A Project Report on

# FAKE CURRENCY DETECTION

Submitted in partial fulfillment for the award of

**Bachelor of Technology**

in

**Computer Science and Engineering**

By

**G. Charan Sai (Y19ACS427)** **T. Sri Dharani (Y19ACS572)**

**T. Lakshmi Sathwika (L20ACS596)** **D. Baji (Y19ACS442)**

Under the guidance of

**Dr. Nagalla Sudhakar, Professor**

Department of Computer Science and Engineering

**Bapatla Engineering College**

(Autonomous)

(Affiliated to Acharya Nagarjuna University)

**BAPATLA – 522 102, Andhra Pradesh, INDIA**

**2022-2023**

## Department of Computer Science and Engineering

## CERTIFICATE

This is to certify that the project report entitled **FAKE CURRENCY DETECTION** that is being submitted by **G. Charan Sai (Y19ACS427), T. Sri Dharani (Y19ACS572), T. Lakshmi Sathwika (L20ACS596), D. Baji (Y19ACS442)** in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Acharya Nagarjuna University is a record of bonafide work carried out by them under our guidance and supervision.

Date:

| | |
|---|---|
| **Dr. Nagalla Sudhakar** | **Dr. P. Pardhasaradhi** |
| **Professor** | **Professor and HoD** |

## Declaration

We declare that this project work is composed by ourselves, and that the work contained herein is our own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

**G. Charan Sai (Y19ACS427)**
**T. Sri Dharani (Y19ACS572)**
**T. Lakshmi Sathwika (L20ACS596)**
**D. Baji (Y19ACS442)**

# Acknowledgement

We sincerely thank the following distinguished personalities who have given their advice and support for successful completion of the work.

We are deeply indebted to our most respected guide **Dr. Nagalla Sudhakar**, Professor, Department of CSE, for his valuable and inspiring guidance, comments, suggestions and encouragement.

We extend our sincere thanks to **Dr. P. Pardhasaradhi**, Prof. and Head of the Dept. for extending his cooperation and providing the required resources.

We would like to thank our beloved Principal, **Dr. Nazeer Shaik,** for providing the online resources and other facilities to carry out this work.

We would like to express our sincere thanks to the project coordination committee and our project coordinator **Dr. Nagalla Sudhakar,** Prof., Dept. of CSE for their helpful suggestions throughout the project work and in presenting this document.

We extend our sincere thanks to all other teaching faculty and non-teaching staff of the department, who helped directly or indirectly for their cooperation and encouragement.

**G. Charan Sai (Y19ACS427)**
**T. Sri Dharani (Y19ACS572)**
**T. Lakshmi Sathwika (L20ACS596)**
**D. Baji (Y19ACS442)**

# Abstract

Fake currency is the money produced without the approval of the government, creation of it is considered as a great offence. The elevation of color printing technology has increased the rate of fake currency note printing on a very large scale. Years before, the printing could be done in a print house, but now anyone can print a currency note with maximum accuracy using a simple laser printer. It is the biggest problem faced by many countries including India.

Though Banks and other large organizations have installed Automatic machines to detect fake currency notes, it is difficult for an average person to distinguish between the two. As a result, the issue of fake notes instead of the genuine ones has been increased very largely. India has been unfortunately cursed with the problems like corruption and black money and counterfeiting of currency notes is also a big problem to it. This leads to design of a system that detects the fake currency note in a less time and in a more efficient manner.

The proposed system gives an approach to verify the Indian currency notes. Verification of currency note is done by the concepts of Deep Learning. This term paper describes detection of fake currency by extracting features of Indian currency notes with MobileNetV2 model with help of Convolutional Neural Network model. The proposed system has got advantages like simplicity for larger dataset and high-performance speed. The result will predict whether the currency note is fake or not.

# Table of Contents

# List of Figures

# 1 Introduction

Currency started serving as a medium for exchanging goods and services thousands of years ago to replace the ancient barter system where any objects could be swapped if two traders agreed. Even nowadays, currency, as a measurement unit in pricing a transaction, still plays an indispensable role in modern society. The monetary form has been extended to cash including coins and banknotes, cashless money like bank cheques, and even electronic data representing currency in bank accounts.

With little revelation of the techniques to produce banknotes, it is impossible to produce counterfeit notes. Nevertheless, there is an enormous amount of research starting to reveal the inside story of the banknote, especially in the field of banknote recognition. Currency notes contains the features that could be detected by security analysis, such as magnetic ink, screen traps, manufacture anomalies, materials interaction, intricate patterns, intricate design, or fluorescence eminence. Nowadays, due to rapid elevation in technology, various ways were used to develop fake currency which couldn’t be detected by security analysis.

## 1.1 Fundamentals

Traditionally, machine learning has been studied either in a supervised paradigm like classification and regression, or in an unsupervised paradigm such as clustering and outlier detection. Supervised learning presumes that the training set has been provided, composed of a set of examples that have been appropriately labelled with the correct output. Based on the training set, a supervised learning method generates a model seeking to meet the two targets which are performing as well as possible on the set of training data and generalizing as well as possible to new data.

On the contrary, in unsupervised learning, the correct output of training data is not provided, or there are no training data at all to speak of. Instead, the algorithm attempts to identify the similarities between the inputs, so that the inputs which have something in common are categorized together. Banknote recognition is a typical case of pattern recognition.

The variation in patterns within a category is partly caused by environmental noises and the sensors, such as the effects of stain, and the quality of paper. The distortion resulting from the random nature of the pattern itself, can be effectively controlled by feature extraction and selection. But pattern recognition techniques were proved to be less accurate.

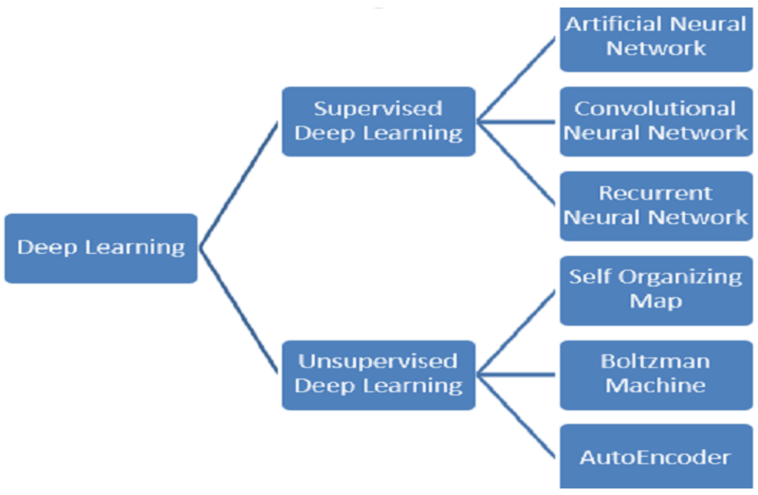


**Figure 1.1 Deep Learning Techniques**

Nowadays, Bank note recognition is regarded as Computer vision problem and solved using Deep Learning Techniques as shown in **Figure 1.1** Deep Neural Networks can be employed as efficient feature extractors and perform detection of counterfeit notes. Over traditional algorithms Deep neural networks have several advantages such as amount of pre-processing required, faster computation, efficiency, and robustness.

## 1.2 Objective

The main objective of the project is to identify the fake currency based on the features present on the currency notes by performing Deep Learning techniques. It also emphasizes on providing cheaper and accurate system to the user, which can be easily accessible and give accurate results and make it available for common people quicky and easily with low cost.

## 1.3 Overview

The proposed method comprises of three modules. For each module different dataset was considered which were created with the help of data augmentation. The images from these data sets were passed to a model which was built with the help of MobileNetV2 pretrained model.

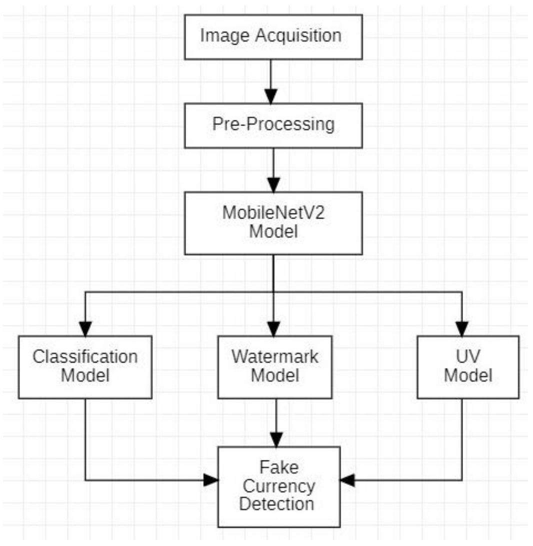


**Figure 1.2 Process Flow**

To differentiate fake and real notes two security features on currency note were considered Classification model is used to classify the currency notes. Watermark model is used to determine whether the currency note contains Gandhiji watermark. UV model was used to check whether the currency note contains security thread or not. The Process flow of our proposed model is shown in **Figure 1.2.**

# 2 Literature Survey

Counterfeiting of money is not a new problem and has been present since the coinage of money was started by the Greek in around 600 B.C. History tells us that counterfeiting of money has been an old evil. In modern times the problem still prevails and hence the use of different types of printing techniques and inclusion of different types of features in currencies has been happening, aiming to provide an easier way to detect forgeries. But with the advancement of technology and the growth of science new ways to detect counterfeit money are coming up that make this task quite easier with a fair amount of accuracy.

This chapter contains the list of literature review of the previous research where it is considered vital in the development of this project. This chapter provides in detail about an analysis, an overview of research reports, relevant articles, thesis that it's topic or issue is related to this project.

## 2.1 Fake Currency Detection using Python and Web Framework

Currency duplication also known as counterfeit currency is a vulnerable threat on economy. So automatic identification of currencies using image processing technique is proposed. The system designed to check the Indian currency note with denominations 10, 20, 50, 100, 200, 500 and 2000. It will pre-process the digital pictures and organize the prepared arrangement of information and it will distinguish in monetary forms. This paper proposes a convenient and cheapest method for identifying Indian currencies. The main limitation of this work is usage of flask framework which is highly complex. [1]

## 2.2 Identification of fake notes and denomination recognition

With the development of sophisticated printing techniques, counterfeit currency has become a significant concern. Some of the consequence of counterfeit notes on society are a reduction in the value of real money, increase in prices due to more money being circulated in the economy and decrease in acceptability of money. To prevent circulation of counterfeit notes, a system to detect fake notes must be developed. Notes with the legal sanction of the government possess certain security features such as intaglio printing, fluorescence, and watermark. The drawback of this method is it takes more cost and complex mechanism. [2]

## 2.3 Indian Currency Recognition and Verification using Image Processing

In this paper MATLAB was used which involves extraction of invisible and visible features of Indian currency notes. The image of the currency note is captured through a digital camera. Processing on the image is done on that acquired image using concepts like image segmentation, edge information of image and characteristics feature extraction. The main drawback of this work is MATLAB takes more time execute than other languages. [3]

## 2.4 Currency Counting Fake Note Detection

The proposed system is implemented the fake note detection unit with MATLAB algorithm. This paper is based on the image processing and UV LED's to give solutions for fake currency problems. The limitation of this project is MATLAB takes more time to execute and it is not open source and expensive. [4]

# 3 Problem Statement

Generally, there are two types of devices to detect counterfeit currency, one is Viewer type and other one is Automated type. Automatic machines that can detect banknotes are widely used in automatic dispensers of a range of products, from beverages to tickets, as well as in many automatic banking operations. In this document, banknotes are taken into consideration. After reviewing the literature about counterfeit detection, it appears that there were several drawbacks in the electronic devices.

Earlier fake currency notes were produced by color scanning and high-resolution printing. Today bank notes contain several security features such as intaglio printing, optically variable ink etc. Counterfeiters even trying to duplicate these security features. So, the electronic devices were lacking authenticity in detecting fake currency. Thus, the main purpose of the proposed system is to seek out a solution for detection of fake currency.

In the proposed system three types of datasets were considered, and two security features were considered for determining the authenticity of currency note. Upon training the model, it will be ready for future predictions. For the convenience of users, we were primarily obligated to design a console that can take input from the user and can display its prediction right next to the input screen. Needless to mention, this will rule out the hesitation associated with variations in handwriting. The proposed work is developed using deep learning algorithm Convolutional Neural Networks. Deep learning algorithms are efficient and give better accurate results for the prediction problems.

# 4 System Analysis

The bogus denominations in the market and illiteracy among the people in a country are causing money recognition problems in today's world, an automatic currency recognition system is essential. The two most critical criteria in such a system are accuracy and speed. By examining the significant qualities, a currency recognizer recognizes the currency and determines the denomination. There were numerous ways presented by researchers. Physical attributes (width, length) are used by some, while internal properties are used by only a few (texture, color).

## 4.1 Existing System

There was an existing system that used algorithms like K-Nearest Neighbors, Support Vector Classifier, Gradient Boost Classifier. The system used various high-quality images of currency notes and extracted features such as: Variance (change in model when using different portions of data.), Skewness (measure of asymmetry of distribution.), Kurtosis (degree of presence of outliers.). The model is preprocessed using data set and required classifiers were imported from the sklearn library and a function is developed that took parameters as the training and testing data sets along with the algorithm. [5]

### 4.1.1 Limitation

In existing system, some drawbacks were observed which hinders the performance of the system. They are Motion blur problems, Noise imposed by image capturing object, Uses less efficient feature extraction technique. In this existing architecture, only the front part of the note is taken into consideration and not the rear part. Another main drawback is the model developed only give accurate results for smaller datasets.

## 4.2 Proposed System

These days, technology advances at a breakneck pace. As a result, the banking sector is becoming more up to date by the day. This necessitates the use of automatic currency recognition. Many academics have been urged to create a robust and effective automatic cash identification machine. Cash recognition technology primarily tries to identify and extract visible and unseen properties of currency notes. To date, several methods for identifying the currency note have been proposed. However, the most effective method is to make use of the currency's visual properties.

The proposed system contains the advantages of the existing system and eliminates the disadvantages of it. The project centers on the design and implementation of Fake Currency Detection. The scope of the project is to provide approaches and strategies, which have proved to be suitable when accessing the image of the desired currency note. The scope of the project is limited to two security features of currency notes.

### 4.2.1 Security Features

**Watermark:** The Mahatma Gandhi Series of banknotes contain the Mahatma Gandhi watermark with a light and shade effect and multi-directional lines in the watermark window. [6] The watermark is shown in **Figure 4.1**

**Security Thread**: When held against the light, the security thread on currency note can be seen as one continuous line. The security thread appears to the left of the Mahatma's portrait. The security thread on currency note is shown in **Figure 4.2**

These security features are considered individually to determine the authenticity of the currency notes.

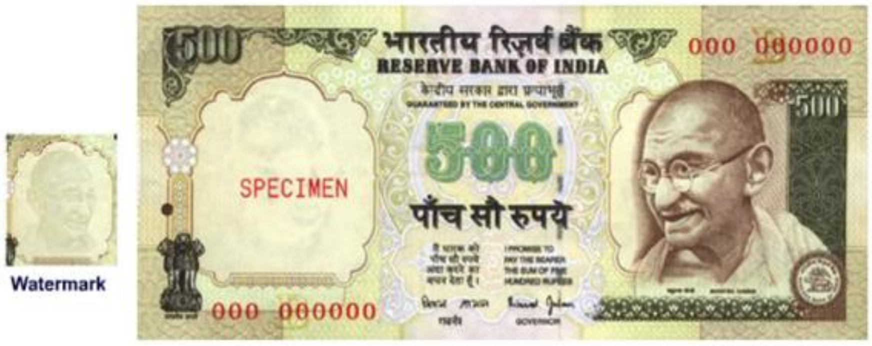


Figure 4.1 Gandhi Watermark

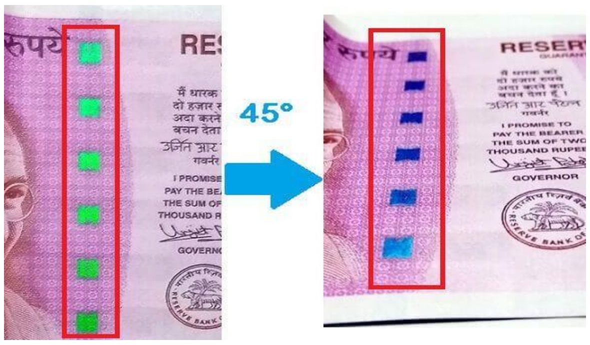


Figure 4.2 Security Thread

### 4.2.2 System Modules

The System uses **Convolutional Neural Network** as algorithm and **MobileNetV2** image classification model to determine whether an Indian currency note is fake or real. The proposed system contains 3 modules:

**Currency Classification Model**: Identifies the denomination, type (old or new), face orientation (front or back), and alignment (up or down) from a front light image of the note.

**Watermark Identification Model**: Identifies whether the note contains the Gandhi watermark under backlight illumination from a backlight image of the note.

**Ultraviolet Strip Detection Model**: Identifies whether the note contains a fluorescent strip or not and if yes then whether the strip is continuous or dashed under Ultraviolet illumination from an ultraviolet image of the note.

# 5 System Requirements and Specification

This chapter contains brief description about the software and hardware requirements and specifications in the project. It also consists of the information about the libraries used in the proposed work.

## 5.1 System Requirement Specification

System Requirement Specification is a fundamental document, which forms the foundation of the software development process. It not only lists the requirements of a system but also has a description of its major feature. An SRS is basically an organization's understanding of a customer or potential client's system requirements and dependencies at a particular point in time prior to any actual design or development work. It's a two- way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it. It is important to note that an SRS contains functional and non-functional requirements only.

### 5.1.1 Hardware Specifications

1. Processor: Either Intel or Ryzen
2. RAM: 4GB or above.

### 5.1.2 Software Requirements

1. Operating System: Windows 8 or above
2. Coding Language: Python
3. Platform: Jupyter notebook

### 5.1.3 Software Libraries

**Matplotlib**: Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open-source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications. A Python matplotlib script is structured so that a few lines of code are all that is required in most instances to generate a visual data plot. The matplotlib scripting layer overlays two APIs: The pyplot API is a hierarchy of Python code objects topped by matplotlib.pyplot. An OO (Object-Oriented) API collection of objects that can be assembled with greater flexibility than pyplot. This API provides direct access to Matplotlib's backend layers.

**NumPy**: NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

**OpenCV**: OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e., whatever operations one can do in Numpy can be combined with OpenCV.

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e., whatever operations one can do in Numpy can be combined with OpenCV.

**Keras**: Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Keras is simple but not simplistic. Keras reduces developer cognitive load to free you to focus on the parts of the problem that really matter. Keras adopts the principle of progressive disclosure of complexity i.e., simple workflows should be quick and easy, while arbitrarily advanced workflows should be possible. Keras provides industry-strength performance and scalability. Keras is the high-level API of Tensor Flow - an approachable, highly productive interface for solving machine learning problems, with a focus on modern deep learning.

**TensorFlow**: TensorFlow is an end-to-end, open-source machine learning platform. You can think of it as an infrastructure layer for differentiable programming. It combines four key abilities - Efficiently executing low-level tensor operations on CPU, GPU, or TPU. Computing the gradient of arbitrary differentiable expressions. Scaling computation to many devices, such as clusters of hundreds of GPUs. Exporting programs ("graphs") to external runtimes such as servers, browsers, mobile and embedded devices.

## 5.2 Functional Requirements

The System must be able to extract the features of the image captured by camera. User should be able to upload the image to the machine. System must be designed in such a way it can differentiate real image or fake image notes. System should process the input given by user i.e., path of the uploaded image.

## 5.3 Non-Functional Requirements

Non-functional requirements are the requirements which are not directly concerned with the specific function delivered by the system. They specify the criteria that can be used to judge the operation of a system rather than specific behaviours. They may relate to emergent system properties such as reliability, response time and store occupancy. Non-functional requirements arise through the user needs, because of budget constraints, organizational policies, and the need for interoperability with other software and hardware systems or because of external factors such as:

**Usability**

The system must be easy to learn for users.

**Reliability**

The reliability of the device essentially depends on the software tools (OpenCV, NumPy, TensorFlow etc.) and hardware tools (camera, computer etc.) used for the system development.

**Performance**

Currency notes from the input should be recognized with an accuracy of about 80% and more.

**Functionality**

This software must be capable of delivering functional requirements mentioned in this document.

# 6 System Design

The main objective of the design chapter is to explore the logical view of system architecture, Activity diagram, Use case diagram of the software for performing the operations such as pre-processing, extracting features, and detecting the counterfeit currency.

## 6.1 Block Diagram of System

The entire system can be divided into three modules and the block diagram represents control flow of the system as shown in **Figure 6.1.** With the help of data augmentation generated datasets were used for each module. Each module is compiled individually, and Counterfeit detection model is developed to detect fake currency notes.

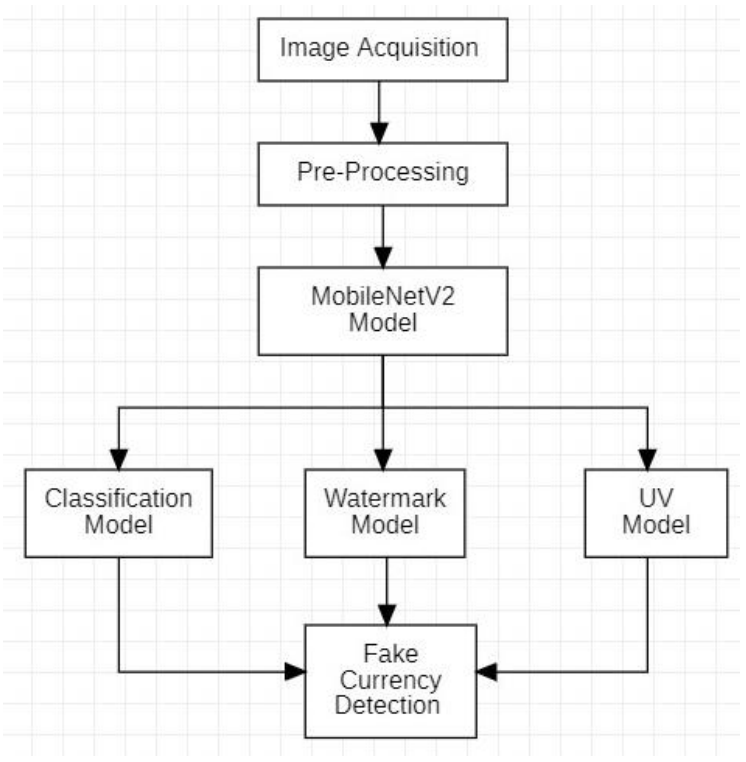


**Figure 6.1 Block Diagram of System**

## 6.2 Activity Diagram

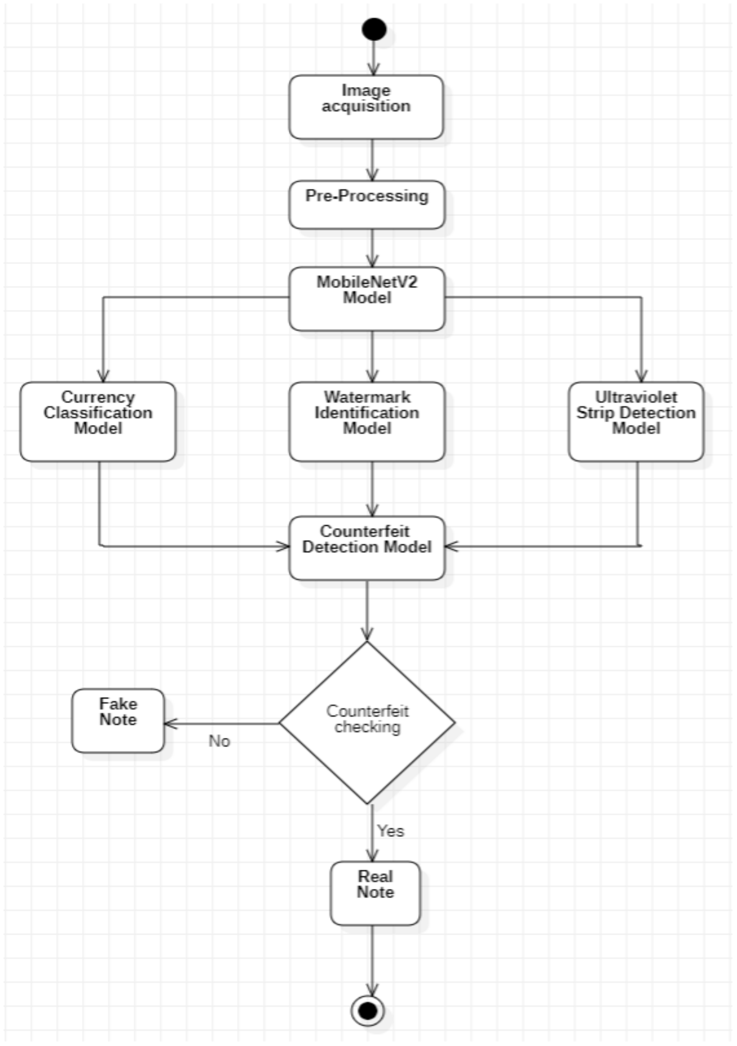


Image acquisition
Pre-Processing
MobileNetV2 Model
Currency Classification Model
Watermark Identification Model
Ultraviolet Strip Detection Model
Counterfeit Detection Model
Counterfeit checking
No
Fake Note
Yes
Real Note

**Figure 6.2 Activity diagram**

We use Activity Diagrams to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram. Activity diagram of our system can be shown in **Figure 6.2**. An activity diagram is a behavioural diagram i.e., it depicts the behaviour of a system. An activity diagram portrays the control flow from a start point to a finish point.

## 6.3 Use case Diagram

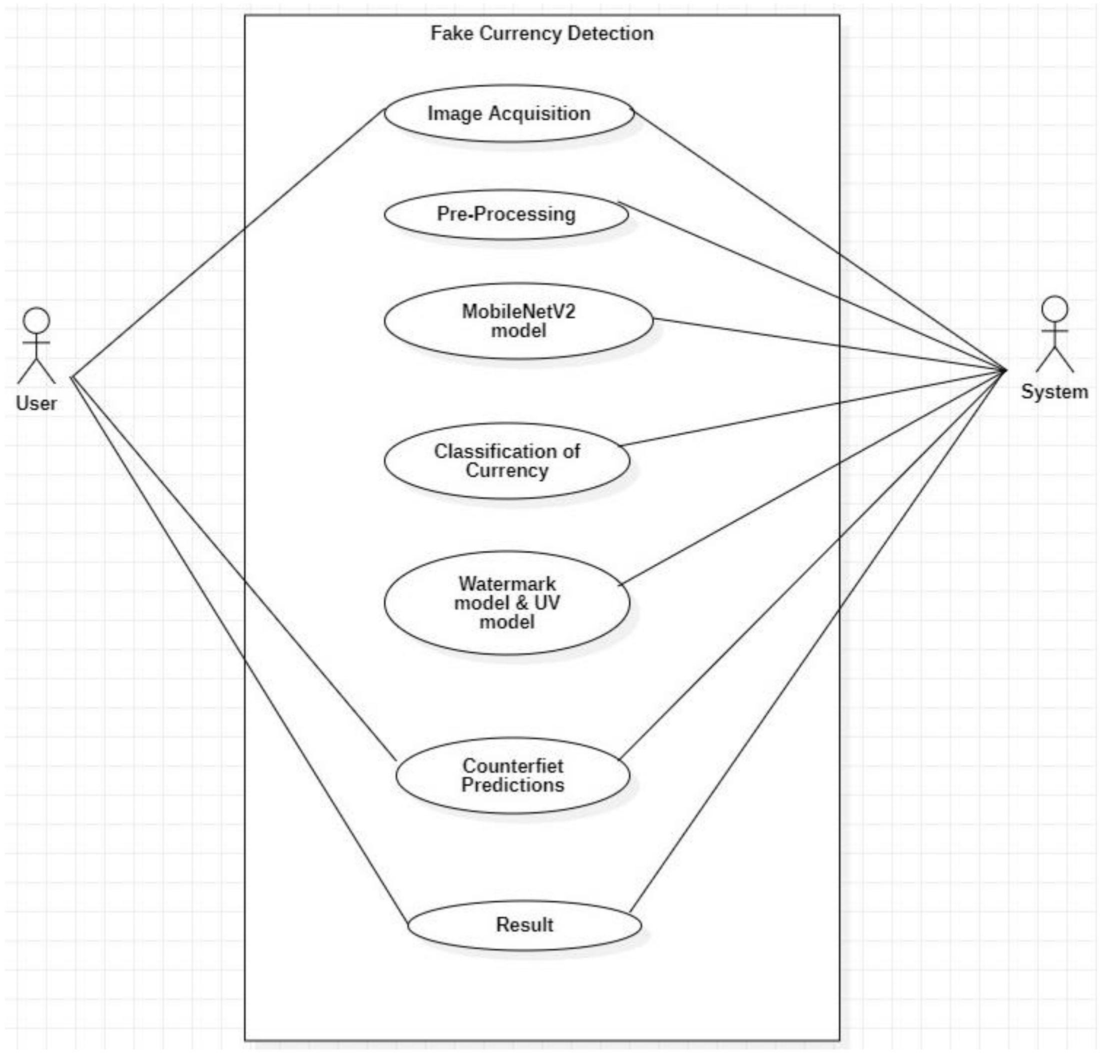


**Figure 6.3 Use Case Diagram**

Use case consists of user and system where user is used to provide the input to the system and system is used to process the input data and provide output. The flow is shown in the **Figure 6.3** First user as to run the system and run the code, model and library packages are imported and loaded. Then user runs all the three models to obtain the individual prediction. After the running the code user must upload the image of currency note. The model takes the path of image as input give prediction as output. With the individual model user can determine whether the currency note is fake or not.

# 7 Methodology

Methodology chapter describes the algorithms and architectures that were used in the proposed model. This chapter mainly focuses on the methodology that required to develop a model to detect fake currency.

## 7.1 Algorithm

The proposed model uses algorithms to determine the authenticity of the currency notes. Animals recognize various objects and make sense out of large amount of visual information, apparently requiring very little effort. Simulating the task performed by animals to recognize to the extent allowed by physical limitations will be enormously profitable for the system. This necessitates study and simulation of Neural Networks.

### 7.1.1 Artificial Neural Network

In Artificial Neural Network, each node performs some simple computation, and each connection conveys a signal from one node to another labeled by a number called the "connection strength" or weight indicating the extent to which signal is amplified or diminished by the connection.

Different choices for weight results in different functions are being evaluated by the network. If in each network whose weight are initial random and given that we know the task to be accomplished by the network, a learning algorithm must be used to determine the values of the weight that will achieve the desired task. Learning Algorithm qualifies the computing system to be called Artificial Neural Network. It is shown in **Figure 7.1**

The node function was predetermined to apply specific function on inputs imposing a fundamental limitation on the capabilities of the network. Typical pattern recognition systems are designed using two passes. The first pass is a feature extractor that finds features within the data which are specific to the task being solved (e.g., finding bars of pixels within an image for character recognition). The second pass is the classifier, which is more general purpose and can be trained using a neural network and sample data sets. Clearly, the feature extractor typically requires the most design effort, since it usually must be hand-crafted based on what the application is trying to achieve.

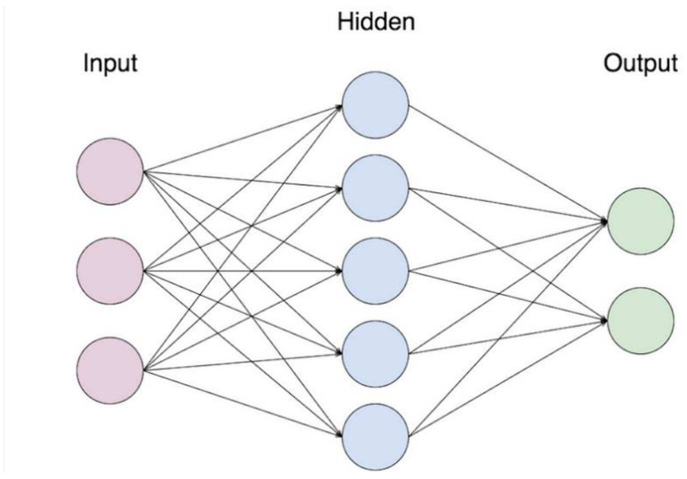


**Figure 7.1 Artificial Neural Network**

Back propagation was created by generalizing the Widrow-Hoff learning rule to multiple-layer networks and nonlinear differentiable transfer functions. Input vectors and the corresponding target vectors are used to train a network until it can approximate a function, associate input vectors with specific output vectors, or classify input vectors in an appropriate way as defined by you. Networks with biases, a sigmoid layer, and a linear output layer can approximate any function with a finite number of discontinuities. [7]

In deep learning, a convolution neural network (CNN, or ConvNet) is a class of deep neural networks, most applied to analysing visual imagery. Convolutional Neural Networks use a variation of multilayer perceptron designed to require minimal pre-processing. They are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariance characteristics. Convolution networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

Convolutional Neural Netwroks use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing.

### 7.1.2 Convolutional Neural Networks

Convolutional neural networks (CNN, ConvNet) are a class of deep, feed-forward (not recurrent) artificial neural networks that are applied to analyzing visual imagery. Convolutional neural networks use three basic factors to implement classification and recognition problems. In the local receptive fields, which are the fully connected layers, the input was taken as a vertical column of pixel intensities as shown in **Figure 7.2.** In a convolutional net, we will take it as a 28 by 28 square matrix of neurons, which corresponds to the input image. Here we won’t connect every input pixel in the first layer to every other neuron in the hidden layer, instead we make connections in small and localized regions of the input image.

Let’s say for example a 5 by 5 region corresponds to 25 input pixels. So, for a neuron we might have connections like the region in the input image is called the local receptive field of hidden layer neurons. Each connection learns a weight. In the next step of Shared weights and biases the main objective that it has a bias and weights connected to its local receptive field.

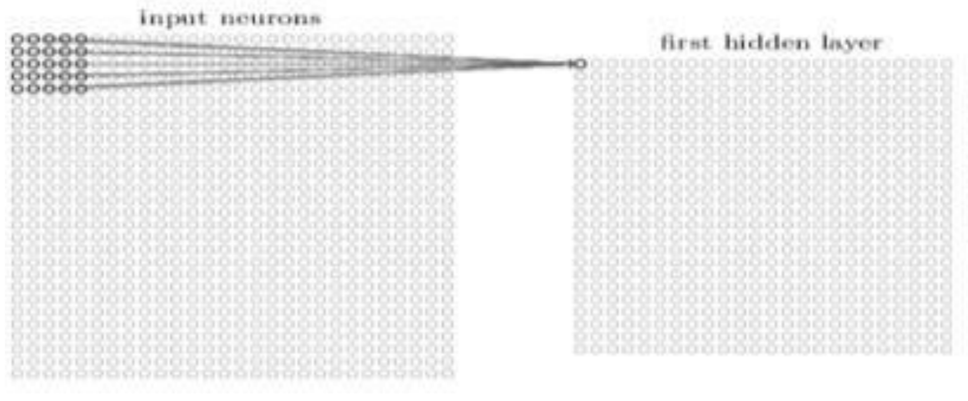


**Figure 7.2 Connections between input layer and hidden layer**

A noteworthy point is that we are going to use the same weights and bias for each of 24 by 24hidden neurons. To make it in simple terms is that all the neurons in the first hidden layer detect the same feature but at different places of the input image as the local receptive field moves through the input. To make it sensible, suppose the weights and bias are in such a way that the hidden layer can predict a vertical edge in a particular local receptive field, this prediction can be useful at other parts of the image.

To put in practical terms convolutional neural networks are well habituated to invariance of images. In our implementation we are going to use MNIST datasets that have less invariance compared to other images. So sometimes we call this mapping from input layer to hidden layer as the feature map. We define weights as shared weights and bias for knowing the feature as the shared bias, both often termed as kernel or filter. In the above figure showing 3 feature mappings defined by a 5by 5 shared weights and single bias per feature map. Now our network can detect 3 different kinds of features, with each feature can be predicted in every part of image.

The next processing step used is termed as Pooling layers which are also a part of the hidden layer and present after the Convolutional layer and gives more finer details of the feature mapped. Pooling layers simplifies the information from the output of a convolutional layer and makes a very thin and condensed feature map which can predict more thinner and finer details for feature extraction. These features can be again predicted at every place of image. To explain in practical terms each unit of the pooling layer may predict a 2 by 2 neurons from the convolutional layer. The procedure used for pooling is coined as Max-pooling.

We can combine these layers together using models of the keras library, but it has a layer of 10 classes of neurons representing the 10 possible values for EMNIST characters. Here the network takes 28 by 28 square matrix of neurons which are used to encode the pixel intensities. Then it is followed by a convolutional layer of 5 by 5 local receptive fields and containing 3 feature maps. The result is passed is passed onto pooling layers, which takes 2 by 2 regions as kernels. The output layer is a fully connected layer, in this layer every neuron from the pooled layer is connected to every one of 10 neurons in the output layer.

**ConvNets over Feed-Forward Neural Nets:**

An image is nothing but a matrix of pixel values. So why not just flatten the image (e.g., 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes. In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout. image as shown in **Figure 7.3** A ConvNet can successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

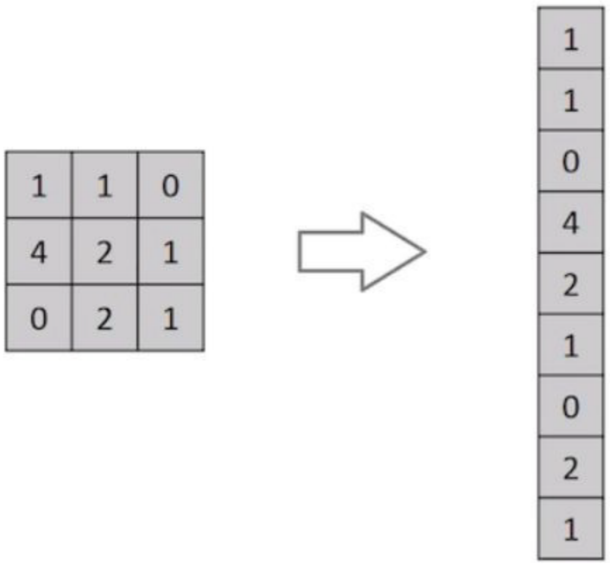


**Figure 7.3 Flattening of a 3x3 image matrix into a 9x1 vector.**

**Convolution Layer — Input Image:**

In the **Figure 7.4**, we have an RGB image that has been separated by its three-color planes — Red, Green, and Blue. There are several such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc. You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of ConvNet is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction. This is important when we are to design an architecture that is not only good at learning features but also scalable to massive datasets. [8]

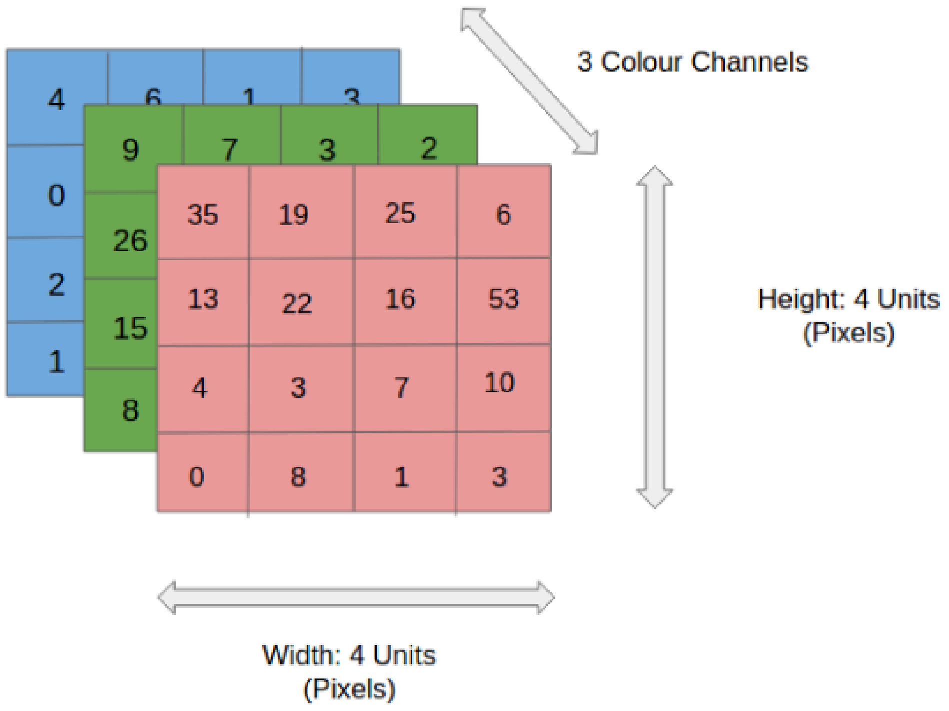


**Figure 7.4 4x4x3 RGB Image**

**Convolution Layer — The Kernel:**

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB). the green section resembles our 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in yellow as shown in **Figure 7.5**. We have selected K as a 3x3x1 matrix. The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing an elementwise multiplication operation (Hadamard Product) between K and the portion P of the image over which the kernel is hovering. The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

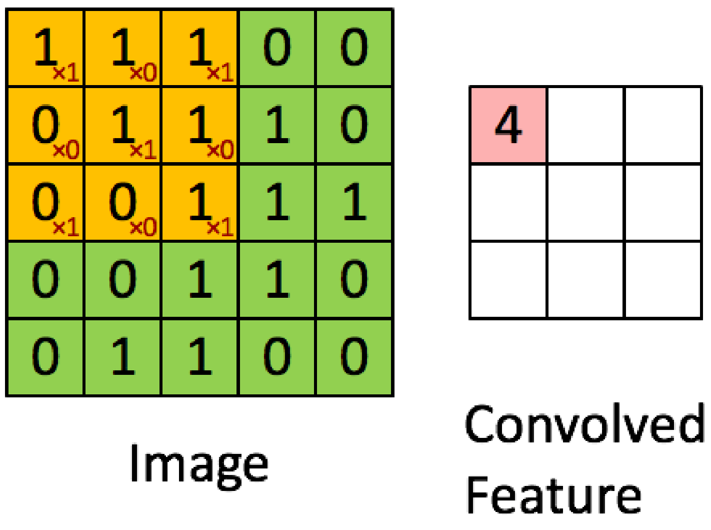


**Figure 7.5 Kernel**

In the case of images with multiple channels (e.g., RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output as shown in **Figure 7.6** The objective of the Convolution Operation is to extract the high-level features from the input image and need not be limited to only one layer.

Conventionally, the first Conv Layer is responsible for capturing the Low-Level features such as edges, colour, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network that has a wholesome understanding of images in the dataset, like how we would.

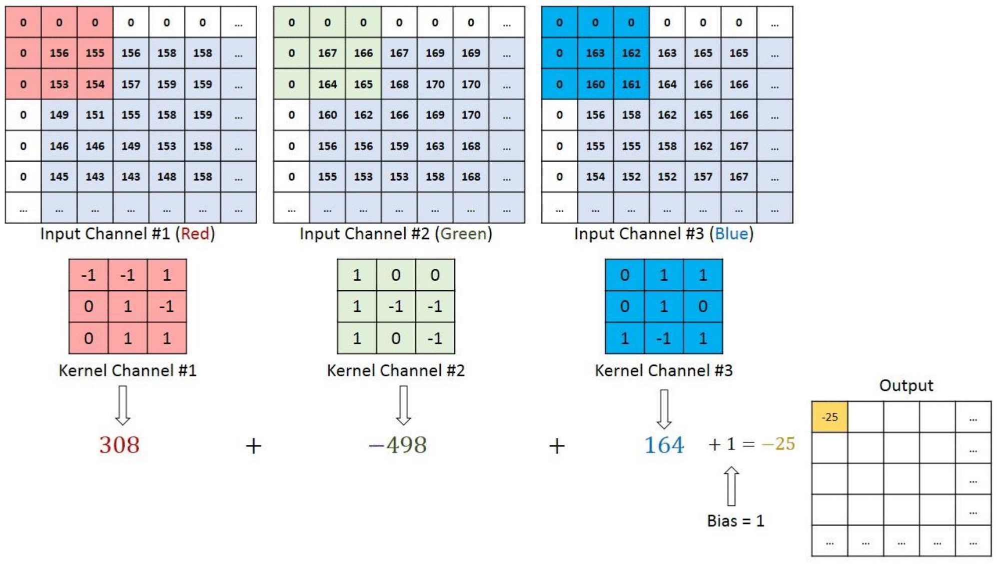


**Figure 7.6 Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel**

There are two types of results to the operation: one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding in the case of the former, or Same Padding in the case of the latter. When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1.Hence the name same padding as shown in **Figure 7.7** On the other hand, if we perform the same operation without padding, we are presented with a matrix that has dimensions of the Kernel (3x3x1) itself then it is valid padding.

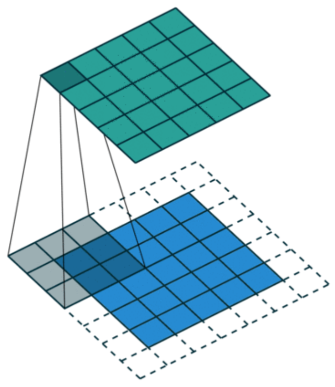


**Figure 7.7 Same padding**

## Pooling Layer

Like the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training the model. There are two types of Pooling: Max Pooling and Average Pooling.

Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise-suppressing mechanism as shown in **Figure 7.8.** Hence, we can say that Max Pooling performs a lot better than Average Pooling.

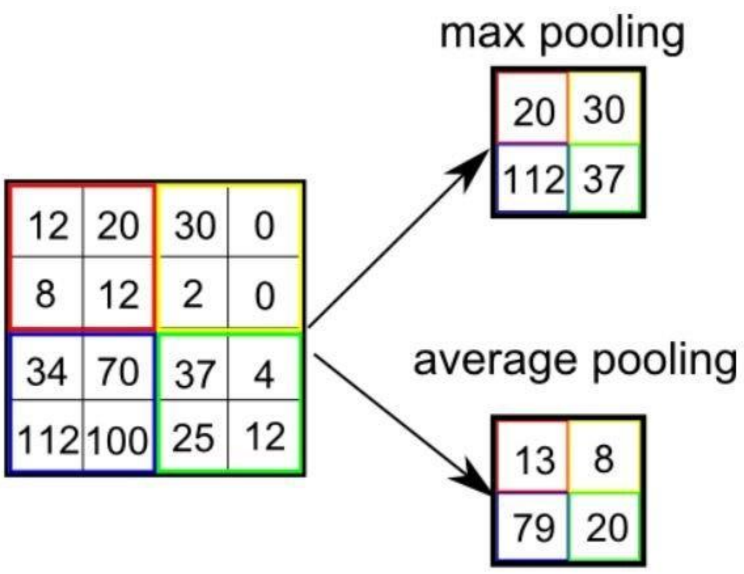


**Figure 7.8 Types of Pooling**

The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-level details even further, but at the cost of more computational power. After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

**Classification — Fully Connected Layer (FC Layer)**

Adding a Fully Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully Connected layer (shown in **Figure 7.9**) is learning a possibly non-linear function in that space. Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation is applied to every iteration of training.

Over a series of epochs, the model can distinguish between dominating and certain low-level features in images and classify them using the SoftMax Classification technique. There are various architectures of Convolutional Neural Networks available which have been key in building algorithms which power and shall power AI in the foreseeable future. The architecture used in our proposed model is MobileNetV2.

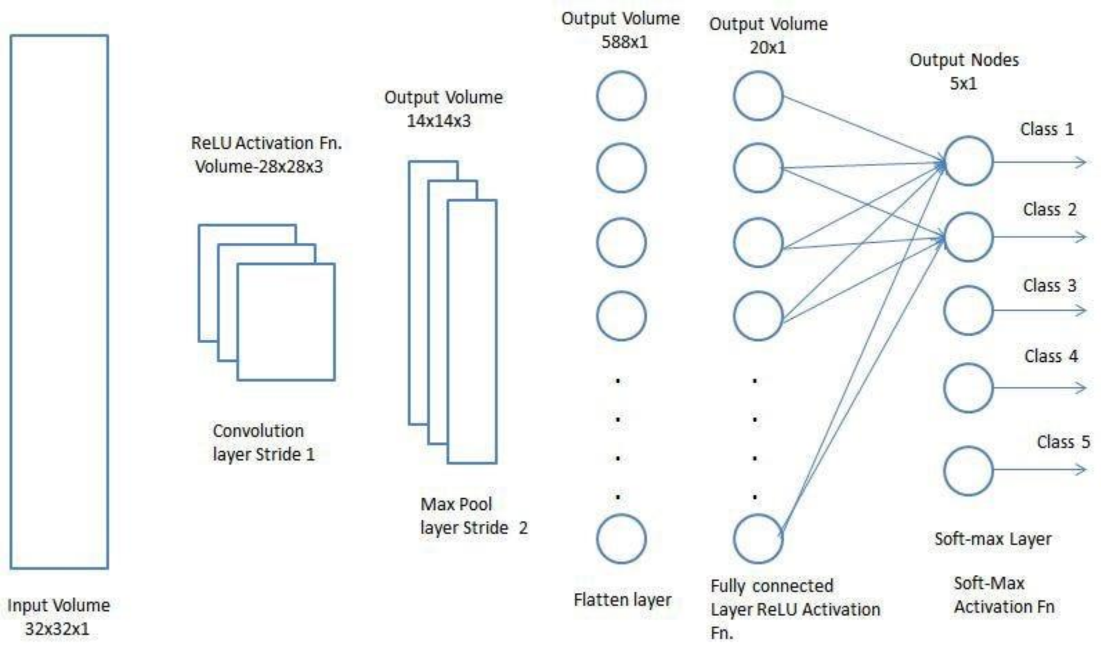


**Figure 7.9 Fully Connected Layer**

## 7.2 Architecture

A group of researchers from Google released a neural network architecture MobileNetV2, which is optimized for mobile devices. The architecture delivers high accuracy results while keeping the parameters and mathematical operations as low as possible to bring deep neural networks to mobile devices. Last year, the company introduced MobileNetV1 for TensorFlow, designed to support classification, detection, embedding and segmentation. The new mobile architecture, MobileNetV2 is the improved version of MobileNetV1 and is released as a part of TensorFlow-Slim Image Classification Library.

MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. According to the research paper, MobileNetV2 improves the state-of-the-art performance of mobile models on multiple tasks and benchmarks as well as across a spectrum of different model sizes. It is a very effective feature extractor for object detection and segmentation.

It builds upon the ideas from MobileNetV1, using depth-wise separable convolutions as efficient building blocks. However, Google says that the 2nd version of Mobile Net has two new features: Linear bottlenecks between the layers: Experimental evidence suggests that using linear layers is crucial as it prevents nonlinearities from destroying too much information. Using non-linear layers in bottlenecks indeed hurts the performance by several percent, further validating our hypothesis and Shortcut connections between the bottlenecks.

**Structure**

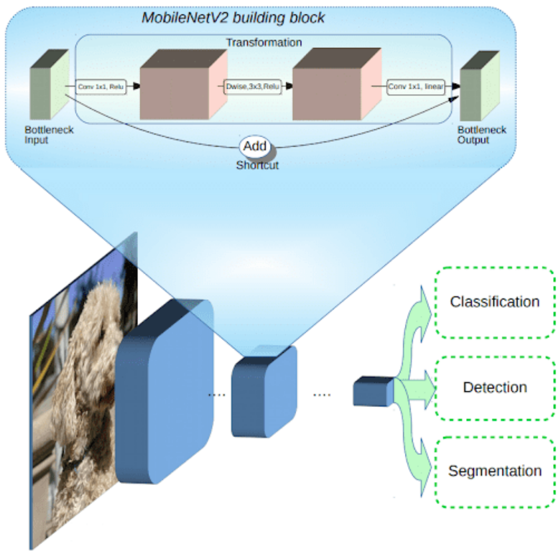
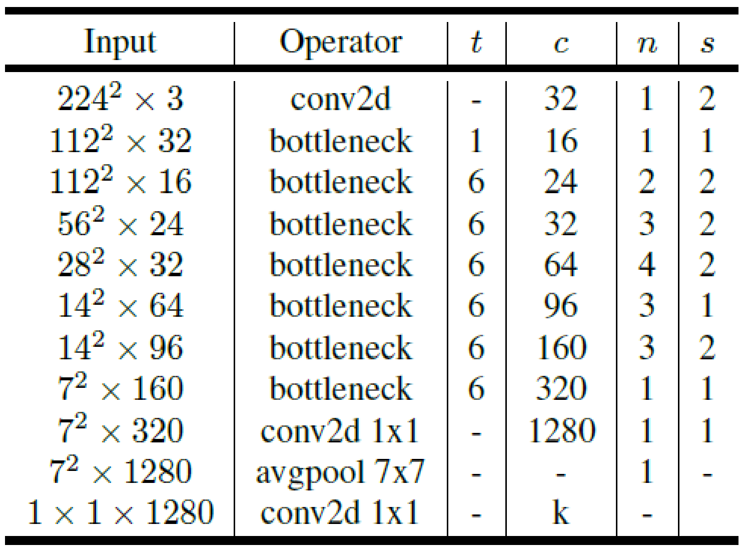


**Figure 7.10 MobileNetV2 Structure**

The bottlenecks of the MobileNetV2 encode the intermediate inputs and outputs while the inner layer encapsulates the model’s ability to transform from lower-level concepts such as pixels to higher level descriptors such as image categories (shown in **Figure 7.10**). With traditional residual connections, shortcuts enable faster training and better accuracy.

**Model Architecture**

The basic building block is a bottleneck depth-separable convolution with residuals. The architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers. The researchers have tailored the architecture to different performance points, by using the input image resolution and width multiplier as tunable hyperparameters, that can be adjusted depending on desired accuracy or performance trade-offs. The primary network (width multiplier 1, 224 × 224), has a computational cost of 300 million multiply-adds and uses 3.4 million parameters. The network computational cost ranges from 7 multiply-adds to 585M MAdds, while the model size varies between 1.7M and 6.9M parameters. [9]



| Input | Operator | $t$ | $c$ | $n$ | $s$ |
|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d | - | 32 | 1 | 2 |
| $112^2 \times 32$ | bottleneck | 1 | 16 | 1 | 1 |
| $112^2 \times 16$ | bottleneck | 6 | 24 | 2 | 2 |
| $56^2 \times 24$ | bottleneck | 6 | 32 | 3 | 2 |
| $28^2 \times 32$ | bottleneck | 6 | 64 | 4 | 2 |
| $14^2 \times 64$ | bottleneck | 6 | 96 | 3 | 1 |
| $14^2 \times 96$ | bottleneck | 6 | 160 | 3 | 2 |
| $7^2 \times 160$ | bottleneck | 6 | 320 | 1 | 1 |
| $7^2 \times 320$ | conv2d 1x1 | - | 1280 | 1 | 1 |
| $7^2 \times 1280$ | avgpool 7x7 | - | - | 1 | - |
| $1 \times 1 \times 1280$ | conv2d 1x1 | - | k | - | |

**Figure 7.11 MobileNetV2 Architecture**

where t: expansion factor, c: number of output channels, n: repeating number, s: stride. 3×3 kernels are used for spatial convolution (shown in **Figure 7.11**).

MobileNetV2 contains Residual blocks which are combination of 3 types of convolutional layers.

1. **Expansion Layer** is also a 1×1 convolution and used to expand the number of channels in the data before it goes into the depth wise convolution. Exactly by how much the data gets expanded is given by the expansion factor.

2. **Depth wise separable convolution Layer** performs two subtasks: first there is a 3×3 depth wise convolution layer that filters the input, followed by a 1×1 pointwise convolution layer that combines these filtered values to create new features.

3. **Projection Layer** is a 1×1 point wise convolution and it projects data with a high number of dimensions (channels) into a tensor with a much lower number of dimensions. This kind of layer is also called a **bottleneck layer** because it reduces the amount of data that flows through the network.

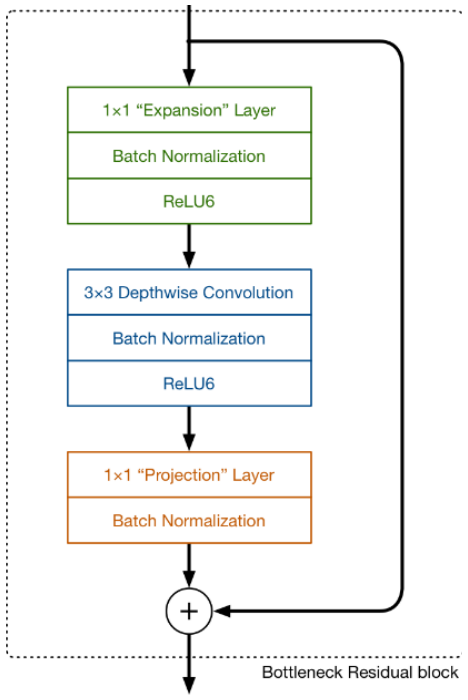


**Figure 7.12 Bottleneck Residual Block**

A **Residual connection** is also added between layers and used to help with the flow of gradients through the network i.e., provides another path for data to reach latter parts of neural network (shown in **Figure 7.12**).

## Normal Convolutions

A typical image, however, is not 2-D; it also has depth as well as width and height. Let us assume that we have an input image of 12x12x3 pixels, an RGB image of size 12x12. Let's do a 5x5 convolution on the image with no padding and a stride of 1. If we only consider the width and height of the image, the convolution process is kind of like this: 12x12 — (5x5) — >8x8. The 5x5 kernel undergoes scalar multiplication with every 25 pixels, giving out1 number every time. We end up with a 8x8 pixel image, since there is no padding (12–5+1 = 8).

However, because the image has 3 channels, our convolutional kernel needs to have 3 channels as well. This means, instead of doing 5x5=25 multiplications, we do 5x5x3=75 multiplications every time the kernel moves. Just like the 2-D interpretation, we do scalar matrix multiplication on every 25 pixels, outputting 1 number. After going through a 5x5x3 kernel, the 12x12x3 image will become a 8x8x1 image.

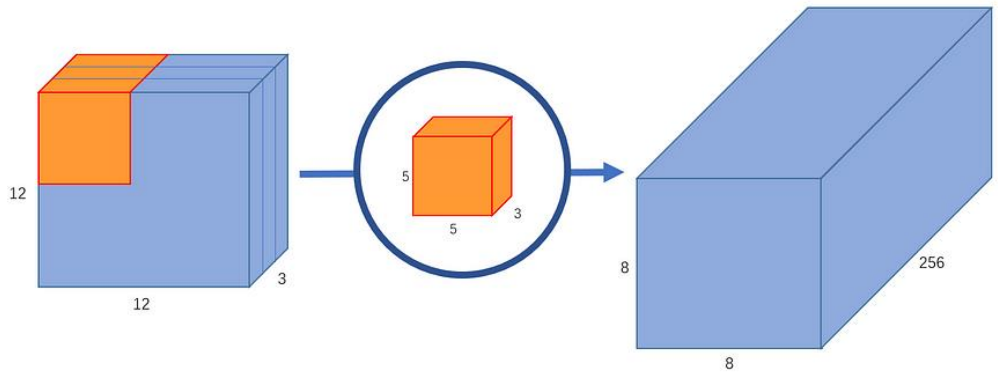


**Figure 7.13 Normal Convolution**

Suppose if we want to increase the number of channels in our output image (say, output of size 8x8x256). we can create 256 kernels to create 256 8x8x1 images, then stack them up together to create an 8x8x256 image output. This is how a normal convolution works. I like to think of it like a function: 12x12x3 — (5x5x3x256) —>12x12x256 (Where 5x5x3x256 represents the height, width, number of input channels, and number of output channels of the kernel as shown in **Figure 7.13**). Not that this is not matrix multiplication; we're not multiplying the whole image by the kernel but moving the kernel through every part of the image and multiplying small parts of it separately. A depth wise separable convolution separates this process into 2 parts: a depth wise convolution and a pointwise convolution.

## Spatial Separable Convolutions

The spatial separable convolution is so named because it deals primarily with the spatial dimensions of an image and kernel: the width and the height. (The other dimension, the "depth" dimension, is the number of channels of each image) A spatial separable convolution simply divides a kernel into two, smaller kernels. The most common case would be to divide a 3x3 kernel into a 3x1 and 1x3 kernel. Now, instead of doing one convolution with 9 multiplications, we do two convolutions with 3 multiplications each (6 in total) to achieve the same effect. With less multiplications, computational complexity goes down, and the network can run faster.

## Understanding Depth Wise Separable Convolutions

Unlike spatial separable convolutions, depth wise separable convolutions work with kernels that cannot be "factored" into two smaller kernels. Hence, it is more commonly used. The depth wise separable convolution is so named because it deals not just with the spatial dimensions, but with the depth dimension — the number of channels as well. An input image may have 3 channels: RGB. After a few convolutions, an image may have multiple channels.

You can image each channel as a particular interpretation of that image; in for example, the “red” channel interprets the “redness” of each pixel, the “blue” channel interprets the “blueness” of each pixel, and the “green” channel interprets the “greenness” of each pixel. An image with 64 channels has 64 different interpretations of that image. Like the spatial separable convolution, a depth wise separable convolution splits a kernel into two separate kernels that do two convolutions: the depth wise convolution and the pointwise convolution.

**Part 1 — Depth wise Convolution:**

In the first part, depth wise convolution, we give the input image a convolution without changing the depth (shown in **Figure 7.14**). We do so by using 3 kernels of shape 5x5x1. Each 5x5x1 kernel iterates 1 channel of the image (note: 1 channel, not all channels), getting the scalar products of every 25-pixel group, giving out a 8x8x1 image. Stacking these images together creates an 8x8x3 image.

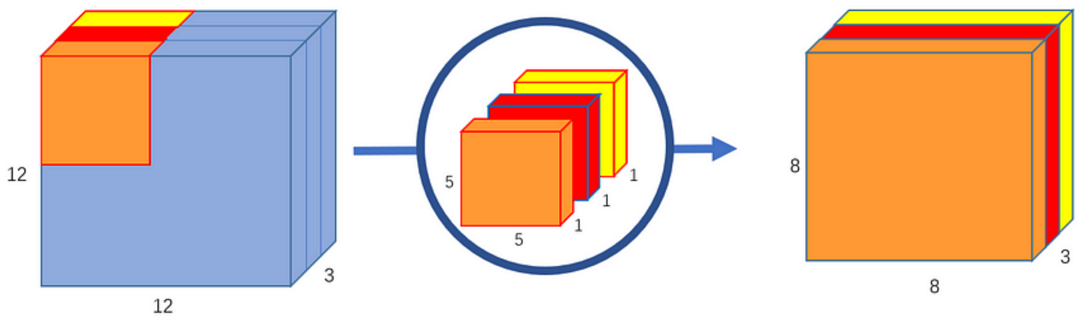


**Figure 7.14 Depth Wise Convolution**

**Part 2 — Point wise Convolution:**

Remember, the original convolution transformed a 12x12x3 image to a 8x8x256 image. Currently, the depth wise convolution has transformed the 12x12x3 image to a 8x8x3 image. Now, we need to increase the number of channels of each image. The pointwise convolution is so named because it uses a 1x1 kernel, or a kernel that iterates through every single point. This kernel has a depth of however many channels the input image has; in our case, 3. Therefore, we iterate a 1x1x3 kernel through our 8x8x3 image, to get an 8x8x1 image.

We can create 256 1x1x3 kernels that output an 8x8x1 image each to get a final image of shape 8x8x256. And that's it! We've separated the convolution into 2: a depth wise convolution and a pointwise convolution (shown in **Figure 7.15**). In a more abstract way, if the original convolution function is 12x12x3 — (5x5x3x256) →12x12x256, we can illustrate this new convolution as 12x12x3 — (5x5x1x1) — > (1x1x3x256) — >12x12x256.

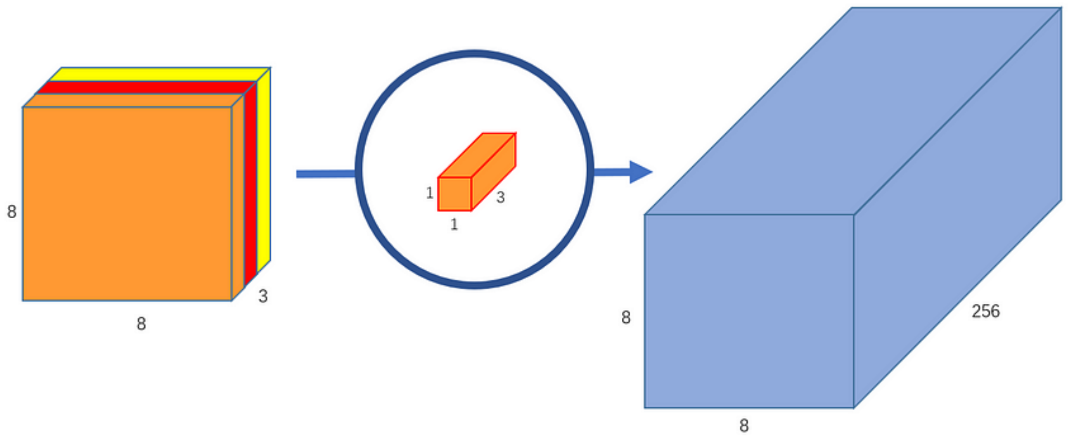


**Figure 7.15 Point Wise Convolution**

## Inverted Residual and Linear Bottleneck Layer

MobileNetV2 model also consists of Inverted residual and linear bottleneck layers. The premise of the inverted residual layer is that feature maps can be encoded in low-dimensional subspaces and non-linear activations result in information loss in spite of their ability to increase representational complexity. These principles guide the design of the new convolutional layer. The layer takes in a low-dimensional tensor with k channels and performs three separate convolutions.

First, a pointwise (1x1) convolution is used to expand the low-dimensional input feature map to a higher-dimensional space suited to non-linear activations, then ReLU6 is applied. The expansion factor is referred to as t throughout the paper, leading to tk channels in this first step. Next, a depth-wise convolution is performed using 3x3 kernels, followed by ReLU6 activation, achieving spatial filtering of the higher-dimensional tensor. Finally, the spatially filtered feature map is projected back to a low-dimensional subspace using another point-wise convolution (shown in **Figure 7.16**).

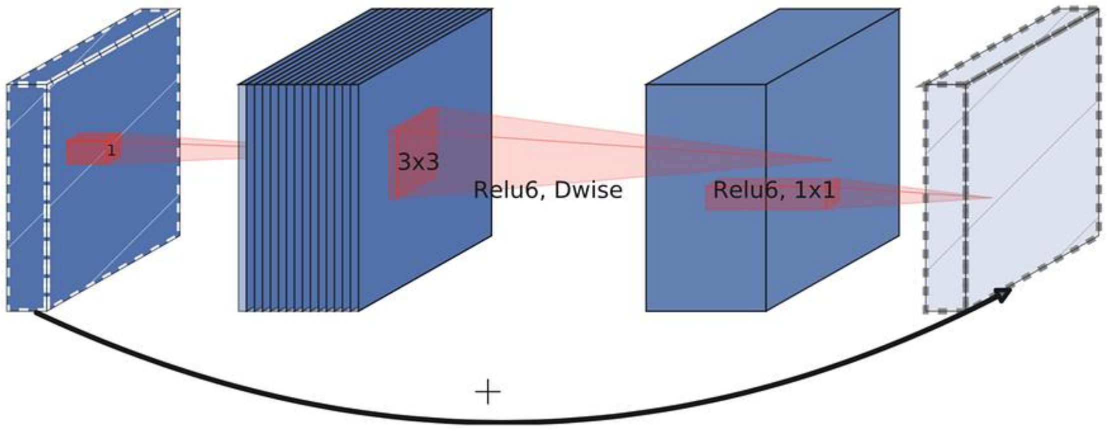


**Figure 7.16 Inverted Residual Layer**

The projection alone results in loss of information, so, intuitively, it’s important that the activation function in the last step be linear activation (see below for an empirical justification). When the initial and final feature maps are of the same dimensions (when the depth-wise convolution stride equals one and input and output channels are equal), a residual connection is added to aid gradient flow during backpropagation. Note that the final two steps are essentially a depth-wise separable convolution with the requirement that there be dimensionality reduction.

The reason we use non-linear activation functions in neural networks is that multiple matrix multiplications cannot be reduced to a single numerical operation. It allows us to build neural networks that have multiple layers. At the same time the activation function ReLU, which is commonly used in neural networks, discards values that are smaller than 0. This loss of information can be tackled by increasing the number of channels to increase the capacity of the network.

With inverted residual blocks we do the opposite and squeeze the layers where the skip connections are linked. This hurts the performance of the network. The authors introduced the idea of a linear bottleneck where the last convolution of a residual block has a linear output before it’s added to the initial activations. Putting this into code is super simple as we simply discard the last activation function of the convolutional block.

# 8 Implementation

Implementation is the process of converting a new system design into an operational one. It is the key stage in achieving a successful new system. It must therefore be carefully planned and controlled. The implementation of a system is done after the development effort is completed. The implementation phase of software development is concerned with translating design specifications into source code. The primary goal of implementation is to write source code and internal documentation so that conformance of the code to its specifications can be easily verified and so that debugging testing and modification are eased. This goal can be achieved by making the source code as clear and straightforward as possible. Simplicity clarity and elegance are the hallmarks of good programs and these characteristics have been implemented in each program module. The proposed methods contain three phases such as: Preprocessing, Feature extraction, Classification, and detection of fake currency.

Pre-processing is a series of operations performed on input text. It essentially enhances the image rendering suitable for segmentation. The role of preprocessing is to segment the interesting pattern from the background. Generally, noise filtering, smoothing and normalization should be done in this step. The pre-processing also defines a compact representation of the pattern. Feature Extraction is performed with the help of Convolutional Neural Network Algorithm. Classification and detection of fake currency notes were performed with the help of MonileNetV2 model.

## 8.1 Datasets

The proposed model consists of three modules and each module uses different dataset. This section mainly deals with the type of datasets that were used in the proposed work. The images in the dataset were generated with the help of random data augmentation.

### 8.1.1 Classification Dataset

The main objective of classification module is to classify the currency using mobilenetv2 architecture. Dataset used in this model contains two directories namely Training set and Test set. Training set contains 9000 images and Test set contains 4500 images belonging to 45 different classes. The dataset contains different denomination directories based on type (old or new), face orientation (front or back), and alignment (up or down) from a front light image of the note. Example images in classification dataset were shown in **Figure 8.1**

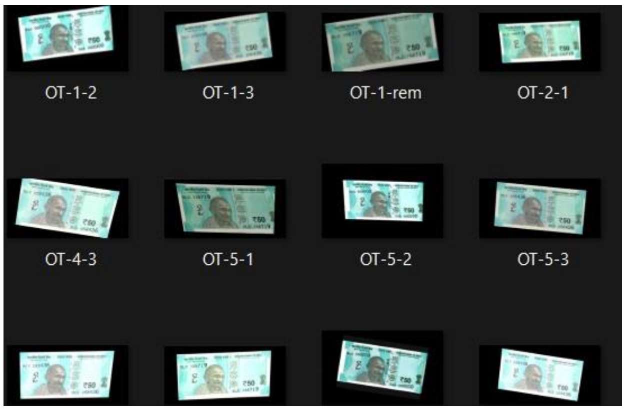


**Figure 8.1 Classification Dataset**

### 8.1.2 Watermark Dataset

The images in the watermark dataset were used to detect the presence of the Gandhi watermark on currency note to determine whether the note is fake or real. Dataset used in this model contains two directories namely Training set and Test set. Training set contains 6000 images and Test set contains 3000 images (shown in **Figure 8.2**) belonging to 2 different classes which specifies whether the note has watermark or not.

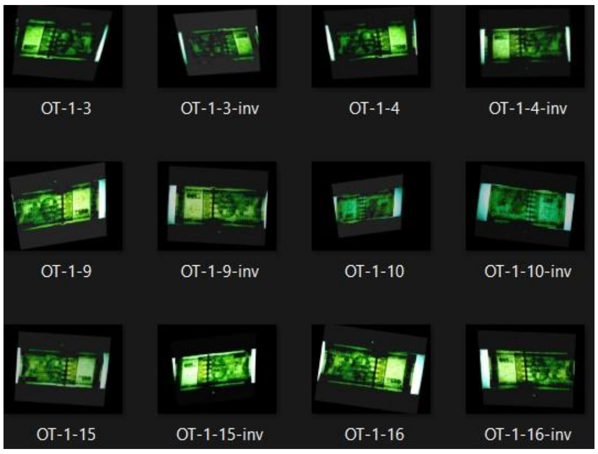


**Figure 8.2 Watermark Dataset**

### 8.1.3 UV Dataset

The images in the uv dataset were used to detect the presence of the Security Thread on currency note to determine whether the note is fake or real. Dataset used in this model contains two directories namely Training set and Test set. Training set contains 6000 images and Test set contains 3000 images belonging to 3 different classes. (Shown in **Figure 8.3**) No patch class determines the authenticity of currency note.

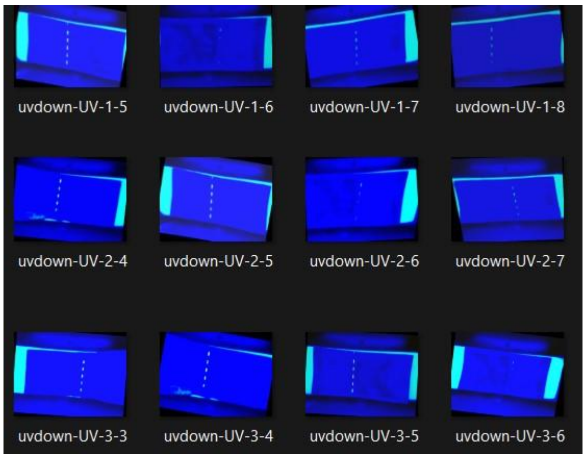


**Figure 8.3 UV Dataset**

## 8.2 Source Code

The proposed model consists of three modules. Classification module is used to classify the currency type. UV and Watermark modules were used to detect security features of the currency notes to determine whether the note is fake or real.

### 8.2.1 Classification Model

**classify_mobilenet_train.ipynb:**

The below code module depicts the libraries used in the training model.

```
import math
import keras
from keras import backend as K
from keras import regularizers
from keras.layers.core import Dense, Activation
from keras.optimizers import Adam, RMSprop, SGD, Adadelta
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Model
from keras.applications import imagenet_utils
from keras.layers import Dense, GlobalAveragePooling2D,
Dropout, BatchNormalization
from keras.applications import MobileNetV2
from keras.applications.mobilenet import preprocess_input
from keras.callbacks import ModelCheckpoint, EarlyStopping
import numpy as np
import matplotlib.pyplot as plt
```

Batch size is defined to specify number of images that can be passed as a batch in one iteration. 2 datasets were generated with the augmentation technique.

```
batch_size = 32
train_gen=ImageDataGenerator(preprocessing_function=preprocess
_input)
training_set=train_gen.flow_from_directory(directory="../datas
et/training_set",
target_size=(224,224),
batch_size=batch_size,
class_mode='categorical',
shuffle=True)
test_gen=ImageDataGenerator(preprocessing_function=preprocess_
input)
test_set=test_gen.flow_from_directory(directory="../dataset/te
st_set",
target_size=(224,224),
batch_size=batch_size,
class_mode='categorical',
shuffle=True)
```

MobileNet model is created with pretrained weights by detaching the fully connected layer. Average pooling is applied, and a dense layer is added with 45 nodes.

```
mobilenetmodel=MobileNetV2(weights='imagenet',input_shape=(224
, 224, 3), include_top=False)
x = mobilenetmodel.output
x = GlobalAveragePooling2D()(x)
preds=Dense(45,activation='softmax',kernel_regularizer=regular
izers.l2(0.001))(x)
model_final=Model(inputs=mobilenetmodel.input,outputs=preds)
```

Compilation of model is performed below code module with the training and testing sizes specified. Stochastic Gradient descent is applied as optimization algorithm.

```
training_size = 9000
validation_size = 4500
steps_per_epoch = math.ceil(training_size / batch_size)
validation_steps = math.ceil(validation_size / batch_size)
optimizer1 = SGD(learning_rate=0.001)
model_final.compile(optimizer=optimizer1,loss='categorical_cro
ssentropy',metrics=['accuracy'])
```

Early stop callback function is used to stop the training process of the model once its performance starts to stop improving. The classes of currency model are written to a file. The plotting of training of model can be seen in subsequent code module.

```
earlystop1=EarlyStopping(monitor='val_loss',min_delta=0,patien
ce=10, verbose=1, mode='auto')
hist1=model_final.fit(training_set,
steps_per_epoch=steps_per_epoch,
epochs=2,
validation_data=test_set,
validation_steps=validation_steps,
callbacks=[earlystop1], workers=10, shuffle=True)
model_final.save("currency_mobilenetmodel.h5")
f = open("mobilenet_currency_class_indices.txt", "w")
f.write(str(training_set.class_indices))
f.close()
```

```
plt.plot(hist1.history["accuracy"])
plt.plot(hist1.history['val_accuracy'])
plt.plot(hist1.history['loss']
plt.plot(hist1.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","ValidationAccuracy","loss","Validation
Loss"])
```

**classify_mobilenet_test.ipynb:**

The below code module depicts the libraries used in the training model.

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.preprocessing import image
from keras.models import load_model
import ast, os
```

The saved model is loaded to test model and currency classes were read from the file and defined as final labels.

```
model = load_model('currency_mobilenetmodel.h5')
f = open("mobilenet_currency_class_indices.txt", "r")
labels = f.read()
labels = ast.literal_eval(labels)
final_labels = {v: k for k, v in labels.items()}
```

The test directory consists of group of test images to determine the authenticity of currency notes and obtain a accuracy score.

```
def verify_test_dir():
    path = '..\\batch-test-images'
    files = os.listdir(path)
    correct_preds = 0
    file_count = len(files)
    finals = []
    for filename in files:
        digits = []
        for char in filename:
            if char.isdigit():
                digits += char
            else:
                break
        num = "".join(digits)
        currency_string = []
        currency_string.append(num)
        if 'old' in filename:
            currency_string.append('old')
        else:
            currency_string.append('new')
        if 'back' in filename:
            currency_string.append('back')
        elif 'front' in filename:
            currency_string.append('front')
        rev = filename.split('.')[0][-1:]
        if rev == 'r':
            currency_string.append('down')
        else:
            currency_string.append('up')
        currency_string = "_".join(currency_string)
        prediction=predict_image(path + '\\' + filename, True)
        if list(prediction.keys())[0] == currency_string:
     print("{}:CORRECTPREDICTION".format(filename),prediction)
            print("Prednote:",list(prediction.keys())[0])
             print("Orginal note:",currency_string)
            correct_preds += 1
        else:
  print("{}:INCORRECTPREDICTION!".format(filename),prediction)
            print("Prednote:",list(prediction.keys())[0])
            print("Orginal note:", currency_string)
    acc=(correct_preds/file_count)*100
    print('accuracy:',acc)
    verify_test_dir()
```

**convert_to_lite_classify.ipynb:**

The saved model is converted to lite model for usage of model in edge devices and mobile devices.

```
import tensorflow as tf
model=tf.keras.models.load_model('currency_mobilenetmodel.h5')
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open("converted_currency_mobilenetmodel.tflite","wb").write(tf
lite_model)
```

### 8.2.2 Watermark Model

**watermark_mobilenet_train.ipynb:**

The below code module depicts the libraries used in the training model.

```
import math
import keras
from keras import backend as K
from keras import regularizers
from keras.layers.core import Dense, Activation
from keras.optimizers import Adam, RMSprop, SGD, Adadelta
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Model
from keras.applications import imagenet_utils
from keras.layers import Dense, GlobalAveragePooling2D,
Dropout, BatchNormalization
from keras.applications import MobileNetV2
from keras.applications.mobilenet import preprocess_input
from keras.callbacks import ModelCheckpoint, EarlyStopping
import numpy as np, matplotlib.pyplot as plt
```

Batch size is defined to specify number of images that can be passed as a batch in one iteration. 2 datasets were generated with the augmentation technique.

```
batch_size = 16
train_gen=ImageDataGenerator(preprocessing_function=preprocess
_input)
training_set=train_gen.flow_from_directory(directory="../datas
et/training_set",
target_size=(224,224),
batch_size=batch_size,
class_mode='categorical',
shuffle=True)
test_gen=ImageDataGenerator(preprocessing_function=preprocess_
input)
test_set=test_gen.flow_from_directory(directory="../dataset/te
st_set",
target_size=(224,224),
batch_size=batch_size,
class_mode='categorical',
shuffle=True)
```

MobileNet model is created with pretrained weights by detaching the fully connected layer. Average pooling is applied, and a dense layer is added with 2 nodes.

```
mobilenetmodel=MobileNetV2(weights='imagenet',input_shape=(224
, 224, 3), include_top=False)
x = mobilenetmodel.output
x = GlobalAveragePooling2D()(x)
preds=Dense(2,activation='softmax',kernel_regularizer=regulari
zers.l2(0.001))(x)
model_final=Model(inputs=mobilenetmodel.input,outputs = preds)
```

Compilation of model is performed below code module with the training and testing sizes specified. Stochastic Gradient descent is applied as optimization algorithm.

```
training_size = 6000
validation_size = 3000
steps_per_epoch = math.ceil(training_size / batch_size)
validation_steps = math.ceil(validation_size / batch_size)
optimizer1 = SGD(learning_rate=0.001)
model_final.compile(optimizer=optimizer1,loss=binary_crossentr
opy',metrics=['accuracy'])
```

Early stop callback function is used to stop the training process of the model once its performance starts to stop improving. The classes of currency model are written to a file. The plotting of training of model can be seen in subsequent code module.

```
earlystop1=EarlyStopping(monitor='val_loss',min_delta=0,patien
ce=10, verbose=1, mode='auto')
hist1=model_final.fit(training_set,
steps_per_epoch=steps_per_epoch,
epochs=2,
validation_data = test_set,
validation_steps=validation_steps,
callbacks=[earlystop1],
workers=10,
shuffle=True)
model_final.save("watermark_mobilenetmodel.h5")
f = open("mobilenet_watermark_class_indices.txt", "w")
f.write(str(training_set.class_indices))
f.close()
```

```
plt.plot(hist1.history["accuracy"])
plt.plot(hist1.history['val_accuracy'])
plt.plot(hist1.history['loss']
plt.plot(hist1.history['val_loss'])
plt.xlabel("Epoch")
plt.legend(["Accuracy","ValidationAccuracy","loss","Validation
Loss"])
```

**watermark_mobilenet_test.ipynb:**

The below code module depicts the libraries used in the training model.

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.preprocessing import image
from keras.models import load_model
import ast, os
```

The saved model is loaded to test model and currency classes were read from the file and defined as final labels.

```
model = load_model('watermark_mobilenetmodel.h5')
f = open("mobilenet_watermark_class_indices.txt", "r")
labels = f.read()
labels = ast.literal_eval(labels)
final_labels = {v: k for k, v in labels.items()}
```

The test directory consists of group of test images to determine the authenticity of currency notes and obtain a accuracy score.

```
def verify_test_dir():
    path = '..\\batch-test-images'
    folders = os.listdir(path)
    correct_preds = 0
    file_count = 0
    for fold in folders:
        files = os.listdir(path + '\\' + fold)
        for filename in files:
            watermark_string = fold
prediction=predict_image(path+'\\{}\\'.format(fold)+filename,T
rue)
            if list(prediction.keys())[0] == watermark_string:
                print("{}\{}:CORRECTPREDICTION".format(fold,
filename),prediction)
print("Predictednote:",list(prediction.keys())[0])
             print("Originalnote:",watermark_string)
                correct_preds += 1
            else:
                print("{}\{}:INCORRECTPREDICTION".format(fold,
filename), prediction)
            print("Predictednote:",list(prediction.keys())[0])
                print("Orginal note:", watermark_string)
            file_count += 1
    acc=(correct_preds/file_count)*100
    print('accuracy:',acc)
verify_test_dir()
```

**convert_to_lite_watermark.ipynb:**

The saved model is converted to lite model for usage of model in edge devices and mobile devices.

```
import tensorflow as tf
model=tf.keras.models.load_model('watermark_mobilenetmodel.h5)
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open("converted_watermark_mobilenetmodel.tflite","wb").write(t
flite_model)
```

## 8.2.3 UV Model

**uv_mobilenet_train.ipynb:**

The below code module depicts the libraries used in the training model.

```
import math
import keras
from keras import backend as K
from keras import regularizers
from keras.layers.core import Dense, Activation
from keras.optimizers import Adam, RMSprop, SGD, Adadelta
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Model
from keras.applications import imagenet_utils
from keras.layers import Dense, GlobalAveragePooling2D,
Dropout, BatchNormalization
from keras.applications import MobileNetV2
from keras.applications.mobilenet import preprocess_input
from keras.callbacks import ModelCheckpoint, EarlyStopping
import numpy as np
import matplotlib.pyplot as plt
```

Batch size is defined to specify number of images that can be passed as a batch in one iteration. 2 datasets were generated with the augmentation technique.

```
batch_size = 16
train_gen=ImageDataGenerator(preprocessing_function=preprocess
_input)
training_set=train_gen.flow_from_directory(directory="../datas
et/training_set",
target_size=(250,250),
batch_size=batch_size,
class_mode='categorical',
shuffle=True)
test_gen=ImageDataGenerator(preprocessing_function=preprocess_
input)
test_set=test_gen.flow_from_directory(directory="../dataset/te
st_set",
target_size=(250,250),
batch_size=batch_size,
class_mode='categorical',
shuffle=True)
```

MobileNet model is created with pretrained weights by detaching the fully connected layer. Average pooling is applied, and a dense layer is added with 3 nodes.

```
mobilenetmodel=MobileNetV2(weights='imagenet',input_shape=(250
, 250, 3),include_top=False)
x = mobilenetmodel.output
x = GlobalAveragePooling2D()(x)
preds=Dense(45,activation='softmax',kernel_regularizer=regular
izers.l2(0.001))(x)
model_final = Model(inputs = mobilenetmodel.input, outputs =
preds)
```

Compilation of model is performed below code module with the training and testing sizes specified. Stochastic Gradient descent is applied as optimization algorithm.

```
training_size = 6000
validation_size = 3000
steps_per_epoch = math.ceil(training_size / batch_size)
validation_steps = math.ceil(validation_size / batch_size)
optimizer1 = SGD(learning_rate=0.001)
model_final.compile(optimizer=optimizer1,loss='categorical_cro
ssentropy',metrics=['accuracy'])
```

Early stop callback function is used to stop the training process of the model once its performance starts to stop improving. The classes of currency model are written to a file. The plotting of training of model can be seen in subsequent code module.

```
earlystop1 = EarlyStopping(monitor='val_loss', min_delta=0,
patience=10, verbose=1, mode='auto')
hist1=model_final.fit(training_set,
steps_per_epoch=steps_per_epoch,
epochs=2,
validation_data = test_set,
validation_steps=validation_steps,
callbacks=[earlystop1],
workers=10,
shuffle=True)
model_final.save("uv_mobilenetmodel.h5")
f = open("mobilenet_uv_class_indices.txt", "w")
f.write(str(training_set.class_indices))
f.close()
```

```
plt.plot(hist1.history["accuracy"])
plt.plot(hist1.history['val_accuracy'])
plt.plot(hist1.history['loss']
plt.plot(hist1.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","ValidationAccuracy","loss","Validation
Loss"])
```

**uv_mobilenet_test.ipynb:**

The below code module depicts the libraries used in the training model.

```
import numpy as np
import matplotlib.pyplot as plt
from keras.preprocessing import image
from keras.models import load_model
import keras, ast, os, cv2
import tensorflow as tf
```

The saved model is loaded to test model and currency classes were read from the file and defined as final labels.

```
model = load_model('uv_mobilenetmodel.h5')
f = open("mobilenet_uv_class_indices.txt", "r")
labels = f.read()
labels = ast.literal_eval(labels)
final_labels = {v: k for k, v in labels.items()}
```

The sub() is defined for preprocessing of uv image by cropping sufficient area of image.

```
def sub(og):
    h, w, c = og.shape
    x_start_r = 0.3125
    x_end_r = 0.703125
    y_start_r = 0.270833
    y_end_r = 0.791666
    x = int(np.floor(x_start_r * w))
    y = int(np.floor(y_start_r * h))
    x2 = int(np.floor(x_end_r * w))
    y2 = int(np.floor(y_end_r * h))
    image = og[y:y2, x:x2].copy()
    thresh = image
    return thresh
```

The test directory consists of group of test images to determine the authenticity of currency notes and obtain a accuracy score.

```
def verify_test_dir():
    path = '..\\batch-test-images'
    folders = os.listdir(path)
    correct_preds = 0
    file_count = 0
    for fold in folders:
        files = os.listdir(path + '\\' + fold)
        for filename in files:
            UV_type_string = fold
prediction=predict_image(path+'\\{}\\'.format(fold)+filename,
True)
            if list(prediction.keys())[0] == UV_type_string:
print("{}\{}:CORRECTPREDICTION".format(fold,filename),predicti
on)
print("Predictednote:",list(prediction.keys())[0])
                print("Orginalnote:",UV_type_string)
                correct_preds += 1
            else:
                print("{}\{}:INCORRECTPREDICTION".format(fold,
filename), prediction)
print("Predictednote:",list(prediction.keys())[0])
                print("Orginal note:", UV_type_string)
            file_count += 1
    acc=(correct_preds/file_count)*100
    print('accuracy:',acc)
verify_test_dir()
```

**convert_to_lite_uv.ipynb:**

The saved model is converted to lite model for usage of model in edge devices and mobile devices.

```
import tensorflow as tf
model = tf.keras.models.load_model('uv_mobilenetmodel.h5')
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open("converted_uv_mobilenetmodel.tflite","wb").write(tflite_m
odel)
```

### 8.2.4 Final Model

**final.ipynb:**

The below code module describes the used libraries in proposed work.

```
import tensorflow as tf
from PIL import Image
import numpy as np
import os, ast, re
import cv2
```

The lite versions of three modules can be accessed with the help of interpreter.

```
classify_interpreter=tf.lite.Interpreter(model_path="converted
_currency_mobilenetmodel.tflite")
watermark_interpreter=tf.lite.Interpreter(model_path="converte
d_watermark_mobilenetmodel.tflite")
uv_interpreter=tf.lite.Interpreter(model_path="converted_uv_mo
bilenetmodel.tflite")
classify_input_details=classify_interpreter.get_input_details(
)
classify_output_details=classify_interpreter.get_output_detail
s()
watermark_input_details=watermark_interpreter.get_input_detail
s()
watermark_output_details=watermark_interpreter.get_output_deta
ils()
uv_input_details = uv_interpreter.get_input_details()
uv_output_details = uv_interpreter.get_output_details()
classify_interpreter.allocate_tensors()
watermark_interpreter.allocate_tensors()
uv_interpreter.allocate_tensors()
```

The labels of the three models can be accessed with the help of following function.

```
def get_labels(path):
      f = open(path, "r")
      labels = f.read()
      labels = ast.literal_eval(labels)
      final_labels = {v: k for k, v in labels.items()}
      return final_labels
currency_labels=get_labels('mobilenet_currency_class_indices.t
xt')
uv_labels = get_labels('mobilenet_uv_class_indices.txt')
watermark_labels=get_labels('mobilenet_watermark_class_indices
.txt')
```

The below code module is used for prediction of the image and determining the authenticity of currency note. The sub () is used to crop the unnecessary area of the image and consider area around the security thread. The function takes the interpreter parameters as input and path of the image is also passed as parameter for prediction

```
def predict_image(imgname, final_labels, interpreter,
input_details, output_details):
    test_image = cv2.imread(imgname)
    test_image = cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB)
    if final_labels==uv_labels:
        h, w, c = test_image.shape #height*width*channel - ex:
480,640,3
        x_start_r = 0.3125
        x_end_r = 0.703125
        y_start_r = 0.270833
        y_end_r = 0.791666
        x = int(np.floor(x_start_r * w))
        y = int(np.floor(y_start_r * h))
        x2 = int(np.floor(x_end_r * w))
        y2 = int(np.floor(y_end_r * h))
        test_image = test_image[y:y2, x:x2].copy()
    else:
```

```
    test_image=cv2.resize(test_image,(224,224),cv2.INTER_AREA)
test_image=np.expand_dims(test_image,axis=0).astype(np.float32
)
    test_image = (2.0 / 255.0) * test_image - 1.0
    test_image=test_image.astype(np.float32)
interpreter.set_tensor(input_details[0]['index'],test_image)
    interpreter.invoke()
    result= interpreter.get_tensor(output_details[0]['index'])
    result_dict = dict()
    for key in list(final_labels.keys()):
        result_dict[final_labels[key]] = result[0][key]
    sorted_results={k:v for k,v in sorted(result_dict.items(),
key=lambda item: item[1], reverse=True)}
    final_result = dict()
final_result[list(sorted_results.keys())[0]]=sorted_results[li
st(sorted_results.keys())[0]] * 100
    return final_result
```

Classification of currency is performed in below module.

```
path_classify=input()
res_classify=predict_image(path_classify,currency_labels,class
ify_interpreter,classify_input_details,classify_output_details
)
if res_classify.get("invalid") is not None:
    print(res_classify)
    a=cv2.imread(path_classify)
    cv2.imshow('fake note',a)
    cv2.waitKey(0)
    print("Fake")
    cv2.destroyAllWindows()
else:
    print(res_classify)
    a=cv2.imread(path_classify)
    cv2.imshow('real note',a)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print("Real")
```

Watermark validation on the currency note is performed on the following code module.

```
path_wm=input()
res_wm=predict_image(path_wm,watermark_labels,watermark_interp
reter, watermark_input_details, watermark_output_details)
if res_wm.get("no_watermark") is not None:
    print(res_wm)
    a=cv2.imread(path_wm)
    cv2.imshow('fake note',a)
    cv2.waitKey(0)
    print("Fake")
    cv2.destroyAllWindows()
else:
    print(res_wm)
    a=cv2.imread(path_wm)
    cv2.imshow('real note',a)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print("Real")
```

UV image is analyzed for determining the presence of security thread to check whether the note is real.

```
path_uv=input()
res_uv=predict_image(path_uv,uv_labels,uv_interpreter,uv_input
_details, uv_output_details)
if res_uv.get("nopatch") is not None:
    print(res_uv)
    a=cv2.imread(path_uv)
    cv2.imshow('fake note',a)
    cv2.waitKey(0)
    print("Fake")
    cv2.destroyAllWindows()
else:
    print(res_uv)
    a=cv2.imread(path_uv)
    cv2.imshow('real note',a)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print("Real")
```

# 9 Results

This chapter briefly discusses about the results obtained after performing the proposed model. There will be three outputs for individual module in proposed system. Classification model determines the type of the currency. The uv model determines the availability of the security thread, whether it is dashed or continuous. If the note is identified as no patch currency it is determined as counterfeit. The watermark model determines the presence of Gandhi watermark, otherwise it will be considered as fake currency. In three modules three directories were created which contains test images to determine the accuracy of the models individually.

**Classification Model:**

Accuracy: 80.37

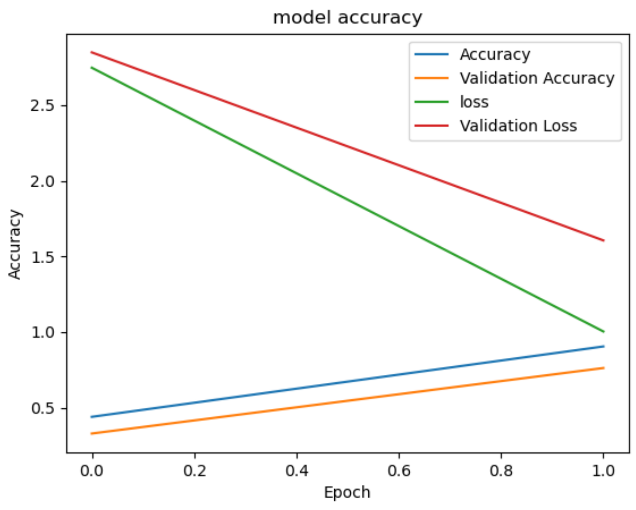


**Figure 9.1 Classification Model Accuracy**

```
D:\Bec\Projects\BEC_Project\4-2\Project\CNN-based-classification-(PRIMARY)\batch-test-images\2000back.jpg
{'2000_new_back_up': 38.54965567588806}
Real
```

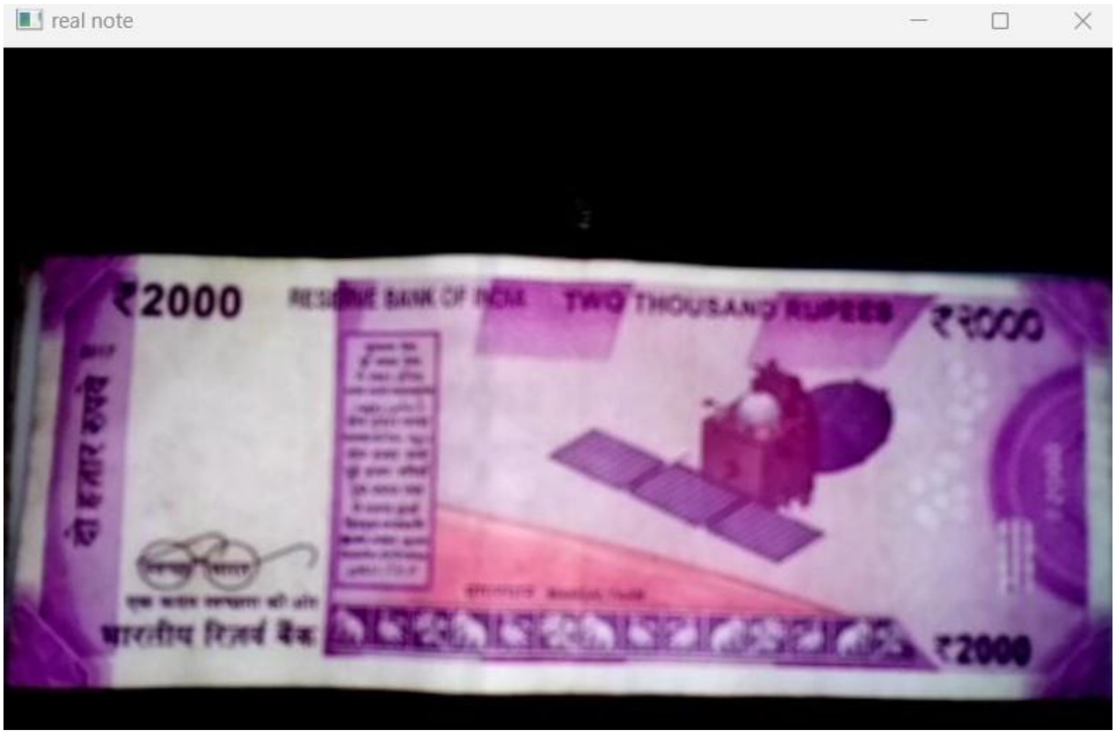


Figure 9.2 Classification of Real Note

```
D:\Bec\Projects\BEC_Project\4-2\Project\CNN-based-classification-(PRIMARY)\test-images\OT-2-11.jpg
{'invalid': 11.735772341489792}
Fake
```

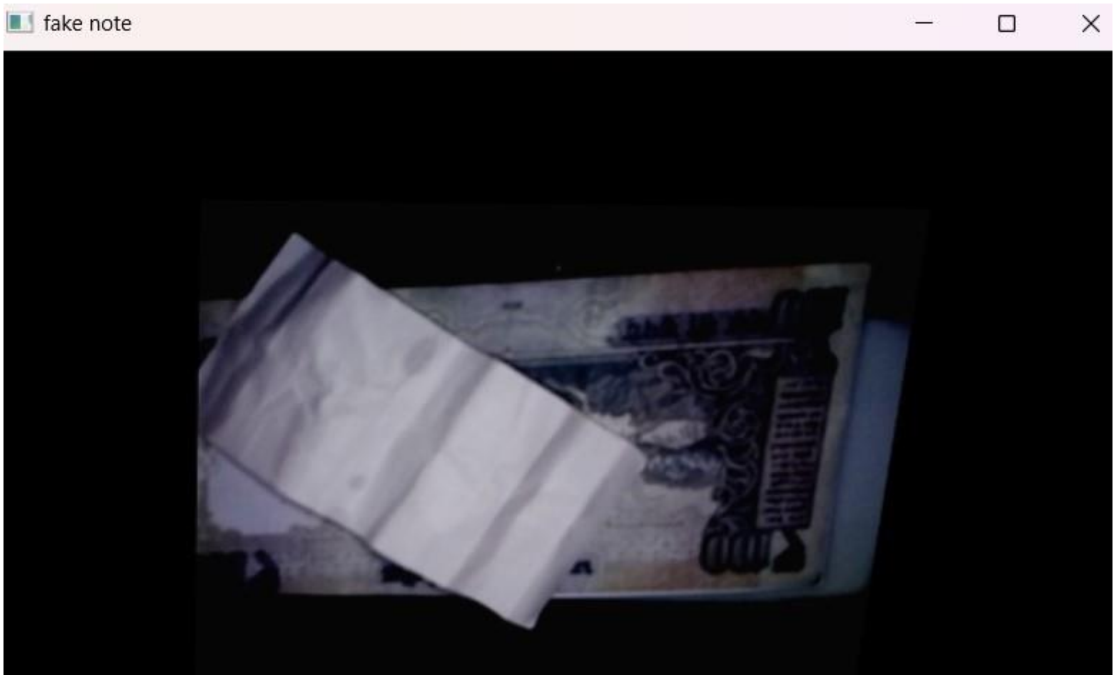


Figure 9.3 Classification of Fake Note

**Watermark Model:**

Accuracy: 91.25

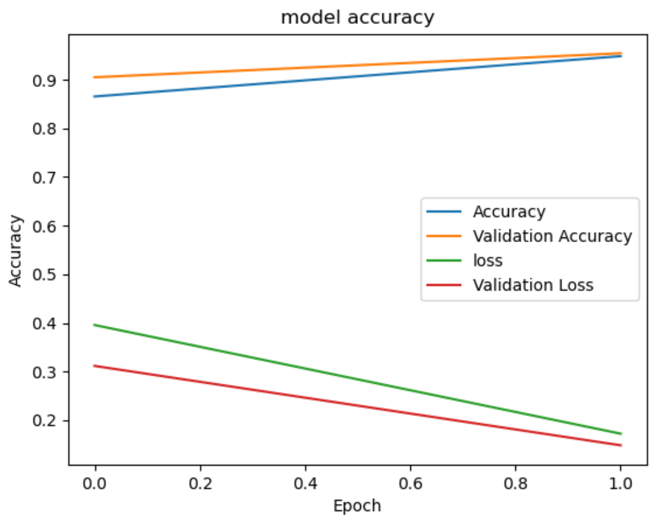
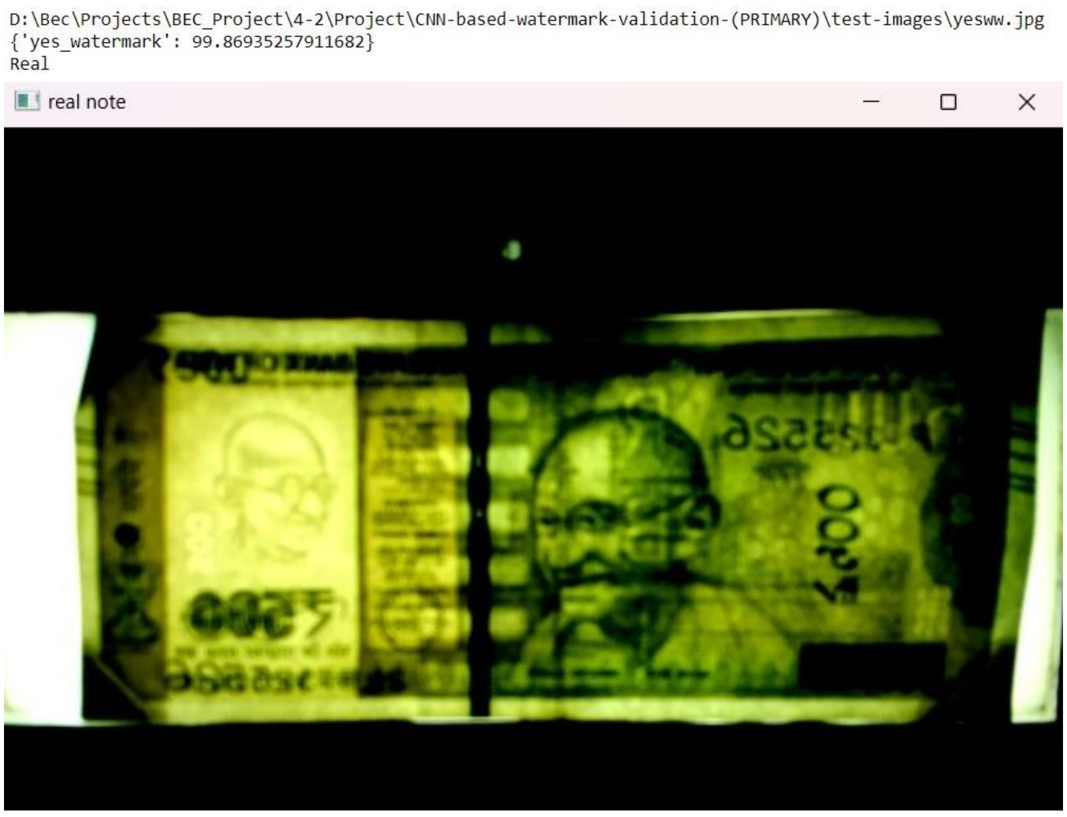
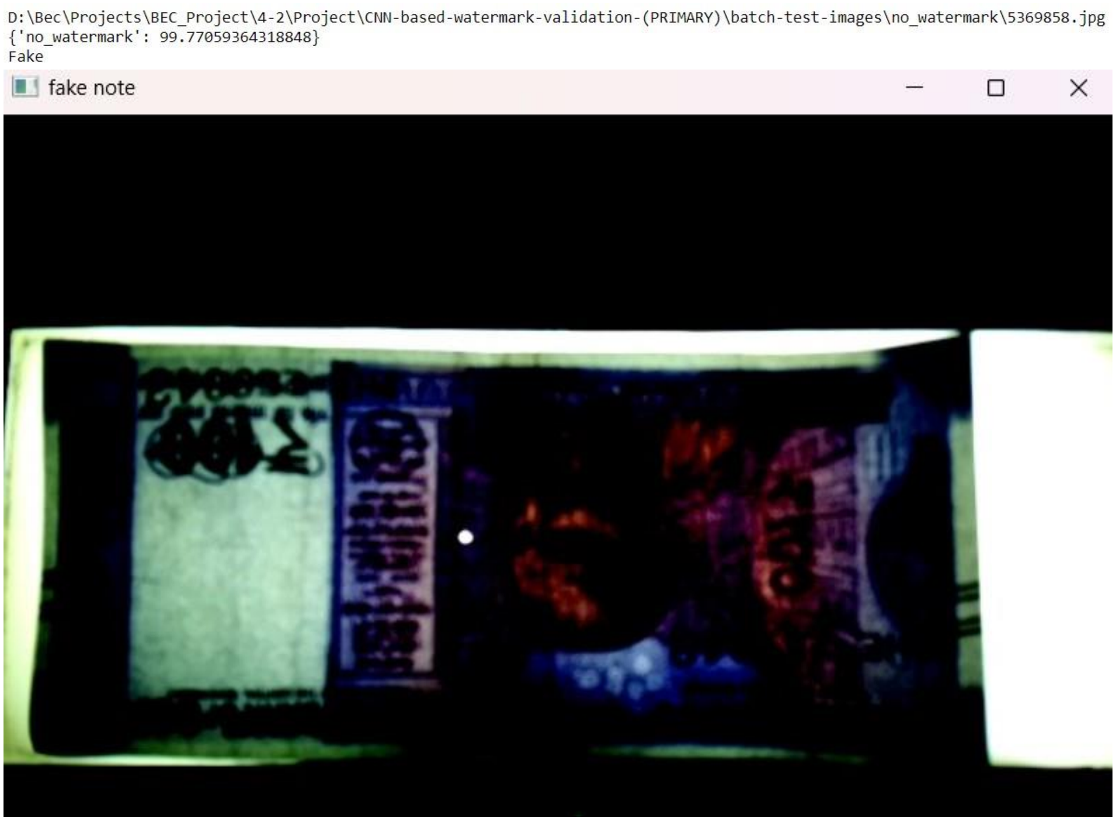


**Figure 9.4 Watermark Model Accuracy**



```
D:\Bec\Projects\BEC_Project\4-2\Project\CNN-based-watermark-validation-(PRIMARY)\test-images\yesww.jpg
{'yes_watermark': 99.86935257911682}
Real
```

**Figure 9.5 Watermark Detection for Real Note**



```
D:\Bec\Projects\BEC_Project\4-2\Project\CNN-based-watermark-validation-(PRIMARY)\batch-test-images\no_watermark\5369858.jpg
{'no_watermark': 99.77059364318848}
Fake
```

**Figure 9.6 Watermark Detection of Fake Note**

## UV Model:

Accuracy: 96.96

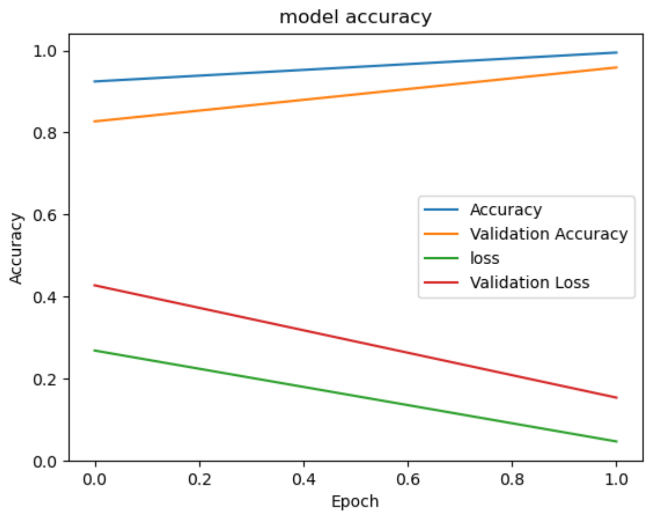
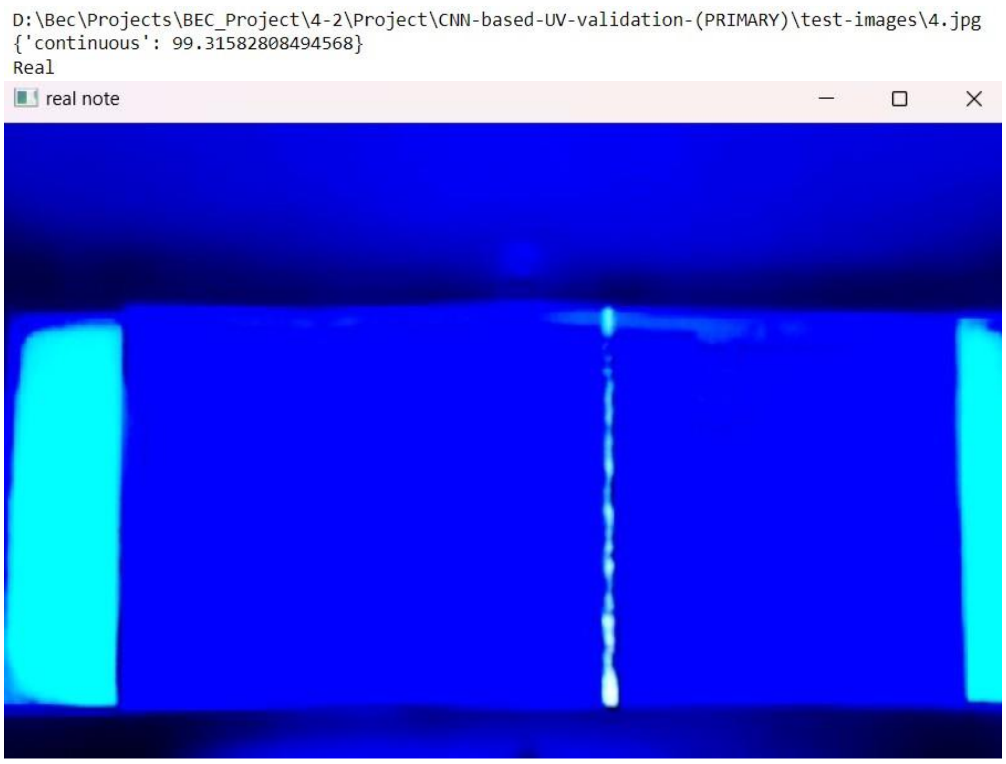
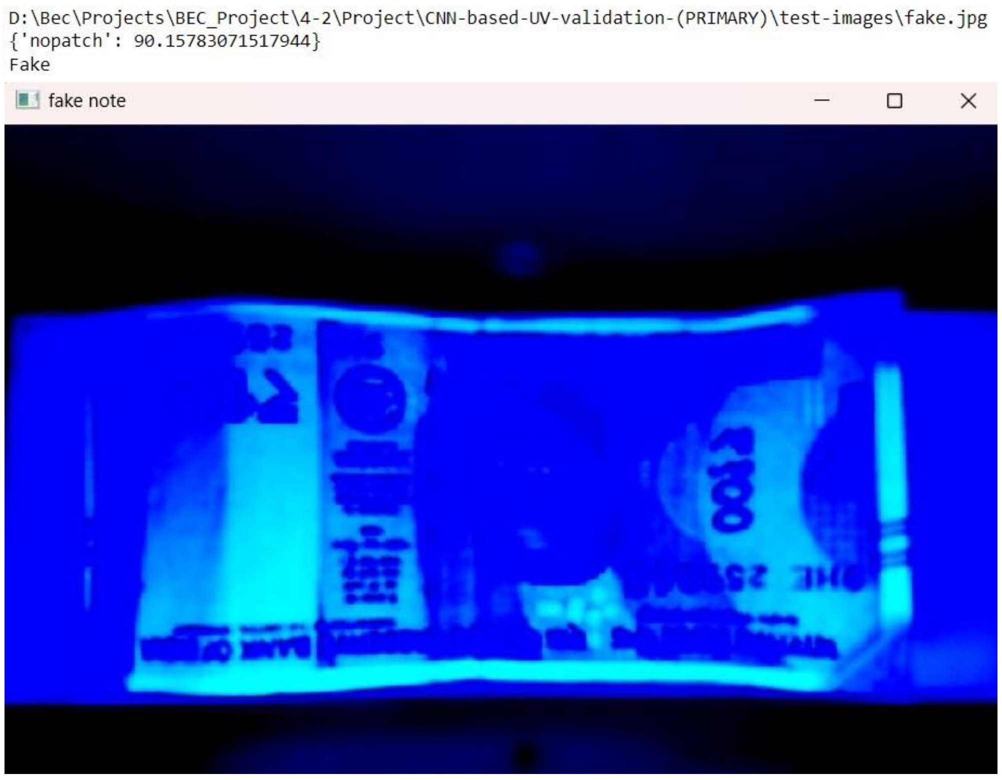


**Figure 9.7 UV Model Accuracy**



```
D:\Bec\Projects\BEC_Project\4-2\Project\CNN-based-UV-validation-(PRIMARY)\test-images\4.jpg
{'continuous': 99.31582808494568}
Real
```

Figure 9.8 Security Thread detection for Real Note



```
D:\Bec\Projects\BEC_Project\4-2\Project\CNN-based-UV-validation-(PRIMARY)\test-images\fake.jpg
{'nopatch': 90.15783071517944}
Fake
```

Figure 9.9 Security Thread Detection for Fake Note

# 10 Conclusion and Future Work

The efficient method of determining the authenticity of currency notes is presented in the proposed model. There are several security features on the Indian currency note, but in the proposed work only two security features were considered. Gandhi watermark and Security thread were considered for determining authenticity of the currency note. Since the monetary property highlights are discovered layer by layer, the discovery precision is often great.

As a result, the various strategies presented in this research were effectively implemented and tested by experiments on the model. Using the modules, Convolutional Neural Network was shown to be the optimal feature for performing the approach. By doing model classification, we were able to attain an 80% accuracy rate. For other two modules we were able to attain more than 90% accuracy rate.

Technology is advancing at a rapid pace these days. The proposed technique can be used to detect coins as well as recognise phoney currencies. Other countries' currencies can be added, and a comparison between them can be made. In the future, we could include all the security features of money by using a fair fundamental structure and providing sufficient preparation information. An application interface for mobile devices can be developed in future for making it available to the end user.

# 11 Bibliography

[1] P. C. More, M. Kumar and R. Singh, "Fake currency Detection using Basic Python Programming and Web Framework," *IRJET,* pp. 1-6, 2020.

[2] A. MR, K. C. P and Prajwa, " Identification of fake notes and denomination recognition," *IJIRSET,* vol. 11, no. 5, pp. 3-7, 2022.

[3] M. R. Pujar, "Indian Currency Recognition and Verification using Image," *IJARnD,* vol. 3, no. 2, 2021.

[4] Mayank, "Currency Counting Fake Note Detection," *SSRN,* pp. 2-5, 2018.

[5] V. K. Aman Bhatia, "Fake Currency Detection with Machine Learning Algorithm and Image Processing," *IEEE,* pp. 755-760, 2021.

[6] Reserve Bank of India, "Bank Notes," [Online]. Available: https://rbi.org.in/Scripts/ic_banknotessecurity.aspx.

[7] C.-F. Wang, "A Newbie's Introduction to Convolutional Neural Networks," Towards Data Science, 2018. [Online]. Available: https://towardsdatascience.com/what-is-a-neural-network-6010edabde2b.

[8] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks," Towards Data Science, 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way--3bd2b1164.

[9] S.-H. Tsang, "Review: MobileNetV2 — Light Weight Model (Image Classification)," Towards Data Science, 2019. [Online]. Available https://towardsdatascience.com/mobilenetv2-model-image-classification--8febb49.