepfake
cele35383.jpgF.jpg is deepfake
```

Step 10 - Determine the performance of the classifier

```
# Figures for performance calculation
# real = 1, fake = 0
TP = true_pos = 50 # real faces predicted as real
TN = true_neg = 96 # fake faces predicted as fake
FP = false_pos = 4 # fake faces predicted as real
FN = false_neg = 50  # real faces predicted as fake
```

```
results = {}
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
ACC is  0.730
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.500
```

```
# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.960
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.926
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.658
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.649
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.518
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_InceptionResNetModel.h5')
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 inception_resnet_v2 (Functi  (None, 3, 3, 1536)       54336736
 onal)

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)
```

```
 dropout (Dropout)           (None, 1536)              0

 dense (Dense)               (None, 1)                 1537

=================================================================
Total params: 54,338,273
Trainable params: 53,510,161
Non-trainable params: 828,112
_________________________________________________________________
```

End of Notebook

This is a notebook for training, validation and testing a binary classifier with MobileNet as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection is lost
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'ab
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

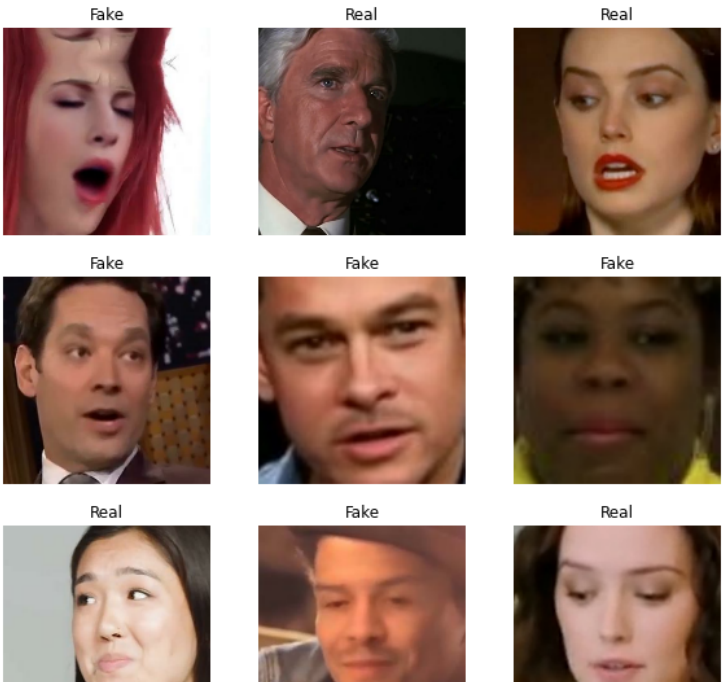
Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (160, 160)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```




```
AUTOTUNE = tensorflow.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```

Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.mobilenet_v2.preprocess_input
```

```
# Create the base model from the pre-trained model MobileNet V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_t
9412608/9406464 [==============================] - 0s 0us/step
9420800/9406464 [==============================] - 0s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 5, 5, 1280)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "mobilenetv2_1.00_160"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 160, 160, 3  0           []
```

```
                                )]

 Conv1 (Conv2D)                 (None, 80, 80, 32)   864         ['input_1[0][0]']

 bn_Conv1 (BatchNormalization)  (None, 80, 80, 32)   128         ['Conv1[0][0]']

 Conv1_relu (ReLU)              (None, 80, 80, 32)   0           ['bn_Conv1[0][0]']

 expanded_conv_depthwise (Depth  (None, 80, 80, 32)  288         ['Conv1_relu[0][0]']
 wiseConv2D)

 expanded_conv_depthwise_BN (Ba  (None, 80, 80, 32)  128         ['expanded_conv_depthwise[0][0]']
 tchNormalization)

 expanded_conv_depthwise_relu (  (None, 80, 80, 32)  0           ['expanded_conv_depthwise_BN[0][0
 ReLU)                                                           ]']

 expanded_conv_project (Conv2D)  (None, 80, 80, 16)  512         ['expanded_conv_depthwise_relu[0]
                                                                 [0]']

 expanded_conv_project_BN (Batc  (None, 80, 80, 16)  64          ['expanded_conv_project[0][0]']
 hNormalization)

 block_1_expand (Conv2D)        (None, 80, 80, 96)   1536        ['expanded_conv_project_BN[0][0]'
                                                                 ]

 block_1_expand_BN (BatchNormal  (None, 80, 80, 96)  384         ['block_1_expand[0][0]']
 ization)

 block_1_expand_relu (ReLU)     (None, 80, 80, 96)   0           ['block_1_expand_BN[0][0]']

 block_1_pad (ZeroPadding2D)    (None, 81, 81, 96)   0           ['block_1_expand_relu[0][0]']

 block_1_depthwise (DepthwiseCo  (None, 40, 40, 96)  864         ['block_1_pad[0][0]']
 nv2D)

 block_1_depthwise_BN (BatchNor  (None, 40, 40, 96)  384         ['block_1_depthwise[0][0]']
 malization)

 block_1_depthwise_relu (ReLU)  (None, 40, 40, 96)   0           ['block_1_depthwise_BN[0][0]']

 block_1_project (Conv2D)       (None, 40, 40, 24)   2304        ['block_1_depthwise_relu[0][0]']

 block_1_project_BN (BatchNorma  (None, 40, 40, 24)  96          ['block_1_project[0][0]']
 lization)

 block_2_expand (Conv2D)        (None, 40, 40, 144)  3456        ['block_1_project_BN[0][0]']

 block_2_expand_BN (BatchNormal  (None, 40, 40, 144)  576        ['block_2_expand[0][0]']
 ization)

 block_2_expand_relu (ReLU)     (None, 40, 40, 144)  0           ['block_2_expand_BN[0][0]']
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 1280)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(160, 160, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv_1 (TFOpLamb  (None, 160, 160, 3)      0
 da)

 tf.math.subtract_1 (TFOpLam  (None, 160, 160, 3)      0
 bda)

 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 372s 4s/step - loss: 0.9497 - accuracy: 0.3792
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.95
initial accuracy: 0.38
```

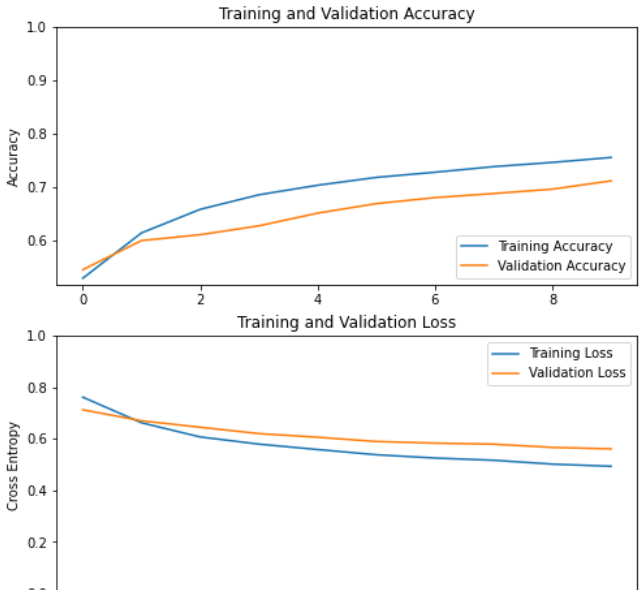
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 1458s 4s/step - loss: 0.7616 - accuracy: 0.5291 - val_loss: 0.7127 - val_accuracy: 0.5451
Epoch 2/10
364/364 [==============================] - 34s 93ms/step - loss: 0.6622 - accuracy: 0.6140 - val_loss: 0.6698 - val_accuracy: 0.5996
Epoch 3/10
364/364 [==============================] - 35s 94ms/step - loss: 0.6071 - accuracy: 0.6582 - val_loss: 0.6447 - val_accuracy: 0.6106
Epoch 4/10
364/364 [==============================] - 35s 94ms/step - loss: 0.5793 - accuracy: 0.6854 - val_loss: 0.6201 - val_accuracy: 0.6273
Epoch 5/10
364/364 [==============================] - 35s 95ms/step - loss: 0.5582 - accuracy: 0.7033 - val_loss: 0.6060 - val_accuracy: 0.6510
Epoch 6/10
364/364 [==============================] - 35s 96ms/step - loss: 0.5379 - accuracy: 0.7180 - val_loss: 0.5895 - val_accuracy: 0.6688
Epoch 7/10
364/364 [==============================] - 35s 95ms/step - loss: 0.5252 - accuracy: 0.7277 - val_loss: 0.5831 - val_accuracy: 0.6801
Epoch 8/10
364/364 [==============================] - 35s 95ms/step - loss: 0.5165 - accuracy: 0.7382 - val_loss: 0.5790 - val_accuracy: 0.6878
Epoch 9/10
364/364 [==============================] - 35s 96ms/step - loss: 0.5015 - accuracy: 0.7461 - val_loss: 0.5665 - val_accuracy: 0.6960
Epoch 10/10
364/364 [==============================] - 35s 95ms/step - loss: 0.4933 - accuracy: 0.7553 - val_loss: 0.5608 - val_accuracy: 0.7115
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  154
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv_1 (TFOpLamb  (None, 160, 160, 3)      0
 da)
```

```
 tf.math.subtract_1 (TFOpLam  (None, 160, 160, 3)      0
 bda)

 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,862,721
Non-trainable params: 396,544
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
56
```

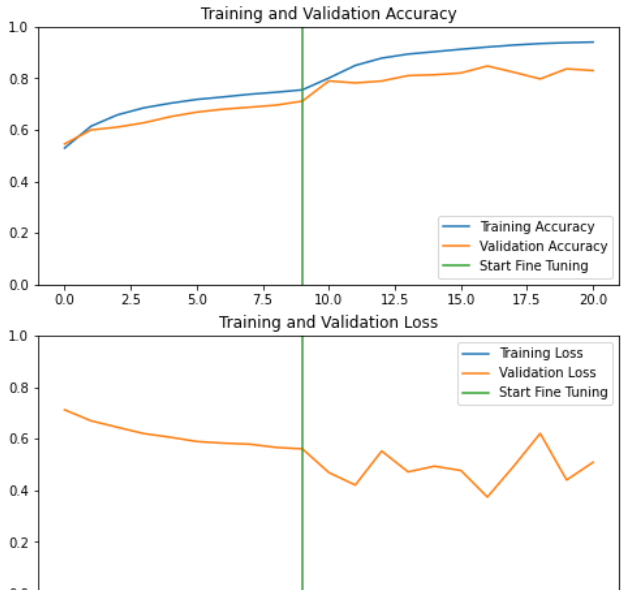
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
364/364 [==============================] - 41s 98ms/step - loss: 0.4127 - accuracy: 0.8009 - val_loss: 0.4689 - val_accuracy: 0.7898
Epoch 11/20
364/364 [==============================] - 38s 103ms/step - loss: 0.3287 - accuracy: 0.8501 - val_loss: 0.4201 - val_accuracy: 0.7821
Epoch 12/20
364/364 [==============================] - 37s 100ms/step - loss: 0.2710 - accuracy: 0.8785 - val_loss: 0.5523 - val_accuracy: 0.7895
Epoch 13/20
364/364 [==============================] - 37s 100ms/step - loss: 0.2396 - accuracy: 0.8942 - val_loss: 0.4715 - val_accuracy: 0.8107
Epoch 14/20
364/364 [==============================] - 37s 100ms/step - loss: 0.2150 - accuracy: 0.9035 - val_loss: 0.4935 - val_accuracy: 0.8138
Epoch 15/20
364/364 [==============================] - 38s 104ms/step - loss: 0.1957 - accuracy: 0.9129 - val_loss: 0.4767 - val_accuracy: 0.8209
Epoch 16/20
364/364 [==============================] - 38s 102ms/step - loss: 0.1782 - accuracy: 0.9217 - val_loss: 0.3737 - val_accuracy: 0.8480
Epoch 17/20
364/364 [==============================] - 37s 101ms/step - loss: 0.1626 - accuracy: 0.9292 - val_loss: 0.4938 - val_accuracy: 0.8237
Epoch 18/20
364/364 [==============================] - 38s 102ms/step - loss: 0.1526 - accuracy: 0.9351 - val_loss: 0.6208 - val_accuracy: 0.7977
Epoch 19/20
364/364 [==============================] - 38s 102ms/step - loss: 0.1392 - accuracy: 0.9385 - val_loss: 0.4399 - val_accuracy: 0.8370
Epoch 20/20
364/364 [==============================] - 37s 99ms/step - loss: 0.1356 - accuracy: 0.9409 - val_loss: 0.5087 - val_accuracy: 0.8302
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

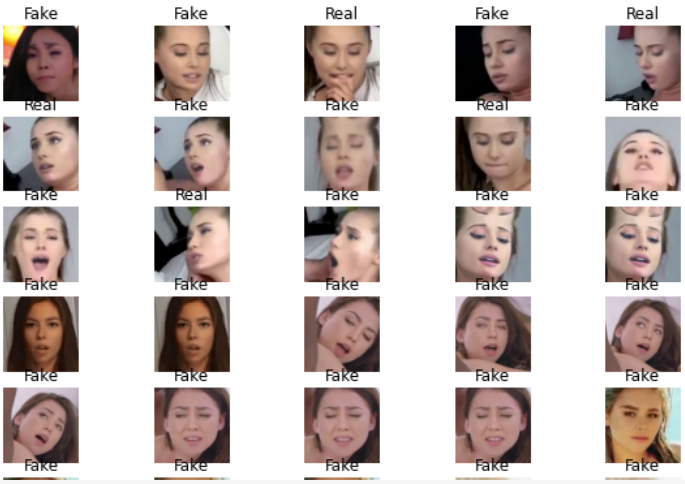
```
5/5 [==============================] - 20s 4s/step - loss: 0.5236 - accuracy: 0.7450
Test accuracy : 0.7450000047683716
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(160, 160))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving 34_0.jpg to 34_0 (1).jpg
Saving 34_6.jpg to 34_6 (1).jpg
Saving 35_0.jpg to 35_0 (1).jpg
Saving 39_360.jpg to 39_360 (1).jpg
Saving 39_372.jpg to 39_372 (1).jpg
Saving 39_384.jpg to 39_384 (1).jpg
Saving 39_402.jpg to 39_402 (1).jpg
Saving 39_780.jpg to 39_780 (1).jpg
Saving 39_786.jpg to 39_786 (1).jpg
Saving 40_0.jpg to 40_0 (1).jpg
Saving 40_12.jpg to 40_12 (1).jpg
Saving 40_24.jpg to 40_24 (1).jpg
Saving 42_168.jpg to 42_168 (1).jpg
Saving 42_198.jpg to 42_198 (1).jpg
Saving 42_204.jpg to 42_204 (1).jpg
```

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(160, 160))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving 114_666.jpg to 114_666.jpg
Saving 115_306.jpg to 115_306.jpg
Saving 115_816.jpg to 115_816.jpg
Saving 116_24.jpg to 116_24.jpg
Saving 116_1056.jpg to 116_1056.jpg
```

Step 10 - Determine the performance of the classifier

```
Saving 118_465.jpg to 118_465.jpg
```

```
# Figures for performance calculation
TP = true_pos = 46 # real faces predicted as real
TN = true_neg = 89 # fake faces predicted as fake
FP = false_pos = 11 # fake faces predicted as real
FN = false_neg = 54  # real faces predicted as fake
```

```
Saving 122_1.jpg to 122_1.jpg
```

```
results = {}
```

```
Saving 124_160.jpg to 124_160.jpg
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
ACC is  0.675
```

```
Saving 131_172.jpg to 131_172.jpg
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.460
```

```
Saving 135_370.jpg to 135_370.jpg
```

```
# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.890
```

```
Saving 142_20.jpg to 142_20.jpg
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.807
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.622
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.586
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.388
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_MobileNetModel.h5')
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask layers require a config and mus
  layer_config = serialize_layer_fn(layer)
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv_1 (TFOpLamb  (None, 160, 160, 3)      0
 da)

 tf.math.subtract_1 (TFOpLam  (None, 160, 160, 3)      0
 bda)

 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,862,721
Non-trainable params: 396,544
_________________________________________________________________
```

End of Notebook

This is a notebook for training, validation and testing a binary classifier with DenseNet as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection is lost
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'aba
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

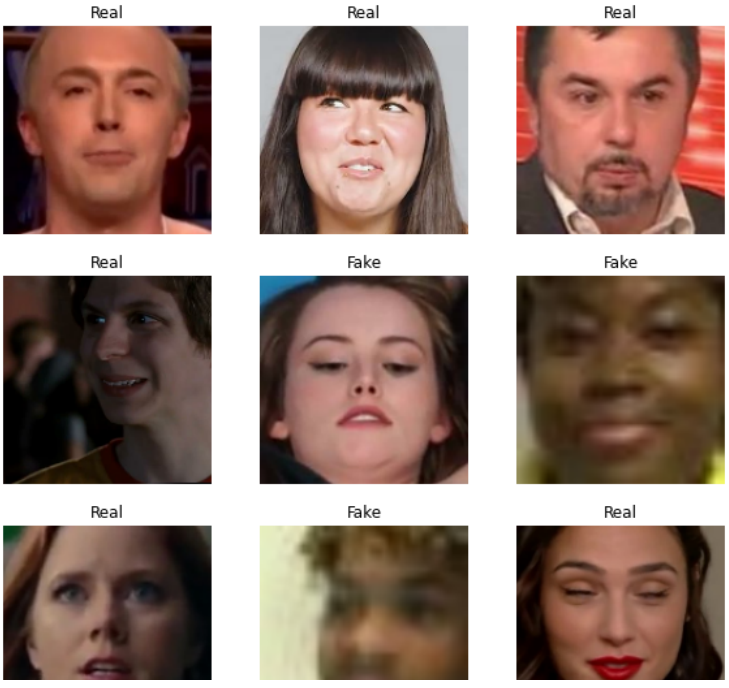
Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```




```
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```

Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.densenet.preprocess_input
```

```
# Create the base model from the pre-trained model
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.densenet.DenseNet201(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet201_weights_tf_dim_ordering_tf_kern
74842112/74836368 [==============================] - 0s 0us/step
74850304/74836368 [==============================] - 0s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 4, 4, 1920)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "densenet201"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 150, 150, 3  0           []
```

```
                                )]

 zero_padding2d (ZeroPadding2D)  (None, 156, 156, 3)  0           ['input_1[0][0]']

 conv1/conv (Conv2D)             (None, 75, 75, 64)   9408        ['zero_padding2d[0][0]']

 conv1/bn (BatchNormalization)   (None, 75, 75, 64)   256         ['conv1/conv[0][0]']

 conv1/relu (Activation)         (None, 75, 75, 64)   0           ['conv1/bn[0][0]']

 zero_padding2d_1 (ZeroPadding2  (None, 77, 77, 64)  0           ['conv1/relu[0][0]']
 D)

 pool1 (MaxPooling2D)            (None, 38, 38, 64)   0           ['zero_padding2d_1[0][0]']

 conv2_block1_0_bn (BatchNormal  (None, 38, 38, 64)  256         ['pool1[0][0]']
 ization)

 conv2_block1_0_relu (Activatio  (None, 38, 38, 64)  0           ['conv2_block1_0_bn[0][0]']
 n)

 conv2_block1_1_conv (Conv2D)    (None, 38, 38, 128)  8192        ['conv2_block1_0_relu[0][0]']

 conv2_block1_1_bn (BatchNormal  (None, 38, 38, 128)  512        ['conv2_block1_1_conv[0][0]']
 ization)

 conv2_block1_1_relu (Activatio  (None, 38, 38, 128)  0          ['conv2_block1_1_bn[0][0]']
 n)

 conv2_block1_2_conv (Conv2D)    (None, 38, 38, 32)   36864       ['conv2_block1_1_relu[0][0]']

 conv2_block1_concat (Concatena  (None, 38, 38, 96)  0           ['pool1[0][0]',
 te)                                                              'conv2_block1_2_conv[0][0]']

 conv2_block2_0_bn (BatchNormal  (None, 38, 38, 96)  384         ['conv2_block1_concat[0][0]']
 ization)

 conv2_block2_0_relu (Activatio  (None, 38, 38, 96)  0           ['conv2_block2_0_bn[0][0]']
 n)

 conv2_block2_1_conv (Conv2D)    (None, 38, 38, 128)  12288       ['conv2_block2_0_relu[0][0]']

 conv2_block2_1_bn (BatchNormal  (None, 38, 38, 128)  512        ['conv2_block2_1_conv[0][0]']
 ization)

 conv2_block2_1_relu (Activatio  (None, 38, 38, 128)  0          ['conv2_block2_1_bn[0][0]']
 n)

 conv2_block2_2_conv (Conv2D)    (None, 38, 38, 32)   36864       ['conv2_block2_1_relu[0][0]']

 conv2_block2_concat (Concatena  (None, 38, 38, 128)  0          ['conv2_block1_concat[0][0]',
 te)                                                              'conv2_block2_2_conv[0][0]']
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 1920)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(150, 150, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.nn.bias_add (TFOpLambda)  (None, 150, 150, 3)      0

 tf.math.truediv_1 (TFOpLamb  (None, 150, 150, 3)      0
 da)

 densenet201 (Functional)    (None, 4, 4, 1920)        18321984

 global_average_pooling2d (G  (None, 1920)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1920)              0

 dense (Dense)               (None, 1)                 1921

=================================================================
Total params: 18,323,905
Trainable params: 1,921
Non-trainable params: 18,321,984
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 374s 4s/step - loss: 0.7477 - accuracy: 0.5259
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.75
initial accuracy: 0.53
```

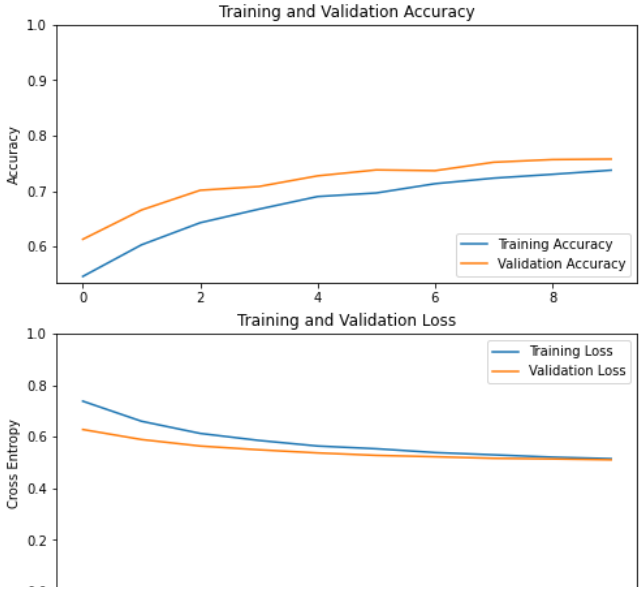
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 1491s 4s/step - loss: 0.7384 - accuracy: 0.5462 - val_loss: 0.6283 - val_accuracy: 0.6132
Epoch 2/10
364/364 [==============================] - 38s 102ms/step - loss: 0.6603 - accuracy: 0.6032 - val_loss: 0.5893 - val_accuracy: 0.6660
Epoch 3/10
364/364 [==============================] - 37s 101ms/step - loss: 0.6131 - accuracy: 0.6430 - val_loss: 0.5640 - val_accuracy: 0.7016
Epoch 4/10
364/364 [==============================] - 38s 102ms/step - loss: 0.5858 - accuracy: 0.6675 - val_loss: 0.5494 - val_accuracy: 0.7084
Epoch 5/10
364/364 [==============================] - 38s 102ms/step - loss: 0.5641 - accuracy: 0.6903 - val_loss: 0.5373 - val_accuracy: 0.7276
Epoch 6/10
364/364 [==============================] - 38s 102ms/step - loss: 0.5535 - accuracy: 0.6967 - val_loss: 0.5278 - val_accuracy: 0.7383
Epoch 7/10
364/364 [==============================] - 38s 102ms/step - loss: 0.5387 - accuracy: 0.7135 - val_loss: 0.5224 - val_accuracy: 0.7369
Epoch 8/10
364/364 [==============================] - 37s 101ms/step - loss: 0.5298 - accuracy: 0.7235 - val_loss: 0.5163 - val_accuracy: 0.7522
Epoch 9/10
364/364 [==============================] - 37s 101ms/step - loss: 0.5208 - accuracy: 0.7304 - val_loss: 0.5140 - val_accuracy: 0.7570
Epoch 10/10
364/364 [==============================] - 37s 101ms/step - loss: 0.5148 - accuracy: 0.7378 - val_loss: 0.5105 - val_accuracy: 0.7578
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  707
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
```

```
)

tf.nn.bias_add (TFOpLambda)  (None, 150, 150, 3)      0

tf.math.truediv_1 (TFOpLamb  (None, 150, 150, 3)      0
da)

densenet201 (Functional)    (None, 4, 4, 1920)        18321984

global_average_pooling2d (G  (None, 1920)             0
lobalAveragePooling2D)

dropout (Dropout)           (None, 1920)              0

dense (Dense)               (None, 1)                 1921

=================================================================
Total params: 18,323,905
Trainable params: 17,290,177
Non-trainable params: 1,033,728
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
521
```

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

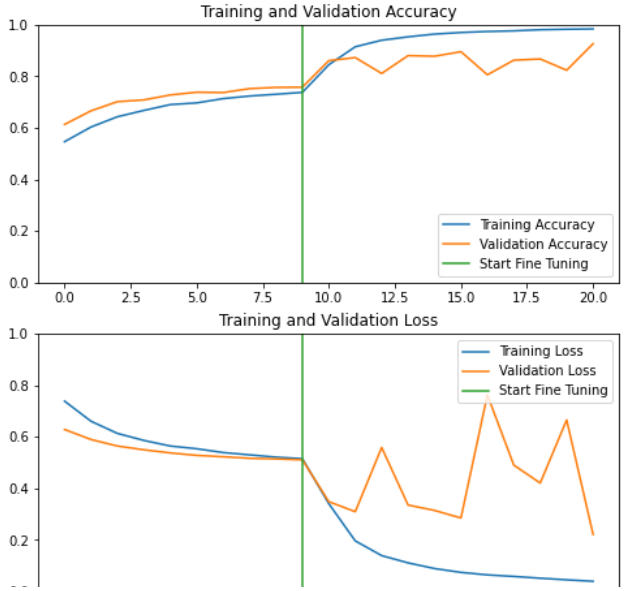
```
Epoch 10/20
364/364 [==============================] - 113s 231ms/step - loss: 0.3401 - accuracy: 0.8450 - val_loss: 0.3479 - val_accuracy: 0.8610
Epoch 11/20
364/364 [==============================] - 81s 222ms/step - loss: 0.1957 - accuracy: 0.9148 - val_loss: 0.3090 - val_accuracy: 0.8728
Epoch 12/20
364/364 [==============================] - 81s 221ms/step - loss: 0.1384 - accuracy: 0.9402 - val_loss: 0.5583 - val_accuracy: 0.8112
Epoch 13/20
364/364 [==============================] - 81s 223ms/step - loss: 0.1101 - accuracy: 0.9534 - val_loss: 0.3348 - val_accuracy: 0.8805
Epoch 14/20
364/364 [==============================] - 81s 223ms/step - loss: 0.0882 - accuracy: 0.9642 - val_loss: 0.3139 - val_accuracy: 0.8779
Epoch 15/20
364/364 [==============================] - 82s 223ms/step - loss: 0.0732 - accuracy: 0.9703 - val_loss: 0.2845 - val_accuracy: 0.8957
Epoch 16/20
364/364 [==============================] - 81s 221ms/step - loss: 0.0635 - accuracy: 0.9742 - val_loss: 0.7626 - val_accuracy: 0.8062
Epoch 17/20
364/364 [==============================] - 81s 222ms/step - loss: 0.0575 - accuracy: 0.9766 - val_loss: 0.4901 - val_accuracy: 0.8627
Epoch 18/20
364/364 [==============================] - 81s 222ms/step - loss: 0.0506 - accuracy: 0.9811 - val_loss: 0.4203 - val_accuracy: 0.8675
Epoch 19/20
364/364 [==============================] - 81s 222ms/step - loss: 0.0444 - accuracy: 0.9829 - val_loss: 0.6653 - val_accuracy: 0.8237
Epoch 20/20
364/364 [==============================] - 82s 223ms/step - loss: 0.0387 - accuracy: 0.9842 - val_loss: 0.2210 - val_accuracy: 0.9265
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
```

```
plt.xlabel('epoch')
plt.show()
```



Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

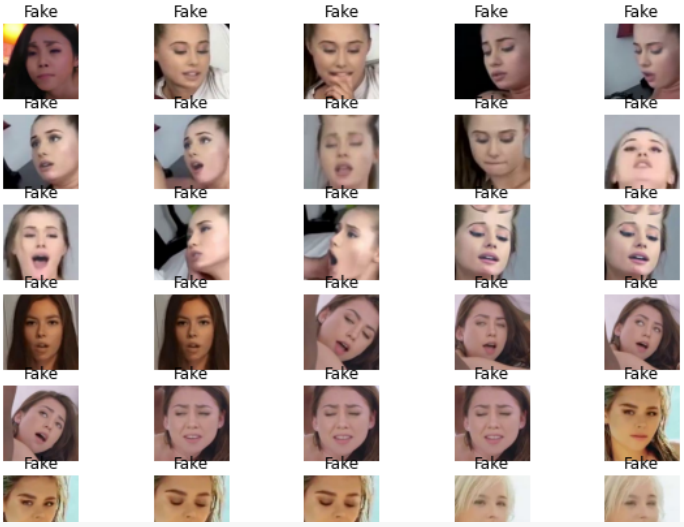
```
5/5 [==============================] - 19s 4s/step - loss: 0.5048 - accuracy: 0.8400
Test accuracy : 0.8399999737739563
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 34_0.jpg to 34_0.jpg
Saving 34_6.jpg to 34_6.jpg
Saving 35_0.jpg to 35_0.jpg
Saving 39_360.jpg to 39_360.jpg
Saving 39_372.jpg to 39_372.jpg
Saving 39_384.jpg to 39_384.jpg
Saving 39_402.jpg to 39_402.jpg
Saving 39_780.jpg to 39_780.jpg
Saving 39_786.jpg to 39_786.jpg
Saving 40_0.jpg to 40_0.jpg
Saving 40_12.jpg to 40_12.jpg
Saving 40_24.jpg to 40_24.jpg
Saving 42_168.jpg to 42_168.jpg
Saving 42_198.jpg to 42_198.jpg
Saving 42_204.jpg to 42_204.jpg
Saving 43_36.jpg to 43_36.jpg
Saving 43_150.jpg to 43_150.jpg
Saving 43_192.jpg to 43_192.jpg
Saving 43_210.jpg to 43_210.jpg
Saving 44_6.jpg to 44_6.jpg
Saving 44_90.jpg to 44_90.jpg
Saving 44_192.jpg to 44_192.jpg
Saving 44_198.jpg to 44_198.jpg
Saving 45_60.jpg to 45_60.jpg
Saving 45_150.jpg to 45_150.jpg
Saving 46_360.jpg to 46_360.jpg
Saving 46_384.jpg to 46_384.jpg
Saving 47_528.jpg to 47_528.jpg
Saving 47_738.jpg to 47_738.jpg
Saving 48_12.jpg to 48_12.jpg
Saving 48_270.jpg to 48_270.jpg
Saving 48_318.jpg to 48_318.jpg
Saving 48_522.jpg to 48_522.jpg
Saving 48_708.jpg to 48_708.jpg
Saving 50_192.jpg to 50_192.jpg
Saving 50_222.jpg to 50_222.jpg
Saving 51_120.jpg to 51_120.jpg
Saving 51_300.jpg to 51_300.jpg
Saving 51_372.jpg to 51_372.jpg
Saving 52_42.jpg to 52_42.jpg
Saving 52_450.jpg to 52_450.jpg
Saving 53_66.jpg to 53_66.jpg
Saving 54_18.jpg to 54_18.jpg
Saving 54_186.jpg to 54_186.jpg
Saving 69_268.jpg to 69_268.jpg
Saving 69_302.jpg to 69_302.jpg
Saving 78_3.jpg to 78_3.jpg
Saving 78_318.jpg to 78_318.jpg
Saving 96_30.jpg to 96_30.jpg
Saving 96_32.jpg to 96_32.jpg
Saving 97_137.jpg to 97_137.jpg
Saving 97_146.jpg to 97_146.jpg
Saving 98_62.jpg to 98_62.jpg
Saving 99_108.jpg to 99_108.jpg
Saving 100_168.jpg to 100_168.jpg
Saving 100_956.jpg to 100_956.jpg
Saving 100_972.jpg to 100_972.jpg
Saving 101_6.jpg to 101_6.jpg
Saving 101_8.jpg to 101_8.jpg
Saving 101_10.jpg to 101_10.jpg
Saving 101_40.jpg to 101_40.jpg
Saving 101_488.jpg to 101_488.jpg
Saving 101_490.jpg to 101_490.jpg
Saving 102_14.jpg to 102_14.jpg
Saving 102_16.jpg to 102_16.jpg
Saving 102_320.jpg to 102_320.jpg
Saving id19_0000.mp40.jpgFace.jpg to id19_0000.mp40.jpgFace.jpg
Saving id19_0001.mp4288.jpgFace.jpg to id19_0001.mp4288.jpgFace.jpg
Saving id19_0001.mp4312.jpgFace.jpg to id19_0001.mp4312.jpgFace.jpg
Saving id19_0005.mp4312.jpgFace.jpg to id19_0005.mp4312.jpgFace.jpg
Saving id19_0006.mp40.jpgFace.jpg to id19_0006.mp40.jpgFace.jpg
Saving id19_0006.mp424.jpgFace.jpg to id19_0006.mp424.jpgFace.jpg
Saving id19_0006.mp448.jpgFace.jpg to id19_0006.mp448.jpgFace.jpg
Saving id19_0006.mp472.jpgFace.jpg to id19_0006.mp472.jpgFace.jpg
Saving id19_0006.mp496.jpgFace.jpg to id19_0006.mp496.jpgFace.jpg
Saving id19_0006.mp4120.jpgFace.jpg to id19_0006.mp4120.jpgFace.jpg
Saving id19_0006.mp4144.jpgFace.jpg to id19_0006.mp4144.jpgFace.jpg
Saving id20_0002.mp424.jpgFace.jpg to id20_0002.mp424.jpgFace.jpg
Saving id20_0002.mp448.jpgFace.jpg to id20_0002.mp448.jpgFace.jpg
Saving id20_0002.mp472.jpgFace.jpg to id20_0002.mp472.jpgFace.jpg
Saving id20_0002.mp496.jpgFace.jpg to id20_0002.mp496.jpgFace.jpg
Saving id20_0002.mp4120.jpgFace.jpg to id20_0002.mp4120.jpgFace.jpg
Saving id20_0002.mp4144.jpgFace.jpg to id20_0002.mp4144.jpgFace.jpg
Saving id23_0007.mp4144.jpgFace.jpg to id23_0007.mp4144.jpgFace.jpg
Saving id23_0008.mp4168.jpgFace.jpg to id23_0008.mp4168.jpgFace.jpg
Saving id23_0008.mp4192.jpgFace.jpg to id23_0008.mp4192.jpgFace.jpg
Saving id23_0008.mp4216.jpgFace.jpg to id23_0008.mp4216.jpgFace.jpg
Saving id23_0008.mp4240.jpgFace.jpg to id23_0008.mp4240.jpgFace.jpg
Saving id23_0008.mp4264.jpgFace.jpg to id23_0008.mp4264.jpgFace.jpg
Saving id23_0008.mp4288.jpgFace.jpg to id23_0008.mp4288.jpgFace.jpg
```

```
Saving id23_0009.mp40.jpgFace.jpg to id23_0009.mp40.jpgFace.jpg
Saving id23_0009.mp424.jpgFace.jpg to id23_0009.mp424.jpgFace.jpg
Saving id23_0009.mp448.jpgFace.jpg to id23_0009.mp448.jpgFace.jpg
Saving id23_0009.mp472.jpgFace.jpg to id23_0009.mp472.jpgFace.jpg
Saving id24_0003.mp40.jpgFace.jpg to id24_0003.mp40.jpgFace.jpg
Saving id24_0003.mp472.jpgFace.jpg to id24_0003.mp472.jpgFace.jpg
Saving id24_0003.mp4120.jpgFace.jpg to id24_0003.mp4120.jpgFace.jpg
Saving id24_0003.mp4144.jpgFace.jpg to id24_0003.mp4144.jpgFace.jpg
Saving id25_0004.mp4288.jpgFace.jpg to id25_0004.mp4288.jpgFace.jpg
Saving id25_0004.mp4312.jpgFace.jpg to id25_0004.mp4312.jpgFace.jpg
34_0.jpg is real
34_6.jpg is real
35_0.jpg is real
39_360.jpg is real
39_372.jpg is real
39_384.jpg is real
39_402.jpg is real
39_780.jpg is deepfake
39_786.jpg is real
40_0.jpg is real
40_12.jpg is real
40_24.jpg is real
42_168.jpg is deepfake
42_198.jpg is real
42_204.jpg is real
43_36.jpg is real
43_150.jpg is deepfake
43_192.jpg is deepfake
43_210.jpg is deepfake
44_6.jpg is deepfake
44_90.jpg is deepfake
44_192.jpg is deepfake
44_198.jpg is deepfake
45_60.jpg is deepfake
45_150.jpg is real
46_360.jpg is deepfake
46_384.jpg is real
47_528.jpg is deepfake
47_738.jpg is real
48_12.jpg is real
48_270.jpg is real
48_318.jpg is real
48_522.jpg is real
48_708.jpg is real
50_192.jpg is real
50_222.jpg is real
51_120.jpg is real
51_300.jpg is real
51_372.jpg is real
52_42.jpg is real
52_450.jpg is real
53_66.jpg is real
54_18.jpg is real
54_186.jpg is real
69_268.jpg is real
69_302.jpg is real
78_3.jpg is real
78_318.jpg is real
96_30.jpg is real
96_32.jpg is real
97_137.jpg is deepfake
97_146.jpg is real
98_62.jpg is real
99_108.jpg is real
100_168.jpg is real
100_956.jpg is real
100_972.jpg is real
101_6.jpg is real
101_8.jpg is real
101_10.jpg is real
101_40.jpg is real
101_488.jpg is real
101_490.jpg is real
102_14.jpg is real
102_16.jpg is real
102_320.jpg is deepfake
id19_0000.mp40.jpgFace.jpg is real
id19_0001.mp4288.jpgFace.jpg is real
id19_0001.mp4312.jpgFace.jpg is real
id19_0005.mp4312.jpgFace.jpg is real
id19_0006.mp40.jpgFace.jpg is real
id19_0006.mp424.jpgFace.jpg is real
id19_0006.mp448.jpgFace.jpg is real
id19_0006.mp472.jpgFace.jpg is real
id19_0006.mp496.jpgFace.jpg is real
id19_0006.mp4120.jpgFace.jpg is real
id19_0006.mp4144.jpgFace.jpg is real
id20_0002.mp424.jpgFace.jpg is real
id20_0002.mp448.jpgFace.jpg is real
id20_0002.mp472.jpgFace.jpg is real
id20_0002.mp496.jpgFace.jpg is real
id20_0002.mp4120.jpgFace.jpg is real
id20_0002.mp4144.jpgFace.jpg is real
```

id23_0007.mp4144.jpgFace.jpg is real
id23_0008.mp4168.jpgFace.jpg is deepfake
id23_0008.mp4192.jpgFace.jpg is real
id23_0008.mp4216.jpgFace.jpg is deepfake
id23_0008.mp4240.jpgFace.jpg is real
id23_0008.mp4264.jpgFace.jpg is real
id23_0008.mp4288.jpgFace.jpg is deepfake
id23_0009.mp40.jpgFace.jpg is deepfake
id23_0009.mp424.jpgFace.jpg is real
id23_0009.mp448.jpgFace.jpg is real
id23_0009.mp472.jpgFace.jpg is real
id24_0003.mp40.jpgFace.jpg is real
id24_0003.mp472.jpgFace.jpg is real
id24_0003.mp4120.jpgFace.jpg is real
id24_0003.mp4144.jpgFace.jpg is real
id25_0004.mp4288.jpgFace.jpg is real
id25_0004.mp4312.jpgFace.jpg is real

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 114_666.jpg to 114_666.jpg
Saving 115_306.jpg to 115_306.jpg
Saving 115_816.jpg to 115_816.jpg
Saving 116_24.jpg to 116_24.jpg
Saving 116_1056.jpg to 116_1056.jpg
Saving 117_12.jpg to 117_12.jpg
Saving 117_750.jpg to 117_750.jpg
Saving 118_9.jpg to 118_9.jpg
Saving 118_405.jpg to 118_405.jpg
Saving 119_65.jpg to 119_65.jpg
Saving 119_660.jpg to 119_660.jpg
Saving 120_50.jpg to 120_50.jpg
Saving 120_235.jpg to 120_235.jpg
Saving 121_24.jpg to 121_24.jpg
Saving 121_28.jpg to 121_28.jpg
Saving 122_0.jpg to 122_0.jpg
Saving 122_1.jpg to 122_1.jpg
Saving 124_154.jpg to 124_154.jpg
Saving 124_158.jpg to 124_158.jpg
Saving 124_160.jpg to 124_160.jpg
Saving 124_170.jpg to 124_170.jpg
Saving 125_65.jpg to 125_65.jpg
Saving 125_68.jpg to 125_68.jpg
Saving 125_69.jpg to 125_69.jpg
Saving 130_2.jpg to 130_2.jpg
Saving 130_34.jpg to 130_34.jpg
Saving 130_50.jpg to 130_50.jpg
Saving 130_78.jpg to 130_78.jpg
Saving 131_172.jpg to 131_172.jpg
Saving 131_212.jpg to 131_212.jpg
Saving 132_36.jpg to 132_36.jpg
Saving 132_105.jpg to 132_105.jpg
Saving 132_237.jpg to 132_237.jpg
Saving 133_144.jpg to 133_144.jpg
Saving 134_90.jpg to 134_90.jpg
Saving 134_190.jpg to 134_190.jpg
Saving 134_555.jpg to 134_555.jpg
Saving 135_370.jpg to 135_370.jpg
Saving 135_830.jpg to 135_830.jpg
Saving 135_1080.jpg to 135_1080.jpg
Saving 142_396.jpg to 142_396.jpg
Saving 142_412.jpg to 142_412.jpg
Saving 142_420.jpg to 142_420.jpg
Saving 143_23.jpg to 143_23.jpg
Saving 143_25.jpg to 143_25.jpg
Saving 143_26.jpg to 143_26.jpg
Saving 143_27.jpg to 143_27.jpg
Saving 143_29.jpg to 143_29.jpg
Saving 143_32.jpg to 143_32.jpg
```

Step 10 - Determine the performance of the classifier

```
# Figures for performance calculation
TP = true_pos = 82 # real faces predicted as real
TN = true_neg = 90 # fake faces predicted as fake
FP = false_pos = 10 # fake faces predicted as real
FN = false_neg = 18 # real faces predicted as fake
```

```
results = {}
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
     ACC is  0.860
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.820
```

```
# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.900
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.891
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.833
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.854
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.722
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_DenseNet.h5')
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.nn.bias_add (TFOpLambda)  (None, 150, 150, 3)      0

 tf.math.truediv_1 (TFOpLamb  (None, 150, 150, 3)      0
 da)

 densenet201 (Functional)    (None, 4, 4, 1920)        18321984

 global_average_pooling2d (G  (None, 1920)             0
```

```
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1920)              0

 dense (Dense)               (None, 1)                 1921

=================================================================
Total params: 18,323,905
Trainable params: 17,290,177
Non-trainable params: 1,033,728
_________________________________________________________________
```

End of Notebook

This is a notebook for training, validation and testing a binary classifier with NASNetMobile as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection is lost
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'aba
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (224, 224)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```


```
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```

Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.nasnet.preprocess_input
```

```
# Create the base model from the pre-trained model
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.nasnet.NASNetMobile(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/nasnet/NASNet-mobile-no-top.h5
19996672/19993432 [==============================] - 0s 0us/step
20004864/19993432 [==============================] - 0s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 7, 7, 1056)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "NASNet"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
```

```
                                )]

 stem_conv1 (Conv2D)            (None, 111, 111, 32  864         ['input_1[0][0]']
                                )

 stem_bn1 (BatchNormalization)  (None, 111, 111, 32  128         ['stem_conv1[0][0]']
                                )

 activation (Activation)        (None, 111, 111, 32  0           ['stem_bn1[0][0]']
                                )

 reduction_conv_1_stem_1 (Conv2  (None, 111, 111, 11  352        ['activation[0][0]']
 D)                             )

 reduction_bn_1_stem_1 (BatchNo  (None, 111, 111, 11  44         ['reduction_conv_1_stem_1[0][0]']
 rmalization)                   )

 activation_1 (Activation)      (None, 111, 111, 11  0           ['reduction_bn_1_stem_1[0][0]']
                                )

 activation_3 (Activation)      (None, 111, 111, 32  0           ['stem_bn1[0][0]']
                                )

 separable_conv_1_pad_reduction  (None, 115, 115, 11  0          ['activation_1[0][0]']
 _left1_stem_1 (ZeroPadding2D)  )

 separable_conv_1_pad_reduction  (None, 117, 117, 32  0          ['activation_3[0][0]']
 _right1_stem_1 (ZeroPadding2D)  )

 separable_conv_1_reduction_lef  (None, 56, 56, 11)  396         ['separable_conv_1_pad_reduction_
 t1_stem_1 (SeparableConv2D)                                     left1_stem_1[0][0]']

 separable_conv_1_reduction_rig  (None, 56, 56, 11)  1920        ['separable_conv_1_pad_reduction_
 ht1_stem_1 (SeparableConv2D)                                    right1_stem_1[0][0]']

 separable_conv_1_bn_reduction_  (None, 56, 56, 11)  44          ['separable_conv_1_reduction_left
 left1_stem_1 (BatchNormalizati                                  1_stem_1[0][0]']
 on)

 separable_conv_1_bn_reduction_  (None, 56, 56, 11)  44          ['separable_conv_1_reduction_righ
 right1_stem_1 (BatchNormalizat                                  t1_stem_1[0][0]']
 ion)

 activation_2 (Activation)      (None, 56, 56, 11)   0           ['separable_conv_1_bn_reduction_l
                                                                 eft1_stem_1[0][0]']

 activation_4 (Activation)      (None, 56, 56, 11)   0           ['separable_conv_1_bn_reduction_r
                                                                 ight1_stem_1[0][0]']

 separable_conv_2_reduction_lef  (None, 56, 56, 11)  396         ['activation_2[0][0]']
 t1_stem_1 (SeparableConv2D)

 separable_conv_2_reduction_rig  (None, 56, 56, 11)  660         ['activation_4[0][0]']
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 1056)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(224, 224, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 sequential_2 (Sequential)   (None, 224, 224, 3)       0

 tf.math.truediv (TFOpLambda  (None, 224, 224, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 224, 224, 3)      0
 a)

 NASNet (Functional)         (None, 7, 7, 1056)        4269716

 global_average_pooling2d (G  (None, 1056)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1056)              0

 dense (Dense)               (None, 1)                 1057

=================================================================
Total params: 4,270,773
Trainable params: 1,057
Non-trainable params: 4,269,716
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 322s 3s/step - loss: 0.7333 - accuracy: 0.4990
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.73
initial accuracy: 0.50
```

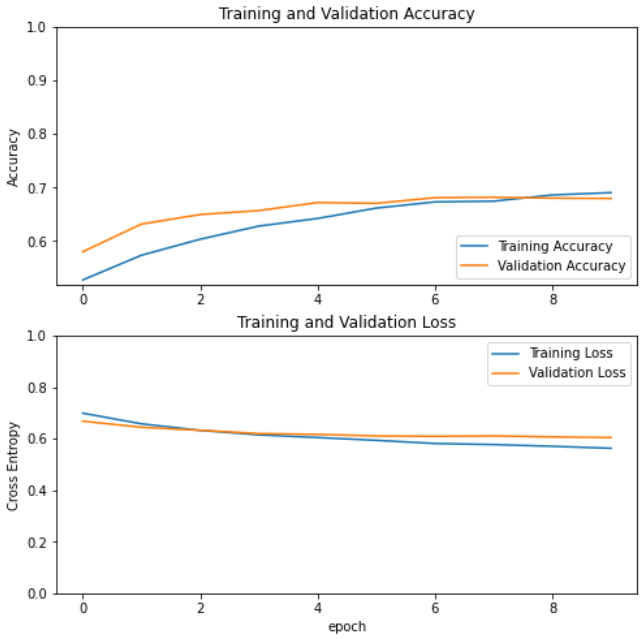
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 1066s 3s/step - loss: 0.6994 - accuracy: 0.5275 - val_loss: 0.6684 - val_accuracy: 0.5801
Epoch 2/10
364/364 [==============================] - 44s 118ms/step - loss: 0.6580 - accuracy: 0.5736 - val_loss: 0.6446 - val_accuracy: 0.6318
Epoch 3/10
364/364 [==============================] - 44s 119ms/step - loss: 0.6327 - accuracy: 0.6035 - val_loss: 0.6329 - val_accuracy: 0.6496
Epoch 4/10
364/364 [==============================] - 45s 122ms/step - loss: 0.6152 - accuracy: 0.6280 - val_loss: 0.6202 - val_accuracy: 0.6570
Epoch 5/10
364/364 [==============================] - 44s 119ms/step - loss: 0.6048 - accuracy: 0.6423 - val_loss: 0.6170 - val_accuracy: 0.6719
Epoch 6/10
364/364 [==============================] - 43s 118ms/step - loss: 0.5940 - accuracy: 0.6616 - val_loss: 0.6108 - val_accuracy: 0.6705
Epoch 7/10
364/364 [==============================] - 44s 118ms/step - loss: 0.5817 - accuracy: 0.6732 - val_loss: 0.6095 - val_accuracy: 0.6810
Epoch 8/10
364/364 [==============================] - 44s 119ms/step - loss: 0.5775 - accuracy: 0.6745 - val_loss: 0.6111 - val_accuracy: 0.6818
Epoch 9/10
364/364 [==============================] - 44s 118ms/step - loss: 0.5708 - accuracy: 0.6862 - val_loss: 0.6073 - val_accuracy: 0.6801
Epoch 10/10
364/364 [==============================] - 44s 119ms/step - loss: 0.5632 - accuracy: 0.6904 - val_loss: 0.6045 - val_accuracy: 0.6796
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
```

```
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  769
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 sequential_2 (Sequential)   (None, 224, 224, 3)       0

 tf.math.truediv (TFOpLambda  (None, 224, 224, 3)      0
 )
```

```
 tf.math.subtract (TFOpLambd  (None, 224, 224, 3)      0
 a)

 NASNet (Functional)         (None, 7, 7, 1056)        4269716

 global_average_pooling2d (G  (None, 1056)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1056)              0

 dense (Dense)               (None, 1)                 1057

=================================================================
Total params: 4,270,773
Trainable params: 4,210,713
Non-trainable params: 60,060
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
653
```

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
364/364 [==============================] - 150s 307ms/step - loss: 0.4172 - accuracy: 0.7994 - val_loss: 0.3847 - val_accuracy: 0.8372
Epoch 11/20
364/364 [==============================] - 108s 296ms/step - loss: 0.2758 - accuracy: 0.8791 - val_loss: 0.3952 - val_accuracy: 0.8389
Epoch 12/20
364/364 [==============================] - 108s 296ms/step - loss: 0.2081 - accuracy: 0.9075 - val_loss: 0.4077 - val_accuracy: 0.8370
Epoch 13/20
364/364 [==============================] - 108s 296ms/step - loss: 0.1731 - accuracy: 0.9264 - val_loss: 0.6415 - val_accuracy: 0.7816
Epoch 14/20
364/364 [==============================] - 108s 296ms/step - loss: 0.1489 - accuracy: 0.9369 - val_loss: 0.3988 - val_accuracy: 0.8488
Epoch 15/20
364/364 [==============================] - 108s 296ms/step - loss: 0.1286 - accuracy: 0.9457 - val_loss: 0.4404 - val_accuracy: 0.8378
Epoch 16/20
364/364 [==============================] - 108s 297ms/step - loss: 0.1185 - accuracy: 0.9495 - val_loss: 0.4735 - val_accuracy: 0.8412
Epoch 17/20
364/364 [==============================] - 109s 297ms/step - loss: 0.1065 - accuracy: 0.9555 - val_loss: 0.5017 - val_accuracy: 0.8271
Epoch 18/20
364/364 [==============================] - 108s 296ms/step - loss: 0.0978 - accuracy: 0.9595 - val_loss: 0.4551 - val_accuracy: 0.8539
Epoch 19/20
364/364 [==============================] - 108s 295ms/step - loss: 0.0847 - accuracy: 0.9655 - val_loss: 0.5847 - val_accuracy: 0.8500
Epoch 20/20
364/364 [==============================] - 108s 296ms/step - loss: 0.0766 - accuracy: 0.9681 - val_loss: 0.5013 - val_accuracy: 0.8533
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

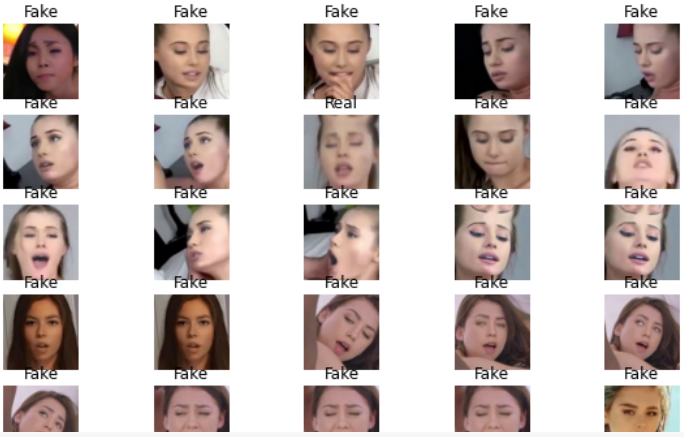
```
5/5 [==============================] - 16s 3s/step - loss: 0.7071 - accuracy: 0.7450
Test accuracy : 0.7450000047683716
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(224, 224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 34_0.jpg to 34_0.jpg
Saving 34_6.jpg to 34_6.jpg
Saving 35_0.jpg to 35_0.jpg
Saving 39_360.jpg to 39_360.jpg
Saving 39_372.jpg to 39_372.jpg
Saving 39_384.jpg to 39_384.jpg
Saving 39_402.jpg to 39_402.jpg
Saving 39_780.jpg to 39_780.jpg
Saving 39_786.jpg to 39_786.jpg
Saving 40_0.jpg to 40_0.jpg
Saving 40_12.jpg to 40_12.jpg
Saving 40_24.jpg to 40_24.jpg
Saving 42_168.jpg to 42_168.jpg
```

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(224, 224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving 114_666.jpg to 114_666.jpg
Saving 115_306.jpg to 115_306.jpg
Saving 115_816.jpg to 115_816.jpg
```

Step 10 - Determine the performance of the classifier

```
Saving 117_750.jpg to 117_750.jpg
```

```
# Figures for performance calculation
TP = true_pos = 58 # real faces predicted as real
TN = true_neg = 93 # fake faces predicted as fake
FP = false_pos = 7 # fake faces predicted as real
FN = false_neg = 42 # real faces predicted as fake
```

```
Saving 121_28.jpg to 121_28.jpg
```

```
results = {}
```

```
Saving 123_1.jpg to 123_1.jpg
Saving 124_154.jpg to 124_154.jpg
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
ACC is  0.755
```

```
Saving 130_50.jpg to 130_50.jpg
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.580
```

```
Saving 134_196.jpg to 134_196.jpg
```

```
# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.930
```

```
Saving 142_26.jpg to 142_26.jpg
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.892
```

```
Saving 143_44.jpg to 143_44.jpg
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.689
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.703
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.544
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_NASNetMobile.h5')
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask layers require a config and must
  layer_config = serialize_layer_fn(layer)
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 sequential_2 (Sequential)   (None, 224, 224, 3)       0

 tf.math.truediv (TFOpLambda  (None, 224, 224, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 224, 224, 3)      0
 a)

 NASNet (Functional)         (None, 7, 7, 1056)        4269716

 global_average_pooling2d (G  (None, 1056)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1056)              0

 dense (Dense)               (None, 1)                 1057

=================================================================
Total params: 4,270,773
Trainable params: 4,210,713
Non-trainable params: 60,060
_________________________________________________________________
```

End of Notebook

This is a notebook for training, validation and testing a binary classifier with EfficientNetB3 as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection is lost
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'aba
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

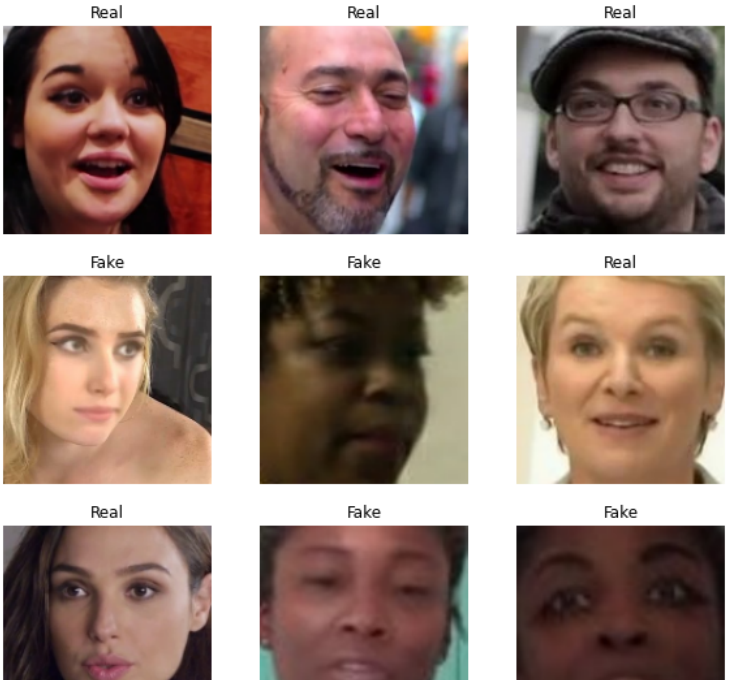
Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

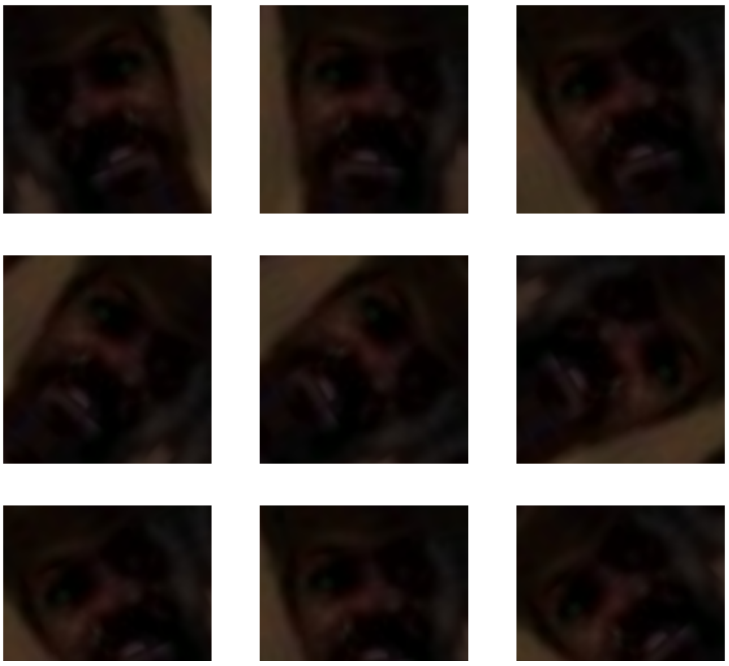



```
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```



Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.efficientnet.preprocess_input
```

```
# Create the base model from the pre-trained model
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.efficientnet.EfficientNetB3(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb3_notop.h5
43941888/43941136 [==============================] - 2s 0us/step
43950080/43941136 [==============================] - 2s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 5, 5, 1536)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "efficientnetb3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 150, 150, 3  0           []
                                )]
```

```
rescaling (Rescaling)          (None, 150, 150, 3)  0           ['input_1[0][0]']

normalization (Normalization)  (None, 150, 150, 3)  7           ['rescaling[0][0]']

stem_conv_pad (ZeroPadding2D)  (None, 151, 151, 3)  0           ['normalization[0][0]']

stem_conv (Conv2D)             (None, 75, 75, 40)   1080        ['stem_conv_pad[0][0]']

stem_bn (BatchNormalization)   (None, 75, 75, 40)   160         ['stem_conv[0][0]']

stem_activation (Activation)   (None, 75, 75, 40)   0           ['stem_bn[0][0]']

block1a_dwconv (DepthwiseConv2  (None, 75, 75, 40)  360         ['stem_activation[0][0]']
D)

block1a_bn (BatchNormalization  (None, 75, 75, 40)  160         ['block1a_dwconv[0][0]']
)

block1a_activation (Activation  (None, 75, 75, 40)  0           ['block1a_bn[0][0]']
)

block1a_se_squeeze (GlobalAver  (None, 40)          0           ['block1a_activation[0][0]']
agePooling2D)

block1a_se_reshape (Reshape)   (None, 1, 1, 40)     0           ['block1a_se_squeeze[0][0]']

block1a_se_reduce (Conv2D)     (None, 1, 1, 10)     410         ['block1a_se_reshape[0][0]']

block1a_se_expand (Conv2D)     (None, 1, 1, 40)     440         ['block1a_se_reduce[0][0]']

block1a_se_excite (Multiply)   (None, 75, 75, 40)   0           ['block1a_activation[0][0]',
                                                                 'block1a_se_expand[0][0]']

block1a_project_conv (Conv2D)  (None, 75, 75, 24)   960         ['block1a_se_excite[0][0]']

block1a_project_bn (BatchNorma  (None, 75, 75, 24)  96          ['block1a_project_conv[0][0]']
lization)

block1b_dwconv (DepthwiseConv2  (None, 75, 75, 24)  216         ['block1a_project_bn[0][0]']
D)

block1b_bn (BatchNormalization  (None, 75, 75, 24)  96          ['block1b_dwconv[0][0]']
)

block1b_activation (Activation  (None, 75, 75, 24)  0           ['block1b_bn[0][0]']
)

block1b_se_squeeze (GlobalAver  (None, 24)          0           ['block1b_activation[0][0]']
agePooling2D)

block1b_se_reshape (Reshape)   (None, 1, 1, 24)     0           ['block1b_se_squeeze[0][0]']
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 1536)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(150, 150, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 efficientnetb3 (Functional)  (None, 5, 5, 1536)       10783535

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1536)              0

 dense (Dense)               (None, 1)                 1537

=================================================================
Total params: 10,785,072
Trainable params: 1,537
Non-trainable params: 10,783,535
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 834s 8s/step - loss: 0.7365 - accuracy: 0.4566
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.74
initial accuracy: 0.46
```

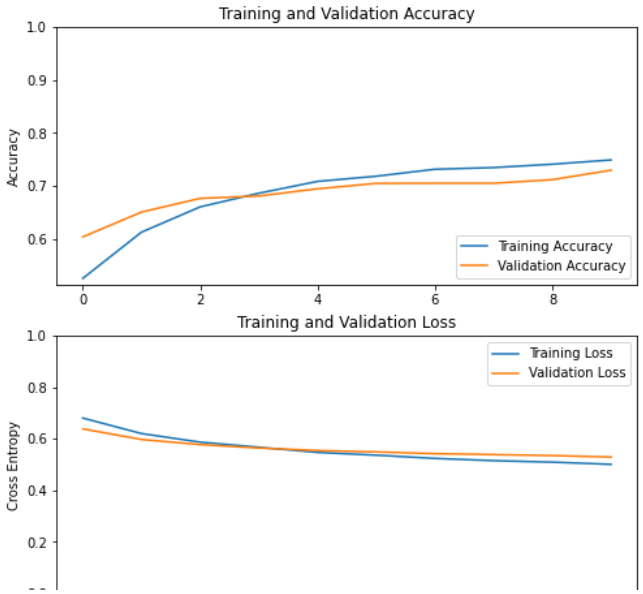
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 3207s 9s/step - loss: 0.6805 - accuracy: 0.5260 - val_loss: 0.6383 - val_accuracy: 0.6041
Epoch 2/10
364/364 [==============================] - 47s 127ms/step - loss: 0.6203 - accuracy: 0.6131 - val_loss: 0.5973 - val_accuracy: 0.6510
Epoch 3/10
364/364 [==============================] - 48s 129ms/step - loss: 0.5869 - accuracy: 0.6608 - val_loss: 0.5774 - val_accuracy: 0.6767
Epoch 4/10
364/364 [==============================] - 47s 127ms/step - loss: 0.5671 - accuracy: 0.6865 - val_loss: 0.5641 - val_accuracy: 0.6813
Epoch 5/10
364/364 [==============================] - 47s 127ms/step - loss: 0.5470 - accuracy: 0.7088 - val_loss: 0.5543 - val_accuracy: 0.6948
Epoch 6/10
364/364 [==============================] - 48s 129ms/step - loss: 0.5363 - accuracy: 0.7186 - val_loss: 0.5489 - val_accuracy: 0.7050
Epoch 7/10
364/364 [==============================] - 48s 129ms/step - loss: 0.5237 - accuracy: 0.7317 - val_loss: 0.5422 - val_accuracy: 0.7053
Epoch 8/10
364/364 [==============================] - 47s 128ms/step - loss: 0.5150 - accuracy: 0.7349 - val_loss: 0.5385 - val_accuracy: 0.7053
Epoch 9/10
364/364 [==============================] - 48s 129ms/step - loss: 0.5092 - accuracy: 0.7412 - val_loss: 0.5348 - val_accuracy: 0.7121
Epoch 10/10
364/364 [==============================] - 48s 131ms/step - loss: 0.5007 - accuracy: 0.7492 - val_loss: 0.5290 - val_accuracy: 0.7299
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
```

```
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  384
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 efficientnetb3 (Functional)  (None, 5, 5, 1536)       10783535

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1536)              0

 dense (Dense)               (None, 1)                 1537

=================================================================
Total params: 10,785,072
```

```
Trainable params: 10,577,383
Non-trainable params: 207,689
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
252
```

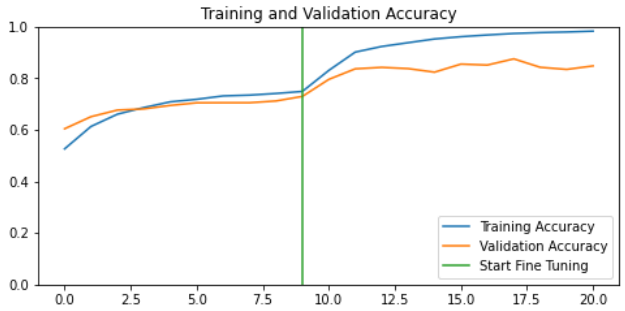
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
364/364 [==============================] - 103s 220ms/step - loss: 0.3711 - accuracy: 0.8309 - val_loss: 0.4788 - val_accuracy: 0.7960
Epoch 11/20
364/364 [==============================] - 77s 211ms/step - loss: 0.2343 - accuracy: 0.9019 - val_loss: 0.3978 - val_accuracy: 0.8370
Epoch 12/20
364/364 [==============================] - 78s 212ms/step - loss: 0.1749 - accuracy: 0.9237 - val_loss: 0.4242 - val_accuracy: 0.8426
Epoch 13/20
364/364 [==============================] - 79s 215ms/step - loss: 0.1388 - accuracy: 0.9384 - val_loss: 0.4624 - val_accuracy: 0.8378
Epoch 14/20
364/364 [==============================] - 78s 212ms/step - loss: 0.1083 - accuracy: 0.9529 - val_loss: 0.5737 - val_accuracy: 0.8240
Epoch 15/20
364/364 [==============================] - 78s 213ms/step - loss: 0.0896 - accuracy: 0.9618 - val_loss: 0.4318 - val_accuracy: 0.8553
Epoch 16/20
364/364 [==============================] - 77s 211ms/step - loss: 0.0766 - accuracy: 0.9685 - val_loss: 0.4954 - val_accuracy: 0.8517
Epoch 17/20
364/364 [==============================] - 78s 212ms/step - loss: 0.0615 - accuracy: 0.9742 - val_loss: 0.4333 - val_accuracy: 0.8757
Epoch 18/20
364/364 [==============================] - 78s 213ms/step - loss: 0.0540 - accuracy: 0.9777 - val_loss: 0.6245 - val_accuracy: 0.8429
Epoch 19/20
364/364 [==============================] - 78s 213ms/step - loss: 0.0483 - accuracy: 0.9800 - val_loss: 0.7137 - val_accuracy: 0.8347
Epoch 20/20
364/364 [==============================] - 78s 213ms/step - loss: 0.0429 - accuracy: 0.9834 - val_loss: 0.6226 - val_accuracy: 0.8483
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

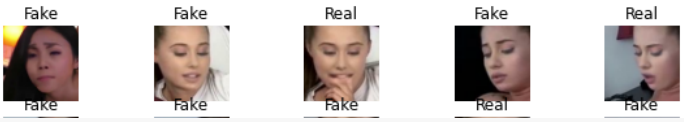
```
5/5 [==============================] - 36s 7s/step - loss: 0.8756 - accuracy: 0.8250
Test accuracy : 0.824999988079071
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
 0 0 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Step 10 - Determine the performance of the classifier

```
# Figures for performance calculation
TP = true_pos = 82 # real faces predicted as real
TN = true_neg = 78 # fake faces predicted as fake
```

```
FP = false_pos = 22 # fake faces predicted as real
FN = false_neg = 18  # real faces predicted as fake
```

```
results = {}
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
ACC is  0.800
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.820
```

```
# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.780
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.788
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.812
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.804
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.600
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
```

```
model.save('my_EfficientNet.h5')
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask layers require a config and must
  layer_config = serialize_layer_fn(layer)
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 efficientnetb3 (Functional)  (None, 5, 5, 1536)       10783535

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1536)              0

 dense (Dense)               (None, 1)                 1537

=================================================================
Total params: 10,785,072
Trainable params: 10,577,383
Non-trainable params: 207,689
_________________________________________________________________
```

End of Notebook

This is a notebook for training, validation and testing a binary classifier with EfficientNetB7 as the base model

Step 1 - Turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Step 2 - Dataset preprocessing

```
# Mount to Google drive in case the connection is lost
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Pivot to relevant directory
%cd '/content/drive/MyDrive/DFD/data/MyDB/'
```

```
/content/drive/MyDrive/DFD/data/MyDB
```

```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
```

```
# Set path
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training fake faces
train_fake_dir = os.path.join(train_dir, 'Fake')

# Directory with our training real pictures
train_real_dir = os.path.join(train_dir, 'Real')

# Directory with our validation fake pictures
validation_fake_dir = os.path.join(validation_dir, 'Fake')

# Directory with our validation real pictures
validation_real_dir = os.path.join(validation_dir, 'Real')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

train_fake_fnames = os.listdir(train_fake_dir)
train_real_fnames = os.listdir(train_real_dir)
```

```
# Print some file names to confirm the right directories are connected
train_fake_fnames = os.listdir(train_fake_dir)
print(train_fake_fnames[:6])

train_real_fnames = os.listdir(train_real_dir)
train_real_fnames.sort()
print(train_real_fnames[:6])
```

```
['df03389.jpg', 'df03464.jpg', 'df03384.jpg', 'df03373.jpg', 'df03453.jpg', 'df03535.jpg']
['abarnvbtwb.mp40.jpgFace.jpg', 'abarnvbtwb.mp4120.jpgFace.jpg', 'abarnvbtwb.mp4144.jpgFace.jpg', 'abarnvbtwb.mp4168.jpgFace.jpg', 'aba
```

```
# Count the number of real and fake images of faces in the train and validation directories
# Check against the figures in metadata file to confirm completeness

# For training, real images are 7,377 and fake images are 7,162.
```

```
print('total real images for training:', len(os.listdir(train_real_dir)))
print('total fake images for training:', len(os.listdir(train_fake_dir)))

# For validation, real images are 1,771 and fake images are 1,768.
print('total real images for validation:', len(os.listdir(validation_real_dir)))
print('total fake images for validation:', len(os.listdir(validation_fake_dir)))
```

```
total real images for training: 7377
total fake images for training: 7162
total real images for validation: 1771
total fake images for validation: 1768
```

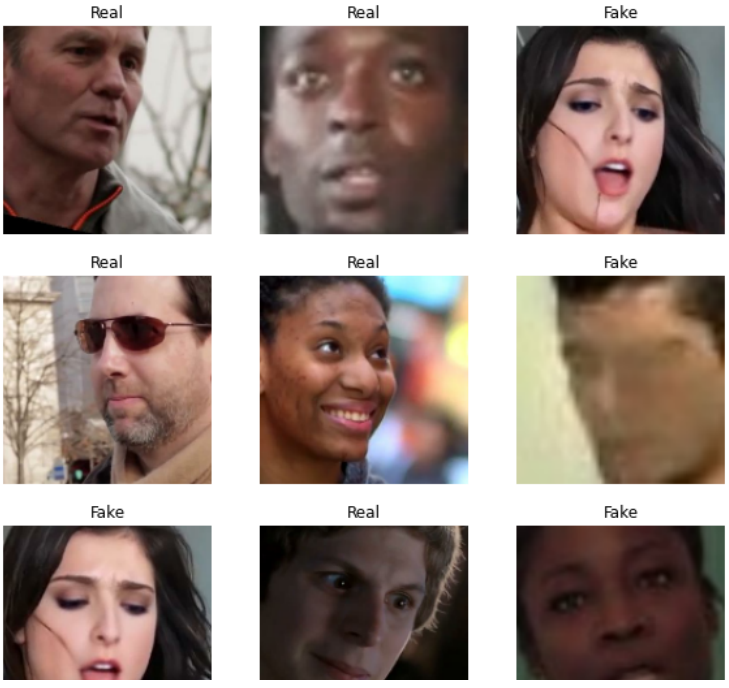
Step 3 - Data augmentation and normalisation

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)
```

```
train_dataset = tensorflow.keras.utils.image_dataset_from_directory(train_dir,
                                                                    shuffle=True,
                                                                    batch_size=BATCH_SIZE,
                                                                    image_size=IMG_SIZE)

validation_dataset = tensorflow.keras.utils.image_dataset_from_directory(validation_dir,
                                                                         shuffle=True,
                                                                         batch_size=BATCH_SIZE,
                                                                         image_size=IMG_SIZE)

test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 14539 files belonging to 2 classes.
Found 3539 files belonging to 2 classes.
Found 200 files belonging to 2 classes.
```

```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```




```
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tensorflow.keras.Sequential([
  tensorflow.keras.layers.RandomFlip('horizontal'),
  tensorflow.keras.layers.RandomRotation(0.2),
])
```

```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tensorflow.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```

Step 4 - Build the model

```
preprocess_input = tensorflow.keras.applications.efficientnet.preprocess_input
```

```
# Create the base model from the pre-trained model
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tensorflow.keras.applications.efficientnet.EfficientNetB7(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb7_notop.h5
258080768/258076736 [==============================] - 2s 0us/step
258088960/258076736 [==============================] - 2s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(40, 5, 5, 2560)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "efficientnetb7"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 150, 150, 3  0           []
                                )]
```

```
rescaling (Rescaling)          (None, 150, 150, 3)  0           ['input_1[0][0]']

normalization (Normalization)  (None, 150, 150, 3)  7           ['rescaling[0][0]']

stem_conv_pad (ZeroPadding2D)  (None, 151, 151, 3)  0           ['normalization[0][0]']

stem_conv (Conv2D)             (None, 75, 75, 64)   1728        ['stem_conv_pad[0][0]']

stem_bn (BatchNormalization)   (None, 75, 75, 64)   256         ['stem_conv[0][0]']

stem_activation (Activation)   (None, 75, 75, 64)   0           ['stem_bn[0][0]']

block1a_dwconv (DepthwiseConv2  (None, 75, 75, 64)  576         ['stem_activation[0][0]']
D)

block1a_bn (BatchNormalization  (None, 75, 75, 64)  256         ['block1a_dwconv[0][0]']
)

block1a_activation (Activation  (None, 75, 75, 64)  0           ['block1a_bn[0][0]']
)

block1a_se_squeeze (GlobalAver  (None, 64)          0           ['block1a_activation[0][0]']
agePooling2D)

block1a_se_reshape (Reshape)   (None, 1, 1, 64)     0           ['block1a_se_squeeze[0][0]']

block1a_se_reduce (Conv2D)     (None, 1, 1, 16)     1040        ['block1a_se_reshape[0][0]']

block1a_se_expand (Conv2D)     (None, 1, 1, 64)     1088        ['block1a_se_reduce[0][0]']

block1a_se_excite (Multiply)   (None, 75, 75, 64)   0           ['block1a_activation[0][0]',
                                                                 'block1a_se_expand[0][0]']

block1a_project_conv (Conv2D)  (None, 75, 75, 32)   2048        ['block1a_se_excite[0][0]']

block1a_project_bn (BatchNorma  (None, 75, 75, 32)  128         ['block1a_project_conv[0][0]']
lization)

block1b_dwconv (DepthwiseConv2  (None, 75, 75, 32)  288         ['block1a_project_bn[0][0]']
D)

block1b_bn (BatchNormalization  (None, 75, 75, 32)  128         ['block1b_dwconv[0][0]']
)

block1b_activation (Activation  (None, 75, 75, 32)  0           ['block1b_bn[0][0]']
)

block1b_se_squeeze (GlobalAver  (None, 32)          0           ['block1b_activation[0][0]']
agePooling2D)

block1b_se_reshape (Reshape)   (None, 1, 1, 32)     0           ['block1b_se_squeeze[0][0]']
```

```
global_average_layer = tensorflow.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(40, 2560)
```

```
prediction_layer = tensorflow.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(40, 1)
```

```
inputs = tensorflow.keras.Input(shape=(150, 150, 3))

x = data_augmentation(inputs)

x = preprocess_input(x)

x = base_model(x, training=False)

x = global_average_layer(x)

x = tensorflow.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tensorflow.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 efficientnetb7 (Functional)  (None, 5, 5, 2560)       64097687

 global_average_pooling2d (G  (None, 2560)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2560)              0

 dense (Dense)               (None, 1)                 2561

=================================================================
Total params: 64,100,248
Trainable params: 2,561
Non-trainable params: 64,097,687
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
2
```

Step 5 - Train the model

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
89/89 [==============================] - 225s 2s/step - loss: 0.7037 - accuracy: 0.5032
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.70
initial accuracy: 0.50
```

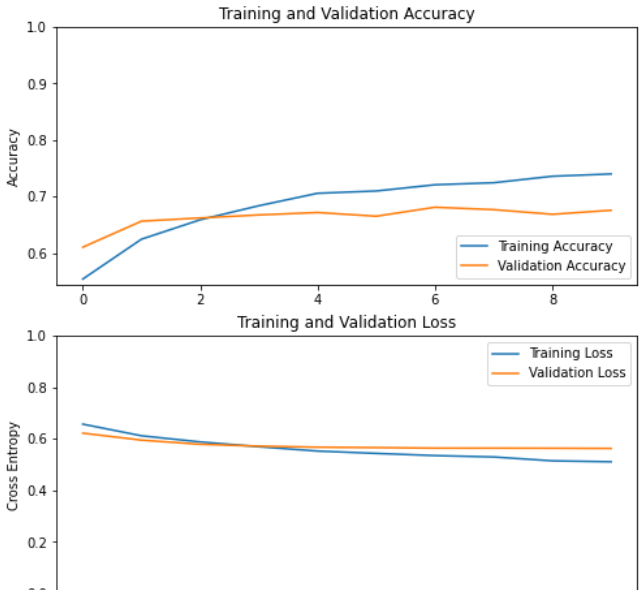
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
364/364 [==============================] - 868s 2s/step - loss: 0.6569 - accuracy: 0.5540 - val_loss: 0.6218 - val_accuracy: 0.6103
Epoch 2/10
364/364 [==============================] - 71s 195ms/step - loss: 0.6116 - accuracy: 0.6245 - val_loss: 0.5948 - val_accuracy: 0.6564
Epoch 3/10
364/364 [==============================] - 71s 194ms/step - loss: 0.5878 - accuracy: 0.6588 - val_loss: 0.5786 - val_accuracy: 0.6621
Epoch 4/10
364/364 [==============================] - 71s 194ms/step - loss: 0.5691 - accuracy: 0.6837 - val_loss: 0.5713 - val_accuracy: 0.6674
Epoch 5/10
364/364 [==============================] - 71s 194ms/step - loss: 0.5522 - accuracy: 0.7057 - val_loss: 0.5673 - val_accuracy: 0.6717
Epoch 6/10
364/364 [==============================] - 71s 193ms/step - loss: 0.5432 - accuracy: 0.7097 - val_loss: 0.5658 - val_accuracy: 0.6652
Epoch 7/10
364/364 [==============================] - 71s 193ms/step - loss: 0.5347 - accuracy: 0.7208 - val_loss: 0.5636 - val_accuracy: 0.6810
Epoch 8/10
364/364 [==============================] - 71s 193ms/step - loss: 0.5294 - accuracy: 0.7243 - val_loss: 0.5636 - val_accuracy: 0.6767
Epoch 9/10
364/364 [==============================] - 71s 194ms/step - loss: 0.5143 - accuracy: 0.7358 - val_loss: 0.5633 - val_accuracy: 0.6686
Epoch 10/10
364/364 [==============================] - 71s 194ms/step - loss: 0.5107 - accuracy: 0.7399 - val_loss: 0.5625 - val_accuracy: 0.6756
```

Step 6 - Evaluate accuracy and loss for the model

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
```

```
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 7 - Fine-tuning

```
base_model.trainable = True
```

```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  813
```

```
model.compile(loss=tensorflow.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tensorflow.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 efficientnetb7 (Functional)  (None, 5, 5, 2560)       64097687

 global_average_pooling2d (G  (None, 2560)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2560)              0

 dense (Dense)               (None, 1)                 2561

=================================================================
Total params: 64,100,248
```

```
Trainable params: 63,663,721
Non-trainable params: 436,527
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
626
```

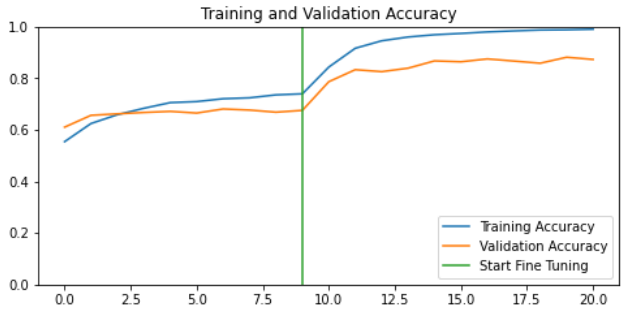
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
364/364 [==============================] - 280s 637ms/step - loss: 0.3391 - accuracy: 0.8439 - val_loss: 0.5489 - val_accuracy: 0.7869
Epoch 11/20
364/364 [==============================] - 228s 625ms/step - loss: 0.1901 - accuracy: 0.9168 - val_loss: 0.4830 - val_accuracy: 0.8336
Epoch 12/20
364/364 [==============================] - 228s 626ms/step - loss: 0.1287 - accuracy: 0.9458 - val_loss: 0.5669 - val_accuracy: 0.8259
Epoch 13/20
364/364 [==============================] - 228s 627ms/step - loss: 0.0963 - accuracy: 0.9606 - val_loss: 0.5291 - val_accuracy: 0.8395
Epoch 14/20
364/364 [==============================] - 229s 628ms/step - loss: 0.0734 - accuracy: 0.9694 - val_loss: 0.4230 - val_accuracy: 0.8678
Epoch 15/20
364/364 [==============================] - 228s 627ms/step - loss: 0.0611 - accuracy: 0.9744 - val_loss: 0.4491 - val_accuracy: 0.8644
Epoch 16/20
364/364 [==============================] - 229s 627ms/step - loss: 0.0490 - accuracy: 0.9803 - val_loss: 0.4655 - val_accuracy: 0.8754
Epoch 17/20
364/364 [==============================] - 229s 627ms/step - loss: 0.0397 - accuracy: 0.9843 - val_loss: 0.5440 - val_accuracy: 0.8672
Epoch 18/20
364/364 [==============================] - 229s 628ms/step - loss: 0.0299 - accuracy: 0.9877 - val_loss: 0.7666 - val_accuracy: 0.8587
Epoch 19/20
364/364 [==============================] - 229s 628ms/step - loss: 0.0281 - accuracy: 0.9886 - val_loss: 0.5677 - val_accuracy: 0.8819
Epoch 20/20
364/364 [==============================] - 229s 628ms/step - loss: 0.0244 - accuracy: 0.9909 - val_loss: 0.6897 - val_accuracy: 0.8734
```

Step 8 - Evaluate accuracy and loss for the fine-tuned model

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 9 - Make prediction with unseen test dataset

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

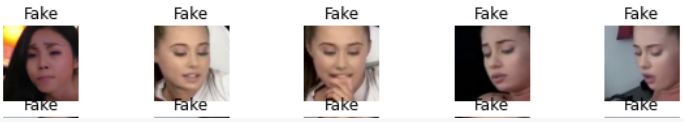
```
5/5 [==============================] - 12s 2s/step - loss: 0.8585 - accuracy: 0.8150
Test accuracy : 0.8149999976158142
```

```
# For testing of unseen images, there are 100 real images of faces and 100 fake images of faces.
base_dir = '/content/drive/MyDrive/DFD/data/MyDB'
test_dir = os.path.join(base_dir, 'test')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'Fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'Real')

print('total real images for test:', len(os.listdir(test_real_dir)))
print('total fake images for test:', len(os.listdir(test_fake_dir)))
```

```
total real images for test: 100
total fake images for test: 100
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(40):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```




```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 34_0.jpg to 34_0.jpg
Saving 34_6.jpg to 34_6.jpg
Saving 35_0.jpg to 35_0.jpg
Saving 39_360.jpg to 39_360.jpg
Saving 39_372.jpg to 39_372.jpg
Saving 39_384.jpg to 39_384.jpg
Saving 39_402.jpg to 39_402.jpg
Saving 39_780.jpg to 39_780.jpg
Saving 39_786.jpg to 39_786.jpg
Saving 40_0.jpg to 40_0.jpg
Saving 40_12.jpg to 40_12.jpg
Saving 40_24.jpg to 40_24.jpg
Saving 42_168.jpg to 42_168.jpg
Saving 42_198.jpg to 42_198.jpg
Saving 42_204.jpg to 42_204.jpg
Saving 43_36.jpg to 43_36.jpg
Saving 43_150.jpg to 43_150.jpg
Saving 43_192.jpg to 43_192.jpg
Saving 43_210.jpg to 43_210.jpg
Saving 44_6.jpg to 44_6.jpg
Saving 44_90.jpg to 44_90.jpg
Saving 44_192.jpg to 44_192.jpg
Saving 44_198.jpg to 44_198.jpg
Saving 45_60.jpg to 45_60.jpg
Saving 45_150.jpg to 45_150.jpg
Saving 46_360.jpg to 46_360.jpg
Saving 46_384.jpg to 46_384.jpg
Saving 47_528.jpg to 47_528.jpg
Saving 47_738.jpg to 47_738.jpg
Saving 48_12.jpg to 48_12.jpg
Saving 48_270.jpg to 48_270.jpg
Saving 48_318.jpg to 48_318.jpg
Saving 48_522.jpg to 48_522.jpg
Saving 48_708.jpg to 48_708.jpg
Saving 50_192.jpg to 50_192.jpg
Saving 50_222.jpg to 50_222.jpg
Saving 51_120.jpg to 51_120.jpg
Saving 51_300.jpg to 51_300.jpg
Saving 51_372.jpg to 51_372.jpg
Saving 52_42.jpg to 52_42.jpg
Saving 52_450.jpg to 52_450.jpg
Saving 53_66.jpg to 53_66.jpg
Saving 54_18.jpg to 54_18.jpg
Saving 54_186.jpg to 54_186.jpg
Saving 69_268.jpg to 69_268.jpg
Saving 69_302.jpg to 69_302.jpg
Saving 78_3.jpg to 78_3.jpg
Saving 78_318.jpg to 78_318.jpg
Saving 96_30.jpg to 96_30.jpg
Saving 96_32.jpg to 96_32.jpg
Saving 97_137.jpg to 97_137.jpg
Saving 97_146.jpg to 97_146.jpg
Saving 98_62.jpg to 98_62.jpg
Saving 99_108.jpg to 99_108.jpg
Saving 100_168.jpg to 100_168.jpg
Saving 100_956.jpg to 100_956.jpg
Saving 100_972.jpg to 100_972.jpg
Saving 101_6.jpg to 101_6.jpg
Saving 101_8.jpg to 101_8.jpg
Saving 101_10.jpg to 101_10.jpg
Saving 101_40.jpg to 101_40.jpg
Saving 101_488.jpg to 101_488.jpg
Saving 101_490.jpg to 101_490.jpg
Saving 102_14.jpg to 102_14.jpg
Saving 102_16.jpg to 102_16.jpg
Saving 102_320.jpg to 102_320.jpg
Saving id19_0000.mp40.jpgFace.jpg to id19_0000.mp40.jpgFace.jpg
Saving id19_0001.mp4288.jpgFace.jpg to id19_0001.mp4288.jpgFace.jpg
Saving id19_0001.mp4312.jpgFace.jpg to id19_0001.mp4312.jpgFace.jpg
Saving id19_0005.mp4312.jpgFace.jpg to id19_0005.mp4312.jpgFace.jpg
Saving id19_0006.mp40.jpgFace.jpg to id19_0006.mp40.jpgFace.jpg
Saving id19_0006.mp424.jpgFace.jpg to id19_0006.mp424.jpgFace.jpg
Saving id19_0006.mp448.jpgFace.jpg to id19_0006.mp448.jpgFace.jpg
Saving id19_0006.mp472.jpgFace.jpg to id19_0006.mp472.jpgFace.jpg
Saving id19_0006.mp496.jpgFace.jpg to id19_0006.mp496.jpgFace.jpg
Saving id19_0006.mp4120.jpgFace.jpg to id19_0006.mp4120.jpgFace.jpg
Saving id19_0006.mp4144.jpgFace.jpg to id19_0006.mp4144.jpgFace.jpg
Saving id20_0002.mp424.jpgFace.jpg to id20_0002.mp424.jpgFace.jpg
Saving id20_0002.mp448.jpgFace.jpg to id20_0002.mp448.jpgFace.jpg
Saving id20_0002.mp472.jpgFace.jpg to id20_0002.mp472.jpgFace.jpg
Saving id20_0002.mp496.jpgFace.jpg to id20_0002.mp496.jpgFace.jpg
Saving id20_0002.mp4120.jpgFace.jpg to id20_0002.mp4120.jpgFace.jpg
Saving id20_0002.mp4144.jpgFace.jpg to id20_0002.mp4144.jpgFace.jpg
Saving id23_0007.mp4144.jpgFace.jpg to id23_0007.mp4144.jpgFace.jpg
Saving id23_0008.mp4168.jpgFace.jpg to id23_0008.mp4168.jpgFace.jpg
Saving id23_0008.mp4192.jpgFace.jpg to id23_0008.mp4192.jpgFace.jpg
Saving id23_0008.mp4216.jpgFace.jpg to id23_0008.mp4216.jpgFace.jpg
Saving id23_0008.mp4240.jpgFace.jpg to id23_0008.mp4240.jpgFace.jpg
Saving id23_0008.mp4264.jpgFace.jpg to id23_0008.mp4264.jpgFace.jpg
Saving id23_0008.mp4288.jpgFace.jpg to id23_0008.mp4288.jpgFace.jpg
```

```
Saving id23_0009.mp40.jpgFace.jpg to id23_0009.mp40.jpgFace.jpg
Saving id23_0009.mp424.jpgFace.jpg to id23_0009.mp424.jpgFace.jpg
Saving id23_0009.mp448.jpgFace.jpg to id23_0009.mp448.jpgFace.jpg
Saving id23_0009.mp472.jpgFace.jpg to id23_0009.mp472.jpgFace.jpg
Saving id24_0003.mp40.jpgFace.jpg to id24_0003.mp40.jpgFace.jpg
Saving id24_0003.mp472.jpgFace.jpg to id24_0003.mp472.jpgFace.jpg
Saving id24_0003.mp4120.jpgFace.jpg to id24_0003.mp4120.jpgFace.jpg
Saving id24_0003.mp4144.jpgFace.jpg to id24_0003.mp4144.jpgFace.jpg
Saving id25_0004.mp4288.jpgFace.jpg to id25_0004.mp4288.jpgFace.jpg
Saving id25_0004.mp4312.jpgFace.jpg to id25_0004.mp4312.jpgFace.jpg
34_0.jpg is deepfake
34_6.jpg is deepfake
35_0.jpg is real
39_360.jpg is real
39_372.jpg is real
39_384.jpg is real
39_402.jpg is deepfake
39_780.jpg is deepfake
39_786.jpg is deepfake
40_0.jpg is real
40_12.jpg is real
40_24.jpg is real
42_168.jpg is deepfake
42_198.jpg is real
42_204.jpg is real
43_36.jpg is real
43_150.jpg is deepfake
43_192.jpg is deepfake
43_210.jpg is deepfake
44_6.jpg is deepfake
44_90.jpg is deepfake
44_192.jpg is deepfake
44_198.jpg is deepfake
45_60.jpg is deepfake
45_150.jpg is real
46_360.jpg is real
46_384.jpg is real
47_528.jpg is deepfake
47_738.jpg is real
48_12.jpg is real
48_270.jpg is deepfake
48_318.jpg is real
48_522.jpg is real
48_708.jpg is real
50_192.jpg is real
50_222.jpg is real
51_120.jpg is deepfake
51_300.jpg is real
51_372.jpg is real
52_42.jpg is real
52_450.jpg is real
53_66.jpg is real
54_18.jpg is real
54_186.jpg is real
69_268.jpg is real
69_302.jpg is real
78_3.jpg is real
78_318.jpg is real
96_30.jpg is real
96_32.jpg is real
97_137.jpg is real
97_146.jpg is real
98_62.jpg is real
99_108.jpg is real
100_168.jpg is deepfake
100_956.jpg is real
100_972.jpg is real
101_6.jpg is real
101_8.jpg is real
101_10.jpg is real
101_40.jpg is real
101_488.jpg is real
101_490.jpg is real
102_14.jpg is real
102_16.jpg is real
102_320.jpg is deepfake
id19_0000.mp40.jpgFace.jpg is real
id19_0001.mp4288.jpgFace.jpg is real
id19_0001.mp4312.jpgFace.jpg is real
id19_0005.mp4312.jpgFace.jpg is real
id19_0006.mp40.jpgFace.jpg is real
id19_0006.mp424.jpgFace.jpg is real
id19_0006.mp448.jpgFace.jpg is real
id19_0006.mp472.jpgFace.jpg is real
id19_0006.mp496.jpgFace.jpg is real
id19_0006.mp4120.jpgFace.jpg is deepfake
id19_0006.mp4144.jpgFace.jpg is real
id20_0002.mp424.jpgFace.jpg is real
id20_0002.mp448.jpgFace.jpg is real
id20_0002.mp472.jpgFace.jpg is real
id20_0002.mp496.jpgFace.jpg is real
id20_0002.mp4120.jpgFace.jpg is real
id20_0002.mp4144.jpgFace.jpg is real
```

```
    id23_0007.mp4144.jpgFace.jpg is real
    id23_0008.mp4168.jpgFace.jpg is deepfake
    id23_0008.mp4192.jpgFace.jpg is deepfake
    id23_0008.mp4216.jpgFace.jpg is deepfake
    id23_0008.mp4240.jpgFace.jpg is deepfake
    id23_0008.mp4264.jpgFace.jpg is real
    id23_0008.mp4288.jpgFace.jpg is deepfake
    id23_0009.mp40.jpgFace.jpg is deepfake
    id23_0009.mp424.jpgFace.jpg is real
    id23_0009.mp448.jpgFace.jpg is deepfake
    id23_0009.mp472.jpgFace.jpg is real
    id24_0003.mp40.jpgFace.jpg is deepfake
    id24_0003.mp472.jpgFace.jpg is deepfake
    id24_0003.mp4120.jpgFace.jpg is deepfake
    id24_0003.mp4144.jpgFace.jpg is deepfake
    id25_0004.mp4288.jpgFace.jpg is deepfake
    id25_0004.mp4312.jpgFace.jpg is deepfake
```

```
# Upload the deepfake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/drive/MyDrive/DFD/data/MyDB/test/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=1)
```

```
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 114_666.jpg to 114_666.jpg
Saving 115_306.jpg to 115_306.jpg
Saving 115_816.jpg to 115_816.jpg
Saving 116_24.jpg to 116_24.jpg
Saving 116_1056.jpg to 116_1056.jpg
Saving 117_12.jpg to 117_12.jpg
Saving 117_750.jpg to 117_750.jpg
Saving 118_9.jpg to 118_9.jpg
Saving 118_405.jpg to 118_405.jpg
Saving 119_65.jpg to 119_65.jpg
Saving 119_660.jpg to 119_660.jpg
Saving 120_50.jpg to 120_50.jpg
Saving 120_235.jpg to 120_235.jpg
Saving 121_24.jpg to 121_24.jpg
Saving 121_28.jpg to 121_28.jpg
Saving 122_0.jpg to 122_0.jpg
Saving 122_1.jpg to 122_1.jpg
Saving 124_154.jpg to 124_154.jpg
Saving 124_158.jpg to 124_158.jpg
Saving 124_160.jpg to 124_160.jpg
Saving 124_170.jpg to 124_170.jpg
Saving 125_65.jpg to 125_65.jpg
Saving 125_68.jpg to 125_68.jpg
Saving 125_69.jpg to 125_69.jpg
Saving 130_2.jpg to 130_2.jpg
Saving 130_34.jpg to 130_34.jpg
Saving 130_50.jpg to 130_50.jpg
Saving 130_78.jpg to 130_78.jpg
Saving 131_172.jpg to 131_172.jpg
Saving 131_212.jpg to 131_212.jpg
Saving 132_36.jpg to 132_36.jpg
Saving 132_105.jpg to 132_105.jpg
Saving 132_237.jpg to 132_237.jpg
Saving 133_144.jpg to 133_144.jpg
Saving 134_90.jpg to 134_90.jpg
Saving 134_190.jpg to 134_190.jpg
Saving 134_555.jpg to 134_555.jpg
Saving 135_370.jpg to 135_370.jpg
Saving 135_830.jpg to 135_830.jpg
Saving 135_1080.jpg to 135_1080.jpg
Saving 142_396.jpg to 142_396.jpg
Saving 142_412.jpg to 142_412.jpg
Saving 142_420.jpg to 142_420.jpg
Saving 143_23.jpg to 143_23.jpg
Saving 143_25.jpg to 143_25.jpg
Saving 143_26.jpg to 143_26.jpg
Saving 143_27.jpg to 143_27.jpg
Saving 143_29.jpg to 143_29.jpg
Saving 143_32.jpg to 143_32.jpg
Saving 143_33.jpg to 143_33.jpg
Saving 143_34.jpg to 143_34.jpg
Saving 143_39.jpg to 143_39.jpg
Saving 143_41.jpg to 143_41.jpg
Saving 143_43.jpg to 143_43.jpg
Saving 143_44.jpg to 143_44.jpg
Saving 143_46.jpg to 143_46.jpg
Saving 143_48.jpg to 143_48.jpg
Saving 143_52.jpg to 143_52.jpg
Saving 166_71.jpg to 166_71.jpg
Saving 166_74.jpg to 166_74.jpg
Saving 166_76.jpg to 166_76.jpg
Saving 167_24.jpg to 167_24.jpg
Saving 172_600.jpg to 172_600.jpg
Saving 172_610.jpg to 172_610.jpg
Saving 172_615.jpg to 172_615.jpg
Saving cele16396.jpgF.jpg to cele16396.jpgF.jpg
Saving cele16748.jpgF.jpg to cele16748.jpgF.jpg
Saving cele17100.jpgF.jpg to cele17100.jpgF.jpg
Saving cele17543.jpgF.jpg to cele17543.jpgF.jpg
Saving cele17750.jpgF.jpg to cele17750.jpgF.jpg
Saving cele17783.jpgF.jpg to cele17783.jpgF.jpg
Saving cele18188.jpgF.jpg to cele18188.jpgF.jpg
Saving cele18514.jpgF.jpg to cele18514.jpgF.jpg
Saving cele18909.jpgF.jpg to cele18909.jpgF.jpg
Saving cele33231.jpgF.jpg to cele33231.jpgF.jpg
Saving cele33349.jpgF.jpg to cele33349.jpgF.jpg
Saving cele33353.jpgF.jpg to cele33353.jpgF.jpg
Saving cele33363.jpgF.jpg to cele33363.jpgF.jpg
Saving cele33385.jpgF.jpg to cele33385.jpgF.jpg
Saving cele33393.jpgF.jpg to cele33393.jpgF.jpg
Saving cele33483.jpgF.jpg to cele33483.jpgF.jpg
Saving cele33493.jpgF.jpg to cele33493.jpgF.jpg
Saving cele33499.jpgF.jpg to cele33499.jpgF.jpg
Saving cele33502.jpgF.jpg to cele33502.jpgF.jpg
Saving cele34056.jpgF.jpg to cele34056.jpgF.jpg
Saving cele34064.jpgF.jpg to cele34064.jpgF.jpg
Saving cele34066.jpgF.jpg to cele34066.jpgF.jpg
Saving cele34070.jpgF.jpg to cele34070.jpgF.jpg
Saving cele34073.jpgF.jpg to cele34073.jpgF.jpg
Saving cele35162.jpgF.jpg to cele35162.jpgF.jpg
```

```
Saving cele35173.jpgF.jpg to cele35173.jpgF.jpg
Saving cele35180.jpgF.jpg to cele35180.jpgF.jpg
Saving cele35320.jpgF.jpg to cele35320.jpgF.jpg
Saving cele35328.jpgF.jpg to cele35328.jpgF.jpg
Saving cele35330.jpgF.jpg to cele35330.jpgF.jpg
Saving cele35347.jpgF.jpg to cele35347.jpgF.jpg
Saving cele35353.jpgF.jpg to cele35353.jpgF.jpg
Saving cele35370.jpgF.jpg to cele35370.jpgF.jpg
Saving cele35374.jpgF.jpg to cele35374.jpgF.jpg
Saving cele35383.jpgF.jpg to cele35383.jpgF.jpg
114_666.jpg is deepfake
115_306.jpg is deepfake
115_816.jpg is deepfake
116_24.jpg is deepfake
116_1056.jpg is deepfake
117_12.jpg is deepfake
117_750.jpg is deepfake
118_9.jpg is real
118_405.jpg is deepfake
119_65.jpg is deepfake
119_660.jpg is deepfake
120_50.jpg is deepfake
120_235.jpg is deepfake
121_24.jpg is deepfake
121_28.jpg is deepfake
122_0.jpg is deepfake
122_1.jpg is deepfake
124_154.jpg is deepfake
124_158.jpg is deepfake
124_160.jpg is deepfake
124_170.jpg is deepfake
125_65.jpg is deepfake
125_68.jpg is deepfake
125_69.jpg is deepfake
130_2.jpg is deepfake
130_34.jpg is deepfake
130_50.jpg is deepfake
130_78.jpg is deepfake
131_172.jpg is deepfake
131_212.jpg is deepfake
132_36.jpg is real
132_105.jpg is real
132_237.jpg is deepfake
133_144.jpg is deepfake
134_90.jpg is deepfake
134_190.jpg is deepfake
134_555.jpg is deepfake
135_370.jpg is deepfake
135_830.jpg is deepfake
135_1080.jpg is deepfake
142_396.jpg is deepfake
142_412.jpg is deepfake
142_420.jpg is deepfake
143_23.jpg is deepfake
143_25.jpg is deepfake
143_26.jpg is real
143_27.jpg is real
143_29.jpg is deepfake
143_32.jpg is deepfake
143_33.jpg is deepfake
143_34.jpg is deepfake
143_39.jpg is deepfake
143_41.jpg is deepfake
143_43.jpg is deepfake
143_44.jpg is deepfake
143_46.jpg is deepfake
143_48.jpg is deepfake
143_52.jpg is deepfake
166_71.jpg is deepfake
166_74.jpg is deepfake
166_76.jpg is deepfake
167_24.jpg is deepfake
172_600.jpg is deepfake
172_610.jpg is deepfake
172_615.jpg is deepfake
cele16396.jpgF.jpg is deepfake
cele16748.jpgF.jpg is deepfake
cele17100.jpgF.jpg is deepfake
cele17543.jpgF.jpg is deepfake
cele17750.jpgF.jpg is deepfake
cele17783.jpgF.jpg is real
cele18188.jpgF.jpg is deepfake
cele18514.jpgF.jpg is real
cele18909.jpgF.jpg is real
cele33231.jpgF.jpg is deepfake
cele33349.jpgF.jpg is deepfake
cele33353.jpgF.jpg is deepfake
cele33363.jpgF.jpg is deepfake
cele33385.jpgF.jpg is deepfake
cele33393.jpgF.jpg is deepfake
cele33483.jpgF.jpg is deepfake
cele33493.jpgF.jpg is deepfake
cele33499.jpgF.jpg is deepfake
```

```
cele33502.jpgF.jpg is deepfake
cele34056.jpgF.jpg is deepfake
cele34064.jpgF.jpg is deepfake
cele34066.jpgF.jpg is deepfake
cele34070.jpgF.jpg is deepfake
cele34073.jpgF.jpg is deepfake
cele35162.jpgF.jpg is deepfake
cele35173.jpgF.jpg is deepfake
cele35180.jpgF.jpg is deepfake
cele35320.jpgF.jpg is deepfake
cele35328.jpgF.jpg is deepfake
cele35330.jpgF.jpg is deepfake
cele35347.jpgF.jpg is deepfake
cele35353.jpgF.jpg is deepfake
cele35370.jpgF.jpg is deepfake
cele35374.jpgF.jpg is deepfake
cele35383.jpgF.jpg is deepfake
```

Step 10 - Determine the performance of the classifier

```
# Figures for performance calculation
# real = 1, fake = 0
TP = true_pos = 67 # real faces predicted as real
TN = true_neg = 92 # fake faces predicted as fake
FP = false_pos = 8 # fake faces predicted as real
FN = false_neg = 33  # real faces predicted as fake
```

```
results = {}
```

```
# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
ACC is  0.795
```

```
# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.670
```

```
# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.920
```

```
# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.893
```

```
# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.736
```

```
# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.766
```

```
# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.609
```

Step 11 - Save model

```
# Requirement to save the entire model (include architecture, weights, and training configuration in a single file/folder)
!pip install pyyaml h5py
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
```

```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_EfficientNetB7.h5')
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask layers require a config and must
  layer_config = serialize_layer_fn(layer)
```

```
# Print a summary of model architecture
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 efficientnetb7 (Functional)  (None, 5, 5, 2560)       64097687

 global_average_pooling2d (G  (None, 2560)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2560)              0

 dense (Dense)               (None, 1)                 2561

=================================================================
Total params: 64,100,248
Trainable params: 63,663,721
Non-trainable params: 436,527
_________________________________________________________________
```

End of Notebook

This is a notebook for using all trained models for final testing to check its effectiveness on classifying real videos or deepfake videos in the wild

Step 1 - In a new Google Colab notebook, turn on GPU in runtime

```
# Check GPU information
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Thu Dec  9 10:38:47 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 495.44       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   37C    P0    33W / 250W |   8795MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Step 2 - Upload the video and model to Google Drive

Create several folders in Google Drive

Save the videos (.mp4) and model (.h5) inside the following three folders:

- My Drive/MyVideo/Real/yourrealvideo.mp4
- My Drive/MyVideo/Fake/yourfakevideo.mp4
- My Drive/MyModel/mytrainedmodel.h5

Also create the following folders for keeping the frames and chopped faces during image processing.

- My Drive/MyVideo/Frames/Real/realframestobecreated.jpg
- My Drive/MyVideo/Frames/Fake/fakeframestobecreated.jpg
- My Drive/MyVideo/Faces/Real/realfacestobecreated.jpg
- My Drive/MyVideo/Faces/Fake/fakefacestobecreated.jpg

```
# Confirm Google Drive is connected with Google Colab
google = !if [ -d 'GDrive/' ]; then echo "1" ; else echo "0"; fi
if (google[0] is '0' ):
  from google.colab import drive
  drive.mount('/content/GDrive/')

!if [ -d 'GDrive/' ]; then echo "Connection to Google drive successful" ; else echo "Error to connect to Google drive"; fi
```

```
Mounted at /content/GDrive/
Connection to Google drive successful
```

```
# List the video files and model in Google drive which are to be transferred to a temporary directory in Google Colab
!ls GDrive/My\ Drive/MyVideo/Real/
!ls GDrive/My\ Drive/MyVideo/Fake/
!ls GDrive/My\ Drive/MyModel/
```

```
BO_1.mp4  EM_1.mp4  JL_1.mp4  QE_1.mp4  TC_1.mp4
BO_0.mp4  EM_0.mp4  JL_0.mp4  QE_0.mp4  TC_0.mp4
my_DenseNet.h5        my_InceptionResNetModel.h5  my_VGG19.h5
my_EfficientNetB3.h5  my_MobileNetModel.h5        my_Xception.h5
my_EfficientNetB7.h5  my_NASNetMobile.h5
my_InceptionNetV3.h5  my_ResNet152V2.h5
```

Step 3 - Create a temporary directory in Google Colab. Copy the mp4 files and the model from your Google Drive to Google Colab

```
# Set up environment and import libraries in Google Colab
!pip install --upgrade pip > /dev/null
!pip install scikit-image
!pip install opencv-python
```

```
Requirement already satisfied: scikit-image in /usr/local/lib/python3.7/dist-packages (0.18.3)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.7/dist-packages (from scikit-image) (1.2.0)
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image) (2.6.3)
Requirement already satisfied: scipy>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from scikit-image) (1.4.1)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,>=4.3.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image) (7.1.2)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image) (3.2.2)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.7/dist-packages (from scikit-image) (1.19.5)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.7/dist-packages (from scikit-image) (2021.11.2)
Requirement already satisfied: imageio>=2.3.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image) (2.4.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image) (
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-ima
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib!=3.0.0,>=2.0.
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.19.5)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
```

```
# Import libraries and modules in Google Colab
import cv2
from skimage.transform import resize
import matplotlib.pyplot as plt
import math
import os
```

```
# Make a temporarty directory in Google colab
!mkdir -p /content/DetectionTest/Sources/Real > /dev/null
!mkdir -p /content/DetectionTest/Sources/Fake > /dev/null
!mkdir -p /content/DetectionTest/Sources/frames/real > /dev/null
!mkdir -p /content/DetectionTest/Sources/frames/fake > /dev/null
!mkdir -p /content/DetectionTest/Sources/faces/real > /dev/null
!mkdir -p /content/DetectionTest/Sources/faces/fake > /dev/null
!mkdir -p /content/DetectionTest/Model > /dev/null

# copy video stored in Google Drive to Google Colab
!cp GDrive/My\ Drive/MyVideo/Real/*.mp4 DetectionTest/Sources/Real/
!cp GDrive/My\ Drive/MyVideo/Fake/*.mp4 DetectionTest/Sources/Fake/

# copy model stored in Google Drive to Google Colab
!cp GDrive/My\ Drive/MyModel/*.h5 DetectionTest/Model/

# List files Google Colab's temporary directory
!ls /content/DetectionTest/Sources/Real
!ls /content/DetectionTest/Sources/Fake
!ls /content/DetectionTest/Model
```

```
BO_1.mp4  EM_1.mp4  JL_1.mp4  QE_1.mp4  TC_1.mp4
BO_0.mp4  EM_0.mp4  JL_0.mp4  QE_0.mp4  TC_0.mp4
my_DenseNet.h5        my_InceptionResNetModel.h5  my_VGG19.h5
my_EfficientNetB3.h5  my_MobileNetModel.h5        my_Xception.h5
my_EfficientNetB7.h5  my_NASNetMobile.h5
my_InceptionNetV3.h5  my_ResNet152V2.h5
```

Step 4 - Extract frames from the video

Real videos

```
# Folder path contains the videos
INPUT_VIDEOS_PATH = '/content/DetectionTest/Sources/Real'

# Folder path for storing image frames to be extracted from videos
OUTPUT_FRAMES_PATH = '/content/DetectionTest/Sources/frames/real'

# Create variables
one_frame_each = 120                                                    # Para = 24, Extract 1 frame from a video for every 24 frame

!if [ -d {OUTPUT_FRAMES_PATH} ]; then echo "Output to be stored in "{OUTPUT_FRAMES_PATH} ; else mkdir {OUTPUT_FRAMES_PATH} && echo "Output d

videofiles = !ls {INPUT_VIDEOS_PATH}/*.mp4                              # Video file names in INPUT VIDEOS PATH
```

```
Output to be stored in /content/DetectionTest/Sources/frames/real
```

```
for videofile in videofiles:
    count = 0
    success = True
    filename = os.path.basename(videofile)
    vidcap = cv2.VideoCapture(videofile)

    while success:
        if (count%one_frame_each == 0):                                          # checks frame number and keeps one_frame_each
            success, image = vidcap.read()                                        # reads next frame
            if image.shape[1]>640:                                               # if image width > 640, resize it
                tmp = resize(image, (math.floor(640 / image.shape[1] * image.shape[0]), 640))
            plt.imsave("%s/%s%d.jpg" % (OUTPUT_FRAMES_PATH,filename,count), tmp, format='jpg') # saves images to frame folder
            print ('*', end="")
        else:
            success,image = vidcap.read()                                        # reads next frame
        count += 1                                                               # loops counter
```

```
**************
```

```
# List the image frames in Google Colab's temporary directory
!ls /content/DetectionTest/Sources/frames/real

# Copy the frames to Google Drive
!cp /content/DetectionTest/Sources/frames/real/* GDrive/My\ Drive/MyVideo/Frames/Real
```

```
BO_1.mp40.jpg    EM_1.mp4120.jpg  JL_1.mp4240.jpg  TC_1.mp40.jpg
BO_1.mp4120.jpg  EM_1.mp4240.jpg  QE_1.mp40.jpg    TC_1.mp4120.jpg
BO_1.mp4240.jpg  JL_1.mp40.jpg    QE_1.mp4120.jpg
EM_1.mp40.jpg    JL_1.mp4120.jpg  QE_1.mp4240.jpg
```

```
# List the image frames in Google Drive
!ls GDrive/My\ Drive/MyVideo/Frames/Real
```

```
BO_1.mp40.jpg    EM_1.mp4120.jpg  JL_1.mp4240.jpg  TC_1.mp40.jpg
BO_1.mp4120.jpg  EM_1.mp4240.jpg  QE_1.mp40.jpg    TC_1.mp4120.jpg
BO_1.mp4240.jpg  JL_1.mp40.jpg    QE_1.mp4120.jpg
EM_1.mp40.jpg    JL_1.mp4120.jpg  QE_1.mp4240.jpg
```

Fake videos

```
# Folder path contains the videos
INPUT_VIDEOS_PATH = '/content/DetectionTest/Sources/Fake'

# Folder path for storing image frames to be extracted from videos
OUTPUT_FRAMES_PATH = '/content/DetectionTest/Sources/frames/fake'

# Create variables
one_frame_each = 120                                                         # Para = 24, Extract 1 frame from a video for every 24 frame

!if [ -d {OUTPUT_FRAMES_PATH} ]; then echo "Output to be stored in "{OUTPUT_FRAMES_PATH} ; else mkdir {OUTPUT_FRAMES_PATH} && echo "Output d

videofiles = !ls {INPUT_VIDEOS_PATH}/*.mp4                                   # Video file names in INPUT VIDEOS PATH
```

```
Output to be stored in /content/DetectionTest/Sources/frames/fake
```

```
for videofile in videofiles:
    count = 0
    success = True
    filename = os.path.basename(videofile)
    vidcap = cv2.VideoCapture(videofile)

    while success:
        if (count%one_frame_each == 0):                                          # checks frame number and keeps one_frame_each
            success, image = vidcap.read()                                        # reads next frame
            if image.shape[1]>640:                                               # if image width > 640, resize it
                tmp = resize(image, (math.floor(640 / image.shape[1] * image.shape[0]), 640))
            plt.imsave("%s/%s%d.jpg" % (OUTPUT_FRAMES_PATH,filename,count), tmp, format='jpg') # saves images to frame folder
            print ('*', end="")
        else:
            success,image = vidcap.read()                                        # reads next frame
        count += 1                                                               # loops counter
```

```
*************
```

```
# List the image frames in Google Colab's temporary directory
!ls /content/DetectionTest/Sources/frames/fake
```

```
# Copy the frames to Google Drive
!cp /content/DetectionTest/Sources/frames/fake/* GDrive/My\ Drive/MyVideo/Frames/Fake
```

```
BO_0.mp40.jpg    JL_0.mp40.jpg    QE_0.mp4120.jpg  TC_0.mp4240.jpg
BO_0.mp4120.jpg  JL_0.mp4120.jpg  QE_0.mp4240.jpg
EM_0.mp40.jpg    JL_0.mp4240.jpg  TC_0.mp40.jpg
EM_0.mp4120.jpg  QE_0.mp40.jpg    TC_0.mp4120.jpg
```

```
# List the image frames in Google Drive
!ls GDrive/My\ Drive/MyVideo/Frames/Fake
```

```
BO_0.mp40.jpg    JL_0.mp40.jpg    QE_0.mp4120.jpg  TC_0.mp4240.jpg
BO_0.mp4120.jpg  JL_0.mp4120.jpg  QE_0.mp4240.jpg
EM_0.mp40.jpg    JL_0.mp4240.jpg  TC_0.mp40.jpg
EM_0.mp4120.jpg  QE_0.mp40.jpg    TC_0.mp4120.jpg
```

Step 5 - Extract faces from the frames

```
!apt-get install build-essential cmake
!apt-get install libopenblas-dev liblapack-dev
!pip install dlib
!pip install face_recognition
!pip install opencv-python
!pip install opencv-contrib-python
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
cmake is already the newest version (3.10.2-1ubuntu2.18.04.2).
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
Reading package lists... Done
Building dependency tree
Reading state information... Done
liblapack-dev is already the newest version (3.7.1-4ubuntu1).
libopenblas-dev is already the newest version (0.2.20+ds-4).
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
Requirement already satisfied: dlib in /usr/local/lib/python3.7/dist-packages (19.18.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
Collecting face_recognition
  Downloading face_recognition-1.3.0-py2.py3-none-any.whl (15 kB)
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (from face_recognition) (7.1.2)
Collecting face-recognition-models>=0.3.0
  Downloading face_recognition_models-0.3.0.tar.gz (100.1 MB)
     |████████████████████████████████| 100.1 MB 1.2 MB/s
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from face_recognition) (1.19.5)
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.7/dist-packages (from face_recognition) (7.1.2)
Requirement already satisfied: dlib>=19.7 in /usr/local/lib/python3.7/dist-packages (from face_recognition) (19.18.0)
Building wheels for collected packages: face-recognition-models
  Building wheel for face-recognition-models (setup.py) ... done
  Created wheel for face-recognition-models: filename=face_recognition_models-0.3.0-py2.py3-none-any.whl size=100566185 sha256=d3af9e7
  Stored in directory: /root/.cache/pip/wheels/d6/81/3c/884bcd5e1c120ff548d57c2ecc9ebf3281c9a6f7c0e7e7947a
Successfully built face-recognition-models
Installing collected packages: face-recognition-models, face-recognition
Successfully installed face-recognition-1.3.0 face-recognition-models-0.3.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.19.5)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
Requirement already satisfied: opencv-contrib-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-contrib-python) (1.19.5)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It
```

```
# Import OpenCV and face_recognition libraries to extract the faces from frames
from PIL import Image
import cv2
import face_recognition
```

Real video images

```
# Folder path contains the frames
INPUT_FRAMES_PATH = '/content/DetectionTest/Sources/frames/real'

# Folder path for storing faces to be extracted from images
OUTPUT_FACES_PATH = '/content/DetectionTest/Sources/faces/real'

!if [ -d {OUTPUT_FACES_PATH} ]; then echo "Output to be stored in "{OUTPUT_FACES_PATH} ; else mkdir {OUTPUT_FACES_PATH} && echo "Output dire

imagefiles = !ls {INPUT_FRAMES_PATH}/*.jpg                                    # Image file names in INPUT FRAMES PATH
```

```
# Extract face from all image files in the given directory
for imagefile in imagefiles:
    filename = os.path.basename(imagefile)
    image = face_recognition.load_image_file(imagefile)
    face_locations = face_recognition.face_locations(image)
    print("Found {} face(s) in this photograph - %s.".format(len(face_locations)),filename)

    %matplotlib inline
    plt.rcParams['figure.figsize'] = [32, 8]

    for face_location in face_locations:
        top, right, bottom, left = face_location
        print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right: {} for %s".format(top, left, bottom, right), filena
        face_image = image[top:bottom, left:right]

        if(len(face_locations) == 1):
            cv2.imwrite("%s/%sF.jpg" % (OUTPUT_FACES_PATH,filename), face_image)
            print ('*', end="")
```

```
 Output to be stored in /content/DetectionTest/Sources/faces/real
 Found 1 face(s) in this photograph - %s. BO_1.mp40.jpg
 A face is located at pixel location Top: 66, Left: 275, Bottom: 156, Right: 364 for %s BO_1.mp40.jpg
 *Found 1 face(s) in this photograph - %s. BO_1.mp4120.jpg
 A face is located at pixel location Top: 80, Left: 295, Bottom: 155, Right: 370 for %s BO_1.mp4120.jpg
 *Found 1 face(s) in this photograph - %s. BO_1.mp4240.jpg
 A face is located at pixel location Top: 66, Left: 285, Bottom: 156, Right: 374 for %s BO_1.mp4240.jpg
 *Found 1 face(s) in this photograph - %s. EM_1.mp40.jpg
 A face is located at pixel location Top: 82, Left: 253, Bottom: 211, Right: 382 for %s EM_1.mp40.jpg
 *Found 1 face(s) in this photograph - %s. EM_1.mp4120.jpg
 A face is located at pixel location Top: 82, Left: 239, Bottom: 211, Right: 368 for %s EM_1.mp4120.jpg
 *Found 1 face(s) in this photograph - %s. EM_1.mp4240.jpg
 A face is located at pixel location Top: 64, Left: 253, Bottom: 219, Right: 408 for %s EM_1.mp4240.jpg
 *Found 1 face(s) in this photograph - %s. JL_1.mp40.jpg
 A face is located at pixel location Top: 80, Left: 306, Bottom: 187, Right: 414 for %s JL_1.mp40.jpg
 *Found 1 face(s) in this photograph - %s. JL_1.mp4120.jpg
 A face is located at pixel location Top: 53, Left: 253, Bottom: 182, Right: 382 for %s JL_1.mp4120.jpg
 *Found 1 face(s) in this photograph - %s. JL_1.mp4240.jpg
 A face is located at pixel location Top: 68, Left: 235, Bottom: 175, Right: 342 for %s JL_1.mp4240.jpg
 *Found 1 face(s) in this photograph - %s. QE_1.mp40.jpg
 A face is located at pixel location Top: 92, Left: 366, Bottom: 199, Right: 474 for %s QE_1.mp40.jpg
 *Found 1 face(s) in this photograph - %s. QE_1.mp4120.jpg
 A face is located at pixel location Top: 104, Left: 354, Bottom: 211, Right: 462 for %s QE_1.mp4120.jpg
 *Found 1 face(s) in this photograph - %s. QE_1.mp4240.jpg
 A face is located at pixel location Top: 96, Left: 354, Bottom: 225, Right: 483 for %s QE_1.mp4240.jpg
 *Found 1 face(s) in this photograph - %s. TC_1.mp40.jpg
 A face is located at pixel location Top: 80, Left: 163, Bottom: 187, Right: 270 for %s TC_1.mp40.jpg
 *Found 1 face(s) in this photograph - %s. TC_1.mp4120.jpg
 A face is located at pixel location Top: 68, Left: 211, Bottom: 175, Right: 318 for %s TC_1.mp4120.jpg
 *
```

```
# Count number of faces in Google Colab's temporary directory
!ls /content/DetectionTest/Sources/faces/real | wc -l
```

```
 14
```

```
# List the faces in Google Colab's temporary directory
!ls /content/DetectionTest/Sources/faces/real
```

```
 BO_1.mp40.jpgF.jpg    EM_1.mp4240.jpgF.jpg  QE_1.mp4120.jpgF.jpg
 BO_1.mp4120.jpgF.jpg  JL_1.mp40.jpgF.jpg    QE_1.mp4240.jpgF.jpg
 BO_1.mp4240.jpgF.jpg  JL_1.mp4120.jpgF.jpg  TC_1.mp40.jpgF.jpg
 EM_1.mp40.jpgF.jpg    JL_1.mp4240.jpgF.jpg  TC_1.mp4120.jpgF.jpg
 EM_1.mp4120.jpgF.jpg  QE_1.mp40.jpgF.jpg
```

```
# Copy the faces to Google Drive
!cp /content/DetectionTest/Sources/faces/real/* GDrive/My\ Drive/MyVideo/Faces/Real
```

```
# List the faces in Google Drive
!ls GDrive/My\ Drive/MyVideo/Faces/Real
```

```
 BO_1.mp40.jpgF.jpg    EM_1.mp4240.jpgF.jpg  QE_1.mp4120.jpgF.jpg
 BO_1.mp4120.jpgF.jpg  JL_1.mp40.jpgF.jpg    QE_1.mp4240.jpgF.jpg
 BO_1.mp4240.jpgF.jpg  JL_1.mp4120.jpgF.jpg  TC_1.mp40.jpgF.jpg
 EM_1.mp40.jpgF.jpg    JL_1.mp4240.jpgF.jpg  TC_1.mp4120.jpgF.jpg
 EM_1.mp4120.jpgF.jpg  QE_1.mp40.jpgF.jpg
```

```
# Visualise the face
plt.figure(figsize = (5,2))
viewreal = plt.imread("/content/DetectionTest/Sources/faces/real/BO_1.mp40.jpgF.jpg")   # update file name
plt.imshow(viewreal)
```

```
<matplotlib.image.AxesImage at 0x7f17eda8fe90>
```

Fake video images

```
# Folder path contains the frames
INPUT_FRAMES_PATH = '/content/DetectionTest/Sources/frames/fake'

# Folder path for storing faces to be extracted from images
OUTPUT_FACES_PATH = '/content/DetectionTest/Sources/faces/fake'

!if [ -d {OUTPUT_FACES_PATH} ]; then echo "Output to be stored in "{OUTPUT_FACES_PATH} ; else mkdir {OUTPUT_FACES_PATH} && echo "Output dire

imagefiles = !ls {INPUT_FRAMES_PATH}/*.jpg                                        # Image file names in INPUT FRAMES PATH

# Extract face from all image files in the given directory
for imagefile in imagefiles:
    filename = os.path.basename(imagefile)
    image = face_recognition.load_image_file(imagefile)
    face_locations = face_recognition.face_locations(image)
    print("Found {} face(s) in this photograph - %s.".format(len(face_locations)),filename)

    %matplotlib inline
    plt.rcParams['figure.figsize'] = [32, 8]

    for face_location in face_locations:
        top, right, bottom, left = face_location
        print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right: {} for %s".format(top, left, bottom, right), filena
        face_image = image[top:bottom, left:right]

        if(len(face_locations) == 1):
            cv2.imwrite("%s/%sF.jpg" % (OUTPUT_FACES_PATH,filename), face_image)
            print ('*', end="")
```

```
Output to be stored in /content/DetectionTest/Sources/faces/fake
Found 1 face(s) in this photograph - %s. BO_0.mp40.jpg
A face is located at pixel location Top: 80, Left: 247, Bottom: 187, Right: 354 for %s BO_0.mp40.jpg
*Found 1 face(s) in this photograph - %s. BO_0.mp4120.jpg
A face is located at pixel location Top: 76, Left: 255, Bottom: 166, Right: 344 for %s BO_0.mp4120.jpg
*Found 1 face(s) in this photograph - %s. EM_0.mp40.jpg
A face is located at pixel location Top: 92, Left: 235, Bottom: 199, Right: 342 for %s EM_0.mp40.jpg
*Found 1 face(s) in this photograph - %s. EM_0.mp4120.jpg
A face is located at pixel location Top: 96, Left: 253, Bottom: 225, Right: 382 for %s EM_0.mp4120.jpg
*Found 1 face(s) in this photograph - %s. JL_0.mp40.jpg
A face is located at pixel location Top: 66, Left: 325, Bottom: 156, Right: 414 for %s JL_0.mp40.jpg
*Found 1 face(s) in this photograph - %s. JL_0.mp4120.jpg
A face is located at pixel location Top: 76, Left: 305, Bottom: 166, Right: 394 for %s JL_0.mp4120.jpg
*Found 1 face(s) in this photograph - %s. JL_0.mp4240.jpg
A face is located at pixel location Top: 88, Left: 295, Bottom: 163, Right: 370 for %s JL_0.mp4240.jpg
*Found 0 face(s) in this photograph - %s. QE_0.mp40.jpg
Found 1 face(s) in this photograph - %s. QE_0.mp4120.jpg
A face is located at pixel location Top: 107, Left: 291, Bottom: 159, Right: 343 for %s QE_0.mp4120.jpg
*Found 1 face(s) in this photograph - %s. QE_0.mp4240.jpg
A face is located at pixel location Top: 101, Left: 287, Bottom: 163, Right: 349 for %s QE_0.mp4240.jpg
*Found 1 face(s) in this photograph - %s. TC_0.mp40.jpg
A face is located at pixel location Top: 139, Left: 282, Bottom: 268, Right: 411 for %s TC_0.mp40.jpg
*Found 1 face(s) in this photograph - %s. TC_0.mp4120.jpg
A face is located at pixel location Top: 140, Left: 282, Bottom: 247, Right: 390 for %s TC_0.mp4120.jpg
*Found 1 face(s) in this photograph - %s. TC_0.mp4240.jpg
A face is located at pixel location Top: 86, Left: 275, Bottom: 176, Right: 364 for %s TC_0.mp4240.jpg
*
```

```
# Count number of faces in Google Colab's temporary directory
!ls /content/DetectionTest/Sources/faces/fake | wc -l
```

```
12
```

```
# List the faces in Google Colab's temporary directory
!ls /content/DetectionTest/Sources/faces/fake
```

```
BO_0.mp40.jpgF.jpg    JL_0.mp40.jpgF.jpg    QE_0.mp4240.jpgF.jpg
BO_0.mp4120.jpgF.jpg  JL_0.mp4120.jpgF.jpg  TC_0.mp40.jpgF.jpg
EM_0.mp40.jpgF.jpg    JL_0.mp4240.jpgF.jpg  TC_0.mp4120.jpgF.jpg
EM_0.mp4120.jpgF.jpg  QE_0.mp4120.jpgF.jpg  TC_0.mp4240.jpgF.jpg
```

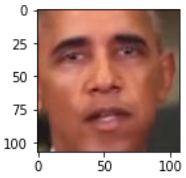
```
# Copy the faces to Google Drive
!cp /content/DetectionTest/Sources/faces/fake/* GDrive/My\ Drive/MyVideo/Faces/Fake
```

```
# List the faces in Google Drive
!ls GDrive/My\ Drive/MyVideo/Faces/Fake
```

```
BO_0.mp40.jpgF.jpg    JL_0.mp40.jpgF.jpg    QE_0.mp4240.jpgF.jpg
BO_0.mp4120.jpgF.jpg  JL_0.mp4120.jpgF.jpg  TC_0.mp40.jpgF.jpg
EM_0.mp40.jpgF.jpg    JL_0.mp4240.jpgF.jpg  TC_0.mp4120.jpgF.jpg
EM_0.mp4120.jpgF.jpg  QE_0.mp4120.jpgF.jpg  TC_0.mp4240.jpgF.jpg
```

```
# Visualise the face
plt.figure(figsize = (5,2))
viewreal = plt.imread("/content/DetectionTest/Sources/faces/fake/BO_0.mp40.jpgF.jpg")   # update file name
plt.imshow(viewreal)
```

```
<matplotlib.image.AxesImage at 0x7f17eda35fd0>
```



Step 6 - Create my test dataset

```
import numpy as np
import tensorflow
from tensorflow import keras
```

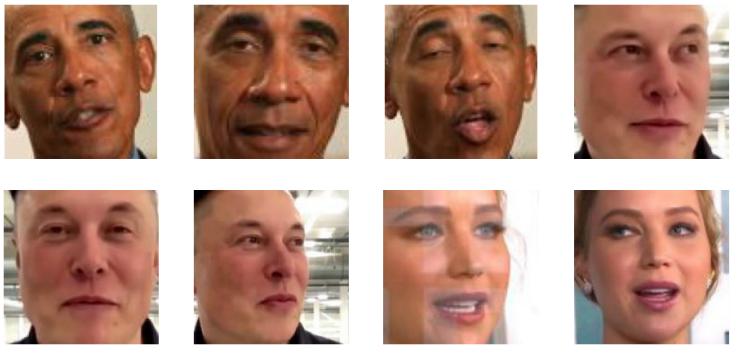
```
# Set path
base_dir = '/content/DetectionTest/Sources/'
test_dir = os.path.join(base_dir, 'faces')

# Directory with our test fake pictures
test_fake_dir = os.path.join(test_dir, 'fake')

# Directory with our test real pictures
test_real_dir = os.path.join(test_dir, 'real')

# Print some file names to confirm the right directories are connected
test_real_fnames = os.listdir(test_real_dir)
test_real_fnames.sort()
print(test_real_fnames[:5])

test_fake_fnames = os.listdir(test_fake_dir)
test_fake_fnames.sort()
print(test_fake_fnames[:5])

# Count the number of real and fake images of faces in the train and validation directories
print('total real video images for testing:', len(os.listdir(test_real_dir)))
print('total fake video images for testing:', len(os.listdir(test_fake_dir)))
```

```
['BO_1.mp40.jpgF.jpg', 'BO_1.mp4120.jpgF.jpg', 'BO_1.mp4240.jpgF.jpg', 'EM_1.mp40.jpgF.jpg', 'EM_1.mp4120.jpgF.jpg']
['BO_0.mp40.jpgF.jpg', 'BO_0.mp4120.jpgF.jpg', 'EM_0.mp40.jpgF.jpg', 'EM_0.mp4120.jpgF.jpg', 'JL_0.mp40.jpgF.jpg']
total real video images for testing: 14
total fake video images for testing: 12
```

```
# Define matplotlib parameters for output images as 4x4 images
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

nrows = 4
ncols = 4

pic_index = 0                               # index for image iteration
```

```
# Show a batch of 8 real images
# Rerun the cell to fetch a new batch of images

fig = plt.gcf()                             # set up matplotlib fig
fig.set_size_inches(ncols * 4, nrows * 4)   # size matplotlib fig to fit into a 4x4 image

pic_index += 8
next_real_pix = [os.path.join(test_real_dir, fname)
                 for fname in test_real_fnames[pic_index-8:pic_index]]

for i, img_path in enumerate(next_real_pix):
  sp = plt.subplot(nrows, ncols, i + 1)     # Set up subplot, subplot indices start at 1
  sp.axis('Off')                            # Turn off axis
```

```
  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```



```
pic_index = 0                               # reset index for image iteration for fake images
```

```
# Show a batch of 8 fake images
# Rerun the cell to fetch a new batch of images

fig = plt.gcf()                                 # set up matplotlib fig
fig.set_size_inches(ncols * 4, nrows * 4)     # size matplotlib fig to fit into a 4x4 image

pic_index += 8
next_fake_pix = [os.path.join(test_fake_dir, fname)
                 for fname in test_fake_fnames[pic_index-8:pic_index]]

for i, img_path in enumerate(next_fake_pix):
  sp = plt.subplot(nrows, ncols, i + 1)       # Set up subplot, subplot indices start at 1
  sp.axis('Off')                                # Turn off axis

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```

```
BATCH_SIZE = 40
IMG_SIZE = (150, 150)

# Create test dataset with dataset from directory method
test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)
```

```
Found 26 files belonging to 2 classes.
```

```
# Check the result of test dataset
class_names = test_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in test_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```


```
# Performance enhancement
AUTOTUNE = tensorflow.data.AUTOTUNE

test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Step 7 - Load the Model

```
# List the models in Google Colab's temporary directory
!ls -la /content/DetectionTest/Model
```

```
total 2013040
drwxr-xr-x 2 root root      4096 Dec  9 09:24 .
drwxr-xr-x 4 root root      4096 Dec  9 09:24 ..
-rw------- 1 root root 144739920 Dec  9 09:24 my_DenseNet.h5
-rw------- 1 root root  86592688 Dec  9 09:24 my_EfficientNetB3.h5
-rw------- 1 root root 513642616 Dec  9 09:24 my_EfficientNetB7.h5
-rw------- 1 root root 166580856 Dec  9 09:24 my_InceptionNetV3.h5
-rw------- 1 root root 433641880 Dec  9 09:24 my_InceptionResNetModel.h5
-rw------- 1 root root  16919968 Dec  9 09:24 my_MobileNetModel.h5
-rw------- 1 root root  36567648 Dec  9 09:24 my_NASNetMobile.h5
-rw------- 1 root root 460713944 Dec  9 09:24 my_ResNet152V2.h5
-rw------- 1 root root  80178680 Dec  9 09:24 my_VGG19.h5
-rw------- 1 root root 121727000 Dec  9 09:24 my_Xception.h5
```

```
# Load models
modelM1 = keras.models.load_model('/content/DetectionTest/Model/my_Xception.h5')
modelM2 = keras.models.load_model('/content/DetectionTest/Model/my_VGG19.h5')
modelM3 = keras.models.load_model('/content/DetectionTest/Model/my_ResNet152V2.h5')
modelM4 = keras.models.load_model('/content/DetectionTest/Model/my_InceptionNetV3.h5')
modelM5 = keras.models.load_model('/content/DetectionTest/Model/my_InceptionResNetModel.h5')
modelM6 = keras.models.load_model('/content/DetectionTest/Model/my_MobileNetModel.h5')
modelM7 = keras.models.load_model('/content/DetectionTest/Model/my_DenseNet.h5')
modelM8 = keras.models.load_model('/content/DetectionTest/Model/my_NASNetMobile.h5')
modelM9 = keras.models.load_model('/content/DetectionTest/Model/my_EfficientNetB3.h5')
modelM10 = keras.models.load_model('/content/DetectionTest/Model/my_EfficientNetB7.h5')
```

```
modelM1.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 xception (Functional)       (None, 5, 5, 2048)        20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 20,863,529
Trainable params: 9,480,393
Non-trainable params: 11,383,136
_________________________________________________________________
```

```
modelM2.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.__operators__.getitem (S  (None, 150, 150, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 150, 150, 3)      0

 vgg19 (Functional)          (None, 4, 4, 512)         20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 1)                 513

=================================================================
Total params: 20,024,897
Trainable params: 513
Non-trainable params: 20,024,384
_________________________________________________________________
```

```
modelM3.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 resnet152v2 (Functional)    (None, 5, 5, 2048)        58331648

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 58,333,697
```

```
Trainable params: 56,396,545
Non-trainable params: 1,937,152
_________________________________________________________________
```

```
modelM4.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 inception_v3 (Functional)   (None, 3, 3, 2048)        21802784

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 21,804,833
Trainable params: 19,628,417
Non-trainable params: 2,176,416
_________________________________________________________________
```

```
modelM5.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 150, 150, 3)      0
 a)

 inception_resnet_v2 (Functi  (None, 3, 3, 1536)       54336736
 onal)

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1536)              0

 dense (Dense)               (None, 1)                 1537

=================================================================
Total params: 54,338,273
Trainable params: 53,510,161
Non-trainable params: 828,112
_________________________________________________________________
```

```
modelM6.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv_1 (TFOpLamb  (None, 160, 160, 3)      0
 da)

 tf.math.subtract_1 (TFOpLam  (None, 160, 160, 3)      0
 bda)

 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)
```

```
 dropout_1 (Dropout)         (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,862,721
Non-trainable params: 396,544
_________________________________________________________________
```

```
modelM7.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 tf.math.truediv (TFOpLambda  (None, 150, 150, 3)      0
 )

 tf.nn.bias_add (TFOpLambda)  (None, 150, 150, 3)      0

 tf.math.truediv_1 (TFOpLamb  (None, 150, 150, 3)      0
 da)

 densenet201 (Functional)    (None, 4, 4, 1920)        18321984

 global_average_pooling2d (G  (None, 1920)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1920)              0

 dense (Dense)               (None, 1)                 1921

=================================================================
Total params: 18,323,905
Trainable params: 17,290,177
Non-trainable params: 1,033,728
_________________________________________________________________
```

```
modelM8.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 sequential_2 (Sequential)   (None, 224, 224, 3)       0

 tf.math.truediv (TFOpLambda  (None, 224, 224, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 224, 224, 3)      0
 a)

 NASNet (Functional)         (None, 7, 7, 1056)        4269716

 global_average_pooling2d (G  (None, 1056)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1056)              0

 dense (Dense)               (None, 1)                 1057

=================================================================
Total params: 4,270,773
Trainable params: 4,210,713
Non-trainable params: 60,060
_________________________________________________________________
```

```
modelM9.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 efficientnetb3 (Functional)  (None, 5, 5, 1536)       10783535

 global_average_pooling2d (G  (None, 1536)             0
 lobalAveragePooling2D)
```

```
 dropout (Dropout)           (None, 1536)              0

 dense (Dense)               (None, 1)                 1537

=================================================================
Total params: 10,785,072
Trainable params: 10,577,383
Non-trainable params: 207,689
_________________________________________________________________
```

```
modelM10.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 sequential (Sequential)     (None, 150, 150, 3)       0

 efficientnetb7 (Functional)  (None, 5, 5, 2560)       64097687

 global_average_pooling2d (G  (None, 2560)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2560)              0

 dense (Dense)               (None, 1)                 2561

=================================================================
Total params: 64,100,248
Trainable params: 63,663,721
Non-trainable params: 436,527
_________________________________________________________________
```

Step 8 - Make prediction with unseen test dataset from the wild

```
loss, accuracy = modelM1.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
1/1 [==============================] - 8s 8s/step - loss: 0.8578 - accuracy: 0.6538
Test accuracy : 0.6538461446762085
```

```
loss, accuracy = modelM2.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
1/1 [==============================] - 1s 1s/step - loss: 0.6524 - accuracy: 0.6538
Test accuracy : 0.6538461446762085
```

```
loss, accuracy = modelM3.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
1/1 [==============================] - 2s 2s/step - loss: 2.8144 - accuracy: 0.6538
Test accuracy : 0.6538461446762085
```

```
loss, accuracy = modelM4.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
1/1 [==============================] - 1s 1s/step - loss: 2.9405 - accuracy: 0.6923
Test accuracy : 0.692307710647583
```

```
loss, accuracy = modelM5.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_test_function.<locals>.test_function at 0x7f1691179ef0> triggered
1/1 [==============================] - 3s 3s/step - loss: 3.1116 - accuracy: 0.6538
Test accuracy : 0.6538461446762085
```

```
IMG_SIZE = (160, 160)


# Create test dataset with dataset from directory method
test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)

# Performance enhancement
AUTOTUNE = tensorflow.data.AUTOTUNE
```

```
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)


loss, accuracy = modelM6.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
Found 26 files belonging to 2 classes.
1/1 [==============================] - 1s 832ms/step - loss: 0.9838 - accuracy: 0.7308
Test accuracy : 0.7307692170143127
```

```
loss, accuracy = modelM7.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_test_function.<locals>.test_function at 0x7f169028a440> triggered
1/1 [==============================] - 3s 3s/step - loss: 2.7244 - accuracy: 0.6538
Test accuracy : 0.6538461446762085
```

```
IMG_SIZE = (224, 224)

# Create test dataset with dataset from directory method
test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)

# Performance enhancement
AUTOTUNE = tensorflow.data.AUTOTUNE

test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)


loss, accuracy = modelM8.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
Found 26 files belonging to 2 classes.
1/1 [==============================] - 3s 3s/step - loss: 0.8920 - accuracy: 0.7308
Test accuracy : 0.7307692170143127
```

```
loss, accuracy = modelM9.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
1/1 [==============================] - 2s 2s/step - loss: 1.7097 - accuracy: 0.6923
Test accuracy : 0.692307710647583
```

```
loss, accuracy = modelM10.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
1/1 [==============================] - 5s 5s/step - loss: 1.5947 - accuracy: 0.6154
Test accuracy : 0.6153846383094788
```

```
# Demonstration for model M3 ResNet152V2

# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tensorflow.nn.sigmoid(predictions)
predictions = tensorflow.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(26):
  ax = plt.subplot(8, 5, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

Step 9 - Image level testing

Image size setting for models

Setting to be updated before testing model M6 and M8.

```
# For all models, except for model M6 and M8
IMG_SIZE = (150, 150)

# Create test dataset with dataset from directory method
test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)

# Performance enhancement
AUTOTUNE = tensorflow.data.AUTOTUNE

test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
Found 26 files belonging to 2 classes.
```

```
# For model M6
IMG_SIZE = (160, 160)

# Create test dataset with dataset from directory method
test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)

# Performance enhancement
AUTOTUNE = tensorflow.data.AUTOTUNE

test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
Found 26 files belonging to 2 classes.
```

```
# For model M8
IMG_SIZE = (224, 224)

# Create test dataset with dataset from directory method
test_dataset = tensorflow.keras.utils.image_dataset_from_directory(test_dir,
                                                                   shuffle=False,
                                                                   batch_size=BATCH_SIZE,
                                                                   image_size=IMG_SIZE)

# Performance enhancement
AUTOTUNE = tensorflow.data.AUTOTUNE

test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
Found 26 files belonging to 2 classes.
```

Testing

Model M1

```
# Upload the real faces to make prediction
```

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM1.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
Saving TC_1.mp40.jpgF.jpg to TC_1.mp40.jpgF.jpg
Saving TC_1.mp4120.jpgF.jpg to TC_1.mp4120.jpgF.jpg
BO_1.mp40.jpgF.jpg is real
BO_1.mp4120.jpgF.jpg is deepfake
BO_1.mp4240.jpgF.jpg is real
EM_1.mp40.jpgF.jpg is real
EM_1.mp4120.jpgF.jpg is real
EM_1.mp4240.jpgF.jpg is real
JL_1.mp40.jpgF.jpg is real
JL_1.mp4120.jpgF.jpg is real
JL_1.mp4240.jpgF.jpg is real
QE_1.mp40.jpgF.jpg is real
QE_1.mp4120.jpgF.jpg is real
QE_1.mp4240.jpgF.jpg is real
TC_1.mp40.jpgF.jpg is deepfake
```

```
# Upload the fake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM1.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
BO_0.mp40.jpgF.jpg is real
BO_0.mp4120.jpgF.jpg is deepfake
EM_0.mp40.jpgF.jpg is deepfake
EM_0.mp4120.jpgF.jpg is deepfake
JL_0.mp40.jpgF.jpg is deepfake
JL_0.mp4120.jpgF.jpg is real
JL_0.mp4240.jpgF.jpg is real
QE_0.mp4120.jpgF.jpg is deepfake
QE_0.mp4240.jpgF.jpg is deepfake
TC_0.mp40.jpgF.jpg is real
TC_0.mp4120.jpgF.jpg is real
```

```
# Figures for performance calculation
TP = true_pos = 12 # real faces predicted as real
TN = true_neg = 6 # fake faces predicted as fake
FP = false_pos = 6 # fake faces predicted as real
FN = false_neg = 2 # real faces predicted as fake
M1results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M1results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {M1results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M1results[metric] = TP / (TP + FN)
print(f"{metric} is {M1results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M1results[metric] = TN / (TN + FP)
print(f"{metric} is {M1results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
M1results[metric] = TP / (TP + FP)
print(f"{metric} is {M1results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
M1results[metric] = TN / (TN + FN)
print(f"{metric} is {M1results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M1results[metric] = 2 / (1 / M1results["PPV"] + 1 / M1results["TPR"])
print(f"{metric} is {M1results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
```

```
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
M1results[metric] = num / den
print(f"{metric} is {M1results[metric]: .3f}")
```

```
ACC is  0.692
TPR is  0.857
TNR is  0.500
PPV is  0.667
NPV is  0.750
F1 is  0.750
MCC is  0.386
```

Model M2

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM2.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
Saving TC_1.mp40.jpgF.jpg to TC_1.mp40.jpgF.jpg
Saving TC_1.mp4120.jpgF.jpg to TC_1.mp4120.jpgF.jpg
BO_1.mp40.jpgF.jpg is deepfake
BO_1.mp4120.jpgF.jpg is deepfake
BO_1.mp4240.jpgF.jpg is deepfake
EM_1.mp40.jpgF.jpg is real
EM_1.mp4120.jpgF.jpg is real
EM_1.mp4240.jpgF.jpg is real
JL_1.mp40.jpgF.jpg is real
JL_1.mp4120.jpgF.jpg is real
JL_1.mp4240.jpgF.jpg is real
QE_1.mp40.jpgF.jpg is real
QE_1.mp4120.jpgF.jpg is real
QE_1.mp4240.jpgF.jpg is deepfake
TC_1.mp40.jpgF.jpg is deepfake
TC_1.mp4120.jpgF.jpg is real
```

```
# Upload the fake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image
```

```
uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM2.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
BO_0.mp40.jpgF.jpg is deepfake
BO_0.mp4120.jpgF.jpg is real
EM_0.mp40.jpgF.jpg is deepfake
EM_0.mp4120.jpgF.jpg is real
JL_0.mp40.jpgF.jpg is real
JL_0.mp4120.jpgF.jpg is real
JL_0.mp4240.jpgF.jpg is real
QE_0.mp4120.jpgF.jpg is deepfake
QE_0.mp4240.jpgF.jpg is deepfake
TC_0.mp40.jpgF.jpg is real
TC_0.mp4120.jpgF.jpg is deepfake
TC_0.mp4240.jpgF.jpg is real
```

```
# Figures for performance calculation
TP = true_pos = 9 # real faces predicted as real
TN = true_neg = 5 # fake faces predicted as fake
FP = false_pos = 7 # fake faces predicted as real
FN = false_neg = 5 # real faces predicted as fake
M2results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M2results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {M2results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M2results[metric] = TP / (TP + FN)
print(f"{metric} is {M2results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M2results[metric] = TN / (TN + FP)
print(f"{metric} is {M2results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
M2results[metric] = TP / (TP + FP)
```

```
print(f"{metric} is {M2results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
M2results[metric] = TN / (TN + FN)
print(f"{metric} is {M2results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M2results[metric] = 2 / (1 / M2results["PPV"] + 1 / M2results["TPR"])
print(f"{metric} is {M2results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
M2results[metric] = num / den
print(f"{metric} is {M2results[metric]: .3f}")
```

```
ACC is  0.538
TPR is  0.643
TNR is  0.417
PPV is  0.562
NPV is  0.500
F1 is  0.600
MCC is  0.061
```

Model M3

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM3.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
```

```
# Upload the fake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM3.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
BO_0.mp40.jpgF.jpg is real
BO_0.mp4120.jpgF.jpg is deepfake
EM_0.mp40.jpgF.jpg is real
EM_0.mp4120.jpgF.jpg is deepfake
JL_0.mp40.jpgF.jpg is real
JL_0.mp4120.jpgF.jpg is real
JL_0.mp4240.jpgF.jpg is real
QE_0.mp4120.jpgF.jpg is real
QE_0.mp4240.jpgF.jpg is real
TC_0.mp40.jpgF.jpg is real
TC_0.mp4120.jpgF.jpg is real
TC_0.mp4240.jpgF.jpg is real
```

```
# Figures for performance calculation
TP = true_pos = 14 # real faces predicted as real
TN = true_neg = 2 # fake faces predicted as fake
FP = false_pos = 10 # fake faces predicted as real
FN = false_neg = 0 # real faces predicted as fake
M3results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M3results[metric] = (TP + TN) / (TP + TN + FP + FN)
```

```
print(f"{metric} is {M3results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M3results[metric] = TP / (TP + FN)
print(f"{metric} is {M3results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M3results[metric] = TN / (TN + FP)
print(f"{metric} is {M3results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
M3results[metric] = TP / (TP + FP)
print(f"{metric} is {M3results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
M3results[metric] = TN / (TN + FN)
print(f"{metric} is {M3results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M3results[metric] = 2 / (1 / M3results["PPV"] + 1 / M3results["TPR"])
print(f"{metric} is {M3results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
M3results[metric] = num / den
print(f"{metric} is {M3results[metric]: .3f}")
```

```
ACC is  0.615
TPR is  1.000
TNR is  0.167
PPV is  0.583
NPV is  1.000
F1 is  0.737
MCC is  0.312
```

Model M4

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM4.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
Saving TC_1.mp40.jpgF.jpg to TC_1.mp40.jpgF.jpg
Saving TC_1.mp4120.jpgF.jpg to TC_1.mp4120.jpgF.jpg
BO_1.mp40.jpgF.jpg is real
BO_1.mp4120.jpgF.jpg is deepfake
BO_1.mp4240.jpgF.jpg is real
EM_1.mp40.jpgF.jpg is real
EM_1.mp4120.jpgF.jpg is real
EM_1.mp4240.jpgF.jpg is real
JL_1.mp40.jpgF.jpg is real
JL_1.mp4120.jpgF.jpg is real
JL_1.mp4240.jpgF.jpg is real
QE_1.mp40.jpgF.jpg is real
QE_1.mp4120.jpgF.jpg is real
QE_1.mp4240.jpgF.jpg is real
TC_1.mp40.jpgF.jpg is real
TC_1.mp4120.jpgF.jpg is real
```

```
# Upload the fake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM4.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
BO_0.mp40.jpgF.jpg is real
BO_0.mp4120.jpgF.jpg is real
EM_0.mp40.jpgF.jpg is deepfake
EM_0.mp4120.jpgF.jpg is real
JL_0.mp40.jpgF.jpg is real
JL_0.mp4120.jpgF.jpg is real
JL_0.mp4240.jpgF.jpg is deepfake
QE_0.mp4120.jpgF.jpg is deepfake
QE_0.mp4240.jpgF.jpg is deepfake
TC_0.mp40.jpgF.jpg is real
TC_0.mp4120.jpgF.jpg is real
TC_0.mp4240.jpgF.jpg is real
```

```
# Figures for performance calculation
TP = true_pos = 13 # real faces predicted as real
TN = true_neg = 4 # fake faces predicted as fake
FP = false_pos = 8 # fake faces predicted as real
FN = false_neg = 1 # real faces predicted as fake
M4results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M4results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {M4results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M4results[metric] = TP / (TP + FN)
print(f"{metric} is {M4results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M4results[metric] = TN / (TN + FP)
print(f"{metric} is {M4results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
M4results[metric] = TP / (TP + FP)
print(f"{metric} is {M4results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
M4results[metric] = TN / (TN + FN)
print(f"{metric} is {M4results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M4results[metric] = 2 / (1 / M4results["PPV"] + 1 / M4results["TPR"])
print(f"{metric} is {M4results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
M4results[metric] = num / den
print(f"{metric} is {M4results[metric]: .3f}")
```

```
ACC is  0.654
TPR is  0.929
TNR is  0.333
PPV is  0.619
NPV is  0.800
F1 is  0.743
MCC is  0.331
```

Model M5

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM5.predict(images, batch_size=1)
```

```
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
Saving TC_1.mp40.jpgF.jpg to TC_1.mp40.jpgF.jpg
Saving TC_1.mp4120.jpgF.jpg to TC_1.mp4120.jpgF.jpg
BO_1.mp40.jpgF.jpg is real
BO_1.mp4120.jpgF.jpg is deepfake
BO_1.mp4240.jpgF.jpg is deepfake
EM_1.mp40.jpgF.jpg is deepfake
EM_1.mp4120.jpgF.jpg is deepfake
EM_1.mp4240.jpgF.jpg is real
JL_1.mp40.jpgF.jpg is real
JL_1.mp4120.jpgF.jpg is real
JL_1.mp4240.jpgF.jpg is real
QE_1.mp40.jpgF.jpg is real
QE_1.mp4120.jpgF.jpg is real
QE_1.mp4240.jpgF.jpg is real
TC_1.mp40.jpgF.jpg is real
TC_1.mp4120.jpgF.jpg is real
```

```
# Upload the fake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM5.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
BO_0.mp40.jpgF.jpg is deepfake
BO_0.mp4120.jpgF.jpg is deepfake
EM_0.mp40.jpgF.jpg is deepfake
EM_0.mp4120.jpgF.jpg is deepfake
JL_0.mp40.jpgF.jpg is real
JL_0.mp4120.jpgF.jpg is real
JL_0.mp4240.jpgF.jpg is real
QE_0.mp4120.jpgF.jpg is deepfake
QE_0.mp4240.jpgF.jpg is deepfake
TC_0.mp40.jpgF.jpg is real
```

```
# Figures for performance calculation
TP = true_pos = 10 # real faces predicted as real
TN = true_neg = 6 # fake faces predicted as fake
FP = false_pos = 6 # fake faces predicted as real
FN = false_neg = 4 # real faces predicted as fake
M5results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M5results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {M5results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M5results[metric] = TP / (TP + FN)
print(f"{metric} is {M5results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M5results[metric] = TN / (TN + FP)
print(f"{metric} is {M5results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
M5results[metric] = TP / (TP + FP)
print(f"{metric} is {M5results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
M5results[metric] = TN / (TN + FN)
print(f"{metric} is {M5results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M5results[metric] = 2 / (1 / M5results["PPV"] + 1 / M5results["TPR"])
print(f"{metric} is {M5results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
M5results[metric] = num / den
print(f"{metric} is {M5results[metric]: .3f}")
```

```
ACC is  0.615
TPR is  0.714
TNR is  0.500
PPV is  0.625
NPV is  0.600
F1 is  0.667
MCC is  0.220
```

Model M6

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(160, 160))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM6.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
Saving TC_1.mp40.jpgF.jpg to TC_1.mp40.jpgF.jpg
Saving TC_1.mp4120.jpgF.jpg to TC_1.mp4120.jpgF.jpg
BO_1.mp40.jpgF.jpg is real
BO_1.mp4120.jpgF.jpg is deepfake
BO_1.mp4240.jpgF.jpg is deepfake
EM_1.mp40.jpgF.jpg is real
EM_1.mp4120.jpgF.jpg is real
EM_1.mp4240.jpgF.jpg is real
JL_1.mp40.jpgF.jpg is deepfake
JL_1.mp4120.jpgF.jpg is real
JL_1.mp4240.jpgF.jpg is real
QE_1.mp40.jpgF.jpg is deepfake
QE_1.mp4120.jpgF.jpg is deepfake
QE_1.mp4240.jpgF.jpg is real
TC_1.mp40.jpgF.jpg is real
TC_1.mp4120.jpgF.jpg is real
```

```
# Upload the fake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(160, 160))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM6.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
```

```
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
BO_0.mp40.jpgF.jpg is deepfake
BO_0.mp4120.jpgF.jpg is real
EM_0.mp40.jpgF.jpg is deepfake
EM_0.mp4120.jpgF.jpg is real
JL_0.mp40.jpgF.jpg is real
JL_0.mp4120.jpgF.jpg is deepfake
JL_0.mp4240.jpgF.jpg is deepfake
QE_0.mp4120.jpgF.jpg is deepfake
QE_0.mp4240.jpgF.jpg is deepfake
TC_0.mp40.jpgF.jpg is real
TC_0.mp4120.jpgF.jpg is deepfake
TC_0.mp4240.jpgF.jpg is deepfake
```

```
# Figures for performance calculation
TP = true_pos = 9 # real faces predicted as real
TN = true_neg = 8 # fake faces predicted as fake
FP = false_pos = 4 # fake faces predicted as real
FN = false_neg = 5 # real faces predicted as fake
M6results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M6results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {M6results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M6results[metric] = TP / (TP + FN)
print(f"{metric} is {M6results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M6results[metric] = TN / (TN + FP)
print(f"{metric} is {M6results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
M6results[metric] = TP / (TP + FP)
print(f"{metric} is {M6results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
M6results[metric] = TN / (TN + FN)
print(f"{metric} is {M6results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M6results[metric] = 2 / (1 / M6results["PPV"] + 1 / M6results["TPR"])
print(f"{metric} is {M6results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
```

```
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
M6results[metric] = num / den
print(f"{metric} is {M6results[metric]: .3f}")
```

```
ACC is  0.654
TPR is  0.643
TNR is  0.667
PPV is  0.692
NPV is  0.615
F1 is  0.667
MCC is  0.309
```

Model M7

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM7.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
Saving TC_1.mp40.jpgF.jpg to TC_1.mp40.jpgF.jpg
Saving TC_1.mp4120.jpgF.jpg to TC_1.mp4120.jpgF.jpg
BO_1.mp40.jpgF.jpg is real
BO_1.mp4120.jpgF.jpg is real
BO_1.mp4240.jpgF.jpg is real
EM_1.mp40.jpgF.jpg is real
EM_1.mp4120.jpgF.jpg is real
EM_1.mp4240.jpgF.jpg is real
JL_1.mp40.jpgF.jpg is real
JL_1.mp4120.jpgF.jpg is real
JL_1.mp4240.jpgF.jpg is real
QE_1.mp40.jpgF.jpg is real
QE_1.mp4120.jpgF.jpg is real
QE_1.mp4240.jpgF.jpg is real
TC_1.mp40.jpgF.jpg is real
TC_1.mp4120.jpgF.jpg is real
```

```
# Upload the fake faces to make prediction
import numpy as np
```

```
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM7.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
BO_0.mp40.jpgF.jpg is real
BO_0.mp4120.jpgF.jpg is real
EM_0.mp40.jpgF.jpg is real
EM_0.mp4120.jpgF.jpg is real
JL_0.mp40.jpgF.jpg is real
JL_0.mp4120.jpgF.jpg is real
JL_0.mp4240.jpgF.jpg is real
QE_0.mp4120.jpgF.jpg is real
QE_0.mp4240.jpgF.jpg is real
TC_0.mp40.jpgF.jpg is real
TC_0.mp4120.jpgF.jpg is real
TC_0.mp4240.jpgF.jpg is real
```

```
# Figures for performance calculation
TP = true_pos = 14 # real faces predicted as real
TN = true_neg = 0 # fake faces predicted as fake
FP = false_pos = 12 # fake faces predicted as real
FN = false_neg = 0 # real faces predicted as fake
M7results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M7results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {M7results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M7results[metric] = TP / (TP + FN)
print(f"{metric} is {M7results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M7results[metric] = TN / (TN + FP)
print(f"{metric} is {M7results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
```

```
metric = "PPV"
M7results[metric] = TP / (TP + FP)
print(f"{metric} is {M7results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
if (TN + FN) == 0:
    M7results[metric] = 0.000
else:
    M7results[metric] = TN / (TN + FN)
print(f"{metric} is {M7results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M7results[metric] = 2 / (1 / M7results["PPV"] + 1 / M7results["TPR"])
print(f"{metric} is {M7results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5

if den == 0:
    M7results[metric] = 0.000
else:
    M7results[metric] = num / den
print(f"{metric} is {M7results[metric]: .3f}")
```

```
 ACC is  0.538
 TPR is  1.000
 TNR is  0.000
 PPV is  0.538
 NPV is  0.000
 F1 is  0.700
 MCC is  0.000
```

Model M8

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(224, 224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM8.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
Saving TC_1.mp40.jpgF.jpg to TC_1.mp40.jpgF.jpg
Saving TC_1.mp4120.jpgF.jpg to TC_1.mp4120.jpgF.jpg
BO_1.mp40.jpgF.jpg is real
BO_1.mp4120.jpgF.jpg is real
BO_1.mp4240.jpgF.jpg is real
EM_1.mp40.jpgF.jpg is real
EM_1.mp4120.jpgF.jpg is real
EM_1.mp4240.jpgF.jpg is real
JL_1.mp40.jpgF.jpg is real
JL_1.mp4120.jpgF.jpg is real
```

```
# Upload the fake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(224, 224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM8.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
BO_0.mp40.jpgF.jpg is real
BO_0.mp4120.jpgF.jpg is deepfake
EM_0.mp40.jpgF.jpg is real
EM_0.mp4120.jpgF.jpg is deepfake
JL_0.mp40.jpgF.jpg is deepfake
JL_0.mp4120.jpgF.jpg is real
JL_0.mp4240.jpgF.jpg is real
QE_0.mp4120.jpgF.jpg is deepfake
QE_0.mp4240.jpgF.jpg is real
TC_0.mp40.jpgF.jpg is real
TC_0.mp4120.jpgF.jpg is real
TC_0.mp4240.jpgF.jpg is deepfake
```

```
# Figures for performance calculation
TP = true_pos = 14 # real faces predicted as real
```

```
TN = true_neg = 5 # fake faces predicted as fake
FP = false_pos = 7 # fake faces predicted as real
FN = false_neg = 0 # real faces predicted as fake
M8results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M8results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {M8results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M8results[metric] = TP / (TP + FN)
print(f"{metric} is {M8results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M8results[metric] = TN / (TN + FP)
print(f"{metric} is {M8results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
M8results[metric] = TP / (TP + FP)
print(f"{metric} is {M8results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
M8results[metric] = TN / (TN + FN)
print(f"{metric} is {M8results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M8results[metric] = 2 / (1 / M8results["PPV"] + 1 / M8results["TPR"])
print(f"{metric} is {M8results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
M8results[metric] = num / den
print(f"{metric} is {M8results[metric]: .3f}")
```

```
ACC is  0.731
TPR is  1.000
TNR is  0.417
PPV is  0.667
NPV is  1.000
F1 is  0.800
MCC is  0.527
```

Model M9

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM9.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
Saving TC_1.mp40.jpgF.jpg to TC_1.mp40.jpgF.jpg
Saving TC_1.mp4120.jpgF.jpg to TC_1.mp4120.jpgF.jpg
BO_1.mp40.jpgF.jpg is real
BO_1.mp4120.jpgF.jpg is real
BO_1.mp4240.jpgF.jpg is real
EM_1.mp40.jpgF.jpg is real
EM_1.mp4120.jpgF.jpg is real
EM_1.mp4240.jpgF.jpg is real
JL_1.mp40.jpgF.jpg is real
JL_1.mp4120.jpgF.jpg is real
JL_1.mp4240.jpgF.jpg is real
QE_1.mp40.jpgF.jpg is real
QE_1.mp4120.jpgF.jpg is real
QE_1.mp4240.jpgF.jpg is real
TC_1.mp40.jpgF.jpg is real
TC_1.mp4120.jpgF.jpg is real
```

```
# Upload the fake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM9.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
```

```
# Figures for performance calculation
TP = true_pos = 14 # real faces predicted as real
TN = true_neg = 2 # fake faces predicted as fake
FP = false_pos = 10 # fake faces predicted as real
FN = false_neg = 0 # real faces predicted as fake
M9results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M9results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {M9results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M9results[metric] = TP / (TP + FN)
print(f"{metric} is {M9results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M9results[metric] = TN / (TN + FP)
print(f"{metric} is {M9results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
M9results[metric] = TP / (TP + FP)
print(f"{metric} is {M9results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
M9results[metric] = TN / (TN + FN)
print(f"{metric} is {M9results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M9results[metric] = 2 / (1 / M9results["PPV"] + 1 / M9results["TPR"])
print(f"{metric} is {M9results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
M9results[metric] = num / den
print(f"{metric} is {M9results[metric]: .3f}")
```

```
ACC is  0.615
TPR is  1.000
TNR is  0.167
PPV is  0.583
NPV is  1.000
F1 is  0.737
MCC is  0.312
```

Model M10

```
# Upload the real faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():
```

```
  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Real/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM10.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_1.mp40.jpgF.jpg to BO_1.mp40.jpgF.jpg
Saving BO_1.mp4120.jpgF.jpg to BO_1.mp4120.jpgF.jpg
Saving BO_1.mp4240.jpgF.jpg to BO_1.mp4240.jpgF.jpg
Saving EM_1.mp40.jpgF.jpg to EM_1.mp40.jpgF.jpg
Saving EM_1.mp4120.jpgF.jpg to EM_1.mp4120.jpgF.jpg
Saving EM_1.mp4240.jpgF.jpg to EM_1.mp4240.jpgF.jpg
Saving JL_1.mp40.jpgF.jpg to JL_1.mp40.jpgF.jpg
Saving JL_1.mp4120.jpgF.jpg to JL_1.mp4120.jpgF.jpg
Saving JL_1.mp4240.jpgF.jpg to JL_1.mp4240.jpgF.jpg
Saving QE_1.mp40.jpgF.jpg to QE_1.mp40.jpgF.jpg
Saving QE_1.mp4120.jpgF.jpg to QE_1.mp4120.jpgF.jpg
Saving QE_1.mp4240.jpgF.jpg to QE_1.mp4240.jpgF.jpg
Saving TC_1.mp40.jpgF.jpg to TC_1.mp40.jpgF.jpg
Saving TC_1.mp4120.jpgF.jpg to TC_1.mp4120.jpgF.jpg
BO_1.mp40.jpgF.jpg is real
BO_1.mp4120.jpgF.jpg is real
BO_1.mp4240.jpgF.jpg is real
EM_1.mp40.jpgF.jpg is real
EM_1.mp4120.jpgF.jpg is real
EM_1.mp4240.jpgF.jpg is real
JL_1.mp40.jpgF.jpg is deepfake
JL_1.mp4120.jpgF.jpg is real
JL_1.mp4240.jpgF.jpg is deepfake
QE_1.mp40.jpgF.jpg is real
QE_1.mp4120.jpgF.jpg is real
QE_1.mp4240.jpgF.jpg is real
TC_1.mp40.jpgF.jpg is real
TC_1.mp4120.jpgF.jpg is real
```

```
# Upload the fake faces to make prediction
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/GDrive/My Drive/MyVideo/Faces/Fake/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = modelM10.predict(images, batch_size=1)
  if classes[0]>0.5:
    print(fn + " is real")
  else:
    print(fn + " is deepfake")
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BO_0.mp40.jpgF.jpg to BO_0.mp40.jpgF.jpg
Saving BO_0.mp4120.jpgF.jpg to BO_0.mp4120.jpgF.jpg
Saving EM_0.mp40.jpgF.jpg to EM_0.mp40.jpgF.jpg
Saving EM_0.mp4120.jpgF.jpg to EM_0.mp4120.jpgF.jpg
Saving JL_0.mp40.jpgF.jpg to JL_0.mp40.jpgF.jpg
Saving JL_0.mp4120.jpgF.jpg to JL_0.mp4120.jpgF.jpg
Saving JL_0.mp4240.jpgF.jpg to JL_0.mp4240.jpgF.jpg
Saving QE_0.mp4120.jpgF.jpg to QE_0.mp4120.jpgF.jpg
Saving QE_0.mp4240.jpgF.jpg to QE_0.mp4240.jpgF.jpg
Saving TC_0.mp40.jpgF.jpg to TC_0.mp40.jpgF.jpg
Saving TC_0.mp4120.jpgF.jpg to TC_0.mp4120.jpgF.jpg
Saving TC_0.mp4240.jpgF.jpg to TC_0.mp4240.jpgF.jpg
BO_0.mp40.jpgF.jpg is real
BO_0.mp4120.jpgF.jpg is deepfake
EM_0.mp40.jpgF.jpg is deepfake
EM_0.mp4120.jpgF.jpg is deepfake
JL_0.mp40.jpgF.jpg is real
JL_0.mp4120.jpgF.jpg is real
JL_0.mp4240.jpgF.jpg is real
QE_0.mp4120.jpgF.jpg is deepfake
QE_0.mp4240.jpgF.jpg is real
TC_0.mp40.jpgF.jpg is deepfake
TC_0.mp4120.jpgF.jpg is deepfake
TC_0.mp4240.jpgF.jpg is real
```

```
# Figures for performance calculation
TP = true_pos = 12 # real faces predicted as real
TN = true_neg = 6 # fake faces predicted as fake
FP = false_pos = 6 # fake faces predicted as real
FN = false_neg = 2 # real faces predicted as fake
M10results = {}

# Accuracy
# Accuracy: the number of correctly predicted samples / total number of samples
metric = "ACC"
M10results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {M10results[metric]: .3f}")

# True Positive Rate
# Recall / Sensitivity: the number of samples actually and predicted as Positive / total number of samples actually Positive
metric = "TPR"
M10results[metric] = TP / (TP + FN)
print(f"{metric} is {M10results[metric]: .3f}")

# True Negative Rate
# Specificity: the number of samples actually and predicted as Negative / total number of samples actually Negative
metric = "TNR"
M10results[metric] = TN / (TN + FP)
print(f"{metric} is {M10results[metric]: .3f}")

# Positive Predictive Value
# Precision: the number of samples actually and predicted as Positive / total number of samples predicted as Positive
metric = "PPV"
M10results[metric] = TP / (TP + FP)
print(f"{metric} is {M10results[metric]: .3f}")

# Negative Predictive Value
# The number of samples actually and predicted as Negative / total number of samples predicted as Negative
metric = "NPV"
M10results[metric] = TN / (TN + FN)
print(f"{metric} is {M10results[metric]: .3f}")

# F1 score
# Harmonic Mean of Precision and Recall
metric = "F1"
M10results[metric] = 2 / (1 / M10results["PPV"] + 1 / M10results["TPR"])
print(f"{metric} is {M10results[metric]: .3f}")

# Matthew's correlation coefficient
# Matthew's coefficient range between [-1, 1]. 0 usually means totally random predictions. 1 means a perfect classifier, while a negative va
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
M10results[metric] = num / den
print(f"{metric} is {M10results[metric]: .3f}")
```

```
ACC is  0.692
TPR is  0.857
```

```
TNR is  0.500
PPV is  0.667
NPV is  0.750
F1 is  0.750
MCC is  0.386
```

End of notebook