DETECT FORGERY VIDEO BY PERFORMING TRANSFER LEARNING ON DEEP NEURAL NETWORK

_____

A Thesis

Presented to

The Faculty of the Department of Computer Science

Sam Houston State University

_____

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

_____

by

Zhaohe Zhang

May, 2019

DETECT FORGERY VIDEO BY PERFORMING TRANSFER LEARNING ON DEEP NEURAL NETWORK

by

Zhaohe Zhang

_____

APPROVED:

Qingzhong Liu, PhD
Thesis Director

Bing Zhou, PhD
Committee Member

Hyuk Cho, PhD
Committee Member

John B. Pascarella, PhD
Dean, College of Science and Engineering Technology

ABSTRACT

Zhang, Zhaohe , *Detect forgery video by performing transfer learning on deep neural network*. Master of Science (Computing and Information Science), May, 2019, Sam Houston State University, Huntsville, Texas.

Nowadays, the authenticity of digital image and videos becomes hard while the forgery techniques are more advanced. Given the recent progress on Generative Neural Network (GNN) development that may generate realistic images and videos, it becomes more difficult to detect the authenticity of digital photographs. In this thesis, we expose a popular open-source video forgery library called "DeepFaceLab" by making use of deep learning. We retrain the existing state-of-the-art image classification neural networks to capture the features from manipulated video frames. After passing various sets of forgery video frames through a well-trained neural network, a bottleneck file is created for each image, it contains the features and artifacts in forgery video that could not be captured by the human eye. Our testing accuracy is over 99% when testing DeepFake videos. We also examined our method on FaceForensics dataset and achieved good detection results on both testing set and validation set. Experiments under different data sizes confirm the effectiveness and efficiency of the proposed method.

KEY WORDS: Forgery detection, Transfer learning, Deep neural network, Face Forensics, DeepFake

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure **Page**

CHAPTER I

Introduction

Statement of the Problem

Watching video is one of the most common methods for information gathering. Every day, millions of videos are shared through social media and video websites such as YouTube. In this era of information explosion, it is vital to be able to distinguish between the authentic information from the forgery. Just five years ago, forging video was a daunting task which requires enormous amounts of money and resources. Movie studios in Hollywood has spent millions of dollars trying to create most realistic Computer Graphical (CG) effects that could amaze their audiences. Nowadays, with the advance of neural network, people without any professional equipment or CGI experience could achieve the same result by just a few mouse clicks.

In recent years, AI DeepFakes or "Deepfaking", is emerging through social media. Initially Nicolas Cage's face appeared on various video clips in which he did not participate; then a user on Reddit posted a clip that shows a pornographic wherein a porn actress face is replaced with a famous movie actress face. Even though the post has been discovered and deleted quickly, the technology used to produce such clips is still available to the public on GitHub. As its popularity increases, by possessing a high-performance GPU, anyone with moderate computer skills could make a seamless DeepFake video that looks like made by Hollywood special effect team. In early 2018, a video published by BuzzFeed shows president Obama was addressing to the public (BuzzFeedVideo, 2018). In reality, the same forgery technique was used to morph actor Jordan Peels' face to be president's look-a-like. Just within a year, thousands of clips are

created with many celebrities' faces but without their consent. These DeepFake Videos have caught many public figures amidst of confusion and panic.

Deep Network & Autoencoder

Deep Neural Networks (DNNs), which employ materialized deep learning algorithms to capture the meaningful representation of input dataset (Bengio, 2013), is a very popular research topic in recent years. There are many types of DNNs differentiated by their structures, serving different purposes; however, these functions can be categorized into several areas such as classification, feature extraction and language processing. Some notable deep learning structures like Convolutional deep neural networks (CNNs) and Recurrent neural network (RNN) have built solid foundation to produce several state-of-the-arts neural networks.

The most fundamental component of the neural network is called artificial neurons. The artificial neurons receive one or more inputs and produce the summation as the output. Each neuron usually has a separate weight that could affect the output value. The output is passed through an activation function, usually sigmoid function (Wikipedia, 2019), that could transfer the output into a binary number which represents the neuron either "Fire" or "Off" (Wikipedia, 2019). A mathematical representation of each neuron's function is given below:

Let x be the input from previous neurons, w be the weight, m represents the number of previous neurons, and sigma will be the activation function, then we have formula:

$$y_k = \sigma(\sum_{j=0}^{m} w_{kj} x_j)$$

Once the formula for single neuron is established, the neurons can be organized into different structures. A basic CNN is a layered-structure network, each layer is composed by different numbers of neurons. When data is fed into the network, the first layer is called input layer, this layer transforms the input data into appropriate size of matrices for next layer; the next few layers can be various in size and structures depending on the task; however, the last two layers are generally fully-connected, the function of these layers is to transform the features that are captured by previous layers into the number of classes that are predefined in the network, thus the final output is the prediction probability on the classes. Figure 1 demonstrates a typical three-layered CNN structure.

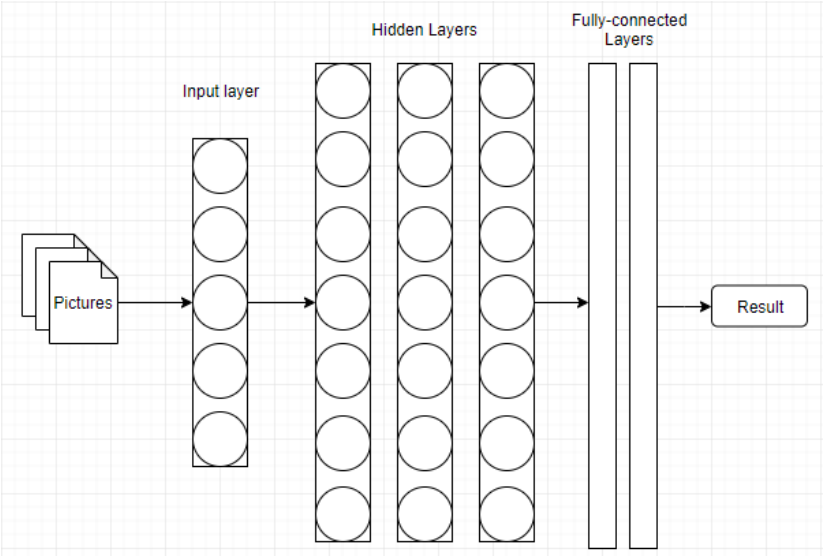


Figure 1. Typical 3-layered CNN structure.

With various structure in neural network, autoencoder is a network structure that can effectively encode the features of input data then generate new data based on the encoded information. Autoencoder is mainly used in reducing the dimension of dataset and feature extraction at beginning; it is also used to regenerate lost data in recent years.

Autoencoder has two parts: 1) encoder reduces the dimension of input data and captures the useful feature to encode; and 2) decoder takes the compressed results from encoder and decompresses them into original information. Figure 2 shows the structure and general functionality of a full autoencoder. While encoder and decoder can work separately and unsupervised, the generated result from decoder can be used to compare with the original input. The process is similar to Generative Adversary Network or GAN, which is described in the fundamental process of creating DeepFake videos.

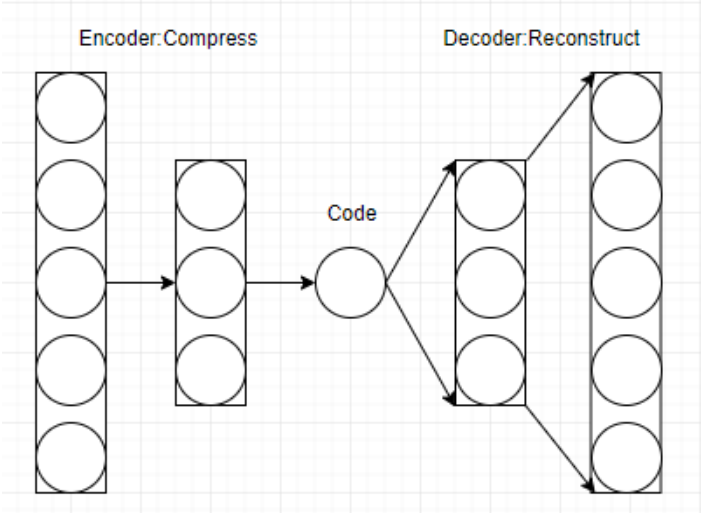


Figure 2. Autoencoder and Decoder. Notice that decoder can either work separately from encoder, or working together as GAN.

TensorFlow Hub & Transfer Learning

This section introduces TensorFlow for a widely used open-source deep learning library (Clark, 2018).

TensorFlow has become the most ubiquitous open-source deep learning library for many years (Hale, 2018). Not only TensorFlow supports the high-performance computation through its cloud service, but also it has a well-established community for the library update and maintenance. TensorFlow Hub is a library for reusable machine learning modules. A reusable machine learning module is a self-contained piece of a

TensorFlow graph, along with its weights and assets that can be reused across different tasks in a process known as transfer learning (TensorFlow, 2019). Compared to normal training process of a neural network module, a module trained by transfer learning is usually trained with a smaller dataset, other advantages such as improved generalization and increasing training speed could also be gained from transfer learning. To develop a modern image recognition module from scratch, it normally requires hundreds of GPU-hours or more (TensorFlow, 2019). By applying transfer learning on a trained module, the size of training dataset can be greatly reduced, such technique could be used to solve the classification problems with relatively small datasets. Figure 3 shows how transfer learning replaces the original layer and creates a new layer to classify new labels.

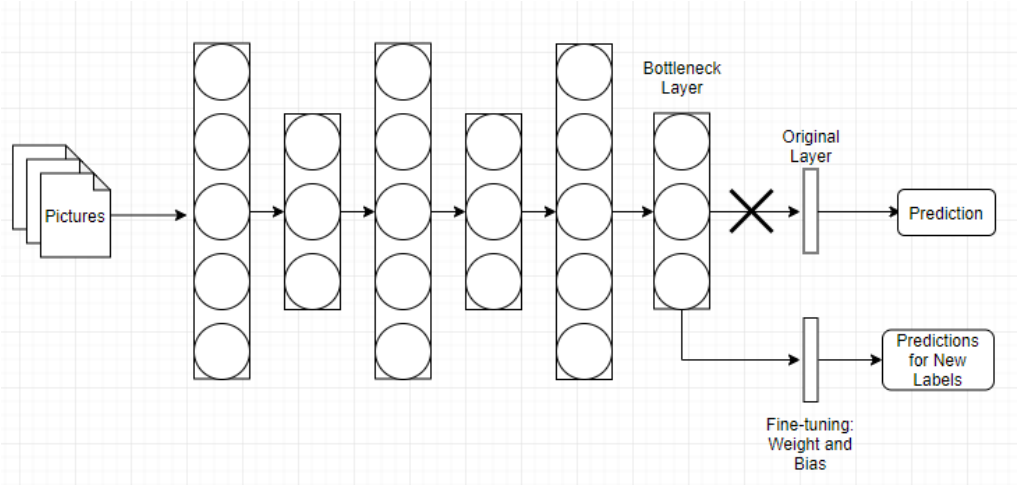
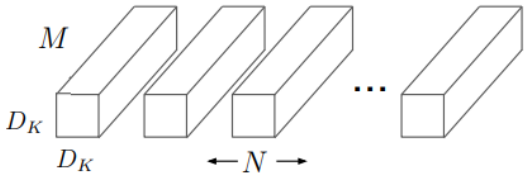


Figure 3. Transfer Leaning. The Original Layer will be replaced with a layer which its weight and bias are fine-tuned for classifying images on new labels.

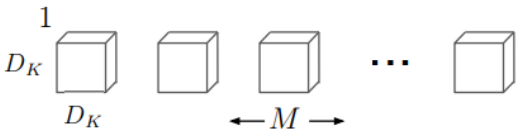
Retraining Module

This section introduces the neural networks that are retrained in our experiments. Since the main library used in this experiment is TensorFlow, we conduct our experiment with two of most prominent neural networks modules that are developed by google: MobileNetV1 and InceptionV3.

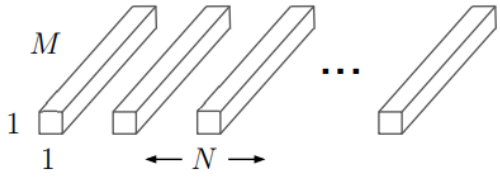
MobileNet V1. MobileNet is a light weighted neural network designed for deep learning experiment on mobile devices (Howard, 2017). While other Deep Convolutional Neural Networks focus on improving the overall accuracy, MobileNets are proposed for mobile and embedded vision applications. MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks. MobileNetV1 is a Convolutional Neural Network (CNN) that uses depthwise separable convolutions which means it could drastically decrease the number of parameters and computational cost. Figure 4 shows the structure difference between depthwise separable convolution filters and standard convolution filters.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 4. Standard Convolutional Filters vs MobileNet Depthwise Separable Convolutions. (a) Standard convolution filter parameters, where D_k represents the size of input and N represent the number of input channel; (b)(c) Depthwise separable convolution (Howard, 2017).

Inception V3. Google's Inception networks are heavy-parameter and well-engineered deep neural networks. Since Inception V1 was announced in 2014, the network structure has become even more complex, from Inception V1 with 22 layers to Inception V3 with 42 layers (Tsang, 2019); however, the deeper network structure comes with improved accuracy. The Inception V3 module was proposed with Inception V2 in December 2015. Based on the V2 module, Inception V3 factorized a 7×7 into two series connected 1×7 and 7×1 network in order to reduce the computational cost; it also introduced a new optimizer to achieve a better gradient: RMSProp. Detailed explanations will be presented in methodology section.

Remaining Chapter Description

In the next few chapters, we will first discuss the one of the newest DeepFake Apps, namely DeepFaceLab (Iperov, 2019). In order to get a good grasp on the subject, we will then explore DeepFaceLab to estimate the time and quality of this tool. Methodology used by DeepFaceLab will be introduced in the chapter 4, followed by a demonstration and experimentation in chapter 5. In the end, we will analyze the performance against DeepFaceLab and discuss our future study that could improve our proposed method. Figures and tables that appears during the experimentation will be attached in the appendix section.

CHAPTER II

Related Research

Synthesized facial reenactment

Synthesized facial reenactment has gained its popularity in recent years. Similar technology has also been used by Facebook on Facebook Camera for dynamically generating emojis based on analysis of facial features (Matas, 2017). After Face2Face proposed an approach to capture real-time facial movement and reenact to other RGB videos (Thies, 2018), other alternatives methods to reenact portrait video have also been proposed (Thies, 2018). An audio-to-lip synthesizing technique proposed by Suwajanakorn et al. (Suwajanakorn, 2017) make use of a recurrent neural network that could map the raw audio feature with mouth shapes.

GAN related forgery

Generative Adversarial Networks (GANs) are showing distinguished results on generating module based on limited information. Researchers from NVDIA have developed a style-transfer method to generate photo-realistic human facial images (Karras, 2018). Similar results can be achieved by manipulating the attributes of encoded image (Lample, 2017). An attribute-guided face generator was also developed by Lu et al. (Lu, 2017) to generate high-res facial image from low-res image. A Celebrity face generator was presented by Sharma (Sharma, 2018), this GAN based generator was trained on a celebrity face dataset called CelebA (Li, 2018), after sixth training epochs, the generator is capable to generate low-res facial images.

Face Morphing

The face morpher uses algorithms to extract feature points on original face and then maps each feature point to the target face to create a new synthesized face. This type of attack generally produces high resolution and natural looking facial expression. Seibold et al. (Seibold, 2017) proposed an approach based on CNN, their team tested their dataset on AlexNet, GoogLeNet and VGG19, and achieved around 16% to 10% FRR and 2.2% to 1.8% FAR.

Forgery video detection

Facial reenactment. Cozzolino et al. (Cozzolino, 2017) applied a CNN to a class of residual-based descriptors to extract the image details, and achieved 79.8% accuracy on easy compressed videos and 55.77% accuracy on Strong compressed FaceForensics (Rössler, 2018) videos. Bayar and Stamm (Bayar, 2016) demonstrated an 8 layers CNN-based network: a constrained convolutional layer, 2 additional convolutional layers with 2 Max-pooling layers and 3 fully-connected layers. Their method has achieved 86.10% accuracy on easy compressed videos and 73.63% accuracy on Strong compressed videos. Rahmouni et al. (Rahmouni, 2017) trained a CNN with custom pooling layer to optimize the feature extraction algorithms. By local estimates of class probabilities to predict the label of image, they achieved 88.5% testing accuracy on easy compressed videos and 61.5% testing accuracy on Strong compressed videos on FaceForensics dataset. Raghavendra et al. (Raghavendra, 2017) used two fully connected CNN (VGG19 and AlexNet) to detect the feature, followed by a probabilistic collaborative Representation Classifier (P-CRC) to detect the morphed images. They achieved 93.5% accuracy on

easy compressed videos and 82.13% accuracy on Strong compressed videos on FaceForensics dataset.

DeepFake. Guera et al. (Guera, 2018) applied a CNN network for feature extraction and concatenated with a convolutional LSTM network for sequence processing. They tested on DeepFake video clips and 300 videos from HOHA dataset. Their method has achieved 97% testing accuracy. Li and Lyu (Li, 2018) tested their DeepFake dataset on VGG16, resNet50, resNet101 and resNet152 to capture the artifacts within the video. Their AUC reached around 84.5 to 99%.

CHAPTER III

DeepFaceLab & Workflows

Main concept

In general, face swapping in DeepFake will first extract the important features from original face, encode these features into a feature map, then use trained decoder to generate destination face thus accomplish the face swapping in pictures and videos. The process is referred to as autoencoder. DeepFaceLab is the latest development on related face swapping apps. DeepFaceLab also brings several new features compared to its predecessors FakeApp, including new training models, and training progress preview. It also allows user to utilize CPU to train their model (Hui, 2018).

For extracting faces from video data, DeepFaceLab makes use of MTCNN extractor (cyberfire, 2017). Although MTCNN can capture more false positives during the extraction compared to DLIBCNN (King, 2009), the DLIBCNN produces less jittered aligned faces when video frame becomes unstable. The difference between these two facial extractors can be seen in Figure 5.

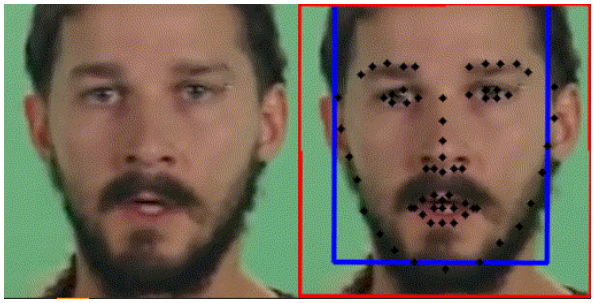


Figure 5. Comparison of dlib and MTCNN facial extraction. Left: dlib, Right: MTCNN (Iperov, 2019).

To capture the facial frame in specific angle or obstructed by other objects, DeepFaceLab provides a GUI extractor which allows user to extract face from a specific

frame and landmark face manually; this function also allows full manual extraction from source film in order to get the most ideal results.

Training Models for DeepFakes

As we previously introduced, DeepFake Apps mainly utilize autoencoder as the basis for producing the face masks. There are various modules in DeepFaceLab, depending on the quality and resource allocated for create a DeepFake video, here are some of the modules currently useable in DeepFaceLab:

H64 is a model designed for GPU with minimum 2GB memory and this module produces relatively low resolution (64 * 64) face. This module is also used by the FakeApp and re-implemented in DeepFaceLab with TensorFlow 1.8 SSIM loss function, separated mask decoder and improved converter mask function. For videos containing many straight face-on scenes H64 is a low-cost training model. H128 is variation of H64, it provides the highest resolution for front face generation; however, user must have at least 3GB VRAM in order to use this module.

DF produces a full-face model, it is also good for side-face generation. The result of this module covers multiple angles of faces thus it is unlikely to find noticeable artifacts on a fully trained model; The disadvantage of this module requests at least 5GB VRAM and it demands the source face shapes and light condition.

Similar to DF, LIAEF128 produces a lower quality module in order to partially fix dissimilar face shape and behaves less aggressive while morphing facial features. It requires at least 5GB VRAM and has a problem with tracking eye blinking.

SAE is a “Face Morpher”. Compared to other generator such as LIAEF128, SAE trainer will morph the original face to match the target video facial style. The trainer can

produce good resolution video, however, SAE also have a higher risk to create collapsed facial frames.

Convert Options

One of the most important operations to identify the mask is to locate the possible artifact boarder around the model's face. Such boarder could be either a visible gap between the mask edge and model or a thin line that could be easily omitted by human eye. Luckily, this boarder region is unavoidable due to the nature of the DeepFake; however, most DeepFake apps will use various convert operations to hide this transition area. The details of convert operations are given below (Iperov, 2019):

Blurring. Blurring refers to the operations that use to blur the boarder of the cropped face mask. Since the DeepFaceLab uses the predicted mask as defined broader of as predicted source face, this broader will be visible when the blurring value is set to negative. On the other hand, setting blurring value to high positive will derogate the overall quality and increase the training time.

Erosion. Erosion is a parameter that essentially increases the area of the source face when converting onto the destination face. The parameter could range from -100 to +100, -100 represents the dilation which destination face will be completely covered by the source face. This term was originally used in morphological image processing as a matrix operation on binary images to reduce the boundaries of regions of foreground pixels. Outcomes of erosion operation will enlarge the holes and shrink the original shape (Fisher, 2019).

Seamless Erosion. Seamless Erosion will perform same operation as typical erosion while maintaining a seamless fashion. This operation will increase the difficulty

to detect the visible artifacts but it also increases the training time and downgrades the overall quality of the video.

Hist-match Threshold. Hist-match Threshold also refers to as histogram match threshold; this parameter is used to adjust the lights that could be reflected on the model mask. Default value in DeepFaceLab is set to the maximum, this will result in the unnatural brightness appearing on the face mask.

Face Scale. Face Scale is a parameter to adjust the scale of source face mask proportionally towards the center of the model face. Depending on the size difference between the source face and destination face, this parameter could be used to adjust the face mask size to suit the model face.

Transfer Color. Transfer Color from destination image to source face will convert the skin color of destination model to the source face mask; however, this transformation is not without flaws, when trained with dataset only consisting of monolithic skin tone, color transfer will have high probability to fail.

Degrade Color Power. The reason behind this operation is deliberately downgrading the quality of converted image to belie the details such as the boundary of the mask and lighting. Such action could be countered by other methods that could improve the original video quality and analyze the refined video frame by frame.

CHAPTER IV

Methodology

Breaching the DeepFakes videos

Generally, DeepFake videos are short in length as well as low on resolution. Therefore, we can logically infer that if a short video contains more blurry frames, especially it occurs around facial area, indicating a high possibility that this video is a forgery video. Nevertheless, for research purpose, we must delve deeper into the creation of DeepFake videos in order to understand the cause of these flaws. Figure 6 shows the blurry around facial area after DeepFake masked the original face.

Figure 6. Original face (Left) vs Compressed DeepFake product (Right). (Iperov, 2019).

Common flaws that DeepFakes and GAN generated image possess

There are three key aspects that needs to be considered during the creation of DeepFake videos: First will be the overall quality of the video. For example, the result of the forgery must possess relatively high resolution, introduce less artifacts and more natural facial expression that fit the video context; Second, speed for creating the video should be reasonable. The training speed is predominantly affected by the size of VRAM, as well as the structure of deep neural network and the training data size; Third, the duration of the video. Creating longer clips require more training data and time. Knowing these aspects can help us better understanding the creation step of DeepFake video and discovering the weak points.

Most fraudulent videos generated by DeepFake apps possess certain flaws. Such as visible artifacts, distinct resolution around the facial area and sudden change of color when module have abrupt face movement. These flaws are caused by the DeepFake learning algorithm when the training dataset is insufficient to cover all the angels or lighting of the face. On the other hand, most users of the DeepFake application do not possess enough computing resource or the time to produce a well-trained a module. Therefore, it is very challenging to create a flawless DeepFake video.

In DeepFaceLab, a "predicted mask" will be created for destination face to match the actual facial area that needs to be covered; however, this process will also create visible artifacts on the mask's edge. There are three methods used by DeepFaceLab to eliminate the artifacts: (1) applying Gaussian filters to mask the boundary area; (2) expanding the masking region (e.g. Include forehead and jaws); and (3) manually adjusting the mask shape. Directly applying mask could produce fully-covered face mask for destination face; such practice could also introduce double chain and blurry edges. It causes most DeepFake videos produced with relatively low resolution, especially around the facial area, compared to other synthesized videos.

Aside from the resolution, skin color and environmental lighting reflected on the face are also important factors when detecting DeepFake video. In order to create a perfect mask for the target face model, selecting the target source video is crucial. Since the source model's head movement will directly affect the size of training data. Therefore, it is essential for a target to have less head movement while maintaining the environment lighting consistent. As a result, we propose to use transfer learning on some

existing Convolutional Neural Networks (CNN) to detect graphical inconsistencies within each frame.

Transfer learning & Approach

When applying transfer learning on an existing module, there are two parts in affecting the training efficiency: (1) The original weight and bias from pre-trained network; (2) Retrain the network with new dataset. Therefore, focusing on which process will determine the effectiveness of retrained module and its classification accuracy (Li, 2018).

On the one hand, aggressively retaining the whole network will not always yield the most effective network. This approach is appropriate when the retraining data size is larger than the pre-trained data size; or the retraining data have little relevance with pre-trained data. With this setting, the hyperparameters from pre-trained modules will only serve as “initializer”; the training algorithm will need to be hyper-tuned intensively to achieve higher accuracy, thus it will greatly increase the overall training time as well as the risk of overfitting.

On the other hand, by using the training parameters from pre-trained module, a passive approach only focuses on constructing the classification layer: SoftMax and Fully-connected layer. Although the accuracy in this approach will be constrained by the initial parameter setting of pre-trained network, the total training time can be greatly reduced. As the data size for pre-trained module is decreased, the constrains on hyperparameters are also eased. Inception V3 is the state-of-the-art neural network trained on ImageNet dataset; ImageNet is a dataset contain more than 14 million images of various objects. Therefore, the feature information (or Image feature vectors) gathered

from the ImageNet data are also proved to be useful when identifying the artifacts within each frame.

Advantage of Inception V3

Avoid representation bottleneck. Representation bottleneck means after a feature map passing through a pooling layer, the feature map dimension will be reduced, thus increase the possibility of feature loss; however, the important features should be always kept within the feature map, and Inception V3 suggests a pooling structure to achieve this goal. This could provide learning stability and maximize our transfer learning efficiency when we retraining on different sets of data.

Reduced kernel size and increasing the width of the network. As shown in Figure 7, the Inception layer performs convolution on the input with different sizes of kernels then concatenates the results together, which could drastically improve the performance on capturing local features across the image.

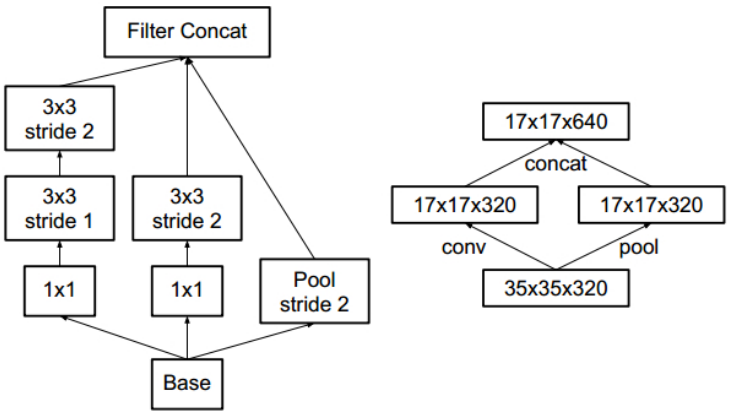


Figure 7. Parallel filter structure of Inception V3. It is both cheap in computation and avoid representation bottleneck (Szegedy, 2016).

Reduce the dimension of the filters. The input image size for Inception V3 is 299 * 299. During the inception layer, a 7 by 7 layer will be replaced with three 3 by 3 layers, a 5 by 5 layer will be replaced with two 3 by 3 layers; furthermore, the n by n

convolution filter will be factorized into n by 1 and 1 by n filters to reduce the computational cost. Figure 8 shows the structure of inception layer (Szegedy, 2016).

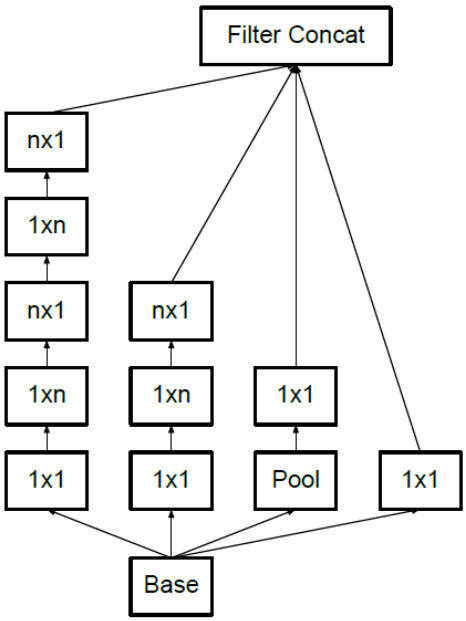


Figure 8. Filter factorization for Inception filter (Szegedy, 2016).

Advantage of MobileNet V1

Reduce the computational cost. Unlike the normal convolution, the Depthwise separable convolution will apply a single kernel to a single input channel. For example, a colored picture will contain 3 input channels, for each input channel, a 3 by 3 filter will be assigned to generate feature map; all three filters will be executed in a concurrent manner. After Depthwise convolution, the pointwise convolution will process all the feature map information obtained from previous step on a 1 by 1 filter. Assume the K is representing the Depthwise convolutional kernel size and F is the size of each filer, M will be the number of input channel and N will be the number of filters, then we have (Howard, 2017):

$$D_K * D_K * M * D_F * D_F + N * M * D_F * D_F$$

For VGG16, the computational cost for a 224 by 224, 3 channel input will cost:

$$3 * 3 * 128 * 64 * 112 * 112 = 924{,}844{,}032$$

However, the computational cost is much lower when using Depthwise separable:

$$3 * 3 * 64 * 112 * 112 + 128 * 64 * 112 * 112 = 109{,}985{,}792$$

Therefore, the cost ratio between MobileNet and traditional 3-dimentional convolution is (GHB, 2017):

$$\text{MobileNet} : \text{Traditional} \approx 1 : 9$$

Image Feature Vectors

Image Feature Vector (IFV) or transfer-value is the layer before the classification layer. This layer contains meaningful summary of the images. Different networks will create different IFV for each input image. There are total 2048 transfer-values for Inception V3 and 1001 for MobileNet_V1_0.5_224. Figure 9 shows the feature vector file in text format for an image.

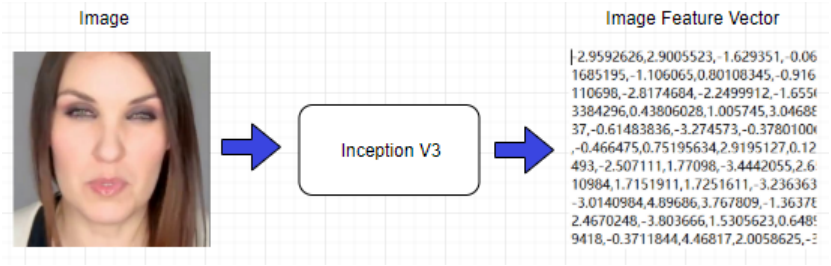


Figure 9. Image feature Vector creation process. Each image will have a *.txt* file with 2048 either positive or negative value representing the extracted features.

Creating Image Feature Vectors

The objective of transfer learning is to recreate final layer based on new classification labels. The weight and bias of these newly created layers will be determined by the back-propagation results. The bottleneck value, also referred as "Image Feature Vector", is the classification layer just before the final output layer

(TensorFlow Hub, n.d.). The bottleneck layer will output a set of values that could be used by classification layer to distinguish the new labels. Since each image will be reused during training, the files containing bottleneck values will be written on the disk to avoid repeat recalculation and for future use.

Because of the bottleneck files contains the feature map of each image, and the utilization of bottleneck files are essential while constructing new softmax and FC layer, we will discuss these functions in detail:

During the creation of bottleneck layers, there are several functions would affect the process, a progress flowchart can be seen in Figure 10:

create_model_graph(). is loaded from TensorFlow Hub to create graph which contain three parameters:

Resized_input_tensor. Represent the inputs to create graph.

Bottleneck_tensor. Represents the output bottleneck value.

Wants_quantization. A parameter used to quantify bottleneck file.

Run_bottleneck_on_image(). Use to feed image data into the create_module_graph() to generate bottleneck_values. The propose of this function is to extract the image features.

Get_or_create_bottleneck(). This function will take the bottleneck_values generated from previous layer and save it as bottleneck_data on the local disk.

Add_final_retrain_ops(). This is the final layer that need to be changed. This layer will redefine the weight, bias, passing softmax layer and output final_tensor based on bottleneck_tensor. The ground_truth value represents the computational loss on output label.

Save_weight(), Evaluate.py. At the end of the creation process, the weight and bias of classification layer will be plotted and saved. The retrained module will be also be saved as *.pb* file for reuse.

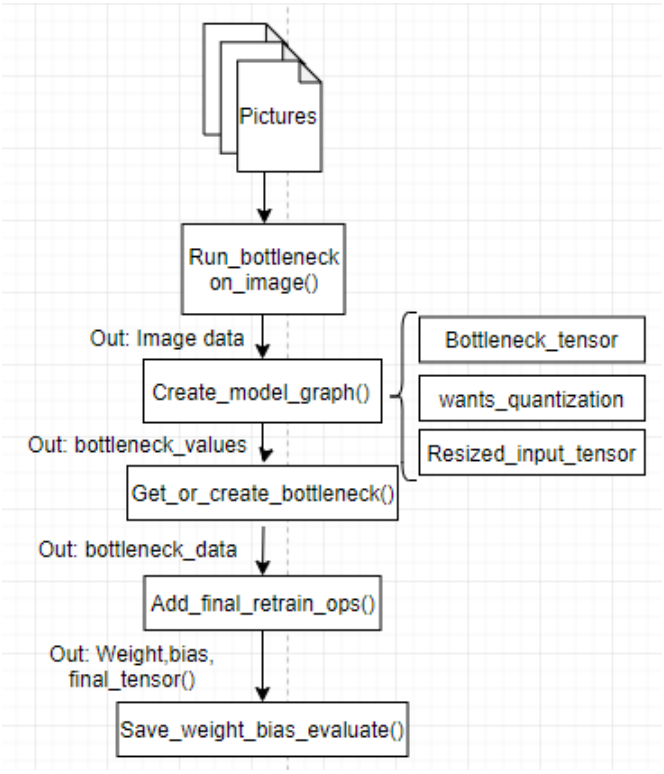


Figure 10. Creation process for Image Feature Vector.

CHAPTER V

Experiment

The experimentation will be divided into two major sections for retraining two different neural networks. Before starting the retraining process, the image data will go through preprocessing stage, the image mainly consists of two categories, image that labeled as real and images that labeled as fake. The difference between most of these images are very difficult to be distinguished by naked eye due to the low resolution and hidden artifacts. The objective of this experiment is to analyze the performance difference of retrained neural network on detecting the forgery videos.

Dataset

Our dataset contains three different sets of images from FaceForensics (Rössler, 2018) image set. These images are cropped face sets from multiple videos and possess various size and resolutions. For example, set A is a lightweight set, this set includes 1000 images in total, the objective of this set is to test the neural network's performance when trained by small sets of images. The images are evenly divided into 50 percent of fake images and 50 percent of real images in all three sets, they are placed in separate directories with labels. The retraining algorithm will randomly select images from these directories and further divide them into training, validation and testing set. More information regarding this process will be discussed in the following section. Set B contains 5000 images with same attributes and Set C will have 25000 images. Furthermore, the name of misclassified images will be printed out after the testing is completed, these images will be future studied for improving the future training session.

A separated dataset that consists of extracted face set from DeepFake generated video and real video will be used to retrain both modules. Each frame in the video will be extract as PNG file by 1 frame per second. To make sure the training dataset will only contain the clear facial image from target actor, frames that involved multiple faces as well as the facial obstructions will be deleted in this process. Next, the frames will be passed into the facial recognition module (Kazemi, 2014) for facial extraction. To prevent the false-positive results (non-facial captures) generated by the extractors, it is beneficial to delete the outliers before fed them into the module. In addition, the DeepFake dataset is gathered from multiple trending videos amongst DeepFake community. More specific specs regarding to the dataset is shown in Table 1.

Table 1

Detailed Dataset Information

	Size	Cropped Facial Image size	Number of videos contained	Real to Fake Ratios
Set A	1000	406*406 to 420*420	1	1:1
Set B	5000	180*180 to 430*430	6	1:1
Set C	25000	150*150 to 750*750	29	1:1
Elon Set (Deep Fake)	6659	256*256	Fake:4 / Real: 2	35:31

	Size	Cropped Facial Image size	Number of videos contained	Real to Fake Ratios
Nic Set (Deep Fake)	7894	256*256	Fake:3 / Real:2	33:46

The real to fake ratio in Elon set is 3537:3122; and the Nic set is 3294:4600.

Hyperparameters

There are multiple parameters we can adjust to control the retraining process. Learning rate (LR) controls the speed to finding local-minimum by using the gradient descent function. In other word, higher learning rate can speed up the retraining process, at the cost of accuracy; however, set LR too low may prolong the retraining time and increasing the possibility of getting trapped in local minimum. Thus, choosing this value carefully is one of the most important aspect of retaining a successful network.

For MobileNetV1 4 different "alpha value" could be selected. Ranging from $\in \{1, 0.75, 0.5, 0.25\}$, and resolutions from $\{224, 192, 160, 128\}$ (Howard, 2017). Choosing the appropriate "alpha value" and resolution for retraining module could strike the balance between the efficiency and module complexity. For this experimentation, we set alpha = 0.5 and resolution = 224 to get an equilibrium between training time and accuracy. The resulting image feature vector (bottleneck file) size will be 1001.

For Inception V3, the input image size is fixed to 299 by 299. The input channel will be standard 3 channels RGB image. The standard deviation and mean for the network will be set to default. The size of image feature vector will be a 64 by 32 feature map.

Training steps will set the limit for module to train. Increasing the number of training steps will increase the overall training time as well as the test accuracy; however,

the rate of improvement will hit the celling during the retraining, and test accuracy may decrease due to overfitting. Therefore, setting training step to an appropriate value is also essential to retrain a module. To use the retrain script in python, Figure 11 provides some most used parameters.

```
python -m scripts.retrain
        \--architecture="mobilenet_0.50_224"
        \--learning_rate=0.005
        \--image_dir=MobileNet_nic_7300/nic_img
        \--model_dir=MobileNet_nic_7300/models/"mobilenet_0.50_224"
        \--summaries_dir=inceptionV3_nic_7300/training_summaries/LR_0.005
        \--output_graph=MobileNet_nic_7300/retrained_graph.pb
        \--output_labels=MobileNet_nic_7300/retrained_labels.txt
        \--how_many_training_steps=5000
        \--print_misclassified_test_images
        \--bottleneck_dir=MobileNet_nic_7300/bottlenecks
```

Figure 11. A Typical Parameter Set Used When Training. To apply Transfer Learning, we must specify the training directory, bottleneck directory, as well as the value of hyperparameters. In this case, we set Learning Rate to 0.005 (half of default value) and training steps to 5000.

Retraining

In this experiment, the retraining process will have 5000 training steps, ten images will be randomly selected from the training set. The bottleneck files (image feature vector) of selected images will be feed into the classification layer to get the prediction. Furthermore, the comparison results will be back-propagated to the classification layer and update the layers bias and weight, thus refine the next prediction accuracy.

The FaceForensics Lab (Rössler, 2018) dataset will be divided into three different sets: a set contains 1000 images, 5000 images and 25000 images. The MobileNet and InceptionV3 will be trained on each set separately, results and performance will be presented in the next chapter.

CHAPTER VI

Results & Discussions

In this section we provide details on the performance of our retrained modules. Table 2 gives an overview of our experiment. From Figure 12 to 23 shows the accuracy and cross entropy of MobileNetV1 and Inception V3, trained by FaceForensics data. Figure 24 to 31 shows both neural networks' performance when trained on DeepFake dataset. The result of training set is represented by the orange line and validation set is represented by the blue line. During the experimentation, the learning rate was set to 0.005 and the training steps was set to 5000 to demonstrate the comparability. In each step, 10 images will be randomly selected from the training set and calculate for prediction; image in validation set is also being used to avoid overfitting. After 5000 steps, the training accuracy and cross entropy for this module will be plotted on TensorBoard. The x-axis marks the total training steps have taken, measuring from 0 to 5000; the y-axis shows the percentage of module accuracy or the loss on cross entropy, measuring from 0 to 1.

A summarized training time and hyperparameters is presented in the Table 2:

Table 2

Overall Performance and Parameters at Step=4999, Learning Rate=0.005

	Training Accuracy	Validation Accuracy	Test Accuracy	Training Cross Entropy	Training time (mins)	Test Sample Size
Mob_1000	100%	97%	96.8%	0.0123	2:40	93

	Training Accuracy	Validation Accuracy	Test Accuracy	Training Cross Entropy	Training time (mins)	Test Sample Size
Mob_5000	100%	97%	96.8%	0.0346	6:15	525
Mob_25000	85%	91%	88.8%	0.5467	40:32	2484
InV3_1000	99%	98%	96.3%	0.0376	2:42	134
InV3_5000	99%	98.5%	98.1%	0.1076	5:58	523
InV3_25000	95%	93%	92.7%	0.2459	50:12	2513
Mob_Nic	99%	100%	100%	0.0037	8:45	741
Mob_Elon	100%	100%	100%	0.0095	8:50	695
InV3_Nic	99.8%	99.7%	100%	0.0163	38:35	825
InV3_Elon	99.7%	99.8%	100%	0.0206	33:12	688

Note. Nic dataset contains total 7894 images; Elon dataset contains 6659 images

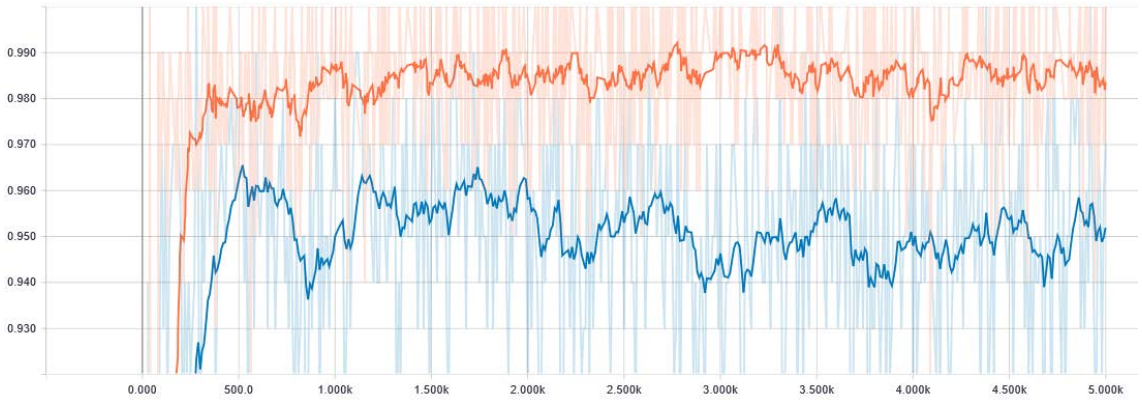


Figure 12. MobileNet V1 Accuracy on 1000 FaceForensics Images. Final train accuracy = 100.0%, validation accuracy = 97.0%, final test accuracy = 96.8% (N=93)

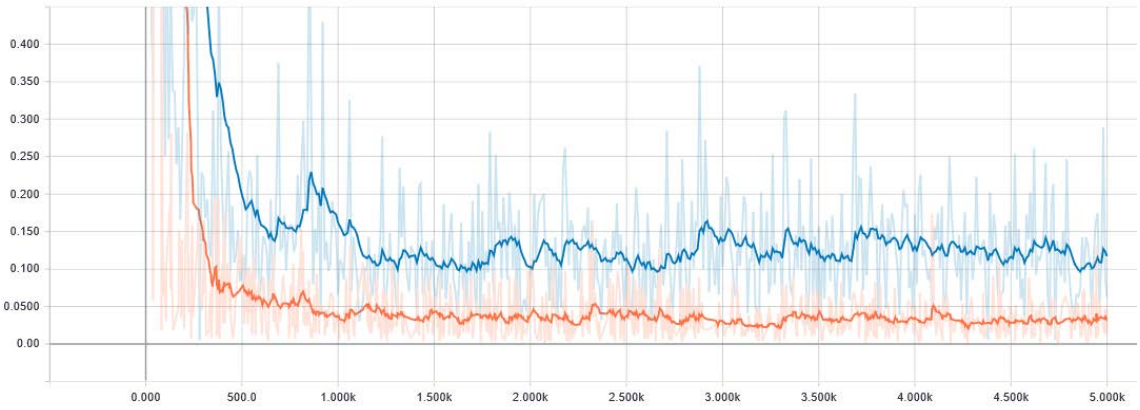


Figure 13. MobileNet V1 Cross Entropy on 1000 FaceForensics Image. Cross entropy = 0.0122 at step 4999.

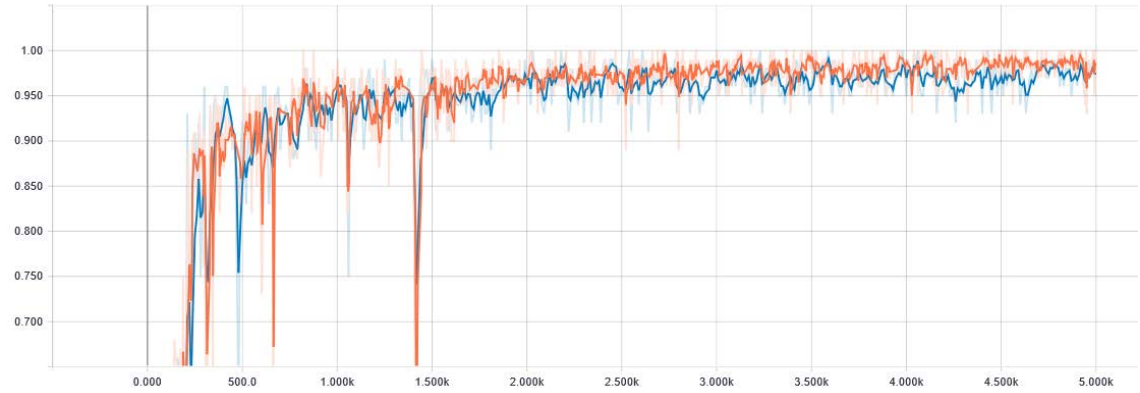


Figure 14. MobileNet V1 Accuracy on 5000 FaceForensics Images. Final train accuracy = 100.0%, validation accuracy = 97.0% (N=100), final test accuracy = 96.8% (N=525)

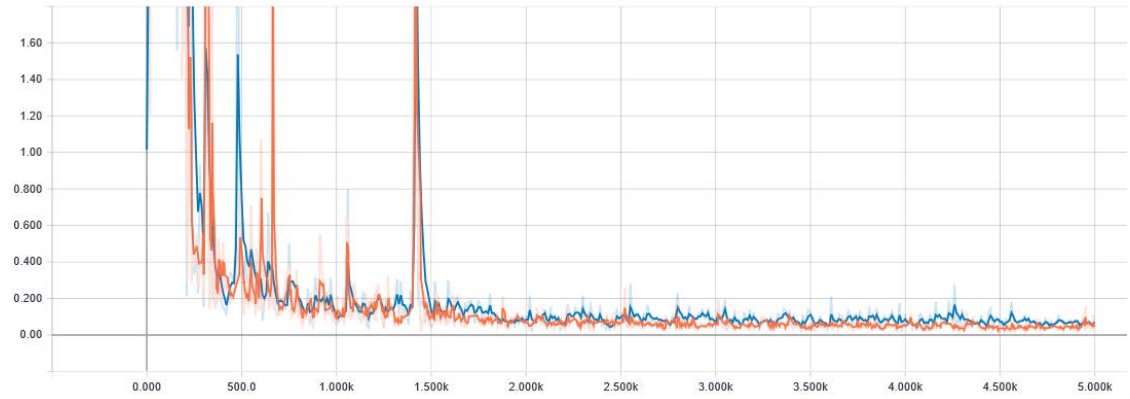


Figure 15. MobileNet V1 Cross Entropy on 5000 FaceForensics Images. Cross entropy = 0.0346 at step 4999.

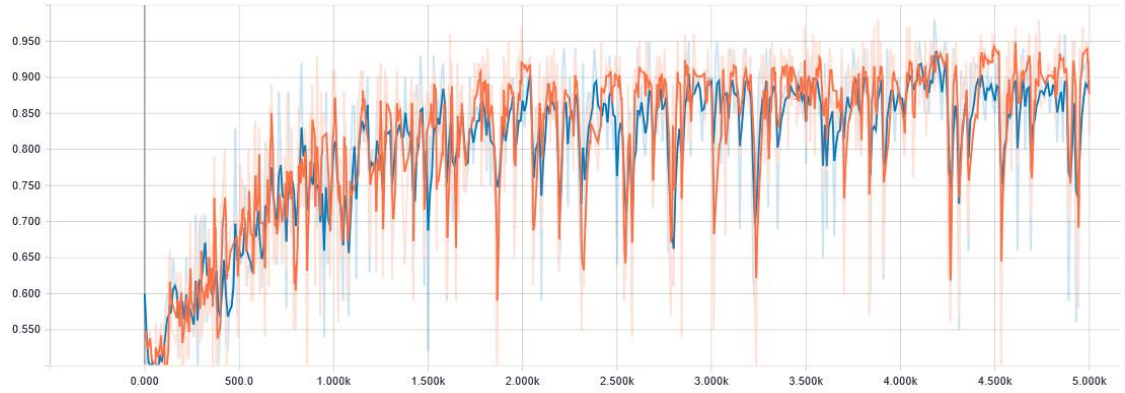


Figure 16. MobileNet V1 Accuracy on 25000 FaceForensics Images. Train accuracy = 85.0%, validation accuracy = 91.0% (N=100), final test accuracy = 88.8 % (N = 2484).

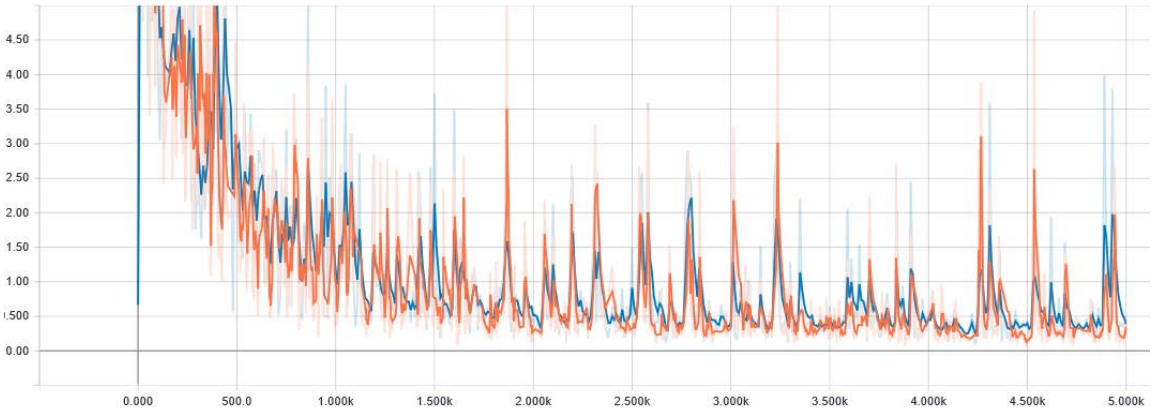


Figure 17. MobileNet V1 Cross Entropy on 25000 FaceForensics Images. Cross entropy = 0.5466 at step 4999.

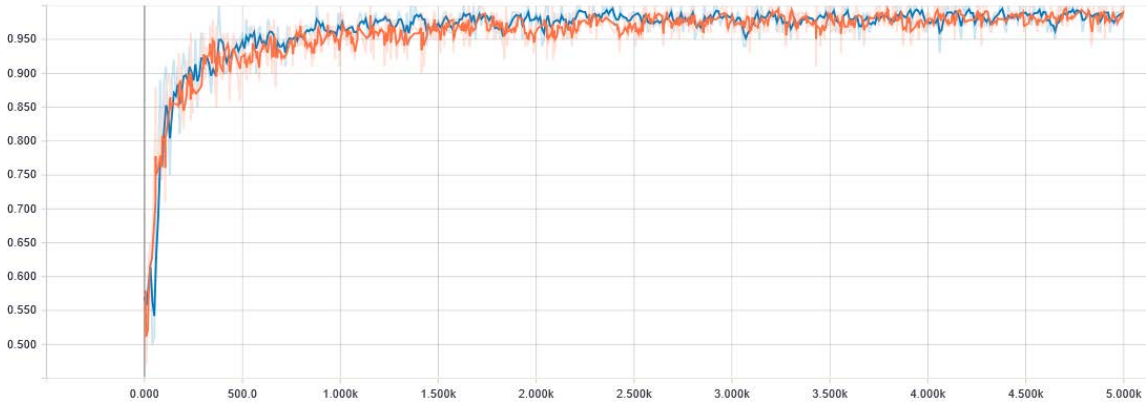


Figure 18. InceptionV3 Accuracy on 1000 FaceForensics Images. Train accuracy = 99.0%, validation accuracy = 98.0% (N=100), final test accuracy = 96.3 % (N = 134).

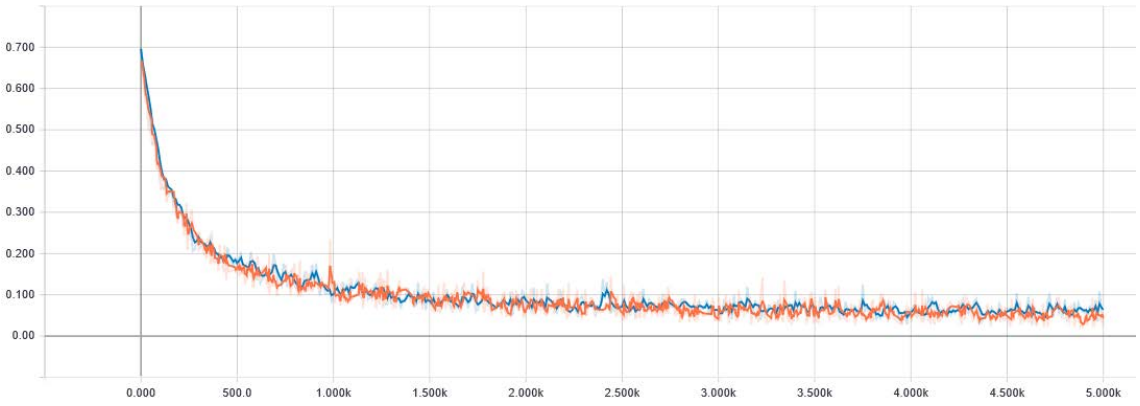


Figure 19. InceptionV3 Cross Entropy on 1000 FaceForensics Images. Cross entropy = 0.0376 at step 4999.

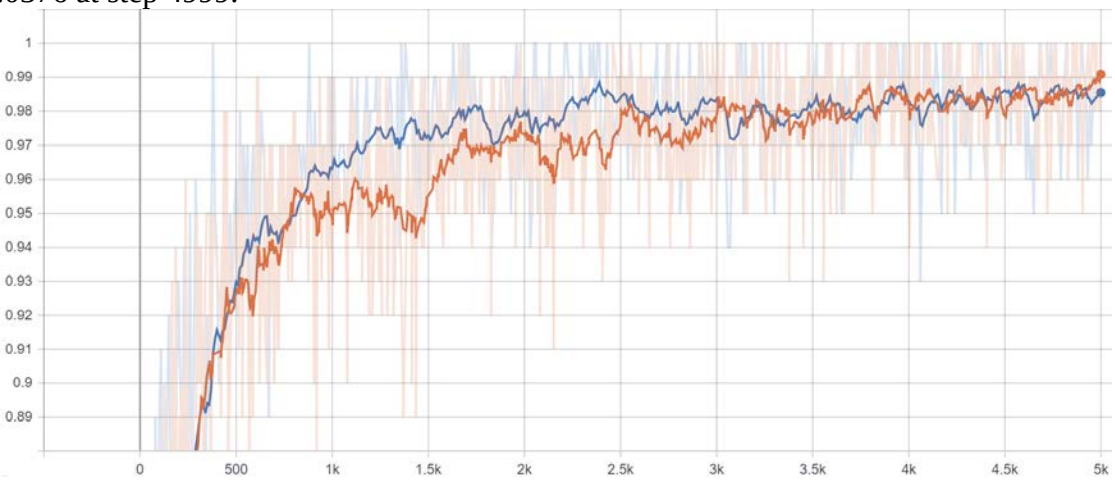


Figure 20. InceptionV3 Accuracy on 5000 FaceForensics Images. Train accuracy = 99.0%, validation accuracy = 98.5% (N=100), final test accuracy = 98.1 % (N = 523).

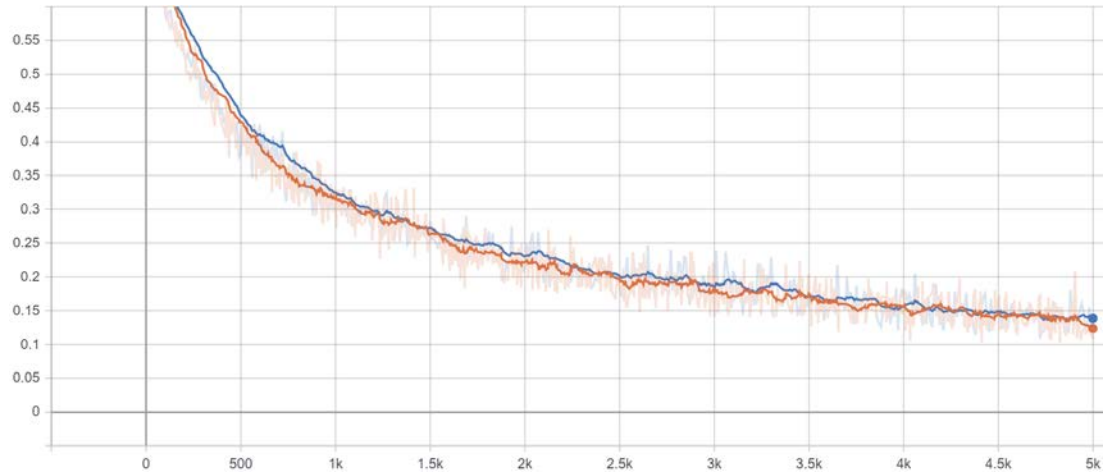


Figure 21. InceptionV3 Cross Entropy on 5000 FaceForensics Images. Cross entropy = 0.1076at step 4999.

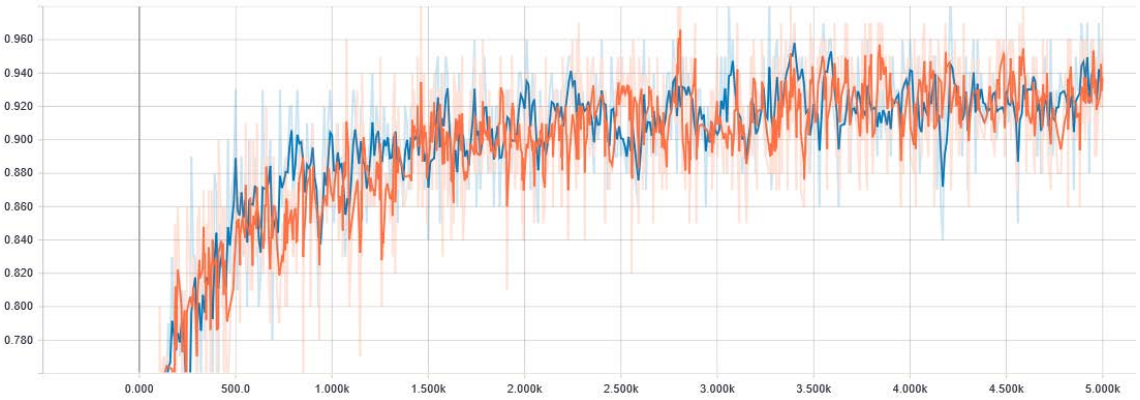


Figure 22. InceptionV3 Accuracy on 25000 FaceForensics Images. Train accuracy = 95.0%, validation accuracy = 93.0% (N=100), final test accuracy = 92.7 % (N = 2513).

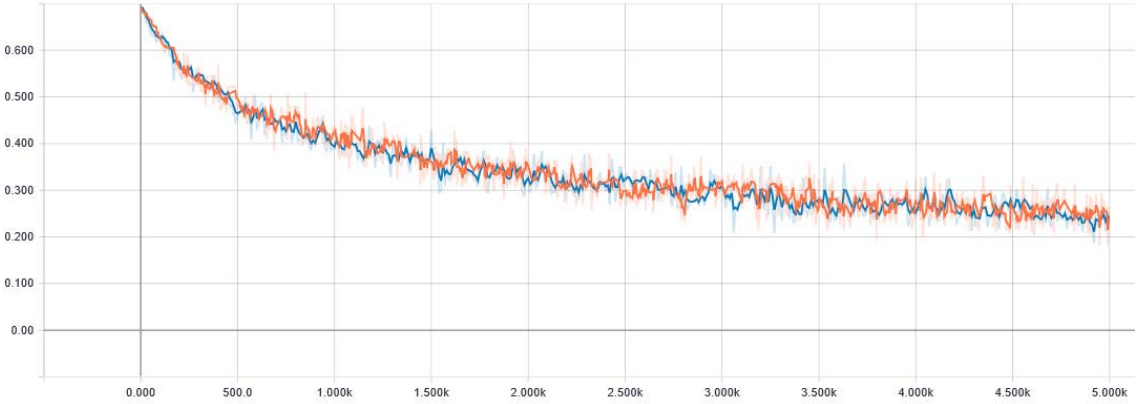


Figure 23. InceptionV3 Cross Entropy on 25000 FaceForensics Images. Cross entropy = 0.246 at step 4999.

Figure 24 to 31 shows the performance of Inception V3 and MobileNet V1 when trained on DeepFake image set.

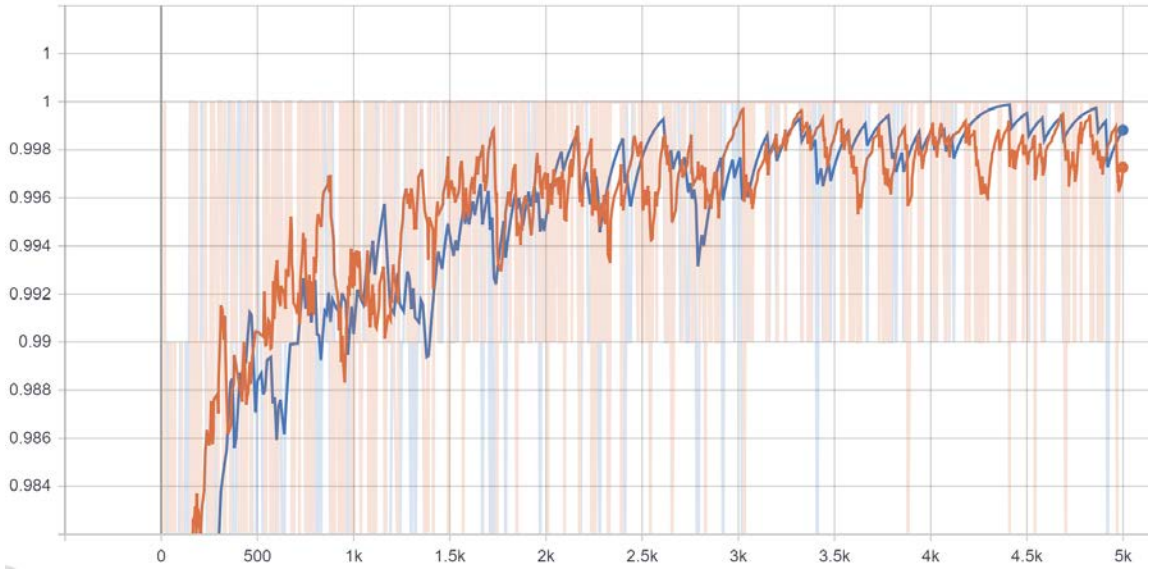


Figure 24. Inception V3 Accuracy on DeepFake Elon test set. Train accuracy = 99.7%, validation accuracy = 99.9% (N=100), final test accuracy = 100 % (N = 688).

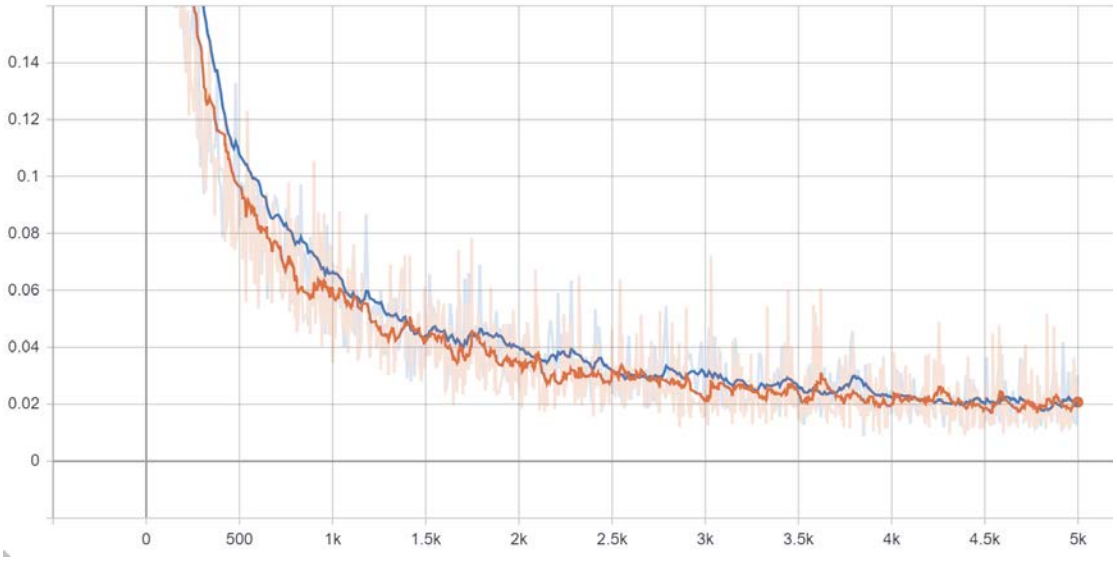


Figure 25. Inception V3 Cross Entropy on DeepFake Elon test set. Cross entropy = 0.0206 at step 4999.

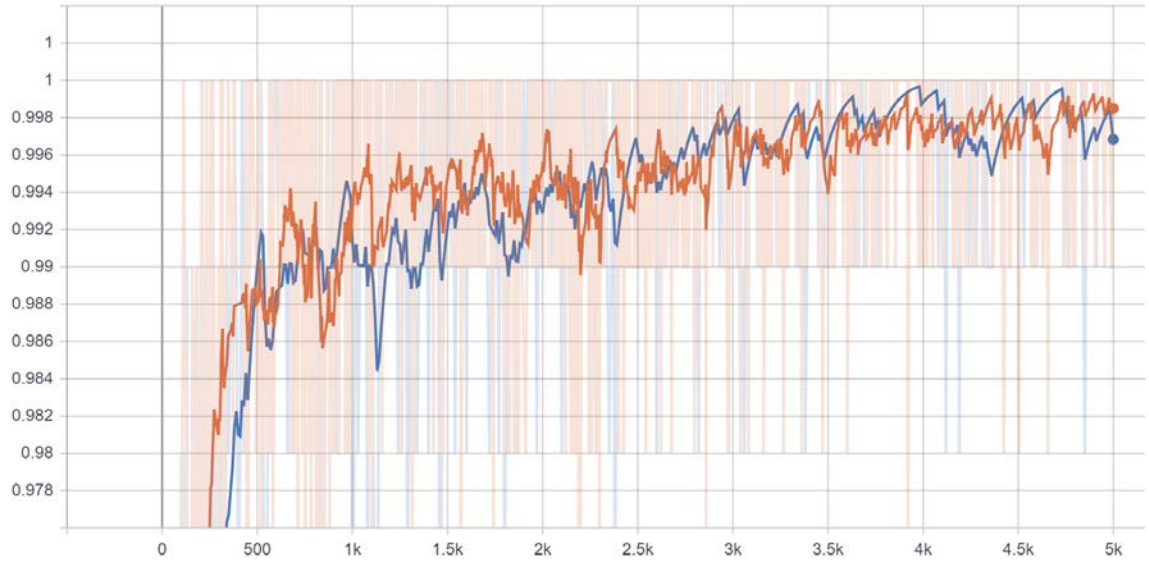


Figure 26. Inception V3 Accuracy on DeepFake Nic Cage test set. Train accuracy = 99.8%, validation accuracy = 99.7% (N=100), final test accuracy = 100 % (N = 825)

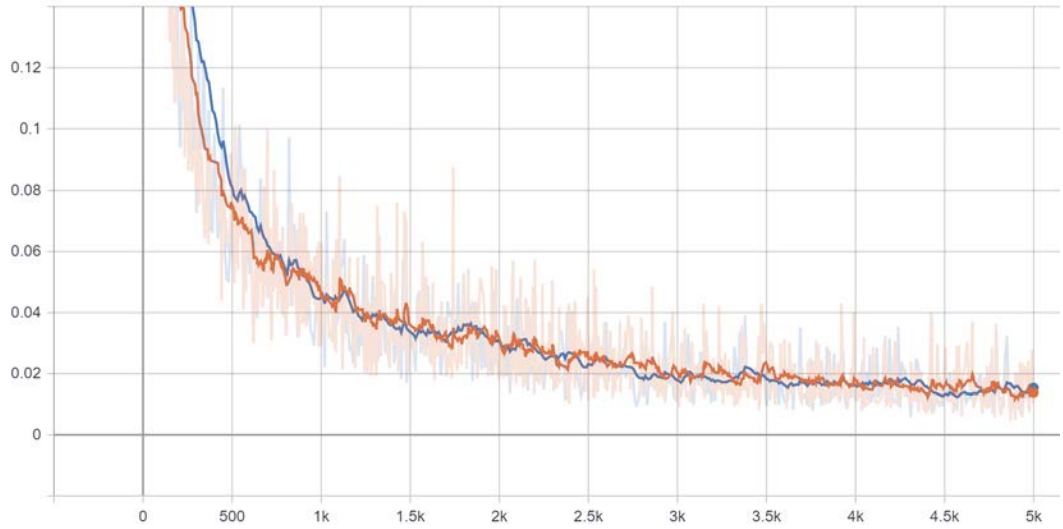


Figure 27. Inception V3 Cross Entropy on DeepFake Nic Cage test set. Cross entropy = 0.0163 at step 4999.

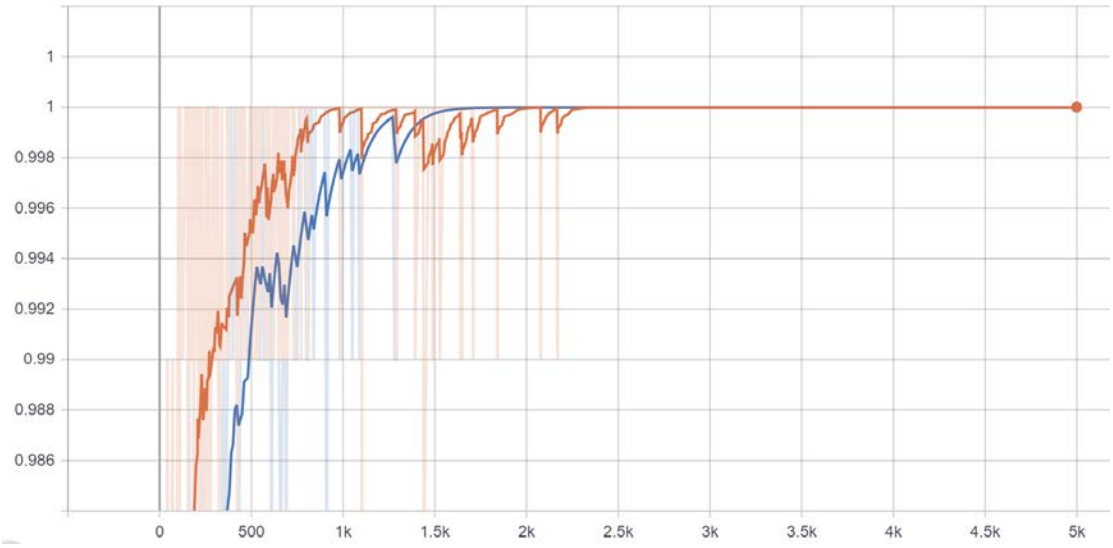


Figure 28. MobileNet Accuracy on DeepFake Elon test set. Train accuracy = 100.0%, validation accuracy = 100.0% (N=100), final test accuracy = 100.0 % (N = 695).

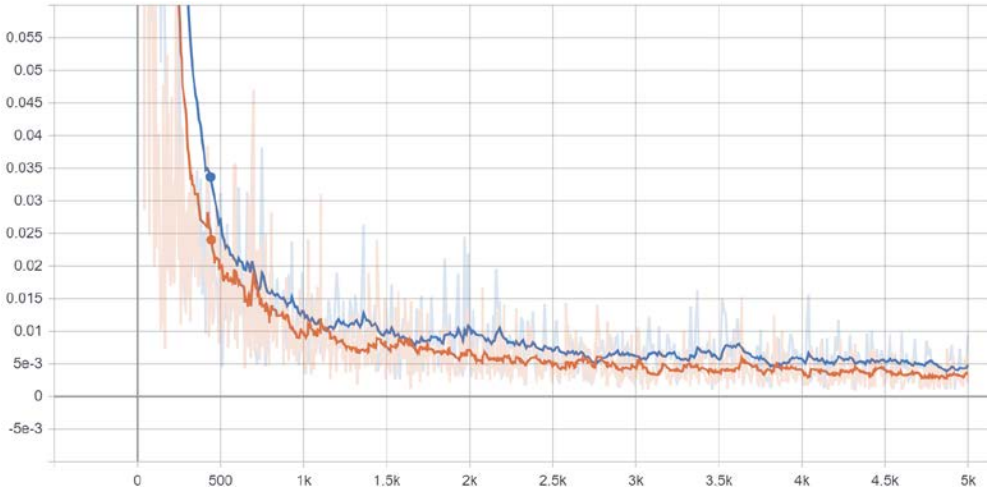


Figure 29. MobileNet Cross Entropy on DeepFake Elon test set. Train cross entropy = 0.0034; validation cross entropy = 0.0095.

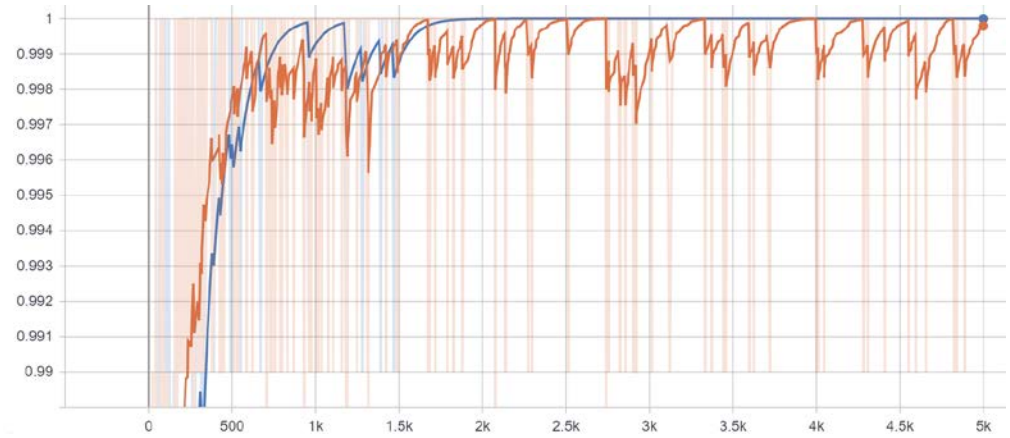


Figure 30. MobileNet Accuracy on DeepFake Nic Cage test set. Train accuracy = 99.0%, validation accuracy = 100.0% (N=100), final test accuracy = 100.0 % (N=741)

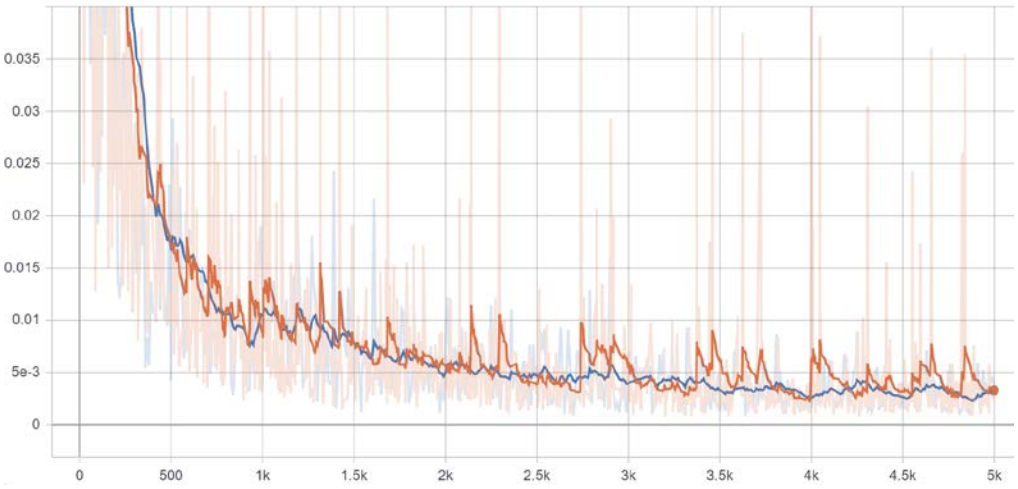


Figure 31. MobileNet Cross Entropy on DeepFake Nic Cage test set. Training cross entropy = 0.0024; validation cross entropy = 0.0037.

CHAPTER VII

Conclusion & Future Work

Conclusion

In this work, we proposed an approach to detecting the forgery videos generated by DeepFake. Our approach is based on the limitations that current DeepFake software can only generates short duration and low-resolution videos; Moreover, during the face mask conversion, various setting will further downgrade the resolution around the edge of the facial mask. Such distinctive artifacts can be captured by pre-trained deep neural networks. The testing results from experimentation indicate that our method is effective. Our retrained module is effective in detecting both DeepFake videos and the forgery videos from FaceForensics. Our approach has reached average 94.9% accuracy with significantly less training time.

Future Work

As the GAN related technology continues evolving, our detection method should also improve accordingly. We aim to improve the reliability of results by performing robust testing on various learning rate and different data sets; furthermore, transfer learning is better performed when trained on a pre-build forgery detection neural network, which requires dedicated training data set for image forgery classification. In the future, our research could aim to build a dedicate neural network for detecting various AI-generated forgery products.

REFERENCES

BuzzFeedVideo. (2018, April 17). You Won't Believe What Obama Says in This Video! Retrieved from https://www.youtube.com/watch?v=cQ54GDm1eL0

Bengio, Y., Courvile, A., Vincent, P., August 2013. A Review and New Perspectives. IEEE trans. Pattern Anal. Mach Intel., vol. 35, no. 8, pp. 1798-1828, Aug. 2013.

Wikipedia contributors. (2019, February 26). Artificial neuron. In Wikipedia, The Free Encyclopedia. Retrieved 18:29, February 26, 2019, from https://en.wikipedia.org/w/index.php?title=Artificial_neuron&oldid=885171186

Wikipedia contributors. (2019, February 22). Activation function. In Wikipedia, The Free Encyclopedia. Retrieved 18:30, February 26, 2019, from https://en.wikipedia.org/w/index.php?title=Activation_function&oldid=884537566

Clark, D. (2018, April). KDnuggets. Retrieved February, 2019, from https://www.kdnuggets.com/2018/04/top-16-open-source-deep-learning-libraries.html

Hale, J. (2018, September 20). Deep Learning Framework Power Scores 2018 – Towards Data Science. Retrieved February 20, 2019, from https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

TensorFlow Hub. (n.d.). TensorFlow Hub Introduction. Retrieved February 20, 2019, from https://www.tensorflow.org/hub

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. CoRR, abs/1704.04861.

Tsang, S. (2019). Review: Inception-v3: 1st Runner Up (Image Classification) in ILSVRC 2015. Retrieved February 20, 2019. from: https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c.

Matas, M. J., Reckhow, M. W., & Taigman, Y. (2017). U.S. Patent No. US20170140214A1. Washington, DC: U.S. Patent and Trademark Office.

Thies, J., Zollh246fer, M., Stamminger, M., Theobalt, C., & Nie223ner, M. (2018). Face2Face: real-time face capture and reenactment of RGB videos. CACM.

Thies, J., Zollhöfer, M., Theobalt, C., Stamminger, M., & Nießner, M. (2018). Headon: real-time reenactment of human portrait videos. ACM Trans. Graph., 37, 164:1-164:13.

Suwajanakorn, S., Seitz, S.M., & Kemelmacher-Shlizerman, I. (2017). Synthesizing Obama: learning lip sync from audio. ACM Trans. Graph., 36, 95:1-95:13.

Karras, T., Laine, S., & Aila, T. (2018). A Style-Based Generator Architecture for Generative Adversarial Networks. CoRR, abs/1812.04948.

Lample, G., Zeghidour, N., Usunier, N., Bordes, A., Denoyer, L., & Ranzato, M. (2017). Fader Networks: Manipulating Images by Sliding Attributes. NIPS.

Lu, Y., Tai, Y., & Tang, C. (2017). Conditional CycleGAN for Attribute Guided Face Image Generation. CoRR, abs/1705.09966.

Sharma, S. (2018, August 04). Celebrity Face Generation using GANs (Tensorflow Implementation). Retrieved February, 2019, from https://medium.com/coinmonks/celebrity-face-generation-using-gans-tensorflow-implementation-eaa2001eef86

Li, J. (2018, June 01). CelebFaces Attributes (CelebA) Dataset. Retrieved March 17, 2019, from https://www.kaggle.com/jessicali9530/celeba-dataset

Seibold, C., Samek, W., Hilsmann, A., & Eisert, P. (2017). Detection of Face Morphing Attacks by Deep Learning. IWDW.

Cozzolino, D., Poggi, G., & Verdoliva, L. (2017). Recasting Residual-based Local Descriptors as Convolutional Neural Networks: an Application to Image Forgery Detection. IH&MMSec.

Bayar, B., & Stamm, M.C. (2016). A Deep Learning Approach to Universal Image Manipulation Detection Using a New Convolutional Layer. IH&MMSec.

Rahmouni, N., Nozick, V., Yamagishi, J., & Echizen, I. (2017). Distinguishing computer graphics from natural images using convolution neural networks. 2017 IEEE Workshop on Information Forensics and Security (WIFS), 1-6.

Raghavendra, R., Raja, K.B., Venkatesh, S., & Busch, C. (2017). Transferable Deep-CNN Features for Detecting Digital and Print-Scanned Morphed Face Images. 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 1822-1830.

Guera, D., & Delp, E.J. (2018). DeepFake Video Detection Using Recurrent Neural Networks. 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 1-6.

Li, Y., & Lyu, S. (2018). Exposing DeepFake Videos by Detecting Face Warping Artifacts. CoRR, abs/1811.00656.

Hui, J. (2018, April 28). How deep learning fakes videos (DeepFakes) and how to detect it? Retrieved February 20, 2019, from https://medium.com/@jonathan_hui/how-deep-learning-fakes-videos-DeepFakes-and-how-to-detect-it-c0b50fbf7cb9

Li, C., & Balaban, S. (2018). Transfer Learning with TensorFlow Tutorial: Image Classification Example. Retrieved from https://lambdalabs.com/blog/transfer-learning-with-tensorflow-tutorial-image-classification-example/

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2818-2826.

GHB. (2017, December 23). Understating Google MobileNet. Retrieved March 15, 2019, from https://blog.csdn.net/T800GHB/article/details/78879612

cyberfire. (2017). GitHub tensorflow-mtcnn. Retrieved February 20, 2019, from https://github.com/cyberfire/tensorflow-mtcnn.

King, D.E. (2009). Dlib-ml: A Machine Learning Toolkit. Journal of Machine Learning Research, 10, 1755-1758.

Iperov. (2019). GitHub DeepFaceLab. Retrieved February 20, 2019, from https://github.com/iperov/DeepFaceLabm/iperov/DeepFaceLab.

Fisher, R., & Perkins, S. (2003, May). Erosion. Retrieved February 20, 2019, from https://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm

Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., & Nießner, M. (2018). FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces. CoRR, abs/1803.09179.

Kazemi, V., & Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. 2014 IEEE Conference on Computer Vision and Pattern Recognition, 1867-1874.

TensorFlow Hub. (n.d.). Create bottleneck file for Retraining an Image. Retrieved February 20, 2019, from https://www.tensorflow.org/hub/tutorials/image_retraining#bottlenecks

APPENDIX

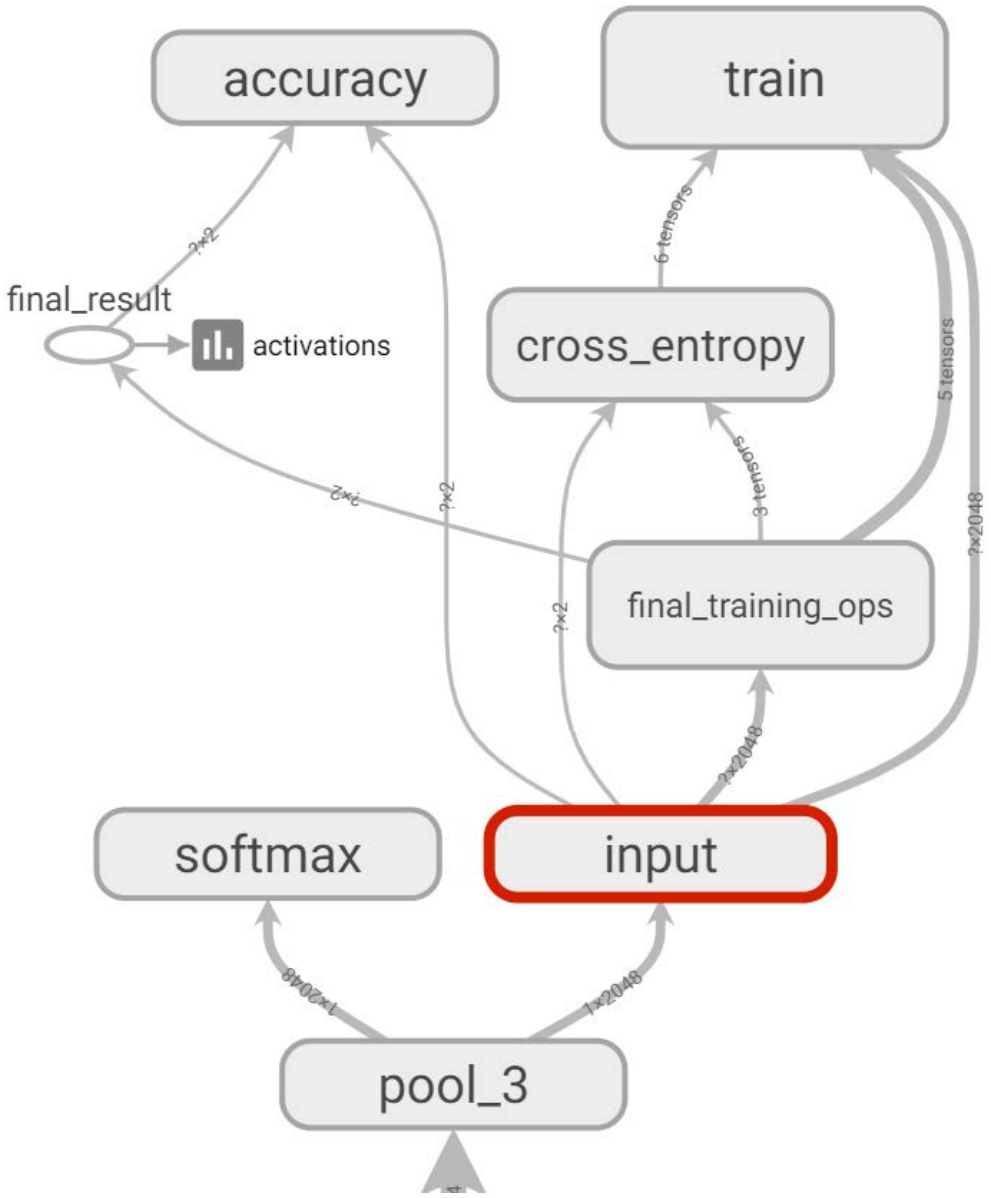


Figure 32. Inception V3 Bottleneck layer structure. Newly formed input layer will replace the original Softmax layer as classifier for new dataset.

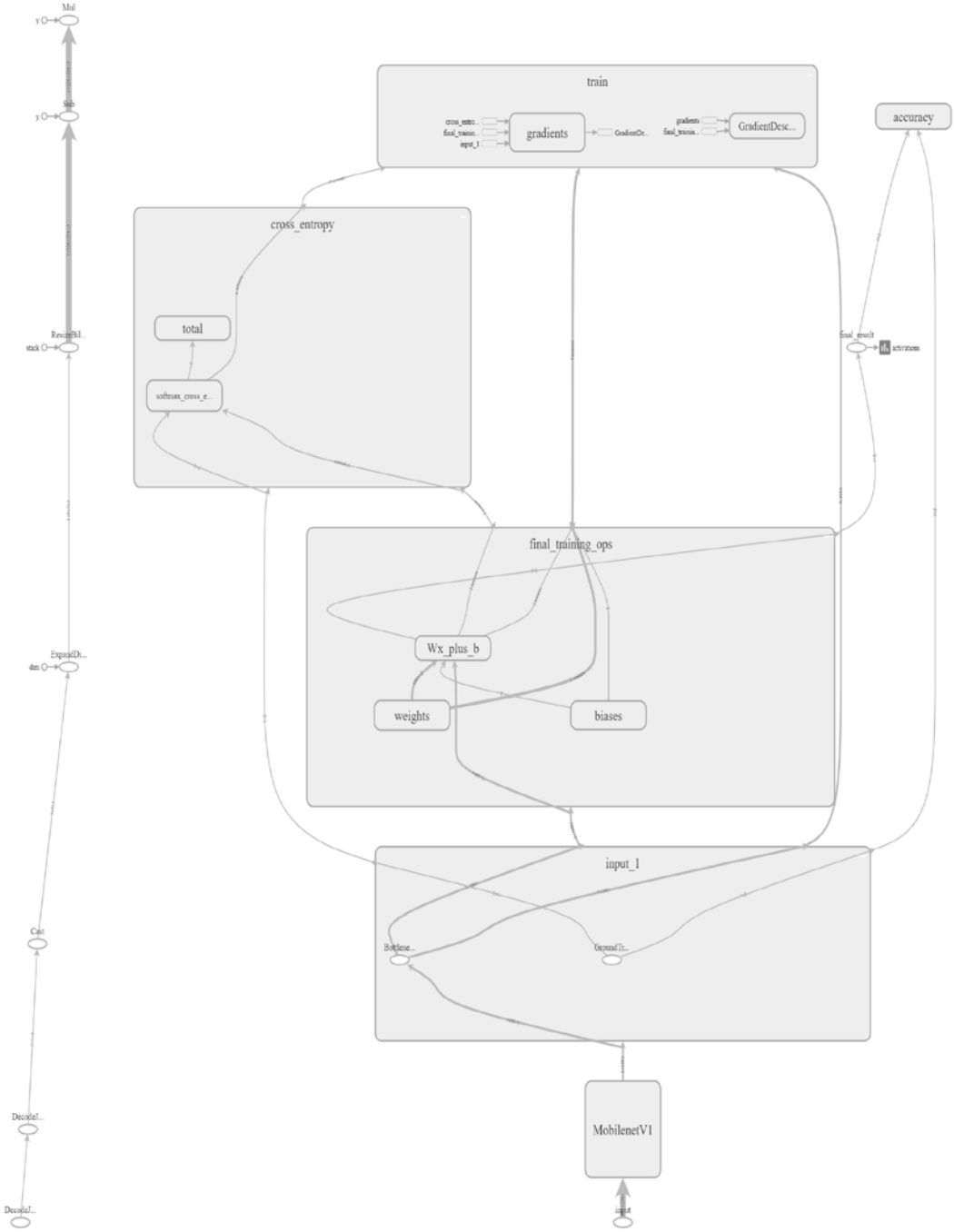


Figure 33. MobileNet V1 Bottleneck layer structure. Input_1 layer will replace the original Softmax and FC layer and update the value of weight and bias base on prediction of previous testing dataset result.

```
# run TensorBoard
tensorboard --logdir [IMAGE DIR]/training_summaries &
pkill -f "tensorboard"

#set initial parameter
IMAGE_SIZE=224
ARCHITECTURE="mobilenet_0.50_${IMAGE_SIZE}"

#Need Help?
python -m scripts.retrain -h

#Retrain Module (Train) REMEMBER TO CHANGE THE DIRECTORY!!!
#test 10%, validation 10%, train 10%

#Test retrain module on certain picture(Test)
python -m scripts.label_image
        \--graph=inceptionV3_5000/retrained_graph.pb
        \--labels=inceptionV3_5000/retrained_labels.txt
        \--image=inceptionV3_5000/Train_5000/Real_test.jpg

python -m scripts.label_image
        \--graph=inceptionV3_nic_7300/retrained_graph.pb
        \--labels=inceptionV3_nic_7300/retrained_labels.txt
        \--image=inceptionV3_nic_7300/nic_img/test_fakenic.jpg

#Convert PNG to JPG
$ ls -- *.png | xargs -n 1 bash -c 'convert "$0" "${0%.png}.jpg"'
```

Figure 34. Initial parameter settings. From top to bottom: Use Tensorboard monitor to the training progress; Setting initial parameters of MobileNet V1; Parameter Help; Use retrained module to predict the authenticity of specific picture;

```
#mobilenet_0.50_224 1000 (lr=0.005, STEP = 5000)
python -m scripts.retrain
    \--bottleneck_dir=MobileNet_1000/bottlenecks
    \--model_dir=MobileNet_1000/models/"mobilenet_0.50_224"
    \--summaries_dir=MobileNet_1000/training_summaries/LR_0.005
    \--output_graph=MobileNet_1000/retrained_graph.pb
    \--output_labels=MobileNet_1000/retrained_labels.txt
    \--architecture="mobilenet_0.50_224"
    \--learning_rate=0.005
    \--image_dir=MobileNet_1000/1000_images
    \--how_many_training_steps=5000
    \--print_misclassified_test_images

#mobilenet_0.50_224 5000 (lr=0.005, STEP = 5000)
python -m scripts.retrain
    \--architecture="mobilenet_0.50_224"
    \--learning_rate=0.005
    \--image_dir=MobileNet_5000/Train_5000
    \--model_dir=MobileNet_5000/models/"mobilenet_0.50_224"
    \--summaries_dir=MobileNet_5000/training_summaries/LR_0.005
    \--output_graph=MobileNet_5000/retrained_graph.pb
    \--output_labels=MobileNet_5000/retrained_labels.txt
    \--how_many_training_steps=5000
    \--print_misclassified_test_images

#mobilenet_0.50_224 25000 (lr=0.005, STEP = 5000)
python -m scripts.retrain
    \--architecture="mobilenet_0.50_224"
    \--learning_rate=0.005
    \--image_dir=MobileNet_25000/Image_25000
    \--model_dir=MobileNet_25000/models/"mobilenet_0.50_224"
    \--summaries_dir=MobileNet_25000/training_summaries/LR_0.005
    \--output_graph=MobileNet_25000/retrained_graph.pb
    \--output_labels=MobileNet_25000/retrained_labels.txt
    \--how_many_training_steps=5000
    \--print_misclassified_test_images
```

MobileNet V1_0.5_244

```
#inception v3 1000 (lr=0.005, STEP = 5000)
python -m scripts.retrain
    \--bottleneck_dir=inceptionV3_1000/bottlenecks
    \--model_dir=inceptionV3_1000/models/"inception_v3"
    \--summaries_dir=inceptionV3_1000/training_summaries/LR_0.005
    \--output_graph=inceptionV3_1000/retrained_graph.pb
    \--output_labels=inceptionV3_1000/retrained_labels.txt
    \--architecture="inception_v3"
    \--learning_rate=0.005
    \--image_dir=inceptionV3_1000/1000_images
    \--how_many_training_steps=5000
    \--print_misclassified_test_images

#inception v3 5000 (lr=0.005, STEP = 5000)
python -m scripts.retrain
    \--architecture="inception_v3"
    \--learning_rate=0.005
    \--image_dir=inceptionV3_5000/Train_5000
    \--model_dir=inceptionV3_5000/models/"inception_v3"
    \--summaries_dir=inceptionV3_5000/training_summaries/LR_0.005
    \--output_graph=inceptionV3_5000/retrained_graph.pb
    \--output_labels=inceptionV3_5000/retrained_labels.txt
    \--how_many_training_steps=5000
    \--print_misclassified_test_images
    \--bottleneck_dir=inceptionV3_5000/bottlenecks

#inception v3 25000 (lr =0.005, step = 5000)
python -m scripts.retrain
    \--architecture="inception_v3"
    \--learning_rate=0.005
    \--image_dir=inceptionV3_25000/Image_25000
    \--model_dir=inceptionV3_25000/models/"inception_v3"
    \--summaries_dir=inceptionV3_25000/training_summaries/LR_0.005
    \--output_graph=inceptionV3_25000/retrained_graph.pb
    \--output_labels=inceptionV3_25000/retrained_labels.txt
    \--how_many_training_steps=5000
    \--print_misclassified_test_images
    \--bottleneck_dir=inceptionV3_25000/bottlenecks
```

Inception V3

Figure 35. Parameter settings for retraining on 1000, 5000 and 25000 FaceForensics data. Left: MobileNetV1; Right Inception V3.

```
#MobileNet NIC (lr=0.005, STEP = 5000)
python -m scripts.retrain
    \--architecture="mobilenet_0.50_224"
    --learning_rate=0.005
    --image_dir=MobileNet_nic_7300/nic_img
    --model_dir=MobileNet_nic_7300/models/"mobilenet_0.50_224"
    --summaries_dir=MobileNet_nic_7300/training_summaries/LR_0.005
    --output_graph=MobileNet_nic_7300/retrained_graph.pb
    --output_labels=MobileNet_nic_7300/retrained_labels.txt
    --how_many_training_steps=5000
    --print_misclassified_test_images
    --bottleneck_dir=MobileNet_nic_7300/bottlenecks

#MobileNet Elon (lr=0.005, STEP = 5000)
python -m scripts.retrain
    \--architecture="mobilenet_0.50_224"
    \--learning_rate=0.005
    \--image_dir=MobileNet_elon_3600/elon_img
    \--model_dir=MobileNet_elon_3600/models/"mobilenet_0.50_224"
    \--summaries_dir=MobileNet_elon_3600/training_summaries/LR_0.005
    \--output_graph=MobileNet_elon_3600/retrained_graph.pb
    \--output_labels=MobileNet_elon_3600/retrained_labels.txt
    \--how_many_training_steps=5000
    \--print_misclassified_test_images
    \--bottleneck_dir=MobileNet_elon_3600/bottlenecks

#inception v3 NIC (lr=0.005, STEP = 5000)
python -m scripts.retrain
    \--architecture="inception_v3"
    \--learning_rate=0.005
    \--image_dir=inceptionV3_nic_7300/nic_img
    \--model_dir=inceptionV3_nic_7300/models/"inception_v3"
    \--summaries_dir=inceptionV3_nic_7300/training_summaries/LR_0.005
    \--output_graph=inceptionV3_nic_7300/retrained_graph.pb
    \--output_labels=inceptionV3_nic_7300/retrained_labels.txt
    \--how_many_training_steps=5000 \--print_misclassified_test_images
    \--bottleneck_dir=inceptionV3_nic_7300/bottlenecks

#inception v3 Elon (lr=0.005, STEP = 5000)
python -m scripts.retrain
    \--architecture="inception_v3"
    \--learning_rate=0.005
    \--image_dir=inceptionV3_elon_3600/elon_img
    \--model_dir=inceptionV3_elon_3600/models/"inception_v3"
    \--summaries_dir=inceptionV3_elon_3600/training_summaries/LR_0.005
    \--output_graph=inceptionV3_elon_3600/retrained_graph.pb
    \--output_labels=inceptionV3_nic_7300/retrained_labels.txt
    \--how_many_training_steps=5000
    \--print_misclassified_test_images
    \--bottleneck_dir=inceptionV3_elon_3600/bottlenecks
```

Figure 36. Parameter settings for retraining on DeepFake dataset. From top to bottom: MobileNet V1 trained on Nic and Elon; Inception V3 trained on Nic and Elon.

VITA

Zhaohe Zhang

EDUCATION

Master of Science student in Computing and Information Science at Sam Houston State University, May 2017 – present. Thesis title: “Detect Forgery Video by Performing Transfer Learning on Deep Neural Network.”

Bachelor of Science (August 2016) in, Sam Houston State University, Huntsville, Texas.

ACADEMIC EMPLOYMENT

Graduate Teaching Assistant, Department of Computer Science, Sam Houston State University, August 2017 - present. Responsibilities include: assisting professors with the preparation and presentation of undergraduate courses, grading, and hosting TA office hours.

Math Tutor, Academic Success Center, Sam Houston State University, August 2015 – August 2016. Responsibilities include: Providing instructions to students and assist them improve their math skills, prepare study guides for Calculus and Algebra students.

PUBLICATIONS

Hutchinson S., **Zhang Z**., Liu Q. (2018) Detecting Phishing Websites with Random Forest. In: Meng L., Zhang Y. (eds) Machine Learning and Intelligent Communications. MLICOM 2018. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 251. Springer, Cham

ACADEMIC AWARDS

COSET Special Graduate Scholarship, College of Science and Engineering Technology, SHSU, 2017 – 2019.

Computer Science Department Scholarship, Department of Computer Sciences, SHSU, Fall 2015.

PROFESSIONAL MEMBERSHIP

Institute of Electrical and Electronics Engineers Graduate Student Member, Southwestern USA, Houston Section.