A Project Report

FAKE FACE IMAGE DETECTION

Submitted in partial fulfillment of the requirements for the award of degree

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

by

B. Spoorthi (160117733077)
V. Vishal Reddy (160117733121)

Department of Computer Science and Engineering

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)
(Affiliated to Osmania University; Accredited by NBA(AICTE) and NAAC(UGC), ISO Certified 9001:2015)
GANDIPET, HYDERABAD – 500 075
Website: www.cbit.ac.in
[2020-2021]

CERTIFICATE

This is to certify that the project titled "**Fake Face Image Detection**" is the bonafide work carried out by **B Spoorthi (160117733077) and V Vishal Reddy (160117733121)**, students of B.E.(CSE) of Chaitanya Bharathi Institute of Technology(A), Hyderabad, affiliated to Osmania University, Hyderabad, Telangana (India) during the academic year 2020-2021, submitted in partial fulfillment of the requirements for the award of the degree in **Bachelor of Engineering (Computer Science and Engineering)** and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship, or any other similar title.

Supervisor

J. Shiva Sai
Asst. Professor, Dept. of CSE
of CSE

Head, CSE Dept.

Dr. Y. Rama Devi
Professor, Dept.

Place: Hyderabad

Date: 06-06-2021

DECLARATION

We hereby declare that results embodied in this Report of Project on **"FAKE FACE IMAGE DETECTION"** are from work carried out by us in partial fulfillment of the requirements for the award of degree of Bachelor of Engineering. We have not submitted this report to any other university/institute for the award of any other degree.

Spoorthi Badikala (160117733077)

V. Vishal Reddy (160117733121)

Place: Hyderabad

Date: 06-06-2021

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to **J. Shiva Sai**, our project guide, for his invaluable guidance and constant support, along with his capable instruction and persistent encouragement.

We are grateful to our Head of Department, **Dr. Y. Ramadevi**, for her steady support and the provision of every resource required for the completion of this project as the management of the institute, for having designed an excellent learning atmosphere.

We are also extremely thankful to our Project Coordinators **Dr. Y. Ramadevi**, Professor, Dept. of CSE, **Smt. I. Srujana**, Assistant Professor, Dept. of CSE, and **Smt. D. Naga Jyothi**, Assistant Professor, Dept. of CSE for their valuable suggestions and interest throughout the project.

Our thanks are due to all members of the staff and our lab assistants for providing us with the help required to carry out the groundwork of this project.

ABSTRACT

The emergence of social networking services such as Facebook and Instagram, led to a huge increase in the volume of image data generated in the last decade. Use of image (and video) processing software's like Adobe photoshop and DeepFake to create doctored images and videos is a major concern for internet companies like Facebook. These images are prime sources of fake news and are often used in malevolent ways such as for mob incitement.

Currently, a fake image detector is of at most concern to a lot of countries. So, we propose a deep learning framework which differentiates fabricated parts of the image from the real image using supervised learning strategies. We propose a modified neural network structure called the Fake Feature Network which consists of advanced convolution networks. The application also uses a two-step learning method that combines a modified neural structure based on pairwise learning strategy and classifier learning to obtain an effective model for the fake image detection. To enhance the performance of the proposed method, we introduce contrastive loss to learn the common fake features using pairwise learning, which is achieved by incorporating Siamese neural network.

FIGURE INDEX

TABLE INDEX

LIST OF ABBREVATIONS

FFN	Fake Feature Network
CNN	Convolution Neural Network
GAN	Generative Adversarial Network
PGGAN	Progressive Growing Generative Adversarial Network
CFF	Common Fake Feature
JPEG	Joint Photographic Experts Group
ELA	Error Level Analysis
GUI	Graphical User Interface
RAM	Random Access Memory
GPU	Graphics Processing Unit
VRAM	Video Random Access Memory

TABLE OF CONTENTS

1.INTRODUCTION

1.1. Problem Definition

In this technological era, social media has become an integral part of our lives. Over the years we have seen social media take different forms than it was created for. One of the problematic forms of social media is spreading fake information and misinformation. People started using social media as a tool to spread fake information to exploit others or damage their reputation. One of the main ways of doing this was creating fake images or videos. The major contributors of creating fake images are Generative Adversarial Networks aka GANs.

GANs are deep learning based generative models that have been widely used to create partial or whole realistic images or videos. These neural networks have been progressing every year creating more sophisticated networks like BigGAN, CycleGAN and PGGAN, which create highly realistic images. They also have the capability to create synthesized speech videos, which was used in the past to sabotage several politician's credibility.

Current forensic techniques require an expert to analyze the integrity of an image. Since the fake images created are highly sophisticated and almost impossible to detect a system is needed which is on par to the latest GANs models to detect them. A model using customized advanced neural networks to determine fake images has been implemented. A modified neural network structure called Fake Feature Network (FFN) has been proposed, which consists of Convolution Neural Networks, DenseNETs and Siamese Neural Networks. The model created will be trained using pairwise learning strategy to obtain an efficient model.

1.2. Methodologies

The motivation behind creating this network was to implement discriminative feature learning which results in better learning and extraction of fake features in an image. Since the traditional CNNs are not designed to learn discriminative features, integrating the Siamese network with any advanced CNNs has been proposed in developing FFN. Any of the advanced CNNs available today like DenseNet, ResNet, Inception Network,

etc., can be used to learn fake features from the training set. For this model DenseNets have been used as the backbone. The resulting FFN is a two streamed network which is designed to allow a pair of input images to facilitate discriminative feature learning.

The major drawback of traditional supervised learning is that it is hard to identify the subject that is excluded from the training phase. Contrastive loss has been integrated to learn the common fake features by pairwise learning. Contrastive loss is incorporated into the energy function of general loss function used in supervised learning techniques. For classification of the image received from FFN one can use any classification technique here.

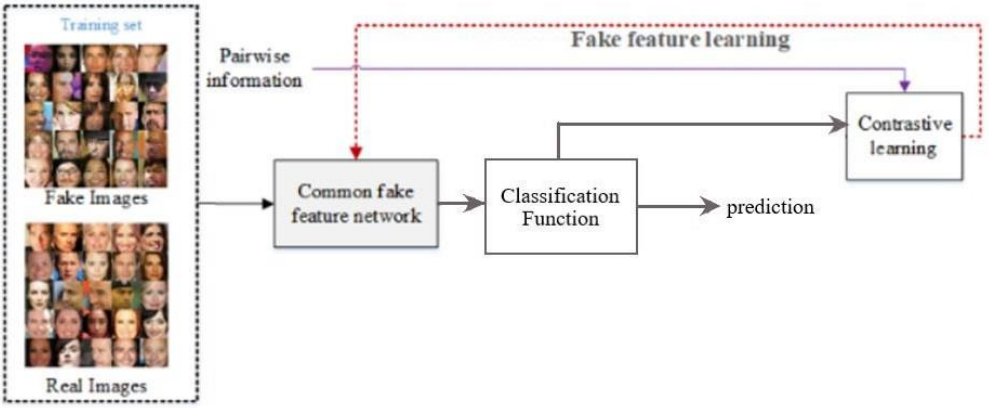

Fig 1.2: Model overview

1.3. Outline of Results

The neural networks will be trained using pairs of fake images and the corresponding real image. Pairwise learning and a simple classifier were used for discriminative feature learning and classification of the images. The data sets used are PGGAN, Style GAN, Star GAN and Celeb A which have approximately more than 200000 real images and 100000 GAN generated images. Out of which 1,00,000 images of fake and real images were used.

After training an image will be passed to the system. The neural network built will try to extract the discriminative features and passes it to the classifier. The classifier will randomly take a set of pictures from the validation set and tries to classify the image given to it. After classification is done the system will predict the label of the image and shows the predicted label to the user. After testing the efficiency of the

proposed application with a training, validation, and test set where in each set the number of fake images was equal to real images, the accuracy of the proposed model was recorded as 99.5%. The model proposed is also compared to Random model to test its efficiency.

1.4. Scope of the project

In the proposed FFN and DeepFD, the fake face and general image detection ability is provided using deep neural networks. Since the main contribution of this work is that the CFFs are learned from the pairwise training samples, the proposed CFFs may fail when the fake features of the results of a new generator are significantly different from most of those used in the training phase. In such a situation, the fake face and general image detector should be retrained. Another limitation of the proposed method is related to the collection of training samples. The technical details of some fake image generators may have not been revealed, so the training samples might be hard to collect in practice. To overcome this limitation, a few-shot learning policy should be employed in the learning of the CFF from a small-scale training set.

This can be improved in the future to detect altered images of documents and altered videos using other machine learning techniques.

1.5. Organization of the report

The project report is organized as follows:

- Chapter 1 entitled “INTRODUCTION” is intended to be standalone and explains everything about the project from problem definition to the future scope. It contains a ‘Methodologies’ sub-chapter in which our project algorithm and other processes are explained briefly. In the ‘outline of the results’ the results obtained on implementing the proposed system are discussed. In the ‘Scope of the project’, the environments where this project can be implemented and its practicality in such situations are explained.

- Chapter 2 discusses the literature survey of this project which gives an insight into the existing systems and the core concepts in the field of our project.

- Chapter 3 discusses the block diagram that depicts the workflow of the developed model and describes the modules that act as a separate chain of sub-models and explains the theoretical foundation of the technology based on which the proposed system has been developed.

- Chapter 4 comprises the implementation details such as flow charts that are needed to display the flow between different entities of the model, the algorithm used in detail along with a brief description of the Dataset and the testing process.

- Chapter 5 discusses the results that display the insight output of the model.

- Chapter 6 concludes the work along with future works and limitations.

2.LITERATURE SURVEY

2.1 Introduction to Problem Domain Terminology

It has already been stated that the primary goal of this project is to be able to determine a fake image. Neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through process that mimics the way the human brain operates. As mentioned earlier two different neural networks to create a modified model to recognize fake images, they are advanced Convolutional Neural Networks and Siamese Neural Network

CNNs are designed to perform feature extraction. Features of input data are measurable properties of observations, which on uses to analyze or classify these instances. When CNN performs feature extraction it chooses relevant features that discriminate the instances well and are independent of each other. One of the advanced CNNs are DenseNets, where each layer obtains additional inputs from all preceding layers and passes on its own feature map to all subsequent layers. It also performs classification, which is defined as the task of assign labels to yet unseen observations. Unlike traditional CNNs, DenseNets concatenate the data which results in collective knowledge. Siamese Neural Network contains two or more identical subnetworks i.e., with same parameters and weights. It is used to find the similarity of inputs by comparing its feature vectors. Siamese neural network is used to train the Fake Feature Model to enable pairwise learning.

Tensorflow is an end-to-end opensource platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets research push the state-of-the-art in machine learning and developers easily build and deploy machine learning powered applications. This platform was used to build the neural network layers.

2.2 Existing Solutions

Image analysis in frequency domain

One of the most popular methods is to analyze images in frequency domain. When images are compressed using JPEG, the compression history is recorded. Analyzing this compression history in the frequency domain can reveal different values in the

manipulated areas, as shown by many researchers. However, the frequency domain analysis-based methods do not work well against images with sophisticated and smoothed edges. Tso address some of the limitations of frequency domain-based detection methods, JPEG Ghost has been proposed. The JPEG Ghost searches for different JPEG quality on the same image and extracts the difference between them. When an image is modified, the forged part is usually copied from other images with different JPEG quality. The normalized pixel distance between an original image and a re-created image with different JPEG quality can clearly show regions with differing image quality. This method, however, might not be useful if the forged areas maintain the same image quality level of the original images, which are not difficult to create.

Error Level Analysis

If the images manipulated are of format JPGE, which is a lossy format, one can easily recognize the regions manipulated using Error Level Analysis. JPGE images maintains stable areas throughout the image which have minimum error. Any modification done to a picture will result in altering of stable areas. ELA tries identifies the areas manipulated since they are no longer at their minimum error level. By analyzing the patter of the images after ELA once can determine which parts of the image are possibly faked. One can integrate machine learning to detect the anomalies in the error level analyzed images too. This will only work well, where parts of an image are modified. However, for GAN generated images, ELA is not efficient.

2.3 Related Work

In the past, various digital image forensic algorithms and tools have been proposed by many researchers. Many prior approaches exploited the properties of image formats and metadata information to determine the authenticity of the images. Despite many forensic tools and new algorithms to detect forgery and fake components in images, it remains a challenging problem because attackers are also employing the latest image processing and machine learning techniques to bypass well-known forgery detection techniques.

There are some investigations for detecting face forgery using deep learning such as FaceForensics in “Is object localization for free? weekly supervised learning

with convolution neural networks." [3] And "Self-attention generative adversarial networks." [4], which generates refined fake face dataset from utilizing Face2Face. Providing more features to deep learning model "Improved training of Wasserstein gans" [6]and "Least square generative adversarial networks" [7] have been proposed for image forgery detection. Specifically, noise features extracted from a steganalysis rich model are used as an input to CNN-based model. However, these approaches may not work well with fake images containing smoothed-edge spliced images, since it is difficult to extract the obvious noise features. More advanced tools such as FotoForensics in "Spectral normalization for generative adversarial networks" [8] and MMC image forensic tool "ImageNet large scale visual recognition challenge" [9] utilize a variety of information such as metadata and JPEG quality. However, sophisticated attackers can easily detour these tools by hiding or modifying metadata information. Also, these approaches do not work for images generated by GANs.

Also, Huh et al. "Detection of FAN-generated fake images over social networks" [5] proposed the state-of-the-art learning algorithm to detect fake imagery based on the self-consistency, which is to determine whether its content could have been produced by a single imaging pipeline. However, their methods require the EXIF metadata during training. Their approach fails to detect many image forgeries created under a strong adversary who can manipulate metadata information, while our approach can detect fake images well.

The model proposed by "Fake image detection using machine learning" [2] To detect GANs generated images uses shallow networks instead of deep neural networks. They also use an adversary model to completely remove the metadata information of the image. The main limitation of their model is that they manually created fake face images to create the model, which is quite time consuming.

2.4 Tools/Technologies Used

2.4.1 Tools Used

- Windows 10 operating system
- Collab Pro
- AWS SageMaker
- 64 GB RAM

- 100 GB drive space
- Tensor Processing Unit

2.4.2 Technologies Used

- Python
- TensorFlow
- Keras
- Tkinter
- Jupyter
- CUDA
- cuDNN

3. DESIGN OF THE PROPOSED SYSTEM

3.1 Block Diagram

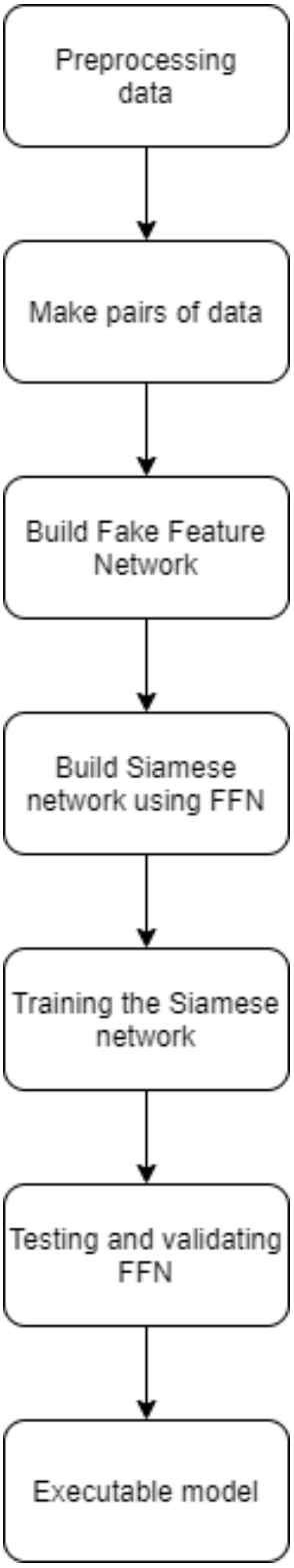

Fig 3.1: Block diagram of proposed model

3.2 Module Description

3.2.1 Pre-processing data

i. Input: Image that must be identified and the target image size.

ii. Processing:

- Changes the image size into the target image size specified through performing stretching, re-sizing on the image.

iii. Output: Modified pre-processed image.

3.2.2 Loading the data

i. Input: Path of the image, batch size, image size

ii. Processing:

- Loads the data specified at the corresponding file path or location for dataset at given location.
- Reads samples and splits the dataset into two sets, i.e., training set and testing set.
- Creates pairs from the training set.
- Provides iterating functions to go through the batches.

iii. Output: Batches of data and their corresponding labels.

3.2.3 Model of the modified network

i. Input: Threshold, learning rate

ii. Processing:

- Contains functions to build a model containing Siamese neural network where the two sister networks are FFNs.
- Creates an executable model which will be used to train.

iii. Output: working model ready to be trained

3.2.5 Training

i. Input: Training data set

ii. Processing:

- The data is loaded and trained with all the samples in batches and stores the accuracy of the system by validation.

- The neural network model automatically saves whenever there is improvement in character error rate.

iii. Output: Trained model

3.2.6 Validation and testing

i. Input: Testing data set

ii. Processing:

- Testing dataset is given to the model to make predictions.
- The predictions are verified with the labels of the image and checked how accurate the model is.

iii. Output: Summary of the model i.e., accuracy, loss, etc.

3.3 Theoretical Foundation/Algorithms

3.3.1 Model Training architecture

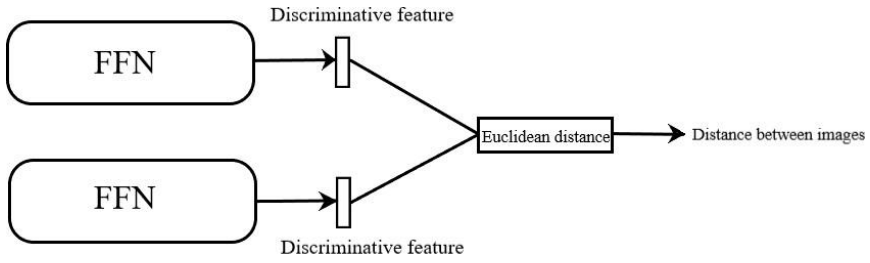

Fig 3.3.1: Model neural network architecture

Fig 3.3.1 depicts the architecture of modified neural network. Modified network is constructed using Siamese neural network with two sister networks. Sister networks have same architecture and parameters. Both networks work and train simultaneously to enable pairwise learning. A pair of images is given to the model where each sister network takes one image from the pair as input. The sister networks then give extract discriminative features as output, which is passed to a function that calculates Euclidean distance between the features. If the distance between two images is less than 0.5, they belong to the same class else they belong to different classes.

3.3.2 FFN

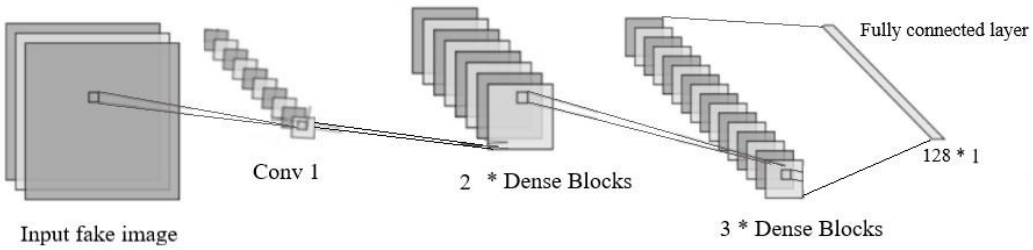

Fig 3.3.2: Design of FFN

The architecture of the sister network is shown in Fig 3.3.2. The sister network initially consists of a convolution layer with 7 x 7 kernel size and 48 channels with stride value being 4. The output of this layer is sent to dense blocks DenseNets with different number of blocks in each unit are used to construct the Fake Feature Network Model (FFN). A total of two dense units containing 2 and 3 blocks, respectively are created. Each unit has 24 and 30 channels, respectively. A translation layer is added in between every dense block, where the input tensor is halved. The dense block consists of batch normalization, ReLu activation and convolutions. Convolution with k * 4 filters and 1 x 1 kernel size are used, where k is the growth rate. Consecutively another batch normalization, ReLu and convolution is done. This time convolutions are done with k number of filters and 3 x 3 kernel size. Two sister networks are trained using discriminative feature learning. The Siamese neural network

Following these is a fully connected layer of 128 neurons i.e, the output of the first dense unit is concatenated with the final output of the last dense unit and is sent to the fully connected layer.

3.3.3 Discriminative Feature Learning

To enhance the performance of the system, contrastive loss was introduced to learn the common fake features. Contrastive loss is extensively used in Siamese neural networks to calculates the distance between two images in vector space. The loss is low if the images are closer and high when images are farther apart. To achieve this, we

incorporate contrastive loss in energy function of traditional loss function for supervised learning. The energy function between the images will be defined as:

$$E_W(\mathbf{x}_1, \mathbf{x}_2) = ||f_{CFFN}(\mathbf{x}_1) - f_{CFFN}(\mathbf{x}_2)||_2^2.$$

Where (x1, x2) is the face image pair.

After incorporating contrastive loss into the function, it will be defined as:

$$L(W, (P, \mathbf{x}_1, \mathbf{x}_2)) = 0.5 \times (y_{ij} E_w^2) + (1 - y_{ij}) \times max(0, (m - E_w)_2^2),$$

Where y indicated pairwise label, that *y=0* indicates an imposter pair and *y=1* indicates a genuine pair. *m* denoted the predefined threshold. When input is genuine, the cost function minimizes the energy by making $(1 - y_{ij}) \times max(0, (m - E_w))^2_2$ equate to zero. When the input pair is an imposter pair the contrastive loss will maximize energy by minimizing function max (0, (m - E_w)). In this way by iteratively training the network using contrastive loss, common fake features of collected GANs can be learned.

3.3.4 N-way One Shot Learning

N-way one shot learning is used to validate the Siamese neural network. The algorithm works in such a way that same character is compared to n different characters out of which only one of them matches the original character. The model must give minimum distance to the one that matches the parameter when compared to the rest. If so, it is treated as a correct prediction. This procedure is repeated k times, and the percentage of correct predictions is calculated as follows.

percent_correct = (100 * n_correct) / k

where k represents the total number of trials and n_correct represents the number of correct predictions out of k trials.

3.3.5 Random Model

Random model is a common technique which is used while testing neural network incorporated with Siamese neural networks. Random model working is similar to that of N-way one shot learning. It classifies the image given as input by comparing it to 16 random images. The set of images it uses to compare the test image is called Support

Set. The model then generates n random numbers between 0 and 1. n random numbers denote the similarity score between the test image and one of the images in the support set. While testing, if the model gives a maximum similarity score to the one it is like, then it is considered as correct prediction.

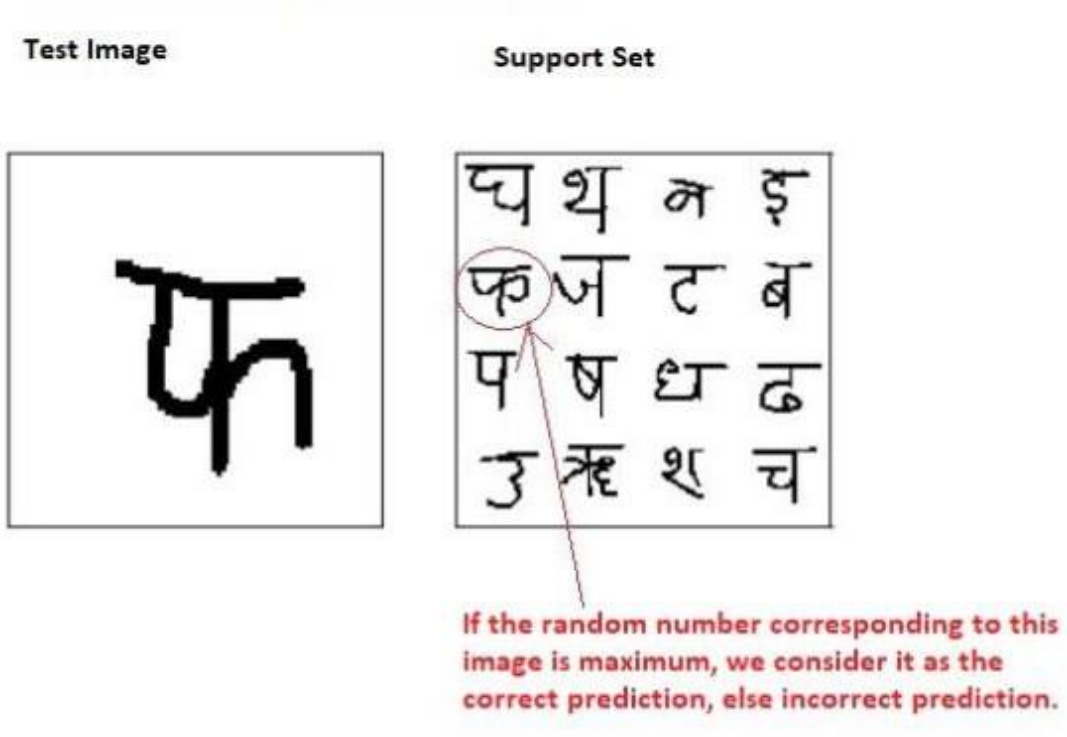

Fig 3.3.5: Random model example

4. IMPLEMENTATION OF THE PROPOSED SYSTEM

4.1 Flow Chart

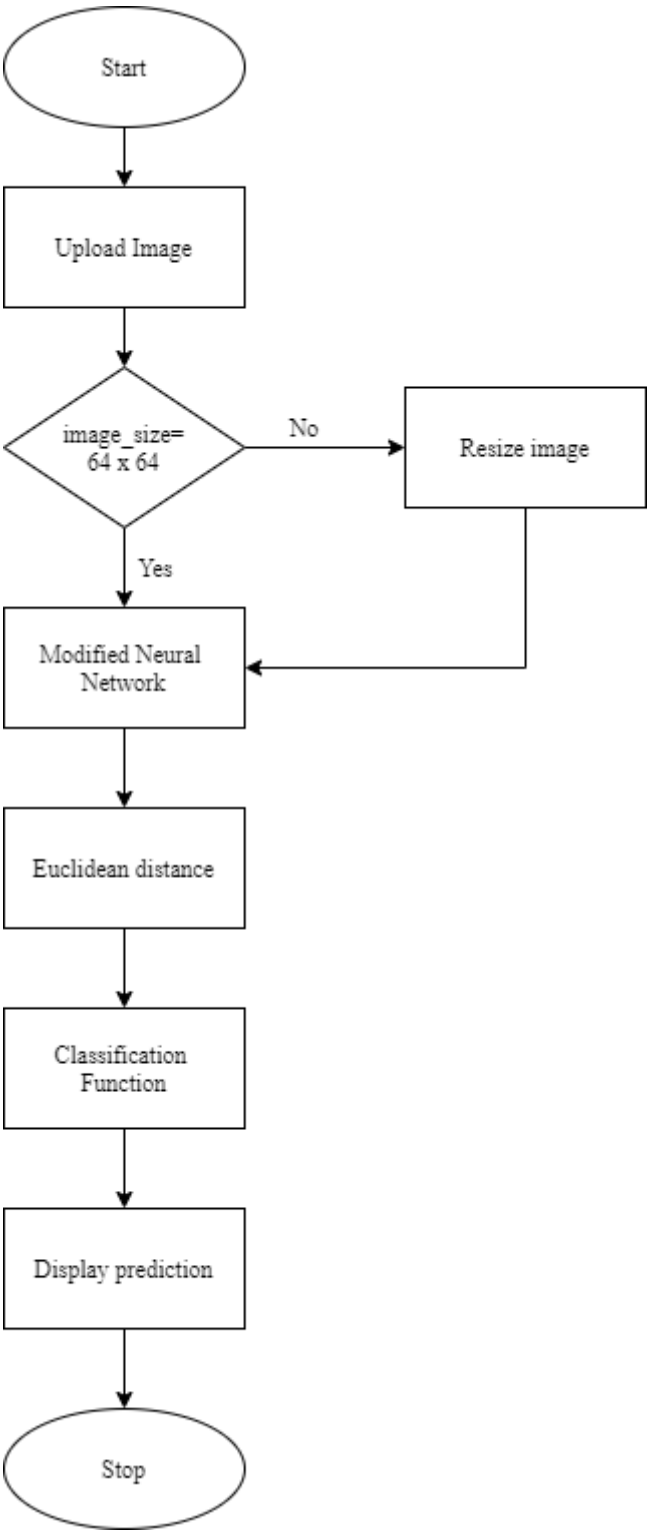

Fig 4: Flowchart of the proposed model after training

The application follows the flow of activities shown in figure 3. The user is expected to upload an image using the GUI. The image uploaded is then checked to see if it matches the input dimensions required. If it does not match the input dimensions required, the image is then resized to 64 x 64 image size. The image is then passed to a trained Fake feature network model which tries to predict the class of the input image.

4.2 Algorithm of the proposed system

Before building the model, the data required to train and validate was prepared by different data processing methods.

Data preprocessing:

The images from the dataset were loaded to the model using the tensorflow.keras.preprocessing.image.ImageDataGenerator class. This class has a function flow_from_directory which loads the images from the directory. For this the directory was organized in such a way that it consisted of two folders which contain images of each class respectively i.e., a folder with all real images and a folder with all fake images.

An instance of the flow_from_directory function is sent as the argument for ninput for tensorflow.keras.Model.fit() function and the parameters were passed to this function in such a way that the images loaded were reshaped to 64x64 pixels and the number of images loaded once per call, i.e., batch size was 16. These images are then passed to the modified neural network to train and validate it.

The algorithm for this system which is designed as illustrated in Fig 4.1 is as follows:

Upload Image:

Once the user executes the application, a Graphical User Interface pops up on the screen. The GUI prompts the user to upload an image which he wants to classify. The dialogs and buttons on the GUI were constructed using Tkinter library in python.

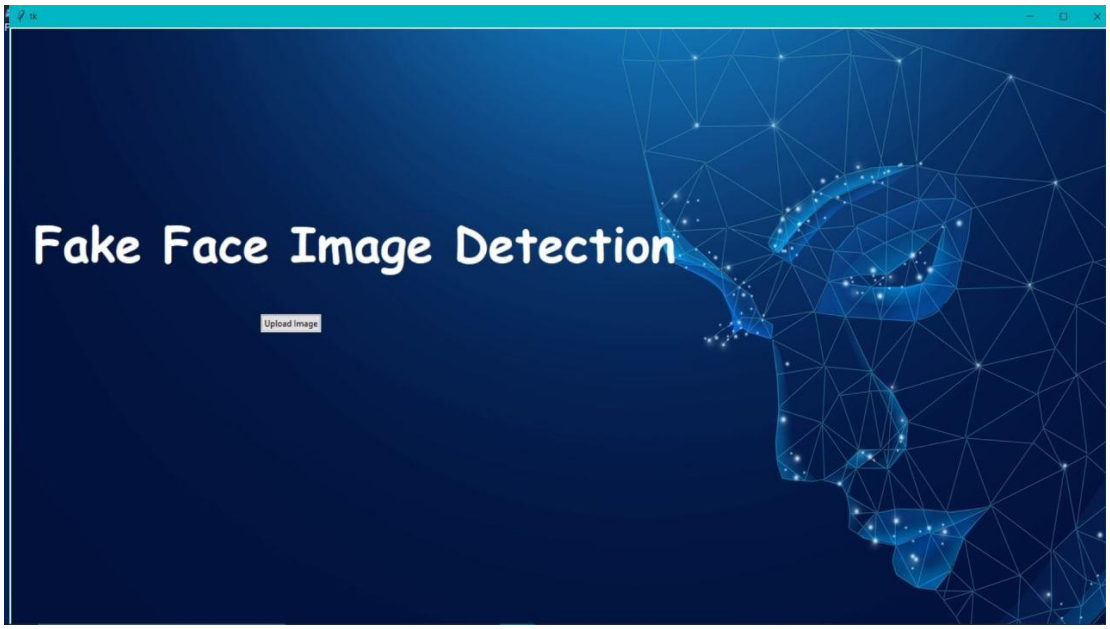

Fig 4.2.1: Screenshot of user Interface

Upon clicking the upload image button, the user is directed to another window which allows him to choose his picture. The selected image`s path is returned and passed to the modified neural network.

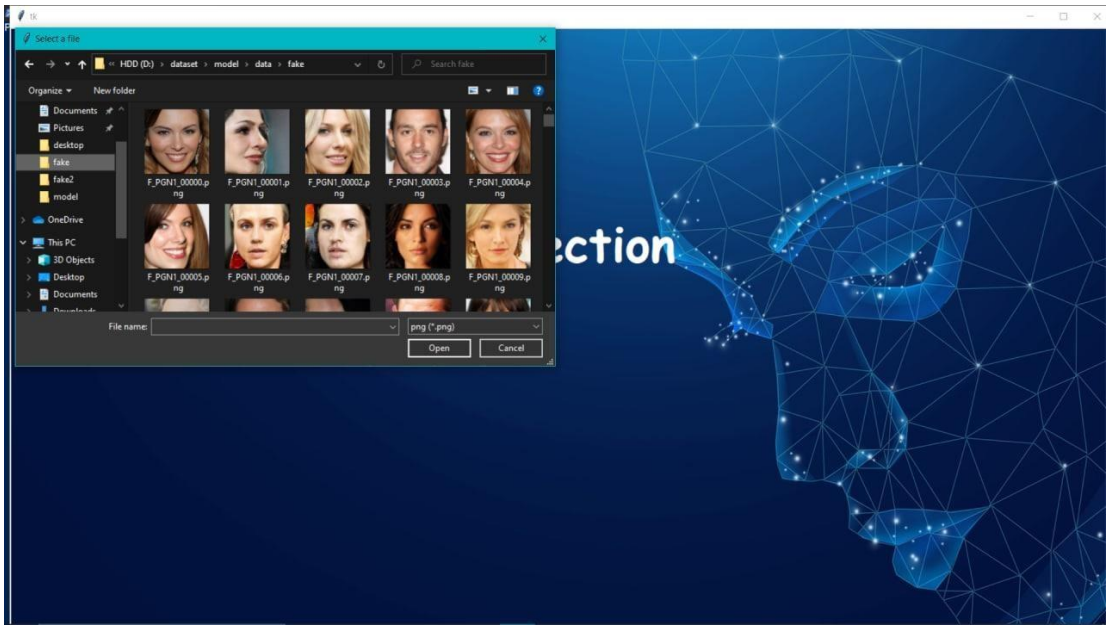

Fig 4.2.2: Screenshot of uploading image

Resize Image:

Dimension of the image uploaded by the user are first checked before sending it to the modified neural network. The input image is passes as it is if its is 64 x 64, else the image is resized using cv2.resize function which is a part of python OpenCV library. The resized image is then passed to the neural network.

Modified neural network:

The FFN model consists of a convolution layer with 48 filters and 7x7 kernel size which was implemented using the 'Conv2D' function of the tensorflow.keras.layers package. The dense units after the convolution layer were implemented using two functions: dense_block() and transition_layer(). The dense_block() function consists of a series of operations on the input tensor. These operations are repeated 'n' times where n is the number of channels and the output from each iteration is concatenated with the previous output and the total tensor is sent as input for the next iteration. The series of operations were done using the function bn_relu_conv() which does batch normalization, relu activation and convolution with a set of filters and kernel size. This function is first called on the input with k*4 number of filters and 1x1 kernel size for the convolution layer. The output is sent to another bn_relu_conv() function but this time with k number of filters and 3x3 kernel size. Here k is the growth rate.

The operations were implemented using 'BatchNormalization()', 'ReLU()', and 'Conv2D()' functions which are a part of the tensorflow.keras.layers package. As each dense unit has multiple dense blocks, there must be a transition layer between two dense blocks. This was implemented using the transition_layer() function which takes the input and halves its size. This was done by passing the input to a bn_relu_conv() function with f number of filters where f is half the size of the last dimension of the input tensor. The output is sent to an average pooling layer which was implemented using the 'AvgPool2D' function of the same tensorflow.keras.layers package. The output of the final dense unit is sent to a fully connected layer with 128 neurons. This layer was implemented using the 'Dense()' function of the same keras package.

A Siamese model was built which takes 2 inputs.The loss function was contrastive_loss() which implemented contrastive loss formula. The training was done for 2 epochs. Adam optimizer was used with default learning rate of 10^-3.

Euclidean distance:

The fake features given from the two sister networks are passed a lambda layer which executes the function euclidean_distance(). The lambda layer was implemented using 'Lambda()' function of the tensorflow.keras.layers package. The euclidean_distance() function takes an array of feature vectors as arguments. These sum of the squares of the difference between these vectors were calculated using tensorflow.keras.backend.sum() and tensorflow.keras.backend.square() functions. The lambda layer is trained as a part of the modified neural network. The Euclidean distance represents the distance between two images in a vector space. If the two images belong to the same class their Euclidean distance is less than 0.5 and more than 0.5 when they belong to different classes.

Classification function:

The classification used works with a simple logic and uses simple python language. When the Euclidean distance is passed to the function it tries to check how close it is to the random picture it is being compared to. If the distance between them is less than 0.5 the random image label is checked, and the input image is classified in the same class as the random image. If the distance between them is higher than 0.5 the input image is classified in a different class than that of the random image`s.

Display Prediction:

Once the image is classified the class name is shown on the GUI through create_text function present in Tkinter library of python.

Fig 4.2.3: Screenshot of prediction

4.3 Data Description

In this project, the dataset was created by extracting images from several sources on the internet. The dataset is mainly divided into two classes: Fake and Real images. Images in Fake class contain face images generated by different GANs. Three different types of GAN generated images have been used to make this class, they are PGGAN, StarGAN and StyleGAN Images in Real class contains real images of different celebrities. Both classes contain around 50,000 images out of which 45,000 images from each class were used to train the model and 5000 images from each class were used to validate the model.

Fake Samples:

This part of the data set contains GAN generated images from PGGAN, StarGAN and StyleGAN.

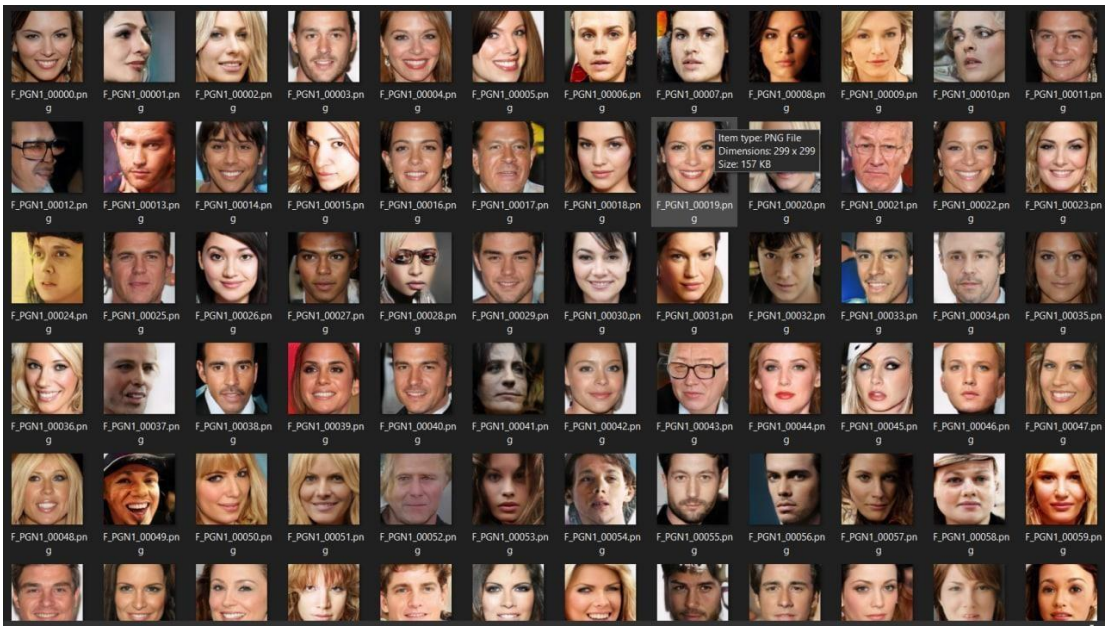

Fig 4.3.1: Screenshot of fake Images

Real Samples:

This part of the data set contains images of different celebrities faces. Most of the images of this dataset are taken from CelebA dataset.

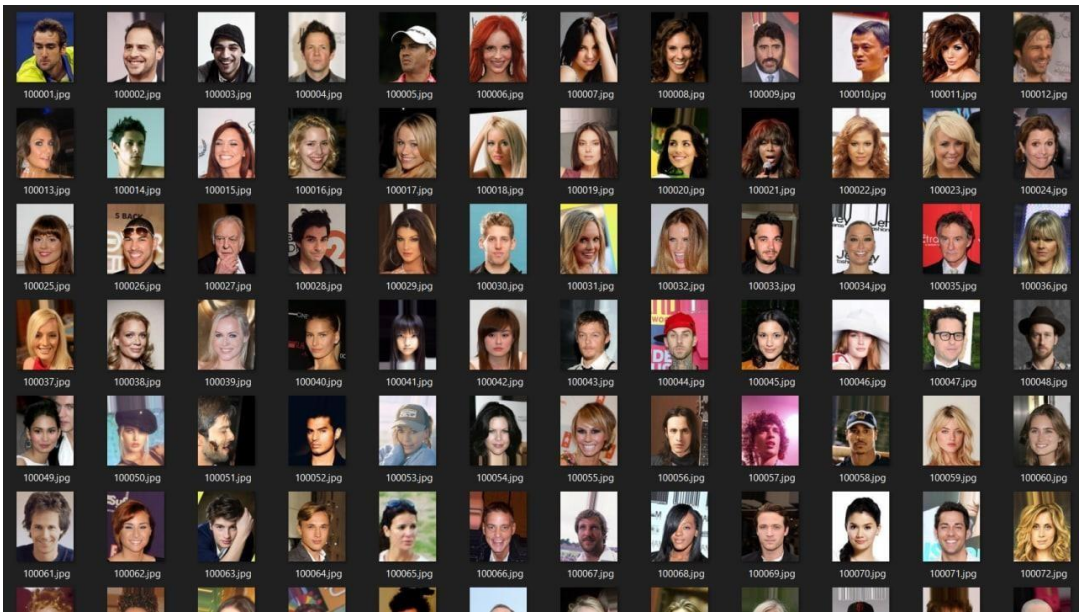

Fig 4.3.2: Screenshot of real images

4.4 Testing Process

The proposed model has been tested using N-way one shot learning technique. Where n number of pairs will be sent to the model out of which only one pair of images belong to the same class. The model is tested k number of times to check how accurately it predicts that one pair. To test this model 2,4,16 and 32 pairs of images have been tested 100 times for both real and fake images. The results are shown in Table 4.4.1.

Table 4.4.1: N-way one shot results

Class	Number of pairs	Repetitions	Accuracy
Real	2	100	100
Real	8	100	100
Real	4	100	100
Real	16	100	100
Real	32	100	99
Fake	2	100	100
Fake	4	100	100
Fake	8	100	100
Fake	16	100	98
Fake	32	100	98

The model has also been tested by changing the batch size of input given to the model. Time taken in training has also been added as a constraint to test the performance of the model. The results are shown in Table 4.4.2

Table 4.4.2: Test results of proposed model

Batch size	**Time taken per epoch**	**Loss**
8	6 hrs	0.0017
16	3.5 hrs	0.0017
32	2 hrs	0.0018
64	1.5 hrs	0.0020

The model created was also tested with a Random model. A random model was created and tested using 2,00,000 images using different n values. 2, 4 8,16 and 32 n values were used to test the model, where each n value prediction was repeated 100 times.

Table 4.4.3: Test results for random model

N Value	**Number of trials**	**Accuracy**
2	100	44
4	100	25
8	100	16
16	100	7
32	100	5

Since the model compares the given image with one real image and one fake image, the accuracy and latency with n number of real images and fake images was also compared. The given image was compared against n number of real images and n number of fake images and if the average distance between for real images was less than that with fake images, the given image was declared as a real image. Otherwise, if the average distance between real images was more than the average distance between fake images, it was declared to be fake. The testing was done with the following n values: 2, 4, 8, 16, 32. A total of 1000 images were used for this testing method. These include 500 fake and 500 real images.

Table 4.4.4 Test results with n number of real and fake images for comparison

N value	**Average latency (ms)**	**Average Accuracy**
2	963.586	97.29
4	1912.837	97.5
8	3908.09	97.3
16	7732.203	96.2
32	14673.223	98.1

5. RESULTS AND DISCUSSIONS

The metrics used in evaluation of this model and results are discusses.

5.1 Evaluation metrics

- Accuracy: The accuracy metric is the measure of all the correctly identified cases. It is the most favored metric of the proposed system since all the classes are equally important.

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

- Loss: Loss is a prediction error of the Neural network. Loss is used to calculate the gradients where gradients are used to update the weights of the neural net. The loss function we used here is

$$L(W,(P,\mathbf{x}_1,\mathbf{x}_2)) = 0.5 \times (y_{ij}E_w^2) + (1 - y_{ij}) \times max(0,(m - E_w)_2^2),$$

- Latency: Latency is the time taken to process one unit of data. Latency is important as it is directly tied with the real time performance of the system.

5.2 Outputs and Discussions

To get the model with low training time per epoch and loss, models with 8,16,32 and 64 batch sizes were test and are show in Table 4.4.1. Based on our testing process, it was found that modified neural model with 8 batch sizes performed better when compared to the rest.

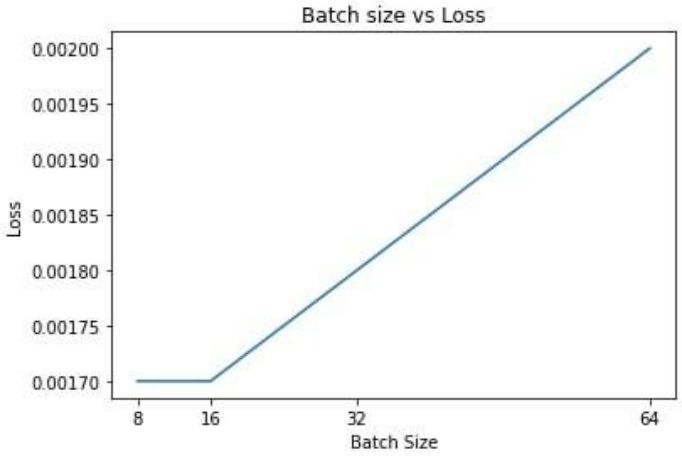

Fig 5.2.1: Batch size vs loss curve

On evaluating the model built over the validation data, the accuracy and loss obtained were 99.5% and 0.0017, respectively.

```
Epoch 1/2
 103/2812 [>.............................] - ETA: 3:22:45 - loss: 3.2688
```

Fig 5.2.2: Output when training

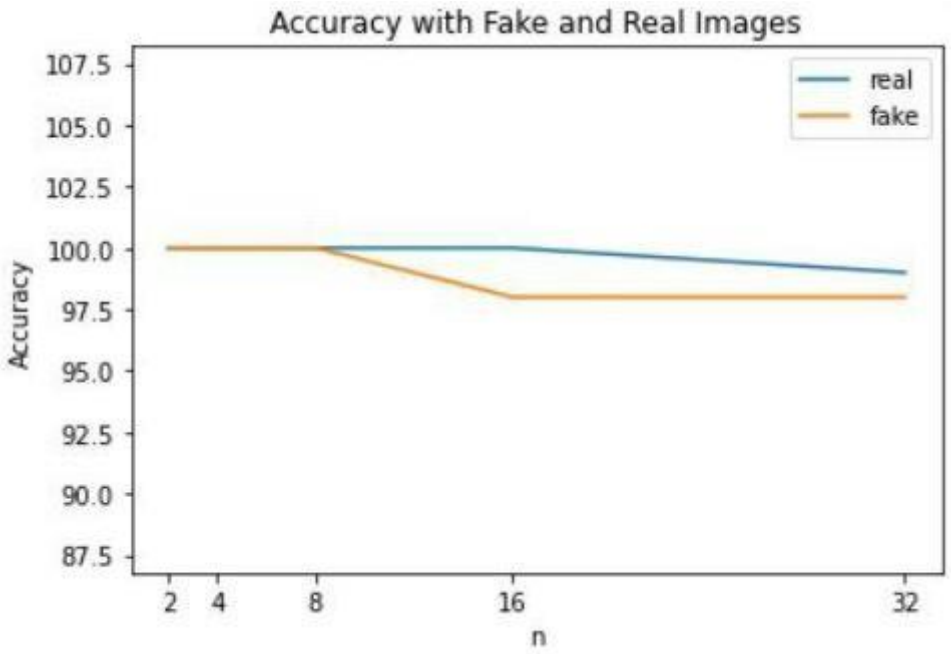

Fig 5.2.3: Accuracy curve

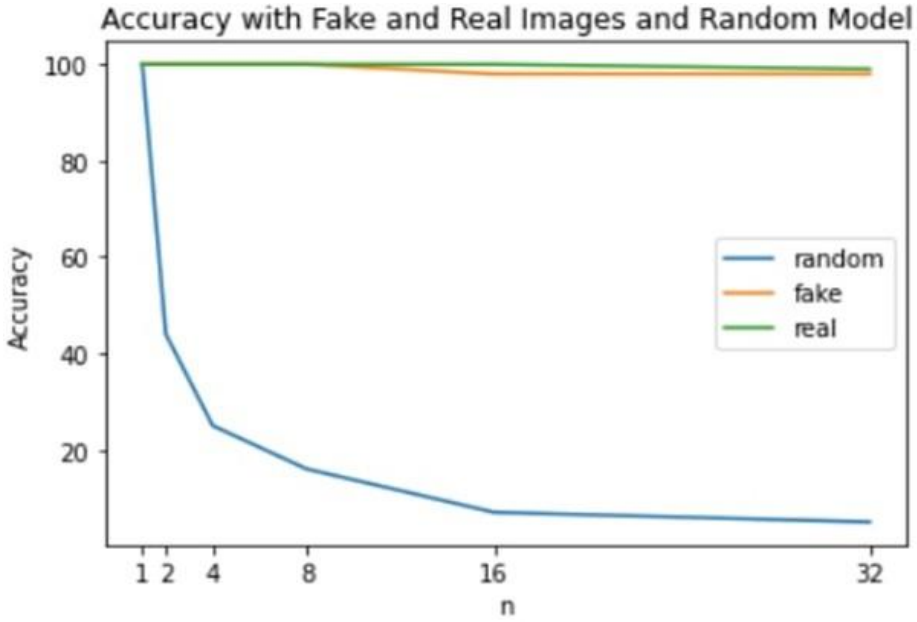

Fig 5.2.4: Random model vs FFN

The model was also tested with a random model. Table 4.4.3 represents the results of the random model. The results were compared to the model proposed and the results are show by Fig 5.2.4. Random model`s performance deteriorated as the number of numbers it generated increased. Whereas the proposed model`s performance was almost linear when batch size increased. After comparing the results of both model FFN model proved to be more efficient.

The model was tested against different number of real images and fake images. This testing method compares the image with n number of fake and real images and the average of distances between each are compared. If the average distance from real images is lesser than that of fake images, the input image is declared as real. Otherwise, if the average distance from fake images is less than that from real images, the input image is declared as fake. The testing was done with the following n values: 2, 4, 8, 16, 32. As shown in Table 4.4.4, we can see that the maximum accuracy is when the given image was compared with 32 images. But the latency of the model using these many images is very high. It takes almost 15 seconds to give the final prediction. The accuracy with different n values has been plotted in Fig 5.2.5 and the latency with different n values is plotted in Fig 5.2.6. Since the accuracy with n=4 is pretty high and the latency is not too high, this value can be considered to be optimal for the prediction of image label.

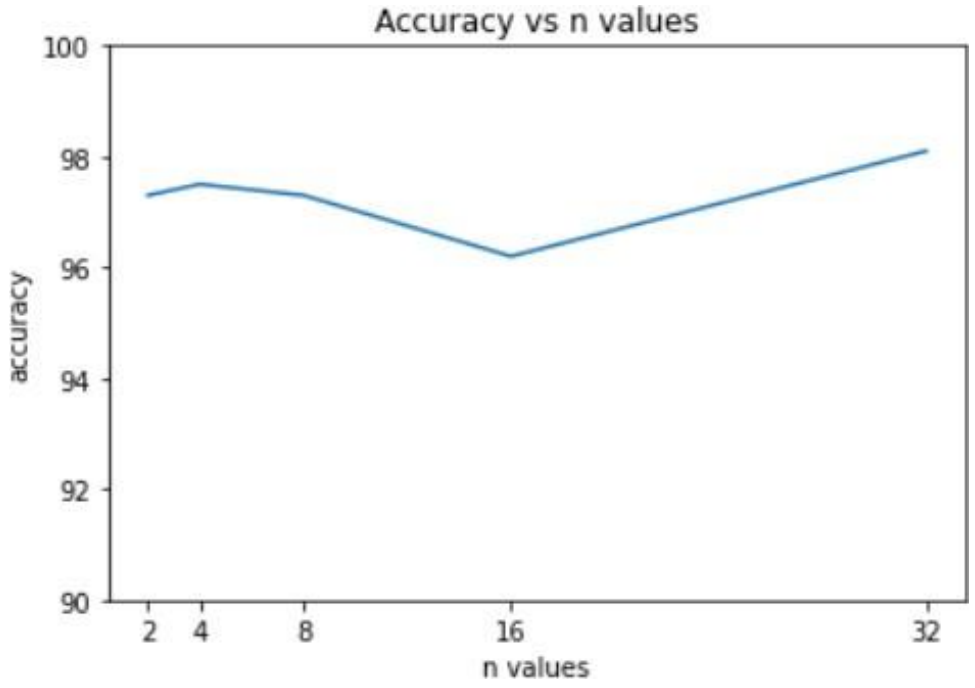

Fig 5.2.5 Accuracy with different n values

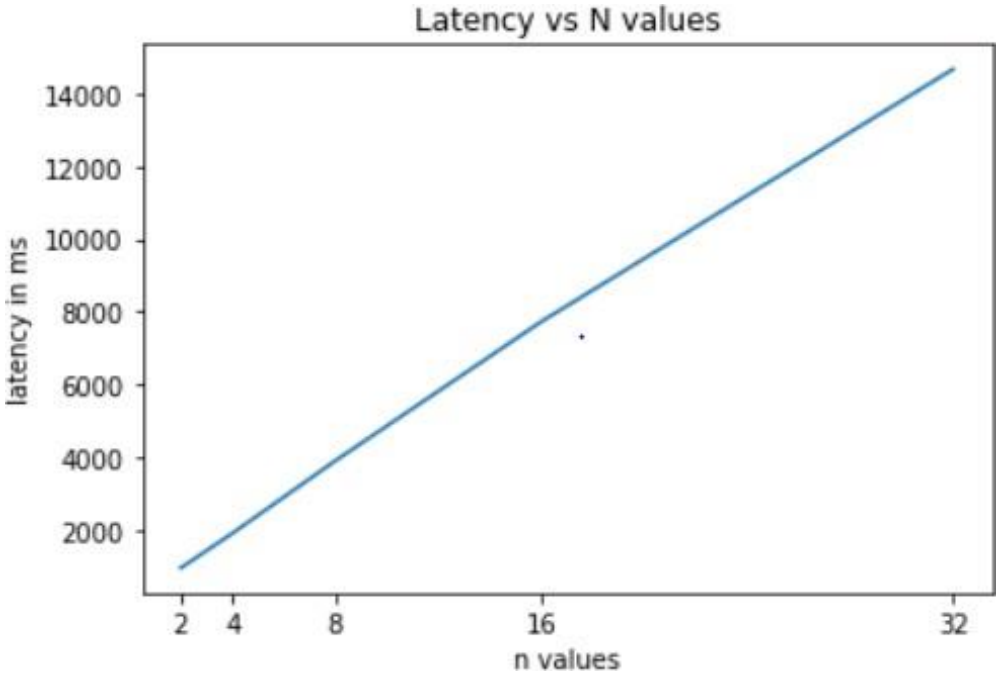

Fig 5.2.6 Latency with different n values

With group images:

The model was also tested with images having multiple faces. Even though it was only trained using single face images the model showed satisfactory results. A series of group real images and fake images were given. Figure 5.2.6 shows the result when real group image in Fig 5.2.5 was given.

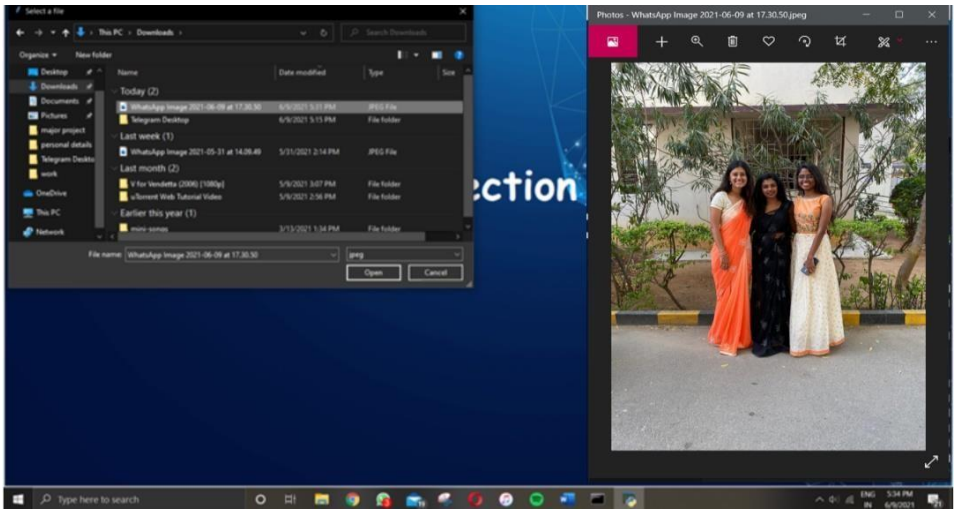

Fig 5.2.7: Screenshot of real group input image

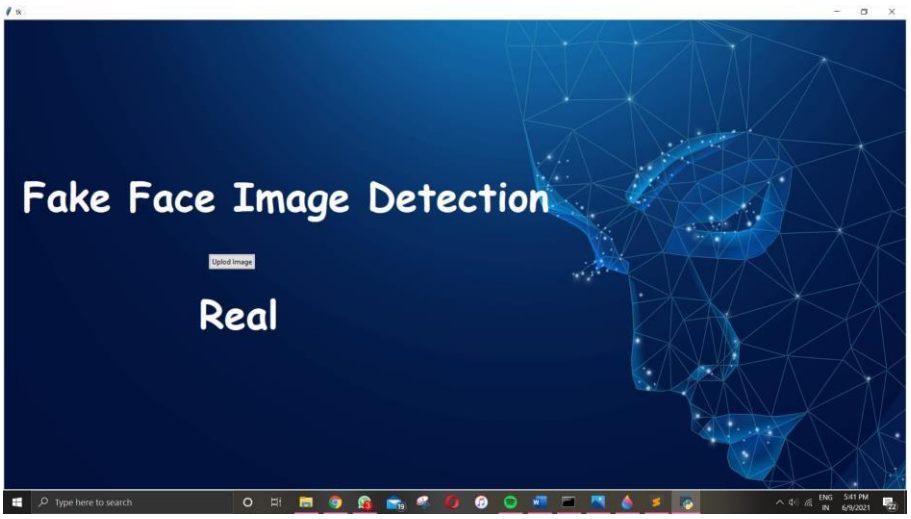

Fig 5.2.8: Screenshot of output of real group image

Fig 5.2.8 shows the prediction given when the same image in Fig 5.2.5 was manipulated and given to the model. The model gave correct predictions in both cases.

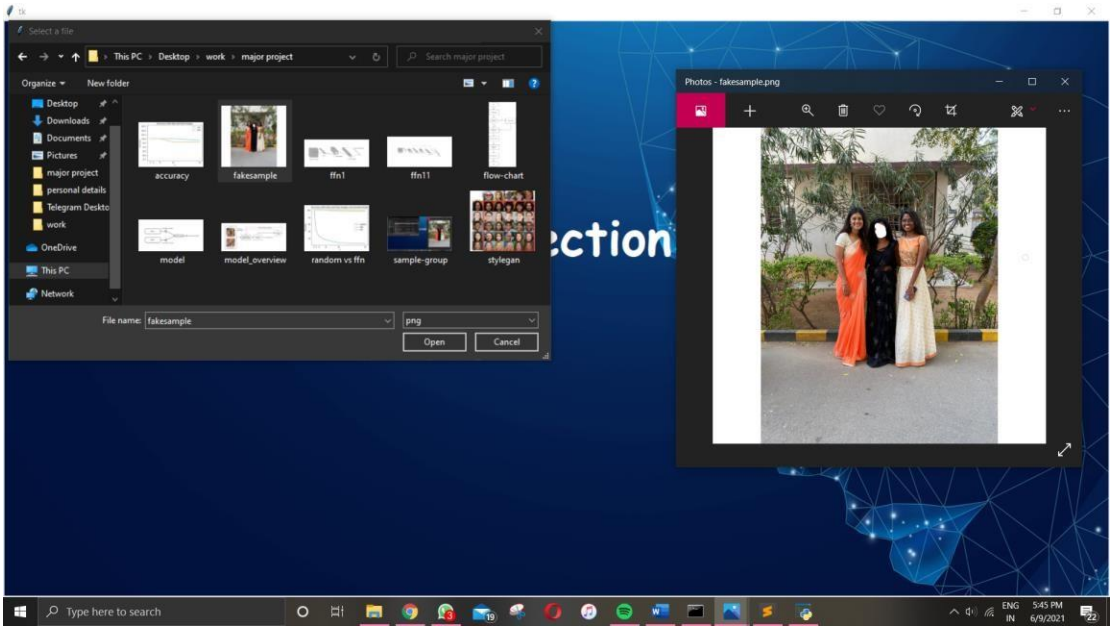

Fig 5.2.9: Screenshot of fake group input image

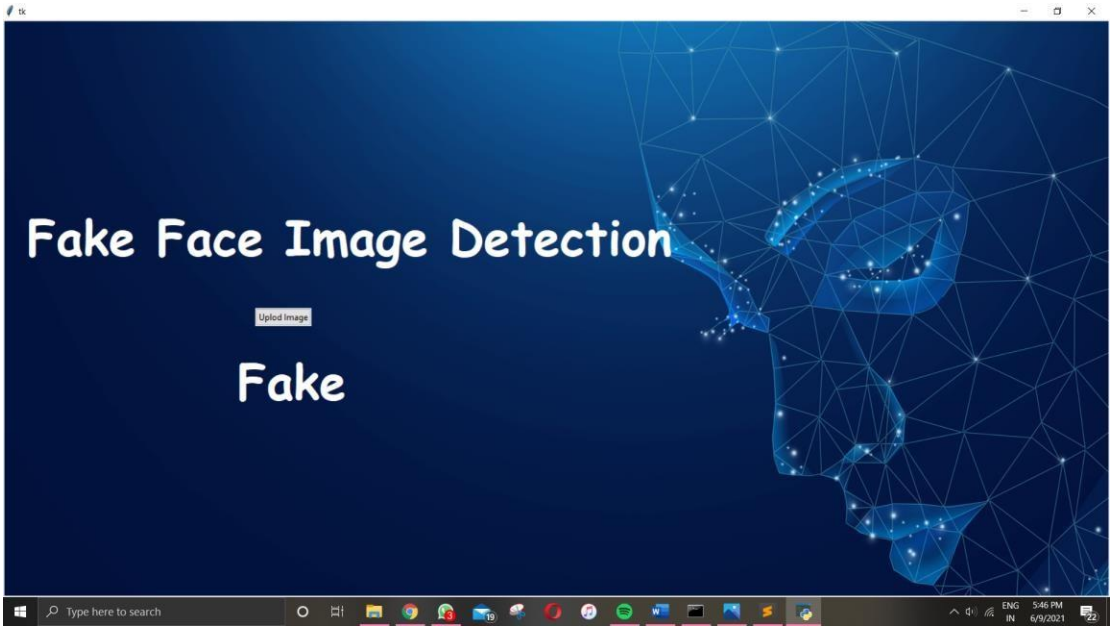

Fig 5.2.10: Screenshot of output of fake group image

The model was given a series of images with multiple faces. It was tested with 100 images containing more than one face, where 50 images were real group images and the rest were fake group images. The model showed impressive result of 60% accuracy in this case.

The model was also tested with real time pictures and videos. The model showed impressive results in case of real time pictures. In case of videos the model took a lot time required to predict and crashed at the end.

6. CONCLUSIONS

6.1 Conclusions

A novel modified neural network is proposed to detect the fake face images generated by state-of-the-art GANs. The proposed model can be used to learn the middle and high-level discriminative features using discriminative feature learning. The proposed strategy allows the trained fake image detector to have the ability to detect the fake image generator by a new GAN, even if it was not included in the training phase.

6.1.1 Limitations

The main limitation of this system is that it requires systems having a minimum RAM of 62 GB and powerful GPU. GPUs with less than 8 GB VRAM were not compatible with the model for training. Since the model has only been trained using single face images it gave satisfactory results with images with multiple imaged were given as input. Using a hybrid model would give better results.

6.2 Recommendations / Future Work

In this project, we have worked on fake face detection and reporting. The ability of this system can be extended to detect fake videos and images with more than one face. One can detect fake news or videos and assures the credibility of the data.

This system can be further extended to identify the GAN that generated the image given by adding more dense layers. Also, by training the model using dataset with multiple classes. Incorporating face detection will help in case of images with multiple faces. The model can used face detection to identify the faces in the image and classify them individually.

REFERENCES

[1] Chih-Chung Hsu, Yi-Xiu Zuhang and Chia-Yen Lee. “Deep Fake Image Detection Based on Pairwise Learning”, *Published in MDPI, Jan 3 2020*

[2] Muhammed Asfal Villain, Johns Paul, Kuncheria Kuruvilla and Eldo P Elias. “Fake Image Detection using Machine Learning” *In proceedings of IEE, Mar-Arpil 2017.*

[3] Oquab, M.; Bottou, L.; Laptev, I.; Sivic, J.” Is object localization for free?-weakly-supervised learning with convolutional neural networks.” *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015.*

[4] Zhang, H.; Goodfellow, I.; Metaxas, D.; Odena, A. “Self-Attention generative adversarial networks”. *In Proceedings of the 36th International Conference on Machine Learning; Chaudhuri, K., Salakhutdinov, R., Eds.; PMLR: Long Beach, CA, USA, 2019*

[5] Marra, F.; Gragnaniello, D.; Cozzolino, D.; Verdoliva, L. “Detection of GAN-Generated Fake Images over Social Networks.” *In Proceedings of the IEEE Conference on Multimedia Information Processing and Retrieval, Miami, FL, USA, 10–12 April 2018.*

[6] Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A.C. “Improved training of wasserstein gans.” *In Proceedings of the Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017.*

[7] Mao, X.; Li, Q.; Xie, H.; Lau, R.Y.; Wang, Z.; Smolley, S.P. “Least squares generative adversarial networks”. *In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017.*

[8] Miyato, T.; Kataoka, T.; Koyama, M.; Yoshida, Y.” Spectral normalization for generative adversarial networks.” *In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.*

[9] Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.;Bernstein, M.; et al. “ImageNet large scale visual recognition challenge.” *Int. J. Comput. Vis. (IJCV) 2015, 115, 211–252.*

APPENDIX

Generative Adversarial Network and its working:

GANs are generative models: they create new data instances that resemble your training data. For example, GANs can create images that look like photographs of human faces, even though the faces do not belong to any real person. A generative adversarial network (GAN) has two parts:

- The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake. As training progresses, the generator gets closer to producing output that can fool the discriminator. Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

Progressive Growing GAN (PGGAN):

Progressive Growing GAN is an extension to the GAN training process that allows for the stable training of generator models that can output large high-quality images.

It involves starting with a very small image and incrementally adding blocks of layers that increase the output size of the generator model and the input size of the discriminator model until the desired image size is achieved.

This approach has proven effective at generating high-quality synthetic faces that are startlingly realistic. Although some images it generates are not correct such as sometimes the hair gets mixed up with forehead, eyes in some photos are not similar to

each other, etc. But overall, the network generates pretty good images and that too in HD.

Style GAN:

Generative Adversarial Networks, or GANs for short, are effective at generating large high-quality images. Most improvement has been made to discriminator models in an effort to train more effective generator models, although less effort has been put into improving the generator models.

The Style Generative Adversarial Network, or StyleGAN for short, is an extension to the GAN architecture that proposes large changes to the generator model, including the use of a mapping network to map points in latent space to an intermediate latent space, the use of the intermediate latent space to control style at each point in the generator model, and the introduction to noise as a source of variation at each point in the generator model.

The resulting model is capable not only of generating impressively photorealistic high-quality photos of faces, but also offers control over the style of the generated image at different levels of detail through varying the style vectors and noise.

Star GAN:

StarGAN is a generative adversarial network capable of learning mappings among multiple domains. It has a unified modeling architecture that allows simultaneous training of multiple datasets and different domains within a single network.
StarGAN is a novel and scalable approach to perform image-to-image translation among multiple.
domains using a single model. It can translate input images to any target domain. It can generate higher visual quality images compared to existing methods. It can perform facial attribute transfer and facial expression synthesis.

Resources Used:

Tensorflow:

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization to conduct machine learning and deep neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well.

TensorFlow provides stable Python and C++ APIs, as well as non-guaranteed backward compatible API for other languages.

Keras:

Keras runs on top of open-source machine libraries like TensorFlow, Theano or Cognitive Toolkit (CNTK). Theano is a python library used for fast numerical computation tasks. TensorFlow is the most famous symbolic math library used for creating neural networks and deep learning models. TensorFlow is very flexible, and the primary benefit is distributed computing. CNTK is deep learning framework developed by Microsoft. It uses libraries such as Python, C#, C++, or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries but difficult to understand for creating neural networks.

Keras is based on minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications.

Keras leverages various optimization techniques to make high level neural network API easier and more performant. It supports the following features

- Consistent, simple and extensible API.
- Minimal structure - easy to achieve the result without any frills.
- It supports multiple platforms and backends.

- It is user friendly framework which runs on both CPU and GPU.
- Highly scalability of computation.

NumPy:

NumPy is a Python library that allows one to work with arrays. It also has functions for dealing with matrices, Fourier transforms, and linear algebra. There are lists in Python that serve as arrays, but they are slow to process. NumPy aims to provide 50 times faster array object than standard Python lists. The array object in NumPy is called ndarray, and it comes with a slew of helper functions to make interacting with it seamless. A developer can use NumPy to perform the following tasks:

a. Array operations (mathematical and logical).

b. Shape manipulation using Fourier transforms and routines.

c. Operations related to linear algebra. NumPy includes linear algebra and random number generation features.

Matplotlib:

Matplotlib is a visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. Matplotlib comes with a wide variety of plots. Plots helps to understand trends, patterns, and to make correlations. It allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Tkinter:

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows, and Mac OS X installs of Python. The name Tkinter comes from Tk interface.

Keras Layers API:

Layers are the basic building blocks of neural networks in Keras. A layer consists of a tensor-in tensor-out computation function (the **layer's** call method) and some state, held in TensorFlow variables (the layer'**s** weights).

Google Colab:

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.

AWS:

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 200 fully featured services from data centers globally. Millions of customers—including the fastest-growing startups, largest enterprises, and leading government agencies—are using AWS to lower costs, become more agile, and innovate faster.

AWS SageMaker:

Amazon SageMaker is a cloud machine-learning platform that was launched in November 2017. SageMaker enables developers to create, train, and deploy machine-learning models in the cloud. SageMaker also enables developers to deploy ML models on embedded systems and edge-devices.

OpenCV:

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source Apache 2 License.

CUDA:

CUDA C++ is just one of the ways you can create massively parallel applications with CUDA. It lets you use the powerful C++ programming language to develop high

performance algorithms accelerated by thousands of parallel threads running on GPUs. Many developers have accelerated their computation- and bandwidth-hungry applications this way, including the libraries and frameworks that underpin the ongoing revolution in artificial intelligence known as Deep Learning..

NVIDIA cuDNN:

The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. Deep learning researchers and framework developers worldwide rely on cuDNN for high-performance GPU acceleration. It allows them to focus on training neural networks and developing software applications rather than spending time on low-level GPU performance tuning. cuDNN accelerates widely used deep learning frameworks, including Caffe2, Chainer, Keras, MATLAB, MxNet, PaddlePaddle, PyTorch, and TensorFlow.

Jupyter Notebook:

The Jupyter Notebook is an open-source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. The Jupyter Notebook has several menus that you can use to interact with your Notebook. The menu runs along the top of the Notebook just like menus do in other applications.

File menu is used to create a new Notebook or open a pre-existing one. This is also where you would go to rename a Notebook. I think the most interesting menu item is the Save and Checkpoint option. This allows you to create checkpoints that you can roll back to if you need to.

Edit menu can be used to cut, copy, and paste cells. This is also where you would go if you wanted to delete, split, or merge a cell. You can reorder cells here too.

View menu is useful for toggling the visibility of the header and toolbar. You can also toggle Line Numbers within cells on or off. This is also where you would go if you wanted to mess about with the cell's toolbar.

Insert menu is just for inserting cells above or below the currently selected cell.

Cell menu allows you to run one cell, a group of cells, or all the cells. You can also go here to change a cell's type, although I personally find the toolbar to be more intuitive for that.

Kernel cell is for working with the kernel that is running in the background. Here you can restart the kernel, reconnect to it, shut it down, or even change which kernel your Notebook is using.

Widgets menu is for saving and clearing widget state. Widgets are basically Java Script widgets that you can add to your cells to make dynamic content using Python (or another Kernel).

Help menu, which is where you go to learn about the Notebook's keyboard shortcuts, a user interface tour, and lots of reference material.