

IMAGE FORGERY DETECTION USING CONVOLUTION NEURAL NETWORKS



A Project report submitted in partial fulfillment of
requirements for the award of degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

by

S. ABRARUL MATEEN (199X1A05J0)

P. VENKATA SUNIL KUMAR (199X1A05H6)

V. THARUN KUMAR REDDY (199X1A05J6)

Y. NEHA (199X1A05G9)

Under the esteemed guidance of

Dr. K. Ishthaq Ahamed

Associate Professor

Department of C.S.E.

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTAPURAMU)

2022 – 2023

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTAPURAMU)



CERTIFICATE

This is to certify that the Project Work entitled 'Image Forgery Detection Using Convolution Neural Networks' is a bonafide record of work carried out by

S. ABRARUL MATEEN (199X1A05J0)

P. VENKATA SUNIL KUMAR (199X1A05H6)

V. THARUN KUMAR REDDY (199X1A05J6)

Y. NEHA (199X1A05G9)

Under my guidance and supervision in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE & ENGINEERING

Dr. K. Ishthaq Ahamed

Associate Professor,
Department of C.S.E.,
G. Pulla Reddy Engineering College,
Kurnool.

Dr. N. Kasiviswanath

Professor & Head of the Department,
Department of C.S.E.,
G. Pulla Reddy Engineering College,
Kurnool.

DECLARATION

We hereby declare that the project titled “**IMAGE FORGERY DETECTION USING CONVOLUTION NEURAL NETWORKS**” is an authentic work carried out by us as the student of **G. PULLA REDDY ENGINEERING COLLEGE(Autonomous) Kurnool**, during 2022-23 and has not been submitted elsewhere for the award of any degree or diploma in part or in full to any institute.

S. ABRARUL MATEEN
(199X1A05J0)

P. VENKATA SUNIL KUMAR
(199X1A05H6)

V.THARUN KUMAR REDDY
(199X1A05J6)

Y. NEHA
(199X1A05G9)

ACKNOWLEDGEMENT

We wish to express our deep sense of gratitude to our project guide **Dr.K.Ishthaq Ahamed**, Associate Professor of CSE Department, G. Pulla Reddy Engineering College, for his immaculate guidance, constant encouragement and cooperation which have made possible to bring out this project work.

We are grateful to our project incharge **Sri. K.Sudhakar**, Assistant Professor of CSE Department, G.Pullu Reddy Engineering College, for helping us and giving us the required information needed for our project work.

We are thankful to our Head of the Department **Dr.N.Kasiviswanath**, for his whole hearted support and encouragement during the project sessions.

We are grateful to our respected Principal **Dr.B.Sreenivasa Reddy** for providing requisite facilities and helping us in providing such a good environment.

We wish to convey our acknowledgements to all the staff members of the Computer Science and Engineering Department for giving the required information needed for our project work.

Finally, we wish to thank all our friends and well-wishers who have helped us directly or indirectly during the course of this project work.

ABSTRACT

The information is shared in form of images through newspapers, magazines, internet, or scientific journals. Due to the software like Photoshop, GIMP, and Coral Draw, it becomes very hard to differentiate between original image and tampered image. Traditional methods for image forgery detection mostly use handcrafted features. The problem with the traditional approaches of detection of image tampering is that most of the methods can identify a specific type of tampering by identifying a certain feature in the image.

Nowadays, deep learning methods are used for image tampering detection. These methods reported better accuracy than traditional methods because of their capability of extracting complex features from image. In this project, we are going to use convolution neural networks along with the Error Level Analysis (ELA) to find whether the image is forged or not.

CONTENTS

	Page No
1. Introduction	1
1.1 Introduction	2
1.2 Motivation.....	6
1.3 Problem Definition.....	6
1.4 Objective of the Project	7
1.5 Limitations of the Project	7
1.6 Organization of the Report	8
2. System Specifications.....	9
2.1 Software Specifications.....	10
2.2 Hardware Specifications.....	12
3. Literature Survey.....	13
3.1 Existing System.....	14
3.2 Disadvantages of Existing System.....	15
3.3 Proposed System.....	15
4. Design and Methodology.....	19
4.1 Design Flowchart.....	20
4.2 Dataset Collection.....	22
4.3 ELA Conversion and Preprocessing.....	22
4.4 Splitting data into training and testing sets.....	27
4.5 Convolution Neural Networks.....	28
4.6 Configuring the CNN using Keras.....	40

4.7 Saving and Loading the Model.....	48
4.8 Analyzing the Accuracy and Loss of the Model.....	49
4.9 Predicting the Output.....	51
4.10 Confusion Matrix of the Model.....	51
4.11 Front end for the project.....	55
4.12 Backend using Flask.....	56
4.13 Output Screenshots.....	58
4.14 Results.....	59
5. Conclusion	60
REFERENCES	

LIST OF FIGURES

Fig No	Description	Page No
1.1	Image Forgery Detection Approaches	3
3.1.1	Block Diagram of Existing System	14
3.1.2	Block Diagram of Proposed System	15
3.3.1	ELA Image of Original Image	17
3.3.2	ELA Image of Resaved Image	17
3.3.3	ELA Image of Digitally Modified Resaved Image	18
4.1.1	Design Flowchart	21
4.5.1	Overall CNN Architecture	29
4.5.2	Convolution Operation	30
4.5.3	Applying Padding to an Input Image	32
4.5.4	ReLU Activation Function	34
4.5.5	Max Pooling	37
4.5.6	Flattening Operation	39
4.8.1	Epoch v/s Accuracy over Training & Testing Sets	49
4.8.2	Epoch v/s Loss over Training & Testing Sets	50
4.10.1	Confusion Matrix Format	52
4.13.1	Model Detecting Image as Real	58
4.13.2	Model Detecting Image as Fake	58
4.14.1	Epoch v/s Accuracy of Existing & Proposed Systems	59

LIST OF TABLES

Table No	Description	Page No
3.3.1	Forgery Detection at various properties of the image	16
4.5.1	Different Activation Functions	33

INTRODUCTION

1.1 INTRODUCTION

Nowadays, images play a vital role in several fields like medical, education, digital forensics, sports, scientific research, news media, etc. and they are used as one of the main sources of information. Due to software like Photoshop, GIMP, Coral Draw and android applications like photo hacker, it is very easy to create a forged image. The genuineness of image becomes very crucial in the cases where the image is used as a proof in court of law.

Image manipulation is any type of operation that is performed on digital images by using any software, it is also referred as image editing. Image forgery is the technique to modify the content of an image which contradicts with some fact happened in past. Image tampering is a type of image forgery which replaces some content of an image with new content. If the new content is copied from the same image itself then it is called copy-move tampering and if the new content is copied from different image, then it is called image splicing.

1.1.1 Image Forgery Types

Image forgery is the manipulation of digital images to hide or alter some information. There are different types of image forgery techniques, such as:

- **Morphing:** blending two or more images to create a new one.
- **Compositing:** combining parts of different images to create a new one.
- **Retouching:** modifying the appearance of an image by removing or adding some details.
- **Enhancing:** improving the quality of an image by adjusting the brightness, contrast, color, etc.
- **Computer painting:** creating an image from scratch using digital tools.
- **Generating:** creating an image using artificial intelligence methods such as generative adversarial networks (GANs).

Some of these techniques can be used for benign purposes, such as artistic expression or entertainment, but some can also be used for malicious purposes, such as deception or propaganda. Therefore, it is important to detect and verify the authenticity and integrity of digital images.

1.1.2 Traditional Image Forgery Detection Approaches

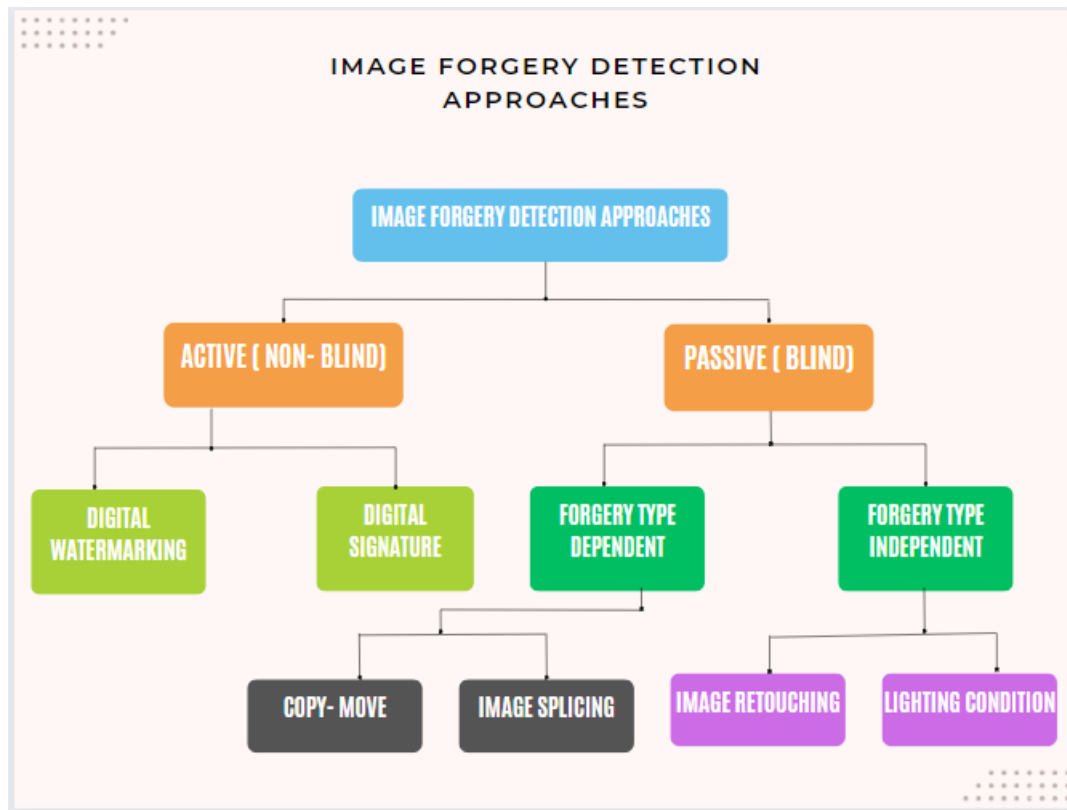


Fig. 1.1 – Image Forgery Detection Approaches

The image manipulation detection approaches can be classified into two categories, namely

- Active
- Passive

1.1.2.1 Active / Non- Blind Approaches

In active approach, additional information (such as digital watermark) is embedded in the image during the image acquisition stage or at some later stage by authorized person. The active approach uses this embedded information for manipulation detection.

Active image forgery detection approaches includes techniques such as

- **Digital Watermarking:** Digital watermarking is a technique of embedding a hidden signal or information into a digital media such as audio, video, or image. The purpose of digital watermarking is to identify the ownership or the source of the media, to verify its authenticity or integrity, or to show its identity.

- **Digital Signature:** A digital signature is a way of verifying the identity and the integrity of a digital message or document. A digital signature is created by using a cryptographic algorithm that uses a pair of asymmetric keys: a private key and a public key.

The sender of the message or document uses their private key to sign the message or document, which produces a unique code or value that is attached to the message or document. This code or value is called the digital signature.

The receiver of the message or document uses the sender's public key to verify the digital signature. The verification process checks if the message or document has been altered or tampered with, and if it matches the sender's identity.

1.1.2.2 Passive / Blind Approaches

The passive approaches do not depend on the additional information for forgery detection. These approaches are also referred as “blind approaches” as the approaches do not use additional information for forgery detection. The passive approaches extract the features from the image and use these features for forgery detection. The passive approaches can be classified as follows:

- Forgery Type Dependent
- Forgery Type Independent

Forgery Type Dependent Approaches

Passive forgery type dependent approaches for forgery detection can be categorized into:

- **Copy- Move**
 - In copy-move tampering, a region (of any size) from the image is selected to perform copy-move operation and it is pasted to some other part of the same image. So there will be very high correlation between these two regions. The objective of the copy-move tampering detection method is to detect duplicated regions in the given image. The similarities (correlation) or distance between features extracted from two different regions of the image indicate the duplication.

- **Image Splicing:**

- Image splicing is composite of two or more images. Copy move tampering involves region duplication which is absent in case of image splicing. This makes localization of image splicing a difficult task as compared to detection of copy move. The image splicing detection methods are based on the clues that are left after tampering process.
- The examples of common tampering clues are discontinuity in edge, inconsistency arise due to lighting condition, geometric situation and camera. The region which is copied in image leaves an abnormal artifact on the edges of tampered region, referred as edge discontinuity.
- Different cameras have different characteristics, so that images taken from different cameras have different properties and based on this clue tampering can be confirmed.
- Generally, the lighting effect of two images is not always same, there may be little difference between them. So the tampered region may have inconsistent lighting with other regions of the image that is lighting inconsistency.

Forgery Type Independent Approaches

Forgery type independent approaches are methods for detecting image forgeries that do not depend on the specific type or technique of manipulation, but rather on some general or common traces or artifacts that are left behind by the manipulation process.

Some examples of forgery type independent approaches are:

- **Image Retouching**

- This method detects the presence of re-sampling operations, such as scaling, rotation, or skewing, that are often applied to manipulate images. Re-sampling operations introduce some periodic patterns or correlations in the pixel values that can be detected by using statistical analysis or frequency domain analysis.

- **Lighting Condition**

- This method detects the inconsistency of lighting conditions or directions among different objects or regions in an image. Lighting inconsistency can indicate that the image has been composed from different sources with different illumination

settings. This method can use techniques such as physics-based modeling, face recognition, or deep learning to estimate and compare the lighting parameters.

1.2 MOTIVATION

Though there are traditional image forgery detection techniques, due to the development of the GPU technologies and the success of Deep Learning (DL) techniques in the domain of computer vision motivated the researchers to apply DL models for image forgery detection.

DL combines features (low/mid/high level) extraction and classification phases. The technique is data-driven and capable of automatically learning abstract and complex features, necessary to identify tampered regions. Moreover, it also saves time and energy required to find hand-crafted features from tampered images. However, training of deep learning models is hard and requires high computational power and very large amount of data. There are many models of DL such as Convolutional Neural Networks (CNN), Deep Neural Network (DNN) and Recurrent Neural Network (RNN). Convolutional Neural Networks (CNN) is popular amongst these DL models.

CNN has the convolution layer which acts as a discriminator and feature extractor. Generally, CNNs extract features based on image content rather than extracting image manipulation features. This layer considers the local structural relationship between pixels instead of considering the content of an image, because manipulation alters some local relationships. This can detect multiple tampering in an image. One problem with any detection method is that it cannot give satisfactory results for multiple tampering attacks.

1.3 PROBLEM DEFINITION

The information is shared in form of images through newspapers, magazines, internet, or scientific journals. Due to software like Photoshop, GIMP, and Coral Draw, it becomes very hard to differentiate between original image and tampered image. Traditional methods for image forgery detection mostly use handcrafted features. The problem with the traditional approaches of detection of image tampering is that most of the methods can identify a specific type of tampering by identifying a certain feature in image.

Nowadays, deep learning methods are used for image tampering detection. These methods reported better accuracy than traditional methods because of their capability of extracting

complex features from image. In this project, we are going to use convolution neural networks along with advanced image forgery detection techniques such as Error Level Analysis (ELA) to find whether the image is forged or not.

1.4 OBJECTIVE OF THE PROJECT

The main objective of the project is to use deep learning techniques to detect the image forgery. This in turn reduces the human work to detect the forgery in the image by using traditional image forgery detection techniques. As deep neural networks extracts the its own features from the image, it is not specific to particular forgery technique. This project will be really helpful in this internet world consisting full of image and video data to detect any forgeries and thereby reducing the misleading of information.

1.5 LIMITATIONS OF THE PROJECT

As we are using CNN, it requires a lot of computational resources and memory, especially when they have many layers and parameters. The ELA technique used in this project cannot be applied to images without lossy compression, such as PNG or BMP images. The technique also cannot detect manipulation that does not affect the compression artifacts, such as color adjustment or sharpening.

The ELA technique may produce false positives or false negatives due to various factors, such as noise, lighting, texture, camera settings, etc. The interpretation of the difference image is subjective and may depend on the context and the expertise of the analyst.

The ELA technique can be fooled by applying multiple or uniform compressions to the manipulated image, or by reducing the color space of the image. These methods can reduce or eliminate the variation of compression artifacts. The ELA should be used in combination with other methods and techniques for image forensics.

1.6 ORGANIZATION OF THE PROJECT

Report includes the stepwise implementation of the working application generated and it describes the overview of how effectively the project can be implemented and makes the viewer of the report to get an overview of the project.

- Chapter two includes project software and hardware specifications.
- Chapter three includes literature survey.
- Chapter four includes design and implementation which includes source code, testing and validation.
- Chapter five includes conclusion.

SYSTEM SPECIFICATIONS

2. SYSTEM SPECIFICATIONS

To be used efficiently, all computer software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to existing computer systems than technological advancements.

Non-functional requirements

Non-functional requirements are the functions offered by the system. It includes time constraints and constraints on the development process and standards. The non-functional requirements are as follows: 1. Speed: The system should process the given input into output within appropriate time. 2. Ease of use: The software should be user friendly. Then the customers can use easily, so it doesn't require much training time. 3. Reliability: The rate of failures should be less then only the system is more reliable 4. Portability: It should be easy to implement in any system.

2.1 SOFTWARE SPECIFICATIONS

Software Requirement Specification is the starting point of the software developing activity. As the system grew complex it became evident that the goal of the entire system cannot be easily comprehended. Hence the need for the requirement phase arose. The software is initiated by the client's needs. The SRS is the means of translating the ideas of the minds of the client's (the input) into a formal document (the o/p of the requirement phase).

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed. The system should be able to interface with the existing system.

- The system should be accurate
- The system should be better than the existing system

Software requirements

- Supported web browsers - Google Chrome, Microsoft Internet Explorer (Version 11.3.3 onward)
- Compatible Version of any Mac, Linux, Windows OS
- Python 3.9.5 (32 bit)
- Flask
- Anaconda Python Distribution
- Libraries
 - Python Imaging Library
 - Scikit -Learn
 - Keras
 - Tensorflow
 - ngrok
 - NumPy
 - Pandas
 - Matplotlib

2.2 HARDWARE SPECIFICATIONS

The most common set of requirements used by any operating system or application is the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in the operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub- sections discuss the various aspects of hardware requirements. They are peripherals, secondary storage, processing power, memory. All computer operating systems are designed for a particular computer architecture. Most software applications are limited to operating systems running on architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture.

The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored.

Server Recommendations

- ≥ 8 GB RAM
- ≥ 8 Core Processor
- 500 GB hard disk space
- Ensure C: drive has 100 GB plus free hard disk space.
- Fast and High-speed internet connection

LITERATURE SURVEY

3. LITERATURE SURVEY

3.1 EXISTING SYSTEM

In digital forensics, the detection of the presence of tampered images is of significant importance. The problem with the existing literature is that majority of them identify certain features in images tampered by a specific tampering method (such as copy-move, splicing, etc). This means that the method does not work reliably across various tampering methods. In addition, in terms of tampered region localization, most of the work targets only JPEG images due to the exploitation of double compression artifacts left during the re-compression of the manipulated image. However, in reality, digital forensics tools should not be specific to any image format and should also be able to localize the region of the image that was modified.

In this paper, they have proposed a convolution neural network model with two convolution layers, two pooling layers, and a fully connected layer in order to detect tampered images in different image formats. In our experiments, we were able to obtain an overall tampered region localization accuracy of 66.33% over both JPEG and TIFF images from CASIA dataset, with a fall-out of 62.17%.

3.1.1 Block Diagram of Existing System

Here is the block diagram or design framework of existing system. It consists of two convolution layers to extract features from the input image, two pooling layers used to extract main features by and reduces the dimensionality of the data and has a fully connected layer to classify the images as either forged or authentic. The below figure describes the architecture of the proposed system.

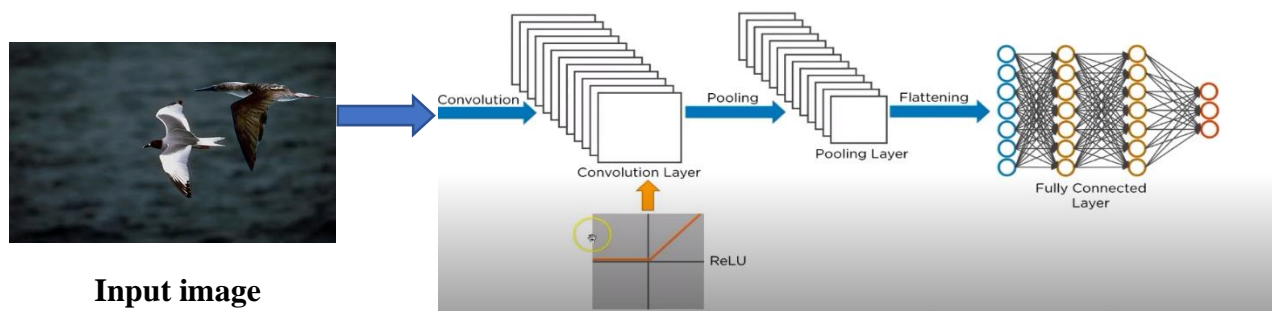


Fig. 3.1.1 – Block Diagram of Existing System

3.2 DISADVANTAGES OF EXISTING SYSTEM

The proposed system gives the raw input pixels of the image to the convolution neural network which does not detect the forgery in the image as desired.

The existing system only concentrates on whether the image is forged or not but it does not focus on the area where the digital forgery has occurred. It also concentrates on finding the forged edges of an image but not on the surface of the objects in the image. It doesn't use advanced image forgery techniques to build the model. The accuracy of the model can be improved further using the advanced image forgery detection techniques such as Error Level Analysis (ELA).

3.3 PROPOSED SYSTEM

3.3.1 Block Diagram of Proposed System

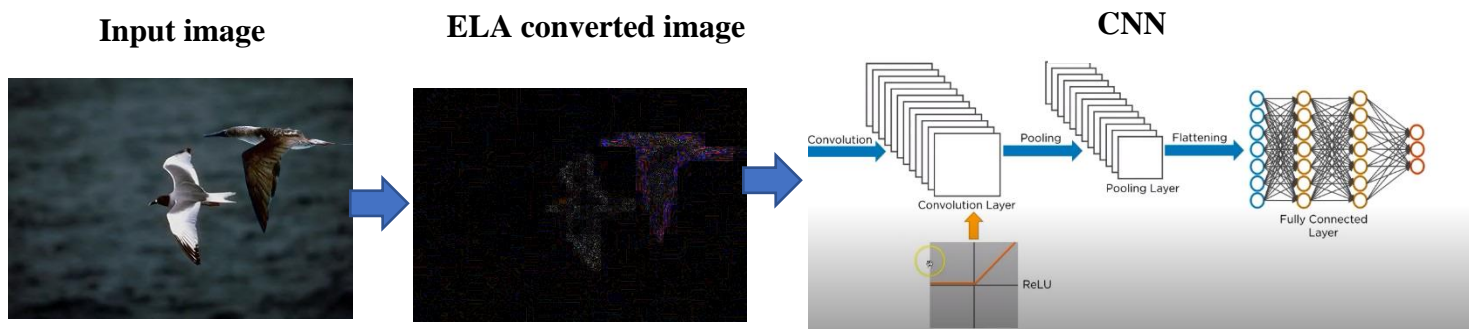


Fig. 3.1.2 – Block Diagram of Proposed System

In this project, we are going to use convolution neural networks to detect the image forgery. Instead of giving just raw pixels of an image as an input to the CNN, we are giving the Error Level Analysis (ELA) version of the image to the model. ELA is a scientific technique which is used to find whether the image is forged or not. This improves the model efficiency to a great extent.

Steps followed in proposed system:

- The input is converted into Error Level Analysis (ELA) version of the image.
- The ELA version of the image is given as input to the convolution neural network

- Our CNN has two convolution layers and two pooling layers along with the one fully connected layer.
- The convolution layer extracts various features in the image and the pooling layer is used for dimensionality reduction.
- The result is flattened after the convolution and pooling operation and is fed to the fully connected neural network which is used to classify the images as either forged or authentic.

3.3.2 Error Level Analysis (ELA)

Now, let us also see briefly about Error Level Analysis (ELA). Error Level Analysis (ELA) permits identifying areas within an image that are at different compression levels. With JPEG images, the entire picture should be at roughly the same level. If a section of the image is at a significantly different error level, then it likely indicates a digital modification. ELA highlights differences in the JPEG compression rate. Regions with uniform coloring, like a solid blue sky or a white wall, will likely have a lower ELA result (darker color) than high-contrast edges. The things to look for:

Property	Expectations
Edges	Similar edges should have similar brightness in the ELA result. All high-contrast edges should look similar to each other, and all low-contrast edges should look similar. With an original photo, low-contrast edges should be almost as bright as high-contrast edges.
Textures	Similar textures should have similar coloring under ELA. Areas with more surface detail, such as a close-up of a basketball, will likely have a higher ELA result than a smooth surface.
Surfaces	Regardless of the actual color of the surface, all flat surfaces should have about the same coloring under ELA.

Table- 3.3.1 – Forgery detection at various properties of the image

With ELA, every grid that is not optimized for the quality level will show grid squares that change during a resave. For example, digital cameras do not optimize images for the

specified camera quality level (high, medium, low, etc.). Original pictures from digital cameras should have a high degree of change during any resave (high ELA values). Each subsequent resave will lower the error level potential, yielding a darker ELA result. With enough resaves, the grid square will eventually reach its minimum error level, where it will not change anymore.

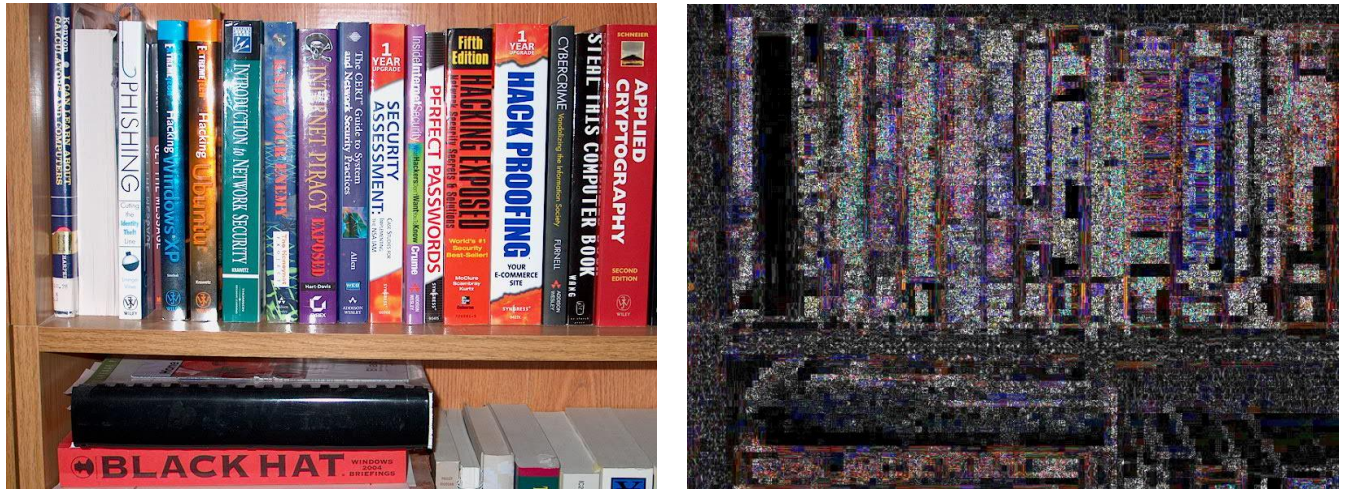


Fig.3.3.1 – ELA image of original image

The fig.3.3.1 is an original digital photograph (Source: Hacker Factor) has high ELA values, represented by white colors in the ELA. The sections that are black correspond to the solid white book and the black 8x8 squares in the original image. Solid colors compress very well, so these are already at their minimum error levels.

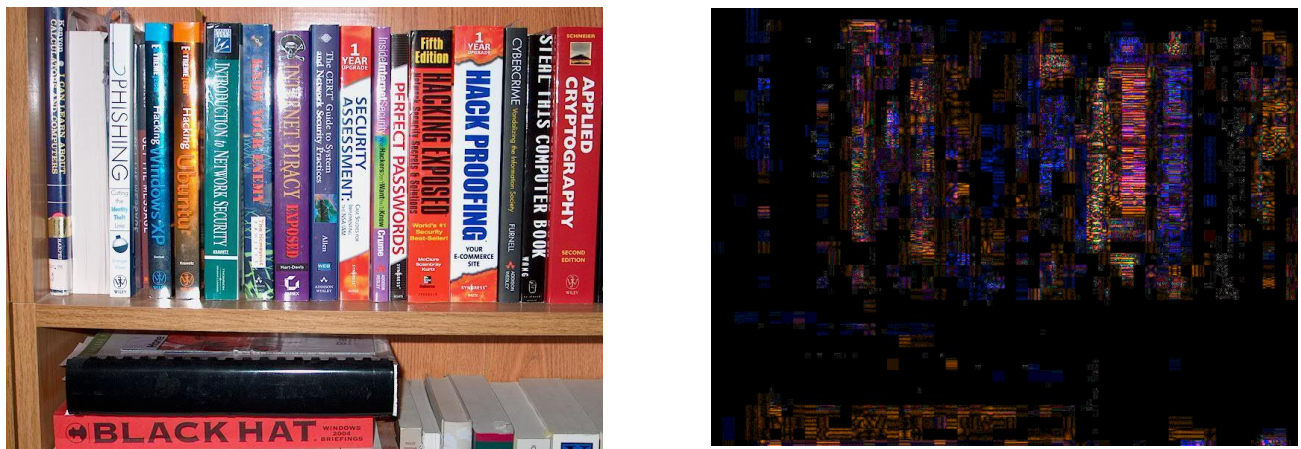


Fig. 3.3.2 – ELA image of resaved image

The fig.3.3.2 is the original image was resaved one time. To the human there is no visible difference between the original and the resave image. However, ELA shows much more black and more dark colors. If this image were resaved again, it will have even lower (darker) ELA values.

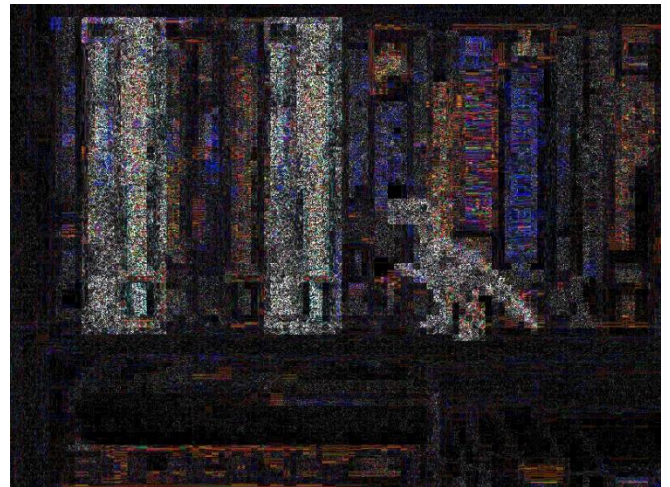
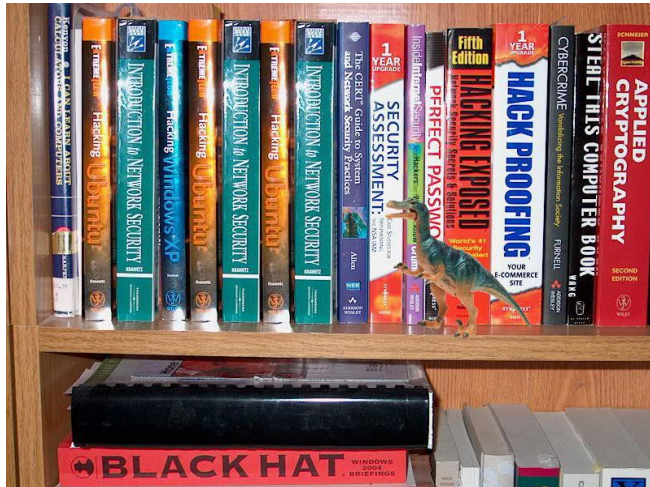


Fig. 3.3.3 – ELA image of digitally modified resaved image

The fig. 3.3.3 is the resaved image was digitally modified: books were copied and a toy dinosaur was added. ELA clearly shows the modified areas as having higher ELA values.

DESIGN METHODOLOGY

4. DESIGN METHODOLOGY

4.1 DESIGN FLOWCHART

Our project is done using the following steps.

- Dataset Collection
- ELA Conversion and Preprocessing
- Splitting data into training and testing sets
- Training the CNN model
- Evaluate the model performance
- Compare Results
- Model Deployment

These steps make our model building easier and efficient. Even after deploying the model, there might be new issues and questions arise from seeing the model in action. So, we need to follow up the project and make this process iterative to gain more efficient results. So, following these steps in the specified order is very crucial for building any machine learning or artificial neural network models. The design flowchart of our project is shown in the fig 4.1.1

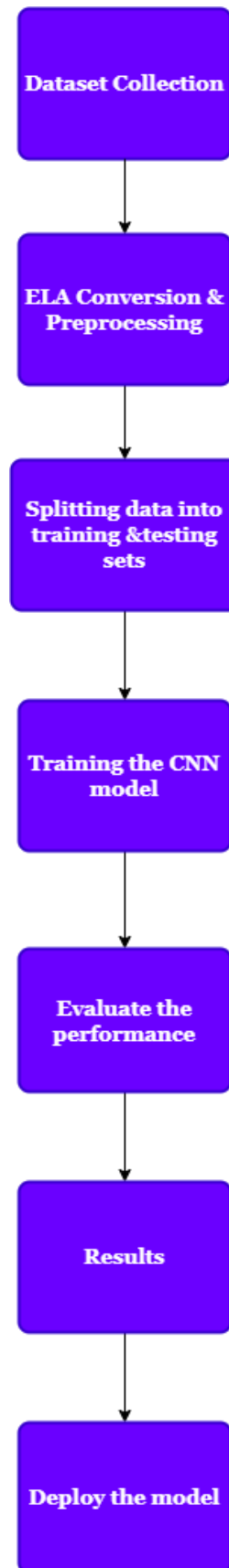


Fig. 4.1.1 Design Flowchart

4.2 DATASET COLLECTION

Here we are using CASIA v2 dataset. The details of the dataset are discussed further in the next section.

4.2.1 CASIA Dataset

CASIA v2 dataset is a large-scale image dataset for forgery classification. It was introduced by Jing Dong et al. in CASIA Image Tampering Detection Evaluation Database. The dataset contains 4795 images, of which 1701 are authentic and 3274 are forged. The forged images are created by using various image editing tools, such as Photoshop, GIMP, etc. The images are divided into eight categories: animals, architecture, natural scene, objects, painting, people, plants and text. The dataset is labeled by using a binary mask that indicates the tampered regions in the forged images. The dataset is publicly available and can be downloaded from <http://forensics.idealtest.org/casiav2/>.

The dataset is widely used to evaluate different methods for image forgery detection, such as copy-move forgery detection, splicing detection, inpainting detection, etc. The dataset poses various challenges for forgery detection, such as low-quality images, complex backgrounds, similar textures, etc. The dataset also provides a benchmark for comparing different methods based on various metrics, such as accuracy, precision, recall, F1-score, etc. This dataset is one of the most popular and comprehensive datasets for image forgery classification research.

4.3 ELA CONVERSION AND PREPROCESSING

4.3.1 Error Level Analysis (ELA)

Error level analysis (ELA) is a technique for detecting image manipulation by analyzing the compression artifacts in digital images with lossy compression, such as JPEG. The technique is based on the principle that different parts of an image may have different levels of compression artifacts due to different editing operations, such as cropping, resizing, cloning, etc. By applying a uniform compression to the image and comparing it with the original image, the technique can highlight the areas with different levels of compression artifacts, which may indicate tampering.

The technique works as follows:

- The original image is re-saved at a known quality level, such as 95%, using a lossy compression algorithm, such as JPEG. This introduces a uniform level of compression artifacts across the image.
- The re-saved image is subtracted from the original image, pixel by pixel, to produce a difference image. This image shows the variation of compression artifacts in different parts of the image.
- The difference image is enhanced by adjusting the brightness and contrast to make the variation more visible. The areas with higher variation are usually brighter than the areas with lower variation.


The technique can be used to detect various types of image manipulation, such as copy-move forgery, splicing, inpainting, etc. The technique can also be used to identify the source of an image by analyzing the metadata and the compression artifacts.

However, the technique has some limitations and challenges:

- The technique cannot be applied to images without lossy compression, such as PNG or BMP images. The technique also cannot detect manipulation that does not affect the compression artifacts, such as color adjustment or sharpening.
- The technique may produce false positives or false negatives due to various factors, such as noise, lighting, texture, camera settings, etc. The interpretation of the difference image is subjective and may depend on the context and the expertise of the analyst.
- The technique can be fooled by applying multiple or uniform compressions to the manipulated image, or by reducing the color space of the image. These methods can reduce or eliminate the variation of compression artifacts.

Therefore, error level analysis is not a conclusive or robust method for detecting image manipulation. It is only a tool that can provide some clues or hints for further investigation. It should be used in combination with other methods and techniques for image forensics.

4.3.2 Implementation of ELA to an image



```
def ela_image(path, quality=90):
    temp_filename = 'temp_file_name.jpg'
    ela_filename = 'temp_ela.png'
    image = Image.open(path).convert('RGB')
    image.save(temp_filename, 'JPEG', quality = quality)
    temp_image = Image.open(temp_filename)
    ela_image = ImageChops.difference(image, temp_image)
    extrema = ela_image.getextrema()
    max_diff = max([ex[1] for ex in extrema])
    if max_diff == 0:
        max_diff = 1
    scale = 255.0 / max_diff
    ela_image = ImageEnhance.Brightness(ela_image).enhance(scale)
    return ela_image
```

4.3.3 Preprocessing an ELA image

Here we are resizing the image to 128 x 128 pixel and then flatten it to a vector and then normalizes the pixel values.

The below function implements the preprocessing task of an ELA image

```
image_size=(128,128)


def preprocessing(image_path):
    return
    np.array(ela_image(image_path).resize(image_size)).flatten()/255
```

Using this preprocessing function, we are preprocessing 3000 authentic images and 5000 tampered images from our CASIA v2 dataset. This step is shown below

```
# ELA converted images
X = []
# 0 for fake, 1 for real
Y = []

#Preprocessing 3000 Authentic Images
path = 'C:/Users/matee/Image_Forgery_Detection/CASIA2/Au/'
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('.jpg') or filename.endswith('.png') or
filename.endswith('.tif'):
            full_path = os.path.join(dirname, filename)
            X.append(preprocessing(full_path))
            Y.append(1)
            if len(Y) % 3000 == 0:
                print(f'Processing {len(Y)} images')
                break
    if len(Y) % 3000 == 0:
        break

print(len(X), len(Y))
```



```
#Preprocessing 5000 Tampered Images
path = 'C:/Users/matee/Image_Forgery_Detection/CASIA2/Tp/'
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('.jpg') or filename.endswith('.png') or
filename.endswith('.tif'):
            full_path = os.path.join(dirname, filename)
            X.append(preprocessing(full_path))
            Y.append(0)
            if len(Y) % 8000 == 0:
                print(f'Processing {len(Y)} images')
                break
    if len(Y) % 8000 == 0:
        break

print(len(X), len(Y))
```

4.3.4 Shuffling the data

Here we are using shuffle method available in sklearn.utils package to shuffle the data in a consistent way.

The shuffle method in sklearn is a function that shuffles arrays or sparse matrices in a consistent way. It is a convenience alias to `resample(*arrays, replace=False)` to do random permutations of the collections. You can use it to shuffle your dataframes or matrices randomly. You can also provide a 'random_state' parameter that allows you to produce reproducible results across multiple function calls by using the same seed number.

```
from sklearn.utils import shuffle
for i in range(10):
    X, Y = shuffle(X, Y, random_state=i)
```

4.3.5 Reshaping the data

We are reshaping the input ELA pixels stored in 'X' to 8000 x 128 x 128 x 3. We also modify the output class stored in 'Y' to categorical variable. The implementation of this case is shown below.

```
from keras.utils import to_categorical


X = np.array(X)
X = X.reshape(-1, 128, 128, 3)
Y = to_categorical(Y, 2)
```

4.4 SPLITTING THE DATA INTO TRAINING AND TESTING SETS

Now it's time to split our data into training and testing sets. This can be done using the `train_test_split` method available in `sklearn.model_selection`.

The `train_test_split` function in `sklearn` is a method that splits arrays or matrices into random train and test subsets. You can use it to divide your data into features and labels, and then split them into training and testing sets. You can specify the 'test_size' and 'train_size' parameters to control the proportion of the data that goes into each set. You can also provide a 'random_state' parameter to control the shuffling of the data before splitting. If you want to split your data in a stratified fashion, using the class labels, you can use the 'stratify' parameter.

The implementation can be seen below



```
from sklearn.model_selection import train_test_split

X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size = 0.3, random_state=42)
print(len(X_train), len(Y_train))
print(len(X_val), len(Y_val))
```

4.5 CONVOLUTION NEURAL NETWORKS

A convolutional neural network (CNN) is a type of artificial neural network that is designed to process structured arrays of data such as images. CNNs are widely used in computer vision and have become the state of the art for many visual applications such as image classification, segmentation, detection and recognition. CNNs can also be applied to other domains such as natural language processing, speech recognition and time series analysis.

4.5.1 Motivation for CNN

Convolutional neural networks (CNNs) are inspired by the biological processes of the animal visual cortex, where individual neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. Different neurons can detect different patterns or features in the input image, such as edges, corners, shapes or textures.

Here are the two main reasons for using CNNs instead of MLPs when working with image data. These reasons will motivate you to learn more about CNNs.

- **Spatial Information:** To use MLPs with images, we need to flatten the image. If we do so, spatial information (relationships between the nearby pixels) will be lost. So, accuracy will be reduced significantly. CNNs can retain spatial information as they take the images in the original format.

- **Reduction in parameters:** Unlike fully connected networks, where each neuron is connected to all neurons in the previous layer, CNNs use convolutional layers, where each neuron is connected only to a small region of the input. This reduces the number of parameters and computations required, and also captures the local dependencies in the data. Thus CNNs can reduce the number of parameters in the network significantly. So, CNNs are parameter efficient.

4.5.2 CNN Architecture

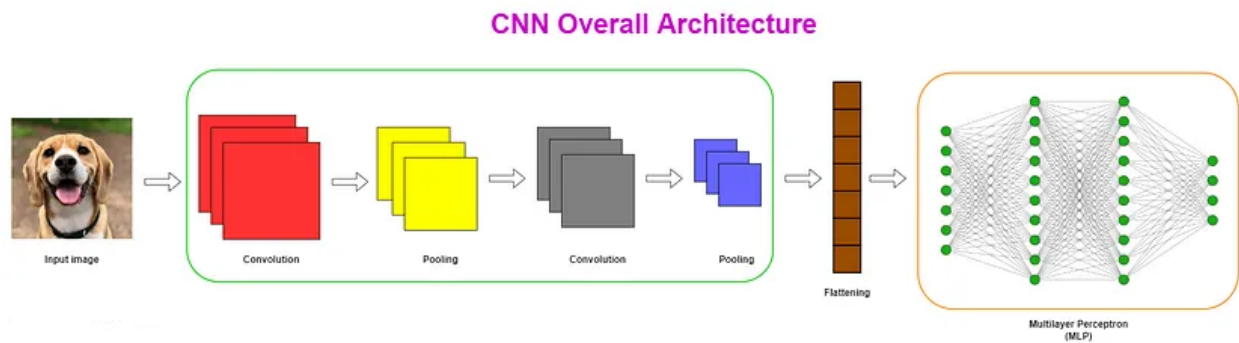


Fig. 4.5.1 Overall CNN Architecture

CNNs are composed of multiple layers that perform different operations on the input data. The main types of layers are

- Input Layer
- Convolution Layer
- Activation Layer
- Pooling Layer
- Fully Connected Layer
- Output Layer

4.5.2.1 Convolution Layers and Convolution Operation

The first layer in a CNN is a convolutional layer. There can be multiple convolutional layers in a CNN. The first convolutional layer takes the images as the input and begins to process.

Objectives

- Extract a set of features from the image while maintaining relationships between the nearby pixels.

There are three elements in the convolutional layer: **Input image**, **Filters** and **Feature map**. The convolution operation occurs in each convolutional layer.

The convolution operation is nothing but an *elementwise multiply-sum* operation between an image section and the filter. Now, refer to the following diagram.

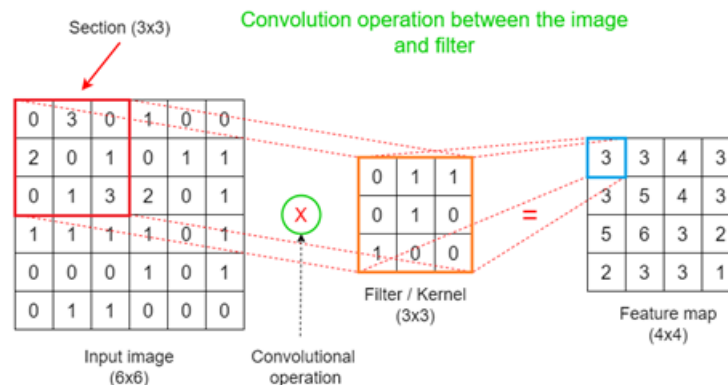


Fig. 4.5.2 Convolution Operation

Filter: This is also called *Kernel* or *Feature Detector*. This is a small matrix. There can be multiple filters in a single convolutional layer. The same-sized filters are used within a convolutional layer. Each filter has a specific function. Multiple filters are used to identify a different set of features in the image. The size of the filter and the number of filters should be specified by the user as hyperparameters. The size should be smaller than the size of the input image. The elements inside the filter define the *filter configuration*. These elements are a type of parameters in the CNN and are learned during the training.

Image section: The size of the image section should be equal to the size of the filter(s) we choose. We can move the filter(s) vertically and horizontally on the input image to create different image sections. The number of image sections depends on the *Stride* we use (more on this shortly).

Feature map: The feature map stores the outputs of different convolution operations between different image sections and the filter(s). This will be the input for the next pooling layer. The number of elements in the feature map is equal to the number of different image sections that we obtained by moving the filter(s) on the image.

The above diagram shows a convolution operation between an image section and a single filter. You can get row-wise or column-wise element multiplications and then summation.

#Row-wise

$$(0*0 + 3*1 + 0*1) + (2*0 + 0*1 + 1*0) + (0*1 + 1*0 + 3*0) = 3$$

The result of this calculation is placed in the corresponding area in the feature map.

Then we make another calculation by moving the filter on the image horizontally by one step to the right. The number of steps (pixels) that we shift the filter over the input image is called **Stride**. The shift can be done both horizontally and vertically. Here, we use `Stride=1`. Stride is also a hyperparameter that should be specified by the user.

The size of the feature map is small than the size of the input image. The size of the feature map also depends on the Stride. If we use `Stride=2`, the size will be further reduced. If there are several convolutional layers in the CNN, the size of the feature map will be further reduced at the end so that we cannot do other operations on the feature map. To avoid this, we use apply **Padding** to the input image. Padding is a hyperparameter that we need to configure in the convolutional layer. It adds additional pixels with zero values to each side of the image. That helps to get the feature map of the same size as the input.

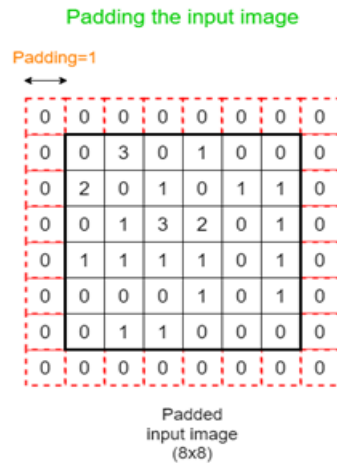


Fig. 4.5.3 Applying padding to an input image

4.5.2.2 Activation Layer

Activation functions are functions used in a neural network to compute the weighted sum of inputs and biases, which is in turn used to decide whether a neuron can be activated or not. They add non-linearities into neural networks, allowing them to learn powerful operations. Some of the common activation functions are:

- **Step function:** A simple function that outputs 0 for negative inputs and 1 for positive inputs.
- **Sigmoid function:** A smooth function that outputs a value between 0 and 1 for any input. It has a characteristic S-shape.
- **ReLU function:** A piecewise linear function that outputs 0 for negative inputs and the input itself for positive inputs. It is the most widely used activation function.
- **Leaky ReLU function:** A variation of ReLU that outputs a small linear component of the input for negative inputs instead of 0. It avoids the problem of zero gradient for negative inputs.

Here is a table that summarizes some of the properties of these activation functions:

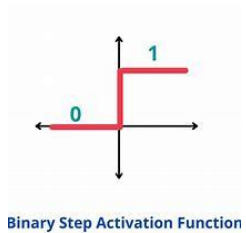
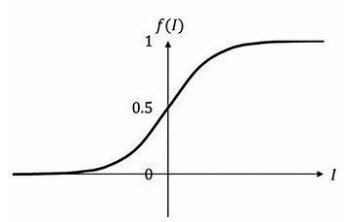
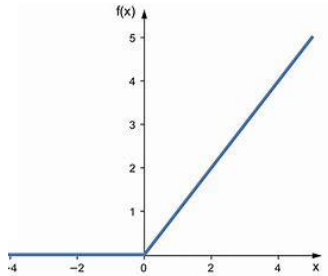
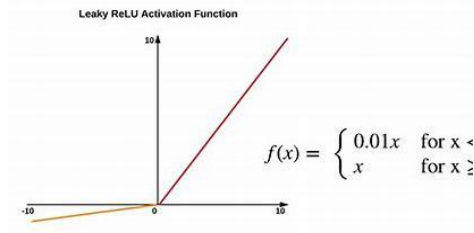
Function	Formula	Range	Graph
Step	$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$	$\{0, 1\}$	 Binary Step Activation Function
Sigmoid	$f(x) = \frac{1}{1 + e^x}$	$(0, 1)$	
ReLU	$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$	$[0, \infty)$	
Leaky ReLU	$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$ Where α is a small constant	$(-\infty, \infty)$	

Table 4.5.1 Different Activation Functions

In this project, we are using ReLU as our activation function.

ReLU Activation Function

ReLU function is given by

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

Eq. 4.5.1 ReLU Activation Function

The graphical representation of ReLU activation function is shown in fig. 4.5.4

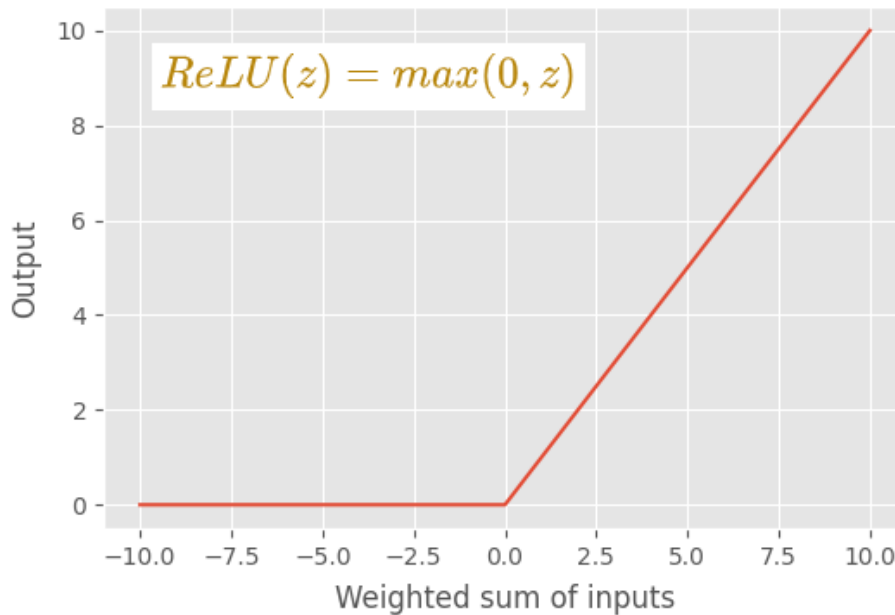


Fig. 4.5.4 ReLU Activation Function

- **Key Features**

- The ReLU (**R**ectified **L**inear **U**nit) activation function is a great alternative to both sigmoid and tanh activation functions.
- Inventing ReLU is one of the most important breakthroughs made in deep learning.

- This function does not have the vanishing gradient problem.
 - This function is computationally inexpensive. It is considered that the convergence of ReLU is 6 times faster than sigmoid and tanh functions.
 - If the input value is 0 or greater than 0, the ReLU function outputs the input as it is. If the input is less than 0, the ReLU function outputs the value 0.
 - The ReLU function is made up of two linear components. Because of that, the ReLU function is a piecewise linear function. In fact, the ReLU function is a non-linear function.
 - The output of the ReLU function can range from 0 to positive infinity.
 - The convergence is faster than sigmoid and tanh functions. This is because the ReLU function has a fixed derivate (slope) for one linear component and a zero derivative for the other linear component. Therefore, the learning process is much faster with the ReLU function.
 - Calculations can be performed much faster with ReLU because no exponential terms are included in the function.
- **Usage**
 - The ReLU function is the default activation function for hidden layers in modern MLP and CNN neural network models.
 - We do not usually use the ReLU function in the hidden layers of RNN models. Instead, we use the sigmoid or tanh function there.
 - We never use the ReLU function in the output layer.

- **Drawbacks**

- The main drawback of using the ReLU function is that it has a dying ReLU problem.
- The value of the positive side can go very high. That may lead to a computational issue during the training.

4.5.2.3 Pooling Layer and Pooling Operation

Pooling layers are the second type of layer used in a CNN. There can be multiple pooling layers in a CNN. Each convolutional layer is followed by a pooling layer. So, convolution and pooling layers are used together as pairs.

Objectives:

- Extract the most important (relevant) features by getting the maximum number or averaging the numbers.
- Reduce the dimensionality (number of pixels) of the output returned from previous convolutional layers.
- Reduce the number of parameters in the network.
- Remove any noise present in the features extracted by previous convolutional layers.
- Increase the accuracy of CNNs.

There are three elements in the pooling layer: **Feature map**, **Filter** and **Pooled feature map**. The **pooling operation** occurs in each pooling layer.

The pooling operation happens between a section of the feature map and the filter. It outputs the pooled feature map.

There are two types of pooling operations.

- **Max pooling:** Get the maximum value in the area where the filter is applied.
- **Average pooling:** Get the average of the values in the area where the filter is applied.

The following diagram shows the max pooling operation applied on the feature map obtained from the previous convolution operation.

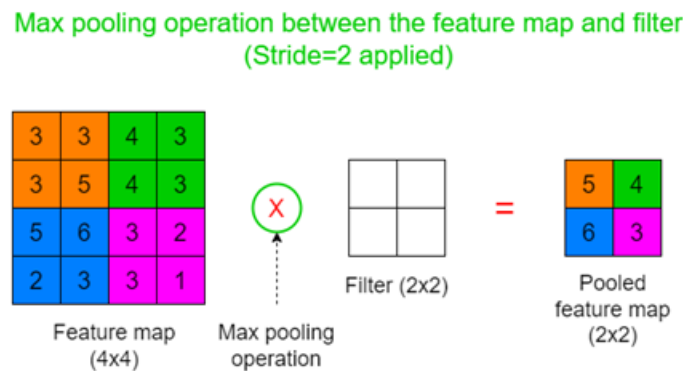


Fig. 4.5.5 Max Pooling

Filter: This time, the filter is just a window as there are no elements inside it. So, there are no parameters to learn in the pooling layer. The filter is just used to specify a section in the feature map. The size of the filter should be specified by the user as a hyperparameter. The size should be smaller than the size of the feature map. If the feature map has multiple channels, we should use a filter with the same number of channels. The pooling operations will be done on each channel independently.

Feature map section: The size of the feature map section should be equal to the size of the filter we choose. We can move the filter vertically and horizontally on the feature map to create different sections. The number of sections depends on the Stride we use.

Pooled feature map: The pooled feature map stores the outputs of different pooling operations between different feature map sections and the filter. This will be the input for the next convolution layer (if any) or for the flatten operation.

Stride and padding in the pooling operation

- **Stride:** Here, the Stride is usually equal to the size of the filter. If the filter size is (2x2), we use Stride=2.
- **Padding:** Padding is applied to the feature map to adjust the size of the pooled feature map.

Both Stride and Padding are hyperparameters that we need to specify in the pooling layer.

Note: After applying pooling to the feature map, the number of channels is not changed. It means that we have the same number of channels in the feature map and the pooled feature map. If the feature map has multiple channels, we should use a filter with the same number of channels. The pooling operations will be done on each channel independently.

4.5.2.4 Flattening Operation

In a CNN, the output returned from the final pooling layer (i.e. the final pooled feature map) is fed to a Multilayer Perceptron (MLP) that can classify the final pooled feature map into a class label.

An MLP only accepts one-dimensional data. So, we need to flatten the final pooled feature map into a single column that holds the input data for the MLP.

Unlike flattening the original image, important pixel dependencies are retained when pooled maps are flattened.

The following diagram shows how we can flatten a pooled feature map that contains only one channel.

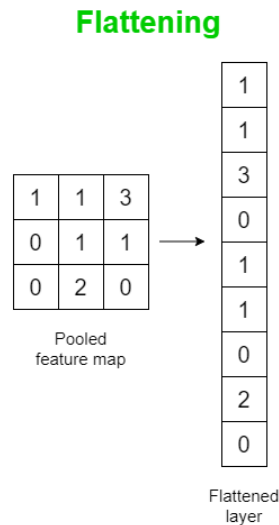


Fig. 4.5.6 Flattening Operation

4.5.2.5 Fully Connected Layers

These are the final layers in a CNN. The input is the previous flattened layer. There can be multiple fully connected layers. The final layer does the classification (or other relevant) task. An activation function is used in each fully connected layer.

Objectives:

- Classify the detected features in the image into a class label.

4.6 CONFIGURING THE CNN USING KERAS

4.6.1 Keras

Keras is a deep learning library for Python that provides a simple and flexible way to build and train neural networks. Keras is designed for human beings, not machines, and aims to reduce the cognitive load for developers and researchers. Keras can run on top of TensorFlow, Theano, or CNTK, and can also be deployed on CPUs or GPUs.

Keras offers a high-level API that abstracts away many low-level details of tensor manipulation and computation. Keras allows users to define their models using either a sequential or a functional style, and provides various layers, optimizers, loss functions, metrics, callbacks, and utilities to facilitate the model development process. Keras also supports multiple input and output modes, complex network architectures, and custom layers and models.

Keras is part of the TensorFlow ecosystem, which means it can leverage other TensorFlow tools and features, such as TensorBoard, TensorFlow Lite, TensorFlow Hub, TensorFlow Serving, etc. Keras also has a large and active community that contributes to its documentation, guides, tutorials, examples, blogs, books, courses, etc.

Keras is suitable for beginners who want to learn the basics of deep learning, as well as experts who want to experiment with new ideas and push the state-of-the-art in various domains. Keras can be used for various applications, such as computer vision, natural language processing, speech recognition, recommender systems, generative models, reinforcement learning, etc.

Some of the benefits of using Keras are:

- easy to use and learn
- consistent and reliable
- modular and extensible
- expressive and productive
- compatible and interoperable

Some of the challenges or limitations of using Keras are:

- It may not support some advanced or custom operations or layers
- It may not offer full control over low-level details or performance optimization
- It may have some compatibility issues with different backends or versions

4.6.2 Building the CNN using Keras

The sequential API in Keras is a way of creating neural network models that consist of a linear stack of layers. The sequential API is suitable for simple models that have a single input and a single output, and where each layer has exactly one input and one output.

To use the sequential API, you need to create an instance of the **Sequential** class and then add layers to it using the **add** method. You can also pass a list of layers to the Sequential constructor. You can specify the input shape of the first layer using the **input_shape** argument, or by using an **Input** layer. You can access the layers of the model using the **layers** attribute, and remove the last layer using the **pop** method.

The sequential API offers a simple and intuitive way to build models layer-by-layer, but it has some limitations. It does not allow you to create models that have multiple inputs or outputs, share layers, or have non-linear connections. For these cases, you need to use the functional API or the subclassing API in Keras.

The functional API and the subclassing API are more flexible and powerful ways of creating models that can handle complex architectures and custom logic. The functional API allows you to create models by connecting input and output tensors of layers, while the subclassing API allows you to create models by inheriting from the **Model** class and defining your own forward pass.

We are building a CNN model which consists of two convolution and pooling layers and two fully connected layers. The code snippet below shows the implementation of this model.

```
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from keras.layers.pooling import MaxPooling2D, AveragePooling2D
from keras.callbacks import EarlyStopping

def build_model():
    model = Sequential()
    model.add(Conv2D(filters = 16, kernel_size = (5, 5), padding = 'valid', activation = 'relu', input_shape = (128, 128, 3)))
    model.add(MaxPool2D(pool_size = (2, 2)))
    model.add(Conv2D(filters = 32, kernel_size = (5, 5), padding = 'valid', activation = 'relu', input_shape = (128, 128, 3)))
    model.add(MaxPool2D(pool_size = (2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(256, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation = 'softmax'))
    return model

model=build_model()
model.summary()
```

Here is the summary of our model

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 124, 124, 16)	1216
max_pooling2d (MaxPooling2D)	(None, 62, 62, 16)	0
conv2d_1 (Conv2D)	(None, 58, 58, 32)	12832
max_pooling2d_1 (MaxPooling2D)	(None, 29, 29, 32)	0
dropout (Dropout)	(None, 29, 29, 32)	0
flatten (Flatten)	(None, 26912)	0
dense (Dense)	(None, 256)	6889728
dropout_1 (Dropout)	(None, 256)	0

dense_1 (Dense) (None, 2) 514

```
=====
Total params: 6,904,290
Trainable params: 6,904,290
Non-trainable params: 0
```

4.6.3 Compiling the model

Compilation of a model includes defining the loss function, optimizer and metrics to evaluate model.

4.6.3.1 Loss Function

Loss function is used to find error or deviation in the learning process. Keras requires loss function during model compilation process.

Keras provides quite a few loss function in the **losses** module and they are as follows –

- mean_squared_error
- mean_absolute_error
- mean_absolute_percentage_error
- mean_squared_logarithmic_error
- squared_hinge
- hinge
- categorical_hinge
- logcosh
- huber_loss
- categorical_crossentropy
- sparse_categorical_crossentropy
- binary_crossentropy
- kullback_leibler_divergence
- poisson
- cosine_proximity
- is_categorical_crossentropy

In our model, we are using binary cross entropy as our loss function.

Binary Cross Entropy/ Log Loss

Binary cross entropy loss is a loss function that measures the difference between the predicted probabilities and the true labels in binary classification problems. It is commonly used in machine learning and deep learning algorithms to optimize the performance of the model.

A simple way to understand binary cross entropy loss is to think of it as a penalty for making wrong predictions. The more confident the model is about a wrong prediction, the higher the penalty. Conversely, the more confident the model is about a correct prediction, the lower the penalty.

Binary Cross Entropy is the negative average of the log of corrected predicted probabilities. It is given by

$$Log Loss = \frac{1}{N} \sum_{i=1}^N -(y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i))$$

Eq. 4.6.1 – Log Loss/ Binary Cross Entropy

Where

N = total no. of instances

y_i = predicted class

p_i = probability of predicted class

4.6.3.2 Optimizer

An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rates, to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function.

The goal of optimization is to find the best set of weights for the neural network that minimizes the loss function and maximizes the accuracy. However, this is not a trivial task, as the loss function can be complex and non-convex, meaning that it can have multiple local minima and maxima.

There are different types of optimizers that use different strategies and techniques to update the weights and navigate the loss surface. Some of the common optimizers are:

- **Gradient descent:** The most basic optimizer that updates the weights in the opposite direction of the gradient of the loss function with respect to the weights. It uses a fixed learning rate and takes the entire dataset into account for each update¹.
- **Stochastic gradient descent:** A variant of gradient descent that updates the weights using one sample or a small batch of samples at a time. It introduces randomness and noise in the updates, which can help escape local minima and find better solutions¹.
- **Momentum:** An extension of stochastic gradient descent that adds a momentum term to the update equation. The momentum term is a fraction of the previous update, which helps accelerate the updates in the direction of consistent gradients and dampen oscillations¹.
- **Adaptive learning rate:** A family of optimizers that adapt the learning rate for each weight based on the historical gradients. Some examples are Adagrad, Adadelata, RMSprop, and Adam. These optimizers can achieve faster convergence and avoid manual tuning of the learning rate.

In our model, we are using stochastic gradient descent as our optimizer.

4.6.3.3 Metrics

Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process. Keras provides quite a few metrics as a module, **metrics** and they are as follows

- accuracy
- binary_accuracy
- categorical_accuracy
- sparse_categorical_accuracy
- top_k_categorical_accuracy
- sparse_top_k_categorical_accuracy
- cosine_proximity

- `clone_metric`

Similar to loss function, metrics also accepts below two arguments –

- **y_true** – true labels as tensors
- **y_pred** – prediction with same shape as **y_true**

In our model, we are using accuracy as our metric to evaluate our model performance on testing data.

Accuracy

Accuracy is one of the simplest metrics available to us for classification models. It is the number of correct predictions as a percentage of the number of observations in the dataset. The score ranges from 0% to 100%, where 100% is a perfect score and 0% is the worst.

Accuracy can be calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Eq. 4.6.2 – Calculating accuracy

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

Accuracy is a good metric to use when the classes are balanced and the errors are equally important. However, accuracy can be misleading when the classes are imbalanced or the errors have different costs. For example, in a medical diagnosis problem, accuracy does not account for the difference between false negatives (missing a disease) and false positives (diagnosing a healthy person as sick).

In such cases, other metrics such as precision, recall, F1-score, or AUC-ROC curve may be more appropriate to evaluate the performance of the model.

4.6.3.4 Implementation

Here is the code to compile the model using Keras.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for compiling a Keras model.

```
from keras import losses
from keras import optimizers
from keras import metrics

model.compile(loss = 'binary_crossentropy', optimizer = 'sgd', metrics = ['accuracy'])
```

4.6.4 Training/ Fitting the model

Models are trained by NumPy arrays using *fit()*. The main purpose of this fit function is used to evaluate your model on training. This can be also used for graphing model performance. It has the following syntax –

```
model.fit(X, Y, epochs = , batch_size = )
```

Here,

- **X, Y** – It is a tuple to evaluate your data.
- **epochs** – no of times the model is needed to be evaluated during training.
- **batch_size** – training instances.

To avoid overfitting of the model, we are using **Early Stopping** technique. Here is the implementation to fit the model.


```
batch_size = 32
epochs = 10

# To find optimal number of epochs to train a model using "Early Stopping Technique"
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor="val_loss",
                                         mode="min", patience=3,
                                         restore_best_weights=True)

#Training or fitting the model
history = model.fit(
    X_train, Y_train,
    epochs=epochs,
    batch_size = batch_size,
    validation_data=(X_val, Y_val),
    callbacks=[earlystopping]
)
```

4.7 SAVING AND LOADING THE MODEL

We are going to save our model using Keras. Keras provides a basic legacy high-level save format using the HDF5 standard. It also provides `load_model()` method to recreate the exact same model, including its weights and the optimizer.

Here is the implementation to save and load our model.

```
model.save('model_a85.h5')

from tensorflow.keras.models import load_model
new_model = load_model("model_a85.h5")
```

4.8 ANALYZING THE ACCURACY AND LOSS OF THE MODEL

Now, we are going to analyse the accuracy of model over training and testing data using **matplotlib** library.

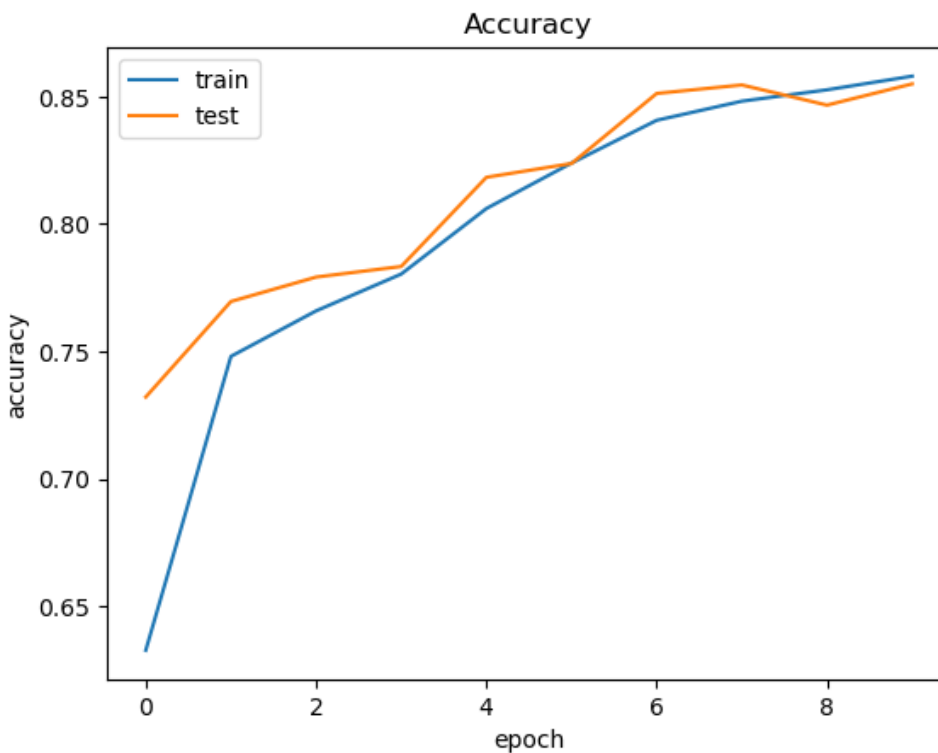


Fig. 4.8.1 – Epoch v/s Accuracy over training & testing sets

The fig. 4.8.1 depicts the improvement in the accuracy levels for each and every epoch of the model training over training and testing data fragments. As we can observe that the model gets more accurate by updating the mistakes done in previous epoch and thereby the accuracy is

increased in successive epochs. If the model is overfitted, then we can see a drastic decline in the accuracy level of the testing data.

We also analyse the loss value of our model over both training and testing data using matplotlib library.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Loss")
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'])
plt.show()
```

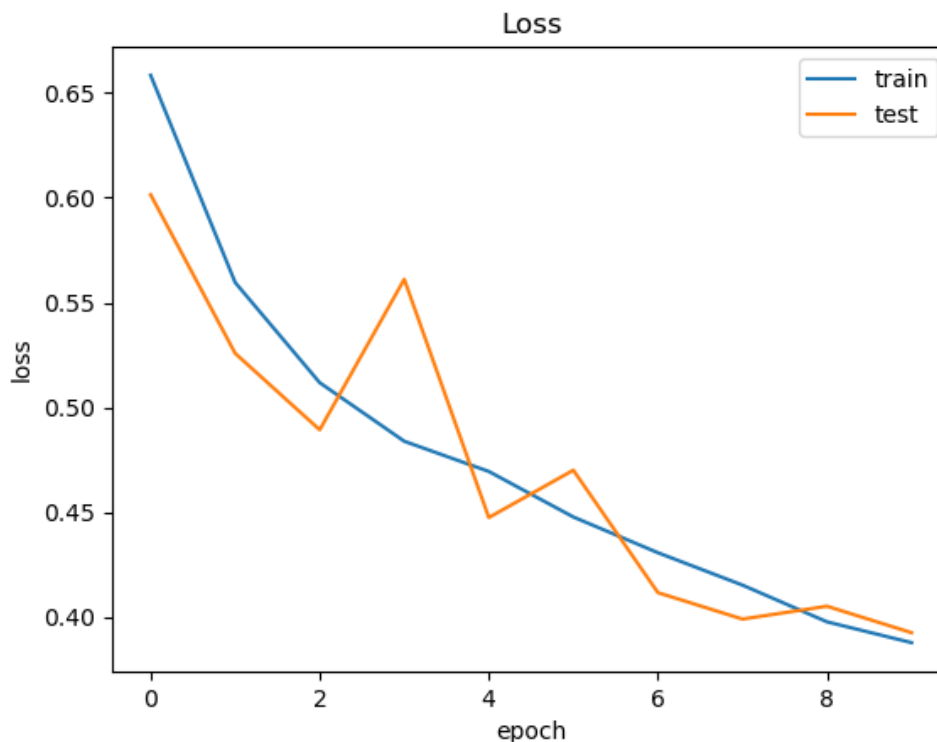


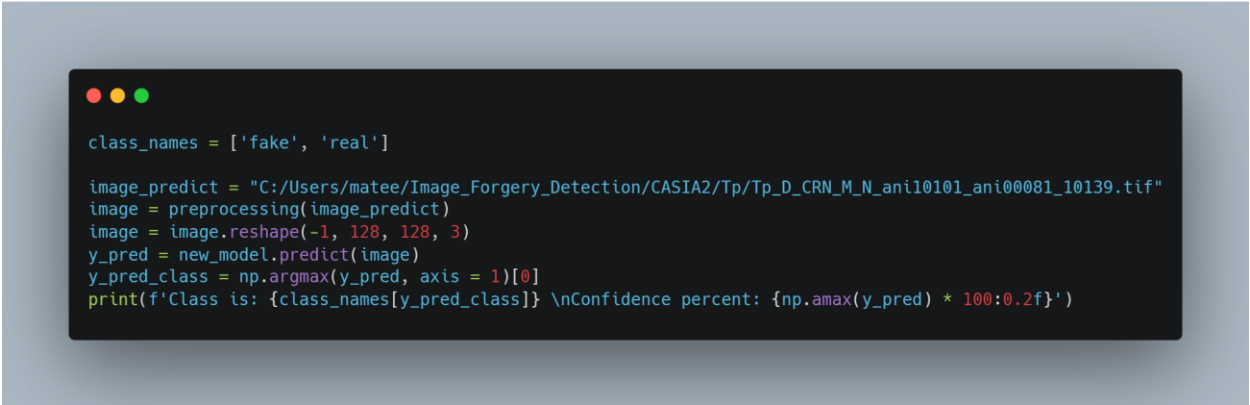
Fig. 4.8.2 – Epoch v/s Loss over training & testing sets

The fig. 4.8.2 describes the variation in the loss function value for each and every epoch over training and testing sets. As we can observe from the graph that the model backpropagates

and updates the model such that it reduces the loss function value for subsequent epochs of the model training.

4.9 PREDICTING THE OUTPUT

Now our model is going to predict whether an image is forged or not using the **predict** method.



```
class_names = ['fake', 'real']

image_predict = "C:/Users/matee/Image_Forgery_Detection/CASIA2/Tp/Tp_D-CRN_M_N-ani10101-ani00081_10139.tif"
image = preprocessing(image_predict)
image = image.reshape(-1, 128, 128, 3)
y_pred = new_model.predict(image)
y_pred_class = np.argmax(y_pred, axis = 1)[0]
print(f'Class is: {class_names[y_pred_class]} \nConfidence percent: {np.amax(y_pred) * 100:0.2f}')
```

Output:

Class is: fake

Confidence percent: 98.97

The model is predicting the tampered image as fake with the 98.97 % of confidence.

4.10 CONFUSION MATRIX

A confusion matrix is a table that summarizes the performance of a classification model on a set of test data. It shows how many predictions are correct and incorrect for each class, as well as the types of errors that the model makes.

A confusion matrix has the following format:

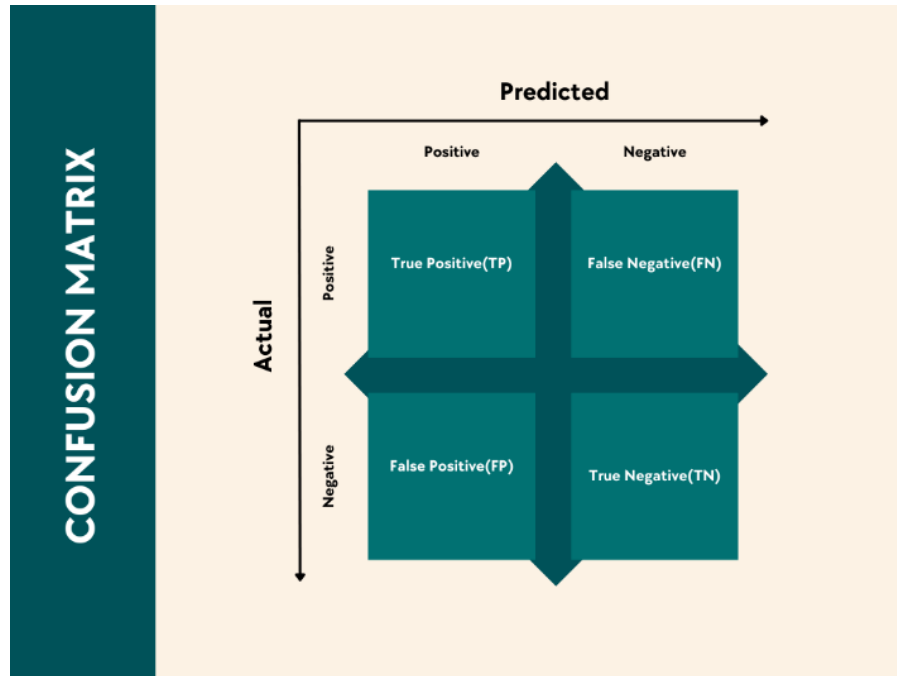


Fig. 4.10.1 – Confusion Matrix Format

Where:

- **True Positive (TP):** The model correctly predicts the positive class.
- **True Negative (TN):** The model correctly predicts the negative class.
- **False Positive (FP):** The model incorrectly predicts the positive class when the actual class is negative.
- **False Negative (FN):** The model incorrectly predicts the negative class when the actual class is positive.

A confusion matrix can be used to calculate various metrics that evaluate the performance of a classification model, such as accuracy, precision, recall, F1-score, and ROC-AUC curve.

A confusion matrix can also help to identify the classes that are most confused by the model, or the errors that are most costly for the problem domain.

Here is the implementation to plot the confusion matrix of our model using **matplotlib** library.

```
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

Y_pred = model.predict(X_val)
Y_pred_classes = np.argmax(Y_pred,axis = 1)
Y_true = np.argmax(Y_val,axis = 1)
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
plot_confusion_matrix(confusion_mtx, classes = range(2))
```

Output:

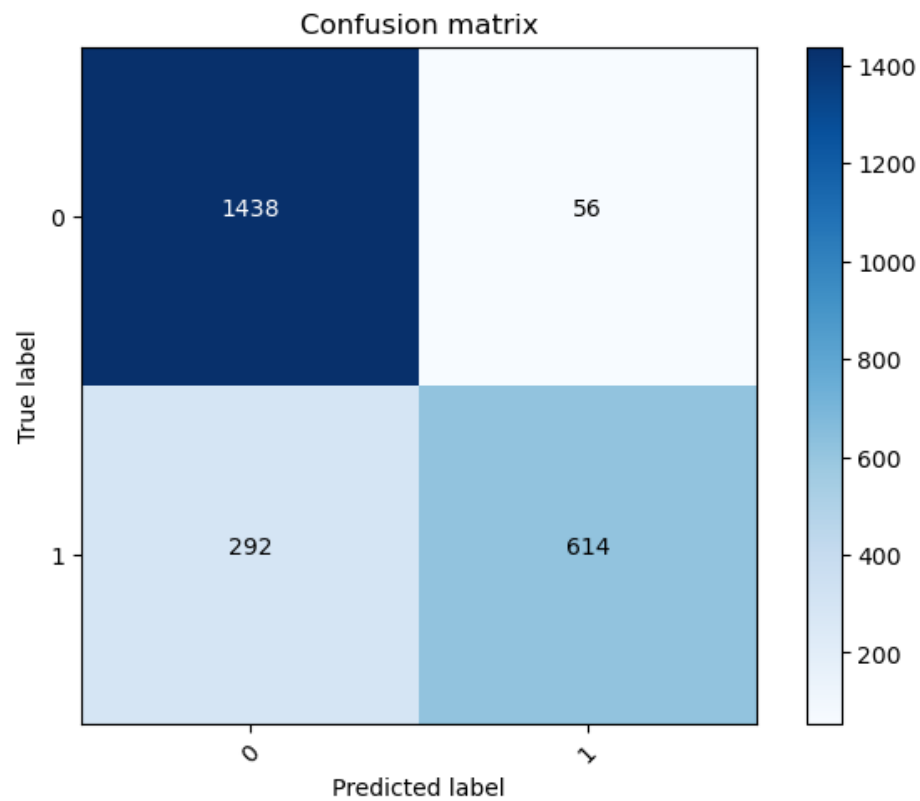


Fig. 4.10.2 – Confusion Matrix of the proposed model

4.11 FRONT END FOR THE PROJECT

Home.html

```

<!doctype html>

<html lang="en">

  <head>

    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztCQTWfspd3yD6SVohhpUuC0mLASjC" crossorigin="anonymous">

    <title>Home</title>

  </head>

  <body>

    <nav class="navbar navbar-dark bg-dark">
      <div class="container-fluid">
        <span class="navbar-brand mb-0 h1">Image Forgery Detection</span>
      </div>
    </nav>
    <br>

    <form action = "/" method = "post" enctype="multipart/form-data">
      <div>
        <label for="formFileLg" class="form-label">Please upload the image</label>
        <input type="file" name="file" class="form-control" autocomplete="off" required>
      </div>
      <br>
      <div class="col-12">
        <p align="center"> <button type="submit" class="btn btn-primary">Upload & Analyze image</button></p>
      </div>
      <div>
        <p>
          {% with messages = get_flashed_messages() %}
          {% if messages %}
            <ul>
              {% for message in messages %}
                <li>{{ message }}</li>
              {% endfor %}
            </ul>
          {% endif %}
          {% endif %}
          {% endwith %}
        </p>
      </div>
    </form>
    <br>
    <div>
      <h2 align="center">Image Analysis</h2><br>
    </div>
    <div display="inline-block">
      {% if result %}
        
        <img src = "{{ela_image}}" width="500" height="500">
        <h3>Output</h3>
        <p style="font-weight: 900;">The image seems to be {{result}}</p>
      {% endif %}
    </div>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-MrcW6ZMFYlzcL8nNl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM" crossorigin="anonymous">
    </script>

  </body>

</html>

```


4.12 BACKEND USING FLASK

Flask is a lightweight web framework for Python that allows you to create web applications quickly and easily. It is based on the Werkzeug WSGI toolkit and the Jinja2 template engine, which provide the core functionality and flexibility of Flask.

Flask is designed to be a microframework, meaning that it does not impose any dependencies or project structure on the developer. Instead, it offers suggestions and best practices, and leaves the choice of tools and libraries to the developer. This makes Flask suitable for a wide range of applications, from simple prototypes to complex web services.

Flask also supports modular development with blueprints, which are reusable components that can be registered with an application. Blueprints allow you to organize your code into logical units and avoid code duplication.

Flask has a large and active community that provides many extensions and resources to enhance its functionality. Some of the common extensions include Flask-SQLAlchemy for database integration, Flask-WTF for form validation, Flask-Login for user authentication, and Flask-RESTful for building RESTful APIs.

The backend implementation of flask is show below:

```

!pip install flask
!pip install pyngrok==4.1.1
!pip install flask_ngrok

from flask import Flask, flash, request, redirect, url_for, render_template
from werkzeug.utils import secure_filename
from flask_ngrok import run_with_ngrok
import urllib.request
import os

!ngrok authtoken 2HFX6zbW3w5q5cZv6703lG5u1HL_3fVS7PL6xxme3ECU4L2qa
app = Flask(__name__, template_folder="static/templates/")
UPLOAD_FOLDER = 'static/uploads'
app.secret_key = "secret key"
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024
run_with_ngrok(app)

ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg', 'gif', 'tif'])

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/', methods=['POST', 'GET'])
def upload_image():
    if 'file' not in request.files:
        flash('No file part')
        return redirect(request.url)
    file = request.files['file']
    if file.filename == '':
        flash('No image selected for uploading')
        return redirect(request.url)
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        #print('upload_image filename: ' + filename)
        flash('Image uploaded successfully')
        full_path = os.path.join("static/uploads/", filename)
        ela_image1 = ela_image(full_path)
        ela_path = os.path.join("static/ela", filename)
        ela_image1.save(ela_path)
        image = preprocessing(full_path)
        image = image.reshape(-1, 128, 128, 3)
        y_pred = new_model.predict(image)
        y_pred_class = np.argmax(y_pred, axis = 1)[0]
        print(class_names[y_pred_class])
        return render_template('home.html',
                               filename=filename, result=class_names[y_pred_class], image=full_path, ela_image = ela_path)
    else:
        flash('Allowed image types are - png, jpg, jpeg, gif, tif')
        return redirect(request.url)

app.run()

```

4.13 OUTPUT SCREENSHOTS

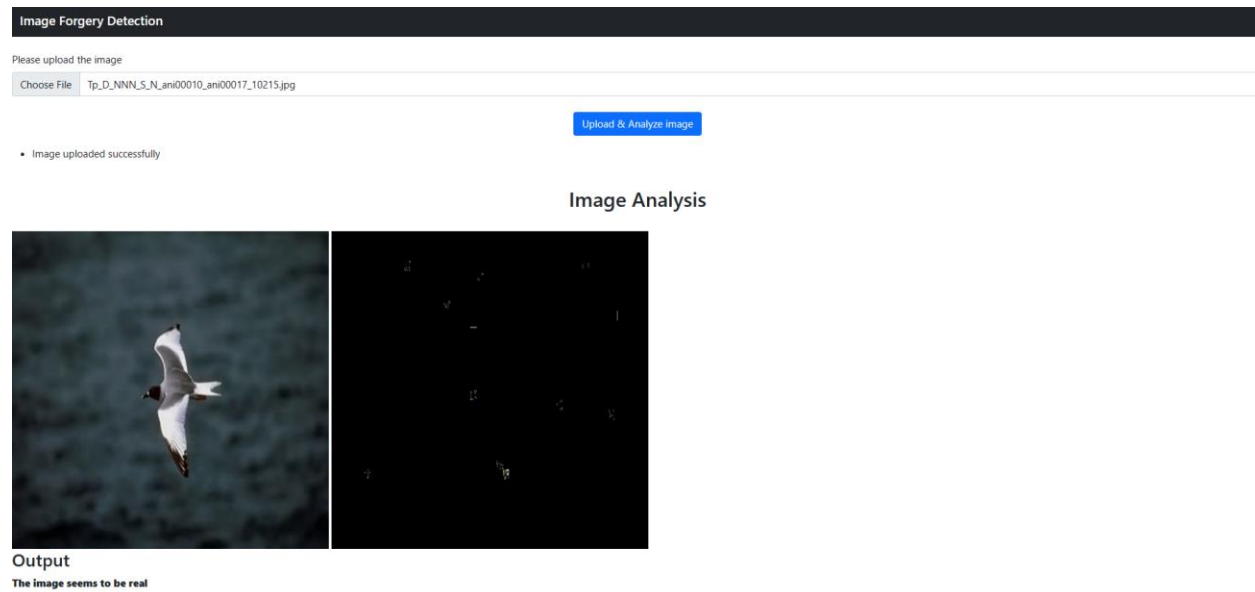


Fig. 4.13.1 – Model detecting image as real

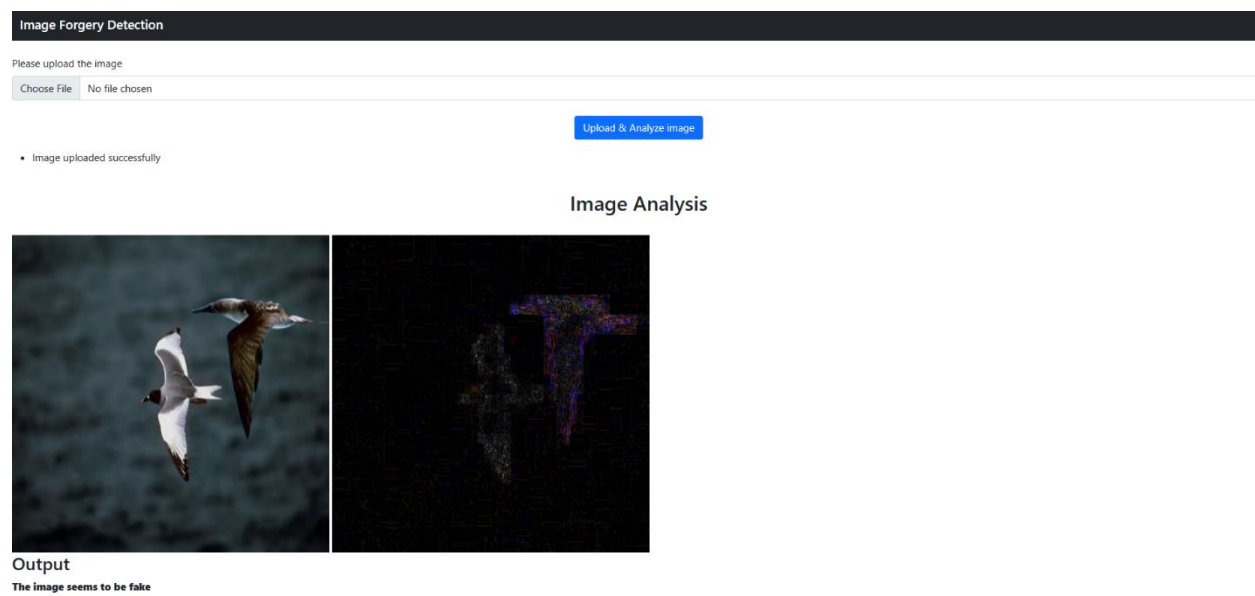


Fig. 4.13.2 – Model detecting image as fake

4.14 RESULTS

Our CNN model to detect image forgery consists of two convolution and pooling layers along with two fully connected layers. Our model has accuracy of 85.5% and loss of 39.24%.

4.14.1 Result Comparison with Existing System

The existing system has an accuracy level of 66.33% and loss of 62.17%. Due to the use of ELA technique, our model has increased the accuracy level to 85.5% and loss of 39.24%.

Here is the graph that represents the accuracy levels of existing and proposed systems using matplotlib library.

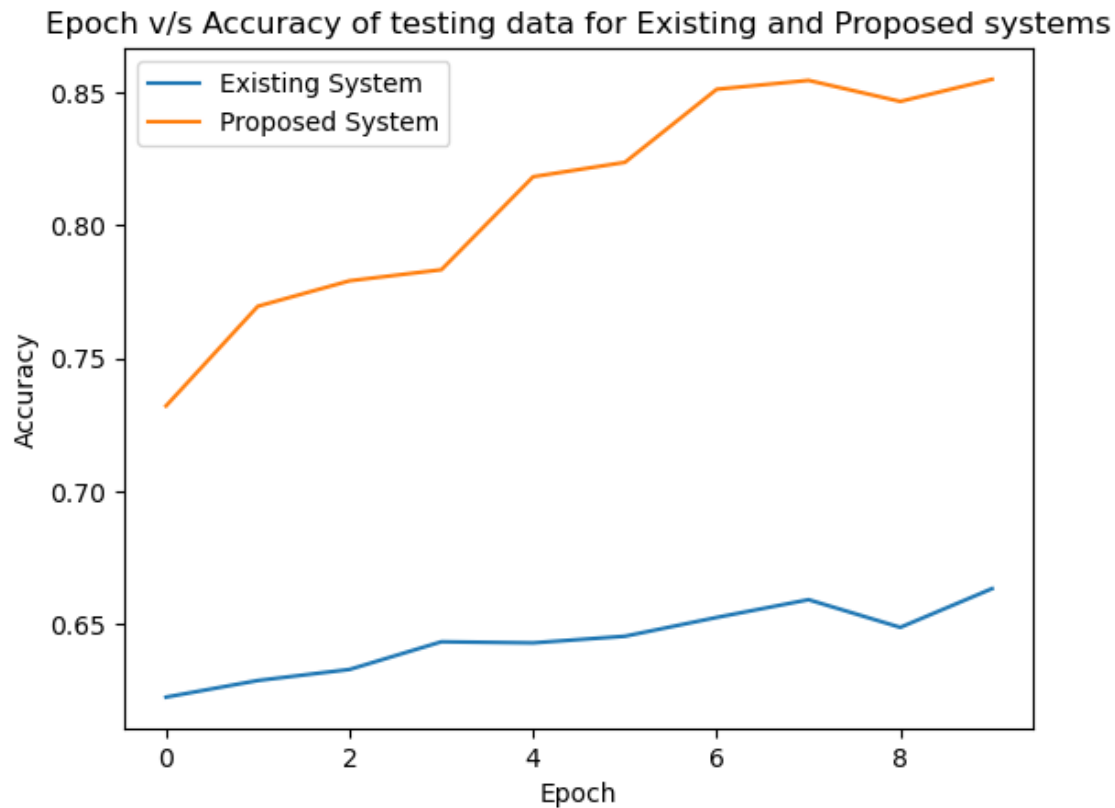


Fig. 4.14.1 – Epoch v/s Accuracy of Existing & Proposed Systems

The fig. 4.14.1 depicts the accuracy levels of existing and proposed system for each epoch. It is showing the clear accuracy improvement in the proposed system due to the use of ELA technique.

CONCLUSION

5. CONCLUSION

Image forgery detection helps to differentiate between the authentic and tampered images. It can be applied in various fields such as forensics to verify the authenticity and integrity of digital evidence, journalism to prevent the spread of fake news and misinformation, social media to protect users from cyberbullying, identity theft, or privacy invasion, by detecting and removing fake or harmful images that may target or impersonate someone and also in art and entertainment fields to preserve the originality and creativity of artists. It can also be used in the fake certificate detection by training the model over the samples of original and various fake certificates.

The image forgery detection model can be enhanced by combining other advanced forgery detection techniques. This project can also be extended to find the forgery in the video and audio data files. We conclude that deep neural networks approaches have their own shortcomings and need further improvement but still, they can perform better than the traditional approaches.

REFERENCES

REFERENCES

- Ritu Agarwal, Deepak Khudaniya, Abhinav Gupta, and Khyati Grover, “Image Forgery Detection and Deep Learning Techniques: A Review”, in 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS).

Available at: <https://ieeexplore.ieee.org/document/9121083>

- Arun Anoop M, “Image forgery and its detection: A survey”, in 2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS).

Available at: <https://ieeexplore.ieee.org/document/7193253>

- Zankhana J. Barad, and Mukesh M. Goswami, “Image Forgery Detection using Deep Learning: A Survey”, in 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS).

Available at: <https://ieeexplore.ieee.org/document/9074408>

- Luca Bondi, Silvia Lameri, David Güera, Paolo Bestagini, Edward J. Delp, and Stefano Tubaro, “Tampering Detection and Localization through Clustering of Camera-Based CNN Features”, in 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).

Available at: <https://ieeexplore.ieee.org/document/8014966>

- Jiansheng Chen, Xiangui Kang, Ye Liu, and Z. Jane Wang, “Median Filtering Forensics Based on Convolutional Neural Networks”, in IEEE Signal Processing Letters (Volume: 22, Issue: 11, November 2015)

Available at: <https://ieeexplore.ieee.org/document/7113799>

- Error Level Analysis (ELA)

<https://fotoforensics.com/tutorial-ela.php>

- ELA Photo Forensics

<https://eforensicsmag.com/ela-photo-forensics/#:~:text=ELA%20Error%20Level%20Analysis%20is%20a%20very%20useful,due%20their%20characteristic%20aspects%20in%20the%20ELA%20representati on.>

- Neural Networks and Deep Learning –

<https://rukshanpramoditha.medium.com/list/neural-networks-and-deep-learning-course-a2779b9c3f75>

- Analysis Techniques in Image Forgery Detection

https://mediaspace.wisc.edu/media/Analysis+Techniques+in+Image+Forgery+Detection/1_gxqs3qdu

- Python Imaging Library - <https://pillow.readthedocs.io/en/stable/>
- Textbook - Hands-On Machine Learning with Scikit-Learn and TensorFlow by Aurélien Géron
- Scikit Learn - <https://scikit-learn.org/stable>
- ngrok documentation - <https://ngrok.com/docs>
- Flask Web Development Tutorial by Edureka
https://www.youtube.com/watch?v=lj4I_CvBnt0