NEAR EAST UNIVERSITY

INSTITUTE OF GRADUATE STUDIES

DEPARTMENT OF COMPUTER ENGINEERING

Deep Fakes Generation Using LSTM based Generative Adversarial Networks

M.Sc. THESIS

SABA ELGAMMUDI

Supervisor
Assist. Prof. Dr. Elbrus Imanov

Nicosia

September, 2021

Approval

We certify that we have read the thesis submitted by Saba Abdulbaset Elgammudi titled "**Deep Fakes Generation LSTM based Generative Adversarial Networks**" and that in our combined opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Educational Sciences.

Examining Committee	Name-Surname	Signature
Head of the Committee:	Pro. Dr. Rahib H. Abiyev	
Committee Member*:	Assoc. Prof. Dr. Kamil Dimililer	
Supervisor:	Assist. Prof. Dr. Elbrus Imanov	

Approved by the Head of the Department

…../…../20…

Pro. Dr. Rahib H. Abiyev
Head of Department

Approved by the Institute of Graduate Studies

…../…../20…

Prof. Dr. Kemal Hüsnü Can Başer
Head of the Institute

Declaration

I hereby declare that all information, documents, analysis and results in this thesis have been collected and presented according to the academic rules and ethical guidelines of Institute of Graduate Studies, Near East University. I also declare that as required by these rules and conduct, I have fully cited and referenced information and data that are not original to this study.

Saba Elgammudi

17/12/2021

ACKNOWLEDGEMENTS

I'd want to convey our heartfelt gratitude to Dr. ELBRUS IMANOV, my project guide, for his unwavering support and encouragement during this endeavor. I am grateful to him for being present at every stage of the project, sharing his technical skills and providing crucial input and suggestions that helped make it a success. I'd also like to express my gratitude to the officials and employees of the Computer Engineering department who assisted me during that time. Last but not least, I want to express my gratitude to my family for their unwavering support and confidence in me throughout our master's degree journey.

Saba Elgammudi

I certify that this research work entitled “Deep fakes Generation Using LSTM based Generative Adversarial Networks” is my own work. No portion of the work presented in this research report has been submitted in support of another award or qualification either at this institution or elsewhere. Where material has been used from other sources it has been properly acknowledged / referred. If any part of this project is proved to be copied or found to be a report of some other, I will stand by the consequences.

Name, Last Name Saba Elgammudi

Signature

[illegible]

Date 1/19/2022

NEAR EAST UNIVERSITY

ETHICAL APPROVAL DOCUMENT

Date: 05/01/2022

To the Institute of Graduate Studies

For the thesis project entitled as “Deep fakes Generation Using LSTM based Generative Adversarial Networks”, the researchers declare that they did not collect any data from human/animal or any other subjects. Therefore, this project does not need to go through the ethics committee evaluation.

Title: Assist. Prof. Dr

Name Surname: Elbrus Imanov

Signature: [illegible]

Role in the Research Project: Supervisor

ACKNOWLEDGEMENTS

I'd want to convey our heartfelt gratitude to Dr. ELBRUS IMANOV, my project guide, for his unwavering support and encouragement during this endeavor. I am grateful to him for being present at every stage of the project, sharing his technical skills and providing crucial input and suggestions that helped make it a success. I'd also like to express my gratitude to the officials and employees of the Computer Engineering department who assisted me during that time. Last but not least, I want to express my gratitude to my family for their unwavering support and confidence in me throughout our master's degree journey.

Abstract

Deep Fakes Generation LSTM based Generative Adversarial Networks

Saba Elgammudi
MA, Department of Computer Engineering
December, 2021, 60 pages

Deep learning has shown promising outcomes in a variety of difficult task categories. Recent advances in deep learning, on the other hand, have been used to develop Software that puts people's privacy and national security at risk. One of these is deep fakes, which makes phony images and videos that are undetectable as forgeries. Humans. Fake speeches by international leaders can put the world's stability and peace in jeopardy. Apart from dangerous applications, deep fakes is used also for good, its used in ex: post production and in translating in films. The scenario was employed in the most recent elections in India, allowing politicians' statements to be translated into a variety of Indian languages. Traditionally, this job was done with computer graphics and 3D models. However, developments of AI and ML and computer vision, particularly the GAN, are displacing early method in favor of DL methods. This study aims to use deep neural networks to create modified faces in photos and videos.

This thesis for master's degree is proposes for generating a full complete video sequence from a source image and a destination video. NVIDIA's work on vidTOvid few shots vidTOvid, so they map video from source domains to target domains by learning, encouraged us. This paper, I offer a united models based on long short term memory on GA networks, as well as a movement module that generates dense motion using a keypoint detector.

To be able to create a real looking video, the network that generates uses a cover to mix the way it appears retrieved taking from video source also includes the movement from the video target. The instruction is completed from beginning to end, and the main concepts are learned independently. The recently released FaceForensics++ and VoxCeleb datasets are used to demonstrate the evaluation.

Our work's key contribution is the development of innovative neural networks for creating deepfake images and movies. In addition, our primary goal is to provide datasets for deepfake detection problems.

Key Words: gan, deep fakes, ml, ai, lstm, deep learning

Table of Contents

CHAPTER 1

INTRODUCTION

DL (deep learning) and computer vision in the recent years have progressed. It has become an inextricable everyone’s life and is utilized practically all around the world, in cell phones and computers. In medicine deep learning technologies and dependencies will be used and installed in the coming decade. Toolkits and software’s have standard components. Self-driving cars, for example, are a result of recent technological advancements. Detecting objects, face recognitions and character recognition is not possible without automobiles. DL (deep learning) is not possible without it. The goal of deep learning approaches is to learn high-level features from the data. Without relying on human-crafted data and mapping inputs to appropriate outputs features. The methods or processes used to extract the features might differ. DL (deep learning) techniques are examples of this type of technology. Emerged in the last ten years. CNNs and long short term memory on GA networks aid extracted features and were shown to solve real-world issues. Such as video and image classification, recognizing objects and attitude analysis, as well as segmentation empirically. Deep learning includes two important tasks: image categorization and recognition. Classification is the process of categorizing items into different groups, whereas recognition is the process of accurately identifying an object in an image. Object detection is a relatively new advancement that employs to recognize instances of items such as faces, cars, and

Roads, image analysis, machine learning, and machine vision algorithms are used. Deep fake operations now include a significant amount of face detection and recognition.

Face detection locates a face in an image, whereas facial recognition establishes the face's identity. Because of the differences in the features, such as facial hair, orientation, and location of a face these operations can be rather difficult. Hence, In the realm of deep learning, it is a major topic of study. Face detection and recognition are also employed in a variety of real-world applications, including Identity verification and cell phones are two examples. Methods for detecting faces include feature based, probabilistic, probabilistic methods [47]. Template matching and based on look. Deep learning technologies such as OpenCV are now widely used. Among the most prominent approaches for achieving these duties are the Dlib networks. These networks have been trained on millions of faces, resulting in improved results. CNNs are used in deep learning take information from photos and mapped to words. Corresponding outputs. The receptive field is sized of the convolution filters, and the filter taps are parameters acquired by a backpropagation procedure [46]. CNNs can include millions of parameters and are interspersed with nonlinear activation functions and normalization procedures. CNNs are used as a backbone in face detection and identification systems. An example of a CNN network for face recognition shown is Figure 1.

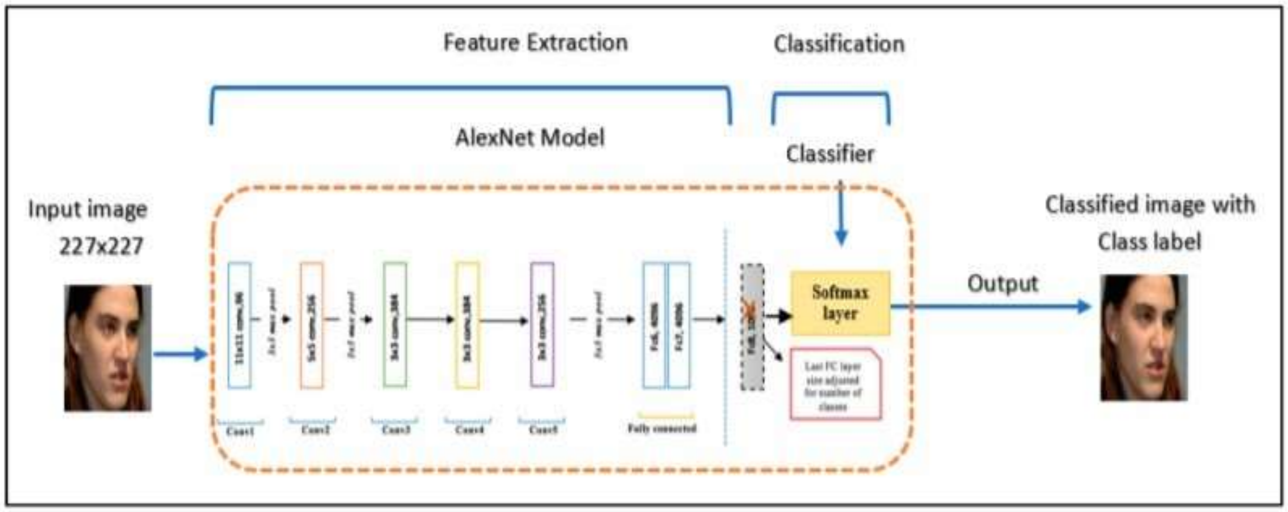


Figure 1: Example of CNN used for facial recognition
[53].

1.2 Types of Deepfakes:

The creation of software that constitutes a public threat has been helped by technological advancements in DL (deep learning). Deep fake is one of the newest dangerous threats they've faced [23]. The algorithms of DL can make fake faces of people that are difficult to distinguish from the real ones it also can make videos and images. Deep fake first video led to the moniker deepfakes. Deepfakes can be made in a variety of ways. Deepfakes were first created using computer graphics techniques, but deep learning approaches have recently taken over [24].

GANs were used to construct the first deep fake video, used a decoding and encoding networks. There are many methods for creating altered photos and films have since

Emerged [18]. Face manipulations are divided into two categories: face swap and recreation of facial expressions. Face swap is the process of moving the source person facial details and expression to a target person, whereas facial reenactment is the process of moving just the emotions and pose from the source person to the target person. Figure 2 is an example of Face swap, while Figure 3 is an example of facial reenactment.

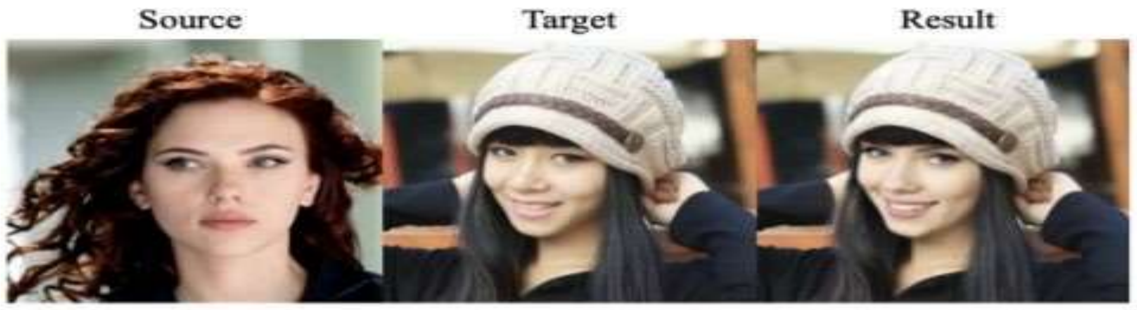


Figure 2: Face swap example [52].

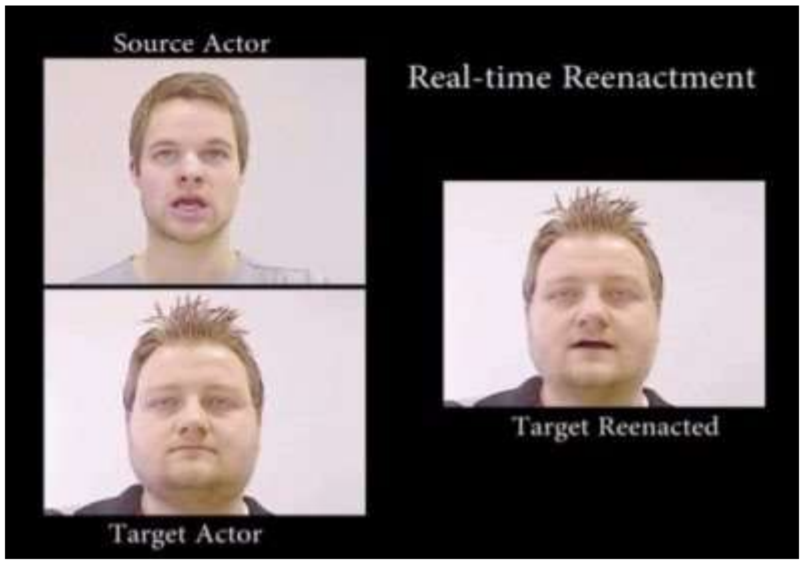


Figure 3: Facial reenactment example [51].

Both sorts of fakes are depicted in Figures 2 and 3. Face swap is seen in Figure 2. To make a fake, the target people’s face is changed with the source person’s face, as shown in Figure 2. Figure 3 is a scenario of facial reenactment. The emotion of the source video is swapped with the emotion of the target image to create a fake image, as seen in Figure 3.

As a result, various deep learning approaches to detect the presence of these technologies have evolved to fix the issue of deep fake.

Facebook and Microsoft recently established a competition to detect deep fake to boost the detecting and addressing deep fake’s research, citing worries that these films could endanger the public by distributing false information or assertions [48].

Deep Learning networks needs a lot of data to detect deep fakes. A lot of datasets are available for the public, including Facebook's Face Forensics++ and Deep Fake Detection Challenge datasets. However, fakes were constructed for all of those datasets using the Face2Face [41], Face swap, Neural Textures [42] and Deep Fakes approaches. Deep Fakes [7] and Face swap [8] are ways that swap the source image face with the target image face. Face swapping is based on graphics, whereas Deep Fakes is based on GANs [15].

The Neural Textures and Face2face methods copy the target image's facial expressions to the source picture [2]. The FacetoFace approach uses blend shape coefficients to transfer source expression to the target video, whereas GANs is used by the Neural Textures

Which uses to learn the facials of the target person [5]. The techniques described above do not have generalization capacity, which means they cannot be used to previously unseen photos or videos. With a source image and a target video, our model attempts to transfer the source person's facial expressions to the target person's image. The major goal of this thesis is to create datasets that will be used to build robust detection models [19].

CHAPTER 2

BACKGROUND

2.1 (CNNs) Convolutional Neural Networks:

One algorithm of deep learning algorithms is convolutional Neural Networks (CNNs) they accept visual inputs and learn for the best output learning tasks utilizing various operations.

Multiple filters move over the image and apply convolution neural network to each part of the image in a convolution layer. Each filter extracts distinct aspects of the image, such as corners, color, and pixels data. The receptive field is the size of the filter, Training algorithm is used to learn the filter values, which are referred to as parameters.

The spatial size of the maps output from the CNN layer output is reduced by a pooling layer.

For obtaining the dominating characteristics from the convolved output, pooling layers can be effective. There is two most common pooling which are max pooling and average pooling. The outputs of the average pooling are average of all the values under

That window, whereas the outputs of the max pooling are maximum pixel values from the section of the image beneath that window. Introducing non-linearity in the output, an activation layer is usually added after the convolution layer. Tanh, sigmoid, ReLU [49], and other general activation functions are utilized. The activation functions aid in the learning of the input image's high-level representations [19].

In many activities, the ultimate layer is a fully connected layer. It reduces a multi-dimensional representation to an n-dimensional representation.

During training, the masses of the filter values and fully connected layer are updated by maximizing the loss between the ground truth and predicted values [36]. Figure 4 depicts a typical CNN.

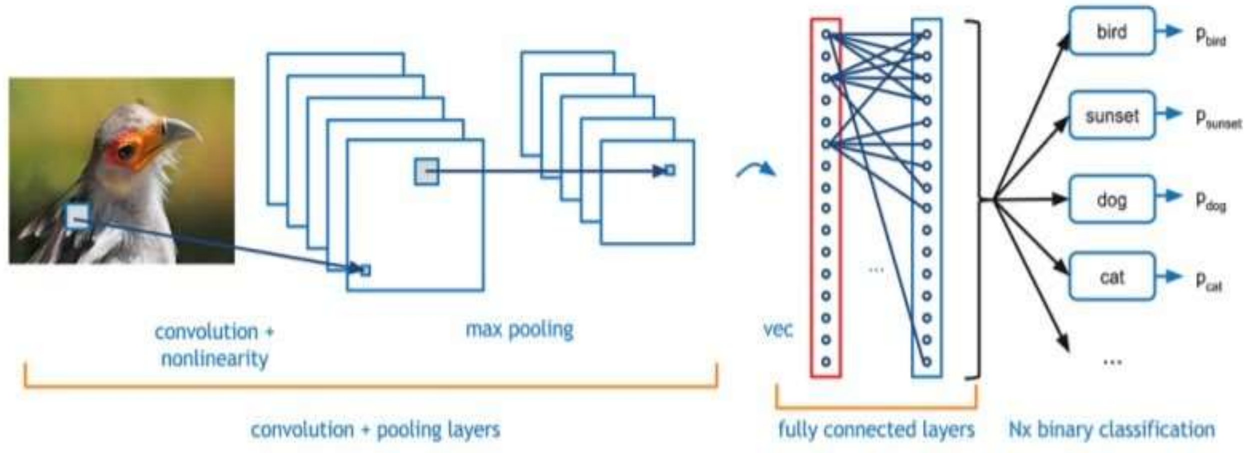


Figure 4: Convolutional Neural Network example [50].

2.2 (GANs)Generative Adversarial Networks:

Goodfellow et al [38]. defined GANs as a type of CNN. The network functions in a similar way to CNNs, it does, however, have mainly two sub-networks: a generator generates new other examples and a discriminator that determines whether input data is real or fake. GANs are classified as unsupervised learning algorithms since they do not require humans to manually identify training images [40].

GANs are generative models that produce new data instances which is similar to the input. GANs, an example of that is, can generate lifelike faces that is imaginary and do not represent any living person. Both the generator and discriminator are trained in an adversarial manner up to the time the discriminator is deceived, indicating that the generator can generate data that appears realistic. The generator generates an image from random noise, which is subsequently sent to the discriminator. Then the discriminator accepts all the real and false images (produced by the generator) as input and outputs a probability is always between 0 and 1, with 0 indicating the fake false and 1 indicating the real true. As a result, generative models attempt to learn the allocation of classes in the GNA, whereas discriminative models attempt to learn the distinctions between actual and fake images [39]. Figure 5 depicts a normal GAN network.

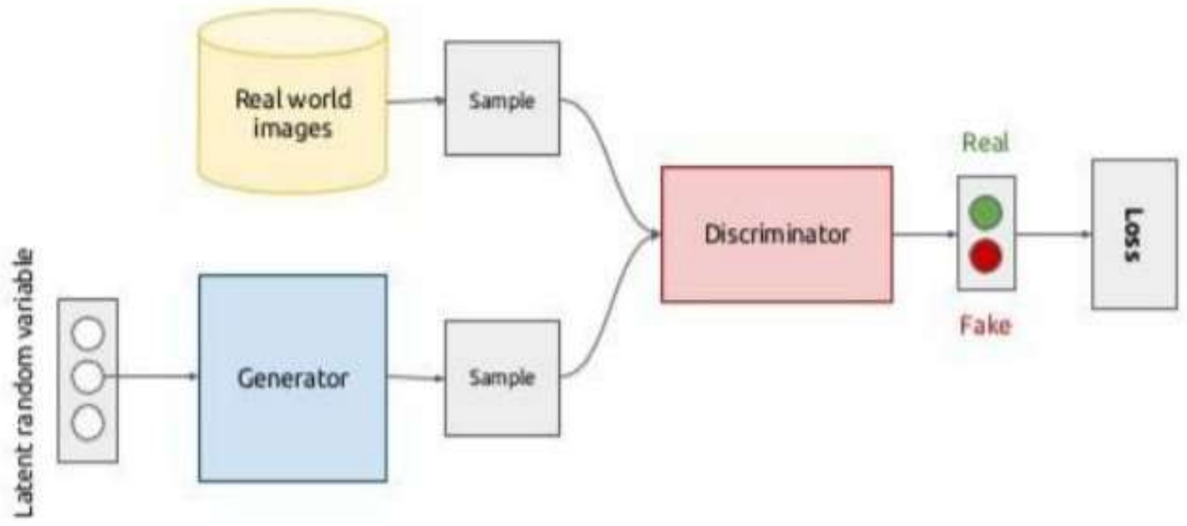


Figure 5: A normal example of Generative Adversarial Network [45]

2.3 (LSTM) Long short term memory:

Recurrent neural networks are different from feed forward neural networks in that they are artificial neural networks. Recurrent neural networks can receive one or more inputs and produce outputs that are influenced by both current and previous inputs and outputs. As a result, even if the inputs are same, the output may alter due to differences in the old inputs that effects output all time. The (LSTM) is a one of RNN forms that can render long sequences. When time lags exceed 5-10 steps [25], standard RNNs sometimes fail to remember so it wasn't affective in long sequences, however LSTMs have been found to be effective in long sequences even longer than 100 steps. RNNs have two major flaws: vanishing gradients and ballooning gradients. The uniquely developed architecture of LSTMs overcomes these problems. The LSTM layer is made

Up of memory blocks stacks that are coupled in a recursively [26]. Each memory stack consists of a recursively connected memory cell and contains three more gates: an input, an output, and a forget gates, all of which provide the cell with write operation, read operation, and reset operation. These gates allow each net to interact with the cells. Speech recognition, video captioning, and machine translation are some of the uses of LSTMs. Figures 6 and 7 depict a typical RNN [22] and LSTM [20].

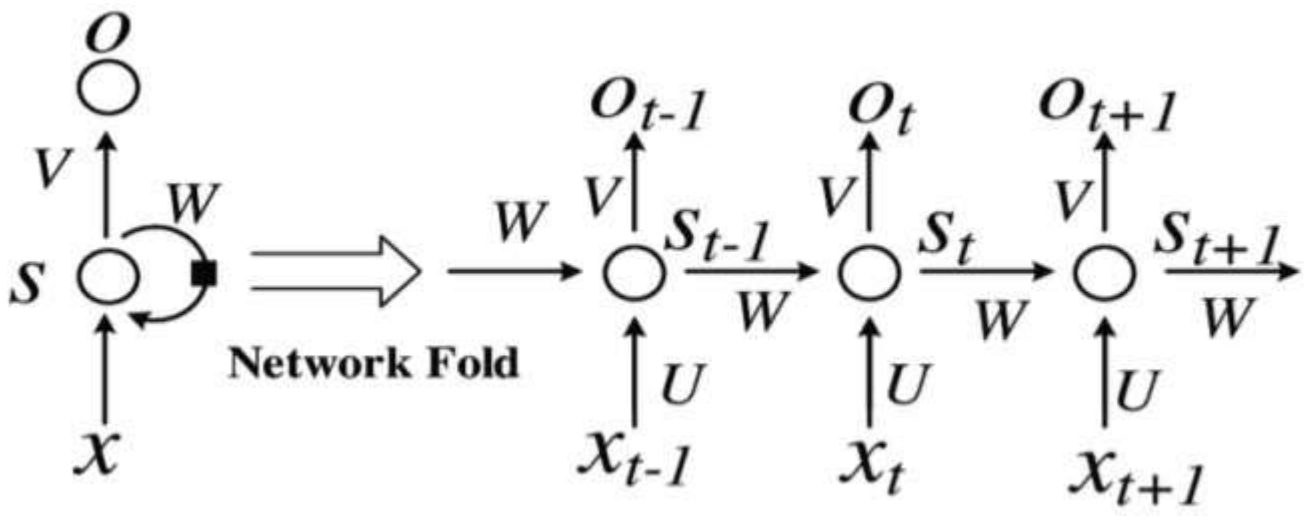


Figure 6: A simple example of Recurrent Neural Networks [44].

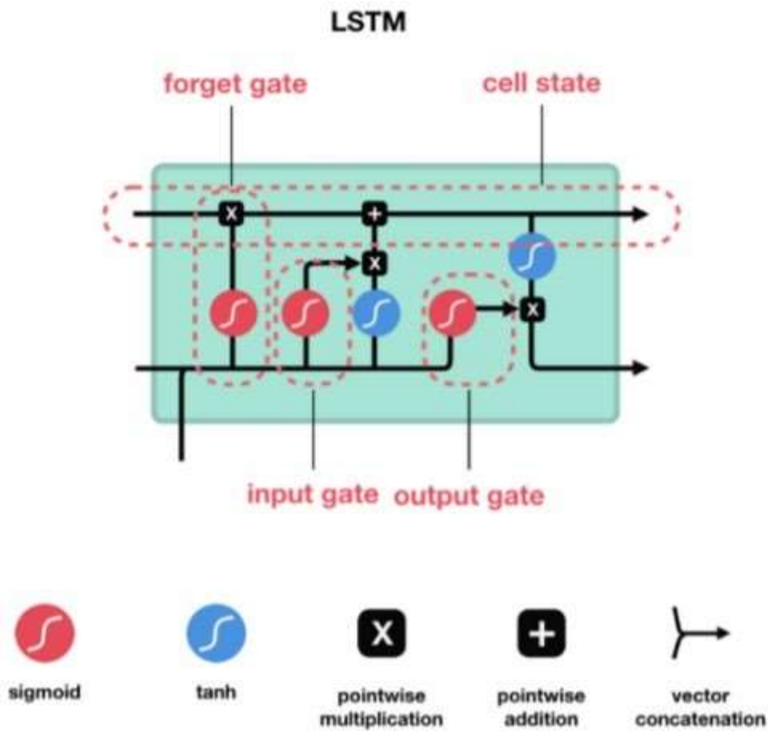


Figure 7: Long Short Term Memory Network Example [14].

2.4 Flow net and Optical Flow:

Optical flow is a technique for calculating picture intensity motion. It essentially mentions the pattern of motion of picture objects between successive frames. Images simply have spatial organization, whereas videos have both temporal and spatial structure. As a result, visual flow aids in temporal comprehension. CNNs has been proved to be effective in a multiple application, including optical flow estimates [26].

Optical flow aids video production using GANs and increases classification

Performance in video classification applications. Smoothing is used in GANs to make realistic videos so that they can produce cohesive outputs. Philipp Fischer has provided Figure 8 as an example of Flow net [16] [43].

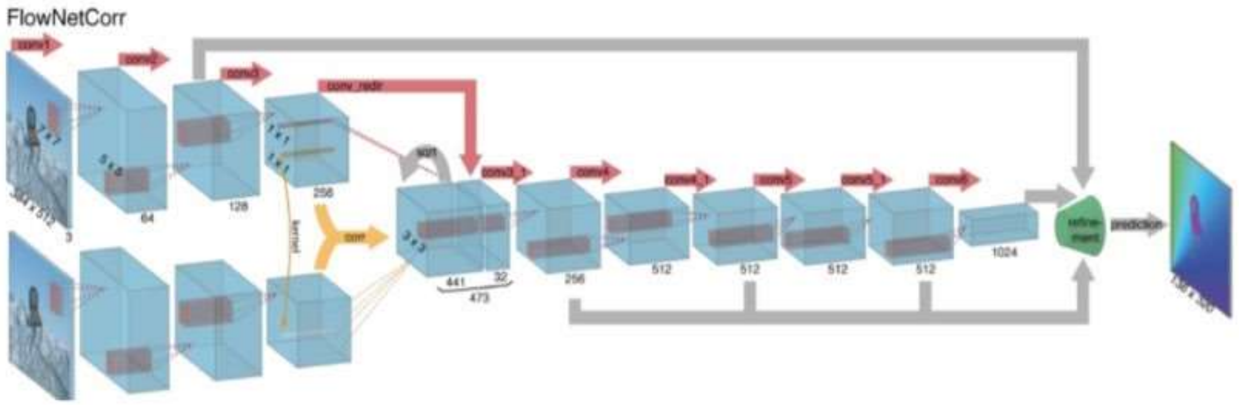


Figure 8: Optical flow network example [43].

CHAPTER 3

Other work

3.1 Face swap:

Face swap is a technique in which the target face of a person is superimposed on the source face of a person to generate a video of the target person performing or speaking in a manner as close to the source person’s face as feasible [12].

Because of their high quality, they have grown in popularity. The majority of them were created utilizing deep learning techniques such as GANs. Deep fake first video was created by a user on internet who went by the handle deep fakes [9], hence the name. Nguyen et al [1]. describe different generation and detection strategies that have been proposed. To make deep fakes, two encoder and decoder pairs are required, with each picture pair being trained separately using the same encoder and different decoder networks. To create a fake, it is separated to different class’s class A uses a normal encoder and decoded with a class B decoder during testing. Many Face swap GANs appeared later [18], however they always utilized extra inputs like dual face masks and facial landmark heat maps. Figure 9 is a common Face swap GAN.

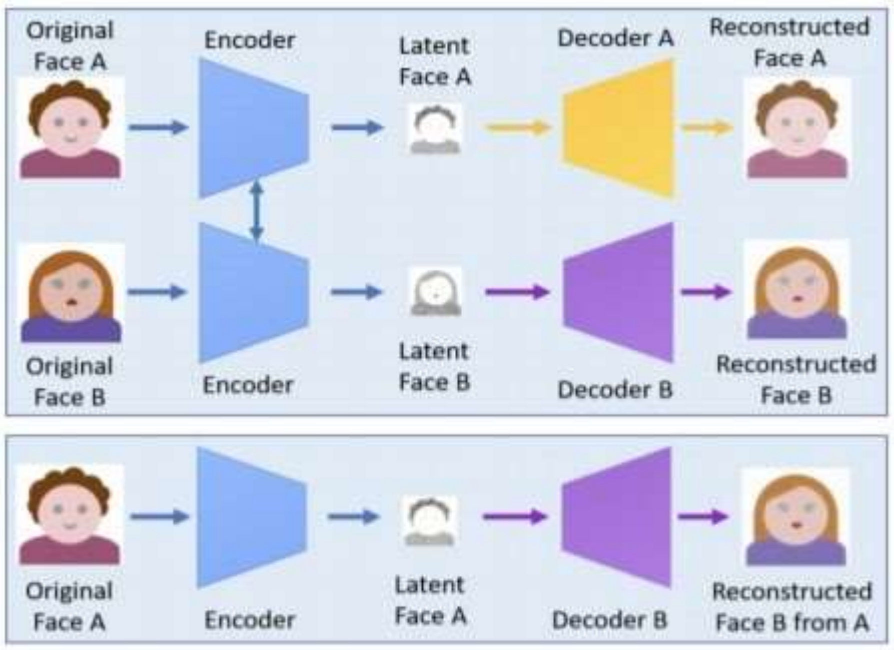


Figure 9: a common Face swap GAN [1].

3.2 Video2Video Synthesis:

Wang et al [27] proposed a video-to-video synthesis approach in which an adversarial learning framework is used to learn a mapping between the input video and the output realistic fake video. Their model is an outgrowth of their prior image-to-image [27] translation work, in which they generated high-resolution images using local and global generators [37]. They employ the pix2pixHD [37] generating network in the vid2vid network.

For generating high-resolution realistic movies, they divide the generator into two main sub networks: some global and local generators. They first train the global generator, then the local generator, and eventually all of the networks are fine-tuned. They employ this method to gather both global and local data. Multiscale discriminators are also used

To distinguish between high resolution actual and synthetic images. They employ three separate discriminators, each with a different receptive field. In the vid2vid network, Wang et al used two types of discriminators, an image and video discriminator. The image discriminator's goal is to ensure that the generated frame looks like a ground truth frame, whereas the video discriminator's goal is to make sure that output generated frames seem like solid truth frames. However, this network has a restriction in that we cannot define the target images and it consumes a lot of data. It also lacks generalization capabilities. The pix2pixHD [21] network is vid2vid's backbone network, an example in Figure 10.

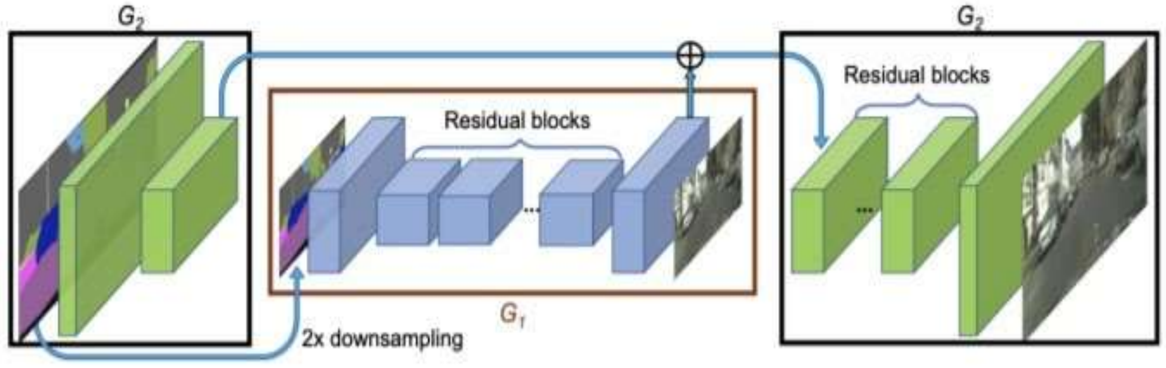


Figure 10: an example of network architecture of pix2pixHD [37].

3.3 Few shot Video2Video Synthesis:

To solve the constraints of vid2vid, the creator of vid2vid synthesis advised a few shot vid2vid architecture. The model, for starters, is data-hungry. For training, a large number of target photos are necessary. Second, the vid2vid network's generalization

Power is limited, as it haves a hard time to generalize to unanticipated images. The main difference between vid2vid and few shot vid2vid is shown in Figure 11. The few shot vid2vid network accepts a target images together with the input video as input, whereas the vid2vid [13] network only accepts one. The few shot vid2vid network uses a weight generation technique to dynamically customize the vid2vid mechanism based on the few target images [17].

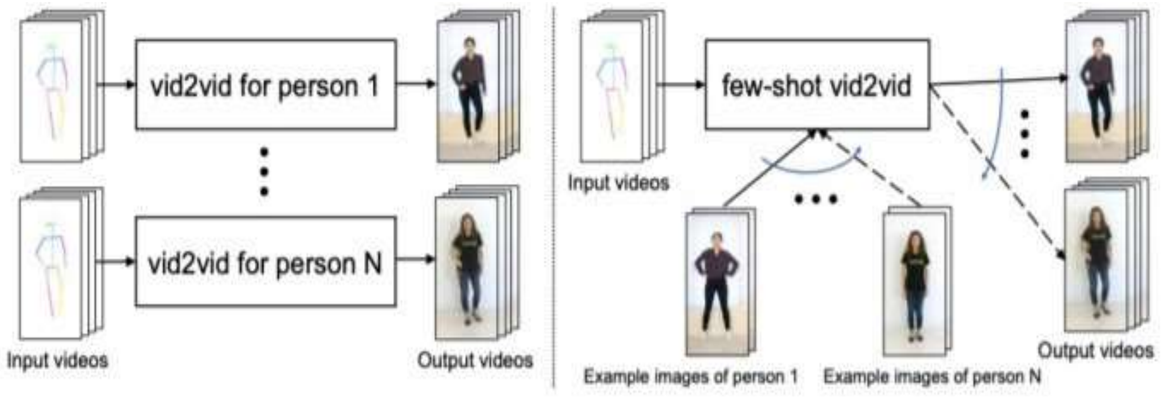


Figure 11: Main difference between vid2vid and few shot vid2vid [29].

Park et al [30]. Presented an adaptive spade network, which is shown in Figure 12, to achieve a few-shot [31] generalization capacity in their model. The goal of the vid2vid synthesis is to discover a scanning that can convert input videos into realistic output videos. Few shot vid2vid, on the other hand, has not been trained to generate videos from unknown domains [5]. Unlike vid2vid, it requires two inputs to create a video.

It accepts a second input in addition to the input video, which contains of a few images of the target image that are made available at test time. For the weight generalization mechanism, these examples are taken dynamically. When the example and target photos are comparable in most locations, they used warping as well. The Face Forensics++ dataset [10] is used to train both vid2vid and few shot vid2vid [11].

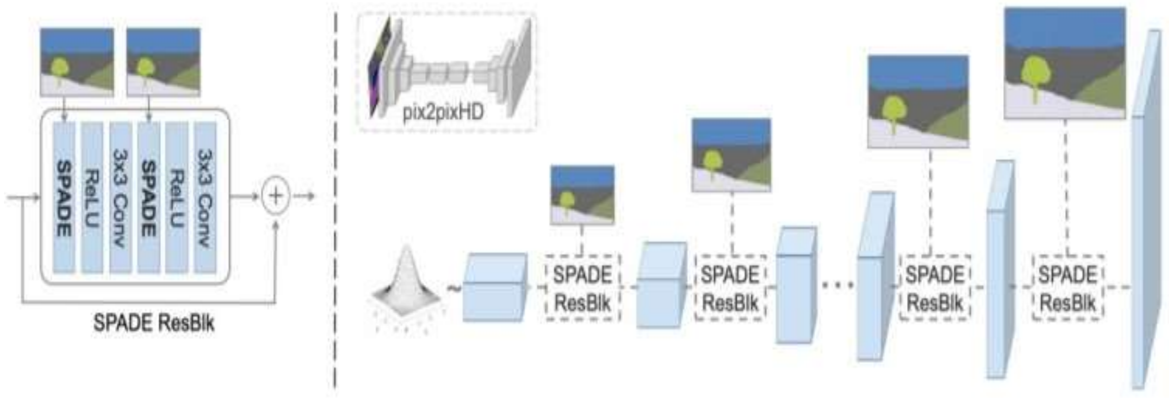


Figure 12: The network consist SPADE [30].

3.4 Image Animation First order model:

By decoupling the existing and movement information via self-controlled training, Siarohin et al [33]. developed a unique architecture. Their method is based on picture animation, which is the process of creating a video sequence from a source image and a driving video. Their study is motivated by the desire to avoid employing pre-trained models for exact point estimate, such as facial structure identification. As a result, Monkey net [32] uses a movement estimation module to recognize key points based on a self-controlled training strategy to alleviate the above restriction. Figure 13 depicts the monkey net network architecture, which established a framework for movement-driven picture animation by integrating the movement of a playing video with the appearance of a source image to make videos. An exact point detector network for removing object key points, a dense movement network for creating heavy heat maps, and a movement transfer network for integrating the showing of the source image with the motion of a moving video are the three primary networks in their model. Because of the large pose variations, monkey net has a severe flaw: poor generation quality [6]. They employ the same monkey network for motion estimation in the first order model study, but they add an additional local harmonic transformation to overcome the restrictions of the net [3].

An image as the source and frame from a moving video are also used as input in the first order model.

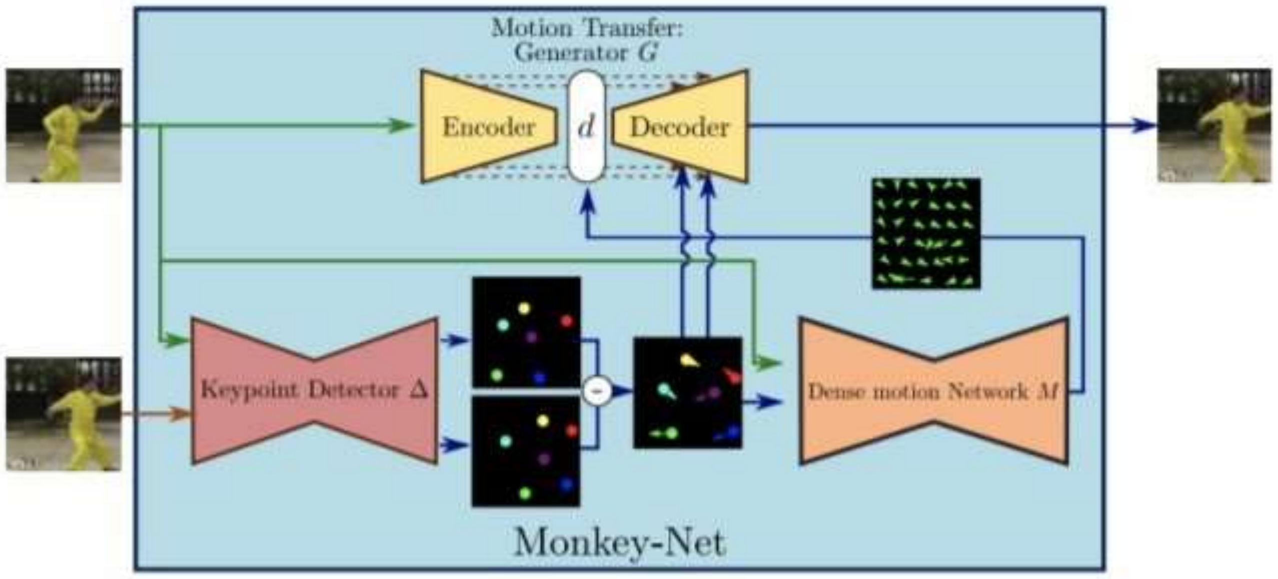


Figure: 13 A monkey net example [32]

Based on the heavy migration of faces from the driving frame to the source image, the key point detector determines sparse key points. The generator takes input which is the source image and generates the target image based on driving frame expressions using heavy motion and the clogging map as conditional input. They use a reconstruction loss and an equivariance loss to train their network from start to finish. [4] The equivariance constraint is used to accurately locate sparse key points.

The VoxCeleb [34] dataset is used to train and assess their network. The following are the loss functions:

$$L_{rec} = \Sigma(G(x) - gt)$$

Where gt is ground truth image and G(x) is generated image

$$L_{eqv} = {}_{k=1}{}^{K}\Sigma||\ g(x_k{}^{1}, y_k{}^{1}) - (x_k, y_k)\ ||^2$$

Where g(xk,yk) is a thin walled linear distortion vector transformation, (xk,yk) are monuments, and K is the total number of monuments.

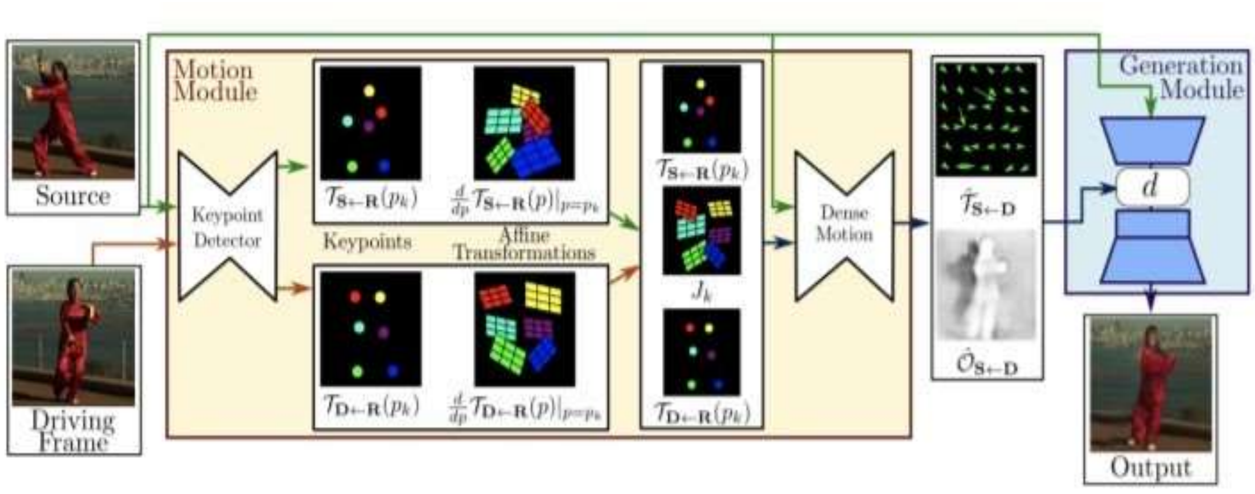

Source
Driving Frame
Motion Module
Keypoint Detector
$\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}}(p_k)$
$\frac{d}{dp}\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}}(p)|_{p=p_k}$
Keypoints
Affine Transformations
$\mathcal{T}_{\mathbf{D}\leftarrow\mathbf{R}}(p_k)$
$\frac{d}{dp}\mathcal{T}_{\mathbf{D}\leftarrow\mathbf{R}}(p)|_{p=p_k}$
J_k
Dense Motion
$\hat{\mathcal{T}}_{\mathbf{S}\leftarrow\mathbf{D}}$
$\hat{\mathcal{O}}_{\mathbf{S}\leftarrow\mathbf{D}}$
Generation Module
d
Output

Figure 14: General idea of first order model for image animation approach [35].

CHAPTER 4

Algorithms

4.1 Uploading the dataset:

By making edited movies of politicians and actors/actresses, recent advances in generating of fake image and video have raised worries in society. Since a result, if the current trend continues, it may become increasingly difficult to trust digital content, as the transmission of misleading information and fake news may become routine. Faces, as we all know, play a crucial role in communicating a message. As a result, deep fakes are becoming a major source of concern today's society.

Face manipulation can be divided into two types, as stated in Section 2, face expression manipulation and face identity modification. Face2Face, Deep fakes, Face swap, and Neural Textures are among the face alteration methods used in the FaceForesiscs++ dataset.

```
ALL_DATASETS = ['original', 'DeepFakeDetection_original',
'Deepfakes','DeepFakeDetection','Face2Face', 'FaceSwap','NeuralTextures']
```

There are 6 types of methods which are used for creating face forensics datasets:

4.1.1 Face swap:

It's a graphics based method for transferring the source image's face to the target video. This is accomplished by extracting facial features from the source image, fitting the 3D model with blend shapes, and projecting the result on the target image. To create a realistic-looking false image, image and color correction are done to integrate the morphing face into the source image. This method is performed for each frame in order to create a video.

4.1.2 Deep Fakes:

Redditor used an encoder-decoder network to generate the first deep fake video. Face swap’s GitHub and Fakeapp's implementations are both available. Two encoder-decoder networks are used in the process. In the phase of training, the two networks use the same encoder but distinct decoders. Figure 16 shows how the face of class A employs encoder from A and decoder from B weights to construct fakes during the testing process.

4.1.3 Face-to-face:

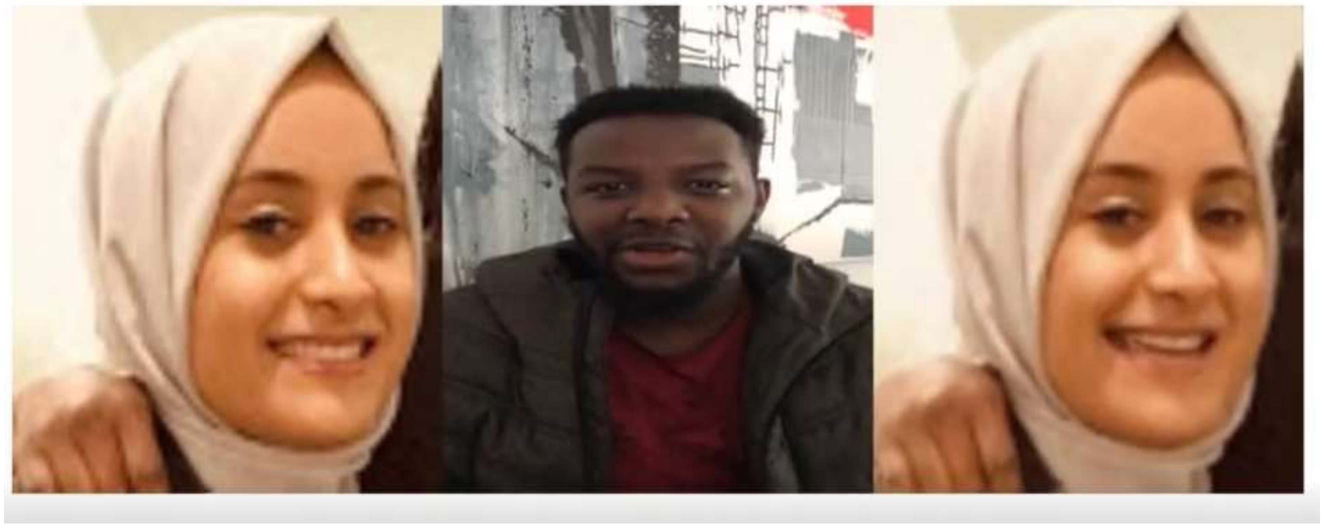
The Face2Face approach is part of the facial renewal category, and it seeks to transfer the source image's facial expressions to the target image. They employ a 3D model with blend shape coefficients to transfer face features and details from one image to another in this method. They use the first frame to create a fake emotion and the other frame key points to create a temporary facial identity.

4.1.4 Neural Texture:

This textures are also part of the face renewal category, as it is based on the Neural Textures recall approach described in the work. Using reconstruction loss and adversarial loss, they use GANs to generate fakes.

4.1.5 Post the process of videos:

The videos after the normal process are post processed to improve video quality by compressing them, which is commonly used by social media sites. In this study, a quantization is used to create high quality videos and a rate of 40 is used to create low-quality videos.

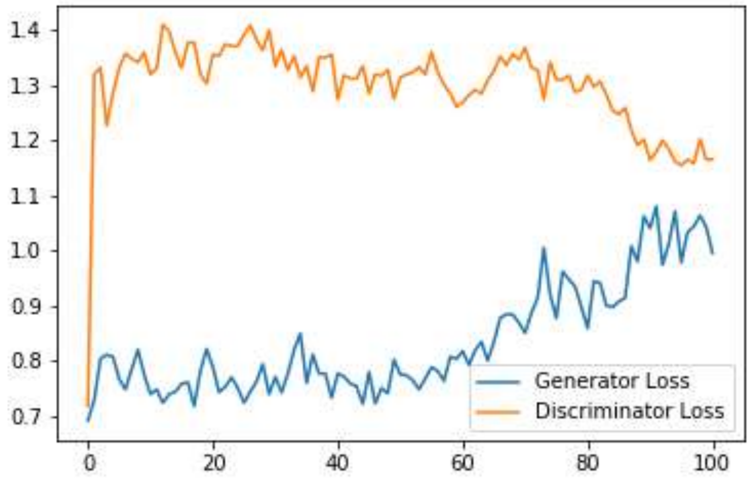


4.2 Model:

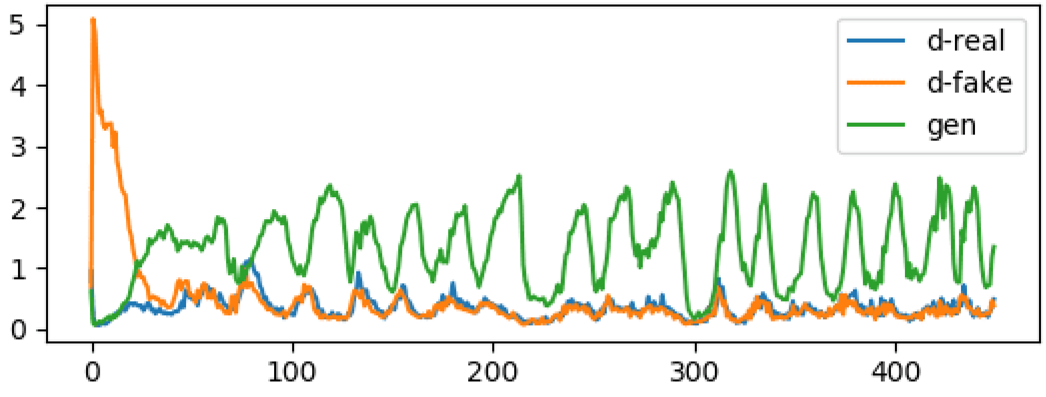
After uploading the dataset the training model comes in using torch as a CNN method and using torchvision.models as models after importing we start with checking the models block then printing the models parameters and getting the trained parameters finishing with creating the model when creating the model we use resnet18 as the model then going to the hidden layer the used model will have torch.Sequential, torch.Dropout, torch.Linear, torch.ReLU and torch.BatchNorm1d

4.3 Training:

In the training process numpy, tourch, apex, tqdm is required a Records class was created having attributes writing and returning and getting in the training function the loss and accs of the training is calculated. For full code check Appendix1.

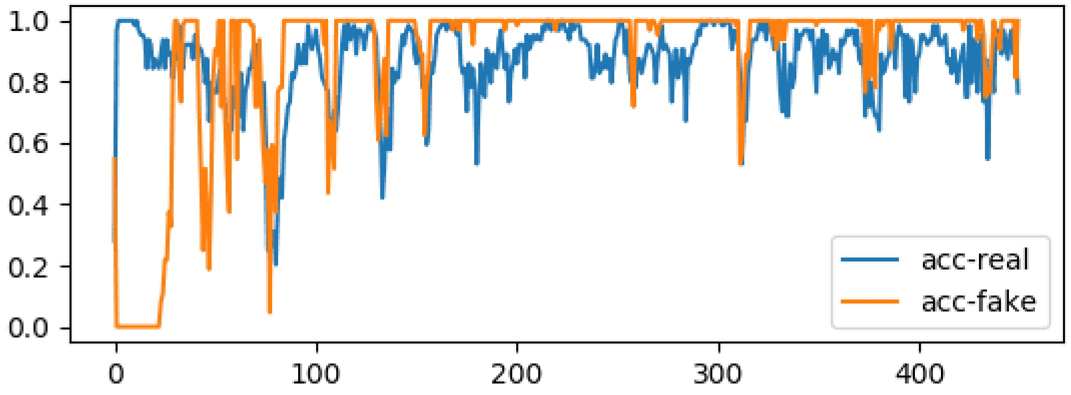


As showing in the matlab diagram the loss function for the generator and discriminator



In this chart it is showing the accuracy of detecting the fake and real images

Once the generator gets better the discriminator preform worse because of the similarity that the discriminator is not able to detect it.



4.4 Compare and Contrast

The results of this study demonstrate that VGG-19 can be a suitable choice not only for partial face images, but also for full-face images confirming the findings of [57]. The better performance of VGG-19 is because it is pretrained on a wide variety of objects. AP was used as an aggregation function to summarize the precision–recall curve into a single value that represents the average of all precisions. VGG-19, even though it had the highest accuracy, had the lowest AP of 95% in comparison to all other analyzed models. The DenseNet architecture on the original dataset and grayscale dataset had a closer performance to VGG-19, with 94% accuracy. Results from DenseNET architecture demonstrates that gray channel-based analysis does not have a huge impact on model accuracy level in classifying images into the two categories of real and fake. The DenseNet architecture, even though was second best in terms of performance, achieved an AP of 99% on both augmented and grayscale datasets, which is slightly in contrast to the results found in [59] in terms of precision rate; however, it aligns with claims regarding detection time. Custom CNN architecture had the lowest accuracy level (89%). The second-highest AP score after DenseNet was the Custom CNN model. Augmented input reduced model performance and accuracy level on both DenseNET and Custom CNN by 5–22%. However, the Custom CNN had a better performance on augmented data in comparison to the DenseNet architecture.

CHAPTER 5

Conclusion

5.1 Discussion:

For generative models, creating video sequences has proven to be a difficult task.

Many previous works produced realistic videos, but they lacked generalization to unanticipated domains and were not temporally coherent. A unique architecture consisting of an LSTM-based generator, a discriminator, and a dense motion network was proposed. By merging the appearance of the source picture with the motion of the driving video, we show that our model can generate realistic films given a single image and a target video. We also demonstrated that our methodology applies to previously unknown videos.

5.2 Future work:

This thesis' main focus is on creating datasets for deepfake detection issues. Our algorithm creates realistic videos using a face source image and target video frame facial emotions.

- Train and test the network on other domains. These are some prospective extensions of our study. Some previous video generating works, such as vid2vid, few-show vid2vid, and first order model for picture animation, produced films in a variety of domains, including faces, human poses, and semantic segmentation tasks. For making realistic films, we can extend our network to train and test on human pose data.

- Utilizing the generator network proposed by pix2pixHD, we can make high-resolution videos using local and global generators. They used both local and global generators to create graphics with a resolution of 2K.
- Assess created videos by submitting both original and generated videos to Amazon Mechanical Turk, where human reviewers will assess them.
- We'd like to lay out a psychophysical study that evaluates the fairness of various deep fake techniques for evaluation. Using our model, first order model, and few shot vid2vid model, we would choose 50 original videos and their generated videos at random.

Refrences:

[1] - Thanh Thi Nguyen, Cuong M. Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen. Deep Learning for Deepfakes Creation and Detection. Institute for Intelligent Systems Research and Innovation, Deakin University, Australia.

[2] - Schroepfer, M. (2019, September 5). Creating a data set and a challenge for deepfakes. Retrieved from https://ai.facebook.com/blog/deepfake-detection-challenge.

[3] - Guera, D., and Delp, E. J. (2018, November). Deepfake video detection using recurrent neural networks. In 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS) (pp. 1-6). IEEE.

[4] - Large-scale CelebFaces Attributes (CelebA) Dataset. Database retrieved from http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html.

[5] - Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess. FaceForensics++: Learning to Detect Manipulated Facial Images. 26 August, 2019.

[6] - Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess. FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces. 24 march, 2018.

[7] - DeepFaceLab for generating deepfake videos. Retrieved from https://github.com/iperov/DeepFaceLab.

[8] - Face swap. Retrieved from https://github.com/deepfakes/faceswap. Department of Computer Engineering 58

[9] - Few shot face translation. Retrieved from https://github.com/shaoanlu/fewshot-facetranslation-GAN.

[10] - FakeApp 2.2.0. Retrieved from https://www.malavida.com/en/soft/fakeapp/

[11] - Faceswap-GAN. Retrieved from https://github.com/shaoanlu/faceswap-GAN.

[12] - VidTIMIT database. Retrieved from http://conradsanderson.id.au/vidtimit/

[13] - Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 815-823).

[14] - Nguyen, H. H., Yamagishi, J., and Echizen, I. (2019, May). Capsule-forensics: Using capsule networks to detect forged images and videos. In 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 2307-2311). IEEE.

[15] - Rossler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., and Niener, M. (2018). FaceForensics: A large-scale video dataset for forgery detection in human faces. arXiv preprint arXiv:1803.09179.

[16] - Pavel Korshunov and Sebastien Marcel Idiap Research Institute, Martigny, Switzerland. Vulnerability assessment and detection of Deepfake videos.

[17] - Yuezun Li, Siwei Lyu. Exposing DeepFake Videos By Detecting Face Warping Artifacts. University at Albany, State University of New York, USA. CVPR. Department of Computer Engineering 59

[18] - Shruti Agarwal and Hany Farid. Protecting World Leaders Against Deepfakes. University of California, Berkeley Berkeley CA, USA.

[19] - Darius Afchar ; Vincent Nozick ; Junichi Yamagishi. MesoNet: a Compact Facial Video Forgery Detection Network. IEEE Xplore.

[20] - Marissa Koopman, Andrea Macarulla Rodriguez, Zeno Geradts. Detection of Deepfake Video Manipulation. University of Amsterdam & Netherlands Forensic Institute. IMVIP 2018.

[21] - Ian J. Goodfellow, Jean Pouget-Abadie , Mehdi Mirza, Bing Xu, David Warde-Farley. Generative Adversarial Nets.

[22] - Karen Simonyan & Andrew Zisserman. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. University of Oxford.

[23] - Brian Dolhansky, Russ Howes, Ben Pflaum, Nicole Baram, Cristian Canton Ferrer. The Deepfake Detection Challenge (DFDC) Preview Dataset. AI Red Team, Facebook AI.

[24] - Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012.

[25] - Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet. Going deeper with convolutions.

[26] - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. Microsoft Research.

[27] - Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao. Video-to-Video Synthesis. NVIDIA, MIT CSAIL Research. Department of Computer Engineering 60

[28] - Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. NVIDIA UC Berkeley Research.

[29] - Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao. Few-shot Videoto-Video Synthesis. NVIDIA Corporation.

[30] - Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Nets. University of California, Berkeley, CVPR, 2017.

[31] - Taesung Park, Ming-Yu Liu, Ting-Chun Wang, Jun-Yan Zhu. Semantic Image Synthesis with Spatially-Adaptive Normalization. NVIDIA UC Berkeley Research.

[32] - Aliaksandr Siarohin, Stephane Lathuiliere, Sergey Tulyakov, Elisa Ricci. Animating Arbitrary Objects via Deep Motion Transfer. University of Trento, Italy.

[33] - Aliaksandr Siarohin, Stephane Lathuiliere, Sergey Tulyakov, Elisa Ricci. First Order Motion Model for Image Animation. University of Trento, Italy.

[34] - Arsha Nagrani, Joon Son Chung, Andrew Zisserman. VoxCeleb: a large-scale speaker identification dataset. University of Oxford, UK.

[35] - http://www.robots.ox.ac.uk/~vgg/data/voxceleb/ . VoxCeleb dataset download blog.

[36] - Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, Jaegul Choo. StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation. Department of Computer Engineering 61

[37] - Phillip Isola, Jun-Yan Zhu, Alexei A.Efros. Image-to-Image Translation with Conditional Adversarial Networks

[38] - Ian J. Goodfellow, Jean Pouget-Abadie† , Mehdi Mirza, Bing Xu, David Warde-Farley. Generative Adversarial Nets. [39] - Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg , Philip Hausser, Caner Hazırbas, Vladimir Golkov. FlowNet: Learning Optical Flow with Convolutional Networks.

[40] - Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, Thomas Brox. FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks. University of Freiburg, Germany.

[41] - Justus Thies1 Michael Zollhofer, Marc Stamminger, Christian Theobalt, Matthias Nießner. Face2Face: Real-time Face Capture and Reenactment of RGB Videos.

[42] - Justus Thies, Michael Zollhofer, and Matthias Nießner. De- ¨ ferred neural rendering: Image synthesis using neural textures. ACM Transactions on Graphics 2019 (TOG), 2019.

[43] - Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Hausser. FlowNet: Learning Optical Flow with Convolutional Networks.

[44] - https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/. Facial landmark detection methods. [45] - Vahid Kazemi and Josephine Sullivan. One Millisecond Face Alignment with an Ensemble of Regression Trees. KTH, Royal Institute of Technology.

[46] - Mamata S. Kalas. REAL TIME FACE DETECTION AND TRACKING USING OPENCV. Department of Computer Engineering 62

[47] - https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

[48] - https://deepfakedetectionchallenge.ai/

[49] - https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activationfunctions-right/

[50] - Justus Thies, Michael Zollhofer, Marc Stamminger. Face2Face: Real-time face capture and reenactment of RGB videos. [51] - Justus Thies, Matthias Nießner, Michael Zollhofer. Deferred Neural Rendering: Image Synthesis using Neural Textures. [52] - Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.

[53] - Yunjey Choi, Minje Choi, Shungun Kin. StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation.

Appendix1

train.py

```
import
torch
        from tqdm import tqdm
        from apex import amp
        import numpy as np
        import os

        from utils import visualize_metrics, display_predictions_on_image
        from sklearn.metrics import roc_auc_score as extra_metric

        import foundations



        class Records:
            def __init__(self):
                self.train_losses, self.train_losses_wo_dropout, self.base_val_losses,
        self.augment_val_losses = [], [], [], []
                self.train_accs, self.train_accs_wo_dropout, self.base_val_accs, self.augment_val_accs
        = [], [], [], []
                self.train_custom_metrics, self.train_custom_metrics_wo_dropout,
        self.base_val_custom_metrics, self.augment_val_custom_metrics = [], [], [], []
                self.lrs = []

            def write_to_records(self, **kwargs):
                assert len(set(kwargs.keys()) - set(self.__dir__())) == 0, 'invalid arguments!'
                for k, v in kwargs.items():
                    setattr(self, k, v)

            def return_attributes(self):
                attributes = [i for i in self.__dir__() if not (i.startswith('__') and
        i.endswith('__') or i in ('write_to_records', 'return_attributes',

        'get_metrics'))]
                return attributes

            def get_metrics(self):
```

```
        return ['train_accs_wo_dropout', 'base_val_accs', 'augment_val_accs',
'base_val_custom_metrics', 'augment_val_custom_metrics']


def train_one_epoch(epoch, model, train_dl, max_lr, optimizer, criterion, scheduler, records):
    model.train()
    train_loss = 0
    train_loss_eval = 0
    train_tk = tqdm(train_dl, total=int(len(train_dl)), desc="Train Epoch")

    optimizer.zero_grad()
    total = 0
    correct_count = 0
    correct_count_eval = 0

    for step, data in enumerate(train_tk):
        model.train()
        inputs = data['image']
        labels = data['label'].view(-1)

        inputs = inputs.cuda(device=0)
        labels = labels.cuda(device=0)

        optimizer.zero_grad()
        with torch.set_grad_enabled(True):
            outputs = model(inputs)

            _, predicted = torch.max(outputs.data, 1)

            total += labels.size(0)

            correct_count += (predicted == labels).sum().item()

            loss = criterion(outputs, labels)
            with amp.scale_loss(loss, optimizer) as scaled_loss:
                scaled_loss.backward()
            #              loss.backward()
            optimizer.step()
            if scheduler is not None:
                records.lrs += scheduler.get_lr()
                scheduler.step()
            else:
                records.lrs.append(max_lr)
            train_loss += loss.item()
            train_tk.set_postfix(loss=train_loss / (step + 1), acc=correct_count / total)

            # eval with dropout turned off
            model.eval()
            with torch.no_grad():
                outputs_eval = model(inputs)

                _, predicted_eval = torch.max(outputs_eval.data, 1)

                correct_count_eval += (predicted_eval == labels).sum().item()

                loss_eval = criterion(outputs_eval, labels)

            train_loss_eval += loss_eval.item()

        records.train_losses_wo_dropout.append(train_loss_eval / (step + 1))
        records.train_accs_wo_dropout.append(correct_count_eval / total)

        records.train_losses.append(train_loss / (step + 1))
        records.train_accs.append(correct_count / total)

        print(f'Epoch {epoch}: train loss={records.train_losses[-1]:.4f} | train
acc={records.train_accs[-1]:.4f}')

        print(f'Epoch {epoch}: eval_ loss={records.train_losses_wo_dropout[-1]:.4f} | train
acc={records.train_accs_wo_dropout[-1]:.4f}')


def validate(model, val_dl, criterion, records, data_name):
    # val
    model.eval()
    val_loss = 0
    correct_count = 0
    total = 0

    all_labels = []
    all_predictions = []

    for data in val_dl:
        inputs = data['image']
        labels = data['label'].view(-1)
```

```
            inputs = inputs.cuda(device=0)  # .type()
            labels = labels.cuda(device=0)

            with torch.no_grad():
                outputs = model(inputs)
                _, predicted = torch.max(outputs.data, 1)

                total += labels.size(0)
                correct_count += (predicted == labels).sum().item()
                val_loss += criterion(outputs, labels)

            all_labels.append(labels.cpu().numpy())
            all_predictions.append(predicted.cpu().numpy())

        all_labels = np.concatenate(all_labels, axis=0)
        all_predictions = np.concatenate(all_predictions, axis=0)
        extra_score = extra_metric(all_labels, all_predictions)

        if data_name == 'base':
            records.base_val_losses.append(val_loss / len(val_dl))
            records.base_val_accs.append(correct_count / total)
            records.base_val_custom_metrics.append(extra_score)
            print(f'\t base val loss={records.base_val_losses[-1]:.4f} | base val
acc={records.base_val_accs[-1]:.4f} | '
                  f'base val {extra_metric.__name__}={records.base_val_custom_metrics[-1]:.4f}')

        else:
            assert data_name == 'augment', f'specified data type is unknown {data_name}'
            records.augment_val_losses.append(val_loss / len(val_dl))
            records.augment_val_accs.append(correct_count / total)
            records.augment_val_custom_metrics.append(extra_score)
            print(f'\t augment val loss={records.augment_val_losses[-1]:.4f} |  augment val
acc={records.augment_val_accs[-1]:.4f} | '
                  f'augment val {extra_metric.__name__}={records.augment_val_custom_metrics[-
1]:.4f}\n')


def train(train_dl, val_base_dl, val_augment_dl, display_dl_iter, model, optimizer, n_epochs,
max_lr, scheduler, criterion, train_source):
    records = Records()
    best_metric = 0.
    os.makedirs('checkpoints', exist_ok=True)

    for epoch in range(n_epochs):
        train_one_epoch(epoch, model, train_dl, max_lr, optimizer, criterion, scheduler,
records)
        validate(model, val_base_dl, criterion, records, data_name='base')
        validate(model, val_augment_dl, criterion, records, data_name='augment')

        if train_source == 'both':
            selection_metric = [getattr(records, 'base_val_accs')[-1], getattr(records,
'augment_val_accs')[-1]]
            selection_metric = np.mean(selection_metric)

        else:
            selection_metric = getattr(records, f"{train_source}_val_accs")[-1]

        if selection_metric >= best_metric:
            print(f'>>> Saving best model metric={selection_metric:.4f} compared to previous
best {best_metric:.4f}')
            checkpoint = {'model': model,
                          'state_dict': model.state_dict(),
                          'optimizer': optimizer.state_dict()}

            torch.save(checkpoint, 'checkpoints/best_model.pth')
            foundations.save_artifact('checkpoints/best_model.pth',
key='pretrained_model_checkpoint')

        display_filename = f'{epoch}_display.png'
        display_predictions_on_image(model, val_base_dl.dataset.cached_path, display_dl_iter,
name=display_filename)

        # Save eyeball plot to Atlas GUI
        foundations.save_artifact(display_filename, key=f'{epoch}_display')

        # Save metrics plot
        visualize_metrics(records, extra_metric=extra_metric, name='metrics.png')

        # Save metrics plot to Atlas GUI
        foundations.save_artifact('metrics.png', key='metrics_plot')

    # Log metrics to GUI
    if train_source == 'both':
        avg_metric = [getattr(records, 'base_val_accs'), getattr(records, 'augment_val_accs')]
        avg_metric = np.mean(avg_metric, axis=0)
        max_index = np.argmax(avg_metric)

    else:
        max_index = np.argmax(getattr(records, f"{train_source}_val_accs"))

    useful_metrics = records.get_metrics()
    for metric in useful_metrics:
        foundations.log_metric(metric, float(getattr(records, metric)[max_index]))
```

model.py

```
import
torchvision.models
as models
import torch.nn as nn


def check_model_block(model):
    for name, child in model.named_children():
        print(name)


def print_model_params(model):
    pytorch_total_params = sum(p.numel() for p in model.parameters())
    print(f"total number of params: {pytorch_total_params:,}")
    return pytorch_total_params


def get_trainable_params(model):
    print("Params to learn:")
    params_to_update = []
    for name, param in model.named_parameters():
        if param.requires_grad:
            print("\t", repr(name))
            params_to_update.append(param)

    return params_to_update
```

```
def create_model(use_hidden_layer, dropout):
    model = models.resnet18(pretrained=True)

    # Uncomment to freeze pre-trained layers
    # for param in model.parameters():
    #     param.requires_grad = False

    in_features = model.fc.in_features
    print(f"Input feature dim: {in_features}")

    if use_hidden_layer:
        model.fc = nn.Sequential(
            nn.Dropout(dropout),
            nn.Linear(in_features, in_features // 2),
            nn.ReLU(),
            nn.BatchNorm1d(in_features // 2),
            nn.Dropout(dropout),
            nn.Linear(in_features // 2, 2)
        )

    else:
        model.fc = nn.Sequential(
            nn.Dropout(dropout),
            nn.Linear(in_features, 2)
        )

    print(model)

    model = model.cuda()
    return model
```

main.py

```
import
numpy
as np
         import torch
         import torch.nn as nn
         import torch.optim as optim
         from torch.optim import lr_scheduler
         from apex import amp

         from data_loader import create_dataloaders
         from model import get_trainable_params, create_model, print_model_params
         from train import train
         from utils import parse_and_override_params

         import foundations


         # Fix random seed
         torch.manual_seed(0)
         np.random.seed(0)
         torch.backends.cudnn.deterministic = True
         torch.backends.cudnn.benchmark = False

         params = foundations.load_parameters()

         data_dict = parse_and_override_params(params)

         # Set job tags to easily spot data in use
         foundations.set_tag(f'{data_dict[params["train_data"]]}: {params["train_data"]}')
         # foundations.set_tag(f'big {params["train_data"]}')

         print('Creating datasets')
         # Get dataloaders
         train_dl, val_base_dl, val_augment_dl, display_dl_iter = create_dataloaders(params)

         print('Creating loss function')
         # Loss function
         criterion = nn.CrossEntropyLoss()

         print('Creating model')
         # Create model, freeze layers and change last layer
         model = create_model(bool(params['use_hidden_layer']), params['dropout'])
         _ = print_model_params(model)
         params_to_update = get_trainable_params(model)

         print('Creating optimizer')
         # Create optimizer and learning rate schedules
         optimizer = optim.Adam(params_to_update, lr=params['max_lr'],
         weight_decay=params['weight_decay'])
         model, optimizer = amp.initialize(model, optimizer, opt_level='O1', verbosity=0)
```

```
# Learning rate scheme
if bool(params['use_lr_scheduler']):
    step_size_up = int(params["n_epochs"] * len(train_dl) * 0.3)
    step_size_down = params["n_epochs"] * len(train_dl) - step_size_up
    scheduler = lr_scheduler.OneCycleLR(optimizer, params["max_lr"], total_steps=None,
                                        epochs=params["n_epochs"],
steps_per_epoch=len(train_dl),
                                        pct_start=params['pct_start'], anneal_strategy='cos',
                                        cycle_momentum=False)
else:
    scheduler = None

print("Training start..")
# Train
train(train_dl, val_base_dl, val_augment_dl, display_dl_iter, model, optimizer,
params["n_epochs"], params["max_lr"], scheduler, criterion,
      train_source=params["train_data"])
```

data_loader.py

```
from
pathlib
import
Path
          import numpy as np

          import dlib
          import torch
          from PIL import Image
          from torch.utils.data import Dataset
          from torchvision import transforms
          from torch.utils.data import DataLoader


          from utils import load_and_preprocess_image


          def collate_fn(batch):
              imgs = [item["image"] for item in batch if item["image"] is not None]
```

```
    targets = [item['label'] for item in batch if item['image'] is not None]
    filenames = [item['filename'] for item in batch if item['image'] is not None]
    imgs = torch.stack(imgs)
    targets = torch.stack(targets)
    return {'image': imgs, 'label': targets, 'filename': filenames}


def get_transforms():
    pre_trained_mean, pre_trained_std = [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]

    train_transforms = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomAffine(degrees=40, scale=(.9, 1.1), shear=0),
        transforms.RandomPerspective(distortion_scale=0.2),
        transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5),
        transforms.ToTensor(),
        transforms.RandomErasing(scale=(0.02, 0.16), ratio=(0.3, 1.6)),
        transforms.Normalize(mean=pre_trained_mean, std=pre_trained_std),
    ])

    val_transforms = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=pre_trained_mean, std=pre_trained_std)
    ])
    return train_transforms, val_transforms


class FFDataset(Dataset):
    def __init__(self, filenames, filepath, transform, output_image_size=224,
recompute=False):
        self.filenames = filenames
        self.transform = transform
        self.image_size = output_image_size
        self.recompute = recompute

        self.cached_path = Path(filepath)
        self.cached_path.mkdir(exist_ok=True)
        self.face_detector = dlib.get_frontal_face_detector()

    def __len__(self):
        return len(self.filenames)

    def __getitem__(self, idx: int):
        filename = self.filenames[idx]
        image_id = filename.stem
        filename = str(filename)
        label = 1 if "fake" in filename.split("/") else 0

        preprocessed_filename = self.cached_path / f"processed_{image_id}.npy"

        if preprocessed_filename.is_file() and not self.recompute:
            image = np.load(preprocessed_filename)
        else:
            image = load_and_preprocess_image(filename, self.image_size, self.face_detector)
            if image is None:
                image = []
            np.save(preprocessed_filename, image)

        if len(image) == 0:
            return {'image': None, "label": None ,'filename": filename}

        image = Image.fromarray(image)
        image = self.transform(image)
        label = torch.tensor(label)

        return {'image': image, "label': label, 'filename': filename}


def create_dataloaders(params):
    train_transforms, val_transforms = get_transforms()
    train_dl = _create_dataloader(f"/datasets/{params["train_data"]}_deepfake", mode="train",
batch_size=params["batch_size"],
                                  transformations=train_transforms,
sample_ratio=params["sample_ratio"])
    val_base_dl = _create_dataloader(f"/datasets/base_deepfake/val", mode="val",
batch_size=params["batch_size"], transformations=val_transforms,
                                     sample_ratio=params["sample_ratio"])
    val_augment_dl = _create_dataloader(f"/datasets/augment_deepfake/val", mode='val",
batch_size=params["batch_size"], transformations=val_transforms,
                                        sample_ratio=params["sample_ratio"])
    display_file_paths = [f"/datasets/{i}_deepfake/val" for i in ["base", "augment"]]
    display_dl_iter = iter(_create_dataloader(display_file_paths, mode="val", batch_size=32,
transformations=val_transforms,
                                              sample_ratio=params["sample_ratio"]))

    return train_dl, val_base_dl, val_augment_dl, display_dl_iter
```

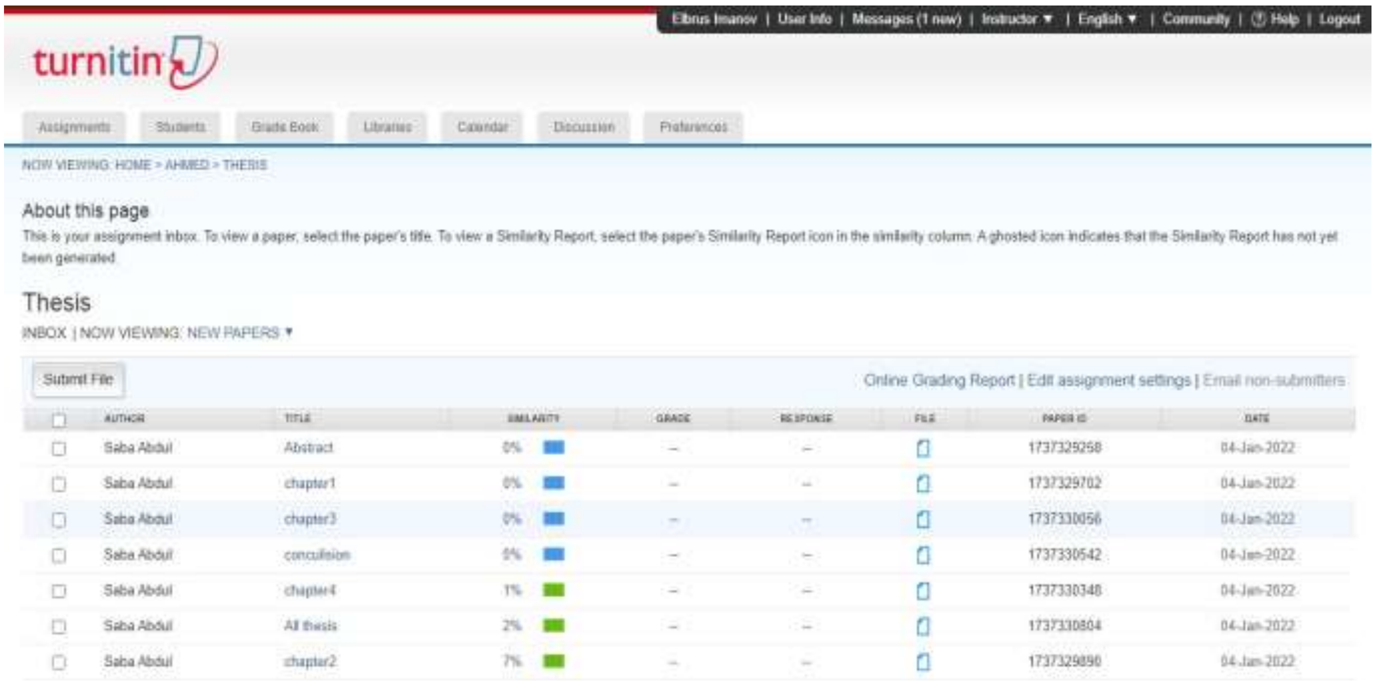
```
def _create_dataloader(file_paths, mode, batch_size, transformations, sample_ratio,
num_workers=80):
    if not isinstance(file_paths, list):
        file_paths = [file_paths]

    filenames = []
    for file_path in file_paths:
        data_path = Path(file_path)

         real_frame_filenames = _find_filenames(data_path / 'real/frames/', '*.png')
         fake_frame_filenames = _find_filenames(data_path / 'fake/frames/', '*.png')

        filenames += real_frame_filenames
        filenames += fake_frame_filenames

    assert len(filenames) != 0, f'filenames are empty {filenames}'
    np.random.shuffle(filenames)

    if mode == 'train':
        filenames = filenames[:int(sample_ratio * len(filenames))]

    ds = FFDataset(filenames, filepath=f'/datasets/precomputed/', transform=transformations,
recompute=False)
    dl = DataLoader(ds, batch_size=batch_size, num_workers=num_workers, shuffle=True,
collate_fn=collate_fn)

    print(f"{mode} data: {len(ds)}")

    return dl


def _find_filenames(file_dir_path, file_pattern): return
list(file_dir_path.glob(file_pattern))
```

utils.py

```
impo
rt
cv2
        import torch
        import numpy as np
        from pathlib import Path

        import matplotlib.pyplot as plt

        import foundations


        def get_boundingbox(face, width, height, scale=1.3, minsize=None):
            x1 = face.left()
            y1 = face.top()
            x2 = face.right()
            y2 = face.bottom()
            size_bb = int(max(x2 - x1, y2 - y1) * scale)
            if minsize:
                if size_bb < minsize:
                    size_bb = minsize
            center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2

            # Check for out of bounds, x-y top left corner
            x1 = max(int(center_x - size_bb // 2), 0)
            y1 = max(int(center_y - size_bb // 2), 0)
            # Check for too big bb size for given x, y
            size_bb = min(width - x1, size_bb)
            size_bb = min(height - y1, size_bb)

            return x1, y1, size_bb


        def load_and_preprocess_image(image_filename, output_image_size, face_detector):
            image = cv2.imread(image_filename)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

            cropped_image = get_face_crop(face_detector, image)
            if cropped_image is None:
                return None

            resized_image = cv2.resize(cropped_image, (output_image_size, output_image_size))
            return resized_image
```

```
def get_face_crop(face_detector, image):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    faces = face_detector(gray, 1)

    height, width = image.shape[:2]

    if len(faces) == 0:
        return None
    else:
        face = faces[0]
        x, y, size = get_boundingbox(face, width, height)
        cropped_face = image[y:y + size, x:x + size]
        return cropped_face



def visualize_metrics(records, extra_metric, name):
    fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(15, 6))
    axes[0].plot(list(range(len(records.train_losses))), records.train_losses, label='train')
    axes[0].plot(list(range(len(records.train_losses_wo_dropout))),
records.train_losses_wo_dropout, label='train w/o dropout')
    axes[0].plot(list(range(len(records.base_val_losses))), records.base_val_losses,
label='base_val')
    axes[0].plot(list(range(len(records.augment_val_losses))), records.augment_val_losses,
label='augment_val')
    axes[0].set_title('loss')
    axes[0].legend()

    axes[1].plot(list(range(len(records.train_accs))), records.train_accs, label='train')
    axes[1].plot(list(range(len(records.train_accs_wo_dropout))),
records.train_accs_wo_dropout, label='train w/o dropout')
    axes[1].plot(list(range(len(records.base_val_accs))), records.base_val_accs,
label='base_val')
    axes[1].plot(list(range(len(records.augment_val_accs))), records.augment_val_accs,
label='augment_val')
    axes[1].axhline(y=0.5, color='g', ls='--')
    axes[1].axhline(y=0.667, color='r', ls='--')
    axes[1].set_title('acc')
    axes[1].legend()

    axes[2].plot(list(range(len(records.train_custom_metrics))), records.train_custom_metrics,
label='train')
```

```
    axes[2].plot(list(range(len(records.train_custom_metrics_wo_dropout))),
records.train_custom_metrics_wo_dropout, label="train w/o dropout')
    axes[2].plot(list(range(len(records.base_val_custom_metrics))),
records.base_val_custom_metrics, label='base_val")
    axes[2].plot(list(range(len(records.augment_val_custom_metrics))),
records.augment_val_custom_metrics, label='augment_val")
    axes[2].axhline(y=0.5, color="g", ls="--")
    axes[2].axhline(y=0.5, color="r", ls="--")
    axes[2].set_title(f"{extra_metric.__name__}")
    axes[2].legend()

    axes[3].plot(list(range(len(records.lrs))), records.lrs)
    _ = axes[3].set_title("lr")
    plt.tight_layout()
    plt.savefig(name, format="png")


def display_predictions_on_image(model, precomputed_cached_path, val_iter, name):
    # val
    model.eval()

    data = next(val_iter)

    inputs = data["image"]
    labels = data["label"].view(-1)
    filenames = data["filename"]

    inputs = inputs.cuda(device=0)
    labels = labels.cuda(device=0)

    with torch.no_grad():
        outputs = model(inputs)
        outputs_predicbilty = torch.nn.functional.softmax(outputs, dim=1)
        assert len(outputs_predicbilty) == len(outputs), f'proba shape:
{len(outputs_predicbilty)}'

        _, predicted = torch.max(outputs.data, 1)

    nrows = int(len(inputs) ** 0.5)
    ncols = int(np.ceil(len(inputs) / nrows))

    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(30, 40))
    step = 0
    for i in range(nrows):
        for j in range(ncols):
            image_id = Path(filenames[step]).stem
            face_crop = precomputed_cached_path / f'processed_{image_id}.npy'
            face_crop = np.load(face_crop)
            axes[i,
j].set_title(f'{outputs_predicbilty[step][0]:.2f},{outputs_predicbilty[step][1]:.2f}|{predicted
[step]}|{labels[step]}')
            axes[i, j].imshow(face_crop)
            step += 1
            if step == len(inputs):
                break
    plt.title('predicted_proba real, fake | prediction | label (0: real 1: fake)')
    plt.tight_layout()
    plt.savefig(name, format="png")


def parse_and_override_params(params):
    data_dict = {"base": 0, "augment": 1, "both": 2}

    parsed_params = params.copy()
    parsed_params["train_data"] = data_dict[params["train_data"]]
    foundations.log_params(parsed_params)
    return data_dict
```


Elbrus Imanov | User Info | Messages (1 new) | Instructor ▼ | English ▼ | Community | Help | Logout
turnitin
Assignments
Students
Grade Book
Libraries
Calendar
Discussion
Preferences

NOW VIEWING: HOME > AHMED > THESIS

About this page

This is your assignment inbox. To view a paper, select the paper's title. To view a Similarity Report, select the paper's Similarity Report icon in the similarity column. A ghosted icon indicates that the Similarity Report has not yet been generated.

Thesis

INBOX | NOW VIEWING: NEW PAPERS ▼

Submit File
Online Grading Report | Edit assignment settings | Email non-submitters

AUTHOR	TITLE	SIMILARITY	GRADE	RESPONSE	FILE	PAPER ID	DATE
Saba Abdul	Abstract	0% 🟦	--	--	▯	1737329258	04-Jan-2022
Saba Abdul	chapter1	0% 🟦	--	--	▯	1737329702	04-Jan-2022
Saba Abdul	chapter3	0% 🟦	--	--	▯	1737330056	04-Jan-2022
Saba Abdul	conculision	0% 🟦	--	--	▯	1737330542	04-Jan-2022
Saba Abdul	chapter4	1% 🟩	--	--	▯	1737330348	04-Jan-2022
Saba Abdul	All thesis	2% 🟩	--	--	▯	1737330804	04-Jan-2022
Saba Abdul	chapter2	7% 🟩	--	--	▯	1737329890	04-Jan-2022

Thesis Title: Deep fakes Generation Using LSTM based Generative Adversarial Networks.

Asst.Prof.Dr. Elbrus Imanov [illegible]