RESEARCH BASED MODERN TECHNIQUES ENHANCEMENT ON DEEPFAKE DETECTION

By

Author: Qiang (John) Li

A RESEARCH PROJECT SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGY
YORK UNIVERSITY
TORONTO, ONTARIO

2020

Abstract

In the last few years, we have seen a surge of manipulated digital images and videos, generated by sophisticated AI and machine learning models and tools. Those convincing-looking forgery could lead to significant social impacts. There is an ever-increased need to develop automatic deepfake detection techniques to fight against this kind of forgery.

Many automatic forgery detection techniques have been developed. In this report, a complete and optimized forgery detection package is proposed and tested with the popular Kaggle's deepfake challenge dataset. In the preprocessing transformation stage, the facenet Mtcnn face detection system is implemented, as it outperforms all the other existing face detection systems in our experiments. In the next stage, the latest state-of-the-art EfficientNet-B7 network model is adapted to extract the feature embedding. Those embedding are further processed by a "PCA + t-SNE " dimension reduction technique in combination with HDBSCAN clustering method. Finally, EfficientNet-B7 + Softmax CNN classifier network is trained to provide very competitive deepfake prediction results.

Index Terms — EfficientNet, Anomaly Detection, Algorithms Optimization, Feature Engineering, Image Augmentation, Face Detection

Acknowledgments

I cannot express enough thanks to my supervisor, Professor Xiangji (Jimmy) Huang. The completion of this project would not have been accomplished without his guidance and encouragement. My thanks also go to my family for their unconditional support through the years of my study.

Table of Contents

1 Introduction

1.1 Video Forgery

Manipulating video is nothing new. It happened decades ago, but at that time, doing video forgery took time, required highly skilled artists, and a lot of money. I can still remember the manipulated footage from “Forrest Gump” from 1994 (Figure 1).

Figure 1.1. Digitally inserted footage of manipulated JFK

However, in the digital world, with the advance of modern computers, the manipulation process has become much cheaper and amazingly easy. Then comes the deepfake, the technology that changed the whole game of video forgery. It provides the ability to make a convincing fake video, including some people who might seek to weaponize it for malicious purposes.

With the advance of sophisticated digital editing software such as deepfakes, the reliability of using digital images and videos as authenticated proofs or corroboratory evidence has been greatly challenged. Deepfakes can replace a person in an existing image or video with someone else's likeness to a high degree of realism by employing powerful and advanced machine learning and artificial intelligence techniques.

1.2 Motivation - Pursuing the Truth

Increasingly, new uses are being found for deepfakes. Recreating long-dead artists in museums or editing video without the need for a reshoot, deepfake technology will allow us to experience things that no longer exist, or that have never existed. And aside from having numerous applications in entertainment and education, Deepfakes have several implications in media and society, media production, media representations, media audiences, gender, law, and regulation, and politics. Deepfakes confuse people, what happens if we can no longer trust our eyes or our ears?

For more than a century, audio and video have functioned as evidence of truth. Not only have sound and images recorded our history, they have also informed and shaped our perception of reality.

Some people already question the facts around events that unquestionably happened, like the Holocaust, the moon landing and 9/11, despite video proof. If deepfakes make people believe they can't trust video, the problems of misinformation and conspiracy theories could get worse. A single convincing deepfake could alter our trust in audio and video for good. Finding the truth in digital domain therefore has become increasingly critical. Reliable detectors are mandatory to assist our judgement.

1.3 Mystery of "Deep Learning + Fake"

Deepfakes is the portmanteau of "deep learning" and "fake". At the core of the deepfakes code is an autoencoder, a deep neural network that learns how to take an input and compress it down

into a small set of concentrated key patterns, and then learns how to regenerate the original input from these key patterns.

Rather than try to explain in pure words, we would like to show how it works through the process of creating a deepfake video by transferring the face of person A to person B in an existing video.

1.3.1 Autoencoder

The autoencoder is basically a Deep Learning Convolutional Neural Network (CNN), which has two major subnetworks, namely an encoder itself and a decoder as the counterpart. Suppose hundreds of facial pictures from both person A and person B were already collected, then they can be used to train the encoder to produce a set of concentrated key patterns or features. In order to catch the real features of a person's face, those unnecessary variants would be treated as noise. The extracted patterns or features are in turn used to train the decoder to regenerate the corresponding pictures, just like drawing a suspect sketch, where the features described by a witness (encoder) helps a composite sketch artist (decoder) to draw a picture of the suspect.

It should be noted that, in order to do the trick, different decoders are used for person A and person B individually. That means one decoder is trained with the extracted features from pictures of person A and another decoder is trained with the extracted features from pictures of person B.

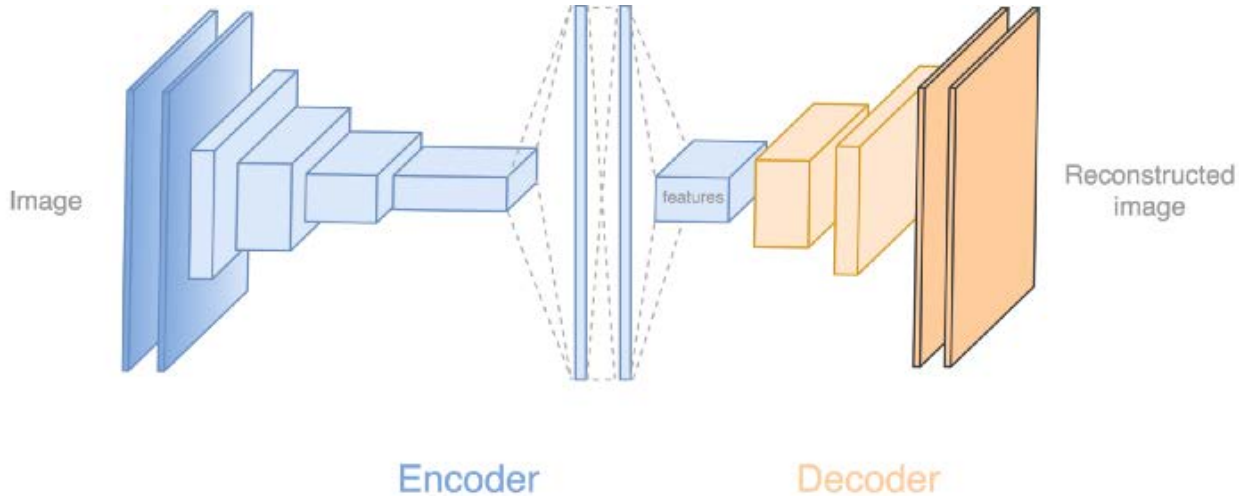

Figure 1.2. Autoencoder's architecture

After a good training, the decoder would be able to generate a picture that closely matches the original picture. Now it is time to manipulate a real video where person A is presented. In each frame of the video, the area of person A's face is located and identified by using a computer vision processing tool called face detection and this portion of the image is fed into the encoder of the autoencoder to extract the features. However, instead of the decoder for person A, the decoder trained for person B is used to reconstruct the face potion. The result looks more like person B and is used to replace the corresponding part in the original frame.

As shown in figure 1.3, the potion that contains the face of person A passes through the encoder and comes with the features related to face angle, skin tone, facial expression, lighting and other information that is important to reconstruct the person A. When the second decoder is used to reconstruct the image, person B's face was drawn instead of person A. The quality of the image forgery depends on the complexity of the model and the amount of the training data.

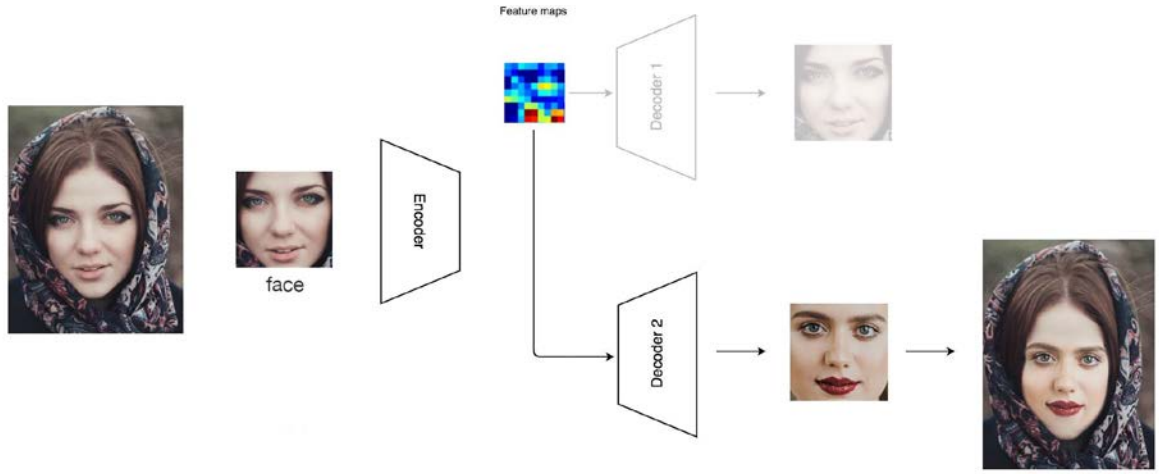

Figure 1.3. Face swapping with trained Autoencoder

1.3.2 The Generative Adversarial Network

A more advanced deepfake system can have an additional discriminator and utilize the architecture of the generative adversarial network, GAN in short, which is a machine learning framework for estimating generative models via an adversarial process. A GAN trains the decoder, and a discriminator (a deep network CNN classifier) in an adversarial relationship simultaneously. The decoder creates new images from extracted features presentation of the source material. While the discriminator attempts to distinguish whether facial images are original or fake.

Given a training set, this technique learns to generate new data with the same statistics as the training set. A well trained GAN can generate new pictures that look at least superficially authentic to human observers.

When we feed real images to this discriminator, we train the discriminator itself to recognize real images better. When we feed created images into the discriminator, we use it to train our autoencoder to create more realistic images. We turn this into a race that eventually the created images are not distinguishable from the real ones.

Powerful autoencoders and GANs can provide solutions for lots of complex computer vision media, generating deepfakes is one of them. The discriminator which can distinguishes whether facial images are original or fake is the forgery detection solution that we are looking for.

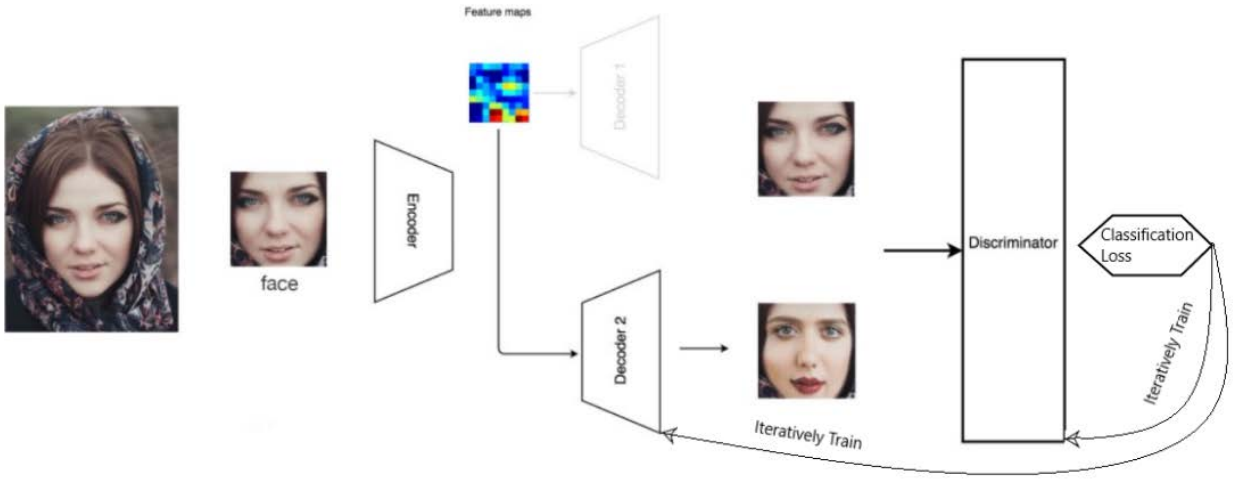

Figure 1.4 Deepfake with GAN architecture

As shown Figure 1.4, the decoder and a discriminator are in an adversarial relationship. The decoder generates new images from the extracted features of the original images and sends them to the discriminator. The discriminator attempts to determine whether images are original or fake. The mistakes made by the discriminator is called the discrimination loss. During the

training process, the discriminator strives to minimize its discrimination loss, while the decoder seeks to maximize the discrimination loss with the forgery images it produces.

1.4 A CNN Classifier – Deepfake Detection Solution

A Convolutional Neural Network (CNN) is a multilayered neural network with a special architecture to detect complex features in data. CNNs have been used in image recognition, powering vision in robots, and for self-driving vehicles.

The deepfake detection network which is a CNN classifier, consists of two parts, one CNN model base and an additional binary classifier as in Figure 1.5.

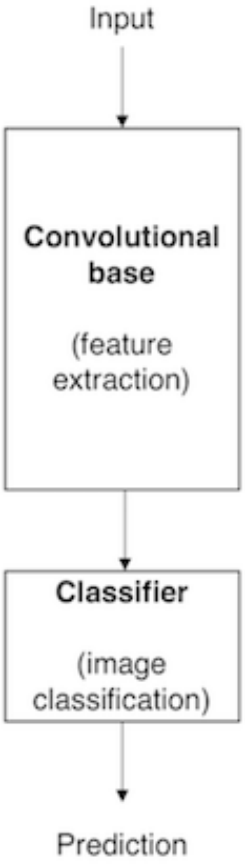

Figure 1.5 CNN classifier

The loss functions of a CNN classifier that reflect the distance between the distribution of the forgery data and the distribution of the real data are used for distinguish whether facial images are original or fake [15].

1.5 Related Work

Multiple video forgery detection techniques have been proposed for a variety of tasks in the last few years.

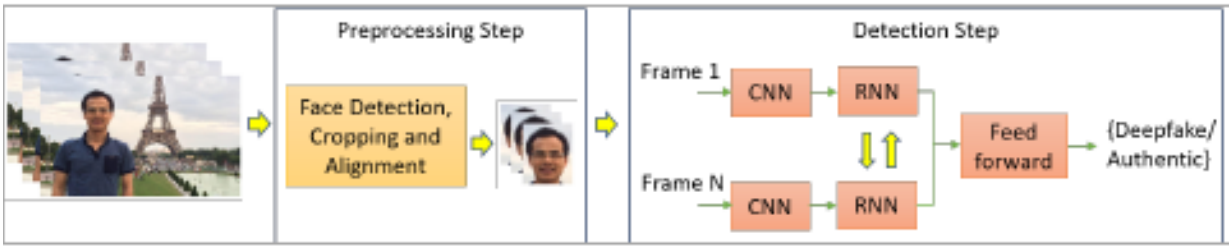

Figure 1.6 A two-steps Detection Process

In Figure 1.6, Sabir et al. concluded a two-step process for face manipulation detection where the preprocessing step aims to detect, crop and align faces on a sequence of frames and the second step distinguishes manipulated and authentic face images by combining convolutional neural network (CNN) and recurrent neural network (RNN) [30].

Andreas R. et al. extract and preprocess the facial region from the input dataset. The FaceForensics++ paper evaluates several learning based methods such as MesoNet, XceptionNet, SVM and others (as in figure 1.7). Among the tested methods, the XceptionNet model achieved the highest performance [6].

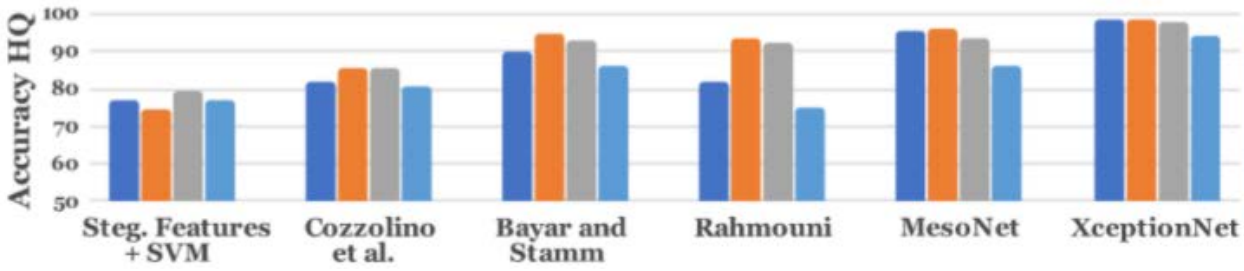

Figure 1.7 Detection Methods Comparison

Classification tasks employ deep learning architectures that usually outperform traditional machine learning models. Following this trend, DeepFake detection approaches are based on

deep learning networks for their embedding performance. Multi CNNs are assembled together as one detection network solution (figure 1.8), Nicolo B et al. demonstrated a promising forgery detection result by combining CNNs with attention layers for training [31].

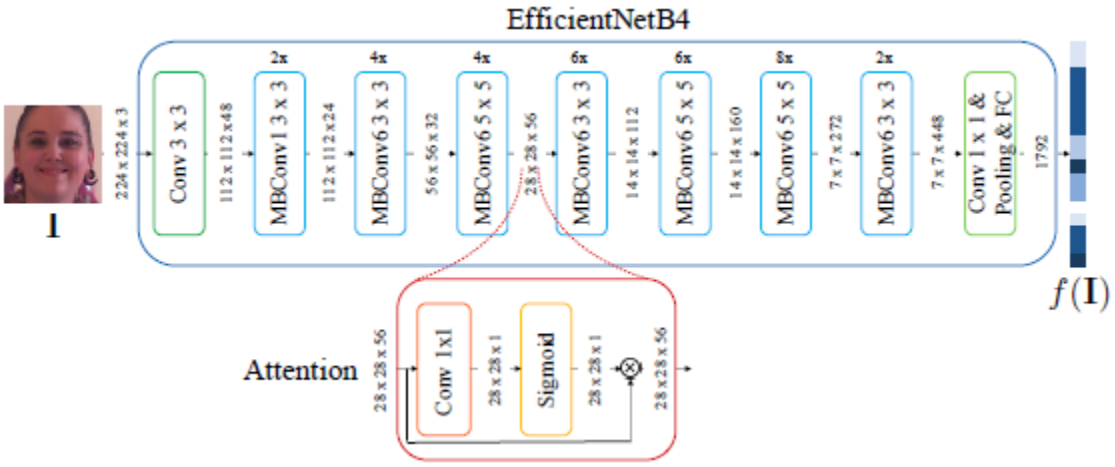

Figure 1.8 Proposed EfficientNetB4Att Architecture

Inspired by those research above, this paper focus on database preprocess, performance comparison, network ensembles, proposing a sufficiently lightweight solution.

1.6 Deepfake Detection Objectives

This paper introduced a competitive video forgery detection solution, a CNN classifier adapted with current computer vision domain knowledge. In Section 2, the chosen deepfake detection challenge dataset is analyzed and preprocessed to improve the forgery prediction and dataset issues.

There are three major tasks required by the forgery detection development. Solutions for those three tasks have been optimized based on research and experiments. First, a performance benchmark compares on various detection packages. The best package is adapted as part of our

final solution. Section 3.3 based on latest research, the most promising state-of-the-art CNN model base is identified and implemented for the face embedding. Those embedding are passed through an introduced “PCA + t-SNE" dimension reduction and the chosen clustering technique for face feature extraction. The Introduced EfficientNet-B7 + Softmax CNN classifier network are simulated for deepfake detection. Competitive evaluation results in section 3.5, proves that the latest domain trends would optimize performance and accuracy of the video forgery project. Future works suggested at the end of this paper for continuing development.

2 Large-Scale Facial Forgery Database

A number of datasets featuring manipulated facial footage have been recently published with the purpose of training and evaluating facial manipulation detection and classification. These include the Celeb-DF, UADFV, DeepFake-TIMIT, VTD dataset, FaceForensics and its successor FaceForensics++. More recently, the DeepFake Detection Challenge. The DFDC dataset aims to improve upon this work through data diversity, agreement from persons depicted in the dataset, size of dataset, and metrics, among others. Table 2.1 summarized the specs of the most relevant datasets in the deepfake space.

Dataset	Ratio tampered:original	Total videos	Source	Participants Consent
Celeb-DF [4]	1 : 0.51	1203	YouTube	N
FaceForensics [8]	1 : 1.00	2008	YouTube	N
FaceForensics++ [9]	1 : 0.25	5000	YouTube	N
DeepFakeDetection [11] (part of FaceForensics++)	1 : 0.12	3363	Actors	Y
DFDC Dataset	1 : 0.28	119154	Actors	Y

Table 2.1. Summarized specs of deepfake datasets

The unique dataset for the challenge consists of about 120,000 .mp4 files over 470 GB. An attached metadata.json contains filename of those .mp4, label (REAL/FAKE), original and split columns. According to Brian D. et al, diversity in several axes (gender, skin-tone, age, etc.) has been considered and actors recorded videos with arbitrary backgrounds thus bringing visual variability [7]. So far, the deepfake detection challenge dataset from Kaggle is the best fit for the development objective.

On the other hand, the competition was hosted by Kaggle and winners were selected using the log-loss score against the public test set. And there is a leaderboard (LB) where winner's scores were listed. So once a solution is made, results can be compared with the LB score list for competition. The LB position is also a quality criteria for the deepfake detection solution. By excluding the audio analysis, this paper focuses on video faces for forgery detection. The DFDC dataset can be accessed at

https://www.kaggle.com/c/16880/datadownload/dfdc_train_all.zip.

2.1 Descriptive Data Analysis

	videoname	original_width	original_height	label	original
0	gcpjokmohj.mp4	129	129	REAL	NaN
1	cxqhrnxgvd.mp4	185	186	FAKE	jwbsicnbyc.mp4
2	boxfdbfgsn.mp4	186	185	FAKE	jcfhtypqnv.mp4
3	rxfawebswj.mp4	267	267	FAKE	mtpvrsdblr.mp4
4	jmrfbhqnqk.mp4	185	185	REAL	NaN

Figure 2.2 Dataset Property Sample

This supervised dataset has some issues for performing training and testing tasks.

- Issue #1: Over 80% of the training videos are “Fake” (Figure 2.3). Turns out the training set does not share the same distribution. Imbalanced classes can be an additional hindrance. While there may be sufficient data quantity for training, equally important, but under sampled classes will suffer from poor class-specific accuracy. This phenomenon is intuitive. If the model learns from a few examples of a given class, it is less likely to predict the class invalidation and test applications.

- Issue #2: Display Aspect Ratio and size of videos are different (Figure 2.2). Those facts could cause an extra input arrangement process.

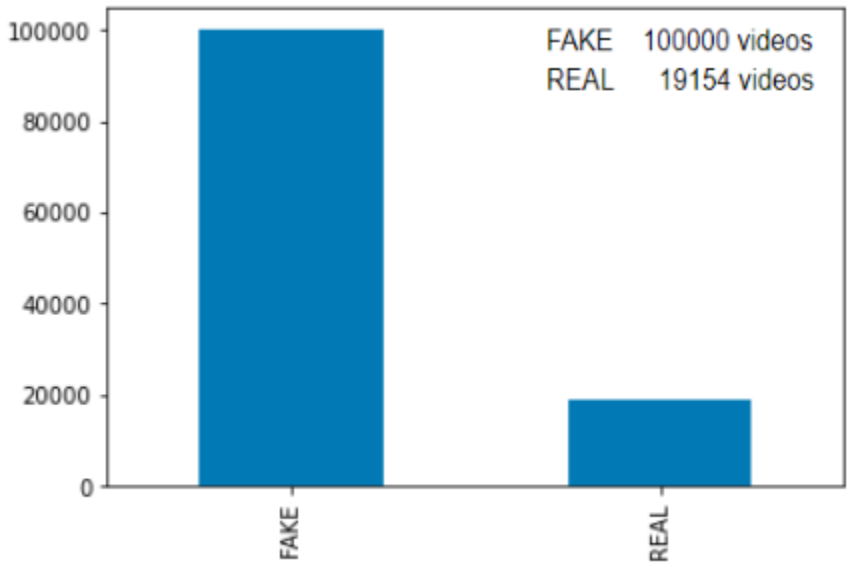

Figure 2.3 Real/ Fake Casses Bar Chart

2.2 Pre-process Recommendations & Ingredients

Polychronis C. et al studied the impact of dataset pre-processing and prediction aggregation on the DeepFake detection problem. They proposed a pre-processing step that improves the training dataset quality. They found that pre-processing has a considerable impact on the detection task and boosts model performance by improving the quality of the generated training set. With 2.3% performance improvement on DFDC test dataset, experimental results show that the proposed pre-processing approach leads to considerable improvements in the performance of detection models [19].

Pre-processing includes all transformations performed on the raw data before they are provided to a model for training or inference. To improve forgery detection In terms of videos, the proposed transformations include the following transformations: (2.2.1) Frame Extraction & Face Detection, (2.2.2) Data Balancing Adjustment, (2.2.3) Face Image Resizing & Normalization, (2.2.4) Face Margin, and (2.2.5) Image Augmentation [6][7][19] .

2.2.1 Frame Extraction & Face Detection

The video forgery detection is to check whether faces in a video are real or fake. We need to crop and extract those faces from .MP4 files and to make predictions. Face instance capturing and extraction through reading videos file was the main bottleneck for the runtime. Applying the right face detection tools would be one of the critical steps.

There are around 300 frames per each video with a total 119154 videos in DFDC database. According to the research, Learning to Detect Manipulated Facial Images, instead of using all frames of videos, sampling partial frames from videos would boost the performance [6].

Face detection is an essential step for building an accurate deepfake detector. Building detection models around a face results in higher accuracy compared to building models for whole images as conclusion by Brian Dolhansky et al. [7].

2.2.2 Data Balancing Adjustment

With a large number of fake labeled data will potentially generate a noisy dataset (figure 2.3), this will hurt the overall performance of the detection system.

To fix the class imbalance issue #1 in Section 2.1, I extract different numbers of frames depending on whether the video is fake or real. Due to the limitation of the datasets, assuming all the faces captured from Fake labeled video are forgeries. As the fake videos are much more than real ones at ratio 5:1; I extracted 8 faces from each FAKE video and 40 faces from each REAL video, then a similar number of Real/Fake faces are ready for training and testing. Such transformations can prevent overfitting in training.

2.2.3 Face Image Resizing & Normalization

We need to realize the fact that the bigger the image the more resources required for the processing, but if the face image is too small then it doesn't obtain enough information for prediction. So the image needs to be justified to a proper size for further data analytics.

Faces need to be cropped and extracted from video frames by using face detection tools. Faces size can be customized to be identical as the input for predictive modeling during the face

extraction process to solve #2 issue in section 2.1. The above solutions are considered as parts of optimization step after face detection which will be further detailed in section 3.2.

Through comparing similar datasets of face recognition studies, a face area about 150*150 size would be a good choice for fine-grained facial patterns analytics.

2.2.4 Face Margin

A recent research by Brian D., Russ H.et al, indicates adding face margin and tuning augmentations as ingredients which can effectively improve test accuracy [7]. To cook the face margin ingredient, I set image margin when cropping face, where

new_face_width=face_width(1+margin) #same with height*

According to the finding in the research, The Deepfake Detection Challenge (DFDC) Preview Dataset, margin of 0.5~0.7 worked significantly better than 0 margin and further reduced the cv-lb gap [7]. I have reason to believe that models learn to detect some inconsistency between manipulated region and surrounding region. After additional margin added, the cropped face size been increase as 240*240 (as in Figure 2.4.).

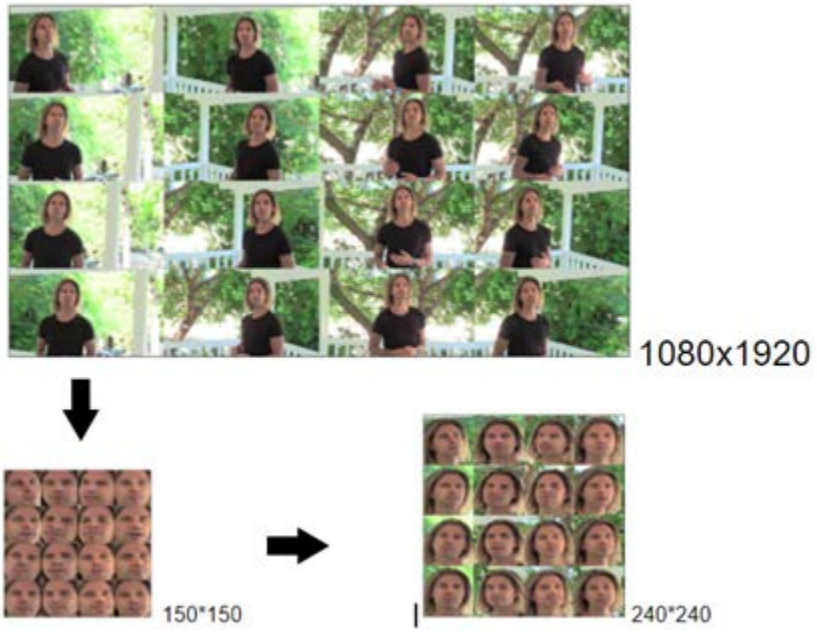

Figure 2.4. Face Crop for Extraction

Figure 2.5. Augmentation

2.2.5 Image Augmentation

As another ingredient suggested by Brian D., Russ H.et al, I create test-time augmentation code as following

```
def create_train_transforms(size=240):
  return Compose([
      ImageCompression(quality_lower=60, quality_upper=100, p=0.5),
      GaussNoise(p=0.1),
      PadIfNeeded(min_height=size, min_width=size,
border_mode=cv2.BORDER_CONSTANT),
      OneOf([RandomBrightnessContrast(), FancyPCA(), HueSaturationValue()],
p=0.7),
      ToGray(p=0.2),
      ShiftScaleRotate(shift_limit=0.1, scale_limit=0.2, rotate_limit=10,
border_mode=cv2.BORDER_CONSTANT, p=0.5),
 ...
  ]
  )
```

I used some basic augmentation here such as compression, noise, shift, scale, rotate, brightness contrast, hue, saturation etc. as in figure 2.5. To tackle the overfitting problem, these augmentations are generated during preprocess process right after it extract faces (Section 3.2).

To improve effectively on both performance and quality, those research based preprocess approaches such as data frames sampling, margins on customized extracted face image, data balancing and test-time augmentations were adopted with our forgery detection solution [6][7][19].

3 Deepfake Detection Enhancements - Divide & Conquer

3.1 Project Brief

When people confront AI-manipulated media, there's no single tell-tale sign of how to spot a fake. Data scientists and data engineers are curious about strategies and techniques for building public awareness of updated forgery detections.

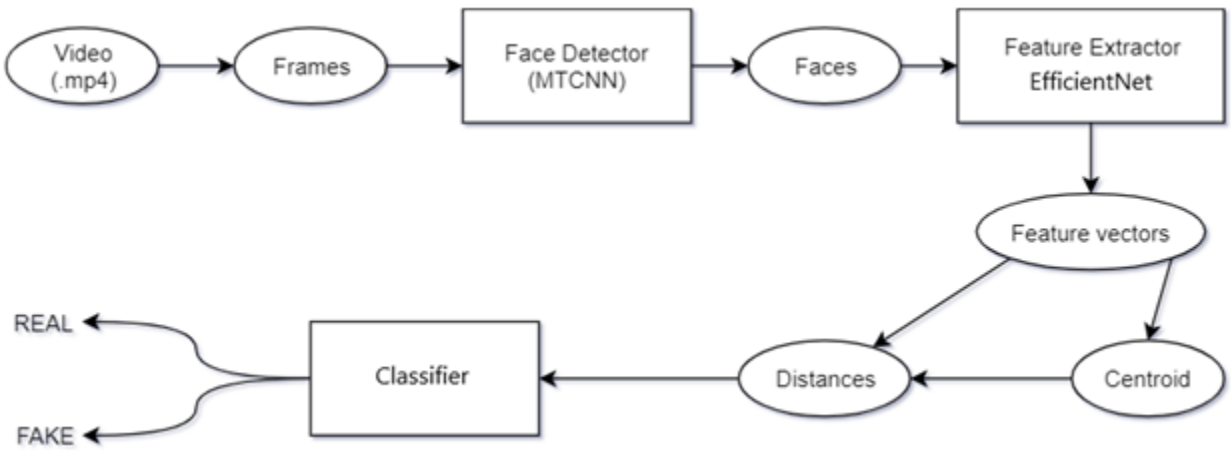

Figure 2.1. Approach Demonstration

Here I introduce a research approach shown in Figure 2.1., which helps to understand the paper progress and the development steps of the forgery detection solution.

My approach for video forgery detection design will be divided into three major subsections as Section 3.2, 3.3 and 3.4.

3.2 Face Detection

The Performance of a deepfake detector depends heavily on the performance of the face detection model [6] [7]. Face detection is a problem in computer vision of locating and localizing one or more faces in a photograph. Locating a face in a photograph refers to finding the coordinate of the face in the image, whereas localization refers to demarcating the extent of the face, often via a bounding box around the face.

3.2.1 Packages & Options

OpenCV would be the first package that popped into my mind when came to computer vision projects. Both the Haarcascade and the DNN (Deep Neural Network) packages of OpenCV can perform face detection tasks.

"To conclude, if we want a fast face detection algorithm we should use Dlib. On the other hand, if we want an algorithm to detect a large number of faces our choice can be Facenet or Mtcnn" [17]. However, according to Strahinja Stefanvic's comparison experiment, Mtcnn, Dlib and Facenet stand out in a crowd (including OpenCV's packages) from performance and quality aspects. The algorithms are briefed as following

Facenet can be described as a unified embedding for Face detection and Clustering. It is a system that, when given a picture of a face, it will extract high-quality features from the face known as face-embedding. A deep convolutional neural network uses a triplet loss function for training. It encourages vectors of the same identity to become more similar, whereas vectors of different identities are expected to become less similar.

Mtcnn Facenet-pytorch is a facenet pretrained based Mtcnn face detection algorithm, typically providing a considerable speed-up. A batch option should be structured as a list of PIL (PILLOW) images of equal dimension. Mtcnn process images in batches when input images are resized to have the same dimension prior to detection [8].

In the following example, we use Mtcnn Facenet-pytorch to detect multiple faces in:

1. 60 batches of PIL images
2. 1 batch PIL image only

Dlib is a very useful and practical toolkit for making real world machine learning and data analysis applications. It is a CNN based detector and it is generally capable of detecting faces from almost all angles.

MTCNN or Multi-Task Cascaded Convolutional Neural Network is unquestionably one of the most popular and most accurate face detection tools today. As such, it is based on a Deep learning architecture, it specifically consists of 3 neural networks (P-Net, R-Net, and O-Net) connected in a cascade.

The following face detection algorithms are listed for comparison.

1. Mtcnn from facenet-pytorch with batch: https://pypi.org/project/facenet-pytorch/
2. Mtcnn from facenet-pytorch without batch: https://pypi.org/project/facenet-pytorch
3. Dlib: https://pypi.org/project/dlib/
4. Mtcnn from tensorflow: https://pypi.org/project/Mtcnn/

3.2.2 Package Benchmarking

Benchmark data: three 300 frames video files with different resolutions as in Table 3.1.

Package	FPS (540x960)	FPS (720x1280)	FPS (1080x1920)
facenet-pytorch MTCNN	25.5	20.32	12.97
facenet-pytorch MTCNN (non-batched)	19.68	14.81	9.75
dlib	14.53	8.39	3.8
TensorFlow MTCNN	8.23	5.7	3.04

Table 3.1. Face Detection Packages Performance

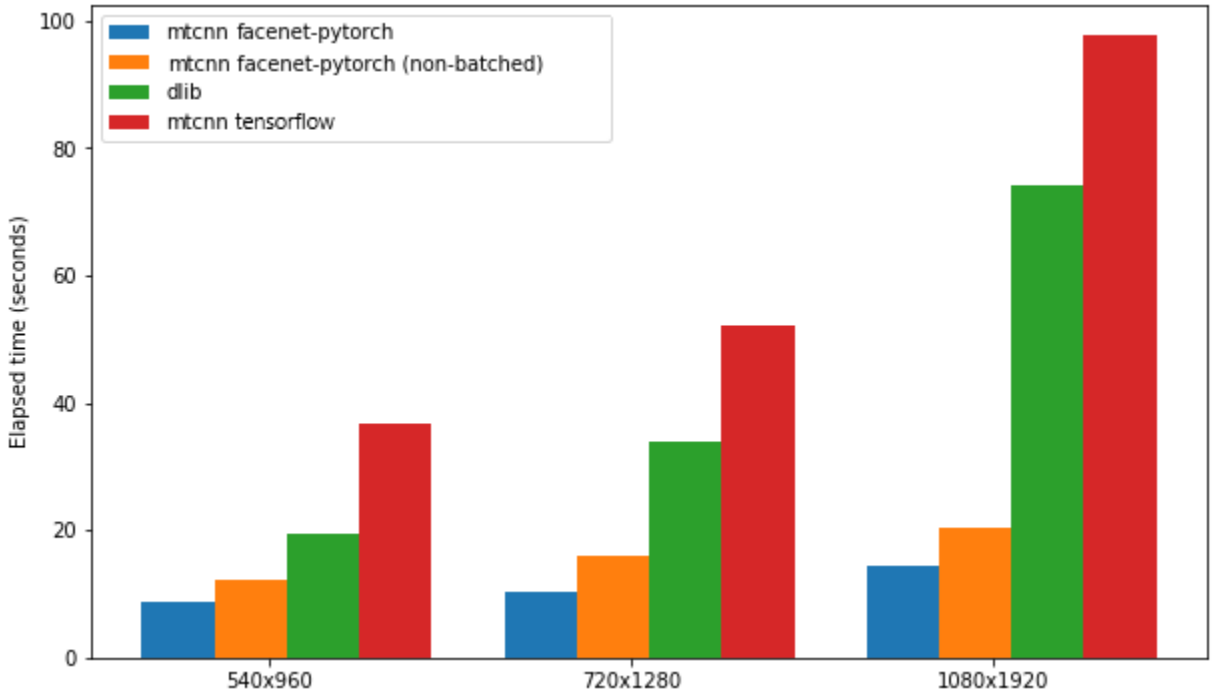

Figure 3.1. Face Detection Packages Simulation Chart

I mostly stick with facenet-pytorch, but for the sake of comparison I also include the tensorflow implementation of Mtcnn, which has significantly slower performance than facenet-pytorch package.

Although we find the winner, batched Mtcnn function from facenet-pytorch, among those packages there are other facts that need to be considered (Figure 3.1). Two questions got my attention.

Question #1: Does the Mtcnn function performance gap between facenet-pytorch and tensorflow mean differences in conversion?

Question #2: Between Dlib and Mtcnn, which package has a better accuracy?

The details of those discussions are addressed in next as Section 3.2.1 and Section 3.2.2.

3.2.2.1 Facenet VS. Tensorflow

Conversion of parameters returned from Tensorflow and Pytorch tells the truth.

```
>>> compare_model_outputs(mdl, sess, torch.randn(5, 160, 160, 3).detach())
Passing test data through TF model

tensor([[-0.0142,  0.0615,  0.0057,  ...,  0.0497,  0.0375, -0.0838],
        [-0.0139,  0.0611,  0.0054,  ...,  0.0472,  0.0343, -0.0850],
        [-0.0238,  0.0619,  0.0124,  ...,  0.0598,  0.0334, -0.0852],
        [-0.0089,  0.0548,  0.0032,  ...,  0.0506,  0.0337, -0.0881],
        [-0.0173,  0.0630, -0.0042,  ...,  0.0487,  0.0295, -0.0791]])

Passing test data through PT model

tensor([[-0.0142,  0.0615,  0.0057,  ...,  0.0497,  0.0375, -0.0838],
```

[-0.0139, 0.0611, 0.0054, ..., 0.0472, 0.0343, -0.0850],
[-0.0238, 0.0619, 0.0124, ..., 0.0598, 0.0334, -0.0852],
[-0.0089, 0.0548, 0.0032, ..., 0.0506, 0.0337, -0.0881],
[-0.0173, 0.0630, -0.0042, ..., 0.0487, 0.0295, -0.0791]],
grad_fn=<DivBackward0>)

The equivalence of the outputs from the original tensorflow models and the facenet-ported models have been tested. Those results are identical, the Tensorflow package will be eliminated from my consideration.

3.2.2.2 Dlib VS. Facenet

A Better accuracy score is what we are looking for. Specifically accuracy is tightly associated with prediction quality. I experimented with a test with 2000 labeled images. While Facenet had 90% accuracy, Dlib had only 75%. So I'll go with the batched Mtcnn facenet-pytorch algorithm.

3.2.2.3 Further Code Improvement

The above sections have successfully proved the efficiency of the Facenet batched Mtcnn. Additionally, I have a new alternative finding which further improves our runtime performance.

For example only catch every 10th frame, don't do the following:

```
capture = cv.VideoCapture(movie_path)
for i in range(0, num_frames):
        ret, frame = capture.read()
        if i % 10 == 0:
        # do something with frame
```

Instead, it's faster to call grab(), which reads the next frame but doesn't decode it, and only call retrieve() on the frames you're interested in:

```
capture = cv.VideoCapture(movie_path)
for i in range(0, num_frames):
        ret = capture.grab()
        if i % 10 == 0:
        ret, frame = capture.retrieve()
        # do something with frame
```

capture.release()

I experienced 162s and 107s as comparison for 10 videos face extraction. This has already made quite a bit of difference in runtime; and think about time it will save for over 100000 videos.

3.2.3 Experiments Summary

Through the benchmark and experiments, the batched MTCNN facenet-pytorch algorithm is the advanced solution among all options. The optimized batched facenet solution and those pre-process recommendations in Section 2.2 have been finalized for overfitting and performance enhancement which further reduce performance cost by 26%+.

3.3 All about Face Embedding

Face embedding are vectors mapping from face images to a compact Euclidean space where distances directly represent face similarity [9]. CNN models' reputation is known for their image process efficiency. Building models would be critical for final performance of the deepfake detection. The extracted face embedding can lead to a range of recognition approaches.

3.3.1 CNN Model

Among the different types of neural networks such as recurrent neural networks (RNN), long short term memory (LSTM), artificial neural networks (ANN), etc., CNNs work phenomenally well in various computer vision tasks like image classification, object detection, and face recognition.

3.3.1.1 CNN Models Brief

Convolutional neural networks (CNNs) is a class of deep learning methods, designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers [26].

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function. A small array of numbers, called a kernel, is applied across the input, which is an array of numbers, called a tensor. An element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a feature map (figure 3.2.1). This procedure is repeated applying multiple kernels to form an arbitrary number of feature maps, which represent different characteristics of the input tensors [16].

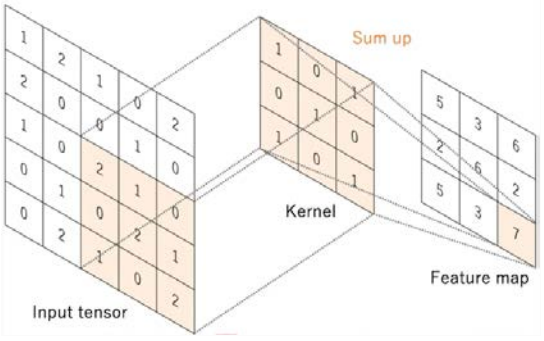

Figure 3.2.1 Convolution Operation

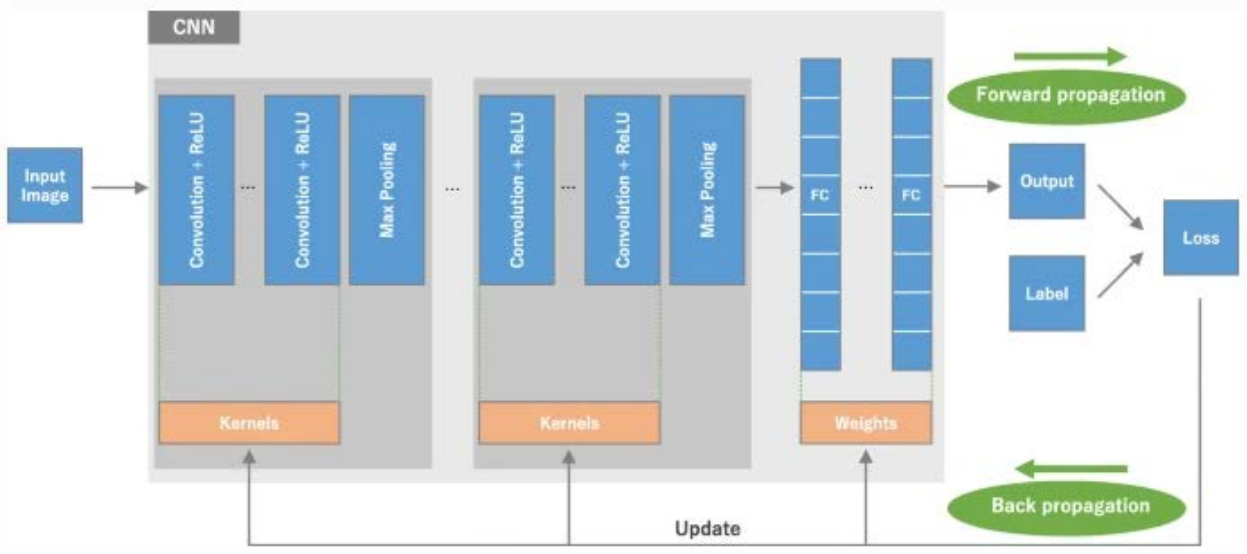

Figure 3.2.2 Example of a CNN architecture

With The CNN architecture example in Figure 3.2.2, we can get familiar with the concepts and advantages, as well as limitations. CNNs are commonly developed at a fixed resource cost, and then scaled up in order to achieve better accuracy when more resources are made available. The conventional practice for model scaling is to arbitrarily increase the CNN depth or width, or to use larger input image resolution for training and evaluation. While these methods do improve accuracy, they usually require tedious manual tuning, and still often yield suboptimal performance.

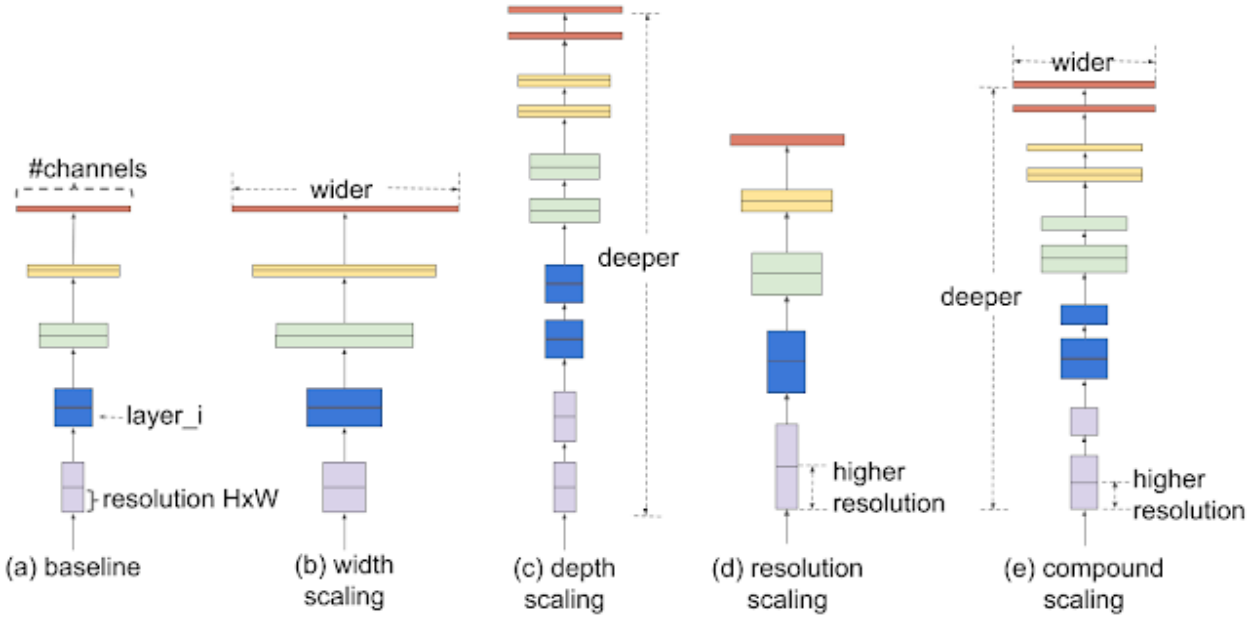

Figure 3.3 Comparison of different Scaling methods

What if, instead, we could find a more principled method to scale up a CNN to obtain better accuracy and efficiency? I have limited knowledge regarding machine learning image processing. Base on a CNN benchmark work posted on Google AI Blog [1], I am equipped with the best CNN option.

3.3.1.2 EfficientNets & Embedding

A model's performance are depend on learnable parameters such as kernels and weights. Through forward and backward propagations, the loss functions will optimize those learnable parameters by using Gradient descent algorithm. The ongoing learnable parameters is essential to leverage CNN potential in image pattern recognition – embedding.

Based on the CNN Models Size vs. Accuracy Comparison chart in Figure 3.4, EfficientNets achieved state-of-the-art accuracy in 5 out of the 8 widely used transfer learning datasets, super

pass the rest with 10x better efficiency (smaller and faster model - model architecture showed in figure 3.6), and could potentially serve as a new state-of-art foundation for future computer vision tasks. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks” suggested that the EfficientNets provide significant improvements to model efficiency [1]. In particular, while being smaller than the best existing CNN, the EfficientNet-B7 model achieves new state-of-the-art accuracy which are chosen for our deepfake detection enhancement.

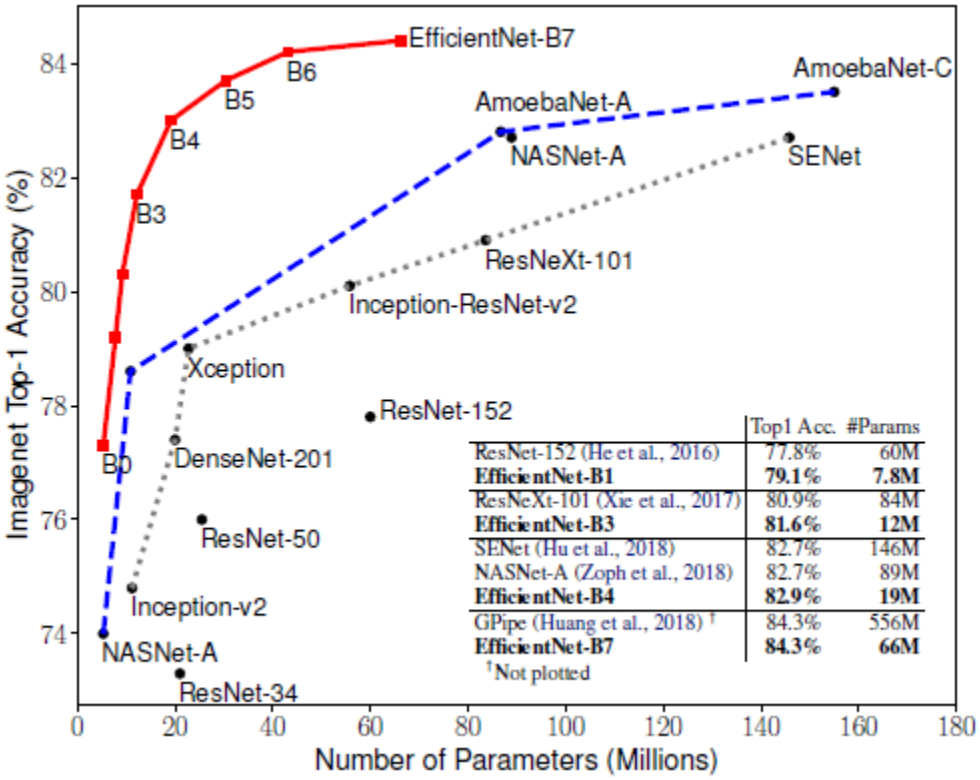

Figure 3.4. CNN Models Size vs. Accuracy Comparison

```
from torchvision.transforms import ToTensor
from efficientnet_pytorch import EfficientNet
model = EfficientNet.from_pretrained('efficientnet-b7')
tf_img = lambda i: ToTensor()(i).unsqueeze(0)
embeddings = lambda input: model(input)
list_embs = []
with torch.no_grad():
        for face in tqdm(face_files):
        t = tf_img(Image.open(face)).to(device)
        e = embeddings(t).squeeze().cpu().tolist()
        list_embs.append(e)
```

With the pre-trained Efficientnet-b7 model code implement above, I am not training “Fake/Real” predictions, but generating 512-dimensional embedding.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

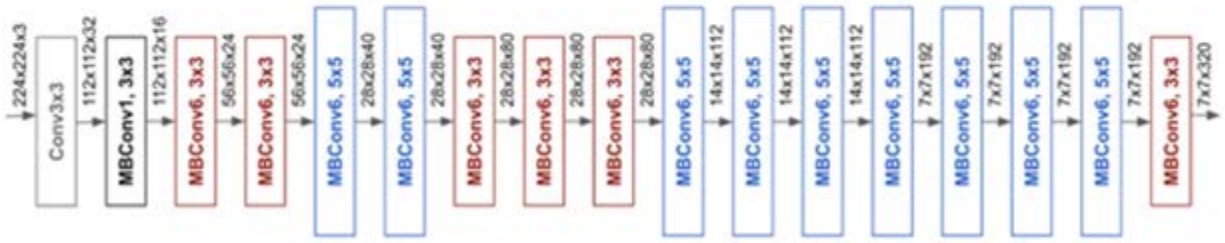

Figure 3.6. EfficientNet model Architecture

3.3.2 Dimension Reduction Comparison & Optimization

More features means more vast time and space complexity cost for the model. Not all the features available to us are relevant to our problem. Training a data model on a dataset with many dimensions often leads to overfitting. Dimension reduction is about discovering non-linear,

non-local relationships in data. Dimension reduction techniques can reduce the noise and unnecessary parts of the data in order to improve relationship accuracy and scalability.

The 512-dimensional high quality embedding from Efficientnet-b7 implementation requires to be extracted for further inference analysis. Different dimension reduction techniques can have quite different computational complexity, the nature of the data trying to reduce can also matter. Here we will take a brief look at the performance characteristics of a number of dimension reduction implementations.

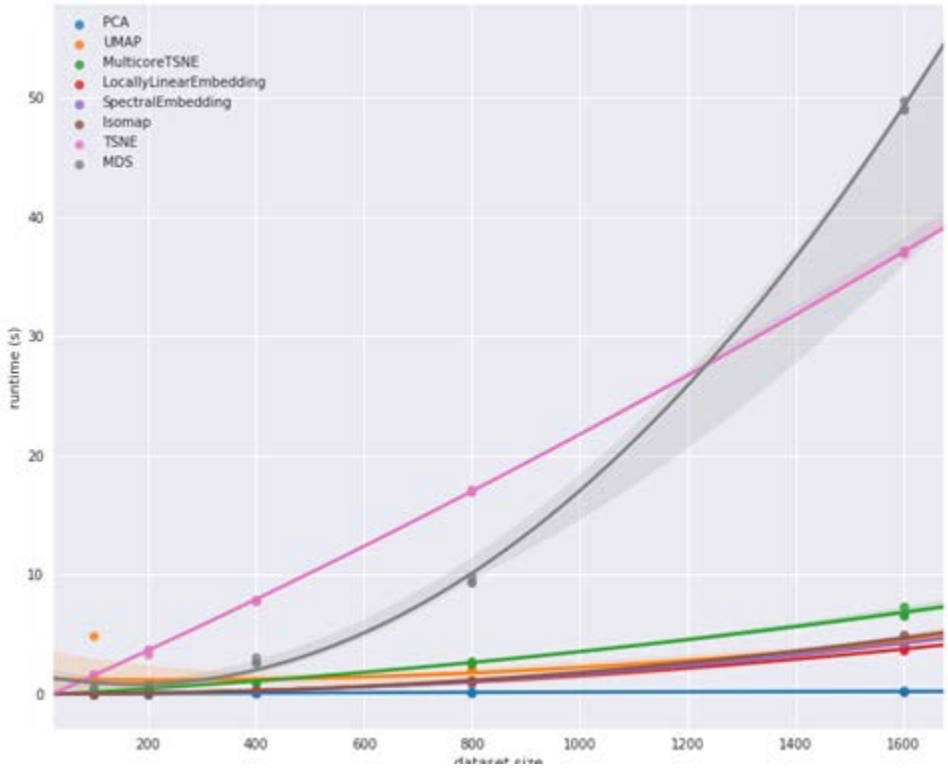

Figure 3.7 Dimension reduction algorithms performance chart

3.3.2.1 The Fastest PCA

According to benchmarking performance and scaling of python clustering algorithms, not difficult to conclude with dimension reduction algorithms performance chart in figure 3.7, PCA, Principal Component Analysis, is far and away the fastest option [2]. It combines your input features in linear mapping that you can drop the least important feature while still retaining the most valuable parts of all of the features [20]. Let's implement the fastest performing option and see what happens.

Figure 3.8 PCA result in 3D space projection

By looking at the PCA 3D space projection in Figure 3.8, we can see that similar images are close to each other. But it looks really hard to separate them in clusters/groups. With the PCA algorithm, I am potentially giving up a lot for that speed.

3.3.2.2 PCA + t-SNE

T-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. It is extensively applied in various works including image processing [20].

T-SNE, while not competitive with PCA in speed, is the alternative option in terms of its none-linear, probabilistic technique. From the conclusion in benchmarking performance and scaling of python clustering algorithms showing in figure 3.7, the t-SNE is clearly much slower than PCA algorithms. However, for a small dataset size (less than 200), t-SNE can be fairly efficient.

What we need is a quality result with a good runtime cost. What if a balance solution exists which combines advantages from both t-SNE and PCA**?** In order to achieve dimension reduction optimization on both performance and the quality, I ensemble PCA and t-SNE techniques together, by applying PCA's output as t-SNE's input. The introduced "PCA + t-SNE" algorithm only sacrifices a little speed for the last mile t-SNE process.

```
from sklearn.manifold import TSNE

# PCA first to speed it up
x = PCA(n_components=50).fit_transform(df['embedding'].tolist())
x = TSNE(perplexity=50,
      n_components=3).fit_transform(x)
```

With dimension reduction space projections on face embedding, PCA does not compete well as compared to t-SNE. Given the quality of results demonstrated in 3D space projections from both PCA and t-SNE as figure 3.8. & 3.9., the introduced PCA and t-SNE techniques is clearly an optimized innovation for this project.

Figure 3.9. T-SNE result in 3D space projection

3.3.3 Clustering Analysis

There are a host of different clustering algorithms and implementations. The performance and scaling can depend on the underlying algorithm. So rather than analyzing the implementations and deriving asymptotic performance numbers for various implementations. I am looking for the benchmark findings to assist my clustering solution.

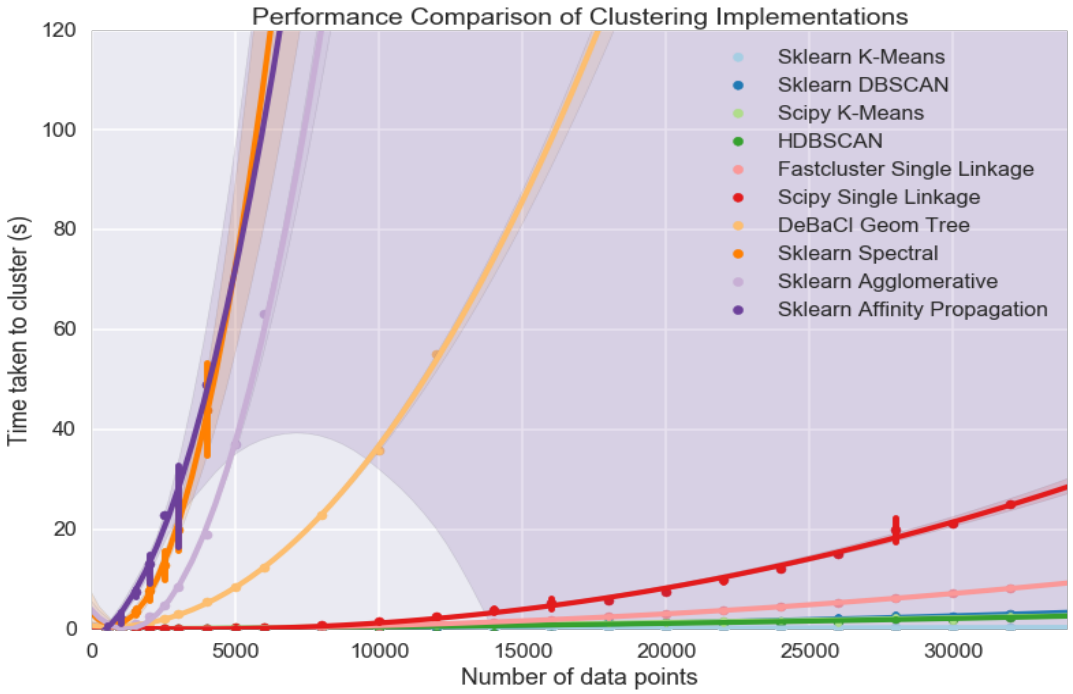

Figure 3.10 Performance Comparison of Clustering Implementation.

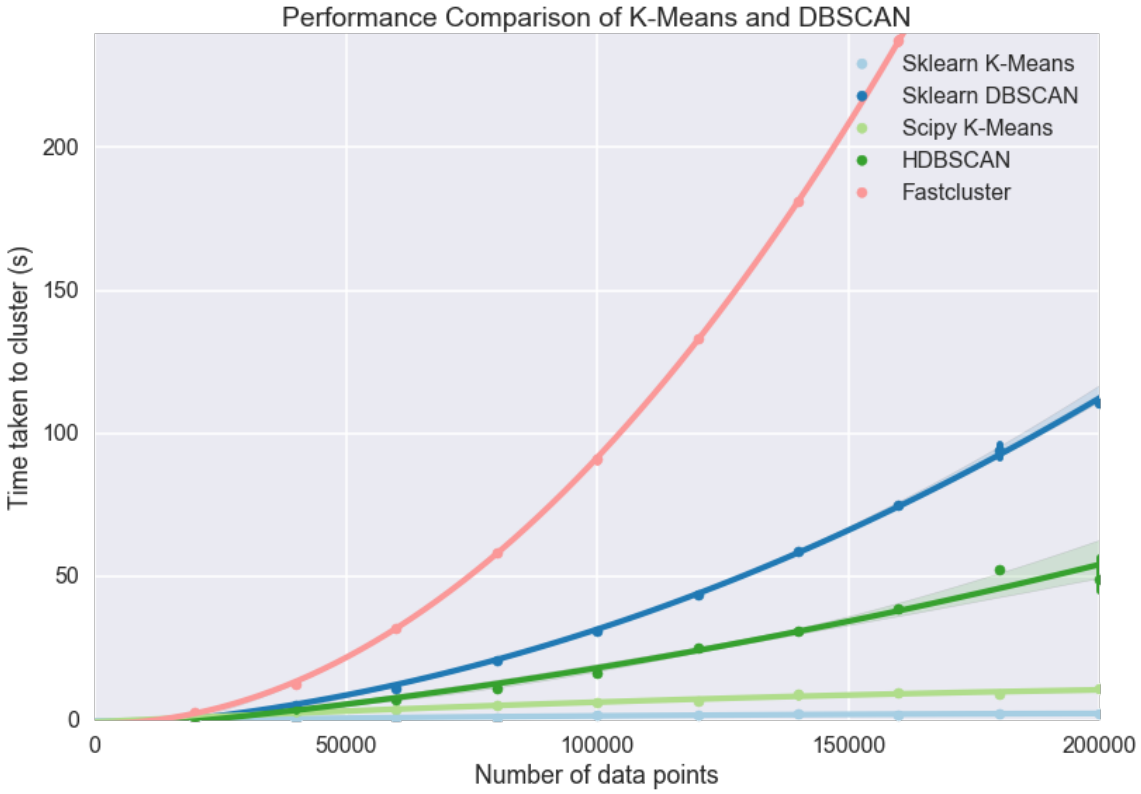

Figure 3.11. Performance Comparison of K-Means and DBSCAN

3.3.3.1 Clustering Algorithms Comparison

Concluded from Benchmarking Performance and Scaling of Python Clustering Algorithms, A few features stand out from clustering implementation performance comparison as shown in figure 3.10 & 3.11. For our approach, the clustering options are significantly constrained with our 100,000 data points. There appear to be only three implementation left, K-Means (the fast implementations), DBSCAN, and HDBSCAN. The rest of slow cases are largely from Sklearn and include agglomerative clustering are filtered out [3].

Density clustering algorithms use the concept of reachability i.e. how many neighbors have a point within a radius. With our scenario, DBSCAN is lovelier because it doesn't need parameter, k, which is the number of clusters we are trying to find, which K-Means need. And K-Means is not a particularly good clustering algorithm, particularly for exploratory data analysis [3]. When you don't know the number of clusters hidden in the dataset. It's a good decision to use DBSCAN. DBSCAN produces a varying number of clusters, based on the input data.

HDBSCAN, a robust hierarchical version of DBSCAN, with better performance and stability than DBSCAN aligned with Performance Comparison in figure 3.11. HDBSCAN algorithms is the optimized Clustering solution that is integrated into our forgery detection project.

3.3.3.2 Embedding Inferential Analysis

I scale out a little further and get a closer look at the result that managed 130,000 points under forty-five minutes in figure 3.12. Finally we got 520 clusters for our analysis.

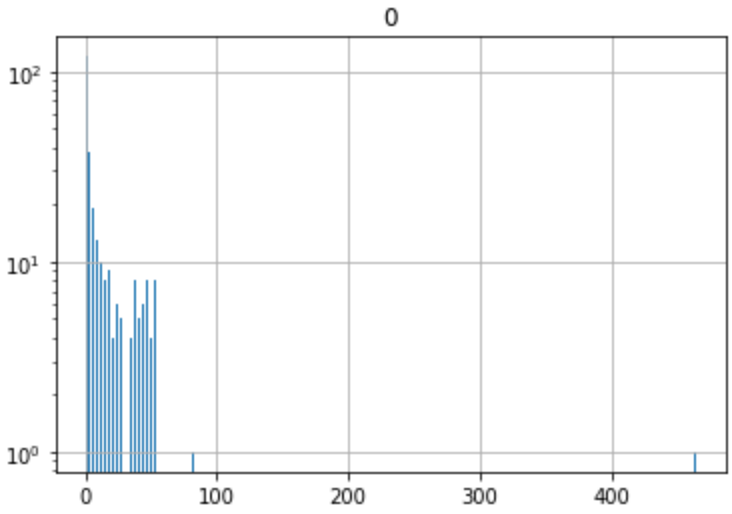

Figure 3.12. Embedding cluster

Through the data visualization, I have a better understanding from those clustered embedding (Figure 3.13 - Figure 3.16).

From the observation of cluster-faces demonstration, face embedding are able to represent facial features for most cases. However, according to figures 16 & 17, the embedding struggle to represent features of the Faces of black people.

Figure 3.13. Cluster 181 with 5 chunks

Figure 3.14. Cluster 91 with 11 chunks

Figure 3.15. Cluster 15 with 6 chunks

Figure 3.16. Cluster 5 with 5 chunks

Black skin face embedding improvement remains as an unsolved challenge according to The Best Algorithms Struggle to Recognize Black Faces Equally, a business trend report in 2019 [11].

3.4 Binary Classifier comparison

3.4.1 Sigmoid VS. Softmax

Sigmoid has been applied with many machine learning applications where a real number needs to be converted to a probability. However, I found Sigmoid functions might have saturation problems. In table 3.18, the values that it receives are probably too far from zero, and the sigmoid is returning 'extreme' results [5].

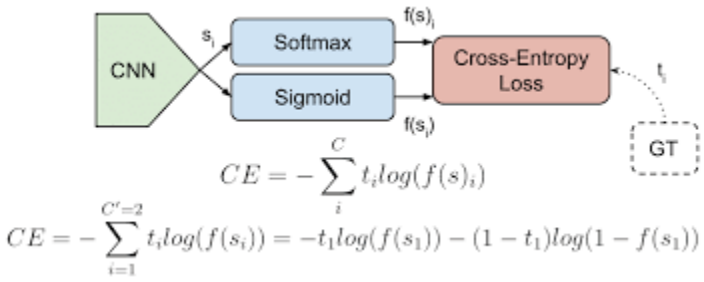

Figure 3.17. Softmax VS. Sigmold and cross-entropy loss

Raw output values	[3.2, -5.7, 0.6]
Applying **sigmoid** to raw output values	sigmoid calculation for the first raw output value: $\sigma(3.2)= \frac{e^{3.2}}{1+e^{3.2}} = 0.96$ result of sigmoid calculation for all three output values: [0.96, 0.0033, 0.65] Sum: 0.96 + 0.0033 + 0.65 = 1.61 ≠ 1
Applying **softmax** to raw output values	softmax calculation for the first raw output value: $\text{softmax}(3.2)= \frac{e^{3.2}}{e^{3.2}+e^{-5.7}+e^{0.6}} = 0.93$ result of softmax calculation for all three output values: [0.93, 0.00013, 0.069] Sum: 0.93 + 0.069 + 0.00013 = 1

Table 3.18. Output value comparison table for Sigmoid and Softmax

Softmax Function has a structure very similar to Sigmoid function. As discussed in “Multi-label vs. Multi-class Classification: Sigmoid vs. Softmax”, the most important difference is that it is preferred in the output layer of deep learning models, especially when it performs a probabilistic interpretation [4].

Nandita Bhaskhar recommended “When you have only one reasonable classifier output, use a Softmax” [5]. Compared with Sigmoid, Softmax performs fairly better when used as a classifier.

```
PREDS=[]
with torch.no_grad():
    for batch_idx, images in enumerate(bar):
    x = images.to(device)
    ...
      logits = model(x)
    ...
    PREDS += [torch.softmax(logits, 1).detach().cpu()]
    LOGITS.append(logits.cpu())
    PREDS = torch.cat(PREDS).cpu().numpy()
```

The Softmax layer classifier image embedding probability distribution over Real/Fake class labels to make probable predictions as Table 3.19.

	video	label	embedding	distance	predict
0	mulebpmvzo.mp4	FAKE	[-0.01554779801517725, -0.05940323695540428, -...	0.720846626	1
1	lbqfkivkqr.mp4	FAKE	[0.012129027396440506, -0.05558238551020622, -...	0.781097192	1
2	ndpytliklg.mp4	REAL	[-0.04476125165820122, 0.01827099174261093, -0...	0.047551088	0
3	nrherwjzle.mp4	FAKE	[-0.027975086122751236, 0.022589877247810364, ...	0.623628607	1
4	ntttcqmyjm.mp4	FAKE	[0.0032095250207930803, -0.042558614164590836,...	0.631369798	1

Table 3.19. Prediction results from Efficientnet-b7 + Softmax

3.5 “Efficientnet-b7 + Softmax” & Evaluation

The introduced CNN classifier, Efficientnet-B7 + Softmax, as our enhanced deepfake detection network (figure 3.20) is tested with our preprocessed deepfake detection challenge datasets. I achieved a logloss score of 0.48702 which is in top 50/2114 private leaderboard (LB) position of the Kaggles' deepface detection challenge (Figure 3.22).

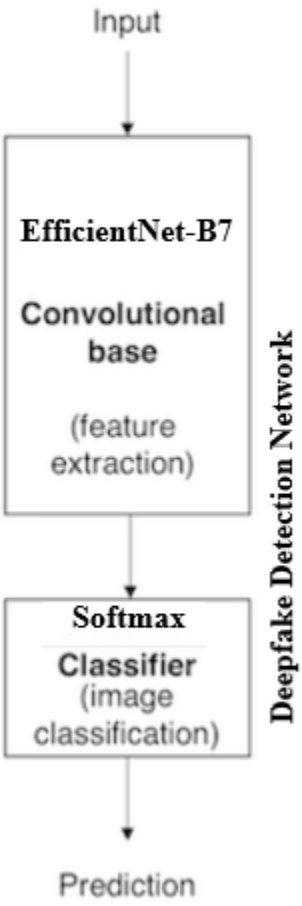

Figure 3.20 Detection Model

 # SKLearn Implementation
from sklearn.metrics import log_loss
log_loss(df['label_number'], df['predict'])

The well performed high rank in LB position not only proves the efficient of the introduced Efficientnet-B7 + Softmax network classifier and improved data preprocess steps, but also demonstrates the important of with domain knowledge adaption.

Scoring

Submissions are scored on log loss:

$$LogLoss = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot log_e(\hat{y}_i) + (1 - y_i) \cdot log_e(1 - \hat{y}_i)]$$

where:

- n is the number of videos being predicted
- y^i is the predicted probability of the video being FAKE
- y_i is 1 if the video is FAKE, 0 if REAL
- $log()$ is the natural (base e) logarithm

Figure 3.21. Log Loss Formula

4 Project Conclusion

Backed up with research and experiments, the key optimized findings include benchmarking the chosen face detection package, the “PCA + t-SNE” dimension reduction and clustering techniques experiments, and the EfficientNet-b7 + Softmax CNN classifier simulation with the preprocessed dataset. Our deepfake detection solution demonstrates competitive efficiency of forgery prediction. Project achieved optimized runtime cost with 26%+ reduction and an impressive accuracy score rank 50th of Kaggle’s privacy leaderboard.

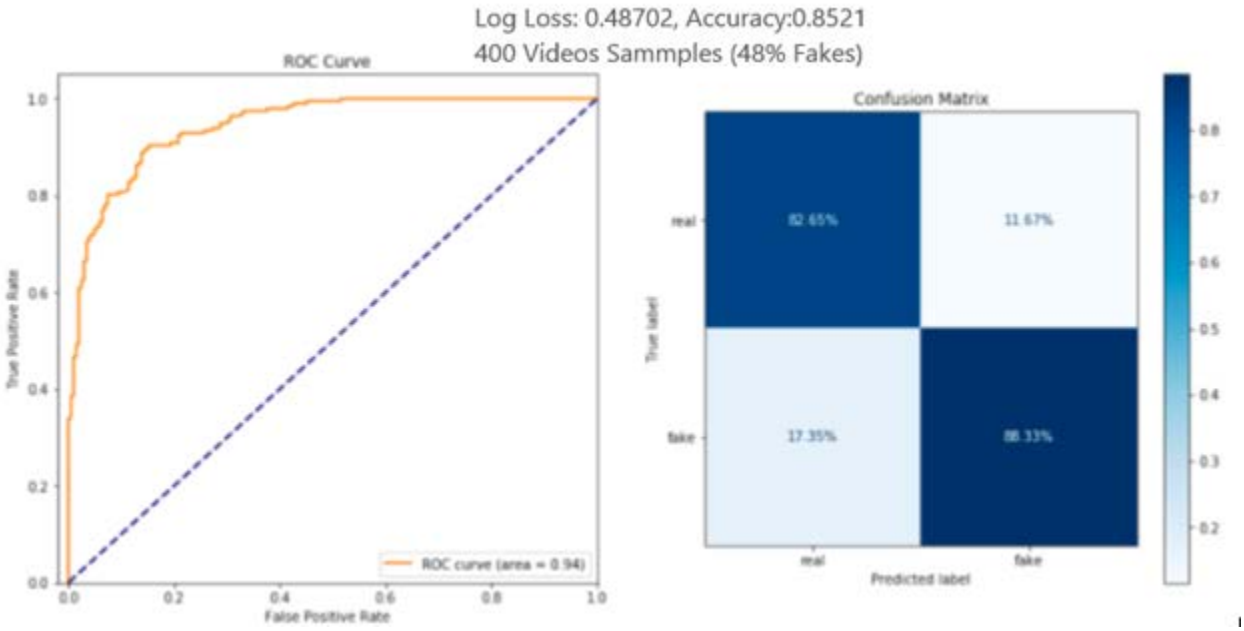

Figure 3.22 ROC-AUC curves & confusion matrix for “Efficientnet-b7 + Softmax” model

5 Issues and Future Work

5.1 Unsolved Issue

This project experienced the embedding issues for representing the face of black people which aligned with industry summary named “The Best Algorithms Struggle to Recognize Black Faces Equally” (2019). Face recognition studies on faces of black people are continued for better solutions.

5.2 Potential for Improvement

Although we got a good optimized video forgery detection solution, there are some great techniques that would make further detection improvement.

5.2.1 Fine Tuning

First, “EfficientNet-b7 + Softmax” model fine-tuning would make further justification on our prediction result. Fine tuning, the compound scaling method performs grid searches to find the relationship between different scaling dimensions of the baseline network under a fixed resource constraint. Additionally, the excluded fake audio analysis would also improve the prediction result.

5.2.2 Attention Augmented Convolutional Networks

Additionally, Convolutional networks have been the paradigm of choice in many computer vision applications. The convolution operation however has a significant weakness in that it only operates on a local neighborhood, thus missing global information. Self-attention, on the other hand, has emerged as a recent advance to capture long range interactions, but has mostly been applied to sequence modeling and generative modeling tasks [23].

Recent rise of the transformer-based architectures seem to possess the capacity to learn the underlying structure of text and, as a consequence, to learn representations that generalize images. Face images processed with Attention Augmented Convolutional Networks would be a good ideal to improve the accuracy of the deepfake detection. At the same time, the parallelization can pass all the image pillow size at the same time into the encoder block and the output which could enhanced the performance.

5.2.4 Transfer Learning

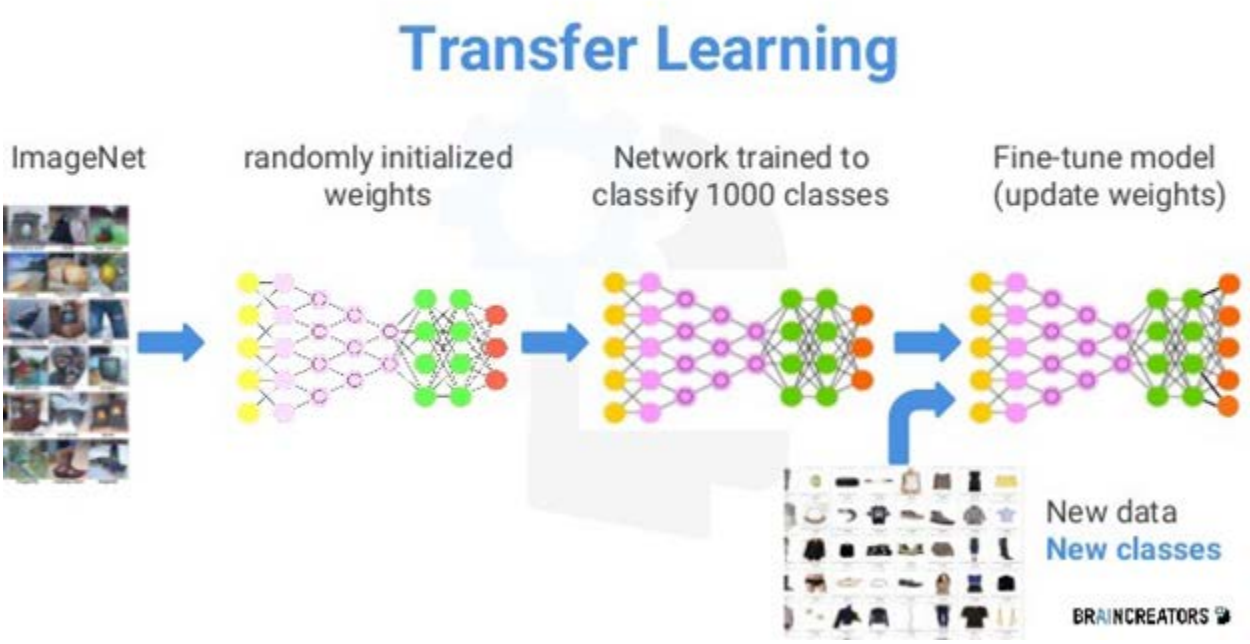

Figure 5.1 Transfer Learning

Furthermore, focusing on storing knowledge gained while solving one problem and applying it to a different but related problem, transfer learning (figure 5.1) allows rapid progress or improved performance. For example, weights from video manipulation GAN discriminator could be stored and loaded to our forgery detection model directly. Thinking about this idea, we grab the exact details of how fraudulent faces are made and update them directly into the face Anti-Fraud

system. Anti-Fraud detection would become a piece of cake. Video forgery detection couldn’t be any easier.

Reference

[1] Mingxing Tan, Quoc V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” *International Conference on Machine Learning 2019, arXiv:1905.11946 [cs.LG]*, 2019

[2] UMAP API Guide, “Performance Comparison of Dimension Reduction Implementations,” 2020. [Online]. Available: https://umap-learn.readthedocs.io/en/latest/benchmarking.html

[3] Read the Docs API Reference, “Benchmarking Performance and Scaling of Python Clustering Algorithms,” 2020. [Online]. Available: https://hdbscan.readthedocs.io/en/latest/performance_and_scalability.html

[4] Glass Box Blog, “Multi-label vs. Multi-class Classification: Sigmoid vs. Softmax,” 2020. [Online]. Available:https://glassboxmedicine.com/2019/05/26/classification-sigmoid-vs-softmax/

[5] Nandita Bhaskhar, “Interpreting logits: sigmoid vs softmax,” 2020. [Online]. Available: http://web.stanford.edu/~nanbhas//blog/sigmoid-softmax.html, 2020

[6] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, Matthias Nießner. “FaceForensics++: Learning to Detect Manipulated Facial Images,” *arXiv:1910.08854[cs.CV]*, 2019

[7] Brian Dolhansky, Russ Howes, Ben Pflaum, Nicole Baram, Cristian Canton Ferrer, “The Deepfake Detection Challenge (DFDC) Preview Dataset,” *arXiv:1910.08854 [cs.CV], 2019*

[8] Kaggle blog, “Guide to MTCNN in facenet-pytorch,” 2020. [Online]. Available: https://www.kaggle.com/timesler/guide-to-Mtcnn-in-facenet-pytorch

[9] Khanh Duc Le, “A Study of Face Embedding in Face Recognition,” 2019. [Online]. Available: https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=3377&context=theses

[10] Weilin Huang and H. Yin, "Linear and nonlinear dimensionality reduction for face recognition," *2009 16th IEEE International Conference on Image Processing (ICIP)*, Cairo, 2009, pp. 3337-3340, doi: 10.1109/ICIP.2009.5413898.

[11] WIRED Blog. “The Best Algorithms Struggle to Recognize Black Faces Equally,” 2019. [Online]. Available: https://www.wired.com/story/best-algorithms-struggle-recognize-black-faces-equally

[12] Yue Cao, Mingsheng Long, Jianmin Wang, Shichen Liu. “Deep Visual-Semantic Quantization for Efficient Image Retrieval,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1328-1337

[13] Sakshi Agarwal, Lav R. Varshney. “Limits of Deepfake Detection: A Robust Estimation Viewpoint,” *arXiv:1905.03493 [cs.LG]*, 2019

[14] A. Chintha *et al.*, "Recurrent Convolutional Structures for Audio Spoof and Video Deepfake Detection," in *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 5, pp. 1024-1037, Aug. 2020, doi: 10.1109/JSTSP.2020.2999185.

[15] Jonathan Hui. “How deep learning fakes videos (Deepfake) and how to detect it?” 2019. [Online]. Available: https://jonathan-hui.medium.com/how-deep-learning-fakes-videos-deepfakes-and-how-to-detect-it-c0b50fbf7cb9

[16] Yamashita, R., Nishio, M., Do, R.K.G. *et al.* “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging* 9, 611–629 (2018). Available: https://doi.org/10.1007/s13244-018-0639-9

[17] Strahinja Stefanovic. Data Hacker Blog. “Face detection algorithms comparison,” 2020. [Online]. Available: http://datahacker.rs/017-face-detection-algorithms-comparison

[18] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, Cristian Canton Ferrer. “The DeepFake Detection Challenge (DFDC) Dataset,” *arXiv:2006.07397[cs.CV]*, 2020

[19] Polychronis Charitidis, Giorgos Kordopatis-Zilos, Symeon Papadopoulos, Ioannis Kompatsiaris. ”Investigating the Impact of Pre-processing and Prediction Aggregation on the DeepFake Detection Task,” *arXiv:2006.07084 [cs.CV]*, 2020

[20] Datacamp Tutorials. “Introduction to t-SNE,” 2020. [Online]. Available: https://www.datacamp.com/community/tutorials/introduction-t-sne

[21] Ali Elmahmudi, Hassan Ugail, “Deep face recognition using imperfect facial data,” *Future Generation Computer Systems* 99, 213–225 (2019) Pages 213-225, ISSN 0167-739X,

[22] Armaan Pishori, Brittany Rollins, Nicolas van Houten, Nisha Chatwani, Omar Uraimov. “Detecting Deepfake Videos: An Analysis of Three Techniques,” *arXiv:2007.08517 [cs.CV]*, 2020

[23] Jean-Baptiste Cordonnier, Andreas Loukas & Martin Jaggi. “On the Relationship between Self-Attention and Convolutional Layers,” *ICLR* 2020. Available: https://arxiv.org/pdf/1911.03584v2.pdf

[24] Li, Yuezun & Lyu, Siwei. “Exposing DeepFake Videos By Detecting Face Warping Artifacts,” *arXiv:1811.00656 [cs.CV]*, 2019

[25] YubaNet Blog. “Best way to detect ‘deepfake’ videos? Check for the pulse,” 2020. [Online]. Available: https://yubanet.com/scitech/best-way-to-detect-deepfake-videos-check-for-the-pulse/

[26] Yamashita, R., Nishio, M., Do, R.K.G. *et al.* “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging* 9, 611–629 (2018). Available: https://doi.org/10.1007/s13244-018-0639-9

[27] Pedro Marcelino. Towards Data Science Blog. “Transfer learning from pre-trained models,” 2020. [Online]. Available:https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751#:~:text=A%20pre%2Dtrained%20model%20is,VGG%2C%20Inception%2C%20MobileNet).

[28] A. Chintha *et al*., "Recurrent Convolutional Structures for Audio Spoof and Video Deepfake Detection," in *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 5, pp. 1024-1037, Aug. 2020, doi: 10.1109/JSTSP.2020.2999185

[29] Google AI Blog. “EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling,” 2020. [Online]. Available: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

[30] Sabir, E., Cheng, J., Jaiswal, A., AbdAlmageed, W., Masi, I., and Natarajan, P. “Recurrent convolutional strategies for face manipulation detection in videos,” In Proceedings of *the IEEE Conference* on Computer Vision and Pattern Recognition Workshops (pp. 80-87). 2019

[31] Nicolò Bonettini, Edoardo Daniele Cannas,Sara Mandelli, Luca Bondi, Paolo Bestagini, Stefano Tubaro. “Video Face Manipulation Detection Through Ensemble of CNNs,” *arXiv:2004.07676* [cs.CV], 2020