Real Time Pakistani Currency Detection for Visually Impaired

Project Team

Muhammad Ammar Abid	p18-0016
Muhammad Zakaullah	p18-0059
Zohaib Nasir	p18-0065

Session 2018-2022

Supervised by

Dr. Omar Usman Khan

Department of Computer Science

National University of Computer and Emerging Sciences
Peshawar, Pakistan

June, 2022

Student's Declaration

We declare that this project titled "*Real Time Pakistani Currency Detection for Visually Impaired*", submitted as requirement for the award of degree of Bachelors in Computer Science, does not contain any material previously submitted for a degree in any university; and that to the best of our knowledge, it does not contain any materials previously published or written by another person except where due reference is made in the text.

We understand that the management of Department of Computer Science, National University of Computer and Emerging Sciences, has a zero tolerance policy towards plagiarism. Therefore, We, as authors of the above-mentioned thesis, solemnly declare that no portion of our thesis has been plagiarized and any material used in the thesis from other sources is properly referenced.

We further understand that if we are found guilty of any form of plagiarism in the thesis work even after graduation, the University reserves the right to revoke our BS degree.

Muhammad Ammar Abid Signature: _____

Muhammad Zakaullah Signature: _____

Zohaib Nasir Signature: _____

_____

Verified by Plagiarism Cell Officer

Dated:

Certificate of Approval

The Department of Computer Science, National University of Computer and Emerging Sciences, accepts this thesis titled *Real Time Pakistani Currency Detection for Visually Impaired*, submitted by Muhammad Ammar Abid (p18-0016), Muhammad Zakaullah (p18-0059), and Zohaib Nasir (p18-0065), in its current form, and it is satisfying the dissertation requirements for the award of Bachelors Degree in Computer Science.

Supervisor

Dr. Omar Usman Khan Signature: _____

Zeeshan Khan

FYP Coordinator
National University of Computer and Emerging Sciences, Peshawar

---

Dr. Hafeez Ur Rehman

HoD of Department of Computer Science
National University of Computer and Emerging Sciences

Acknowledgements

Final year spent on this project is finally over, there are some people we would like to mention as an expression of our gratitude. First and foremost, we are extremely grateful to our supervisor, Dr. Omar Usman for his invaluable guidance, continuous support, and leniency he provided us to explore and ask questions without hesitation. Thank you for taking out time from your really busy schedule. Secondly, Ms. Mashal for bearing with us and helping us out with hardware availability problems, Mr. Zeeshan Khan for his cooperation and Finally, We would like to express our gratitude to our parents for the tremendous understanding, encouragement, and opportunity they have given us.

Muhammad Ammar Abid

Muhammad Zakaullah

Zohaib Nasir

Abstract

In Pakistan, Visually impaired people face problems in their daily. We wanted to use our final year project as an opportunity to empower visually impaired community, somehow, if not completely. "Real-Time Pakistani Currency Detection for the Visually Impaired" was our basic proposal to cope with their problems regarding financial dealings. IOT based eyeglasses that can do currency classification and its counting -all in real-time is our scope and the IOT glasses will also count notes based on push button input. For this expedition, we will be using Raspberry Pi 4b, eyeglasses and Pi-camera for data acquisition. Detection and classification of currency will be done by using the MobileNetV3 model [1]. Throughout the process reality of user would be augmented through audio interaction.

Contents

List of Figures

List of Tables

Chapter 1

Preliminaries and Introduction

1.1 Introduction

Visually impaired person's life is really challenging. They are totally dependent on others even for very basic necessities. With all this tech-bloom going on, we believe we can empower them too, somehow if not thoroughly. The issue we are looking forward to addressing is their financial dealing dependence. They need someone to be there for them for payments otherwise they are totally dependent on the intentions of the shopkeeper and they have no other way to examine/inspect bills. So, we are proposing a device that will be able to recognize the bills and even count for them and give them denomination of currency amount in form of audio.

1.2 Motivation

In Pakistan, there are 1.12 million visually impaired people out of the 207.7 million population.[2] Microsoft, Google, and many companies have worked on many products to empower visually impaired people by solving their problems like including currency detection and classification but in Pakistan, the currency detection problem remains unsolved in large proportion as there are very few apps and products for the visually impaired community of Pakistan due to which people of this community face discrimination

Figure 1.1: Iot glasses developed

from other people. Keeping this in mind we propose an artificial intelligence-based IoT solution that can solve currency detection and counting problem for the visually impaired people of Pakistan empowering approximately one million members of the community. Our work is also targeting one Sustainable Development Goal of the United Nations which is Reducing Inequality as we are empowering visually impaired people of Pakistan to do financial dealings without depending on any other human.

1.2.1 SWOT Analysis

1.2.1.1 Strengths

Computer Science is the art of solving problems through programming computers efficiently and feasibly. This project enables us to do real-world problem solving as we are solving a problem that is largely untouched in Pakistan. A Disabled person has mismatched interactions with any type of environment. So every environment should be inclusive to all sorts of people. But many products are not made accessible for such people. This makes it harder for disabled people to enjoy the services provided by such prod-

ucts. Our IoT solution will be accessible to visually impaired people. We will achieve this by augmenting their reality with sound. Working for a community requires passion, selflessness, and commitment. Our product is designed especially for a largely discriminated community in Pakistan. Most of the products in the global market that are available nowadays require constant internet access. With the help of cloud technology, the companies deploy their models on the cloud and provide customers with the services through using API to the relevant model in the cloud, which means low storage consumption. But in Pakistan quality internet is a luxury. To solve this issue we would be deploying our models on the raspberry pi so there would be no need to connect with the internet which would prove efficient. Real-Time Currency Counting is one of the major strengths of our proposed IoT solution.

1.2.1.2 Weaknesses

Other companies like Microsoft, Google, etc have many products to empower visually impaired people in many problems, helping them in their daily life. If we launch our product in Pakistan's market, we would need to do more marketing because our product would have very little market reach but if the big tech companies launch a product solving the currency detection problem for visually impaired people in Pakistan, they would have a really big market reach due to which they could generate their revenue easily. Our product is dependent on Raspberry Pi so in the future if we sell our project as a product in Pakistan's market, we would need to bear with the expenses of buying all the single-board computers (Raspberry Pi) with enough resources to run our models so it can work better.

But as far as we are concerned, we don't want finance or anything. If we could make what we have visioned, It would be a complete product with other features like detecting surface defects, people's mood detection, etc. So, to cope with manufacturing costs, we will apply for funds or something like that It's just our way of giving back to community.

1.2.1.3 Opportunities

In Pakistan visually impaired face many problems in their daily life. First, they have the challenge to do their daily tasks without proper sight. Many technologies solve many of their problems. In Pakistan, visually impaired people face constant and daily discrimination from their peers, colleagues, family, neighbors, etc which gives them low self-esteem causing them to have no confidence in their intelligence which eventually leads them to depend more on technological resources to complete their daily life needs. Unfortunately in Pakistan, there are very few products for detecting and counting currency which makes visually impaired people dependent on others for fulfilling financial needs. This proves that our project has the potential to fill this hole and complete a need of the Pakistani visually impaired community. As thoroughly discussed our project also empowers the visually impaired community, lifting their burdens to depend on others for their financial dealings.

1.2.1.4 Threats

We would largely work with making our project accessible for the visually impaired community. But our IoT glasses would not be impossible to break, which makes them not that sustainable in long term. Our product's user may not be aware of his environment which could eventually cause our device to not detect notes correctly leading to no correct counting. This could happen because of too much dark area.

1.3 Problem Statement and Scope of Project

Keeping in mind the currency detection and counting problem faced by visually impaired people in Pakistan we propose IoT-based glasses for currency detection and counting. Our scope of the project is:

- Developing a machine learning based model which can detect Pakistani currency.
- Extending our code to count notes using the model.

- Deploying the model to Raspberry Pi based IoT solution
- Raspberry Pi would have push button-based input to initiate the process for detecting currency and counting through Pi Camera in Real-Time.
- Raspberry Pi would augment the reality of the visually impaired user by speaking denomination / counted amount and audio interaction with user.

Chapter 2

Review of Literature

2.1 Euro Banknote Recognition System for Blind People

Figure 2.1: IOT glasses

[3] An IT-based gadget containing Raspberry PI and Pi-NOIR camera specked with extra infrared light - implanted into pair of glasses to distinguish and perceive Euro banknotes for visually-challenged individuals. A cell phone is additionally utilized as an output device. A pi-NOIR camera would do data acquisition in real-time through voice commands and then data would be processed by raspberry-Pi and output would be forwarded to a smartphone for output in audio form.

Viola and Jones proposed Haar features that are being utilized for identification here. Rather than dissecting each picture pixel exclusively, the method is based on the classification of the integral images resulting in reduced processing time. Through Haar features,

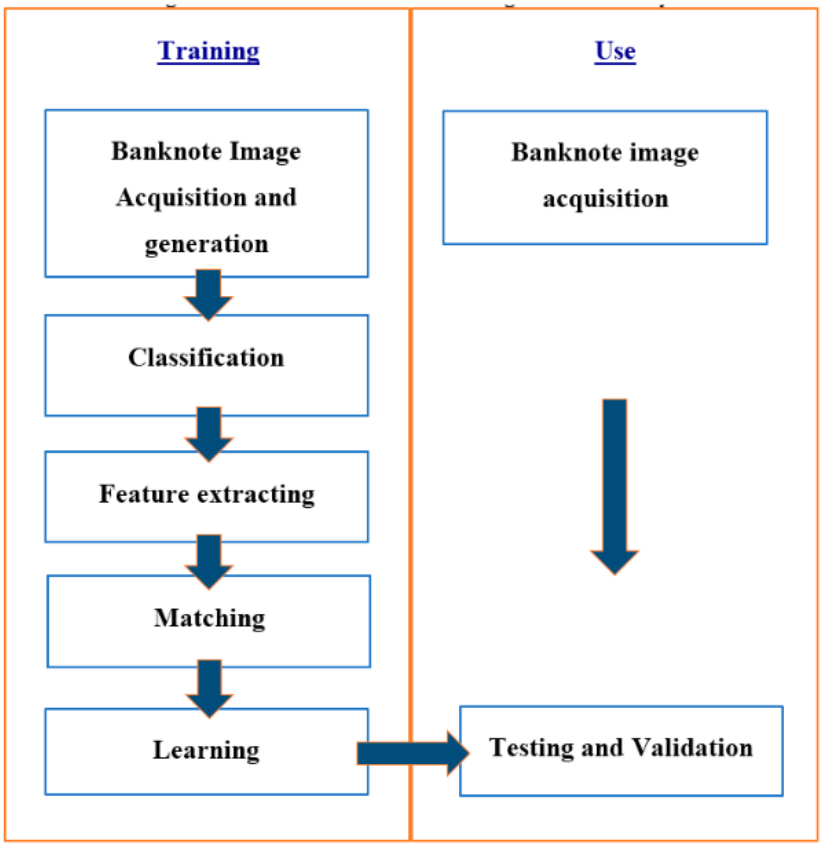

Figure 2.2: Block Diagram

a set of local features is extracted which are then classified with the AdaBoost algorithm. Essentially AdaBoost calculation is being utilized to recognize the Euro banknotes from complex pictures. It assigns a weight to each sample and selects the feature that best classifies. based on those weights. After detection, the Speed-Up Robust Features algorithm is used for classification.

Classification	Banknote correctly classified %	Banknote incorrectly classified %
Positive	76.05	23.95
Negative	8.25	91.75

Table 2.1: Results of euro banknote recognition

2.2 Currency detection and recognition based on deep learning

[4] In this paper, gathered data was basically frames from the video. 5 ,10, and 15 currency bills of New Zealand's currency were used -both front and back. Initially gathered data wasn't enough for Convolutional Neural Network so data was augmented to enhance dataset size.Single Shot MultiBox Detector (SSD) model based on deep learning as the framework is mainly used for detection while Convolutional Neural Network (CNN) model is used to extract the features of paper currency. Accuracy was used as a metric to determine usefulness of model which was up to 96.6

Model	main%	train%	valid%
b-1-w-01	96	96	95
b-1-w-10	96	96	94
b-2-w-01	93	94	87
b-2-w-10	92	95	84

Table 2.2: Results of deep learning

2.3 Indian Currency Note Denomination Recognition in Color Images

[5] Data acquisition was done using web-cam and a smartpone camera. For better performance, obtained images were enhanced using digital image processing techniques. Then, localization is done through Canny Edge Detection and with the help of color, currency bills were classified.

Accuracy	96%

Table 2.3: Results-Indian Currency Note Denomination Recognition in Color Images

2.4 Real-Time Object Detection Using Pre-Trained Deep Learning Models MobileNet- SSD

[6] In this paper, a method was created to identify objects utilizing the deep learning pre-trained model MobileNet. MobileNet is utilized for real-time detection and to distinguish between different objects in a webcam feed/video stream. MobileNet architecture uses depth-wise convolution and point-wise convolution concepts and requires diminished number of parameters, hence, low computational burden and better results. So MobileNet as a feature extractor and the SSD as detector framework is used here as a fast and efficient deep learning-based method for object detection. The outcomes were:

classes	Accuracy%
cars	99
persons	97
chairs	71

Table 2.4: Average Precision -MobileNet-SSD

2.5 MobileNet-Tiny: A Deep Neural Network-Based Real-Time Object Detection for Rasberry Pi

[7]

In this paper, MobileNetV3 is utilized for discovery and classification,It is prepared on the COCO dataset. MobileNetV3 is an endeavor to get continuous article discovery on non-GPU PCs and edge gadgets.. When running on a raspberry pi, it accomplishes a speed of 4.5 FPS. MobileNetV3 architecture uses convolutions of depth-wise and point-wise approach and requires diminished number of parameters, hence, low computational burden.Input data in form of pictures is gone through the convolution layers and Bottleneck Residual Blocks (BRB) to deliver a feature map. Then BRB layers 4,6 and 6 are gone through SSD-Lite predictor layers to create recognitions. Another layer, the

non-maximum suppression layer is there to channel these recognitions and produce rectangular boxes.

Methods	mAP%	FPS
YOLO(lite)	12	-
MobileNetV1	21	0.5
MobileNetV2	22	0.5
MobileNetV3	19	5

Table 2.5: Raspberry Pi's models performances (Dataset of COCO)

Methods	mAP%	FPS
YOLO(lite)	33	-
MobileNetV1	75	0.5
MobileNetV2	75	0.5
MobileNetV3	52	5

Table 2.6: Raspberry Pi's models performances (Dataset of VOC07)

$$\mu_b = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.1}$$

$$\sigma_b^2 = \frac{1}{m}\sum_{i=1}^{m} \left(x_i - \mu_b\right)^2 \tag{2.2}$$

$$x_i = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \varepsilon}} y_i = \gamma x_i + \beta \tag{2.3}$$

As batch-normalization occurs, If values of i over a mini-batch B is 1 to m, then batch, normalization-transforms can be computed through equations mentioned above and more.

Chapter 3

Methodology

3.1 Proposed Methodology

3.1.1 Detection and classification

Many top-notch algorithms for detecting objects in real time require greater GPU processing power currently, but still real time detection may not always happen.

3.1.1.1 MobileNetV3

We are looking forward to use MobileNetV3 for detection and classification. MobileNetV3 is for the purpose to be used on computer, ARM devices having no GPUs. MobileNetV3 can easily run in embedded systems and edge devices. MobileNet introduced convolu-

Method	mAP	FPS	batch-size	Boxes
FasterRCNN	73.2	7	1	600
FastYOLO	52.7	155	1	98
YOLO	66.4	21	1	98
SSD300	74.3	46	1	8732
SSD512	76.8	19	1	24564
SSD300	74.3	46	8	8732
SSD512	76.8	19	8	24564

Table 3.1: Results of SSD on VOC-2007 dataset

tions having depth-wise approach and point-wise approach, resulting in reduced number of parameters, hence, low computational burden. Input images are passed through the convolution layers and Bottleneck Residual Blocks (BRB) to produce a feature map. Then BRB layers 4,5 and 6 are passed through SSD-Lite predictor layers to produce detections. Another layer, the non-maximum suppression layer is there to filter these detections and produce bounding boxes.

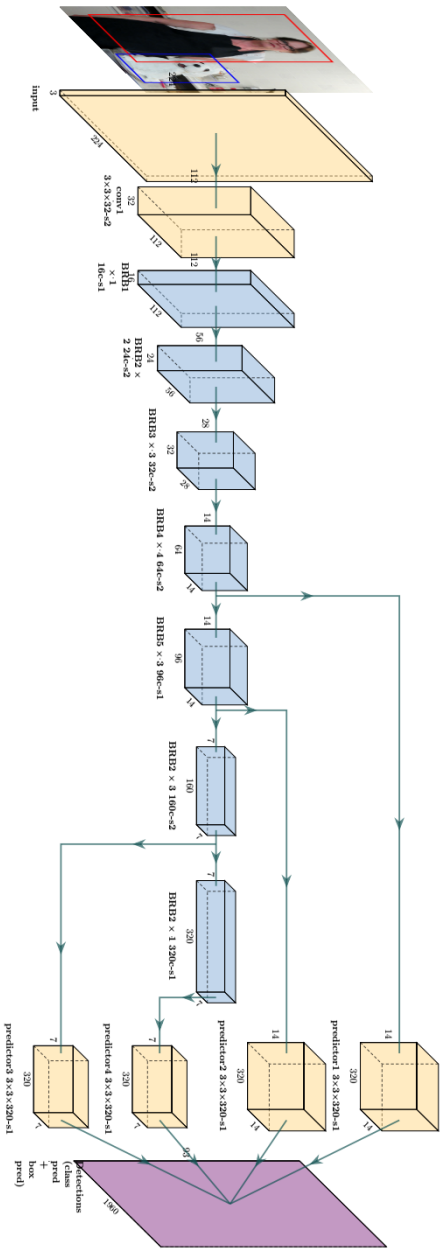

Figure 3.1: Mobilenetv3 model

3.1.2 System Analysis and Design: UML-Diagrams

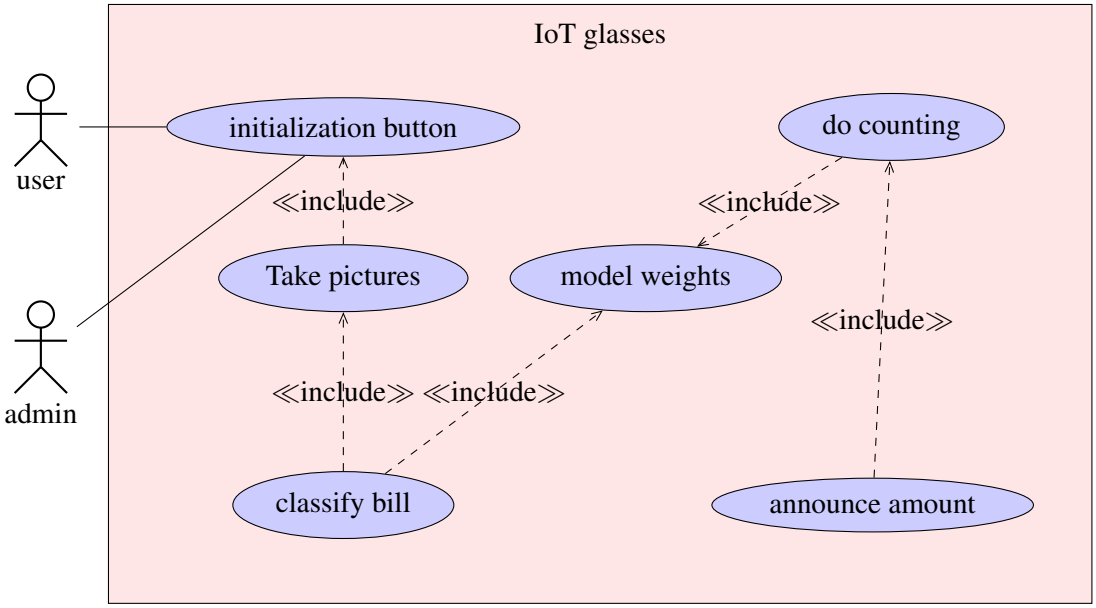

Figure 3.2: Use-case Diagram

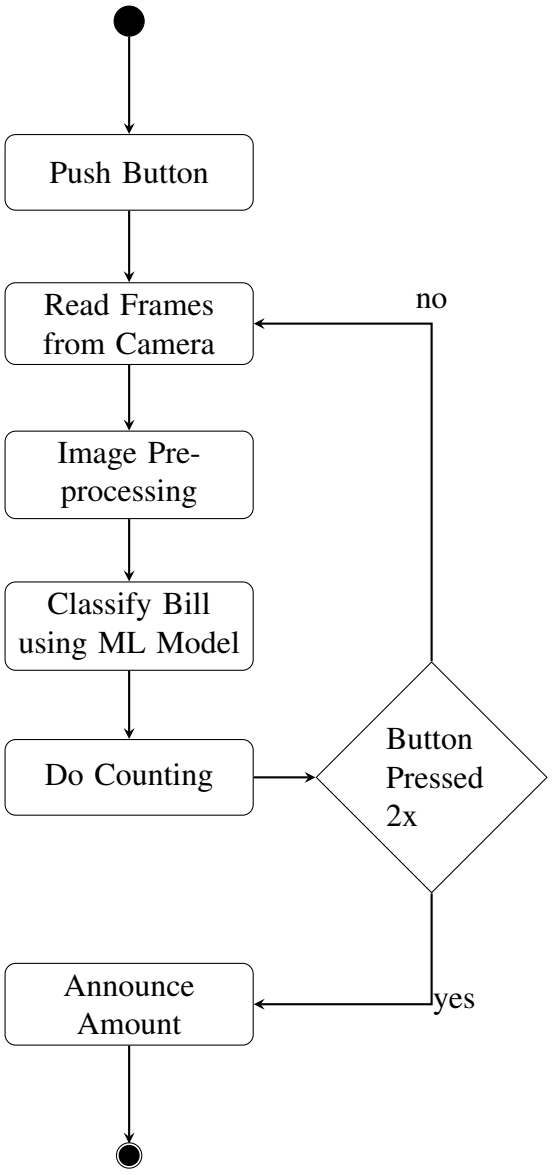

Figure 3.3: Activity Diagram

3.2 Implementation

3.2.1 Data Acquisition

We are using Deep Learning based approach to train our classifier to detect Pakistani currency in an image. For this we need at least 25, 000 images to train our Deep learning based detector according to Andrew Ng [8]. For this we searched through various online platforms like Kaggle, github, opendata.com.pk but we were unable to find sufficient amount of data of Pakistani Currency. So as a result we decided to collect dataset ourselves.

3.2.1.1 Microsoft's Seeing AI Currency Dataset Acquisition Criteria

[1]One of the state of art solutions currently to assist visually impaired people is Microsoft's Seeing AI. It offers many services for free. One service worth mentioning is the Currency Detection service in Seeing AI. They have proposed a criteria to collect currency dataset for purpose of currency detection. They have proposed that currency dataset must include:

1. The currency note should occupy at least 1/6 of the whole image.
2. The currency note should be displayed from different angles in the image
3. The currency note should be present in various locations in the image (top left corner, middle, bottom right corner, etc.)
4. There should be some foreground objects covering part of the currency (no more than 40
5. The background should be as diversified as possible

We followed this criteria and collected almost 4,610 images of Pakistani currency and background. We decided the data into total 15 classes. These classes are:

- 10_front

- 10_back
- 20_front
- 20_back
- 50_front
- 50_back
- 100_front
- 100_back
- 500_front
- 500_back
- 1000_front
- 1000_back
- background

3.2.1.2 Why Background class?

[1]Basically localization of where a note is present is present in an image involves creating bounding boxes. But our target is to assist a visually impaired person to detect note. There is no benefit in our project to use bounding boxes for localizing note in an image. So that's why we have created a separate class of "background" to classify background images. We are just classifying an image/frame into one of the above mentioned classes and we are using "background" class because if we don't use "background" class then if a human comes before camera, human will also be classified as one of the notes, so to handle this scenario we are using "background" class.

3.2.2 Rescaling image to 20% of its actual size to reduce size and train-val-test split

Basically all of the 4,610 images were of 11GB because they were taken from high resolution mobile phones. To reduce the size we wrote few scripts to reduce the size of each image to only 20% of its actual size. This way data 286.9MB only.

Listing 3.1: rescale.py

```
from sys import argv
from skimage.transform import rescale
import matplotlib.pyplot as plt
from matplotlib.pyplot import imread, imsave

orig_image_path = argv[1]
rescaled_image_path = argv[2]

image = imread(orig_image_path)
rescaled_image = rescale(image, 0.2, multichannel=True)
#print(rescaled_image.shape)
imsave(rescaled_image_path, rescaled_image)
```

Listing 3.2: rescale-images.sh

```
#!/bin/bash

for note in $(ls data/$1); do
   $(python3 rescale.py data/$1/$note data-rescaled/$1/$note)
done
```

Listing 3.3: parallel-commands.sh

```
#!/bin/bash

for cmd in "$@"; do {
```

```
  echo "Process \"$cmd\" started";
  $cmd & pid=$!
  PID_LIST+=" $pid";
} done

trap "kill $PID_LIST" SIGINT

echo "Parallel processes have started";

wait $PID_LIST

echo
echo "All processes have completed";
```

Listing 3.4: run-rescale-parallel.sh

```
#!/bin/bash

./parallel-commands.sh "./rescale-images.sh 10_front" "./rescale-images.sh
    10_back" "./rescale-images.sh 20_front" "./rescale-images.sh 20_back"
    "./rescale-images.sh 50_front" "./rescale-images.sh 50_back"
    "./rescale-images.sh 100_front" "./rescale-images.sh 100_back"
./parallel-commands.sh "./rescale-images.sh 500_front"
    "./rescale-images.sh 500_back" "./rescale-images.sh 1000_front"
    "./rescale-images.sh 1000_back" "./rescale-images.sh 5000_front"
    "./rescale-images.sh 5000_back"
./parallel-commands.sh "./rescale-images.sh background"
```

To start the procedure, we executed ./run-rescale-parallel.sh inside the data folder with all other scripts present. This executes codes in parallel for all classes to rescale the images present in that class. The new dataset would be in data-rescaled directory. Now to distribute the dataset into train-val-test, first we created a directory data-rescaled-split then we wrote following scripts:

Listing 3.5: make-val-test.py

```
#!/bin/python3
from os import listdir, system
import random
from sys import argv # argv[1] -> test or val
                # argv[2] -> split percentage like 0.1, 0.3

def split(test_val, percent, notes, currency_side):
   rand_img_to_copy = int(len(notes) * float(percent)) # converting
       argv[1] into float because its a str
   rand_img_nums = []

   while len(rand_img_nums) != rand_img_to_copy:
      rand_num = random.randint(0, len(notes)-1)
      if rand_num in rand_img_nums:
         continue

      img = notes[rand_num]
      system('mv data-rescaled-split/train/{}/{}
          ./data-rescaled-split/{}/{}/'.format(currency_side, img,
          test_val, currency_side))
      rand_img_nums.append(rand_num)

for currency_side in listdir('data-rescaled'):
   notes = listdir('data-rescaled-split/train/{}'.format(currency_side))
   split(argv[1], argv[2], notes, currency_side)
```

Listing 3.6: run-split-test-val.sh

```
#!/bin/bash

$(./make-val-test.py test 10)
$(./make-val-test.py val 10)
```

Executing ./run-split-test-val.sh would split the data present in data-rescaled into train, val, test directories inside data-rescaled-split directory. Each of these directories would have directories of our classes

3.2.3 Transfer Learning on MobileNetV3 model along with Data Augmentation

We have collected 4,610 total images, but deep learning requires large amount of data like at least 25,000 images according to Andrew Ng. To overcome this problem we have researched two techniques and are using these in combo for better results.

1. Data Augmentation
2. Transfer learning

First importing libraries and doing some initializations. Notice that we are also using tensorboard for logs to visualize results later.

```
from tensorflow.keras import applications
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_dir =
    '/home/ammar/Desktop/U/semester-8/fyp/data-rescaled-split/train'
test_dir = '/home/ammar/Desktop/U/semester-8/fyp/data-rescaled-split/test'
val_dir = '/home/ammar/Desktop/U/semester-8/fyp/data-rescaled-split/val'
visualization_dir =
    '/home/ammar/Desktop/U/semester-8/fyp/data-rescaled-split/augmented' #
    For storing augmented images generated in a directory
log_dir = '/home/ammar/Desktop/U/semester-8/fyp/data-rescaled-split/logs'
    # For stroing tensorboard logs
```

```
image_height = 224
image_width = 224
batch_size = 32
epochs = 10
```

3.2.3.1 Data Augmentation

[9]Data Augmentation is basically a method to increase the amount of data by creating synthetic data from original data as much as we want and as much operations are applied. For this purpose we use tensorflow. Each of the operations will be applied to an image to create a new synthetic image. We are doing following operations to create new images:

- Zooming operation with Range 20% of image
- Width Shift with Range 20% of image
- Height Shift with Range 20% of image
- Shear with Range 20% of image
- Rotation with Range 180 Degrees of image

One of these operations will be applied randomly with random range (upto given maximum range) to each image to create a new synthetic image. Longer the epochs (iterations), more would be the data created. These ranges for operations were handpicked after many experimental trainings. Ranges bigger with the above mentioned ranges for the operations will bring bad results e.g, Width Shift with Range 30% will shift the image upto 30% from left side but if the note is present in top left corner then the new image won't have note present in it at all and the model would learn that background is a note which would be a false positive so that's why the above mentioned ranges for each of the operations are carefully hand picked according to our dataset.

```
train_datagen = ImageDataGenerator(
    fill_mode="nearest", # If any pixels are empty should fill using this
        method
```

```
    zoom_range=0.2,      # Number greater than 0.2 would give unwanted
        augmented images
    width_shift_range = 0.2, # Number greater than 0.2 would give unwanted
        augmented images
    height_shift_range = 0.2, # Number greater than 0.2 would give
        unwanted augmented images
    shear_range = 0.2,       # Number greater than 0.2 would give unwanted
        augmented images
    rotation_range=180
)


validation_datagen = ImageDataGenerator(

)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    save_to_dir = visualization_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode="categorical") # As we want multuple categories

validation_generator = validation_datagen.flow_from_directory(
    val_dir,
    target_size=(image_height, image_width),
    class_mode="categorical")


tensorboard = TensorBoard(log_dir=log_dir) # Tensorboard callback used for
    plotting
```

3.2.3.2 Transfer Learning

Transfer learning is a method in machine learning where we use a pretrained model for a new task. Basically we are transferring the knowledge obtained in one domain to another domain. For this purpose we use tensorflow. We are using a pretrained MobileNetV3 trained on imagen data comprising of 1000 classes. We would be freezing all of the layers of model except the last 10 layers. Freezing the layers means that we won't be updating the weights of those layers while training on our data. We are doing this because top layers learn broad level features e.g, door has a rectangle shape. The lower layers learn complex and detailed features e.g, learning how a door knob looks. We need to retain the knowledge of broad level features but want the model to learn detailed features of Pakistani currency data so thats why we are Freezing top layers. In the code below that we are intentionally not doing any data augmentation on validation data as its the required method.

```
def build_model():
    # constructing the model
    model = applications.MobileNetV3Large(weights="imagenet",
        include_top=False, input_shape=(image_width, image_height, 3),
                                pooling='avg')

    # We don't want 10 layers from last to be frozen, practice of transfer
        learning
    for layer in model.layers[:-10]:
        layer.trainable = False

    # Adding custom Layers
    x = model.output

    predictions = Dense(15, activation="softmax")(x) # Connecting our
        required output layer with the MobileNet

    # creating the final model
```

```
    model_final = Model(inputs=model.input, outputs=predictions)

    return model_final

model_final = build_model()

# compile the model
model_final.compile(loss="categorical_crossentropy",
                    optimizer=optimizers.Adam(learning_rate=0.001), # Adam
                        gives better results on this learning rate
                    metrics=["accuracy", "Precision"])

# Train the model
model_final.fit(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    callbacks=[tensorboard])
```

3.2.4 Evaluating on Test data

We splitted the 10% into validation data to visualize overfitting later, after training the model on certain epochs we would be testing the model on 10% unseen data i.e test data. Notice that we are also doing doing some data augmentation on test data.

```
# Now evaluating our model on test data
test_datagen = ImageDataGenerator(
    fill_mode="nearest",
    zoom_range=0.3,
    rotation_range=30
)
```

```
test_generator = test_datagen.flow_from_directory(
                    test_dir,
                    target_size=(image_height, image_width),
                    class_mode="categorical")

model_final.evaluate(test_generator)

model_final.save('model.pb')
```

We are saving the model in .pb format as its the standard format for saving those models which would later be optimized for deployment on mobile phone devices or Raspberry pi

3.2.5 Converting the model to lightweight Tensorflow Lite model for optimized deployment

[10]Tensorflow Lite is a set of techniques and tools that enables machine learning on-device by enabling developers to run their models on mobile, embedded and edge devices. Tensorlfow lite has many key features. Features worth mentioning here are its size(reduced model and binary size) and power consumption(efficient inference). A Tensorflow lite model is basically represented in an efficent portable format known as FlatBuffers. FlatBuffers is a an efficient cross platform serialization support. Tensorflow lite uses a custom memory allocator for execution latency and minimum load. MobileNetV3 is supported in Tensorflow Lite. The code for conversion of our trained model into tflite model is:

```
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_keras_model(model_final)
tflite_model = converter.convert()
with open('model.tflite', 'wb') as f:
    f.write(tflite_model)
```

3.2.6 Raspberry Pi codes

We used hardware:

- Raspberry Pi 4B
- Pi camera v1.3 5MP
- Raspberry Pi Case
- Handsfree with 3.5mm audio jack
- Black shaded glasses

[11]We deployed Raspberry Pi bullseye OS 2020. We did not update the OS with latest updates because our Pi Camera was not supported in it, so we switched to older version of bullseye OS.

3.2.6.1 Reading Real Time Frames using Pi Camera

We used Pi camera version 1.3 for our Raspberry Pi which was later attached to black shaded glasses. We uses python on Raspberry Pi to access the camera. Camera was 5MP so model's performance was not that much good but better performance can be achieved with 8MP camera. Code for capturing frame in real time in raspberry Pi is:

```
import picamera
import numpy as np

with picamera.PiCamera() as camera: # So camera gets closed when required
   camera.resolution = (640, 480) # Setting up fixed resolution

   while True:
      image = np.empty((640, 480, 3), dtype=np.uint8)
      camera.capture(image, 'rgb') # image would have the current frame
```

We need to capture the frame with each pixel values of type np.uint8, if used a different type pixel values would not be in range 0-255, which would devastate model predictions later. We also did not use Open CV on raspberry pi because of large dependency. Instead we utilized the built in PiCamera library for python in Raspberry Pi.

3.2.6.2 Predicting notes using tflite model

Now after reading real time frames from camera we need to predict one of the 15 classes. For this purpose we would deploy tflite model on Raspberry Pi. We would be only using tflite interpreter instead of whole Tensorflow dependency [12]. tflite interpreter only has the interpreter for inference. We would get the classes to indices mapping from train_generator. Our tflite model requires input frame to be of "RGB" as we trained our model also on "RGB" images. We need to go low level and set input of the tflite model ourselves. We then invoke the model and get the output class with maximum probability (as we are using softmax layer as output layer). Notice that we are converting the frame type into np.float32. We are doing this because our tflite model requires input frame's each pixel value type to be float32. We also can't do this while capturing frame because if we change type of frame while capturing the image, results would be disastrous and each pixel would be greater than 1Million which is very bad pixel value. That's why we are first capturing the frame in np.uint8 as pixel values can be represented in 8 bits and then converting it into np.float32 for required model input [10].

```
import picamera
from tflite_runtime.interpreter import Interpreter
import numpy as np


classes_to_inds = { '1000_back': 0,
                    '1000_front': 1,
                    '100_back': 2,
                    '100_front': 3,
                    '10_back': 4,
                    '10_front': 5,
```

```
                    '20_back': 6,
                    '20_front': 7,
                    '5000_back': 8,
                    '5000_front': 9,
                    '500_back': 10,
                    '500_front': 11,
                    '50_back': 12,
                    '50_front': 13,
                    'background': 14 }
inds_to_classes = {ind:currency_side for currency_side,ind in
    classes_to_inds.items()}


# Classifies a frame
def classify_frame(frame, interpreter):
   frame = np.expand_dims(frame, axis=0)

   # Run inference
   interpreter.set_tensor(input_details[0]['index'], frame)
   interpreter.invoke()

   probs = interpreter.get_tensor( output_details[0]['index'] )[0]
   print(probs)

   class_ind = np.argmax(probs)
   class_name = inds_to_classes[class_ind]

   print(class_ind, class_name, '\n')
   return class_name


model_path = './model.tflite'
height, width = 224, 224
```

```
# Initialize our model tflite interpreter
interpreter = Interpreter(model_path)
interpreter.allocate_tensors()

# Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

with picamera.PiCamera() as camera:
   camera.resolution = (height, width)
   camera.start_preview()

   # Start Reading frames
   while True:
      # Input frame
      frame = np.empty((width, height, 3), dtype=np.uint8)
      camera.capture(frame, 'rgb')
      np.set_printoptions(suppress=True)

      # Classify
      class_name = classify_frame(frame.astype(np.float32), interpreter)

      # Display class name
      camera.annotate_text = "{}".format(class_name)
```

3.2.6.3 Algorithm for Counting Total Amount of notes

Our previous logic only detects notes and then informs the detected / predicted note according to class values. But we are extending this logic to not only detect a note but also count amount of notes till the user wants. For this first we came up with a suitable algorithm for counting total amount of notes. This is not easy as it sounds. Limitations

are:

- Cannot count each frame
- Each currency detected should be accurate
- Less precise results for currency detection would give disastrous counting amount
- Should hold note for more than 1 second (depends on fps)
- If model predicts an outlier for few frames, should ignore
- Majority vote note predictions in a window of frames for better confidence

Based on this we developed logic as:

```
...
from collections import Counter
...
...
# Returns currency denomination e,g passing "50_back" would return 50
def denomination(currency_side):
    dash = currency_side.index('_')
    return int(currency_side[:dash])

# Majority votes
def majority_voting(frames_preds):
    majority = Counter(frames_preds.split(' '))
    return majority.most_common()[0][0]

# Count algorithm
def count(frames_preds):
    # Splitting different classes predictions based on intervals
    frames_preds_intervals = frames_preds.split(' ')
    # Discarding those frame predictions which were not constant for at
        least 5 frames
```

```
    reliable_frames_preds = list( filter(lambda x: len(x.split(' ')) > 5
        ,frames_preds_intervals) )
    # Discarding those frames which were predicted background
    notes_preds_frames = list( filter(lambda x: 'background' not in x,
        reliable_frames_preds) ) # Filtering classes that were not
        background

    # Majority voting the frames predictions
    currencies_sides = list( map(lambda x: majority_voting(x),
        notes_preds_frames) )

    # Getting only the denomination as 50 is the information needed if
        note predicted is either 50_back or 50_front
    currencies = list( map(lambda x: denomination(x), currencies_sides) )

    # We got our notes, just sum them
    count = sum(currencies)

    return count
```

In our main loop where are reading frames we do

```
prev_pred = ''          # Necessary for counting to work
frames_preds = ''       # Necessary for counting to work
...
...
            # Counting Logic
            if class_pred != prev_pred:
               frames_preds += ' ' + class_pred
            else:
               frames_preds += ' ' + class_pred
            prev_pred = class_pred
...
```

```
...
count(frames_preds)
```

3.2.6.4 Push Button code

Our main aim for building an IOT device is to assist those visually impaired people who are not trained with accessible technologies, so for this purpose We are using Push button to control the start and end of the procedure. To start user should press button 1 time and to end the process user should press button 2 times. We connect push button's one leg to gpio pin 1 for 3.3V and other to gpio pin 10 for input signal purpose. We attach an event handler to pin 10, so each time button is pressed and a signal is generated, we increment a variable button_pressed and using that variable we can control our overall logic. Code is:

```
import RPi.GPIO as gpio
...
...
gpio.setmode(gpio.BOARD) # Use physical pin numbering
gpio.setup(10, gpio.IN, pull_up_down=gpio.PUD_DOWN) # set pin 10 to be an
    input pin and set initial value to be pulled low (off)
...
...
button_pressed = 0
def button_callback(channel):
   global button_pressed
   button_pressed += 1

gpio.add_event_detect(10, gpio.RISING, callback=button_callback) # Setup
    event on pin 10 rising edge
...
...
gpio.cleanup()
```

3.2.6.5 Augmenting Reality of visually impaired user using Audio

We would be using Audio to augment the reality of visually impaired user for live interaction. This would improve user experience and would satisfy needs of visually impaired user. For this purpose we need a Text to Speech Engine which can be used to convey any message to user. We are prioritizing an offline Text to Speech engine here as we want user to be independent on internet as much as possible, and want everything to be offline. The code mainly used for every text to speech message is:

```
import pyttsx3
...
...
# Text to speech for augmenting reality of user
def speak_sentence(sentence):
   engine = pyttsx3.init()
   engine.say(sentence)
   engine.runAndWait()
...
...
```

3.2.7 Overall Deployed code

Basically we have created three files for our overall deployed code. One file is "main.py" with all of main code, second file is "utils.py" which has all of necessary helper functions for process to succeed. Third file is "model.tflite" which has our trained MobileNetV3 model in Tensorflow lite format.

Listing 3.7: utils.py

```
import numpy as np
import pyttsx3
from collections import Counter

classes_to_inds = { '1000_back': 0,
```

```
                    '1000_front': 1,
                    '100_back': 2,
                    '100_front': 3,
                    '10_back': 4,
                    '10_front': 5,
                    '20_back': 6,
                    '20_front': 7,
                    '5000_back': 8,
                    '5000_front': 9,
                    '500_back': 10,
                    '500_front': 11,
                    '50_back': 12,
                    '50_front': 13,
                    'background': 14 }
inds_to_classes = {ind:currency_side for currency_side,ind in
    classes_to_inds.items()}


# Classifies a frame
def classify_frame(frame, interpreter, input_details, output_details):
   frame = np.expand_dims(frame, axis=0)

   # Run inference
   interpreter.set_tensor(input_details[0]['index'], frame)
   interpreter.invoke()

   probs = interpreter.get_tensor( output_details[0]['index'] )[0]

   class_ind = np.argmax(probs)
   class_name = inds_to_classes[class_ind]

   print(class_ind, class_name, '\n')
   return class_name
```

```
# Returns currency denomination e,g passing "50_back" would return 50
def denomination(currency_side):
    dash = currency_side.index('_')
    return int(currency_side[:dash])

# Majority votes
def majority_voting(frames_preds):
    majority = Counter(frames_preds.split(' '))
    return majority.most_common()[0][0]

# Count algorithm
def count(frames_preds):
    # Splitting different classes predictions based on intervals
    frames_preds_intervals = frames_preds.split(' ')
    # Discarding those frame predictions which were not constant for at
        least 20 frames
    reliable_frames_preds = list( filter(lambda x: len(x.split(' ')) > 5
        ,frames_preds_intervals) )

    # Discarding those frames which were predicted background
    notes_preds_frames = list( filter(lambda x: 'background' not in x,
        reliable_frames_preds) )

    # Majority voting the frames predictions
    currencies_sides = list( map(lambda x: majority_voting(x),
        notes_preds_frames) )

    # Getting only the denomination as 50 is the information needed if
        note predicted is either 50_back or 50_front
    currencies = list( map(lambda x: denomination(x), currencies_sides) )
```

```
    # We got our notes, just sum them
    count = sum(currencies)

    return count

# Text to speech for augmenting reality of user
def speak_sentence(sentence):
   engine = pyttsx3.init()
   engine.say(sentence)
   engine.runAndWait()
```

Listing 3.8: main.py

```
import picamera
import RPi.GPIO as gpio
import numpy as np
from tflite_runtime.interpreter import Interpreter
from utils import *


# Initializing important variables
model_path = './model.tflite'
prev_pred = ''          # Necessary for counting to work
frames_preds = ''       # Necessary for counting to work
start_limit = 1         # How many times should button be pressed to start
end_limit = 2           # How many times should button be pressed to end
button_pressed = 0      # Keeps track of how many times button has been
    pressed
height, width = 224, 224



# Initialize our model tflite interpreter
```

```
interpreter = Interpreter(model_path)
interpreter.allocate_tensors()

# Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()



# Setting up button
def button_callback(channel):
   global button_pressed
   button_pressed += 1
   sentence = 'button pushed and pressed ' + str(button_pressed) + ' times'
   print(sentence)
#  speak_sentence(sentence)

gpio.setmode(gpio.BOARD) # Use physical pin numbering
gpio.setup(10, gpio.IN, pull_up_down=gpio.PUD_DOWN) # set pin 10 to be an
    input pin and set initial value to be pulled low (off)
gpio.add_event_detect(10, gpio.RISING, callback=button_callback) # Setup
    event on pin 10 rising edge



speak_sentence(f'Press the button {start_limit} time to start and Press
    the button {end_limit} times to stop')

while True:

   if button_pressed >= start_limit:        # Start the process only if
       button is pressed required number of times
      speak_sentence('Counting process has started')
```

```
with picamera.PiCamera() as camera:
    camera.resolution = (height, width)
    camera.start_preview()

    # Start Reading frames until the button is not pressed again
        required number of times
    while button_pressed < end_limit:

        # Input frame
        frame = np.empty((height, width, 3), dtype=np.uint8)
        camera.capture(frame, 'rgb')
        #print(np.min(frame), np.max(frame), frame.shape)
        frame = frame.astype(np.float32) # Because model understands
            float32 only, doing it while capturing from camera results
            in unwanted results of model so that's why doing it here

        # Classify
        class_pred = classify_frame(frame, interpreter, input_details,
            output_details)

        # Display class name
        camera.annotate_text = "{}".format(class_pred)

        # Counting Logic
        if class_pred != prev_pred:
            frames_preds += ' ' + class_pred
        else:
            frames_preds += ' ' + class_pred
        prev_pred = class_pred


    camera.stop_preview()
```

```
speak_sentence('Counting process has finished')

# As button has been pressed according to end criteria and need to
    wrap up so count now
total_amount = count(frames_preds)

# Output total_amount through sound
amount_to_speak = f'Total amount is {total_amount} rupees'
speak_sentence(amount_to_speak)

gpio.cleanup()
break
```

Device is interacting with user using Audio and start and end of process is controlled using the amount of times button is pressed. Notice that we are using camera preview, but its only for debugging and seeing predictions on live camera window.

Chapter 4

Results

4.1 Training Results

4.1.1 Training Accuracy VS Validation Accuracy

[13]One way to verify if a machine learning model is not overfitting is to divide the data into three parts

- Train data
- Validation data
- Test data

[13]While training we continuously train over Train data and check the accuracy over validation data. If accuracy on Train data is much higher than accuracy on Validation data then that means the model has overfitted. If the accuracy on train data is near the accuracy on validation data, then that means the deep learning trained model has not overfitted. We would be using Tensorboard for this purpose. Tensorboard is a visualization tool that helps in montoring a tensorflow model.

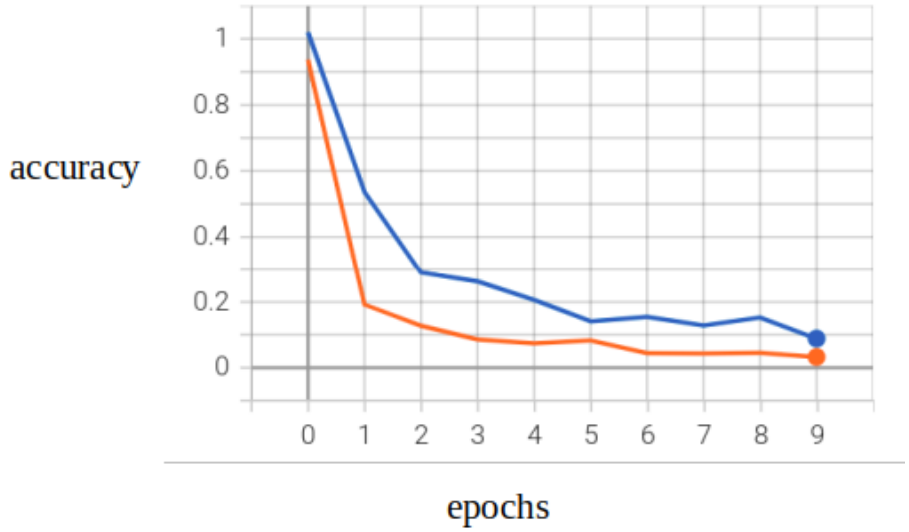

Figure 4.1: Train accuracy vs Validation accuracy in Tensorboard (Red Line is Train Accuracy, Blue Line is Validation Accuracy

4.1.2 Accuracy and Precision as Metrics

[14]Accuracy can be misleading as a metric as it can be biased towards an imbalance class problem. Precision is the fraction of notes predicted correctly from the predicted notes. Precision is more important for us as a metric. We want our model to be as precise as much as possible. If our model is not precise, it will be giving incorrect results, and thus counting results would be disastrous and can cause the user a big financial setback. Below are given results on validation data and test data.

	Accuracy	Precision
Validation	94.05	93.99
Test	90.25	89.75

Table 4.1: Results of Model

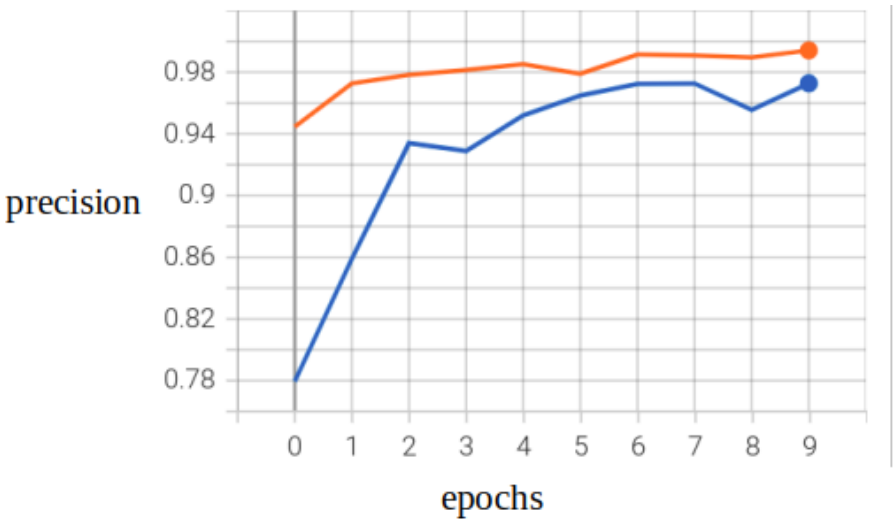

Figure 4.2: Train precision vs Validation precision in Tensorboard

Chapter 5

Discussions

5.1 Pi Camera support

We used Pi Camera v1.3. ALong with it we installed Raspberry Pi OS bullseye and updated it, the camera did not work. There was very less documentation on the problem available on the internet. We tried brute force approach and we came to the solution that Raspberry Pi OS bullseye should not be updated. This solution worked and our Pi Camera started working. This seems to be a problem of OS having no bindings for Pi Camera v1.3. [11]

Figure 5.1: Pi Camera

5.2 No Pakistani Currency Dataset

We collected dataset ourselves as there was no dataset found online. We collected 4,610 images on total of 15 classes. For background class we downloaded a dataset of 1000 images of imagen. The dataset contained one image of each class, this dataset for background class was recommended by the developers of Seeing AI Currency Detection.

5.3 Transfer Learning and Data Augmentation

[1]As we had less data available, so to counter this problem we used both Transfer Learning and Data Augmentation on our dataset. By combining these two strategies we were able to attain a suitable performance for Pakistani Currency Detection.

5.4 Deploying Light Weight Model to Raspberry Pi

[12]We did not deploy the default model to Raspberry Pi. As to run such a model one needs full tensorflow with alot of dependencies. Which if not installed carefully can cause alot of problems, and also result in heat up of Raspberry Pi. We converted the model into a light weight TF Lite model and deployed such a modle to Raspberry Pi. Good thing is to run such a model we needed only the inference module on Raspberry Pi, which makes this collectively a light weight solution. This light weight solution can also be extended to apps.

5.5 Not depending on OpenCV on Pi

OpenCV is a huge dependency, installing it on Raspberry Pi can make most of the work easy but if not installed carefully it can heat up Pi so one should do it while using a cooling fan. We did not depend on the dependency and just used a built python library called picamera. We were able to achieve our goals in a much efficient way without

adding any dependency.

5.6 No Good Papers on Currency Counting

We searched online but couldn't come up with a reputable paper presented in a top-notch conference discussing Algorithms for Currency Counting based on Deep Learning based models for Currency Detection. The approach we came up with takes in certain assumptions that model's detections should be as precice as possible. Our approach can filter most of the outliers (bad predicted classes) if model only predics outliers for few frames. We can also not count each frame, so for that we use a strategy called majority voting to majority vote the predictions of a window of frames for a single note. Our approach also has assumption that only one note should be in front of camera. And to count another note there should come background class.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We started with the problem that in Pakistan there are very less accessible products for Pakistani Currency Detection even from big companies like Google, Microsoft. We came up with solution that we should follow the strategy of Microsoft's Seeing AI Currency Detection Developers and build a machine learning based model based on MobileNetV3. We collected 4,610 dataset of Pakistani currency ourseleves and trained it according to Seeing AI currency detection criteria. Our model gave good accuracy and precision results even on test data. We then deployed the light weight version tflite model to Raspberry Pi along with integration of push button and audio for live interaction with user. We

Figure 6.1: Iot glasses developed

also tested the device on a visually impaired volunteer and he gave a good feedback along with some suggestions for future work. At the end as extra we also developed a basic accessible app based on android for those visually impaired users who are aware of accessible technologies. We have also open sourced the Pakistani currency dataset on Kaggle so society can benefit more from our valuable contribution.

6.2 Future Work

More dataset can be collected to improve the performance of our model, because in deep learning as dataset grows, efficiency of model's performance grows too. Our assumption for classification and counting of notes is that one note should be present before a camera. This can be extended to more advanced scenarios where multiple notes can be present before camera and model can do multi object detection, in this case detection of multi notes and a more suitable counting algorithm in such a scenario. An app for ios can also be made as future work as most visually impaired users who are trained with accessible technologies prefer ios devices.

Listings

Bibliography

[1] Wee Hyong Xiaoyong, Anirudh. Seeing ai currency detection methodolgy. https://docs.microsoft.com/en-us/archive/blogs/machinelearning/how-to-develop-a-currency-detection-model-using-azure-machine-learning, May 2018.

[2] Bilal Hassan, Ramsha Ahmed, Bo Li, Ayesha Noor, and Zahid ul Hassan. A comprehensive study capturing vision loss burden in pakistan (1990-2025): Findings from the global burden of disease (gbd) 2017 study. *PloS one*, 14(5):e0216492, 2019.

[3] Larisa Dunai Dunai, Mónica Chillarón Pérez, Guillermo Peris-Fajarnés, and Ismael Lengua Lengua. Euro banknote recognition system for blind people. *Sensors*, 17(1):184, 2017.

[4] Qian Zhang and Wei Qi Yan. Currency detection and recognition based on deep learning. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2018.

[5] Hanish Aggarwal and Padam Kumar. Indian currency note denomination recognition in color images. *International Journal on Advanced Computer Engineering and Communication Technology*, 1(1):12–18, 2012.

[6] Ayesha Younis, Li Shixin, Shelembi Jn, and Zhang Hai. Real-time object detection using pre-trained deep learning models mobilenet-ssd. In *Proceedings of 2020 the 6th International Conference on Computing and Data Engineering*, pages 44–48, 2020.

[7] Nithesh Singh Sanjay and Ali Ahmadinia. Mobilenet-tiny: A deep neural network-based real-time object detection for rasberry pi. In *2019 18th IEEE International*

Conference On Machine Learning And Applications (ICMLA), pages 647–652. IEEE, 2019.

[8] Andrew Ng. Deep learning specialization. https://www.coursera.org/specializations/deep-learning.

[9] Wikipedia. Data augmentation. https://en.wikipedia.org/wiki/Data$_{a}ugmentation$.

[10] Tensorflow lite. https://www.tensorflow.org/lite.

[11] Raspberry pi bullseye os. https://www.raspberrypi.com/news/raspberry-pi-os-debian-bullseye/.

[12] Tensorflow lite runtime. https://pypi.org/project/tflite-runtime/.

[13] Overfitting. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/.

[14] Precision. https://en.wikipedia.org/wiki/Precision$_{a}nd_{r}ecall$.