Deep Fake Image Detection using Neural Networks

Arif Ali (23m0822),Skand (23m1163), Irshad Khan (23m0824), Suchit Meshram (23m0814), Soumik (23m0826), Mohiboddin Shaikh (23m0827)

***Abstract*—Deepfakes, which involve the use of artificial intelligence and machine learning algorithms to generate realistic yet entirely fabricated media, pose a serious threat to the veracity of visual information in various contexts. As these techniques continue to advance in sophistication and accessibility, the imperative for effective deep fake detection systems becomes increasingly vital. Our project, titled "Deep Fake Image Detection Using Neural Networks," seeks to address the emergent challenges posed by the proliferation of deepfake technology. In this pursuit, we constructed and compared three distinct models—Convolutional Neural Network (CNN), Feedforward Neural Network (FNN), and Vision Transformer (ViT)—for detecting deep fake images. Our objective was to discern the most effective model in mitigating the challenges posed by manipulated visual images.**
Keywords: artificial intelligence; deep fakes; deepfake detection; deep learning; CNN; ANN; ViTransformer.

I. Introduction.

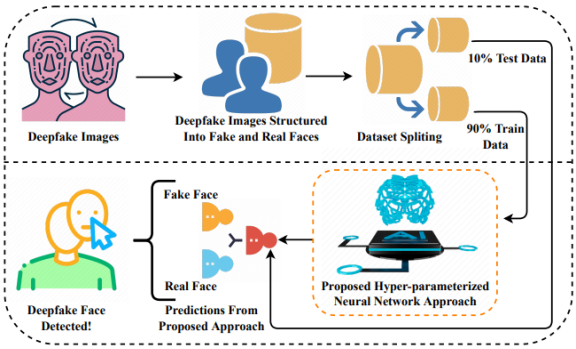
Over the past few years, the proliferation of deepfake technology has raised considerable apprehensions regarding its ability to manipulate visual content, giving rise to numerous ethical, social, and security implications. Deepfakes, employing artificial intelligence and machine learning algorithms to create convincingly fabricated media, present a significant challenge to the credibility of visual information across diverse domains. With the ongoing advancements and growing accessibility of these techniques, the urgent need for implementing robust deep fake detection systems becomes progressively more crucial.

Our project, titled "Deep Fake Image Detection Using Neural Networks," seeks to address the emergent challenges posed by the proliferation of deepfake technology. By leveraging the power of neural networks, our aim is to detect deep fakes images The Convolutional Neural Network, a cornerstone in computer vision, demonstrated prowess in image feature extraction, while the Feedforward Neural Network leveraged its sequential layering for comprehensive pattern recognition. The Vision Transformer, a transformer-based architecture, brought a novel perspective to the task by exploiting self-attention mechanisms.Our comparative analysis involved rigorous evaluations based on key metrics such as accuracy, precision, recall, and F1 score. Each model underwent extensive training and validation on a curated dataset containing both real and deep fake images. The results of this comparative study provided insights into the strengths and weaknesses of each model, facilitating an informed selection based on the specific requirements of deep fake detection.

II. Methodology

Figure 1 illustrates the research methodology for architectural analysis. The neural network techniques are implemented using deepfake

images containing both fake and authentic human faces. A dataset is constructed with fake and real faces, labeled accordingly. This organized deepfake dataset is then divided into training, validation, and test sets. Ninety percent of the dataset is allocated for training the neural network techniques. accuracy in deepfake detection. The performance evaluation of these techniques is conducted on the remaining 10% of the test set.



III. Deepfake Dataset

We will use the "DeepFake and Real Images" dataset from Kaggle, a reputable resource for deepfake detection. This dataset contains a mix of real and deep fake images, providing a diverse set for training and evaluation. For Dataset preprocessing, we have resized the images. This dataset contains manipulated images and real images. The manipulated images are the faces which are created by various means. Each image is a 256 X 256 jpg image of a human face either real or fake.

Fake

Fake

Fake

Real

Real

Real

Real

IV. Feed Forward Neural Network

Firstly, we have made and trained the Feedforward Neural Network. The provided information outlines the architecture and characteristics of a Feedforward Neural Network (FNN). Let's break down the details:

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
         Flatten-1               [-1, 12288]               0
          Linear-2                 [-1, 128]       1,572,992
            ReLU-3                 [-1, 128]               0
         Dropout-4                 [-1, 128]               0
          Linear-5                   [-1, 1]             129
         Sigmoid-6                   [-1, 1]               0
================================================================
Total params: 1,573,121
Trainable params: 1,573,121
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 0.10
Params size (MB): 6.00
Estimated Total Size (MB): 6.14
----------------------------------------------------------------
```

Flatten Layer (1):

This layer flattens the input data, which likely represents a 3D tensor, into a 1D tensor with 12,288 elements.

Linear Layer (2):

This is a fully connected (dense) layer with 128 output units. The layer has 1,572,992 parameters, which are weights and biases.

ReLU Layer (3):
ReLU is an activation function applied element-wise, introducing non-linearity to the model. It has no parameters.

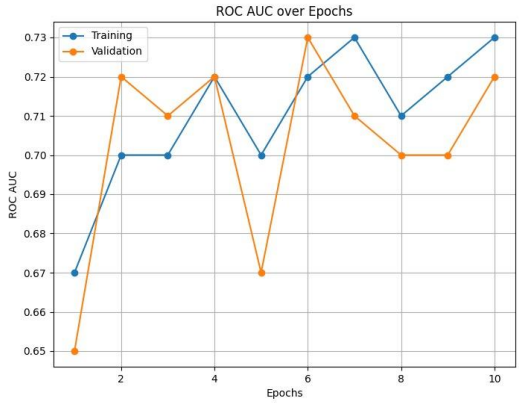
Dropout Layer (4):
Dropout is a regularization technique. During training, it randomly sets a fraction of input units to zero to prevent overfitting.

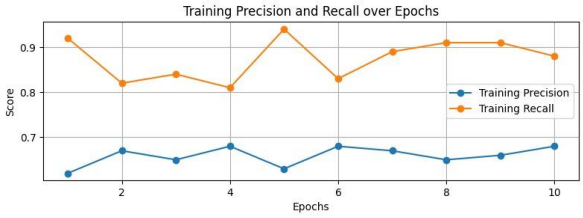
Linear Layer (5):
Another fully connected layer with a single output unit, likely used for binary classification (e.g., real or fake). It has 129 parameters.

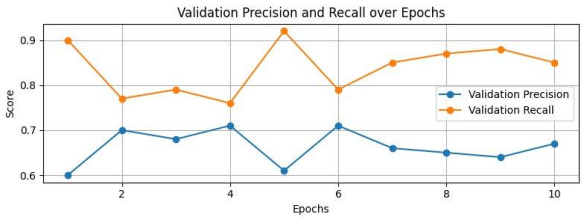
Sigmoid Layer (6):
The Sigmoid activation function squashes the output to a range between 0 and 1, often used for binary classification probability.

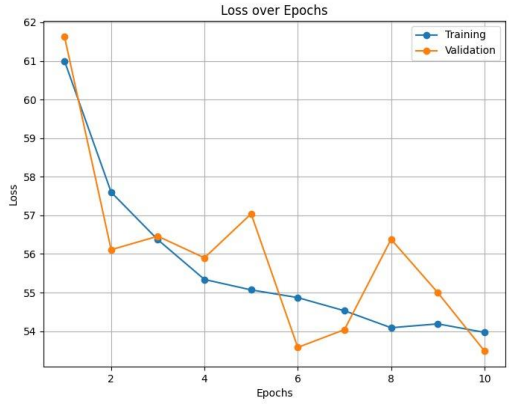
Total Parameters:
The total number of parameters in the network is 1,573,121, all of which are trainable. Non-trainable params usually refer to parameters in layers like Batch Normalization, which are fixed during training.

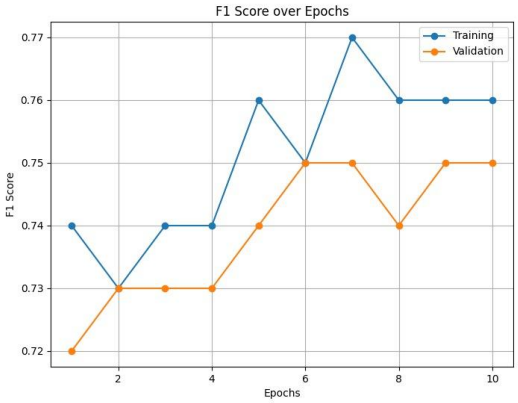
Memory Size:
These values represent the estimated memory requirements during different stages of processing, including input data size, forward/backward pass, and the size of parameters.

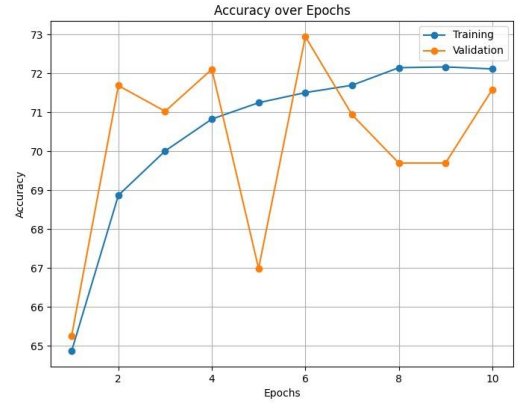












V. Convolution Neural Network

We have made and trained the Convolution Neural Network. The provided information outlines the architecture and characteristics of a Feedforward Neural Network (FNN). Let's break down the details:

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1          [-1, 32, 64, 64]             896
       MaxPool2d-2          [-1, 32, 32, 32]               0
          Conv2d-3          [-1, 32, 32, 32]           9,248
       MaxPool2d-4          [-1, 32, 16, 16]               0
          Conv2d-5          [-1, 32, 16, 16]           9,248
       MaxPool2d-6            [-1, 32, 8, 8]               0
          Linear-7                 [-1, 128]         262,272
          Linear-8                   [-1, 1]             129
================================================================
Total params: 281,793
Trainable params: 281,793
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 1.64
Params size (MB): 1.07
Estimated Total Size (MB): 2.76
----------------------------------------------------------------
```

This neural network architecture consists of various layers, primarily Conv2d (convolutional), MaxPool2d (max pooling), and Linear (fully connected) layers. Below is an explanation of each part:

Conv2d Layer (1):
This is a convolutional layer with 32 filters/kernels, each of size 3x3. The output shape is a 4D tensor with dimensions [-1, 32, 64, 64].

MaxPool2d Layer (2):
Max pooling layer with a pool size of 2x2. It reduces the spatial dimensions of the input by taking the maximum value in each 2x2 region.

Conv2d Layer (3):
Another convolutional layer with 32 filters, producing an output tensor of the same size.

MaxPool2d Layer (4):
Another max pooling layer, reducing the spatial dimensions again.

Conv2d Layer (5):
Another convolutional layer.

MaxPool2d Layer (6):
Another max pooling layer.

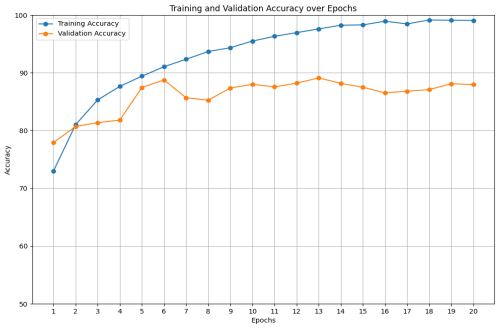
Linear Layer (7):
Fully connected layer with 128 output units, connecting each unit to every unit in the previous layer.

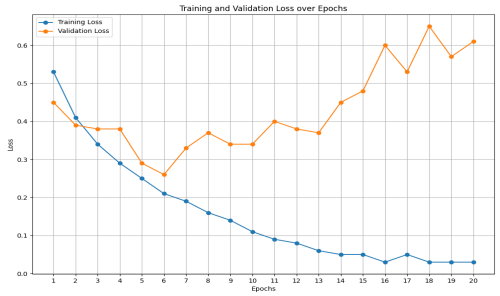
Linear Layer (8):
Final linear layer with a single output unit, likely used for binary classification (e.g., real or fake).

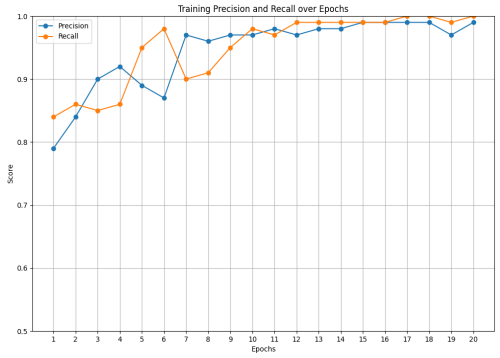
Total Parameters:
The total number of parameters in the network is 281,793, all of which are trainable.

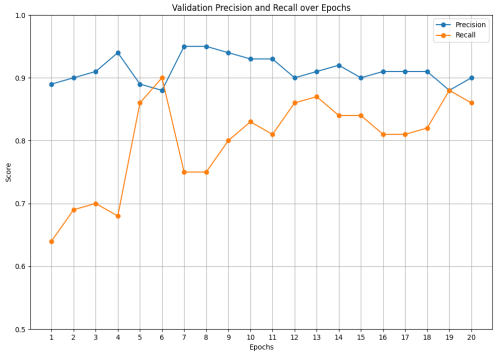
Memory Size:

These values represent the estimated memory requirements during different stages of processing, including input data size, forward/backward pass, and the size of parameters.

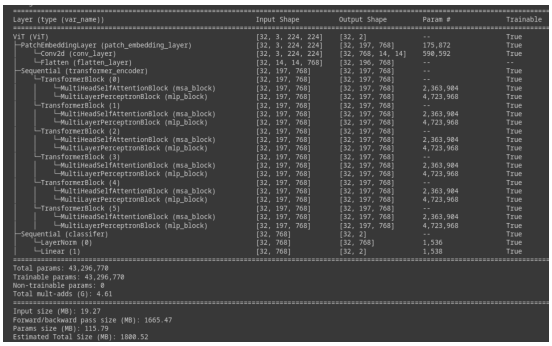








VI. Visual transformer



```
==========================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================
ViT (ViT)                                [32, 3, 224, 224]    [32, 2]              --                   True
├─PatchEmbeddingLayer (patch_embedding_layer) [32, 3, 224, 224]    [32, 197, 768]       175,872              True
│    └─Conv2d (conv_layer)               [32, 3, 224, 224]    [32, 768, 14, 14]    590,592              True
│    └─Flatten (flatten_layer)           [32, 14, 14, 768]    [32, 196, 768]       --                   --
├─Sequential (transformer_encoder)       [32, 197, 768]       [32, 197, 768]       --                   True
│    └─TransformerBlock (0)              [32, 197, 768]       [32, 197, 768]       --                   True
│    │    └─MultiHeadSelfAttentionBlock (msa_block) [32, 197, 768]       [32, 197, 768]       2,363,904            True
│    │    └─MultiLayerPerceptronBlock (mlp_block) [32, 197, 768]       [32, 197, 768]       4,723,968            True
│    └─TransformerBlock (1)              [32, 197, 768]       [32, 197, 768]       --                   True
│    │    └─MultiHeadSelfAttentionBlock (msa_block) [32, 197, 768]       [32, 197, 768]       2,363,904            True
│    │    └─MultiLayerPerceptronBlock (mlp_block) [32, 197, 768]       [32, 197, 768]       4,723,968            True
│    └─TransformerBlock (2)              [32, 197, 768]       [32, 197, 768]       --                   True
│    │    └─MultiHeadSelfAttentionBlock (msa_block) [32, 197, 768]       [32, 197, 768]       2,363,904            True
│    │    └─MultiLayerPerceptronBlock (mlp_block) [32, 197, 768]       [32, 197, 768]       4,723,968            True
│    └─TransformerBlock (3)              [32, 197, 768]       [32, 197, 768]       --                   True
│    │    └─MultiHeadSelfAttentionBlock (msa_block) [32, 197, 768]       [32, 197, 768]       2,363,904            True
│    │    └─MultiLayerPerceptronBlock (mlp_block) [32, 197, 768]       [32, 197, 768]       4,723,968            True
│    └─TransformerBlock (4)              [32, 197, 768]       [32, 197, 768]       --                   True
│    │    └─MultiHeadSelfAttentionBlock (msa_block) [32, 197, 768]       [32, 197, 768]       2,363,904            True
│    │    └─MultiLayerPerceptronBlock (mlp_block) [32, 197, 768]       [32, 197, 768]       4,723,968            True
│    └─TransformerBlock (5)              [32, 197, 768]       [32, 197, 768]       --                   True
│    │    └─MultiHeadSelfAttentionBlock (msa_block) [32, 197, 768]       [32, 197, 768]       2,363,904            True
│    │    └─MultiLayerPerceptronBlock (mlp_block) [32, 197, 768]       [32, 197, 768]       4,723,968            True
├─Sequential (classifer)                 [32, 768]            [32, 2]              --                   True
│    └─LayerNorm (0)                     [32, 768]            [32, 768]            1,536                True
│    └─Linear (1)                        [32, 768]            [32, 2]              1,538                True
==========================================================================================
Total params: 43,296,770
Trainable params: 43,296,770
Non-trainable params: 0
Total mult-adds (G): 4.61
==========================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 1665.47
Params size (MB): 115.79
Estimated Total Size (MB): 1800.52
```

Layers and Components:

The architecture consists of various layers and components such as PatchEmbeddingLayer, Conv2d, Flatten, Sequential, TransformerBlock, MultiHeadSelfAttentionBlock, MultilayerPerceptronBlock, and LayerNorm. The model has a total of 6 TransformerBlocks.
Total Parameters:

Total parameters in the model: 43,296,770
Trainable parameters: 43,296,770
Non-trainable parameters: 0
Computational Characteristics:

Total multiply-adds (G): 4.61 (indicating the computational complexity of the model)
Input size (MB): 19.27

Forward/backward pass size (MB): 1665.47
Params size (MB): 115.79
Estimated Total Size (MB): 1800.52
Input and Output Shapes:

Input shape: [32, 3, 224, 224] (batch size of 32, 3 channels, each with a spatial size of 224x224)
Output shape varies across different layers, e.g., [32, 197, 768]

VII. Results

FNN (Feedforward Neural Network):

Validation Loss: 0.56
Accuracy: 71.12%
Number of Parameters: 1,573,121
Remark: Indicates that the FNN model does not support translation invariance. Translation invariance refers to the model's ability to recognize objects or patterns regardless of their position in the image. The limitation suggests that the FNN model may not perform well when faced with variations in object position.

CNN (Convolutional Neural Network):

Validation Loss: 0.29
Accuracy: 86%
Number of Parameters: 281,793
Remark: Mentions that the CNN model was trained without data augmentation and that the input images were reduced from 256x256 to 64x64 pixels, resulting in feature loss. Despite this, the CNN model achieved higher accuracy compared to the FNN.
Transformer:

Validation Loss: 0.69
Accuracy: 67%
Number of Parameters: 43,296,770
Remark: No specific remark is provided for the Transformer model. However, the lower accuracy and relatively higher validation loss suggest that further analysis or improvements may be needed.

VIII. Experiment with Manually generated Image




Apart from Testing on the test dataset, we have manually created the the fake images as well as tried to gather the image from Midjouney and other face swap tools, and tested whether our model predict it fake or real. The total images created were 17.

CNN Model Accuracy : 13/17 ⇒ 76% (Tested on Total 17 images)
ViT Model Accuracy: 10/17 = 58%

Why accuracy on custom dataset is less than dataset from Kaggle?
=>Deep learning models are sensitive to domain differences. If your custom-generated dataset significantly differs from the Kaggle-trained dataset in terms of content, lighting conditions,

camera angles, or other factors, the model may struggle to generalize well.

Deepfake detection models often focus on identifying inconsistencies in facial features, or artifacts introduced during the deepfake generation process. Custom photoshopped images might not exhibit the same artifacts or inconsistencies, making them harder to detect.

IX. Conclusion

Deepfakes are a serious threat to the integrity of information.There is a need for methods to detect deepfake images.CNNs, FFNNs, and transformers are all promising methods for deepfake detection. Due to resource limitation, we couldn't train the ViT model to converge but it definitely has potential to improve

X. References

https://arxiv.org/abs/2010.11929

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9776410

DATASET and Code references:

https://www.kaggle.com/datasets/manjilkarki/deepfake-and-real-images