Quantifying Deep Fake Generation and Detection Accuracy for a Variety of Newscast Settings.

A Project Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

By

Pratikkumar Prajapati

May 2020

ABSTRACT

DeepFakes are fake videos synthesized using advanced deep-learning techniques. The fake videos are one of the biggest threats of our times. In this project, we attempt to detect DeepFakes using advanced deep-learning and other techniques. The whole project is two semesters long. This report is of the first semester. We begin with developing a Convolution Neural Network-based classifier to classify Devanagari characters. Then we implement various Autoencoders to generate fake Devanagari characters. We extend it to develop Generative Adversarial Networks to generate more realistic looking fake Devanagari characters. We explore a state-of-the-art model, DeepPrivacy, to generate fake human images by swapping faces. We modify the DeepPrivacy model and experiment with the new model to achieve a similar kind of results. The modified DeepPrivacy model can generate DeepFakes using face-swap. The generated fake images look like humans and have variations from the original set of images.

***Index Terms* — Artificial Intelligence, machine learning, deep-learning, autoencoders, generative adversarial networks, DeepFakes, face-swap, fake videos**

TABLE OF CONTENTS

I. INTRODUCTION

Artificial Intelligence (AI) has applications in many domains such as healthcare, automotive, finance, robotics, language processing, predictions, computer vision, and many more. Machine Learning (ML), a subfield of AI, learns about patterns from the input data and can predict the pattern in the unseen data. Deep-learning is a subfield of ML. It has been applied to many real-world problems like object detection, image classification, handwriting recognition, and many more [1, 2]. Advanced techniques like Autoencoders (AEs) [1] and Generative Adversarial Networks (GANs) [1, 2] can generate new data or alter the existing data so that they look like the original class of data. New images and videos can be synthesized which looks like original content but have some alteration to them. These new ideas lead to an unfortunate phenomenon of fake images and fake videos (DeepFakes) generation. These DeepFake contents are mostly circulated in social media as fake messages or fake news [3]. In this project, we attempt to detect DeepFakes. Our goal is to quantify the accuracy of DeepFakes generation and detection for a variety of newscast settings.

Fake news is one of the biggest cyber threats of our current times. Fake videos of many political leaders, cinema actors, and other celebrities are generated to spread ill propaganda or to diminish their social reputation. Imagine a fake video of a political leader announcing something horrific. Such things can lead to catastrophic outcomes in society. With more advanced techniques of fake video generations, more efficient models are needed to detect them.

Fake images and videos generally alter the face of one of more persons. Mainly four types of facial manipulations are popular to generate DeepFakes. The techniques used to generate DeepFakes are: entire face synthesis, face identity swap, facial attributes manipulation, and facial expression manipulation.

Figure 1 shows an example of all four methods [3].

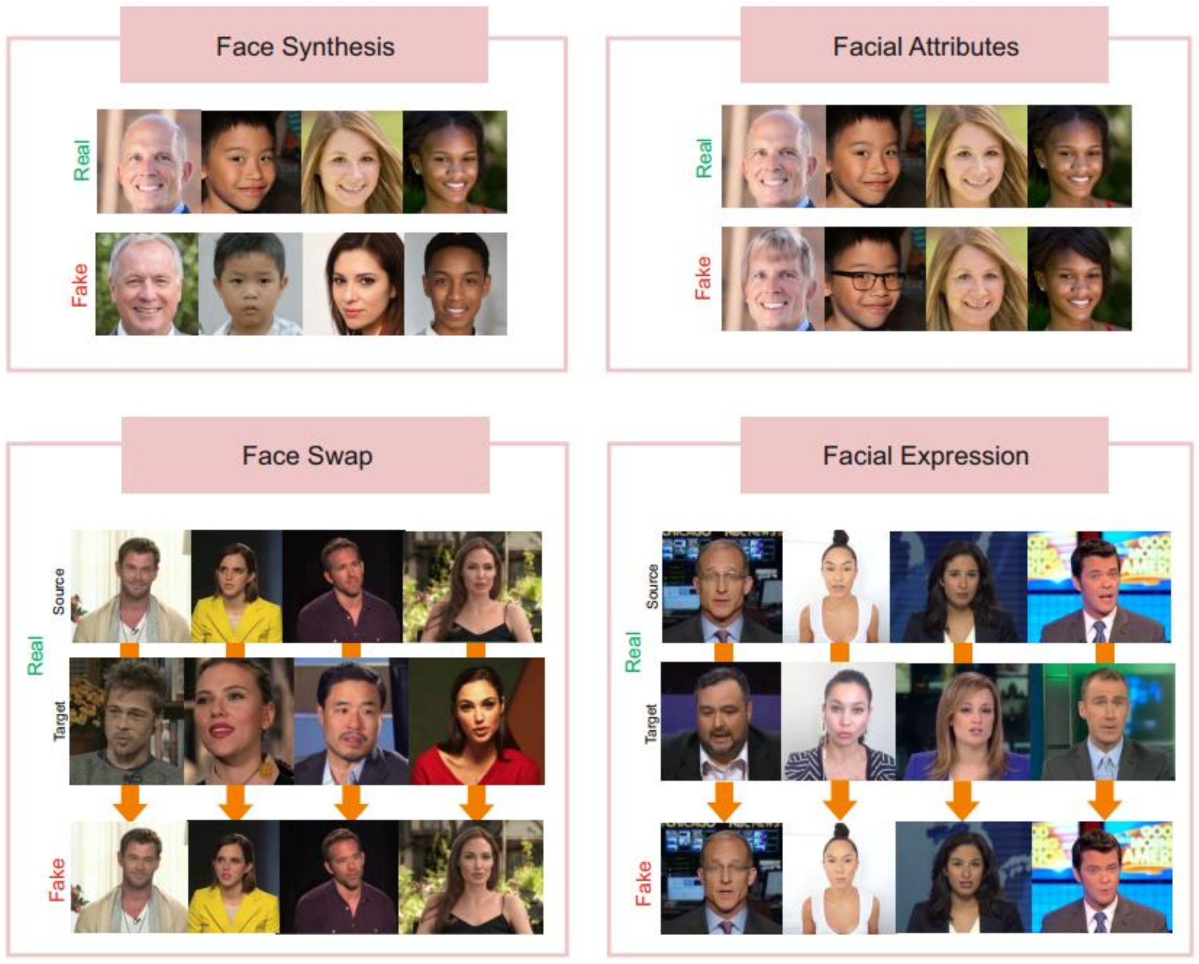

Figure 1. An example of four major fake images and video generation techniques.

With the entire face synthesis method, a new face is generated which is not of any real human but may look like a real human. Using the face swap or face identity swap method, a person from source video is transposed to a person of the target video and a new video is generated. Facial attributes manipulation technique alters the video to change the texture of the skin, hair, etc., add or remove glasses, hats, etc. The facial expression manipulation technique can generate new fake videos by manipulating the facial expression of a target person by taking reference from the source video [3].

In this project, we study various tools, programming frameworks, images, and video synthesis techniques to generate and detect them. We begin with an understanding of the PyTorch framework and write a classifier to classify Devanagari characters as Deliverable 1. We explore AEs to generate Devanagari characters in Deliverable 2. Further, we study GAN techniques and develop a GAN-based model to generate Devanagari characters as part of Deliverable 3. Then, we study a state-of-the-art GAN model to swap faces in images. We make changes to its core architecture and train the model to generate new fake images in Deliverable 4. This core groundwork and research should help develop the background needed to eventually quantify DeepFakes generation and detection processes in a variety of settings including newscasts.

II. BASICS OF DEVANAGARI

Devanagari is the script used by over 120 languages including the Sanskrit language. Sanskrit is a language of ancient India with a 3,500-year history. Its primary sacred language of Hinduism. Sanskrit is written in Bramhi, Gupta, and Devanagari scripts. Most of the literature of Hinduism is written in the Devanagari script. Devanagari is part of the Brahmic family of scripts of India, Nepal, Tibet, and Southeast Asia. The Devanagari script is composed of 47 primary characters which include 33 consonants, 14 vowels, and 2 extra vowels (अं , अः) which are only used only in the Sanskrit. The Devanagari script also has 10 digits, like other scripts. Readers are encouraged to review [4] to learn more and visualize the contents of the script.

A dataset of Devanagari characters can be found at [5]. It comprises 92,000, 32x32 pixels images. These images include handwritten characters of the script. The collection has examples of 33 original consonants, 3 extra Nepali alphabets, and 10 digits of the script. Vowels are missing from this dataset.

III. DELIVERABLE 1: CLASSIFY DEVANAGARI CHARACTERS

We have developed a Convolution Neural Network (CNN)-based model to learn about features of the Devanagari characters and classify them. We have designed and train five of such models and tested their accuracy. Table 1 shows the details of the architecture and accuracy achieved by each of the models. The highest accuracy achieved was 96.21% with Model 1. The confusion matric of Model 1 is shown in Figure 2.

Sr.#	**Model Architecture**	**Test Accuracy achieved**

1	CNNDevnagari(
	  (conv1): Conv2d(1, 12, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
	  (conv2): Conv2d(12, 16, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
	  (fc1): Linear(in_features=1296, out_features=120, bias=True)
	  (fc2): Linear(in_features=120, out_features=90, bias=True)
	  (fc3): Linear(in_features=90, out_features=46, bias=True)
	)
		96.2101%

2	CNNDevnagari(
	  (features): Sequential(
	    (0): Conv2d(1, 15, kernel_size=(15, 15), stride=(1, 1), padding=(2, 2))
	    (1): ReLU(inplace=True)
	    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
	    (3): Conv2d(15, 16, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
	    (4): ReLU(inplace=True)
	    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
	  )
	  (classifier): Sequential(
	    (0): Linear(in_features=576, out_features=460, bias=True)
	    (1): ReLU(inplace=True)
	    (2): Linear(in_features=460, out_features=276, bias=True)
	    (3): ReLU(inplace=True)
	    (4): Linear(in_features=276, out_features=46, bias=True)
	  )
	)
		95.3261%

3	CNNDevnagari(
	  (features): Sequential(
	    (0): Conv2d(1, 15, kernel_size=(15, 15), stride=(1, 1), padding=(2, 2))
	    (1): ReLU(inplace=True)
	    (2): Conv2d(15, 16, kernel_size=(20, 20), stride=(1, 1), padding=(2, 2))
	    (3): ReLU(inplace=True)
		94.7029%

	) (classifier): Sequential((0): Linear(in_features=784, out_features=627, bias=True) (1): ReLU(inplace=True) (2): Linear(in_features=627, out_features=376, bias=True) (3): ReLU(inplace=True) (4): Linear(in_features=376, out_features=46, bias=True)))	
4	CNNDevnagari((features): Sequential((0): Conv2d(1, 15, kernel_size=(15, 15), stride=(1, 1), padding=(2, 2)) (1): ReLU(inplace=True) (2): Conv2d(15, 16, kernel_size=(20, 20), stride=(1, 1), padding=(2, 2)) (3): ReLU(inplace=True)) (classifier): Sequential((0): Linear(in_features=784, out_features=627, bias=True) (1): ReLU(inplace=True) (2): Dropout(p=0.2, inplace=False) (3): Linear(in_features=627, out_features=376, bias=True) (4): ReLU(inplace=True) (5): Dropout(p=0.2, inplace=False) (6): Linear(in_features=376, out_features=46, bias=True)))	94.4783%
5	CNNDevnagari((features): Sequential((0): Conv2d(1, 12, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2)) (1): ReLU(inplace=True) (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) (3): Conv2d(12, 16, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2)) (4): ReLU(inplace=True) (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)) (classifier): Sequential((0): Linear(in_features=1296, out_features=120, bias=True) (1): ReLU(inplace=True) (2): Linear(in_features=120, out_features=90, bias=True) (3): ReLU(inplace=True) (4): Linear(in_features=90, out_features=46, bias=True)))	95.5797%

Table 1. Details of CNN-based Devanagari character classifiers and their results.

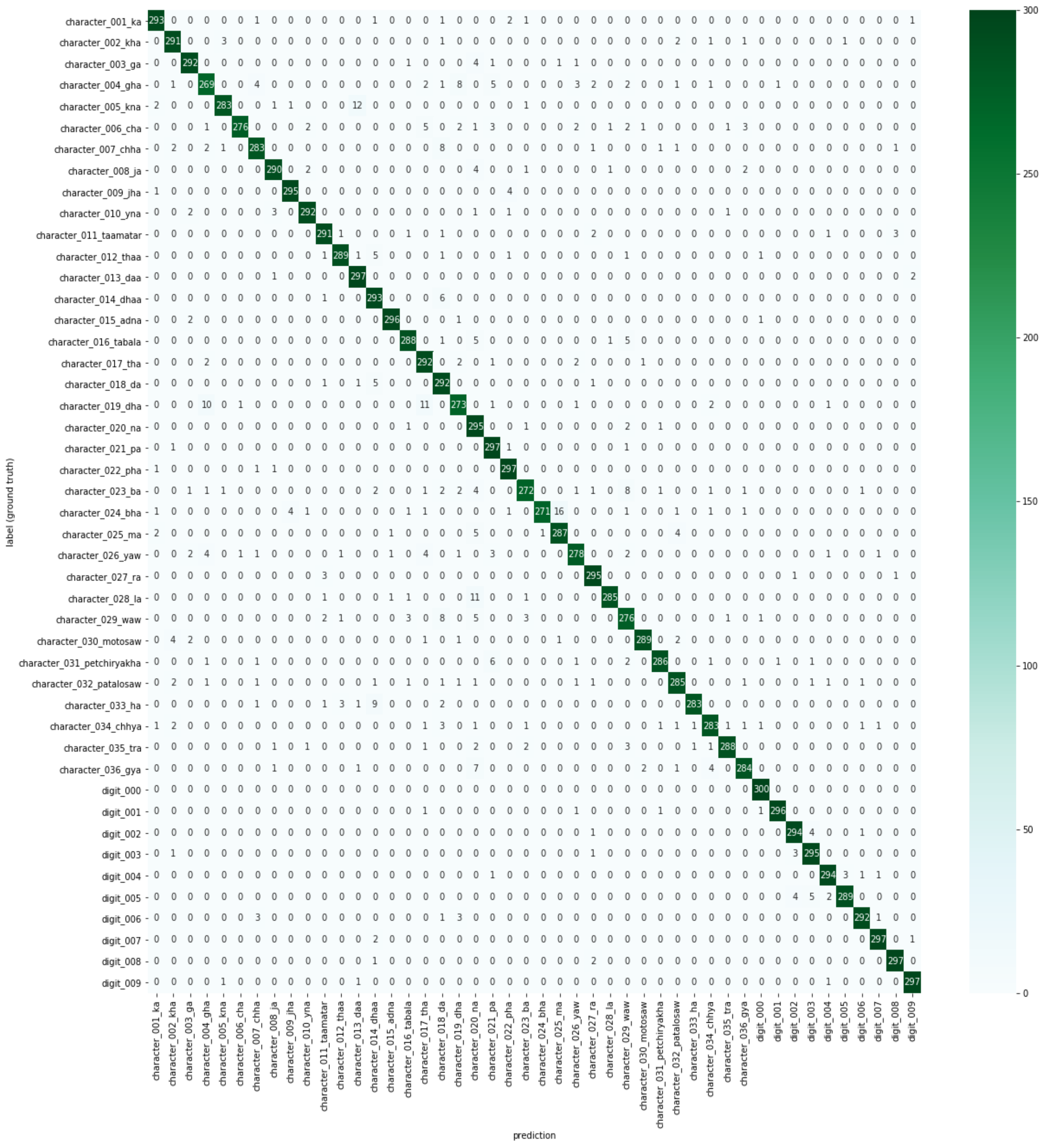

Figure 2. Confusion matrix of Devanagari classification of Model 1.

IV. DELIVERABLE 2: AUTOENCODER TO GENERATE DEVANAGARI CHARACTERS

Autoencoders (AEs) are one of the advanced deep-learning techniques to generate new content. The architecture of AE includes an encoder and a decoder part. Encoder part encodes input image to a latent space which can be a lower-dimensional representation of the inputs. The decoder part can sample a random point from the latent space and it can attempt to generate the original input image [1]. The output of the decoder can be tweaked to generate new images that look like input but has some variations to it. This output of the decoder could be a fake image of a human [3].

Basic AEs and Variation AEs (VAEs) [1] are the most common architectures of AEs. We attempt to generate Devanagari characters using both AE-based models. We implemented basic AEs with simple Artificial Neural Network (ANN) and CNN. We implemented VAE using simple ANN. Table 2 shows the details of the model architectures. Figure 3 shows the sample output of the fake images generated by the AE models.

Sr.#	Model Architecture

1	AutoEncoder(
	  (encoder): Encoder(
	    (encoder): Sequential(
	      (0): Linear(in_features=1024, out_features=921, bias=True)
	      (1): ReLU(inplace=True)
	      (2): Linear(in_features=921, out_features=819, bias=True)
	      (3): ReLU(inplace=True)
	      (4): Linear(in_features=819, out_features=614, bias=True)
	      (5): ReLU(inplace=True)
	      (6): Linear(in_features=614, out_features=409, bias=True)
	      (7): ReLU(inplace=True)
	      (8): Linear(in_features=409, out_features=307, bias=True)
	      (9): ReLU(inplace=True)
	    )
	  )

	(decoder): Decoder(
	    (decoder): Sequential(
	      (0): Linear(in_features=307, out_features=409, bias=True)
	      (1): ReLU(inplace=True)
	      (2): Linear(in_features=409, out_features=614, bias=True)
	      (3): ReLU(inplace=True)
	      (4): Linear(in_features=614, out_features=819, bias=True)
	      (5): ReLU(inplace=True)
	      (6): Linear(in_features=819, out_features=921, bias=True)
	      (7): ReLU(inplace=True)
	      (8): Linear(in_features=921, out_features=1024, bias=True)
	      (9): Tanh()
	    )
	  )
	)

2	autoencoder(
	  (encoder): Sequential(
	    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(2, 2), padding=(2, 2))
	    (1): ReLU(inplace=True)
	    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
	    (3): Conv2d(16, 8, kernel_size=(3, 3), stride=(2, 2))
	    (4): ReLU(inplace=True)
	    (5): MaxPool2d(kernel_size=2, stride=1, padding=0, dilation=1, ceil_mode=False)
	  )
	  (decoder): Sequential(
	    (0): ConvTranspose2d(8, 18, kernel_size=(3, 3), stride=(2, 2))
	    (1): ReLU(inplace=True)
	    (2): ConvTranspose2d(18, 8, kernel_size=(3, 3), stride=(2, 2))
	    (3): ReLU(inplace=True)
	    (4): ConvTranspose2d(8, 1, kernel_size=(2, 2), stride=(3, 3))
	    (5): Tanh()
	  )
	)

3	VAE(
	  (encoder): VAEEncoder(
	    (fc1): Linear(in_features=1024, out_features=819, bias=True)
	    (fc2): Linear(in_features=819, out_features=512, bias=True)
	    (mu): Linear(in_features=512, out_features=307, bias=True)
	    (var): Linear(in_features=512, out_features=307, bias=True)
	  )
	  (decoder): VAEDecoder(
	    (fc1): Linear(in_features=307, out_features=512, bias=True)

	`(fc2): Linear(in_features=512, out_features=819, bias=True)`
	`(out): Linear(in_features=819, out_features=1024, bias=True)`
	`)`
	`)`

Table 2. Details of AE model architectures.

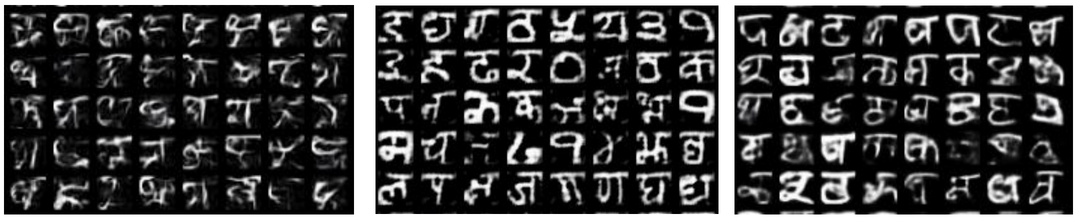

Figure 3. Devanagari characters generated by AE models. The left figure is the output of Model 1, the middle figure is the output of Model 2, and the right figure is the output of Model 3.

We have executed Fréchet Inception Distance (FID score) [8] for the fake samples generated using these models. We sampled 10,000 random images from the latent space and compared them to the same number of real input images to calculate the FID score. Visually it can be seen that the fake images generated by Model 1 do not look like real Devanagari characters. So, we did not calculate the FID score for Model 1. FID score of Model 2 and Model 3 are 677.86 and 657.62, respectively. In our experiments, CNN-based basic AE and VAE produce similar FID scores, and the fake output images of these models also look similar visually. But in general, VAEs produce more realistic images for the more complex image type. VAEs encodes the input to a probability distribution in the latent space, unlike a point or a vector in the case of Basic AEs. So, with VAEs when we sample the data from latent space, we get similar-looking but different images of the input. This property of VAE makes them one of the strong model architectures to generate fake content.

V. DELIVERABLE 3: GAN TO GENERATE DEVANAGARI CHARACTERS

GAN is another advanced deep-learning technique to generate new content. Generator and discriminator are two core parts of the GAN model. The generator network samples random data from latent space and attempts to create new data that may look like original training data. The discriminator network classifies if the output of the generated data from the generator model is like the training set or not. The generator attempts to fool the discriminator and the discriminator attempts to be more robust with each iteration of the training phase. In a successful training, the generator learns to fool the discriminator so that the discriminator cannot classify the contents generated by the generator i.e., the output probability of the discriminator is 0.5 [1].

We have implemented two GAN models to generate fake Devanagari characters. Model 1 is using simple ANN and Model 2 is using CNN. Visually it can be seen that the fake images generated by Model 1 of GAN do not look like real Devanagari characters. So, we did not calculate the FID score for it. The FID score of Model 2 is 625. Model 2 of the GAN generated realistic-looking characters compared to Model 1, as CNN based approach learns the features of the input image. Also compared to VAE Model 3, GAN Model 2 gets a lower FID score and has generated better-looking images of the fake Devanagari characters.

Sr.#	**Model Architecture**
1	Generator((fc1): Linear(in_features=100 out_features=256 bias=True) (fc2): Linear(in_features=256 out_features=512 bias=True) (fc3): Linear(in_features=512 out_features=1024 bias=True) (fc4): Linear(in_features=1024 out_features=1024 bias=True)) Discriminator((fc1): Linear(in_features=1024 out_features=1024 bias=True) (fc2): Linear(in_features=1024 out_features=512 bias=True)

	(fc3): Linear(in_features=512 out_features=256 bias=True) (fc4): Linear(in_features=256 out_features=1 bias=True))
2	Generator((main): Sequential((0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False) (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): ReLU(inplace=True) (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (5): ReLU(inplace=True) (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (8): ReLU(inplace=True) (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (11): ReLU(inplace=True) (12): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (13): Tanh())) Discriminator((main): Sequential((0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (1): LeakyReLU(negative_slope=0.2, inplace=True) (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (4): LeakyReLU(negative_slope=0.2, inplace=True) (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (7): LeakyReLU(negative_slope=0.2, inplace=True) (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (10): LeakyReLU(negative_slope=0.2, inplace=True) (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False) (12): Sigmoid()))

Table 3. GAN model architectures.

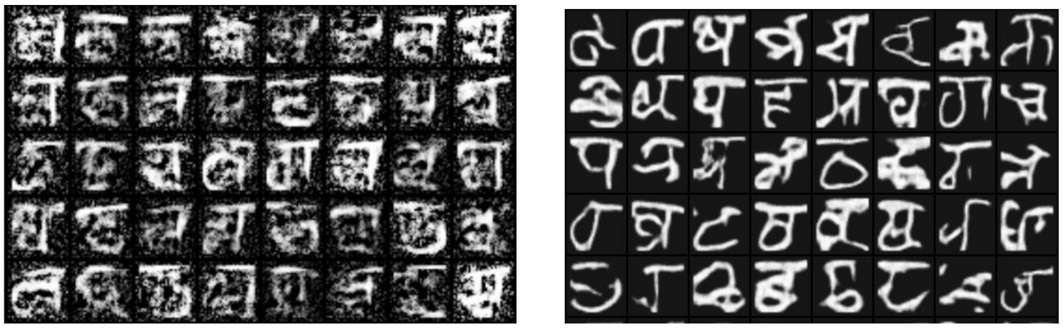

Figure 4. Devanagari characters generated by GAN models. The left Figure is the output of Model 1 and the right Figure is the output of Model 2.

VI. DELIVERABLE 4: FACE-SWAP GAN TO GENERATE FAKE IMAGES

In this sub-task, we redesigned one of the state-of-the-art DeepPrivacy model [6]. The original DeepPrivacy model is a complex GAN architecture. It uses progressive training of GAN [7] to stabilize and improve the training efficiency of the model. Figure 5 shows the changes made architectural for our experiments. We have simplified the generator part of the model by removing alternate layers marked with orange cross signs shown in Figure 5. The overall architecture is like an encoder-decoder pair. The inputs images are encoded into latent space and new images are generated accordingly. The progressive training starts with images of size 8x8 pixels. With each next stage images sizes are doubled until maximum image resolution of 128x128 is reached. During each stage, two sets of layers are added by the DeepPrivacy model. In our modification, we add only one layer. The discriminator uses ResNet50 as the backbone and adds a new layer of ResNet50 each time new layers are added in the generator. We used the same generator for our experiments.

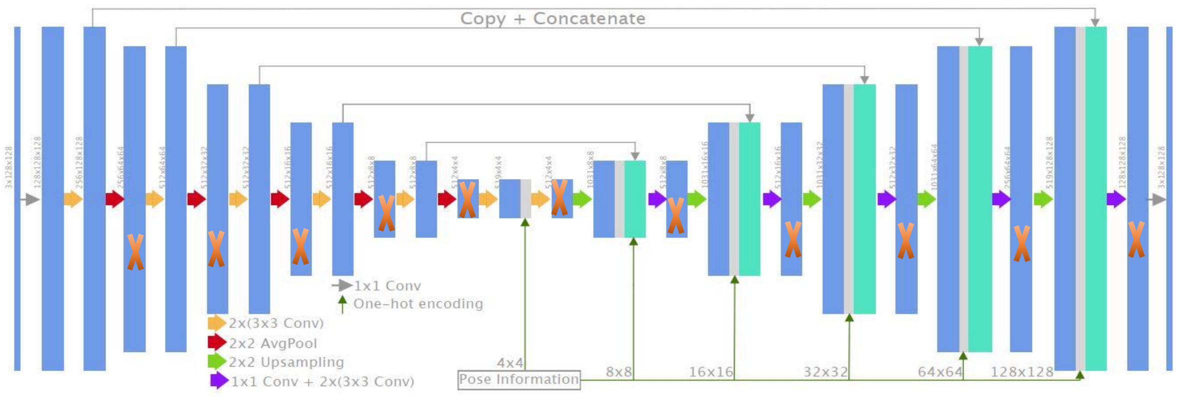

Figure 5. The architecture of the Generator of the modified DeepPrivacy GAN.

We have trained the model for approximately 120 hours in total on Nvidia Titan RTX GPU based computer. Figure 6. shows the images of the validation phase. The modified DeepPrivacy model got the FID score of 51.50. The unmodified model has achieved FID score of 9.327. Even though the FID score of the modified model is quite high, it seems to have generated human-looking new images in many cases. The original model was trained for around 17 days by the authors of [6]. Comparing that if we train the modified model for longer than there is a high chance that our modified model would also get even better output.

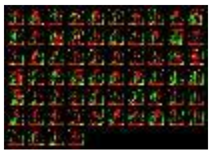
Figure 6A.

Figure 6B.

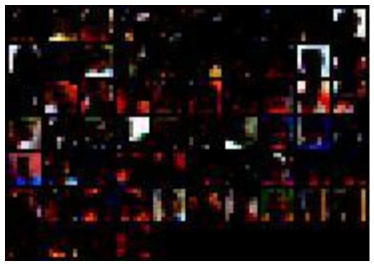
Figure 6C.

Figure 6D.

Figure 6E.

Figure 6F.

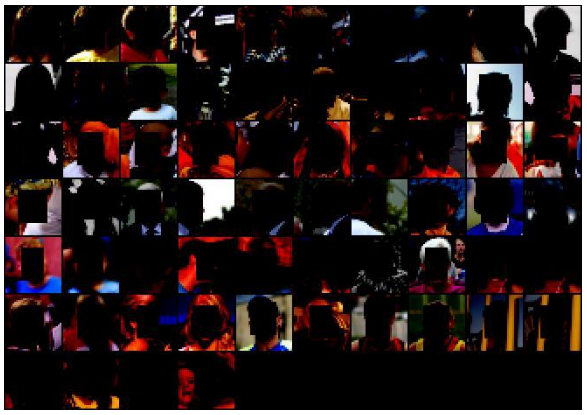
Figure 6G.

Figure 6H.

Figure 6I.

Figure 6J.

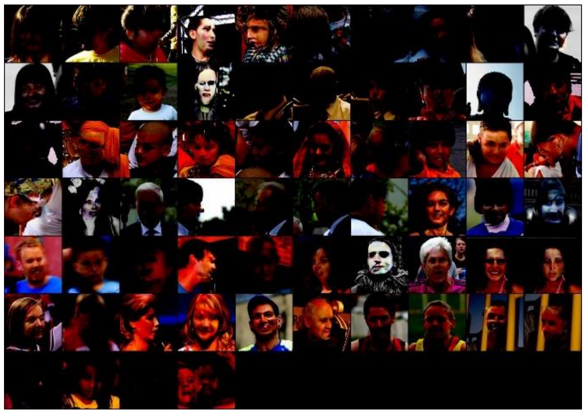

Figure 6K.

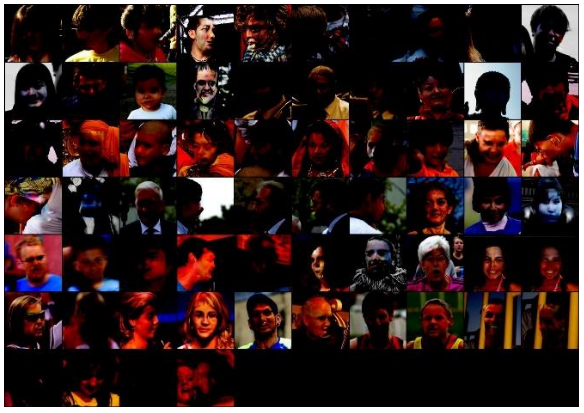

Figure 6L.

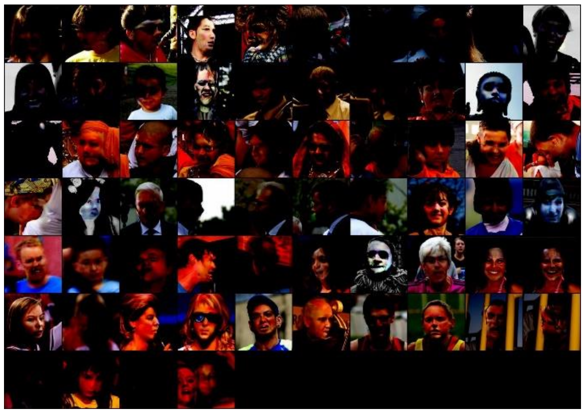

Figure 6M.

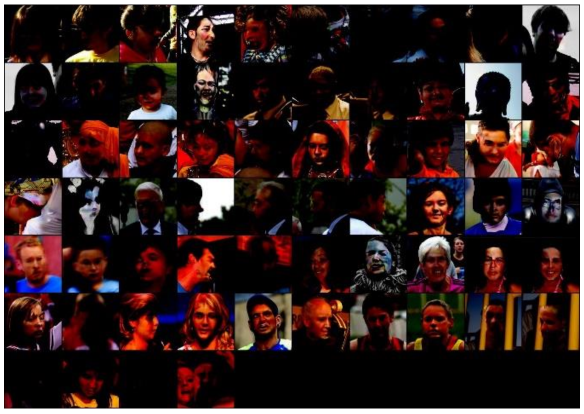

Figure 6N.

Figure 6. Images generated by the modified DeepPrivacy model in the validation phase. See Table 4 for the mapping of the images.

Figure index	Epoch	Image size
6A	0	8x8
6B	256	8x8
6C	1200128	16x16
6D	3400192	16x16
6E	3600128	32x32
6F	5800320	32x32
6G	6000288	64x64

Figure index	Epoch	Image size
6H	8200864	64x64
6I	8400800	128x128
6J	11400832	128x128
6K	11600832	128x128
6L	11800832	128x128
6M	12000832	128x128
6N	12200832	128x128

Table 4. Mappings of sub-images of Figure 6 to Image size and epoch at which the image was generated. Each square of the sub-image is of the resolution mentioned in the corresponding Image size.

VII. CONCLUSION

DeepFakes are fake videos and can create catastrophic issues in our society. In this project, we attempt to detect the DeepFakes. We begin with developing a CNN-based model using PyTorch to classify Devanagari characters. Then we implemented AE and GAN-based complex models to generate fake but realistic-looking Devanagari characters. The GAN-based model was the best performing model and it achieved an FID score of 625. We extended the idea of fake contents generation to the face-swap model by altering the DeepPrivacy model. DeepPrivacy is highly complex and one of the state-of-the-art models to generate fake images by swapping the faces in the given images. We explored and modified the DeepPrivacy model to develop an understanding of the architecture, common design patterns, and techniques to train complex models for very long durations. The model was trained on a high-end GPU-based system for around 120 hours. We achieved the FID score of 51.50 with our model and also the images generated by the model are visually convincing for the face-swapping goal.

In the second semester of the project, we will develop advanced models based on VAE and GAN, to generate and detect DeepFake videos. We would also experiment with extracting various microscopic and mesoscopic features of the videos by using Recurrent Neural Networks (RNNs) and CNNs in detecting DeepFakes. We also plan to investigate the effectiveness of using an ensemble of models and incorporating pre-trained image classification models to increase the efficiency in detecting DeepFakes. The model architectures and PyTorch techniques learned this semester would serve as the basis for achieving the goal of the second semester.

REFERENCES

[1] Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[2] Goodfellow, Ian J. et al. “Generative Adversarial Networks.” *ArXiv* abs/1406.2661 (2014)

[3] Tolosana, Ruben, et al. "DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection." arXiv preprint arXiv:2001.00179 (2020).

[4] https://en.wikipedia.org/wiki/Sanskrit. Accessed: 2020-01-30

[5] https://www.kaggle.com/rishianand/devanagari-character-set. Accessed: 2020-02-01

[6] Hukkelas, H.; Mester, R.; and Lindseth, F. 2019. Deepprivacy: A generative adversarial network for face anonymization. In Bebis, G.; Boyle, R.; Parvin, B.; Koracin, D.; Ushizima, D.; Chai, S.; Sueda, S.; Lin, X.; Lu, A.; Thalmann, D.; Wang, C.; and Xu, P., eds., Advances in Visual Computing, 565-578. Cham: Springer International Publishing.

[7] Karras, Tero et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation.” *ArXiv* abs/1710.10196 (2018)

[8] Borji, Ali. “Pros and Cons of GAN Evaluation Measures.” Computer Vision and Image Understanding 179 (2019): 41–65. Crossref. Web.