A Mini project Report on

"FAKE IMAGE IDENTIFICATION USING MACHINE LEARNING"

Submitted To The

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

In partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

BURABATHULA CHANDU (19TP1A0520)

UNDER THE GUIDANCE OF

Dr.S.KRISHNA MOHAN RAO
Professor

DEPARTMENT OF COMPUER SCIENCE & ENGINEERING

SIDDHARTHA INSTITUTE OF ENGINEERING ANDTECHNOLOGY

(Approved by AICTE, Accredited by NBA & NAAC A+, Affiliated to JNTU Hyderabad)
Vinobhanagar (V), Ibrahimpatnam (M), R.R. Dist. 501506 2022-2023

SIDDHARTHA College Code -TP

INSTITUTE OF ENGINEERING & TECHNOLOGY

(Accredited by NBA, Approved by AICTE & Affiliated toJNTUH)

Vinobha Nagar, Ibrahimpatnam, Ranga Reddy Dist – 501 506, Telangana,

INDIA.Ph: 08414-222299, 222599, Fax: 08414-222399

E-mail:info@siddhartha.ac.in; www.siddhartha.ac.in

CERTIFICATE

This is to certify that the project entitled " FAKE IMAGE IDENTIFICATION USING MACHINE LEARNING" is being Submitted by BURABATHULA CHANDU – 19TP1A0520 in partial fulfilment for the requirement of the award of BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING of the Jawaharlal Nehru Technological University, Hyderabad, during the academic year 2022-2023.

PROJECT GUIDE
Dr.S.Krishna Mohan Rao
M.Tech, Ph.D.,
Professor.

HEAD OFTHE DEPARTMENT
Mr. P. Raghu
M. Tech, (Ph.D,)
Associate Professor.

PROJECT SUPERVISOR

EXTERNAL EXAMINER

DECLARATION

This is to certify that the work reported in titled "FAKE IMAGE IDENTIFICATION USING MACHINE LEARNING" submitted to the Department of COMPUTER SCIENCE AND ENGINEERING, SIET, Ibrahimpatnam, RangaReddy in partial fulfilment of degree for the award of Bachelor of Technology,

No part of this project is copied from any books, journals and wherever the portion is taken, the same has duly need referred in the text. The reported results are based on the project work entirely done by me and not copied from any other source.

My further state to the best of my knowledge that the matter embedded in this project has not been submitted by me in full or partial there of the award of any degree to any other institution or University.

BURABATHULA CHANDU (19TP1A0520)

ACKNOWLEDGEMENT

First and foremost, my sincere thanks to my Internal Guide Dr.S.KRISHNA MOHAN RAO Professor, who has given his full effort in guiding the team, achieving the goals as well as his encouragement to maintain my progress in track.

My indebted to my Project Co-Ordinator G.UDAY KUMAR Assistant Professor ,for his excellent guidance, constant inspiration and encouragement in completion of the project work My Sincere thanks to Mr.P.RAGHU Head of the Department of CSE for his valuable suggestions and advices during my completion of the project.

A Sincere thanks to Dr. ESHWARA PRASAD KOORAPATI, Principal, SIDDHARTHA INSTITUTE OF ENGINEERING AND TECHNOLOGY, Vinobha Nagar , Ibrahimpatnam, Hyderabad, for providing this opportunity to carry out the project work and for his encouragement during the work.

I would like to express my thanks to all the faculty's members for having faith in me in every possible way whenever the need arose. On a personal note, I thank my beloved parents and friends for their moral support during the course of my project.

BURABATHULA CHANDU (19TP1A0520)

CONTENTS

ABSTRACT

Now-a-days biometric systems are useful in recognizing person’s identity but criminals change their appearance in behavior and psychological to deceive recognition system. To overcome from this problem we are using new technique called Deep Texture Features extraction from images and then building train machine learning model using CNN (Convolution Neural Networks) algorithm. This technique refer as LBPNet or NLBPNet as this technique heavily dependent on features extraction using LBP (Local Binary Pattern) algorithm.

In this project we are designing LBP Based machine learning Convolution Neural Network called LBPNET to detect fake face images. Here first we will extract LBP from images and then train LBP descriptor images with Convolution Neural Network to generate training model. Whenever we upload new test image then that test image will be applied on training model to detect whether test image contains fake image or non-fake image. Below we can see some details on LBP.

Local binary patterns (LBP) is a type of visual descriptor used for classification in computer vision and is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and consider the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings.

The LBP feature vector, in its simplest form, is created in the following manner:

Divide the examined window into cells (e.g 16x16 pixels for each cell).

For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.

Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).

Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.

Optionally normalize the histogram. Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.

The feature vector can now be processed using the Support vector machine, extreme learning machines, or some other machine learning algorithm to classify images. Such classifiers can be used for face recognition or texture analysis.

List of Figures

LIST OF SCREEN SHORTS

CHAPTER-1
INTRODUCTION

1.1General:

Recently, the generative model based on deep learning such as the generative adversarial net (GAN) is widely used to synthesize the photo-realistic partial or whole content of the image and video. Furthermore, recent research of GANs such as progressive growth of GANs (PGGAN) and Big GAN could be used to synthesize a highly photo-realistic image or video so that the human cannot 20 recognize whether the image is fake or not in the limited time. In general, the generative applications can be used to perform the image translation tasks [3]. However, it may lead to a serious problem once the fake or synthesized image is improperly used on social network or platform. For instance, cycle GAN is used to synthesize the fake face image in a pornography video [4]. Furthermore, GANs may be used to create a speech video with the synthesized facial content of any famous politician, causing severe problems on the society, political, and commercial activities. Therefore, an effective fake face image detection technique is desired. In this paper, we have extended our previous study associated with paper ID #1062 to effectively and efficiently address these issues.

In traditional image forgery detection approach, two types of forensics scheme are widely used: active schemes and passive schemes. With the active schemes, the externally additive signal (i.e., watermark) will be embedded in the source image without visual artifacts. In order to identify whether the image has tampered or not, the watermark extraction process will be performed on the target image to restore the watermark[6]. The extracted watermark image can be used to localize or detect the tampered regions in the target image. However, there is no "source image" for the generated images by GANs such that the active image forgery detector cannot be used to extract the watermark image. The second one-passive image forgery detector–uses the statistical information in the source image that will be highly consistency between different images. With this property, the intrinsic statistical information can be used to detect the fake region in the image [7][8]. However, the passive image forgery detector cannot be used to identify the fake image generated by GANs since they are synthesized from the low-dimensional random vector. Nothing change in the generated image by GANs because the fake image is not modified from its original image

Intuitively, we can adopt the deep neural network to detect the fake image generated by GAN. Recently, there are some studies that investigate a deep learning-based approach for fake image detection in a supervised way. In other words, fake image detection can be treated as a binary

classification problem (i.e., fake or real image). For example, the convolution neural network (CNN) network is used to learn the fake image detector [9]. In [10], the performance of the fake face image detection can be further improved by adopting the most advanced CNN–Xception network [11]. However, there are many GANs proposed year by year. For example, recently proposed GANs such as [1][12][13][14][15][16][3][2] can be used to produce the photo-realistic images. It is hard and very time-consuming to collect all training samples of all GANs. In addition, such a supervised learning strategy will tend to learn the discriminative features for a fake image generated by each GAN s. In this situation, the learned detector may not be effective for the fake image generated by another new GAN excluded in the training phase.

In order to meet the massive requirement of the fake image detection for GANs-based generator, we propose novel network architecture with a pairwise learning approach, called common fake feature network (CFFN). Based on our previous approach [5], it is clear that the pairwise learning approach can overcome the shortcomings of the supervised learning-based CNN such as methods in [9][10]. In this paper, we further introduce a novel network architecture combining with pairwise learning to improve the performance of the fake image detection. To verify the effectiveness of the proposed method, we apply the proposed deep fake detector (DeepFD) to identify both fake face and generic image. The primary contributions of the proposed method are two-fold:

• We propose a fake face image detector based on the novel CFFN consisting of several dense blocks to improve the representative power of the fake image.

• The pairwise learning approach is first introduced to improve the generalization property of the proposed DeepFD.

A. Error Level Analysis

JPEG is a lossy format, but the amount of error introduced by each resave is not line. Any modification to the picture will alter the image such that stable areas (no additional error) become unstable. Fig.1 shows a modified image usingPhotoshop. The modified picture was based on the first 75%resave. Books on the shelf were duplicated and a toy dinosaur was added to the shelf. The 95% ELA identifies the changes since they are areas that are no longer at their minimal error level. Additional areas of the picture show slightly more volatility because Photoshop merged information from multiple layers, effectively modifying many of the pixels. A 90% image resaved at 90% is equivalent to a one-timesaver of 81%. Similarly, saving an image at 75% and then resaving it at 90% (75% 90%) will generate virtually the same image as 90% 75%, or saved once at 67.5%. 19 The amount of error is limited to the 8x8 cells used by the JPEG algorithm; after roughly 64 resaves, there is virtually no change. However, when an image is modified, the 8x8 cells containing the

modifications are no longer at the same error level as the rest of the unmodified image. Error level analysis (ELA) works by intentionally resaving the image at a known error rate, such as95%, and then computing the difference between the images. If there is virtually no change, then the cell has reached its local minima for error at that quality level. However, if there is a large amount of change, then the pixels are not at their local minima and are effectively "original". JPEG is a lossy format, but the amount of error introduced by each resave is not linear modification to the picture will alter the image such that stable areas (no additional error) become unstable [1].

Books on the shelf were duplicated and a toy dinosaur was added to the shelf. The 95% ELA identifies the changes since they are areas that are no longer at their minimal error level. Additional areas of the picture show slightly more volatility because Photoshop merged information from multiple layers, effectively modifying many of the pixels. Nearly all pixels in the original image are not at their local minima. The first resave(75%) shows large areas where the pixels have reached their local minima. The second resave introduces more areas that have reached their local minima for error. By analyzing the pattern in the ELA applied image (Fig 1left part), we can determine which part of the image is possibly faked. It is hard for the human eye to detect small scale changes to image so that we have decided to use machine learning to detect the anomalies in the error level analyzed images.

Common Fake Feature Network

Many advanced CNN can be used to learn the fake features from the training set. Xception Network was used in [11] to capture the powerful feature from the training images in a purely supervised way. Other advanced CNNs, such as DenseNet [19], ResNet [20], Xception [12], can also be applied to the fake face detector training. However, most of these advanced CNNs are trained in a supervised way, so the classification performance depends on the training set. Rather than learn the fake features from all the GANs' images, we seek the CFF over different GANs. In this way, a suitable backbone network is needed for learning CFFs. However, the traditional CNNs (e.g., the DenseNet [19]) are not designed to learn the discriminative CFF. To overcome this shortcoming, we propose integrating Appl. Sci. 2020, 10, 370 4 of 14 the Siamese network with the DenseNet [19], developing the CFFN to achieve the discriminative CFF learning. A dense block is a basic component in the DenseNet [19], which is one of the state-of-the-art CNN models for image recognition. However, it is trained by the supervised learning strategy, while the proposed pairwise learning strategy for the CFFs denotes a semi-supervised learning strategy. The proposed CFFN is a two-streamed network designed to allow the pairwise input for CFF learning. On the other hand, the traditional CNNs, which are single-streamed networks, are unable to receive the paired information; thus, the common features can be difficultly learned by the traditional CNNs. In the proposed CFFN, the backbone network can be any of the advanced CNNs, such as ResNet [20], Xception [12], or DenseNet [19]. Once the backbone network is

trained to have the best feature representation ability, the performance of the fake image recognition can be improved as well. To this end, DenseNet is selected as a backbone network of the proposed CFFN. Moreover, it is well known that CNNs capture the hierarchical feature representation from a low level to a high level. In other words, the CNNs use only on high-level feature representation to identify whether the image is fake or not. However, the CFFs of fake face images may not exist only in the high-level representation but also in the middle-level feature representation. Inspired by [21], in this work, the cross-layer features are integrated into the classification layer to improve the fake image recognition performance, as shown in Figure 2.

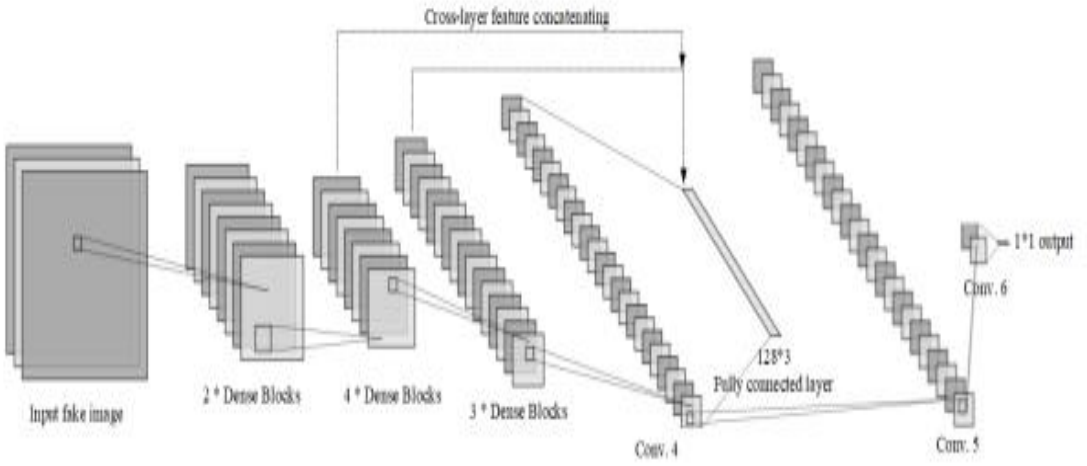


Figure 2. The structure of the proposed common fake feature network.

The proposed CFFN consists of three dense units that include two, four, and three dense blocks, respectively, and the number of channels in the three dense units are 48, 60, 78, and 126, respectively. The parameter θ in the transition layer is 0.5 and the growth rate is 24. Then, a convolution layer with 128 channels and 3 × 3 kernel size is concatenated to the output layer of the last dense unit. Finally, the fully connected layer is added to obtain the discriminative feature representation. To obtain the cross-layer feature representation, we also reshape the last layers of the first and second dense units to aggregate the cross-layer features into the fully connected layer. Therefore, in the final feature representation, there are 128 × 3 = 384 neurons. In general, the classification of fake image can be performed by a different classification learning model, such as random forest, SVM, or Bayes classifier. However, the discriminative feature may be further improved by applying the back-propagation algorithm to the end-to-end structure. Therefore, in this work, the convolution and fully connected layers are concatenated into the last convolution layer of the proposed CFFN to obtain the final decision result.

1.2 PROJECT DESCRIPTION

A project on fake image identification using machine learning would involve building a system that is able to automatically detect whether an image is real or fake. This would be done using machine learning algorithms, which are able to learn patterns and features in data and use them to make predictions.

To build such a system, you would need to gather a large dataset of real and fake images, and use this dataset to train your machine learning model. This would involve feeding the model the images in the dataset and labeling them as either real or fake, so that the model can learn to recognize the features that are indicative of image manipulation.

Once your model is trained, you can test it on a variety of images to see how well it performs. This might include testing the model on images that have been manipulated using different techniques, images that vary in terms of content and resolution, and images that have been partially manipulated.

The goal of the project would be to develop a system that is able to accurately and reliably detect fake images, and that can be used to identify manipulated images in a variety of contexts. Some potential applications of such a system could include detecting and removing fake images from social media platforms, identifying manipulated images in the media, and helping law enforcement agencies detect and investigate cases of image manipulation.

1.3 Methodologies

There are several different methodologies that can be used in a project on fake image identification using machine learning. Some potential approaches include:

Supervised learning: This approach involves training a machine learning model on a labeled dataset of real and fake images, and using the labeled data to teach the model to classify new images.

Unsupervised learning: In this approach, the model is not provided with labels for the images in the training dataset. Instead, it must learn to identify patterns and features in the data that indicate whether an image is real or fake.

Transfer learning: In this approach, a pre-trained machine learning model is fine-tuned on a new dataset of real and fake images. This can be a faster and more efficient way to build a fake image detection model, as the pre-trained model has already learned to recognize many of the features that are relevant to the task.

Deep learning: This approach involves training a deep neural network on the dataset of real and fake images. Deep learning models are able to learn complex patterns in the data and can achieve very high performance on tasks such as image classification.

Which approach is most suitable will depend on the specific requirements of the project, including the size and quality of the available dataset, the desired level of performance, and the resources and time available for training and evaluation.

CHAPTER 2
LITERATURE SURVEY

Very little work has been finalized around detecting forge audio, images, and videos. Yet, several studies and tasks are underway to identify what can be done around the incredible proliferation of counterfeit pictures online. Adobe recognizes the way in which Photoshop is misused and has tried to offer a sort of antidote [8]. The following provides a summary of a few of these studies: According to a study [9] conducted by Zheng et al. (2018), the identification of fake news and images is very difficult, as fact-finding of news on a pure basis remains an open problem and few existing models Can be used to resolve the problem. It has been proposed to study the problem of "detecting false news." Through a thorough investigation of counterfeit news, many useful properties are determined from text words and pictures used in counterfeit news. There are some hidden characteristics in words and images used in fake news, which can be identified through a collection of hidden properties derived from this model through various layers. A pattern called TI-CNN has been proposed. By displaying clear and embedded features in a unified space, TI-CNN is trained with both text and image information at the same time. Raturi's 2018 architecture [10] was proposed to identify counterfeit accounts in social networks, especially on Facebook. In this research, a machine learning feature was used to better predict fake accounts, based on their posts and the placement on their social networking walls. Support Vector Machine (SVM) and Complement Naïve Bayes (CNB) were used in this process, to validate content based on text classification and data analysis. The analysis of the data focused on the collection of offensive words, and the number of times they were repeated. For Facebook, SVM shows a 97% resolution where CNB shows 95% accuracy in recognizing Bag of Words (BOW) -based counterfeit accounts. The results of the study confirmed that the main problem related to the safety of social networks is that data is not properly validated before publishing. In a 2017 study by Bunk et al [11], two systems were proposed to detect and localize fake images using a mix of resampling properties and deep learning. In the initial system, the Radon conversion of resampling properties is determined on overlapping pictures corrections. Deep learning classifiers and a Gaussian conditional domain pattern are then used to construct a heat map. A Random Walker segmentation method uses total areas. In the next system, for identification and localization, software resampling properties are passed on overlapping object patches over a long-term memory (LSTM)-based network. In addition, the detection/ localization performance of both systems was compared. The results confirmed that both systems are active in detecting and settling digital image fraud. Aphiwongsophon and Chongstitvatana [12], aimed to use automated learning techniques to detect counterfeit news. Three common techniques were used in the experiments: Naïve Bayes, Neural Network, and Support Vector Machine (SVM). The normalization method is a major step to disinfect

data before using the automatic learning method to sort information. The results show Naïve Bayes to have a 96.08% accuracy in detecting counterfeit news. There are two other advanced methods, the Neural Network Machine and the Support Network (SVM), which achieve 99.90% accuracy. In [13] by Kuruvilla et al., a neural network was successfully trained by analyzing the 4000 fake and 4000 real images error levels. The trained neural network has succeeded in identifying the image as fake or real, with a high success rate of 83%. The results showed that using this application on mobile platforms significantly reduces the spread of fake images across social networks. In addition, this can be used as a false image verification method in digital authentication, court evidence assessment, etc. This research develops an approach that takes an image as input and classifies it, using the CNN model. For a completely new task/problem, CNNs are very good feature extractors. It extracts useful attributes from an already trained CNN with its trained weights by feeding your data at each level and tuning the CNN a bit for the specific task. This means that a CNN can be retrained for new recognition tasks, enabling it to build on pre-existing networks. This is called pre-training, where one can avoid training a CNN from the beginning and save time. CNN can carry out automatic feature extraction for the given task. It eliminates the need for manual feature extraction since the features are learned directly by the CNN. In terms of performance, CNNs outperform many methods for image recognition tasks and many other tasks where it gives high accuracy and accurate result. Another key feature of CNNs is weight sharing, which basically means that the same weight is used for two layers in the model. Due to the above features and advantages, CNN is used in this research in comparison to other deep learning algorithms.

CHAPTER – 3

SYSTEM ANALYSIS

3.1Existing System:

When capturing an image, additional required hidden information is associated with it for authentication and forgery protection purposes. The passive technique does not rely on extra information, but it analyzes some features extracted from the digital content of the image itself. Copy-move means coping a part of an image and pasting it into another place of the same picture whereas splicing is about taking a part of an image and pasting it into another.

3.2 Disadvantages of Existing System:

- Complexity in analyzing the data.
- Prediction is a challenging task working in the model
- Coding is complex maintaining multiple methods.
- Library's support was not that much familiar.

3.3 Proposed system:

In this project, we have a tendency to area unit planning LBP primarily based machine learning Convolution Neural Network known as LBPNET to sight pretend face pictures. Here 1st we'll extract LBP from pictures and so train LBP descriptor pictures with Convolution Neural Network to get coaching model. Whenever we have a tendency to transfer new take a look at the image then that take a look at image are going to be applied on coaching model to sight whether or not take a look at image contains pretend image or non-fake image. Below we will see some details on LBP.

3.3.1 Advantages of the Proposed System:

- Libraries help to analyze the data.
- Statistical and prediction is very easy compared to existing technologies.
- Results will be accurate compared to other methodologies.

3.4 SYSTEM STUDY

3.4.1 Feasibility Study

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very

general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are,

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**

3.4.2 Economical Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

3.4.3 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

3.4.3 SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system

3.5 Software requirement specification

3.5.1 Requirement Analysis

The project involved analyzing the design of few applications so as to make the application more users friendly. To do so, it was really important to keep the navigations from one screen to the other well-ordered and at the same time reducing the amount of typing the user needs to do. In order to make the application more accessible, the browser version had to be chosen so that it is compatible with most of the Browsers.

3.5.2 REQUIREMENT SPECIFICATION

3.5.2.1 Software Requirements

For developing the application, the following are the Software Requirements:

1. Python

Functional Requirements

- Graphical User interface with the User.

3. Operating Systems supported

1. Windows

Technologies and Languages used to Develop

1. Python

3.5.2.1 Hardware Requirements

For developing the application the following are the Hardware Requirements:

- Processor: DUAL CORE
- RAM: 2 GB
- Space on Hard Disk: 5MB

CHAPTER – 4

SYSTEM DESIGN

4.1 Proposed Architecture

The architecture of a machine learning system for fake image detection refers to the overall design and structure of the system. It determines how the various components of the system fit together and interact with one another to achieve the desired functionality.

A typical system architecture for fake image detection might include components for data storage, data preprocessing, feature extraction, model training, model evaluation, model deployment, and user interface.

The data storage component is responsible for storing the dataset of real and fake images that will be used to train and test the model. This may be a local storage system or a cloud-based storage service.

The data preprocessing component is responsible for cleaning and preparing the raw data from the storage system for use by the model. This may include tasks such as resizing the images, removing noise, and normalizing the data.

The feature extraction component is responsible for extracting relevant features from the images that can be used to classify them as real or fake. These features may include patterns, textures, and other visual characteristics that are indicative of image manipulation.

The model training component is responsible for training the machine learning model on the processed data. This may involve using a supervised learning algorithm to adjust the model's parameters so that it learns to accurately predict the labels for new images.

The model evaluation component is responsible for evaluating the performance of the model on a separate test dataset. This may involve calculating metrics such as accuracy, precision, and recall, and comparing the model's performance to that of other fake image detection methods.

The model deployment component is responsible for deploying the model in a production environment, where it can be used to classify new images as real or fake.

The user interface component is responsible for presenting the results of the model to the user in a clear and understandable way. This may include creating plots, charts, and other visualizations to help the user understand the model's performance.

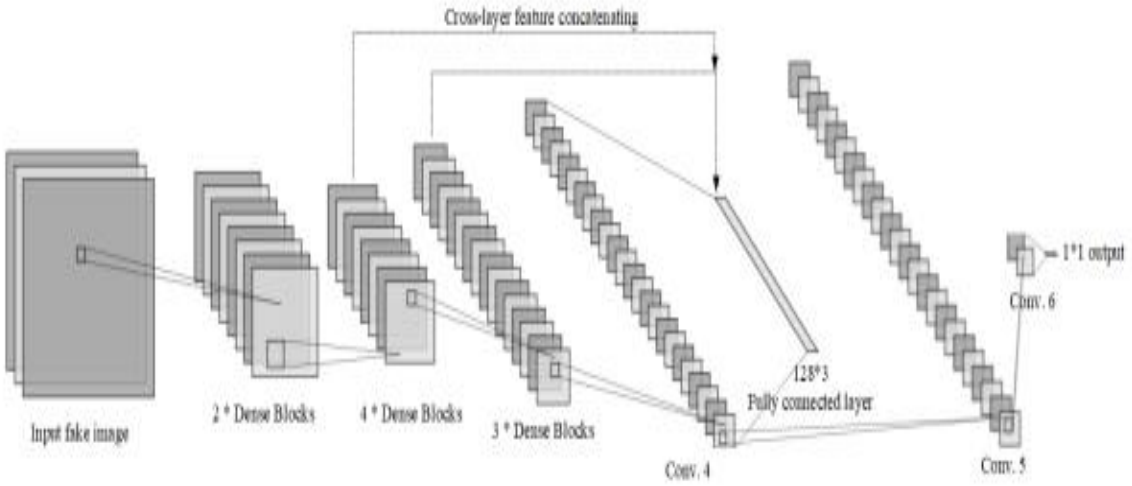


Fig : System Architecture Diagram

4.2 ALGORITHMS

4.2.1 CNN (Convolution Neural Network)

Introduction

When it comes to Machine Learning, Artificial Neural Networks perform really well. Artificial Neural Networks are used in various classification tasks like image, audio, words. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

Before diving into the Convolution Neural Network, let us first revisit some concepts of Neural Network.

Imagine there's an image of a bird, and you want to identify whether it's really a bird or some other object. The first thing you do is feed the pixels of the image in the form of arrays to the input layer of the neural network (multi-layer networks used to classify things). The hidden layers carry out feature extraction by performing different calculations and manipulations. There are multiple hidden layers like the convolution layer, the ReLU layer, and pooling layer, that perform feature extraction from the image. Finally, there's a fully connected layer that identifies the object in the image.

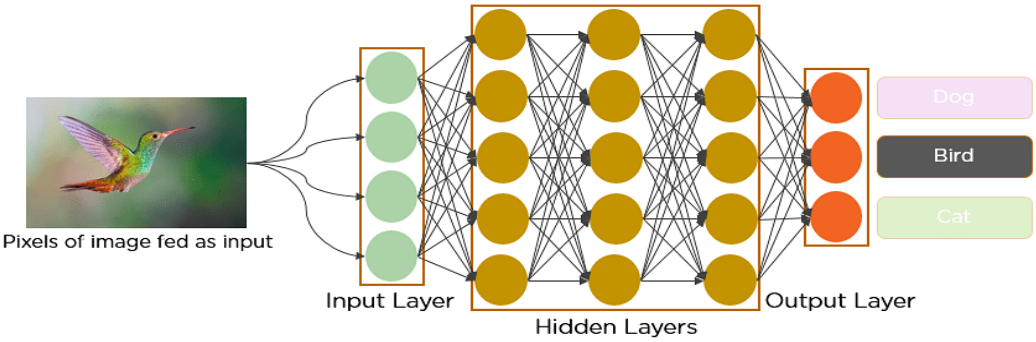
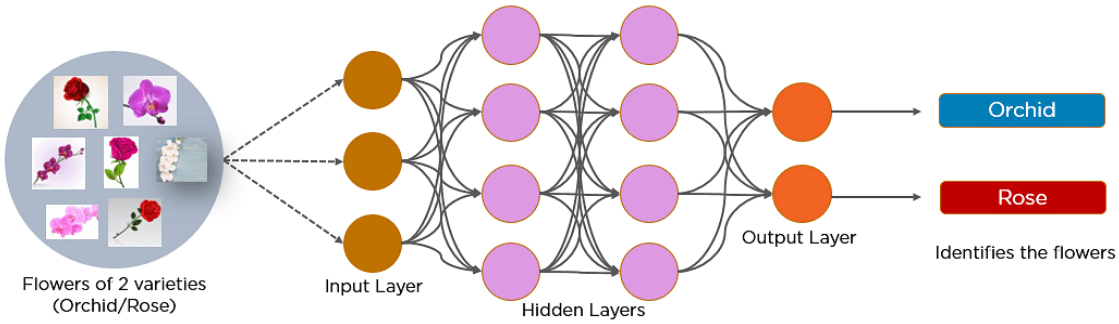


Fig: Convolutional Neural Network to identify the image of a bird

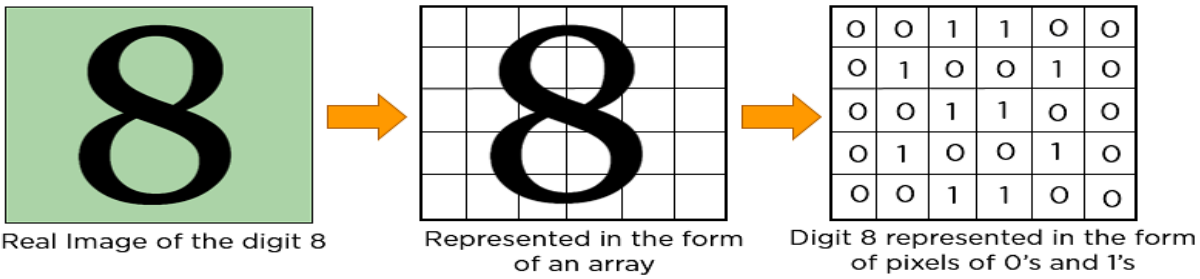
What is Convolutional Neural Network?

A convolutional neural network is a feed-forward neural network that is generally used to analyze visual images by processing data with grid-like topology. It's also known as a ConvNet. A convolutional neural network is used to detect and classify objects in an image.

Below is a neural network that identifies two types of flowers: Orchid and Rose.



In CNN, every image is represented in the form of an array of pixel values.



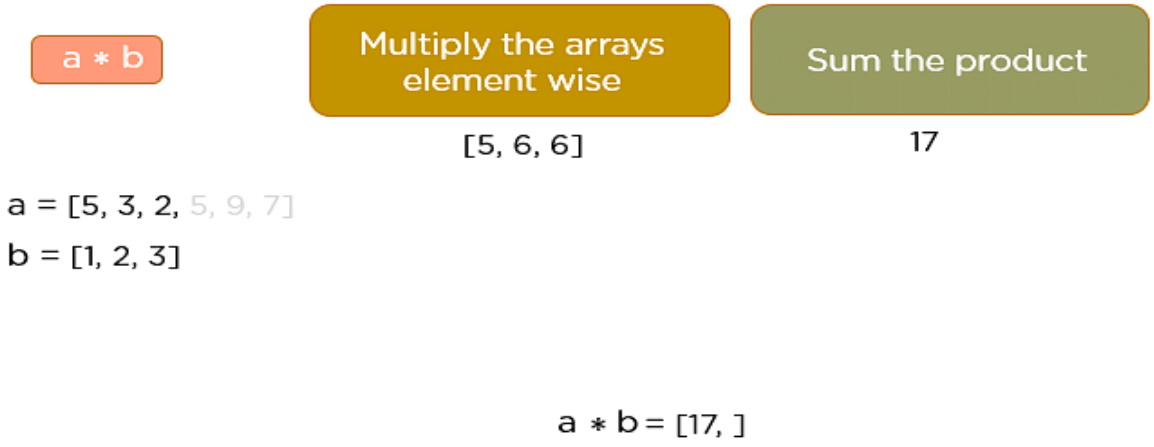
The convolution operation forms the basis of any convolutional neural network. Let's understand the convolution operation using two matrices, a and b, of 1 dimension.

a = [5,3,7,5,9,7]

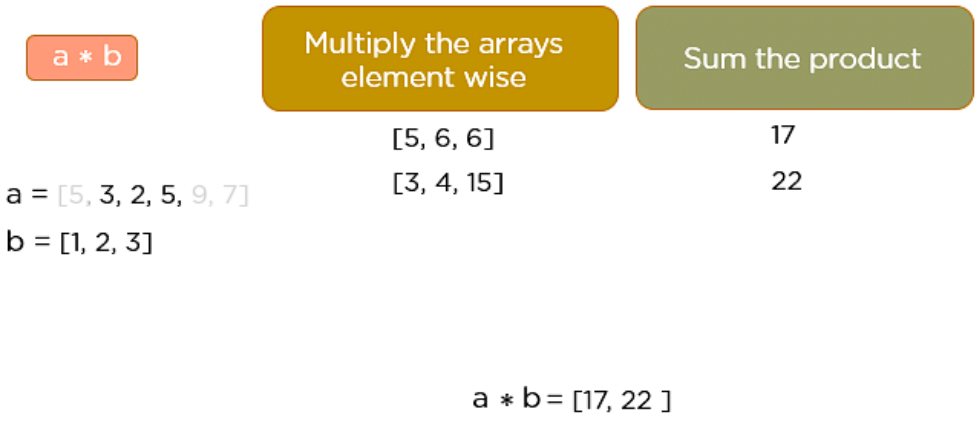
b = [1,2,3]

In convolution operation, the arrays are multiplied element-wise, and the product is summed to create a new array, which represents a*b.

The first three elements of the matrix a are multiplied with the elements of matrix b. The product is summed to get the result.



The next three elements from the matrix a are multiplied by the elements in matrix b, and the product is summed up.



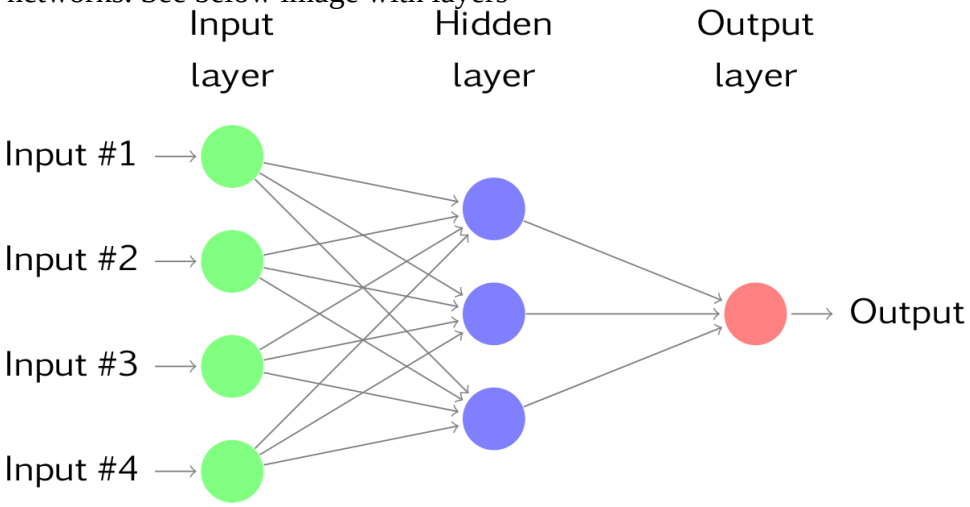
This process continues until the convolution operation is complete.

CNN working procedure

To demonstrate how to build a convolutional neural network based image classifier, we shall build a 6 layer neural network that will identify and separate one image from other. This network that we shall build is a very small network that we can run on a CPU as well. Traditional neural networks that are very good at doing image classification have many more parameters and take a lot of time if trained on normal CPU. However, our objective is to show how to build a real-world convolutional neural network using TENSORFLOW.

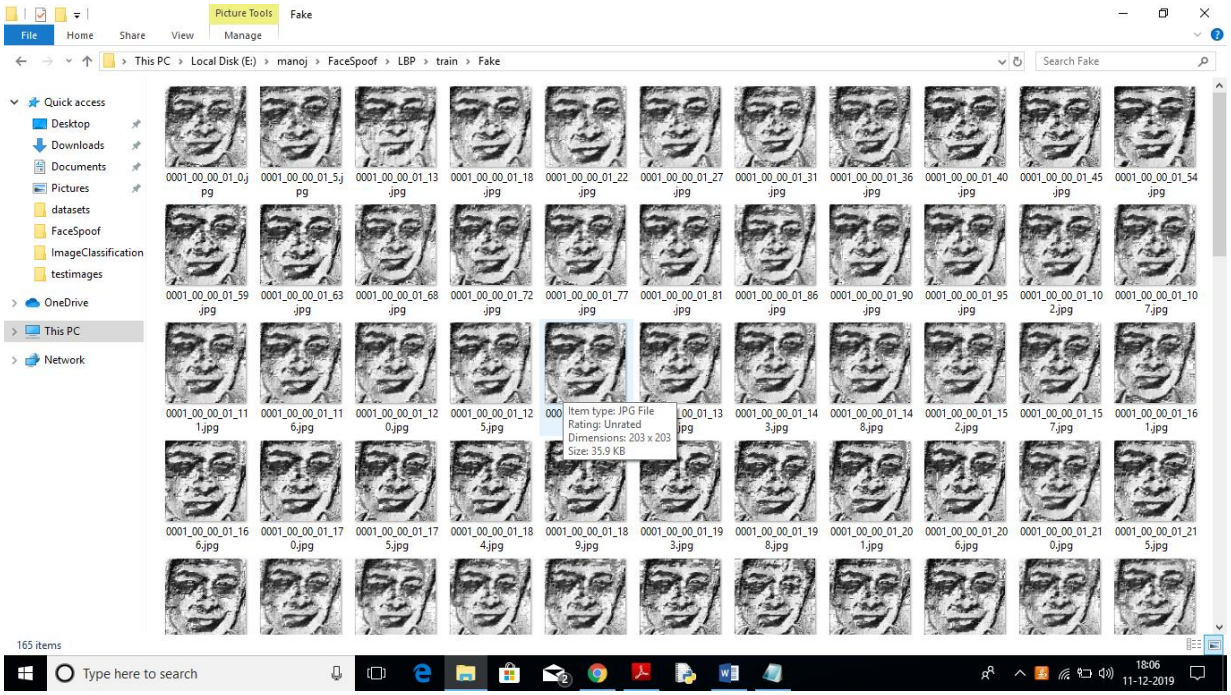
Neural Networks are essentially mathematical models to solve an optimization problem. They are made of neurons, the basic computation unit of neural networks. A neuron takes an input (say x), do some computation on it (say: multiply it with a variable w and adds another variable b) to produce a value (say; z= wx + b). This value is passed to a non-linear function called activation function (f) to produce the final output (activation) of a neuron. There are many kinds of activation functions. One of the popular activation function is Sigmoid. The neuron which uses sigmoid function as an activation function will be called sigmoid neuron. Depending on the activation functions, neurons are named and there are many kinds of them like RELU, TanH.

If you stack neurons in a single line, it's called a layer; which is the next building block of neural networks. See below image with layers



To predict image class multiple layers operate on each other to get best match layer and this process continues till no more improvement left.
Dataset Details:

In this paper author has used NUAA Photograph Imposter (fake) Database with images obtained from real and fake faces. We also used images and convert that image into LBP format. Below are some images from LBP folder



All this fake and real images you can see inside ‘LBP/train’ folder.

This project consists of following modules:

1) Generate NLBPNet Train & Test Model: in this module we will read all LBP images from LBP folder and then train CNN model with all those images.
2) Upload Test Image: In this module we will upload test image from ‘testimages’ folder. Application will read this image and then extract Deep Textures Features from this image using LBP algorithm.
3) Classify Picture In Image: This module apply test image on CNN train model to predict whether test image contains spoof or non-spoof face.

Layers in a Convolutional Neural Network

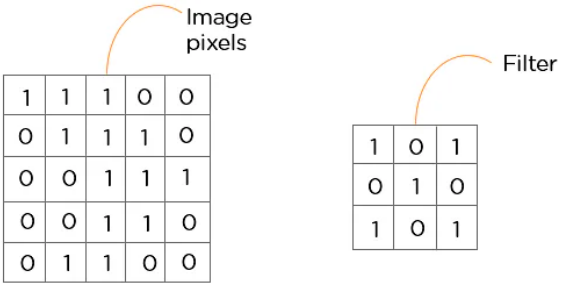
A convolution neural network has multiple hidden layers that help in extracting information from an image. The four important layers in CNN are:

1) Convolution layer
2) ReLU layer
3) Pooling layer
4) Fully connected layer

Convolution Layer

This is the first step in the process of extracting valuable features from an image. A convolution layer has several filters that perform the convolution operation. Every image is considered as a matrix of pixel values.

Consider the following 5x5 image whose pixel values are either 0 or 1. There’s also a filter matrix with a dimension of 3x3. Slide the filter matrix over the image and compute the dot product to get the convolved feature matrix.



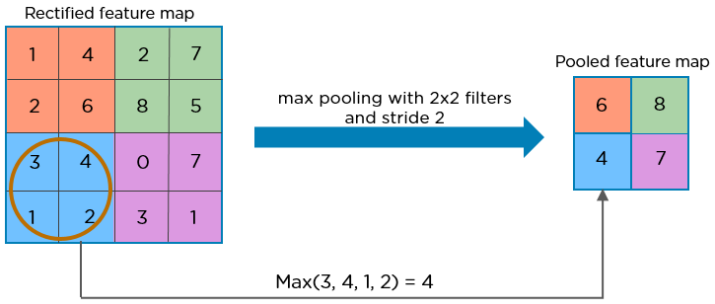
ReLU layer

ReLU stands for the rectified linear unit. Once the feature maps are extracted, the next step is to move them to a ReLU layer.

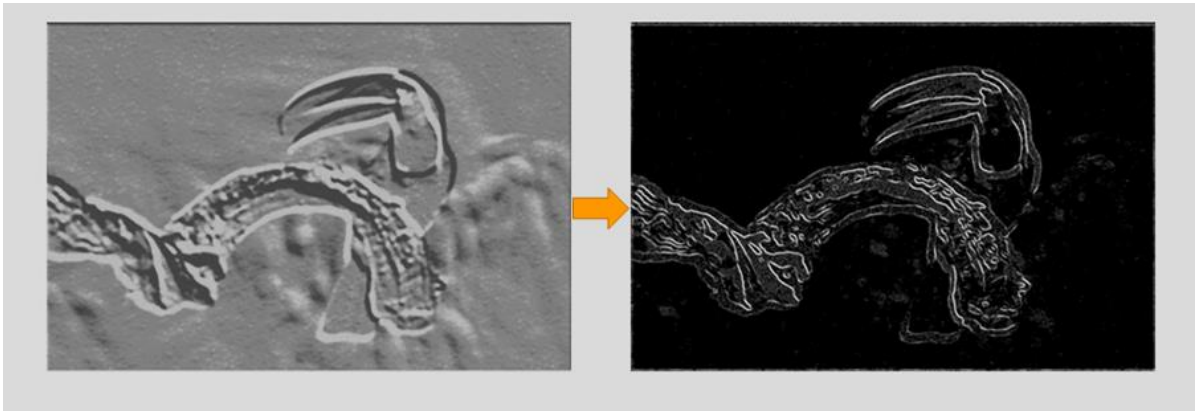
ReLU performs an element-wise operation and sets all the negative pixels to 0. It introduces non-linearity to the network, and the generated output is a rectified feature map. Below is the graph of a ReLU function:

Pooling Layer

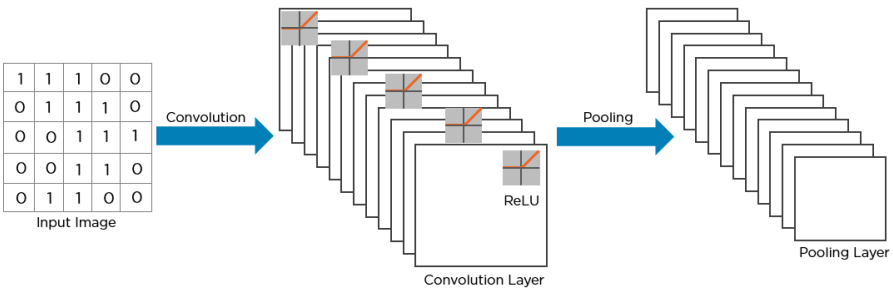
Pooling is a down-sampling operation that reduces the dimensionality of the feature map. The rectified feature map now goes through a pooling layer to generate a pooled feature map.



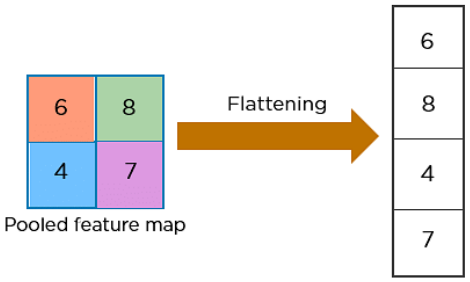
The pooling layer uses various filters to identify different parts of the image like edges, corners, body, feathers, eyes, and beak.



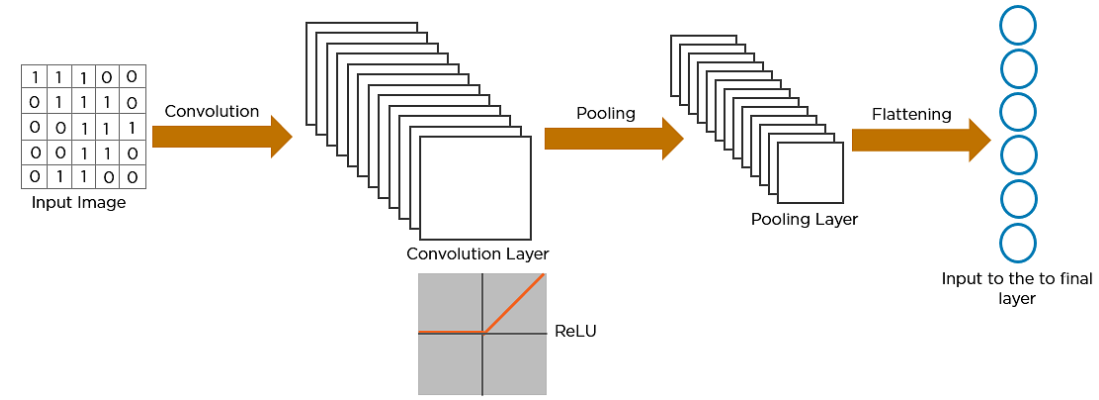
Here's how the structure of the convolution neural network looks so far:

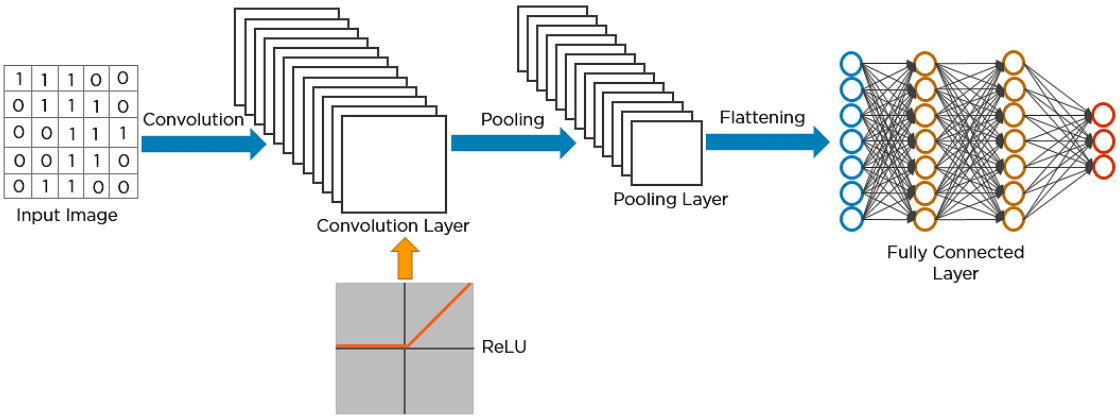


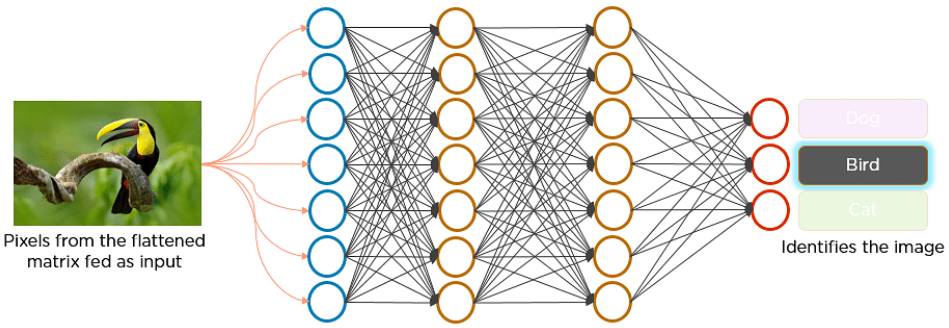
The next step in the process is called flattening. Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.



The flattened matrix is fed as input to the fully connected layer to classify the image.

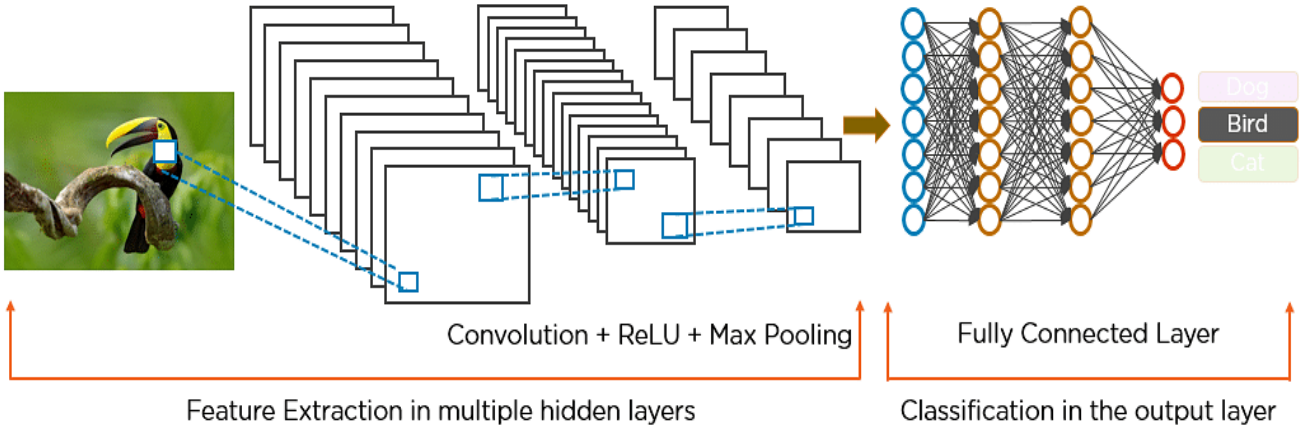






Here's how exactly CNN recognizes a bird:

- The pixels from the image are fed to the convolutional layer that performs the convolution operation
- It results in a convolved map
- The convolved map is applied to a ReLU function to generate a rectified feature map
- The image is processed with multiple convolutions and ReLU layers for locating the features
- Different pooling layers with various filters are used to identify specific parts of the image
- The pooled feature map is flattened and fed to a fully connected layer to get the final output



4.2.2 LBP (Local binary patterns) ALGORITHM

Local binary patterns (LBP) is a type of visual descriptor used for classification in computer vision. LBP is the particular case of the Texture Spectrum model proposed in 1990. LBP was first described in 1994. It has since been found to be a powerful feature for texture classification; it has further been determined that when LBP is combined with the Histogram of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets. A comparison of several improvements of the original LBP in the field of background subtraction was made in 2015 by Silva et al.A full survey of the different versions of LBP can be found in Bouwmans et al.

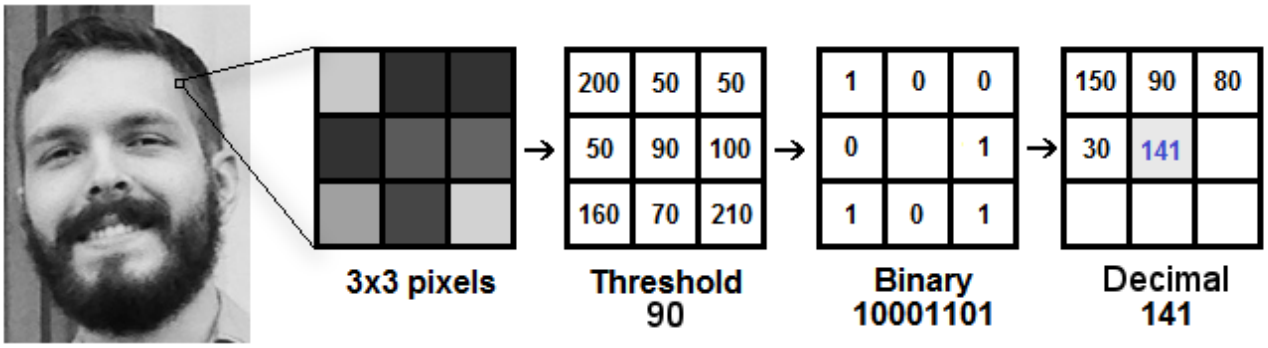
Concept

The LBP feature vector, in its simplest form, is created in the following manner:

- Divide the examined window into cells (e.g. 16x16 pixels for each cell).
- For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
- Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).
- Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.
- Optionally normalize the histogram.
- Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.

The feature vector can now be processed using the Support vector machine, extreme learning machines, or some other machine learning algorithm to classify images. Such classifiers can be used for face recognition or texture analysis.

A useful extension to the original operator is the so-called uniform pattern,[8] which can be used to reduce the length of the feature vector and implement a simple rotation invariant descriptor. This idea is motivated by the fact that some binary patterns occur more commonly in texture images than others. A local binary pattern is called uniform if the binary pattern contains at most two 0-1 or 1-0 transitions. For example, 00010000 (2 transitions) is a uniform pattern, but 01010100 (6 transitions) is not. In the computation of the LBP histogram, the histogram has a separate bin for every uniform pattern, and all non-uniform patterns are assigned to a single bin. Using uniform patterns, the length of the feature vector for a single cell reduces from 256 to 59. The 58 uniform binary patterns correspond to the integers 0, 1, 2, 3, 4, 6, 7, 8, 12, 14, 15, 16, 24, 28, 30, 31, 32, 48, 56, 60, 62, 63, 64, 96, 112, 120, 124, 126, 127, 128, 129, 131, 135, 143, 159, 191, 192, 193, 195, 199, 207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247, 248, 249, 251, 252, 253, 254 and 255.



4.3 UML Diagrams

Unified Modeling Language (UML) is a standardized visual language for representing the design and structure of software systems. UML diagrams are graphical representations that use symbols and notation to show the components and relationships within a system.

There are several different types of UML diagrams that can be used to model a software system, including class diagrams, sequence diagrams, and state diagrams.

UML diagrams are commonly used in software development to communicate the design of a system to stakeholders, and to help designers and developers understand the relationships between the different components of the system. They can be a useful tool for visualizing and documenting the architecture of a software system.

4.3.1 Use case Diagram

A use case diagram is a type of UML diagram that is used to represent the interactions between a system and the external entities (called "actors") that use it. It shows the relationships between the actors and the use cases (i.e., the specific actions or functions that the system can perform) in the system.

A use case diagram typically consists of a number of shapes, including:

1) Actors: These are represented as stick figures and represent the external entities that interact with the system.

2) Use cases: These are represented as ovals and represent the specific actions or functions that the system can perform.

3) Association lines: These are solid lines that connect the actors to the use cases, and represent the interactions between them.

4) Generalization lines: These are dashed lines that show inheritance or specialization relationships between actors or use cases.

Use case diagrams are commonly used in software development to represent the functional requirements of a system and to help design the interactions between the system and its users. They can be a useful tool for visualizing and documenting the different scenarios in which the system will be used.

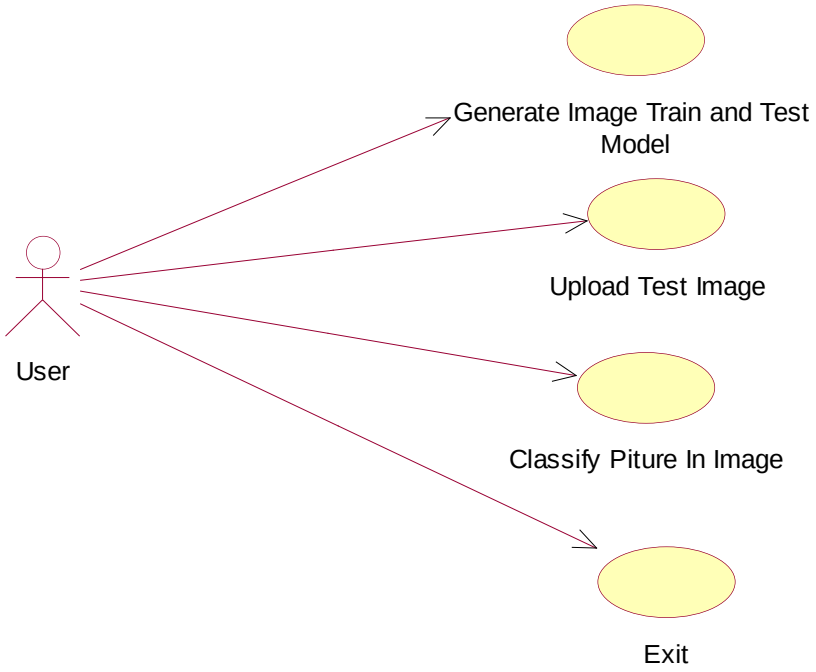


Fig: use case diagram

4.3.2 Class Diagram

A class diagram is a type of UML diagram that shows the classes, attributes, and relationships between them in a system. Classes are represented as boxes with three sections: the top section shows the name of the class, the middle section shows the attributes of the class, and the bottom section shows the operations (i.e., methods) of the class.

Classes can be related to one another in a number of ways, which are represented using lines and arrows in the diagram. Some common types of relationships include:

Inheritance: This relationship represents a "is-a" relationship between classes. It is shown using a solid line with a closed, arrowhead pointing from the subclass to the superclass.

Association: This relationship represents a structural relationship between classes. It is shown using a solid line with an open arrowhead pointing from the associated class to the class that it is associated with.

Aggregation: This relationship represents a "has-a" relationship between classes. It is shown using a solid line with a diamond shape at the end pointing to the class that is aggregated.

Composition: This relationship represents a strong "has-a" relationship between classes, in which the lifetime of the composed class is dependent on the class that it is composed with. It is shown using a

solid line with a filled diamond shape at the end pointing to the class that is composed.

Class diagrams can be a useful tool for visualizing and documenting the structure and relationships within a system, and are commonly used in software development to design and model the components of a system.

In the context of fake image identification using machine learning, a class diagram might be used to represent the various components of the system, including the data storage, data preprocessing, feature extraction, model training, and model evaluation modules.

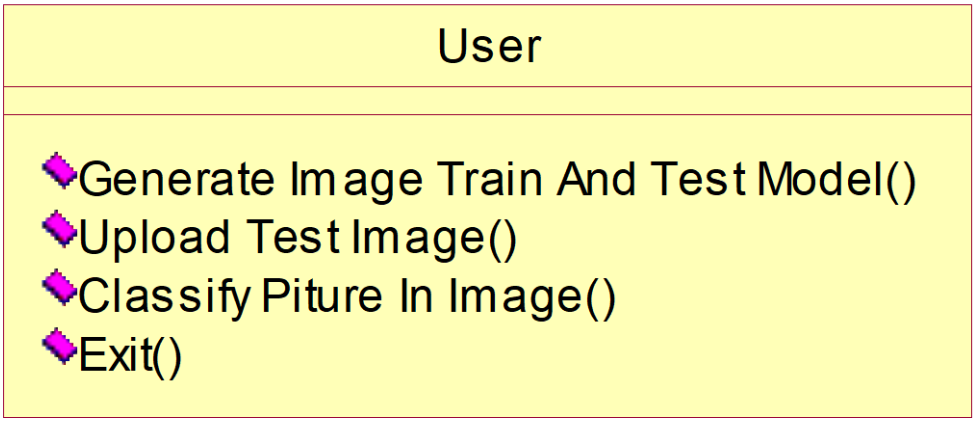


Fig : Class Diagram for fake image identification using ML

4.3.3 Sequence Diagram

A sequence diagram is a type of diagram that shows the interactions between objects or components in a system. In the context of fake image detection using machine learning, a sequence diagram could be used to represent the steps involved in the process of identifying fake images.

For example, the sequence diagram might show the flow of data from the input image to the machine learning model, as well as the steps involved in preprocessing the image and analyzing it for signs of tampering. The diagram could also show the interaction between the machine learning model and other components of the system, such as a database of known fake images or a user interface for displaying the results of the analysis.

Overall, a sequence diagram can be a useful tool for visualizing and understanding the different steps involved in detecting fake images using machine learning, and can help to identify potential points of failure or areas for optimization in the process.

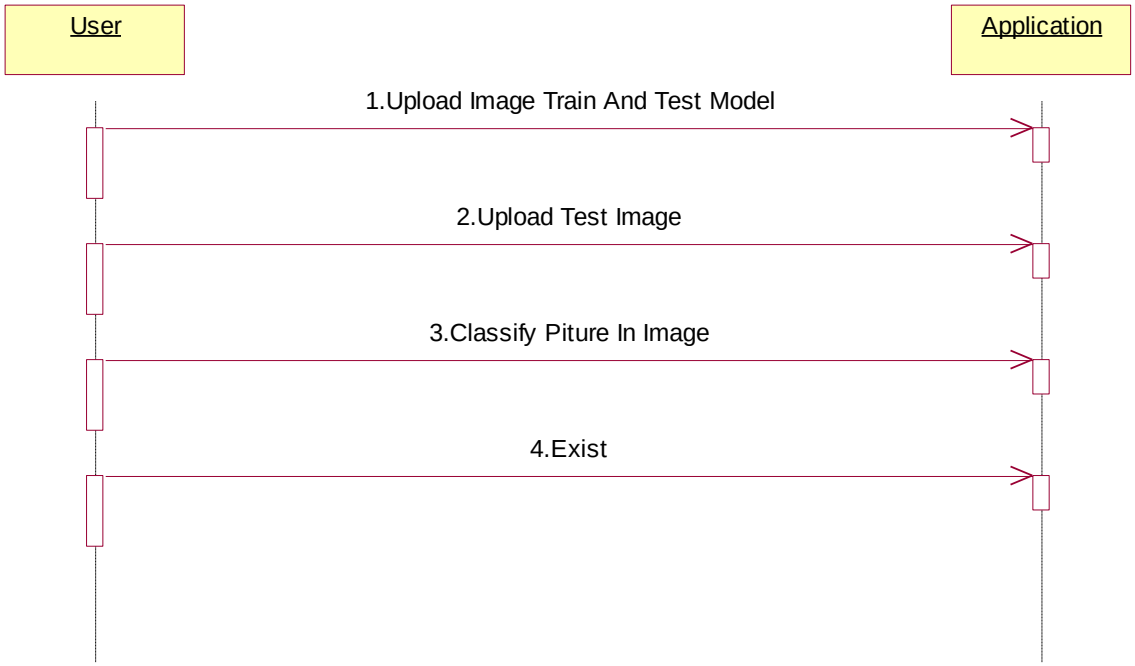


Fig : Sequence Diagram

4.3.4 Collaboration Diagram

A collaboration diagram, also known as a communication diagram or an interaction diagram, is a type of diagram that shows the interactions between objects or components in a system. In the context of fake image detection using machine learning, a collaboration diagram could be used to represent the different components of the system and how they interact with each other in order to detect fake images.

For example, the collaboration diagram might show the machine learning model as a central component, with input data flowing into the model and results flowing out of it. The diagram could also show the interaction between the machine learning model and other components of the system, such as a database of known fake images or a user interface for displaying the results of the analysis.

Overall, a collaboration diagram can be a useful tool for visualizing and understanding the different components of a fake image detection system and how they work together, and can help to identify potential points of failure or areas for optimization in the process.

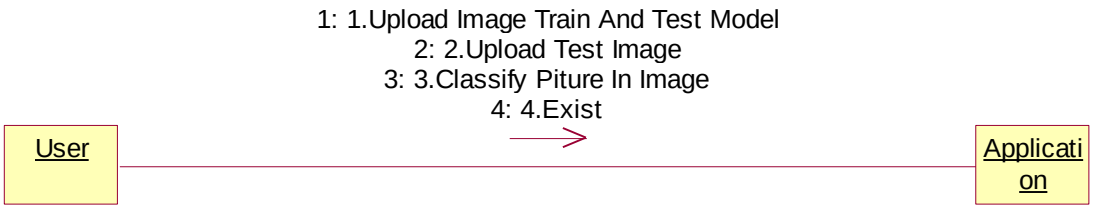


Fig: Collaboration diagram

4.3.5 Flow chart

A sequence diagram is a type of diagram that shows the interactions between objects or components in a system over time. In the context of fake image detection, a sequence diagram could be used to represent the steps involved in the process of analyzing an image to determine whether it is fake. The diagram could show the flow of data from the input image to the machine learning model, as well as the steps involved in preprocessing the image and analyzing it for signs of tampering. The diagram could also show the interaction between the machine learning model and other components of the system, such as a database of known fake images or a user interface for displaying the results of the analysis.

A collaboration diagram, also known as a communication diagram or an interaction diagram, is a type of diagram that shows the interactions between objects or components in a system. In the context of fake image detection, a collaboration diagram could be used to represent the different components of the system and how they interact with each other in order to detect fake images. The diagram could show the machine learning model as a central component, with input data flowing into the model and results flowing out of it. The diagram could also show the interaction between the machine learning model and other components of the system, such as a database of known fake images or a user interface for displaying the results of the analysis.

A flowchart is a type of diagram that represents a series of steps or processes in a clear and logical way. In the context of fake image detection, a flowchart could be used to represent the steps involved in the process of analyzing an image to determine whether it is fake. The flowchart might start with the input of an image, and then show the steps involved in preprocessing the image and analyzing it for signs of tampering. The flowchart could also include steps for verifying the results of the analysis, such as comparing the image to a database of known fake images or seeking confirmation from a human expert.

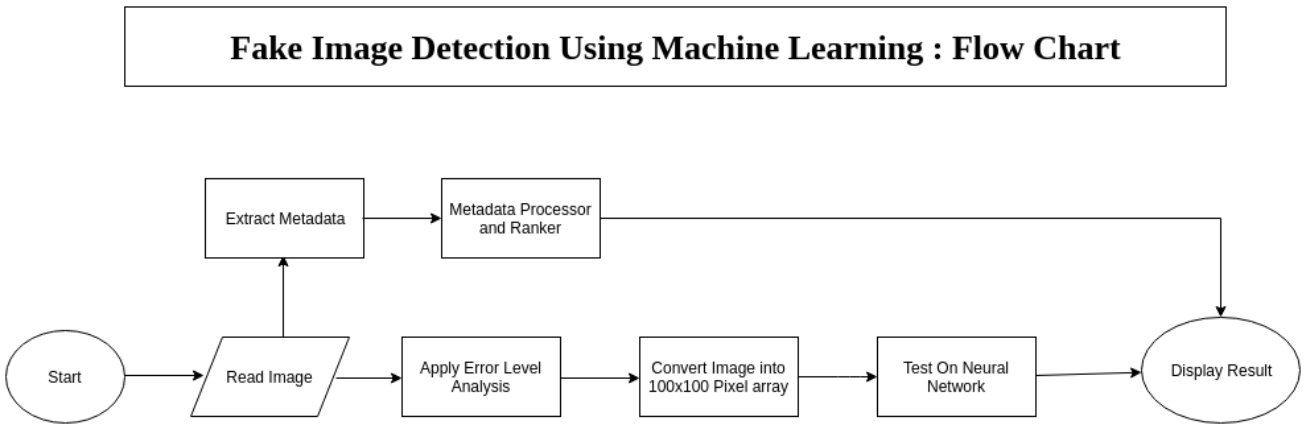


Fig: Flow chart

4.4 INPUT AND OUTPUT DESIGN

Input Design

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

Objective

1.Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

3.When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

Output Design

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements

2.Select methods for presenting information.

3.Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the
- Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

CHAPTER – 5

IMPLEMENTATION

5.1 Technology

5.1.1 Python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++or Java. It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open-source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python

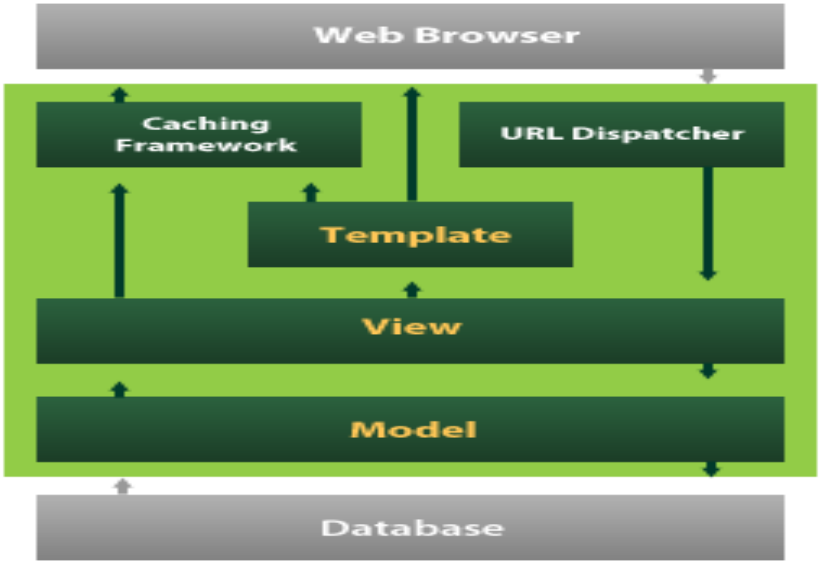
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

- Python can be treated in a procedural way, an object-orientated way or a functional way.

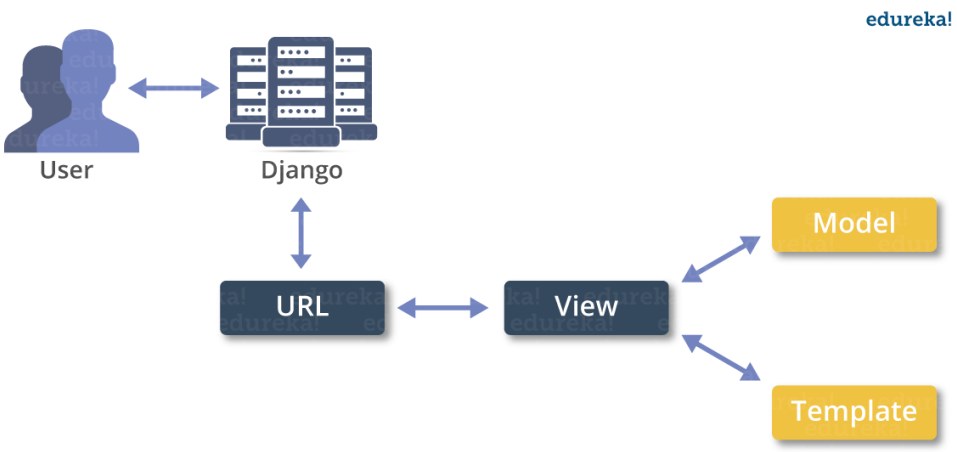
5.1.2 DJANGO

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models.



- Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models



5.1.3 Virtual Environments and Packages

Introduction

Python applications will often use packages and modules that don't come as part of the standard library. Applications will sometimes need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library's interface.

This means it may not be possible for one Python installation to meet the requirements of every application. If application A needs version 1.0 of a particular module but application B needs version 2.0, then the requirements are in conflict and installing either version 1.0 or 2.0 will leave one application unable to run.

The solution for this problem is to create a virtual environment, a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.

Different applications can then use different virtual environments. To resolve the earlier example of conflicting requirements, application A can have its own virtual environment with version 1.0 installed while application B has another virtual environment with version 2.0. If application B requires a library be upgraded to version 3.0, this will not affect application A's environment.

Creating Virtual Environments

The module used to create and manage virtual environments is called venv. venv will usually install the most recent version of Python that you have available. If you have multiple versions of Python on your system, you can select a specific Python version by running python3 or whichever version you want.

To create a virtual environment, decide upon a directory where you want to place it, and run the venv module as a script with the directory path:

python3 -m venv tutorial-env

This will create the tutorial-env directory if it doesn't exist, and also create directories inside it containing a copy of the Python interpreter, the standard library, and various supporting files.

A common directory location for a virtual environment is .venv. This name keeps the directory typically hidden in your shell and thus out of the way while giving it a name that explains why the directory exists. It also prevents clashing with .env environment variable definition files that some tooling supports.

Once you've created a virtual environment, you may activate it.

On Windows, run:

tutorial-env\Scripts\activate.bat

On Unix or MacOS, run:

source tutorial-env/bin/activate

(This script is written for the bash shell. If you use the csh or fish shells, there are alternate activate.csh and activate.fish scripts you should use instead.)

Activating the virtual environment will change your shell's prompt to show what virtual environment you're using, and modify the environment so that running python will get you that particular version and installation of Python. For example:

$ source ~/envs/tutorial-env/bin/activate

(tutorial-env) $ python

Python 3.5.1 (default, May 6 2016, 10:59:36)

...

>>> import sys

>>>sys.path

['', '/usr/local/lib/python35.zip', ...,

'~/envs/tutorial-env/lib/python3.5/site-packages']

>>>

12.3. Managing Packages with pip

You can install, upgrade, and remove packages using a program called pip. By default pip will install packages from the Python Package Index, <https://pypi.org>. You can browse the Python Package Index by going to it in your web browser, or you can use pip's limited search feature:

(tutorial-env) $ pip search astronomy

skyfield - Elegant astronomy for Python

gary - Galactic astronomy and gravitational dynamics.

novas - The United States Naval Observatory NOVAS astronomy library

astroobs - Provides astronomy ephemeris to plan telescope observations

PyAstronomy - A collection of astronomy related tools for Python.

...

pip has a number of subcommands: "search", "install", "uninstall", "freeze", etc. (Consult the Installing Python Modules guide for complete documentation for pip.)

You can install the latest version of a package by specifying a package's name:

(tutorial-env) $ pip install novas

Collecting novas

Downloading novas-3.1.1.3.tar.gz (136kB)

Installing collected packages: novas

Running setup.py install for novas

Successfully installed novas-3.1.1.3

You can also install a specific version of a package by giving the package name followed by == and the version number:

(tutorial-env) $ pip install requests==2.6.0

Collecting requests==2.6.0

Using cached requests-2.6.0-py2.py3-none-any.whl

Installing collected packages: requests

Successfully installed requests-2.6.0

If you re-run this command, pip will notice that the requested version is already installed and do nothing. You can supply a different version number to get that version, or you can run pip install --upgrade to upgrade the package to the latest version:

(tutorial-env) $ pip install --upgrade requests

Collecting requests

Installing collected packages: requests

Found existing installation: requests 2.6.0

Uninstalling requests-2.6.0:

Successfully uninstalled requests-2.6.0

Successfully installed requests-2.7.0

pip uninstall followed by one or more package names will remove the packages from the virtual environment.

pip show will display information about a particular package:

(tutorial-env) $ pip show requests

Metadata-Version: 2.0

Name: requests

Version: 2.7.0

Summary: Python HTTP for Humans.

Home-page: http://python-requests.org

Author: Kenneth Reitz

Author-email: me@kennethreitz.com

License: Apache 2.0

Location: /Users/akuchling/envs/tutorial-env/lib/python3.4/site-packages

Requires:

pip list will display all of the packages installed in the virtual environment:

(tutorial-env) $ pip list

novas (3.1.1.3)

numpy (1.9.2)

pip (7.0.3)

requests (2.7.0)

setuptools (16.0)

pip freeze will produce a similar list of the installed packages, but the output uses the format that pip install expects. A common convention is to put this list in a requirements.txt file:

(tutorial-env) $ pip freeze > requirements.txt

(tutorial-env) $ cat requirements.txt

novas==3.1.1.3

numpy==1.9.2

requests==2.7.0

The requirements.txt can then be committed to version control and shipped as part of an application. Users can then install all the necessary packages with install -r:

```
(tutorial-env) $ pip install -r requirements.txt
Collecting novas==3.1.1.3 (from -r requirements.txt (line 1))
  ...
Collecting numpy==1.9.2 (from -r requirements.txt (line 2))
  ...
Collecting requests==2.7.0 (from -r requirements.txt (line 3))
  ...
Installing collected packages: novas, numpy, requests
  Running setup.py install for novas
Successfully installed novas-3.1.1.3 numpy-1.9.2 requests-2.7.0
```

pip has many more options. Consult the Installing Python Modules guide for complete documentation for pip. When you've written a package and want to make it available on the Python Package Index, consult the Distributing Python Modules guide.

Introduction to Artificial Intelligence

"The science and engineering of making intelligent machines, especially intelligent computer programs". -John McCarthy-

Artificial Intelligence is an approach to make a computer, a robot, or a product to think how smart human think. AI is a study of how human brain think, learn, decide and work, when it tries to solve problems. And finally this study outputs intelligent software systems.The aim of AI is to improve computer functions which are related to human knowledge, for example, reasoning, learning, and problem-solving.

The intelligence is intangible. It is composed of

- Reasoning
- Learning
- Problem Solving
- Perception

- Linguistic Intelligence

The objectives of AI research are reasoning, knowledge representation, planning, learning, natural language processing, realization, and ability to move and manipulate objects. There are long-term goals in the general intelligence sector.

Approaches include statistical methods, computational intelligence, and traditional coding AI. During the AI research related to search and mathematical optimization, artificial neural networks and methods based on statistics, probability, and economics, we use many tools. Computer science attracts AI in the field of science, mathematics, psychology, linguistics, philosophy and so on.

Applications of AI

· Gaming − AI plays important role for machine to think of large number of possible positions based on deep knowledge in strategic games. for example, chess,river crossing, N-queens problems and etc.
Natural Language Processing − Interact with the computer that understands natural language spoken by humans.
· Expert Systems − Machine or software provide explanation and advice to the users.
· Vision Systems − Systems understand, explain, and describe visual input on the computer.
· Speech Recognition − There are some AI based speech recognition systems have ability to hear and express as sentences and understand their meanings while a person talks to it. For example Siri and Google assistant.
· Handwriting Recognition − The handwriting recognition software reads the text written on paper and recognize the shapes of the letters and convert it into editable text.
· Intelligent Robots − Robots are able to perform the instructions given by a human.

Machine Learning

Introduction

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people.

Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs.

Any technology user today has benefitted from machine learning. Facial recognition technology allows social media platforms to help users tag and share photos of friends. Optical character recognition (OCR) technology converts images of text into movable type. Recommendation engines, powered by machine learning, suggest what movies or television shows to watch next based on user preferences. Self-driving cars that rely on machine learning to navigate may soon be available to consumers.

Machine learning is a continuously developing field. Because of this, there are some considerations to keep in mind as you work with machine learning methodologies, or analyze the impact of machine learning processes.

In this tutorial, we'll look into the common machine learning methods of supervised and unsupervised learning, and common algorithmic approaches in machine learning, including the k-nearest neighbor algorithm, decision tree learning, and deep learning. We'll explore which programming languages are most used in machine learning, providing you with some of the positive and negative attributes of each. Additionally, we'll discuss biases that are perpetuated by machine learning algorithms, and consider what can be kept in mind to prevent these biases when building algorithms.

Machine Learning Methods

In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed.

Two of the most widely adopted machine learning methods are **supervised learning** which trains algorithms based on example input and output data that is labeled by humans, and **unsupervised learning** which provides the algorithm with no labeled data in order to allow it to find structure within its input data. Let's explore these methods in more detail.

Supervised Learning

In supervised learning, the computer is provided with example inputs that are labeled with their desired outputs. The purpose of this method is for the algorithm to be able to "learn" by comparing its actual output with the "taught" outputs to find errors, and modify the model accordingly. Supervised learning therefore uses patterns to predict label values on additional unlabeled data.

For example, with supervised learning, an algorithm may be fed data with images of sharks labeled as `fish` and images of oceans labeled as `water`. By being trained on this data, the supervised learning algorithm should be able to later identify unlabeled shark images as `fish` and unlabeled ocean images as `water`.

A common use case of supervised learning is to use historical data to predict statistically likely future events. It may use historical stock market information to anticipate upcoming fluctuations, or be

employed to filter out spam emails. In supervised learning, tagged photos of dogs can be used as input data to classify untagged photos of dogs.

Unsupervised Learning

In unsupervised learning, data is unlabeled, so the learning algorithm is left to find commonalities among its input data. As unlabeled data are more abundant than labeled data, machine learning methods that facilitate unsupervised learning are particularly valuable.

The goal of unsupervised learning may be as straightforward as discovering hidden patterns within a dataset, but it may also have a goal of feature learning, which allows the computational machine to automatically discover the representations that are needed to classify raw data.

Unsupervised learning is commonly used for transactional data. You may have a large dataset of customers and their purchases, but as a human you will likely not be able to make sense of what similar attributes can be drawn from customer profiles and their types of purchases. With this data fed into an unsupervised learning algorithm, it may be determined that women of a certain age range who buy unscented soaps are likely to be pregnant, and therefore a marketing campaign related to pregnancy and baby products can be targeted to this audience in order to increase their number of purchases.

Without being told a "correct" answer, unsupervised learning methods can look at complex data that is more expansive and seemingly unrelated in order to organize it in potentially meaningful ways. Unsupervised learning is often used for anomaly detection including for fraudulent credit card purchases, and recommender systems that recommend what products to buy next. In unsupervised learning, untagged photos of dogs can be used as input data for the algorithm to find likenesses and classify dog photos together.

Approaches

As a field, machine learning is closely related to computational statistics, so having a background knowledge in statistics is useful for understanding and leveraging machine learning algorithms.

For those who may not have studied statistics, it can be helpful to first define correlation and regression, as they are commonly used techniques for investigating the relationship among quantitative variables. **Correlation** is a measure of association between two variables that are not designated as either dependent or independent. **Regression** at a basic level is used to examine the relationship between one dependent and one independent variable. Because regression statistics can be used to anticipate the dependent variable when the independent variable is known, regression enables prediction capabilities.

Approaches to machine learning are continuously being developed. For our purposes, we'll go through a few of the popular approaches that are being used in machine learning at the time of writing.

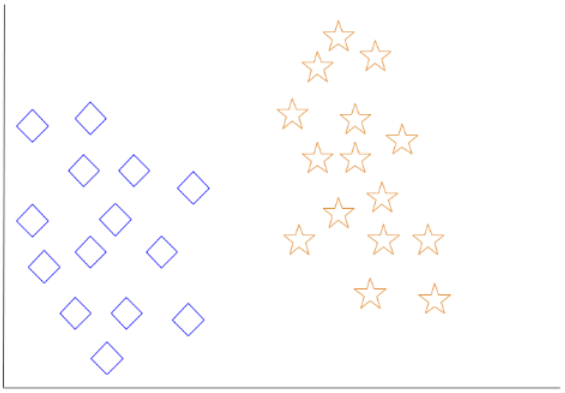
k-nearest neighbor

The k-nearest neighbor algorithm is a pattern recognition model that can be used for classification as well as regression. Often abbreviated as k-NN, the **k** in k-nearest neighbor is a positive integer, which

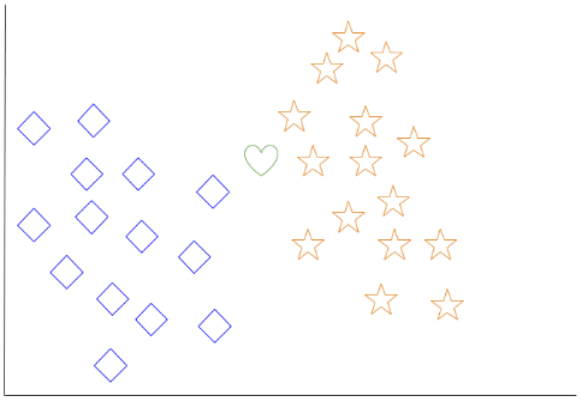
is typically small. In either classification or regression, the input will consist of the k closest training examples within a space.

We will focus on k-NN classification. In this method, the output is class membership. This will assign a new object to the class most common among its k nearest neighbors. In the case of k = 1, the object is assigned to the class of the single nearest neighbor.

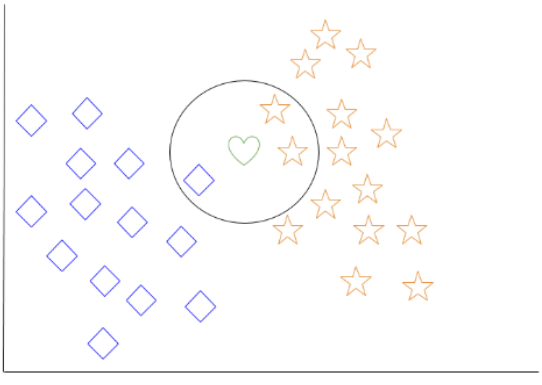
Let's look at an example of k-nearest neighbor. In the diagram below, there are blue diamond objects and orange star objects. These belong to two separate classes: the diamond class and the star class.



When a new object is added to the space — in this case a green heart — we will want the machine learning algorithm to classify the heart to a certain class.



When we choose k = 3, the algorithm will find the three nearest neighbors of the green heart in order to classify it to either the diamond class or the star class.
In our diagram, the three nearest neighbors of the green heart are one diamond and two stars. Therefore, the algorithm will classify the heart with the star class.



Among the most basic of machine learning algorithms, k-nearest neighbor is considered to be a type of "lazy learning" as generalization beyond the training data does not occur until a query is made to the system.

Decision Tree Learning

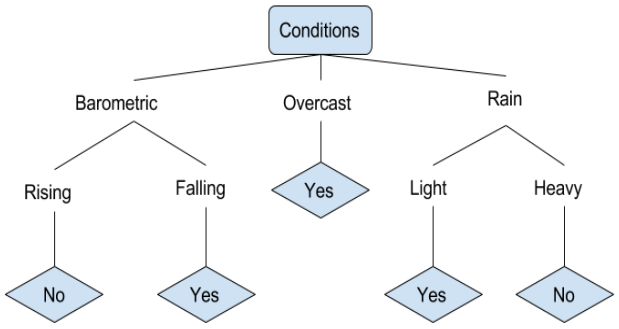
For general use, decision trees are employed to visually represent decisions and show or inform decision making. When working with machine learning and data mining, decision trees are used as a predictive model. These models map observations about data to conclusions about the data's target value.

The goal of decision tree learning is to create a model that will predict the value of a target based on input variables.

In the predictive model, the data's attributes that are determined through observation are represented by the branches, while the conclusions about the data's target value are represented in the leaves.

When "learning" a tree, the source data is divided into subsets based on an attribute value test, which is repeated on each of the derived subsets recursively. Once the subset at a node has the equivalent value as its target value has, the recursion process will be complete.

Let's look at an example of various conditions that can determine whether or not someone should go fishing. This includes weather conditions as well as barometric pressure conditions.



In the simplified decision tree above, an example is classified by sorting it through the tree to the appropriate leaf node. This then returns the classification associated with the particular leaf, which in this case is either a `Yes` or a `No`. The tree classifies a day's conditions based on whether or not it is suitable for going fishing.

A true classification tree data set would have a lot more features than what is outlined above, but relationships should be straightforward to determine. When working with decision tree learning, several determinations need to be made, including what features to choose, what conditions to use for splitting, and understanding when the decision tree has reached a clear ending.

5.2 Modules

The common modules that may be included in a machine learning system for detecting fake images:

Preprocessing: This module is responsible for cleaning and preparing the raw data for use by the model. This may include tasks such as resizing images, removing noise, and normalizing the data.

Feature extraction: This module is responsible for extracting useful features from the images that can be used to identify fake images. These features may include patterns, textures, and other visual characteristics that are indicative of image manipulation.

Classification: This module is responsible for using the extracted features to classify the images as either real or fake. This may be done using a variety of machine learning algorithms, such as support vector machines (SVMs), decision trees, or neural networks.

Evaluation: This module is responsible for evaluating the performance of the model and determining how well it is able to detect fake images. This may include measuring the model's accuracy, precision, recall, and other metrics.

Visualization: This module is responsible for presenting the results of the model to the user in a clear and understandable way. This may include creating plots, charts, and other visualizations to help the user understand the model's performance.

5.3 Sample Code

FakeImageDetect main code:

```
#{'Fake': 0, 'Real': 1}
from tkinter import *
import tkinter
from tkinter import filedialog
import numpy as np
from tkinter.filedialog import askopenfilename
import pandas as pd
from keras.optimizers import Adam
from keras.models import model_from_json
from tkinter import simpledialog
```

```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense,Activation,BatchNormalization
import os
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from tkinter import messagebox
import cv2
from imutils import paths
import imutils
import cv2
import numpy as np



main = tkinter.Tk()
main.title("Fake Image Identification") #designing main screen
main.geometry("600x500")

global filename
global loaded_model

def get_pixel(img, center, x, y):
    new_value = 0
    try:
        if img[x][y] >= center:
            new_value = 1
    except:
        pass
    return new_value

def lbp_calculated_pixel(img, x, y):
    center = img[x][y]
    val_ar = []
```

```
    val_ar.append(get_pixel(img, center, x-1, y+1))     # top_right
    val_ar.append(get_pixel(img, center, x, y+1))       # right
    val_ar.append(get_pixel(img, center, x+1, y+1))     # bottom_right
    val_ar.append(get_pixel(img, center, x+1, y))       # bottom
    val_ar.append(get_pixel(img, center, x+1, y-1))     # bottom_left
    val_ar.append(get_pixel(img, center, x, y-1))       # left
    val_ar.append(get_pixel(img, center, x-1, y-1))     # top_left
    val_ar.append(get_pixel(img, center, x-1, y))       # top

    power_val = [1, 2, 4, 8, 16, 32, 64, 128]
    val = 0
    for i in range(len(val_ar)):
        val += val_ar[i] * power_val[i]
    return val


def upload(): #function to upload tweeter profile
    global filename
    filename = filedialog.askopenfilename(initialdir="testimages")
    messagebox.showinfo("File Information", "image file loaded")


def generateModel():
    global loaded_model
    if os.path.exists('model.json'):
        with open('model.json', "r") as json_file:
            loaded_model_json = json_file.read()
            loaded_model = model_from_json(loaded_model_json)

        loaded_model.load_weights("model_weights.h5")
        loaded_model._make_predict_function()
        print(loaded_model.summary())
        messagebox.showinfo("Model Generated", "CNN Model Generated on Train & Test Data. See black console for details")
    else:
        classifier = Sequential()
```

```
classifier.add(Convolution2D(32, (3, 3), border_mode='valid', input_shape=(48, 48, 1)))
classifier.add(BatchNormalization())
classifier.add(Activation("relu"))
classifier.add(Convolution2D(32, (3, 3),  border_mode='valid'))
classifier.add(BatchNormalization())
classifier.add(Activation("relu"))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Flatten())
classifier.add(Dense(128))
classifier.add(BatchNormalization())
classifier.add(Activation("relu"))
classifier.add(Dense(2))
classifier.add(BatchNormalization())
classifier.add(Activation("softmax"))
# model5 the model
classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
files = []
filename = 'LBP/train/Fake'
label = []
for root, dirs, directory in os.walk(filename):
    for i in range(len(directory)):
        files.append(filename+"/"+directory[i]);
        label.append([1,0])

filename = 'LBP/train/Real'
for root, dirs, directory in os.walk(filename):
    for i in range(len(directory)):
        files.append(filename+"/"+directory[i]);
        label.append([0,1])

print(len(files))
X = np.ndarray(shape=(len(files), 48,48,1), dtype=np.float32)
Y = np.ndarray(shape=(len(files),2),dtype=np.float32)
print(X.shape)
print(Y.shape)
for i in range(len(files)):
```

```
            img = cv2.imread(files[i])
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img = np.resize(img, (48,48,1))
            im2arr = np.array(img)
            im2arr = im2arr.reshape(1,48,48,1)
            X[i] = im2arr
            Y[i] = label[i]
        print("shape == "+str(X.shape))
        #X = X.reshape(X.shape[0],48, 48,3)
        classifier.fit(X, Y,epochs = 10)
        classifier.save_weights('model_weights.h5')
        model_json = classifier.to_json()
        with open("model.json", "w") as json_file:
            json_file.write(model_json)
        print(X.class_indices)
        print(classifier.summary)
        messagebox.showinfo("Model Generated", "NLBPNet Model Generated on Train & Test Data. See black console for details")


def classify():
    name = os.path.basename(filename)
    image_file = filename;
    img_bgr = cv2.imread(image_file)
    height, width, channel = img_bgr.shape
    img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
    img_lbp = np.zeros((height, width,3), np.uint8)
    for i in range(0, height):
        for j in range(0, width):
            img_lbp[i, j] = lbp_calculated_pixel(img_gray, i, j)
    cv2.imwrite('testimages/lbp_'+name, img_lbp)
    img = cv2.imread('testimages/lbp_'+name)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = np.resize(img, (48,48,1))
    im2arr = np.array(img)
    im2arr = im2arr.reshape(1,48,48,1)
```

```
    preds = loaded_model.predict(im2arr)
    print(str(preds)+" "+str(np.argmax(preds)))
    predict = np.argmax(preds)
    msg = ""
    if predict == 0:
        msg = "Image Contains Fake face"
    if predict == 1:
        msg = "Image Contains Real face"
    imagedisplay = cv2.imread(filename)
    orig = imagedisplay.copy()
    output = imutils.resize(orig, width=400)
    cv2.putText(output, msg, (10, 25),  cv2.FONT_HERSHEY_SIMPLEX,0.7, (139, 0, 0), 2)
    cv2.imshow("Predicted Image Result ", output)
    imagedisplay = cv2.imread('testimages/lbp_'+name)
    orig = imagedisplay.copy()
    output = imutils.resize(orig, width=400)
    os.remove('testimages/lbp_'+name)
    cv2.imshow("LBP Image", output)
    cv2.waitKey(0)



def exit():
    global main
    main.destroy()

font = ('times', 16, 'bold')
title = Label(main, text='Fake Image Identification', justify=LEFT)
title.config(bg='lavender blush', fg='DarkOrchid1')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=100,y=5)
title.pack()

font1 = ('times', 14, 'bold')
model = Button(main, text="Generate image Train & Test Model", command=generateModel)
```

```
model.place(x=200,y=100)
model.config(font=font1)

uploadimage = Button(main, text="Upload Test Image", command=upload)
uploadimage.place(x=200,y=150)
uploadimage.config(font=font1)

classifyimage = Button(main, text="Classify Picture In Image", command=classify)
classifyimage.place(x=200,y=200)
classifyimage.config(font=font1)

exitapp = Button(main, text="Exit", command=exit)
exitapp.place(x=200,y=250)
exitapp.config(font=font1)

main.config(bg='light coral')
main.mainloop()
```

LBP code :

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import os

def get_pixel(img, center, x, y):
    new_value = 0
    try:
        if img[x][y] >= center:
            new_value = 1
    except:
        pass
    return new_value

def lbp_calculated_pixel(img, x, y):
    center = img[x][y]
    val_ar = []
```

```
    val_ar.append(get_pixel(img, center, x-1, y+1))     # top_right
    val_ar.append(get_pixel(img, center, x, y+1))       # right
    val_ar.append(get_pixel(img, center, x+1, y+1))     # bottom_right
    val_ar.append(get_pixel(img, center, x+1, y))       # bottom
    val_ar.append(get_pixel(img, center, x+1, y-1))     # bottom_left
    val_ar.append(get_pixel(img, center, x, y-1))       # left
    val_ar.append(get_pixel(img, center, x-1, y-1))     # top_left
    val_ar.append(get_pixel(img, center, x-1, y))       # top

    power_val = [1, 2, 4, 8, 16, 32, 64, 128]
    val = 0
    for i in range(len(val_ar)):
        val += val_ar[i] * power_val[i]
    return val



def main():
    filename = 'data/Fake'
    for root, dirs, files in os.walk(filename):
        for fdata in files:
            image_file = root+"/"+fdata;
            img_bgr = cv2.imread(image_file)
            height, width, channel = img_bgr.shape
            img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
            img_lbp = np.zeros((height, width,3), np.uint8)
            for i in range(0, height):
                for j in range(0, width):
                    img_lbp[i, j] = lbp_calculated_pixel(img_gray, i, j)
            cv2.imwrite('LBP/validation/Fake/'+fdata, img_lbp)
            cv2.imwrite('LBP/train/Fake/'+fdata, img_lbp)

    filename = 'data/Real'
    for root, dirs, files in os.walk(filename):
        for fdata in files:
            image_file = root+"/"+fdata;
```

```
            img_bgr = cv2.imread(image_file)
            height, width, channel = img_bgr.shape
            img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
            img_lbp = np.zeros((height, width,3), np.uint8)
            for i in range(0, height):
                for j in range(0, width):
                    img_lbp[i, j] = lbp_calculated_pixel(img_gray, i, j)
            cv2.imwrite('LBP/validation/Real/'+fdata, img_lbp)
            cv2.imwrite('LBP/train/Real/'+fdata, img_lbp)
    print("LBP Program is finished")

if __name__ == '__main__':
    main()
```

Test code :

```
import numpy as np
import os
import cv2
from sklearn.model_selection import train_test_split
from sklearn_extensions.extreme_learning_machines.elm import GenELMClassifier
from sklearn_extensions.extreme_learning_machines.random_layer import RBFRandomLayer, MLPRandomLayer
from sklearn.metrics import accuracy_score
from numpy.linalg import norm
from numpy import dot
import imutils

files = []
filename = 'data/Fake'
label = []
for root, dirs, directory in os.walk(filename):
    for i in range(len(directory)):
        files.append(filename+"/"+directory[i]);
        label.append(0)

filename = 'data/Real'
```

```
for root, dirs, directory in os.walk(filename):
    for i in range(len(directory)):
        files.append(filename+"/"+directory[i]);
        label.append(1)

print(len(files))
X = np.ndarray(shape=(len(files), 16384), dtype=np.float32)
Y = np.ndarray(shape=(len(files)),dtype=np.float32)
print(X.shape)
print(Y.shape)
for i in range(len(files)):
    img = cv2.imread(files[i])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img,(128,128))
    img = img.reshape(-1)
    X[i] = img
    Y[i] = label[i]

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1, random_state = 0)
print(X_train.shape)
print(y_train.shape)
print(y_train)

def prediction(X_test, cls):
    y_pred = cls.predict(X_test)
    for i in range(len(X_test)):
      print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
    accuracy = accuracy_score(y_test,y_pred)*100
    return accuracy

def get_pixel(img, center, x, y):
    new_value = 0
```

```
    try:
        if img[x][y] >= center:
            new_value = 1
    except:
        pass
    return new_value

def lbp_calculated_pixel(img, x, y):
    center = img[x][y]
    val_ar = []
    val_ar.append(get_pixel(img, center, x-1, y+1))     # top_right
    val_ar.append(get_pixel(img, center, x, y+1))       # right
    val_ar.append(get_pixel(img, center, x+1, y+1))     # bottom_right
    val_ar.append(get_pixel(img, center, x+1, y))       # bottom
    val_ar.append(get_pixel(img, center, x+1, y-1))     # bottom_left
    val_ar.append(get_pixel(img, center, x, y-1))       # left
    val_ar.append(get_pixel(img, center, x-1, y-1))     # top_left
    val_ar.append(get_pixel(img, center, x-1, y))       # top

    power_val = [1, 2, 4, 8, 16, 32, 64, 128]
    val = 0
    for i in range(len(val_ar)):
        val += val_ar[i] * power_val[i]
    return val


srhl_tanh = MLPRandomLayer(n_hidden=500, activation_func='tanh')
cls = GenELMClassifier(hidden_layer=srhl_tanh)
cls.fit(X_train, y_train)
prediction_data = prediction(X_test, cls)
elm_acc = cal_accuracy(y_test, prediction_data)
print(elm_acc)

img_bgr = cv2.imread('testimages/fake.jpg')
height, width, channel = img_bgr.shape
img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
```

```
img_lbp = np.zeros((height, width,3), np.uint8)
for i in range(0, height):
    for j in range(0, width):
        img_lbp[i, j] = lbp_calculated_pixel(img_gray, i, j)
cv2.imwrite('test.jpg', img_lbp)

img1 = cv2.imread('test.jpg')
img1 = cv2.resize(img,(128,128))
img1 = img1.reshape(-1)
y_pred = cls.predict(img1)
print(len(y_pred))
print(y_pred)

img_bgr = cv2.imread('testimages/real.jpg')
height, width, channel = img_bgr.shape
img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
img_lbp = np.zeros((height, width,3), np.uint8)
for i in range(0, height):
    for j in range(0, width):
        img_lbp[i, j] = lbp_calculated_pixel(img_gray, i, j)
cv2.imwrite('test1.jpg', img_lbp)

img1 = cv2.imread('test1.jpg')
img1 = cv2.resize(img,(128,128))
img1 = img1.reshape(-1)
y_pred = cls.predict(img1)
print(len(y_pred))
print(y_pred)
```

CHAPTER - 6

Testing And Validation

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.1 TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

6White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

6.2 Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.

- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

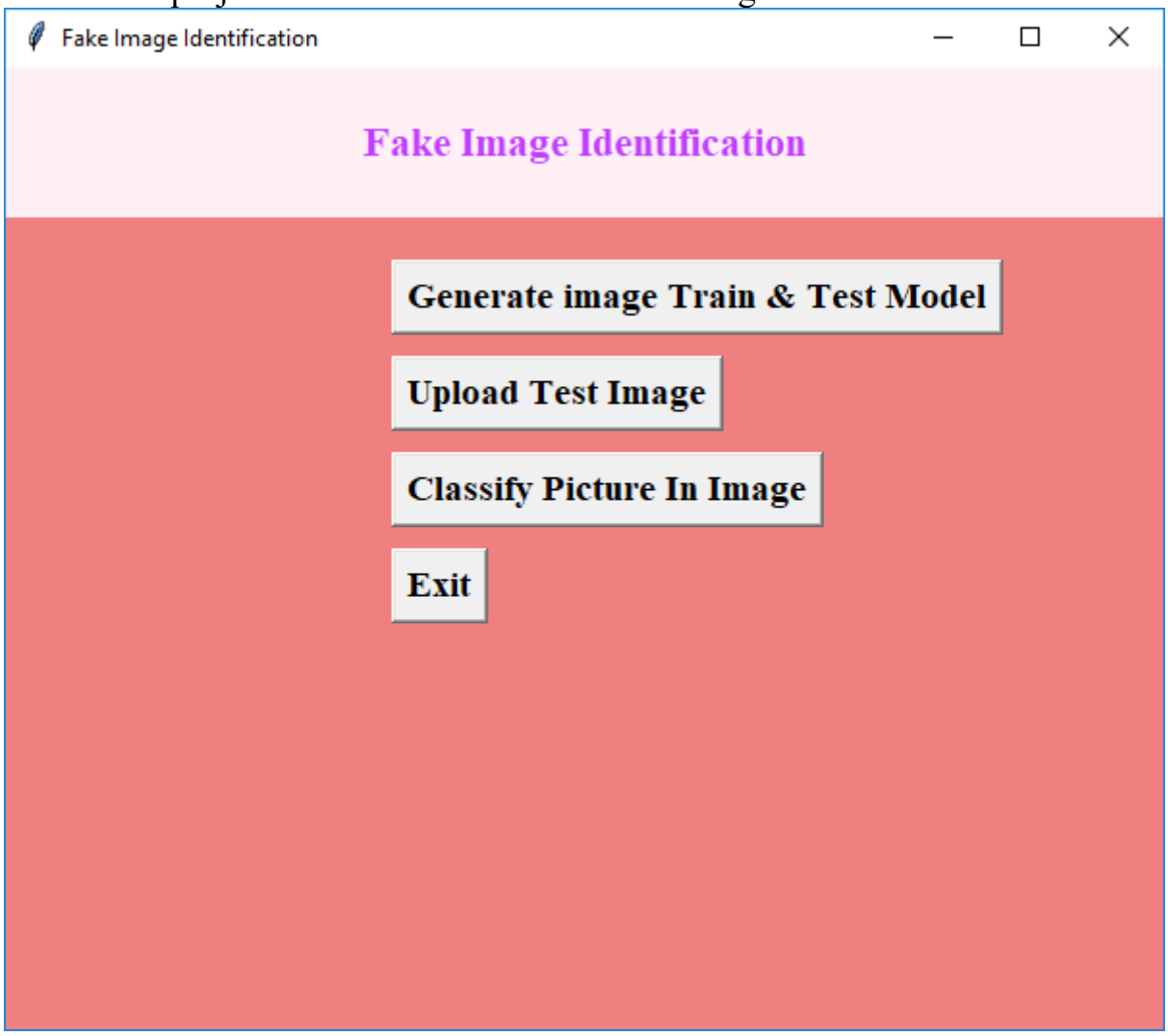
Test Results: All the test cases mentioned above passed successfully. No defects encountered.

CHAPTER – 7

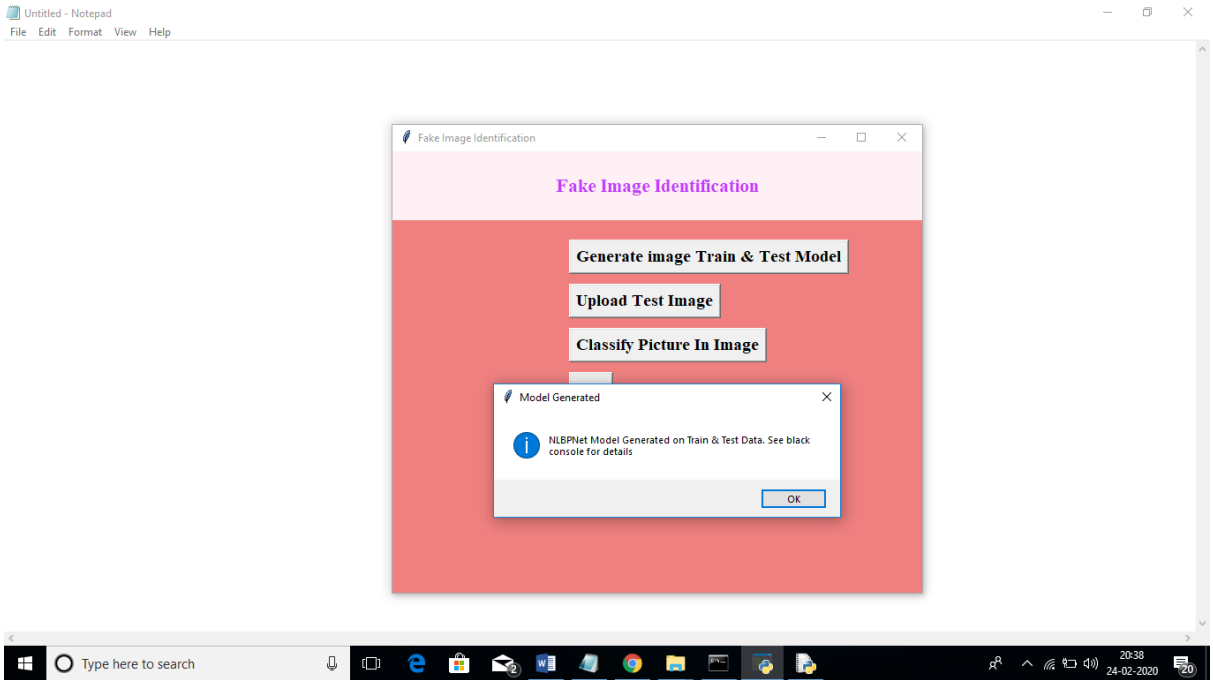
RESULTS AND OUTPUT SCREENS

7.1 Output Screens

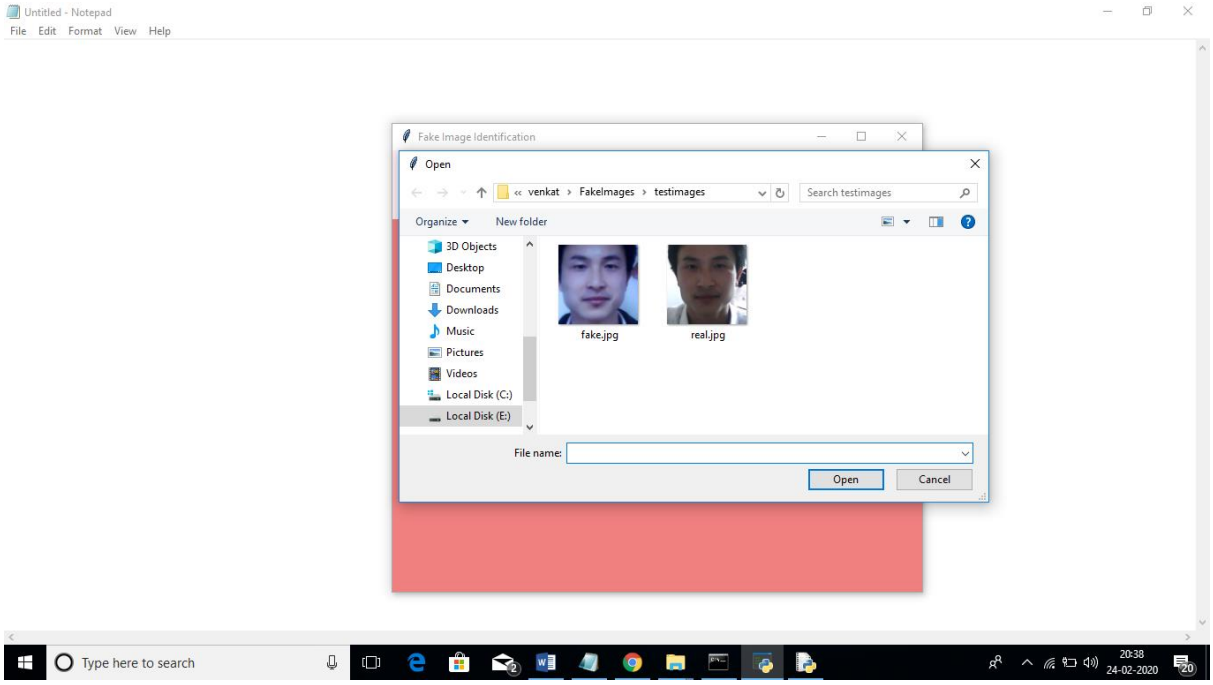
To run this project double click on ‘run.bat’ file to get below screen



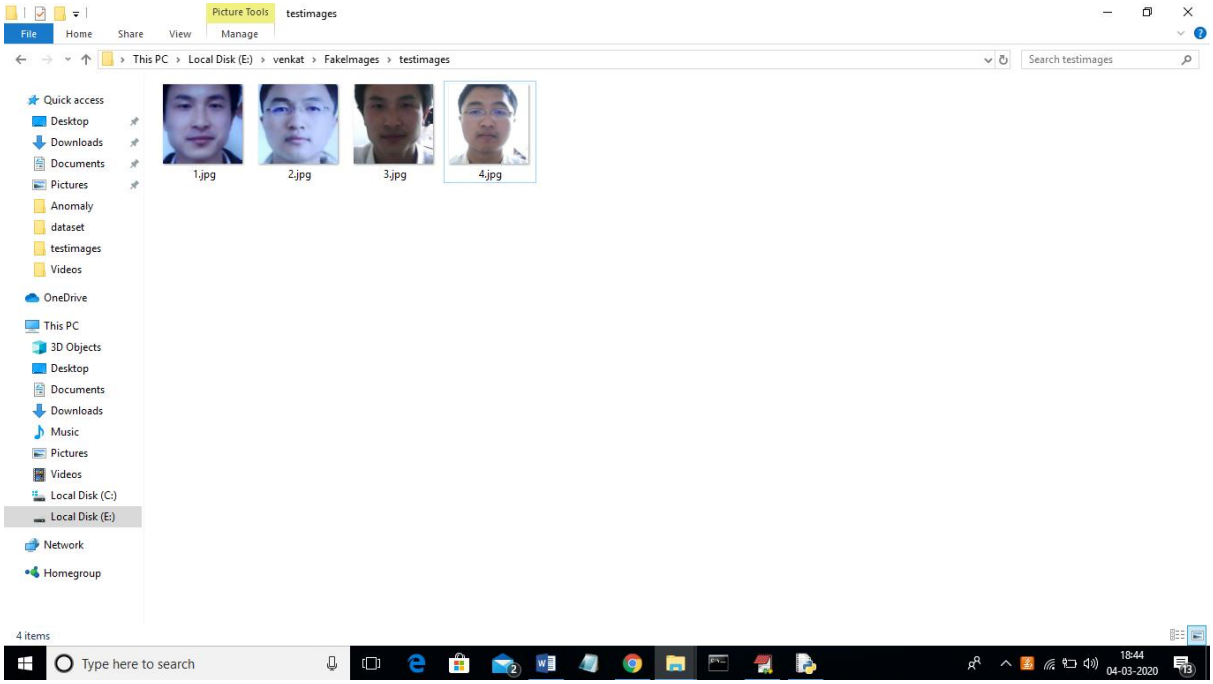
In above screen click on ‘Generate Image Train & Test Model’ button to generate CNN model using LBP images contains inside LBP folder.



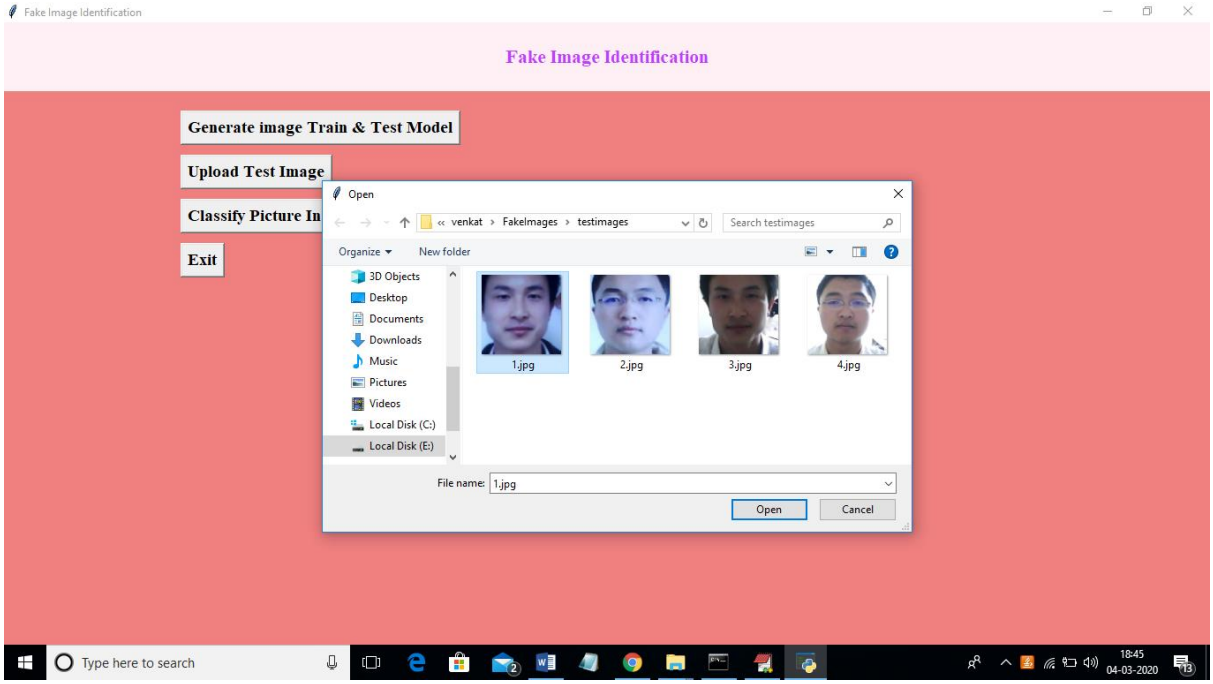
In above screen we can see CNN LBPNET model generated. Now click on 'Upload Test Image' button to upload test image



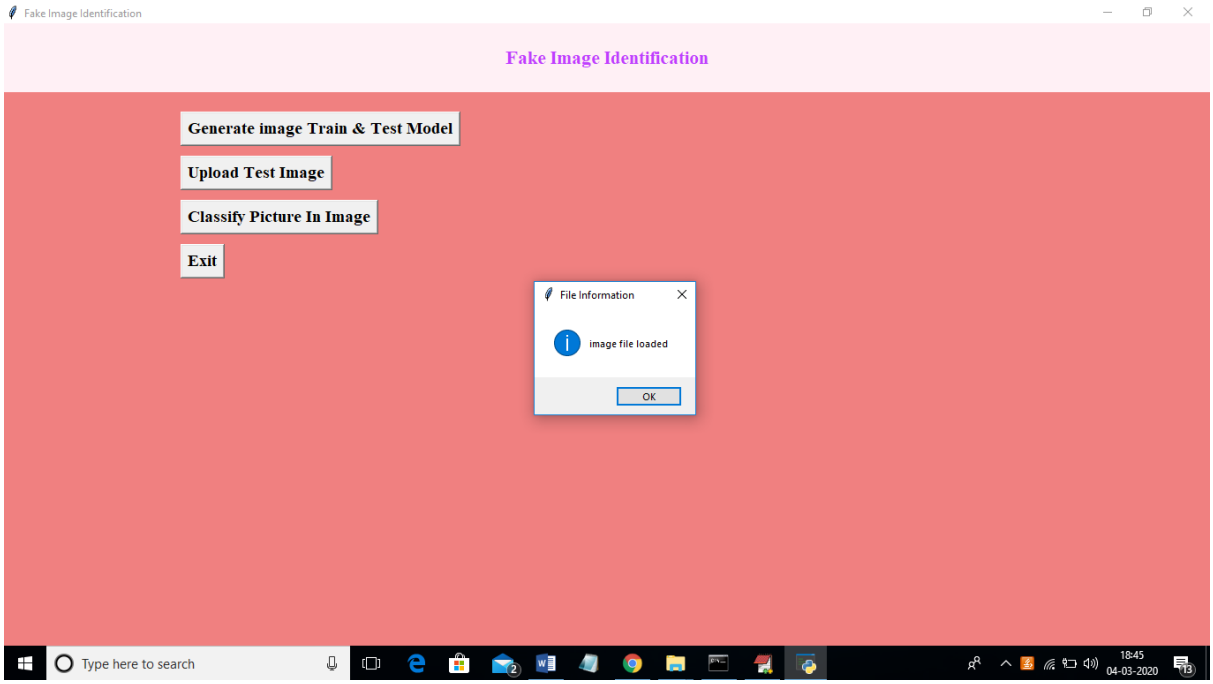
In above screen we can see two faces are there from same person but in different appearances. For simplicity I gave image name as fake and real to test whether application can detect it or not. In above screen I am uploading fake image and then click on 'Classify Picture In Image' button to get below result



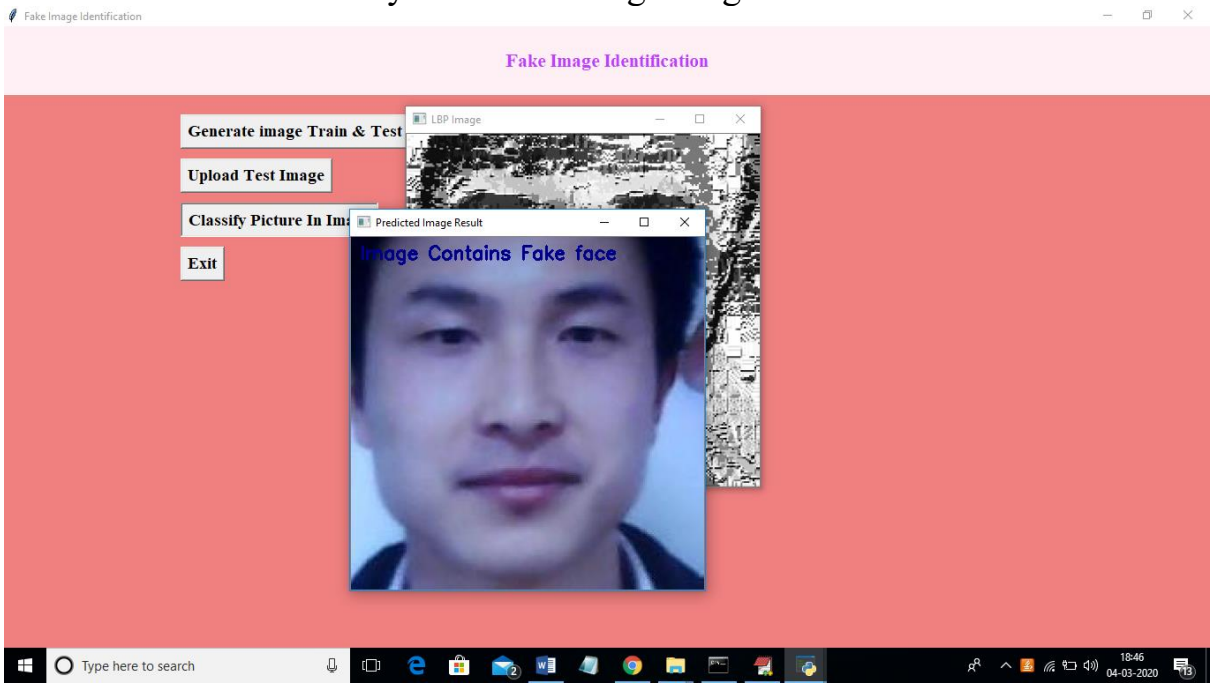
In above screen we can see all real face will have normal light and in fake faces peoples will try some editing to avoid detection but this application will detect whether face is real or fake



In above screen I am uploading 1.jpg and after upload click on open button to get below screen



And now click on ‘classify Picture in Image’ to get below details



In above screen we are getting result as image contains Fake face. Similarly u can try other images also. If u want to try new images then u need to send those new images to us so we will make CNN model to familiar with new images so it can detect those images also.

CHAPTER - 8

CONCLUSION & FUTURE ENHANCEMENT

8.1 CONCLUSION:

In this paper, we have proposed a novel common fake feature network based the pairwise learning, to detect the fake face/general images generated by state-of-the-art GANs successfully. The proposed CFFN can be used to learn the middle- and high-level and discriminative fake feature by aggregating the cross-layer feature representations into the last fully connected layers. The proposed pairwise learning can be used to improve the performance of fake image detection further. With the proposed pairwise learning, the proposed fake image detector should be able to have the ability to identify the fake image generated by a new GAN. Our experimental results demonstrated that the proposed method outperforms other state-of-the-art schemes in terms of precision and recall rate.

FUTURE ENHANCEMENT

The proposed CFFN can be used to learn the

middle- and high-level and discriminative fake features by aggregating the cross-layer feature representations into the

last fully connected layers. The proposed pairwise learning can be used to improve the performance of fake image

detection further. With the proposed pairwise learning, the proposed fake image detector should be able to have theability to identify the fake image generated by a new GAN.

For future, work square measure for instance employing an additional complicated and deeper model for unpredictable

issues. Integration of deep neural networks with the idea of increased learning, wherever the model is simpler. Neural

network solutions seldom take under consideration non-linear feature interactions and non-monotonous short-run serial

patterns, that square measure necessary to model user behavior in thin sequence information. A model is also integrated

with neural networks to unravel this downside. The dataset can be inflated and another variety of images can be used

for coaching, for instance, gray-scale pictures.

REFERENCES

1. Karras, T.; Aila, T.; Laine, S.; Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. arXiv Preprint, arXiv:1710.10196 2017. 256

2. Brock, A.; Donahue, J.; Simonyan, K. Large scale gan training for high fidelity natural image synthesis. arXiv Preprint, arXiv:1809.11096 2018.

3. Zhu, J.Y.; Park, T.; Isola, P.; Efros, A.A. Unpaired image-to-image translation using cycle-consistent 259 adversarial networks. arXiv Preprint, 2017.

4. AI can now create fake porn, making revenge porn even more complicated,. http://theconversation.com/ai-can-now-create-fake-porn-making-revenge-porn-even-more-complicated-92267, 262 2018.

5. Hsu, C.; Lee, C.; Zhuang, Y. Learning to detect fake face images in the Wild. 2018 International Symposium 264 on Computer, Consumer and Control (IS3C), 2018, pp. 388–391. doi:10.1109/IS3C.2018.00104.

6. H.T. Chang, C.C. Hsu, C.Y.a.D.S. Image authentication with tampering localization based on watermark 266 embedding in wavelet domain. Optical Engineering 2009, 48, 057002.

7. Hsu, C.C.; Hung, T.Y.; Lin, C.W.; Hsu, C.T. Video forgery detection using correlation of noise residue. Proc. of the IEEE Workshop on Multimedia Signal Processing. IEEE, 2008, pp. 170–174.

8. Farid, H. Image forgery detection. IEEE Signal Processing Magazine 2009, 26, 16–25.

9. Huaxiao Mo, B.C.; Luo, W. Fake Faces Identification via Convolutional Neural Network. Proc. of the ACM Workshop on Information Hiding and Multimedia Security. ACM, 2018, pp. 43–47.

10. Marra, F.; Gragnaniello, D.; Cozzolino, D.; Verdoliva, L. Detection of GAN-Generated Fake Images over Social Networks. Proc. of the IEEE Conference on Multimedia Information Processing and Retrieval, 2018, 274 pp. 384–389. doi:10.1109/MIPR.2018.00084.

11. Chollet, F. Xception: Deep learning with depthwise separable convolutions. Proc. of the IEEE conference on 276 Computer Vision and Pattern Recognition 2017, pp. 1610–02357.