Deepfake Image Generation & Detection using the ForgeryNet dataset

***Abstract*—Deepfake is a type of synthetic media where an individual's face, voice, or both can be swapped or altered; in other words, artificially producing fake data using deep learning. Deepfake can be achieved using GANs (Generative Adversarial Networks) and Variational Auto Encoders (VAE). This paper will use GANs to generate and detect fake images using the ForgeryNet Dataset. The entirety of the dataset is not used due to the limitation of computation power. Furthermore, GANs involve two convolutional neural networks: the generator model and the discriminator model, where the generator generates fake images and tries to fool the discriminator. On the other hand, the discriminator tries to improve its detection of fake images. In this paper, a generator model and discriminator model will be implemented using the PyTorch library, and those models will be trained using a portion of the ForgeryNet dataset. Later sections will have the experiments and results produced from the experiments. The objective is to show successful deepfake image generation and detection utilizing deep convolution generative adversarial networks using the ForgeryNet dataset.**

***Index Terms*—Deepfake, GANs, Generator, Discriminator, Neural Network, Training, ForgeryNet Dataset**

I. Introduction

DEEPFAKE is related to generating fake images or videos consisting of the face of a specific person, which is altered with another person's face data utilizing the deep convolutional network. Deepfake is a vast field with various implementation opportunities, such as face swapping, face reenactment, talking face generation, and many more. This project paper specifically utilizes Deep Convolutional Generative Adversarial Networks to generate and detect fake images utilizing the ForgeryNet dataset. GANs are a framework for teaching deep learning models to capture the training data distribution and create new data from that similar distribution. GANs utilize generative and discriminative models, and training these two models is usually done by pitting them against each other to generate and detect fake images. The discriminative model determines if an image generated by the generative model is fake or real. The generative model's job is to create fake images to fool the discriminative model. A balanced GAN would be the generator's ability to create perfect fake images that look like training images, and the discriminator will classify if the generated images are fake or real. A DCGAN (Deep Convolutional General Adversarial Networks) is a version of the GAN which utilizes the convolutional layers and convolutional transpose layers within the discriminator and generator. In the upcoming sections, DCGAN methodology will be used to generate fake images and discriminate fake images. Generating high-quality images with DCGAN is computationally heavy. Therefore, this project only focused on generating low-quality fake images utilizing GANs (Generative Adversarial Networks).

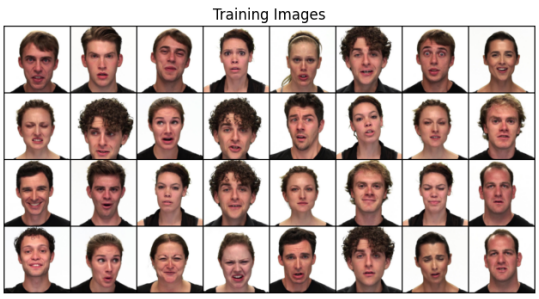

Fig. 1. Sample images of 128x128 size from ForgeryNet Dataset

II. Datasets

The project utilizes the ForgeryNet dataset **[1]**. The dataset was first downloaded, and due to the heavy volume dataset, the dataset was reduced to 1K images from the ForgeryNet dataset to reduce computational time. The dataset is transformed using the ImageFolder function from the PyTorch library. The images within the dataset are then resized into 128 pixels while maintaining their aspect ratio. The images are then cropped, creating perfect

square-shaped images of 128x128 pixels. Then, it is transformed into tensors with channel, height, and width dimensions. The transformed tensors are then normalized between -1 and 1. The DCGAN, a variation of the GANs, is trained using this dataset. The first 32 images are shown in Figure 1.

III. Implementation of DCGAN

A. Discriminative Model

The objective of this model is to detect the fake images generated by the generator model. The architecture used for this model is defined in Table 2. This neural network comprises convolutional layers, batch normalization, and nonlinear activation functions such as LeakyReLU. The convolutional layer 1 is defined with image channels and output feature maps of 3x32. Layers 2 from 5 take the input as the feature maps of the previous layer, and the output channels progressively increase from $[64, 128, 256, 512]$ as the output feature maps increase with the increased layers up until the output layer. All the layers have a 4x4 kernel size. Layers from 1 to 5 have a stride of 2x2, except the output layer stride is 1x1. The final output layer 512x1 is only a single feature map with stride 1x1 and 0 padding, which ensures the spatial dimensions collapse to 1x1. Batch normalization is applied to only layers 2 to 5 $[64, 128, 256, 512]$. This helps normalize the activations to improve training stability and convergence speed. Batch normalization operates by normalizing the input to each layer, ensuring that the activations have a zero mean and unit variance **[2]**. This normalization is done over each mini-batch during training. Batch normalization is applied after convolving each layer. The LeakyReLU activation function is used for all layers except the output layer **[2]**. **LeakyReLU** is defined as below. The alpha used in this case is 0.2 for the forward propagation function.

$$f(x) = \begin{cases} x, & \text{if } x \geq 0, \\ \alpha x, & \text{if } x < 0, \end{cases}$$

During the forward propagation, the input layer data is applied first through the convolutional layer and then the LeakyReLU function. The second layer input is applied through the convolutional layer, followed by the LeakyReLU. It keeps on going until the final layer. Then, the final layer is passed through the sigmoid function, which outputs a probability distribution from 0 to 1 for 128 batches of data.

Attribute	Value
Number of Channels	3
Convolutional Layers	6
Input Layers	3x32
C2	32x64
C3	64x128
C4	128x256
C5	256x512
Ouput Layer	512x1
Batch Normalization	[64, ‘128, 256, 512]
Activation Function	Leaky ReLu
Slope for Negative Values	0.2
Output Layer Activation	Sigmoid
Kernel Size	4x4
stride	2x2
padding	1 except for the final layer

TABLE I
Discriminator Model Architecture

Overall, the discriminator model is a binary classification network which takes an image as its input and outputs a probability distribution of fake or real images. The model takes an image of size 3x128x128, where 3 is the number of channels (RGB channels) and 128x128 is the width and height of the image. Then, the following data is passed through a series of convolutional layers, normalization layer, and LeakyRelu activation layers, finally outputting a probability distribution using a sigmoid activation function.

B. Generative Model

Attribute	Value
Latent Vector	100
Convolutional Layers	6
Input Layers(C0)	100x512
C1	512x256
C2	256x128
C3	128x64
C4	64x32
Ouput Layer(C5)	32x3
Batch Normalization	[512, 256, 128, 64, 32]
Activation Function	Leaky ReLu
Slope for Negative Values	0.2
Output Layer Activation	TanH (Hyperbolic Tangent)
Kernel Size	4x4
stride	2x2 except for the input layer (1x1)
padding	1x1 except for the input layer

TABLE II
Generative Model Architecture

Unlike the discriminative model, the generative model takes a latent vector, a 1D fixed-length vector. The generator network is designed to create synthetic images using random noise, in other words, generating realistic data. GANs utilize this latent vector and map it with output feature maps. The generator takes this latent vector as input and processes it through convolutional layers to produce a realistic output of image size 3x128x128.

The generator model is composed of convolutional transposed layers which effectively invert the operation of regular convolutional layers. The generator model upsamples the input by learning to expand the small feature maps into larger ones. For the first transposed convolutional layer, a latent vector with size 100 is the input channel, which is the depth of the input feature maps, and 512 is the output feature maps. The kernel size is 4x4, similar to the discriminator model. The first layer stride is 1x1, and the rest of the layer has 2x2 strides. Similarly, the first layer has no padding, and all the other layers have 1x1 padding. The batch normalization is performed on the $[512, 256, 128, 64, 32]$ on these layers of inputs. The forward propagation function utilizes the ReLU activation function, and each layer's input is first passed through the normalization and then normalized using the ReLU activation. The output layer is sent through the TanH activation function, normalizing the values from -1 to 1.

The generator model maps the latent vector to the data space, representing an RGB image of size 3x128x128, similar to the training images. The generator model passes through transposed convolutional layers where each is paired with 2-dimensional batch normalization, in this case, 16-size batches, and utilizes ReLU activation layers to normalize the data. The output layers utilize the TanH function. This effectively converts a vector of random noise into a 128x128 pixel image that resembles human faces. Thus the generator is able to generate realistic synthetic data.

C. Training Parameters

Table III shows the values that are used to train the discriminator and generator model. The dataset used is the ForgeryNet dataset for training the models. Dataset training samples are 1K images, with a batch size of 16 images. The models are trained for 100 epochs; however, for each epoch, 1K samples of 16 batch sizes creates a total of approximately 63 batches. With a maximum number of 100 epochs and a total number of 63 batches in the dataloader, a total training run of roughly 6300 runs is done. Both the discriminator and the generator model uses a learning rate of 0.0002.

Attribute	Value
Dataset Name	ForgeryNet
Dataset Training Sample	1000
Batch Size	16
Number of Epochs	100
Discriminator Learning Rate	0.0002
Generator Learning Rate	0.0002
betas	(0.5, 0.999)
Loss Function	Binary Cross Entropy

TABLE III
DCGAN TRAINING INPUTS

D. Binary Cross Entropy Loss

Binary Cross Entropy loss is defined as shown below in the mathematical equation,

$$\text{BCE Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

- N: Number of samples in the batch.
- y_i: Ground truth label for sample i, which is either 0 or 1.
- $\hat{y}_i$: Predicted probability (model's output) for sample i, within the range of $[0, 1]$.
- **First Term:** $y_i \log(\hat{y}_i)$
  - When $y_i = 1$, Penalizes the model for predicting probabilities far from 1.
- **Second Term:** $(1-y_i)\log(1-\hat{y}_i)$
  - When $y_i = 0$, Penalizes the model for predicting probabilities far from 0.

Binary Cross-Entropy loss is widely used for binary classification tasks and to differentiate the probabilities between the actual and predicted labels. It takes N number of batch size, where the ground truth label is denoted as y_i either 0 or 1, $\hat{y}_i$ ranges from 0 to 1. The model penalizes any loss that is far from the original value (ground label) by applying the following equation,

$$y_i \log(\hat{y}_i) \quad \text{if } y_i = 1, \quad \text{and} \quad (1-y_i)\log(1-\hat{y}_i) \quad \text{if } y_i = 0$$

This ensures that the predictions closer to the true labels have lower loss values. The final loss is

then calculated using the BCE Loss formula. Thus averaging over the batch size by providing a single loss value.

The real label is 1, and the fake label is 0. Two Adam optimizers are set for the generator and discriminator model. These models utilize these true and fake labels while model losses are calculated. The learning rate is 0.0002, and beta1 = 0.5 and beta = 0.999 are defined, as discussed in the training parameters sub-section.

E. Training the models

Both models, the generator and discriminator model, are trained alternatingly in an adversarial manner. For each epoch, the discriminator is trained on both real images from the ForgeryNet dataset and fake images from the generator model. The objective of the training is to classify real images with outputs close to 1 and fake images with outputs close to 0. Losses are calculated separately for real and fake data and then combined to compute the total discriminator loss. The discriminator's losses for both real and fake images are computed using the Binary Cross-Entropy Loss (BCE) function. The discriminator tries to minimize the difference between its predictions and the actual labels for real and fake data. The generator is then trained to generate fake images from latent vectors sampled from a normal distribution. The discriminator predicts the output generated from the generator model. The discriminator's prediction is then used to calculate the generator's loss: a wrong classification corresponds to a lower loss for the generator model. Then, the discriminator model's weights are updated. Note that biases are turned off for the models due to the beta term used in the batch normalization, which effectively adds a bias to each channel. Additionally, batch normalization applies two learnable parameters to each channel: beta, a bias term that serves a similar purpose to the bias added in the convolutional layers, and gamma, a scaling term. These beta and gamma parameters effectively replace the need for the bias term in preceding layers as batch normalization re-centers and re-scales the output of the convolutional operation. Furthermore, the generator is trained to generate fake images from the latent vector. The fake images are then again passed through the discriminator for classification. Here, the generator tries to fool the discriminator into classifying the fake images as real. Then, the generator loss is calculated and back propagated to update its weights.

This training process helps the generator create realistic images. Simultaneously, the discriminator model learns to detect real and fake images. With increased training (number of epochs), the generator improves its output quality by reducing the generator's loss, and the discriminator's confidence gets higher in detecting real images.

IV. Results

In the following section, Three different results are shown. One segment discusses models' losses during the training session, the generated fake images, and the prediction quality of the discriminator model on real and fake images. Figure 2 depicts the changes in losses of the generator and discriminator models over training epochs. It also shows that with increased training time, the generator model learns to perform better and better. From epochs 0 to 1000, the generator model doesn't perform well with varying losses between 4 and above. Note that the losses are not normalized between 0 and 1. At the same time, the discriminator model initially has a high loss and decreases rapidly, then stabilizes at a lower value over time. From epoch 1700 until 3000 epochs, the generator's loss stabilized and showed that it improved in generating realistic fake images. On the other hand, the discriminator model provides relatively consistent performance in distinguishing between real and fake images. There are some fluctuations between the models, which shows the adversarial training between the two models.

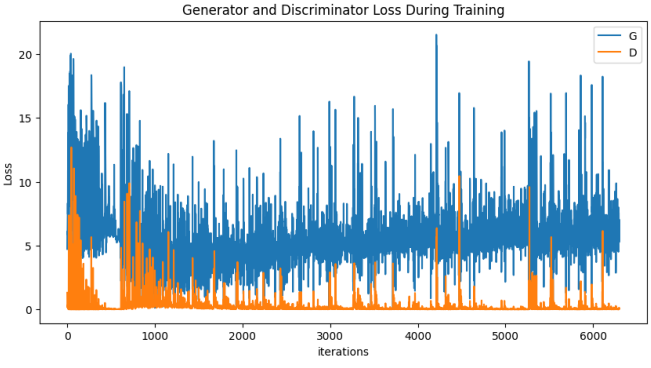

Fig. 2. Generator and Discriminator Loss during training

With increased training time, the generator gets better at generating images, while the discriminator gets better at detecting real or fake images.

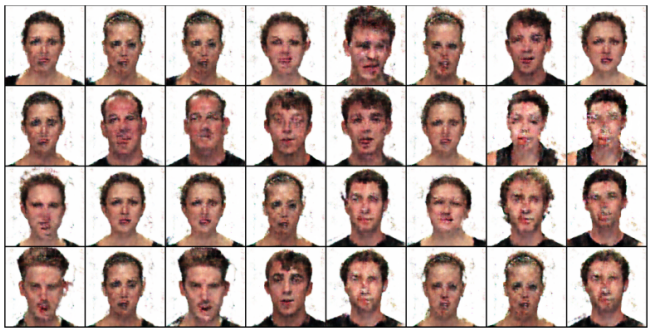

Fig. 3. Fake Images Generated By Generator Model

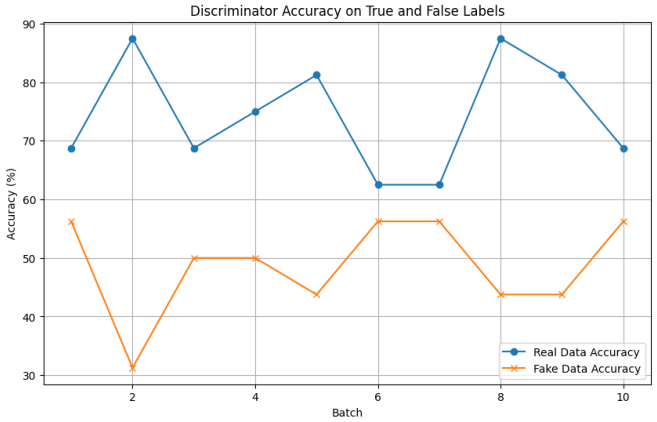

Fig. 4. Fake Images Generated By Generator Model

Figure 3 depicts the 32 fake sample images generated from random noise by the generator model. This suggests that the model can generate synthetic images trained from real images.

In Figure 4, the results show the performance of the discriminator model in detecting real or fake images. From the ForgeryNet dataset, 600 sample images were taken to test the batch-wise accuracy of the discriminator model, trying to distinguish between real and fake images. The blue line determines the real images, and the orange line determines the fake images. Notice one thing from Figure 4: when the discriminator's detecting accuracy is higher for real images, its fake image detection is lower, and vice versa. The peak accuracy of detecting real images is in batch segment 2, and the lowest detection accuracy for fake images is also in batch 2. The fluctuations can be seen as the discriminator model being convinced that fake images are real.

Thus concludes our results section that, this project successfully generates deepfake 128x128 images as well as it is able to detect the fake and real images.

A. *Comparison Between 64 vs 128 Image Sizes*

This section compares the two different Generative Adversarial Networks with the different image sizes as input, comparing the output quality. The DCGAN architecture for the 64 image size differs from the 128 image size DCGAN architecture.

Table IV defines the discriminator model architecture for the 64-image size. On the other hand, Table V defines the generator model. The base difference between 64 vs 128 image-based DCGAN is that their convolutional layers are described in Tables I, II, IV, and V.

Attribute	Value
Number of Channels	3
Convolutional Layers	6
Input Layers	3x64
C2	64x128
C3	128x256
C4	256x512
Output Layer	512x1
Batch Normalization	[128, 256, 512]
Activation Function	Leaky ReLu
Slope for Negative Values	0.2
Output Layer Activation	Sigmoid
Kernel Size	4x4
stride	2x2
padding	1 except for the final layer

TABLE IV
DISCRIMINATOR MODEL ARCHITECTURE: 3X64X64

Attribute	Value
Latent Vector	100
Convolutional Layers	6
Input Layers(C0)	100x512
C1	512x256
C2	256x128
C3	128x64
Output Layer(C5)	64x3
Batch Normalization	[512, 256, 128, 64]
Activation Function	Leaky ReLu
Slope for Negative Values	0.2
Output Layer Activation	TanH (Hyperbolic Tangent)
Kernel Size	4x4
stride	2x2 except for the input layer (1x1)
padding	1x1 except for the input layer

TABLE V
GENERATIVE MODEL ARCHITECTURE: 3X64X64

The following images in Figure 5 are generated by the deep convolutional generative adversarial networks defined in the section on the Implementation of DCGAN (Tables I and II). Figure-6 images are

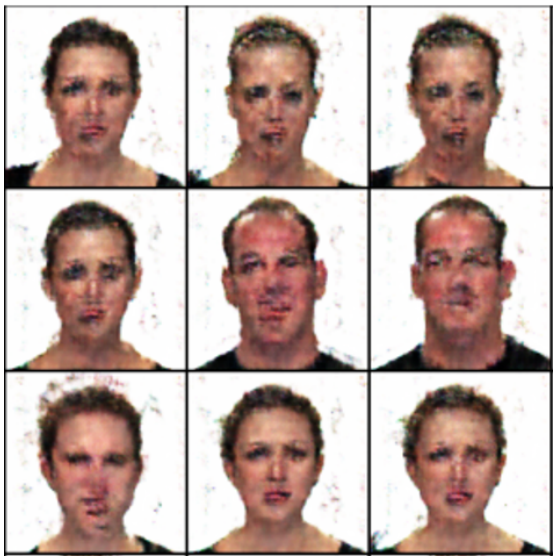

Fig. 5. 128x128 Input Images for Generator Model

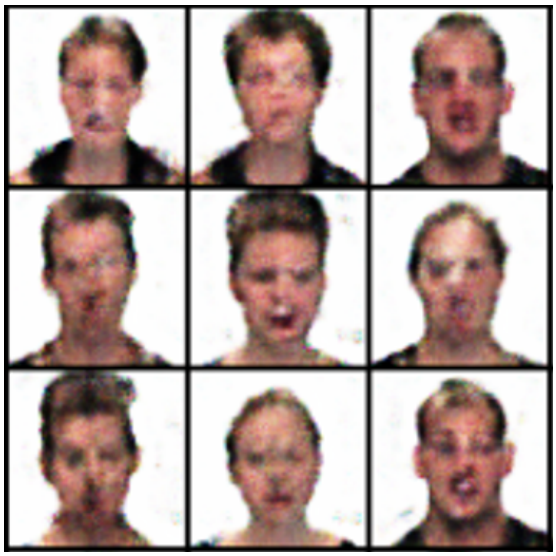

Fig. 6. 64x64 Input Images for Generator Model

generated from the DCGAN architecture described in Table IV and V. The generated images in Figure 6 are better quality than those seen in Figure 6. This is due to training input images. Heavy computational systems are needed for more considerable image training. However, with increased resolution, GANs are able to generate realistic deepfake images, which was the primary goal of this project. Figure 6 is less detailed than the image in Figure 5. The patterns and textures are more coarse or pixelated than Figure 5.

V. Limitations

The following project doesn't generate high-quality fake images. Due to a lack of high computational resources, it is impossible to create high-quality synthetic data. The ForgeryNet dataset consists of 600GB of image data. Only a fractional subset of images was used due to hard disk constraints prohibiting the use of the entire data. Storing and training on the complete dataset requires a tremendous amount of time, computation power and cloud storage.

VI. Conclusion

This report implements a variation of GANs known as DCGAN to generate deepfake images utilizing a portion of the training dataset taken from the ForgeryNet dataset. Furthermore, the implementation of deep convolutional generative adversarial networks is discussed broadly. The DCGAN utilizes two convolutional neural network models pitted against each other, where the generator model creates realistic fake images from random noise, and the discriminator model evaluates whether an image is real or fake. Note that real images are from the training dataset, and the generator model generates fake images. Later the results section shows that the created model successfully generates deepfake images using GANs, and the discriminator classifies or detects real or fake images. Furthermore, another model was implemented to show that the image quality can improve with higher resolution images as training data, though it is computationally intensive. Lastly, deepfake image generation and detection using the ForgeryNet dataset is successfully done utilizing GANs.

References

[1] Y. He, "ForgeryNet: A Comprehensive Benchmark for Deepfake Detection," Yinan He Projects. [Online]. Available: https://yinanhe.github.io/projects/forgerynet.html. [Accessed: Jan. 1, 2025].

[2] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," arXiv preprint arXiv:1511.06434. [Online]. Available: https://arxiv.org/pdf/1511.06434. [Accessed: Jan. 5, 2025].