Faculty of Computer & Information Technology
Artificial Intelligence Department

An advanced system that assists visually impaired individuals

By:

No.	Student Name	Student ID
1	Taha Mohamed Khalaf	21-01656
2	Mahmoud Adel Sadek	21-00940
3	Sama Moustafa Ibrahim	21-01553
4	Nada Ahmed Hamoda	21-01337
5	Nooran Abdulbasit Ali	21-01361
6	Renad Mohamed Fawzy	21-01502

Under Supervision of:

Associate Professor. Amany Magdy Mohamed	***Eng. Dina Mohamed***
- Professor In Computer and Information Technology ***Egyptian E-Learning University.***	***- Teaching Assistant in Computer and Information Technology*** ***Egyptian E-Learning University.***

This thesis is submitted as partial fulfillment of the requirements for the degree of Bachelor in Artificial Intelligence.
EELU – Asyut 2025

ACKNOWLEDGEMENTA

All praise is due to Allah, whose grace and blessings make all things possible. We thank Allah for His endless blessings, which have been the reason for our success in completing this stage of our academic journey.

We have put in significant effort into this project, but it would not have been possible without the kind support and assistance of many individuals and organizations. We would like to extend our sincere thanks to all of them.

Thanks to the **Egyptian E-Learning University**, especially the **Assiut Center**, for their ongoing support and assistance in helping us reach this level of awareness.

We would also like to express our deep and sincere gratitude to our supervisor, **Dr. Amani Magdy**, for giving us the opportunity to work under her guidance and providing invaluable support throughout the project. She was always keen to ensure the work was of the highest quality, solving any issues that arose, and offering continuous support. It was a great privilege to work with her, and we are extremely grateful for everything she has offered us.

Special thanks go to **Eng. Dina Mohamed** for her continuous guidance and effective support, her patience, friendship, empathy, and wonderful sense of humor. Her dynamism and encouragement have deeply motivated and inspired us.
Finally, we would like to express our gratitude to everyone who helped us throughout this project.

Abstract

Our smart glasses offer a groundbreaking solution to enhance the independence and daily lives of visually impaired individuals. By leveraging advanced AI technologies, our system provides real-time assistance, allowing users to navigate their surroundings with confidence and perform everyday tasks more efficiently.

One of the key advantages of our smart glasses is their ability to integrate multiple assistive features into a single device. Unlike existing solutions, our system combines object detection, face recognition, emotion recognition, text reading, and currency detection to provide a comprehensive tool tailored to users' needs.

These smart glasses not only improve mobility by detecting obstacles but also foster social inclusion through facial recognition and emotion analysis, enabling users to engage more effectively with others.

Additionally, our solution is designed with affordability and accessibility in mind, utilizing lightweight, energy-efficient hardware and open-source AI models to ensure cost-effectiveness.

With these features, our smart glasses stand out as a versatile, efficient, and innovative tool that empowers visually impaired individuals to lead more independent and fulfilling lives.

Table of contents

Chapter 3: System Overview

Chapter 4: Implementation and Testing

Chapter 6: User Manual

Chapter 7: Conclusion and Future Work

List of Figures

Table of Abbreviations

1D	One-Dimensional
AdaBoost	Adaptive Boosting
AI	Artificial Intelligence
API	Application Programming Interface
ARM	Advanced RISC Machine
BLIP	Bootstrapped Language-Image Pretraining
CDI	Currency Detection Identification
CNNs	Convolutional Neural Networks
Colab	Google Colaboratory
ConvNet	Convolutional Network
CPU	Central Processing Unit
CRNN	Convolutional Recurrent Neural Network
CSPDarknet	Cross-Stage Partial Darknet
CTC	Connectionist Temporal Classification
CV	Computer Vision
DB	Differentiable Binarization
DPC	Defect Pixel Correction
DRL	Deep Reinforcement Learning
Eq	Equation
Faster R-CNN	Faster Region-based Convolutional Neural Network
FC	Fully Connected
FDRE	Face Detection, Recognition, and Expression
FPGM	Filter Pruning via Geometric Median
GANs	Generative Adversarial Networks
GND	Ground
GPIO	General-Purpose Input/Output
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
HDR	High Dynamic Range
HOG	Histogram of Oriented Gradients
HTTP	Hypertext Transfer Protocol
H/W	Hardware
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IoU	Intersection over Union
ITC	Image-Text Contrastive Learning
ITM	Image-Text Matching
JSON	JavaScript Object Notation
LM	Language Modeling
Micro-SD	Micro Secure Digital
MIPI CSI	Mobile Industry Processor Interface Camera Serial Interface

MIPI DSI	Mobile Industry Processor Interface Display Serial Interface
MLPs	Multilayer Perceptrons
MTCNN	Multi-task Cascaded Convolutional Neural Network
NLP	Natural Language Processing
NoIR	No Infrared (Filter)
OA	Office Automation
OCR	Optical Character Recognition
ONNX	Open Neural Network Exchange
Pact	Parametric Activation
PANet	Path Aggregation Network
PDAF	Phase Detection Autofocus
PP-OCR	PaddlePaddle Optical Character Recognition
PS2	PlayStation 2
PWM	Pulse-Width Modulation
ReLU	Rectified Linear Unit
RNNs	Recurrent Neural Networks
SD	Scene Description
SE	Squeeze-and-Excitation
SGD	Stochastic Gradient Descent
SNR	Signal-to-Noise Ratio
SoCs	System on a Chip
TPUs	Tensor Processing Units
TR	Text Reading
USB	Universal Serial Bus
URL	Uniform Resource Locator
ViT	Vision Transformer
YOLO	You Only Look Once
YOLOv8	You Only Look Once version 8

Chapter 1
Introduction

1.1 Motivation

AI (deep learning approaches) have recently started to play an important role in independence and social services for people with special needs, such as the visually impaired.

There is a broad scope of tasks to address, which visually impaired individuals face daily challenges in navigation, social interaction, and accessing printed information, often relying heavily on caregivers. For many of these tasks, it is of high interest to achieve real-time performance to aid visually impaired individuals.

Existing assistive technologies (e.g., ultrasonic sensors, basic smart glasses) have many limitations, which brings us to create our advanced deep learning project to assist the visually impaired.

1.2 Problem Definition

Visually impaired people face daily challenges in understanding their environment due to limited access to real-time visual information. Existing tools are either too basic or not practical for everyday use. This project aims to solve this by developing smart glasses that use AI (deep learning) to improve independence and safety for blind users.

1.3 Tasks Done Using Deep Learning

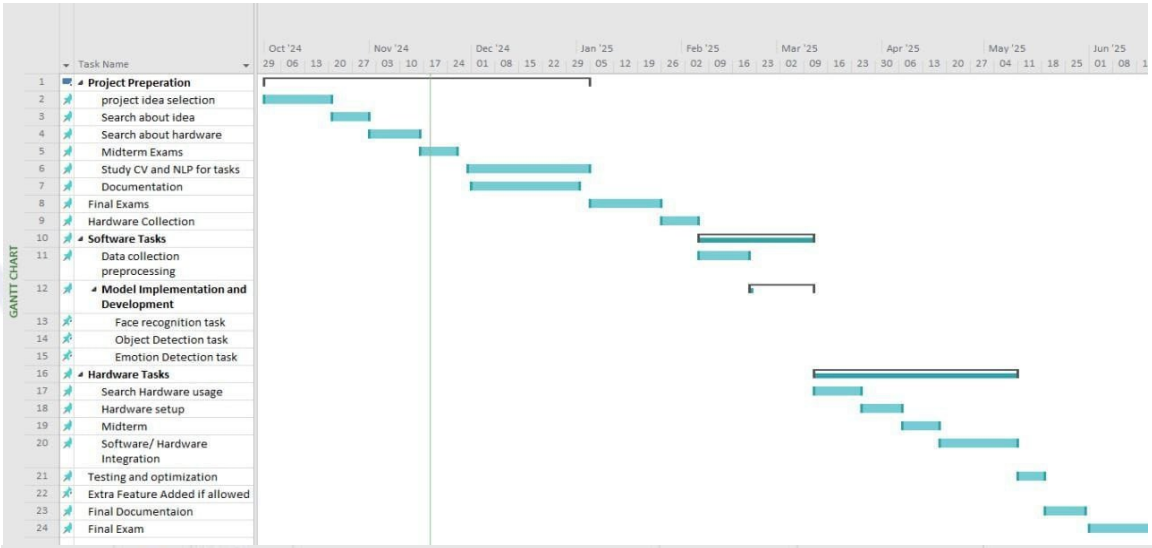
1. scene description
2. object detection
3. currency identification
4. face recognition
5. facial expression
6. reading text

1.4 Objective

Design Smart Glasses Wearable Device That:

- Increased Independence: Reduces reliance on caregivers for basic tasks like navigation, reading, and financial transactions.
- Improved Social Inclusion: Features like face recognition and expression detection enable users to engage more effectively in social settings.
- Enhanced Mobility: Object detection ensures safe and efficient navigation in crowded or unfamiliar environments.
- Accessibility and Affordability: The system is designed to be cost-effective, using open-source tools and efficient hardware for maximum accessibility.

1.5 Time Plan



GANTT CHART

	Task Mode	Task Name	Duration	Start	Finish
1		**Project Preperation**	**69 days**	**Tue 01/10/24**	**Sat 04/01/25**
2		project idea selection	15 days	Tue 01/10/24	Sun 20/10/24
3		Search about idea	9 days	Mon 21/10/24	Thu 31/10/24
4		Search about hardware	11 days	Fri 01/11/24	Fri 15/11/24
5		Midterm Exams	8 days	Sat 16/11/24	Tue 26/11/24
6		Study CV and NLP for tasks	27 days	Sat 30/11/24	Sat 04/01/25
7		Documentation	24 days	Sun 01/12/24	Wed 01/01/25
8		Final Exams	17 days	Sun 05/01/25	Sat 25/01/25
9		Hardware Collection	9 days	Sun 26/01/25	Wed 05/02/25
10		**Software Tasks**	**24 days**	**Thu 06/02/25**	**Tue 11/03/25**
11		Data collection preprocessing	11 days	Thu 06/02/25	Thu 20/02/25
12		**Model Implementation and Development**	**13 days**	**Fri 21/02/25**	**Tue 11/03/25**
13		Face recognition task			
14		Object Detection task			
15		Emotion Detection task			
16		**Hardware Tasks**	**44 days**	**Wed 12/03/25**	**Sat 10/05/25**
17		Search Hardware usage	10 days	Wed 12/03/25	Tue 25/03/25
18		Hardware setup	9 days	Wed 26/03/25	Sun 06/04/25
19		Midterm	9 days	Mon 07/04/25	Thu 17/04/25
20		Software/ Hardware Integration	17 days	Fri 18/04/25	Sat 10/05/25
21		Testing and optimization	7 days	Sun 11/05/25	Sun 18/05/25
22		Extra Feature Added if allowed			
23		Final Documentaion	10 days	Mon 19/05/25	Fri 30/05/25
24		Final Exam	17 days	Sun 01/06/25	Mon 23/06/25

Figure 1-1 Time plan

1.6 Document Organization

Chapter 2: Background This chapter contains a detailed description of the field of the project, all the scientific background related to the project; a survey of the work done in the field and description of existing similar systems.

Chapter 3: System Overview This chapter describes the system design including hardware and the system architecture and how it communicates with internal and external modules.

Chapter 4: Implementation This chapter includes a detailed description of all the functions in the system, A detailed description of all the techniques and algorithms implemented, and Description of any new technologies used in implementation.

Chapter 5: hardware this chapter include a describes of H/W components, setups and integration of algorithms in Raspberry Pi.

Chapter 6: User Manual This chapter describes in detail how to operate the project along with screen shots of the project representing all steps.

Chapter 7: Conclusions and Future Work This chapter contains a complete summary of the project and how we would be able to improve the performance and what are new features that can be added in the future.

Chapter 2
Background

2.1 Neural Networks

Neural Networks are the functional unit of Deep Learning and are known to mimic the behavior of the human brain to solve complex data-driven problems. The input data is processed through different layers of artificial neurons stacked together to produce the desired output. From speech recognition and person recognition to healthcare and marketing, Neural Networks have been used in a varied set of domains.

2.2 Neural Network Elements

Neural Network is comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. As shown in Figure 2-1, each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

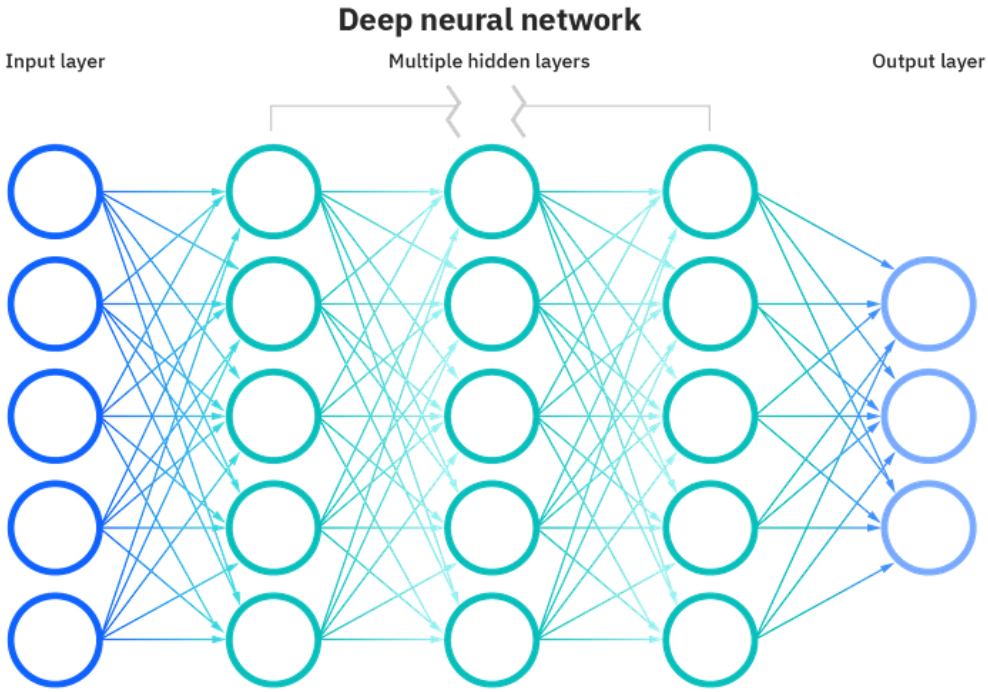


Figure 2-1 Deep Neural Network

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.

2.3 How Do Neural Networks Work

Think of each individual node as its own linear regression model, composed of input data, weights, a bias (or threshold), and an output. The formula would look something like Eq 2-1.

$$\sum_{i=1}^{n} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$

Equation 2.1

$$output = f(x) = \begin{cases} 1, & if \sum w_1 x_1 + b \geq 0 \\ 0, & if \sum w_1 x_1 + b < 0 \end{cases}$$

Equation 2.2

Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs. All inputs are then multiplied by their respective weights and then summed. Afterward, the output is passed through an activation function, which determines the output. If that output exceeds a given threshold, its "fires" (or activates) the node, passing data to the next layer in the network. This results in the output of one node becoming the input of the next node. This process of passing data from one layer to the next layer defines this neural network as a feedforward network.

$$Cost\ Function = MSE = \frac{1}{2m} \sum_{i=1}^{m} (\bar{Y} - Y)^2$$

Equation 2.3

Ultimately, the goal is to minimize our cost function Eq 2-2 to ensure the correctness of fit for any given observation. As the model adjusts its weights and bias, it uses the cost function and reinforcement learning to reach the point of convergence,

Or the local minimum. The process by which the algorithm adjusts its weights is through gradient descent, allowing the model to determine the direction to take to reduce errors (or minimize the cost function). With each training example, the parameters of the model adjust to gradually converge at the minimum, as shown in Figure 2-2.

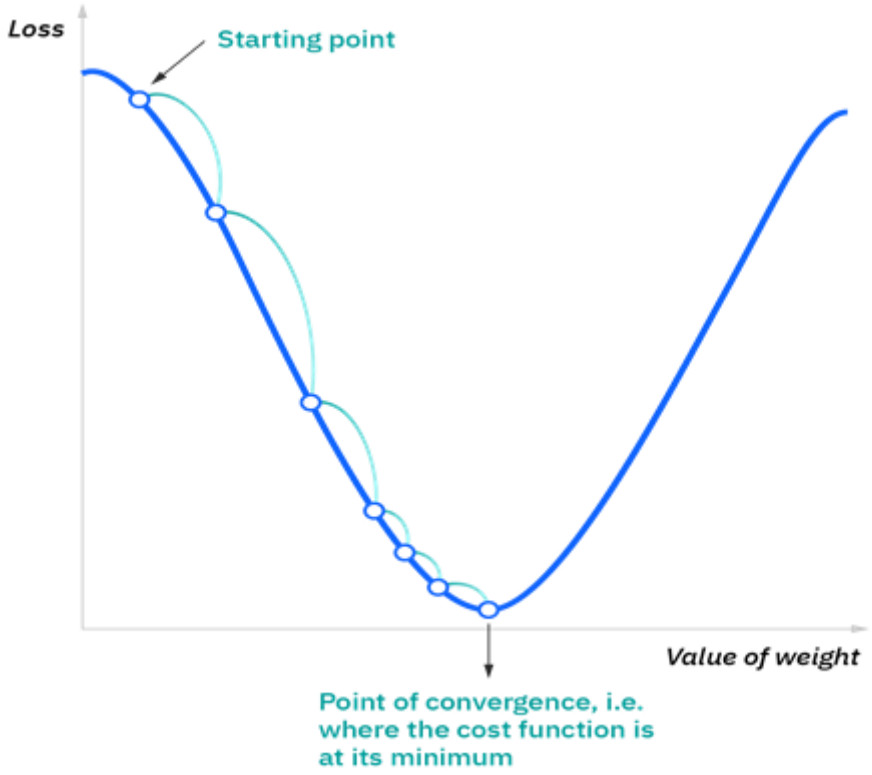


Figure 2-2 Gradient descent

Most deep neural networks are feedforward, meaning they flow in one direction only, from input to output. However, you can also train your model through backpropagation; that is, move in the opposite direction from output to input. Backpropagation allows us to calculate and attribute the error associated with each neuron, allowing us to adjust and fit the parameters of the model(s) appropriately.

2.4 Types of Neural Networks

Neural networks can be classified into different types, which are used for different purposes. While this isn't a comprehensive list of types, the below would be representative of the most common types of neural networks that you'll come across for their common use cases: The perceptron is the oldest neural network, created by Frank Rosenblatt in 1958. It has a single neuron and is the simplest form of a neural network, as shown in Figure 2-3

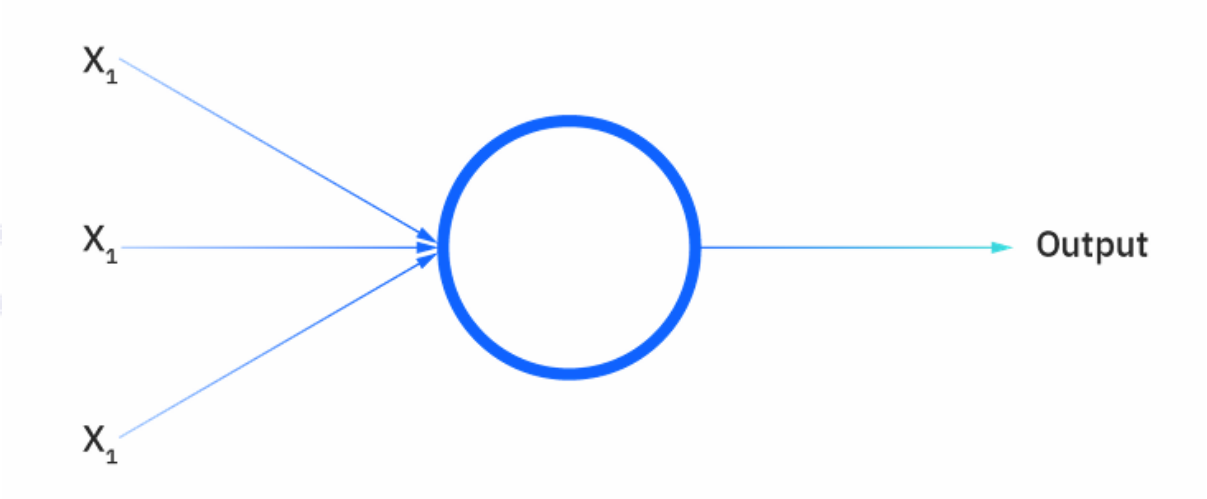


Figure 2-3 Neuron

Feedforward neural networks, or multi-layer perceptron (MLPs) are comprised of an input layer, a hidden layer or layers, and an output layer.

Convolutional neural networks (CNNs) are like feedforward networks, but they're usually utilized for image recognition, pattern recognition, and/or computer vision. These networks harness principles from linear algebra, particularly matrix multiplication, to identify patterns within an image.

2.5 Problems Commonly Solved with Neural Networks

Many different problems can be solved with a neural network. However, neural networks are commonly used to address types of problems. The following types of problems are frequently solved with neural networks:

- Regression.
- Classification.
- Pattern recognition.
- Prediction.
- Optimization.
- Clustering.

2.6 Main Components of Neural Network

Activation Function

Used to achieve non-linearity [14], squashing the range of the input, and used at the last layer to output the probability of each class.

Some common functions:

- Sigmoid.
- ReLU.
- Hyperbolic Tangent.
- Linear (Identity).
- Leaky ReLU.
- SoftMax.

Architecture or Structure

The connectivity of neurons (nodes) determines the neural network structure (architecture).

The Learning Algorithm, or Training Method

Method for determining weights of the connections. How the neurons of neural networks are structured is intimately linked with the learning algorithm used to train the network.

Hyperparameters

Hyperparameters are the variables that determine the network structure (E.g., Number of Hidden Units) and the variables that determine how the network is trained (E.g., Learning Rate). Hyperparameters are set before training (before optimizing the weights and bias). It is not appropriate to learn that on the training set.

2.7 Convolution Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower than compared of other classification algorithms. While in primitive methods, filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

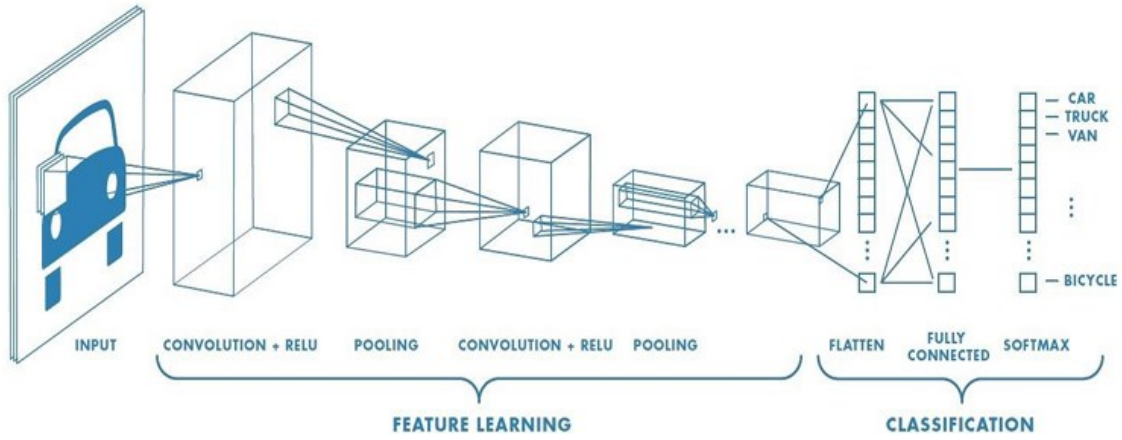


Figure 2-4 Convolution Neural Network

As shown in Figure 2-4, CNN can successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fit to the image dataset due to the reduction in the number of parameters involved and the reusability of weights.

Image Kernel

A small matrix is used to apply techniques such as blurring, sharpening, outlining, edge detecting, or embossing. Used in machine learning for 'feature extraction', a technique for determining the most important portions of an image. In this context, the process is referred to more generally as "convolution", as shown in Figure 2-5

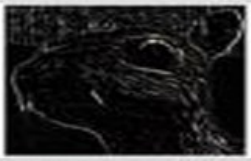
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 2-5 Kernels

2.8 Building Blocks of CNN

Convolutional Layers

Convolution is a specialized type of linear operation used for feature extraction, where a small array of numbers, called a kernel, is applied across the input, which is an array of numbers, called a tensor. An element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a feature map. This procedure is repeated, applying multiple kernels to form an arbitrary number of feature maps, which represent different characteristics of the input tensors; As shown in Figure 2-6, different kernels can, thus, be considered as different feature extractors. Two key hyperparameters that define the convolution operation are size and number of kernels. The former is typically 3×3, but sometimes 5×5 or 7×7. The latter is arbitrary and determines the depth of output feature maps.

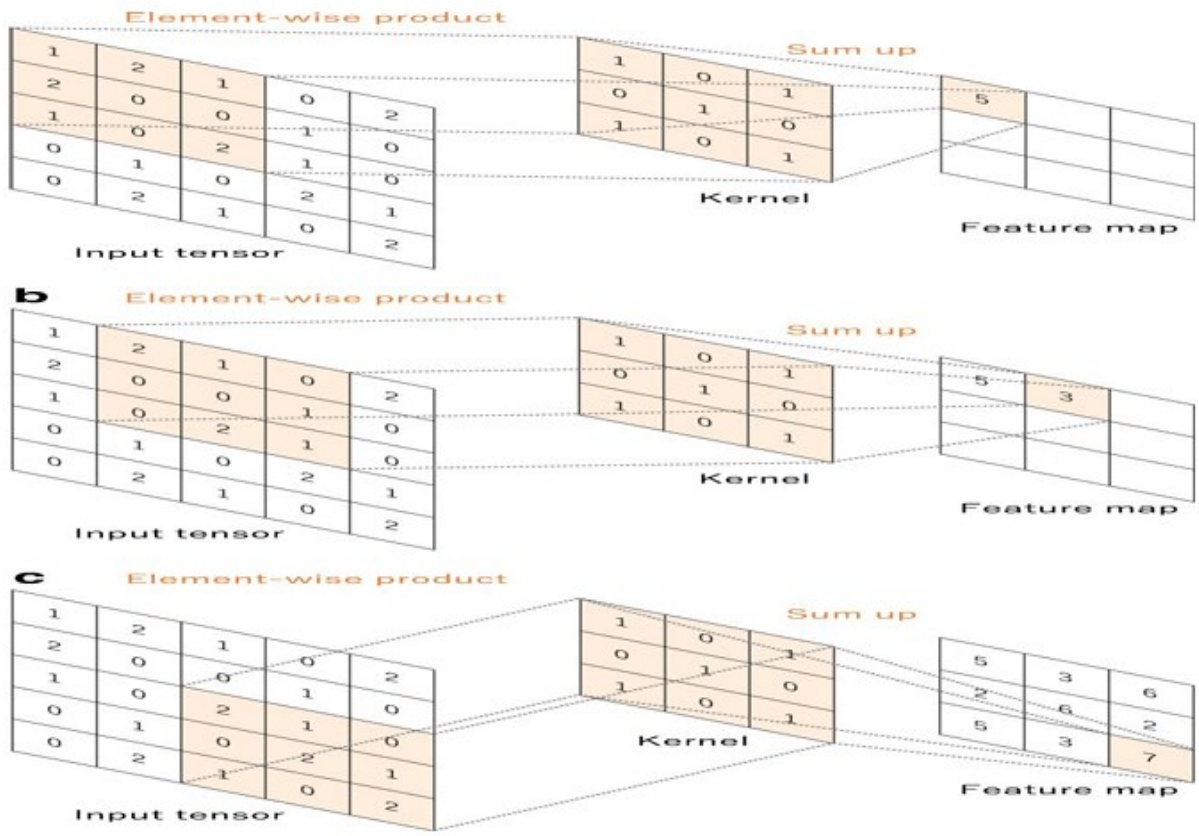


Figure 2-6 Feature Map

Convolution is a linear operation, so an Activation function is used to achieve non-linearity; it's most likely to use RELU in CNN.

Pooling Layers in CNN

A pooling layer provides a typical down sampling operation which reduces the in-plane dimensionality of the feature maps to introduce a translation invariance to small shifts and distortions and decrease the number of subsequent learnable parameters, as shown in figure 2-7, It is of note that there is no learnable parameter in any of the pooling layers, whereas filter size, stride, and padding are hyperparameters in pooling operations, like convolution operations.

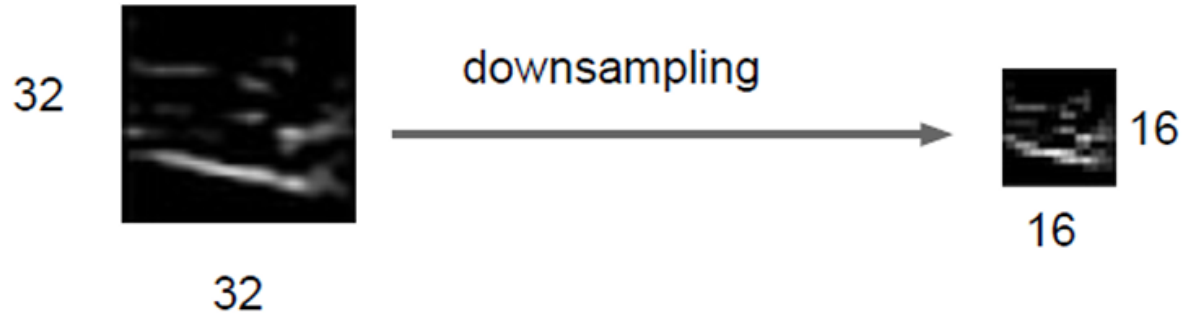


Figure 2-7 Down sampling

Some Pooling Types

- **Max Pooling** – Select maximum value in the section
- **Min Pooling** – Select minimum value in the section
- **Average Pooling** – Average all values in the section
- **Sum Pooling** – Summing all values in the section

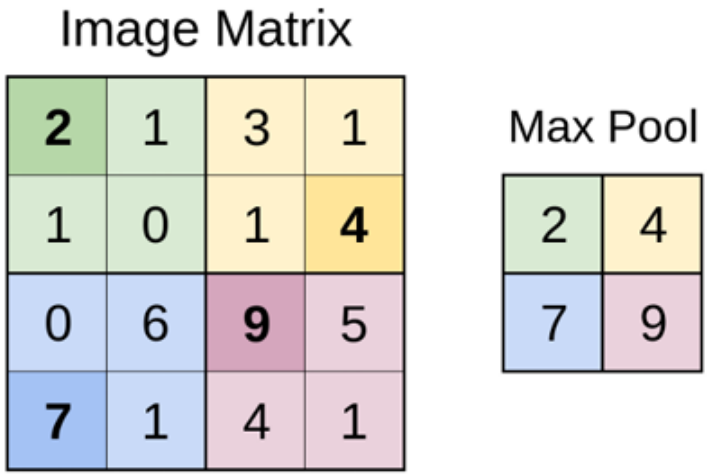


Figure 2-8 Max Pooling

Upon inspection, the darker, bold numbers in each colored section appear in Figure 2-8, as they are the largest values in the respective colored sections. The image above is max pooling with a 2×2 filter, like the convolutional layer, but no mask is being applied. The same can be applied to a 3×3, 5×5 filter, etc.

Fully Connected Layer

The output feature maps of the final convolution or pooling layer is typically flattened, i.e., transformed into a one-dimensional (1D) array of numbers (or vector), and connected to one or more fully connected layers, also known as dense layers, in which every input is connected to every output by a learnable weight. Once the features extracted by the convolution layers and down sampled by the pooling layers are created, they are mapped by a subset of fully connected layers to the final outputs of the network, such as the probabilities for each class in classification tasks. The final fully connected layer typically has the same number of output nodes as the number of classes. Each fully connected layer is followed by a nonlinear function, such as ReLU, as shown in Figure 2-9.

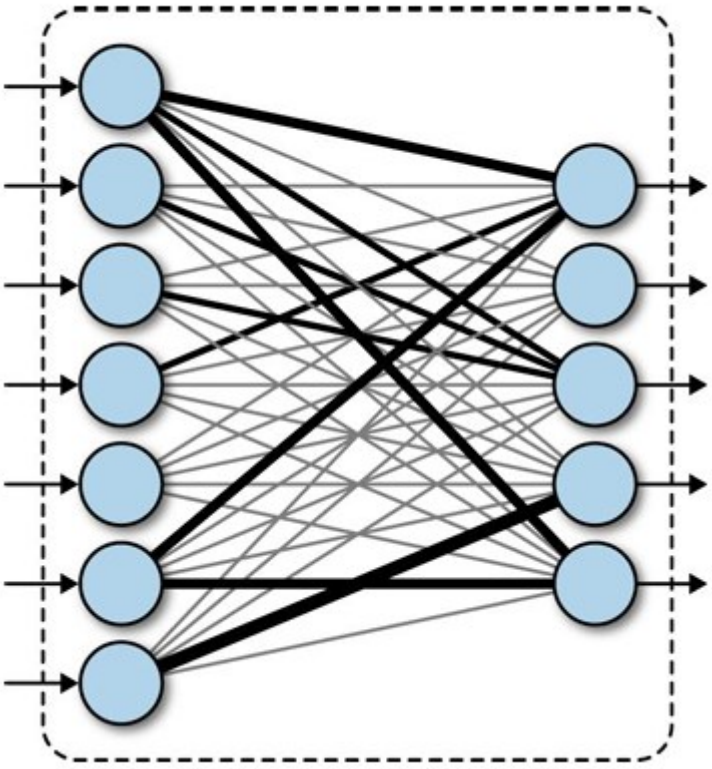


Figure 2-9 Fully Connected Neural Network

Batch Normalization

Batch Norm is a widely used technique in the field of Deep Learning. It improves the learning speed of Neural Networks and provides regularization, avoiding overfitting.

Batch Norm is a normalization technique done between the layers of a Neural Network instead of in the raw data.

It normalizes each scalar feature independently by making

- It has a mean of zero and a variance of 1
- and then scale and shift the normalized value for each training mini batch

It serves to speed up training and use higher learning rates, making learning easier.

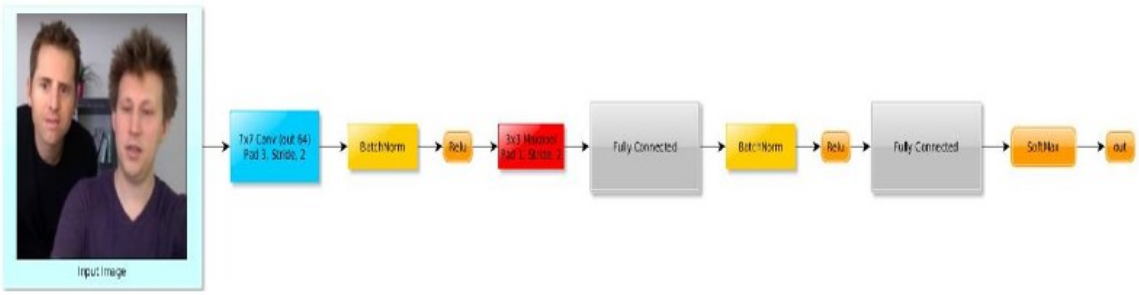


Figure 2-10 Batch Normalization

As shown in Figure 2-10, the batch norm layer is used after linear layers (i.e., FC, conv), and before the non-linear layers (Relu).

Regularization for Good Generalization

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error (any method that prevents over-fitting or helps the optimization).

Overfitting vs Underfitting

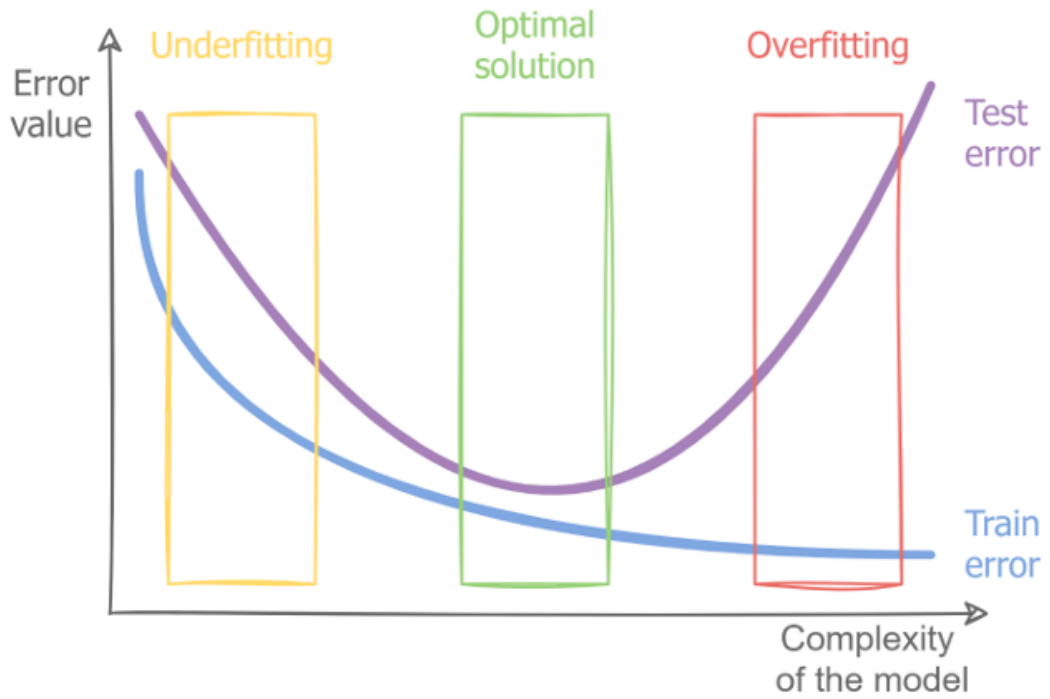


Figure 2-11 Error value vs Train Value

As shown in Figure 2-11:

- **Underfitting:** Inability to obtain a low enough error rate on the training set.
- **Overfitting:** The Gap between training error and testing error is too large.

Regularization Strategies

- **Dropout**

A technique used to improve overfitting on neural networks. Randomly drop units (along with their connections), as shown in Figure 2-12, during training. The probability of dropping is a hyperparameter; 0.5 is common. Neurons are dropped during training, while in testing, all the neurons are used.

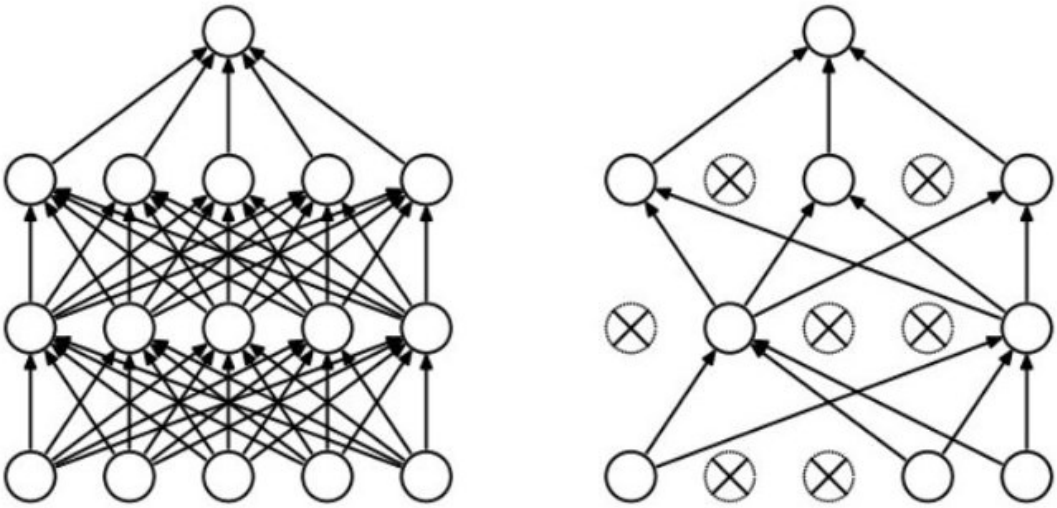


Figure 2-12 Dropout

2.9 Deep Learning

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain, allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy.

Deep learning drives many artificial intelligence applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) as well as emerging technologies (such as self-driving cars).

2.10 Classification of Deep Learning Approaches

Deep Learning techniques are classified into three major categories: unsupervised, semi-supervised, and supervised. Furthermore, deep reinforcement learning (DRL).

Deep Supervised Learning

This technique deals with labeled data. There are several supervised learning techniques, such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and deep neural networks (DNNs).

Deep Semi-Supervised Learning

In this technique, the learning process is based on semi-labeled datasets. Occasionally, generative adversarial networks (GANs) and DRL are employed in the same way as this technique.

Deep Unsupervised Learning

This technique makes it possible to implement the learning process in the absence of available labeled data (i.e., no labels are required). Recurrent neural networks (RNNs) have also been

employed for unsupervised learning in a wide range of applications.

Deep Reinforcement Learning

Reinforcement Learning operates on interacting with the environment.

2.11 Classification Task Architecture

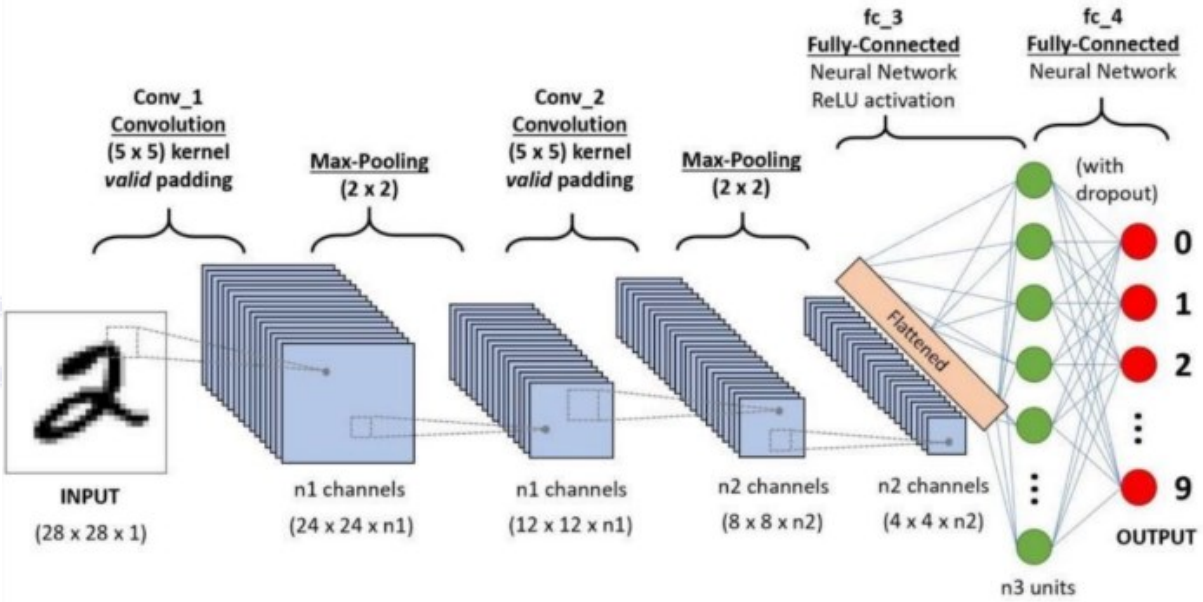


Figure 2-13 Convolution Neural Network – Classifiation

Detection Task

It is a CV task where the goal is to find and locate objects within an image or video.

1. **What** is in the image? → Classification
2. **Where** is it? → Localization (bounding box)

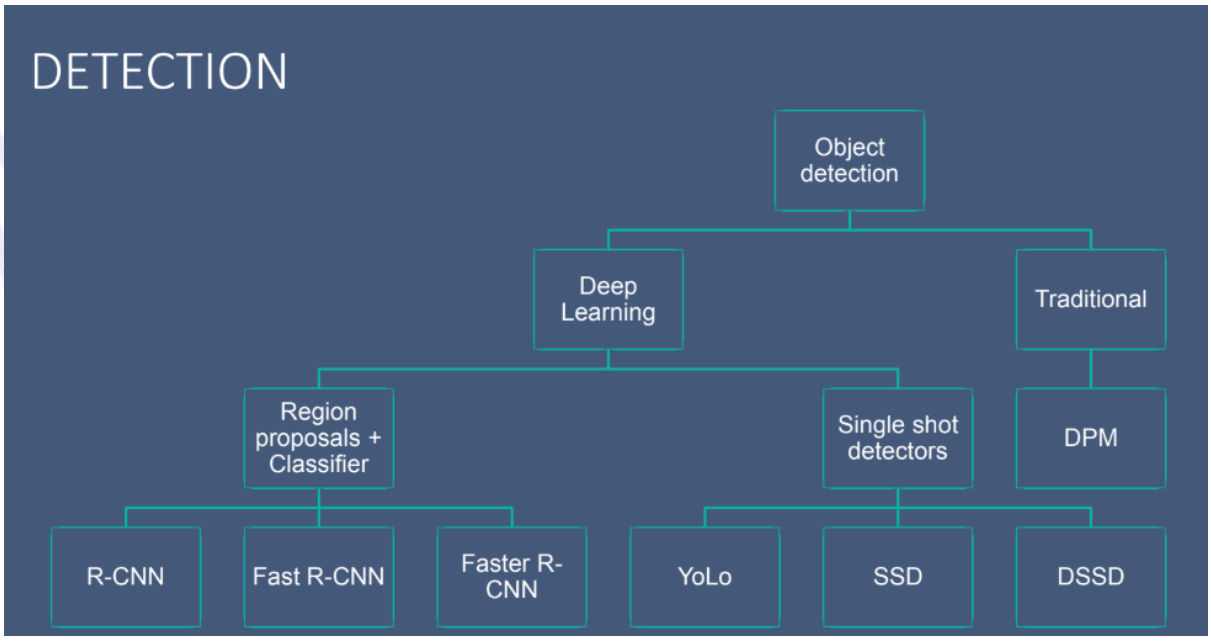


Figure 2-14 Object Detection

YOLO

YOLO (You Only Look Once) is **a real-time object detection** model based on CNN that detects and classifies objects in a single forward pass through a neural network. Unlike traditional object detection methods (e.g., Faster R-CNN), which use region proposals, **YOLO directly predicts bounding boxes and class labels** in a single step, making it extremely fast, as shown in Figure 2-15.

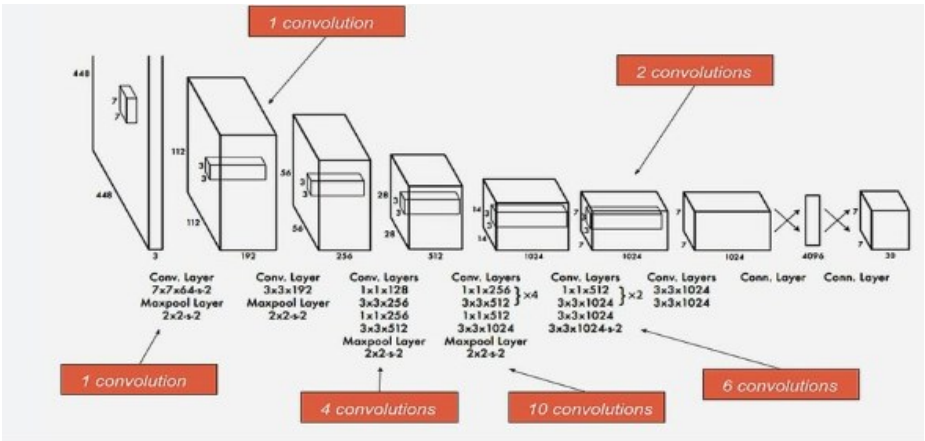


Figure 2-15 You Only Look Once

Visualization results under complex and challenging scenarios:

Figure 2-16 Object Detection Challenges

2.12 History of Assistive Technology for The Visually Impaired

Early Assistive Devices (Pre-19th Century)

- Walking Canes (Ancient Times) – Basic wooden sticks were used to help visually impaired people navigate.
- Tactile Writing Systems (Middle Ages) – Early attempts at raised text and tactile markings appeared in some cultures.

19th Century: Early Innovations

- Braille System (1824) – Invented by Louis Braille, the Braille system provided a tactile way for the blind to read and write using raised dots.
- Raised Letter Books (Mid-19th Century) – Books with embossed letters were created for the visually impaired before Braille became widespread.

20th Century: Technological Advancements

- White Cane (1921) – The modern white cane was introduced by James Biggs in England, later enhanced with mobility training techniques.
- Talking Books (1930s) – The development of audiobooks, particularly for war veterans who lost their sight.
- Guide Dogs (1930s) – Formal training programs for guide dogs began in Germany and later spread worldwide.
- Optacon (1970s) – A device that converted printed text into tactile feedback using a vibrating pin display.

Screen Readers (1980s) – Early versions of screen reading software, like JAWS (Job Access with Speech), were developed to help blind users access computers

21st Century: Smart and AI-Based Technologies

- Refreshable Braille Displays (2000s-Present) – Devices that convert digital text into dynamic Braille patterns.
- Smart Canes (2010s-Present) – Innovations like the Sunu Band and WeWALK Smart Cane incorporate ultrasonic sensors and GPS to aid navigation.

AI-Based Assistive Apps (2010s-Present) – Applications use artificial intelligence and computer vision to describe objects and text to visually impaired users.

2.13 Related Work

Traditional project

The glasses can detect obstacles and transmit this to the blind person using ultrasonic sensors developed by AI robotics & faculty of engineering in Assiut.

Advanced Project focused on one feature

These “Smart Glasses” are designed to help blind people read and translate typed text, which is written in the English language, developed by Prince Mohammad bin Fahd University, as shown in Figure 2-15.

Figure 2-17 Prince Mohammad bin Fahd team glasses

These projects cannot handle daily challenges for visually impaired people

Advances techniques focused on multiple features

Projects we can consider as starting point for proposed project

Amal Glass

is smart glasses designed to assist blind and visually impaired individuals developed by Raqmi Innovation, a Saudi-based technology company focusing on AI and accessibility solutions, as shown in figure 2-16.

Figure 2-18 Amal Glasses

Key Features

- Object Detection
- OCR (Optical Character Recognition)
- Navigation Assistance

Limitations

Although Amal Glass offers foundational assistive functionalities, it lacks advanced features like scene description and face (recognition - emotion detection) limiting its use in complex social and financial scenarios.

Chapter 3
System Overview

3.1 Introduction of the System:

This system enables users to interact with their environment smartly and efficiently using artificial intelligence technologies. The project integrates four main functional areas: facial and emotion recognition, text reading, event description, and currency recognition. The system relies on integrated hardware components that include the glasses, a Raspberry Pi processor, a camera, and a speaker to transmit audio output to the user. This chapter aims to provide a comprehensive overview of the system's design, its components, and how they work together.

3.2 System Objective:

The objective of this system is to assist visually impaired users by providing them with real-time audio feedback about their surroundings. By capturing input images from the environment, the system aims to recognize and audibly convey critical visual information such as the identification of Egyptian currency, the presence and identity of people, their facial expressions, and any readable text. Additionally, it offers a detailed description of the surrounding scenery to enhance spatial awareness. The system is

designed to be intuitive, efficient, and accessible, requiring no prior training for use. It leverages smart hardware and software integration to deliver fast, accurate responses while remaining user-friendly and resource efficient.

3.3 System main Components:

3.3.1 Software models:

The system has 4 main flow includes all models inside it like:

Currency detection identification (CDI) Flow:

The currency detection functionality enables the smart glasses to identify Egyptian banknotes (5, 10, 20, 50, 100, and 200 EGP) in real-time, assisting users, particularly the visually impaired, in financial transactions. It processes images captured by the camera and delivers clear audio announcements of the detected currency denomination through the speaker. This feature ensures reliable recognition under varying environmental conditions, enhancing user independence and accessibility.

Face detection recognition expression (FDRE) Flow:

The flow of face detection and recognition allows them in actual time, with the ability to identify individuals and analyze their emotional expressions. This flow depends on the photos that the camera takes to locate and classify faces, allowing people to know and feelings such as joy, sadness or amazement. The results are delivered as clear audio outputs through the headphone, which enhances social interaction and helps the visual and blind, understand the dynamics of the environment surrounding them, whether in daily situations or complex social interactions, thus improving their personal and social experience significantly.

Scene description SD flow:

The flow of the scene description in the smart glasses allows the creation of natural linguistic descriptions of the visual scenes surrounding the user in actual time. This flow depends on the photos that the camera takes to generate an accurate and brief description of events or things in the environment, such as "a person walking on the street" or "a parked red car". These descriptions are presented as clear audio outputs through the headphone, which helps weakened and blind users to better understand their surroundings and enhances their

circumstantial awareness and their independence in interacting with the daily environment.

Text reading flow:

This flow in the smart glasses allows to recognize the texts in the images, such as signs or documents, and convert them into clear audio outputs. This flow depends on the photos that the camera takes, as it is processed to extract the texts and communicate it as audible words through the headphone. This system, impaired vision and blindness, helps understand the content written in their environment, which enhances their independence and their ability to interact with visual information in daily life.

3.3.2 Hardware:

Raspberry Pi (for Project Implementation):

In this project, the Raspberry Pi 4 Model B serves as the central processing unit and control hub for all the integrated components. It is chosen for its powerful Broadcom BCM2711 quad-core Cortex-A72 processor running at 1.5 GHz, enabling it to handle real-time tasks such as object detection, facial recognition, emotion analysis, and OCR processing simultaneously. With up to 8GB of RAM, it supports concurrent execution of multiple AI models and camera streaming without significant lag. The board's built-in dual-

band Wi-Fi and Bluetooth simplify the wireless transmission of data or connection to audio peripherals for voice feedback. Its 40-pin GPIO header allows easy interfacing with sensors and control modules like the joystick and physical buttons, enabling user interaction and navigation. The Raspberry Pi's small form factor, efficient power consumption, and strong community support make it an ideal platform for a wearable, portable assistive system.

Raspberry Pi Camera Module 3 (for Image and Video Processing):

The Raspberry Pi Camera Module 3 is the primary image acquisition device used in this project. With a high-resolution 12-megapixel Sony IMX708 sensor and HDR capabilities, it captures detailed images even in challenging lighting environments, which is crucial for accurate scene description, facial detection, and currency recognition. Its support for phase detection autofocus (PDAF) ensures sharp focus on subjects, enhancing the accuracy of face recognition and emotion analysis algorithms. The module is compact and easily mountable on wearable hardware such as smart glasses or a head-mounted frame. It connects to the Raspberry Pi via the CSI-2 port using a ribbon cable, ensuring high-speed image transfer necessary for real-time processing. This camera also

supports multiple video resolutions including 1080p and 720p, balancing quality and frame rate based on the processing needs of the AI models running on the Pi.

Joystick Module (for User Interaction):

The Joystick Module is used in this project as a simple and intuitive user interface, allowing the visually impaired user to interact with the system by selecting modes or confirming commands through directional input and button presses. It consists of two potentiometers that capture analog signals along the X and Y axes, and a push-button switch for selection, which connects to the Raspberry Pi's GPIO pins. In the resting state, the joystick outputs a neutral voltage (~2.5V), and changes in direction are interpreted by the Pi to switch between features like currency detection, text reading, or face recognition. The tactile nature of the joystick makes it accessible without visual guidance. Its use significantly enhances the usability of the system, enabling users with little to no technical experience to navigate between functions easily and efficiently in real time.

3.4 System Architecture

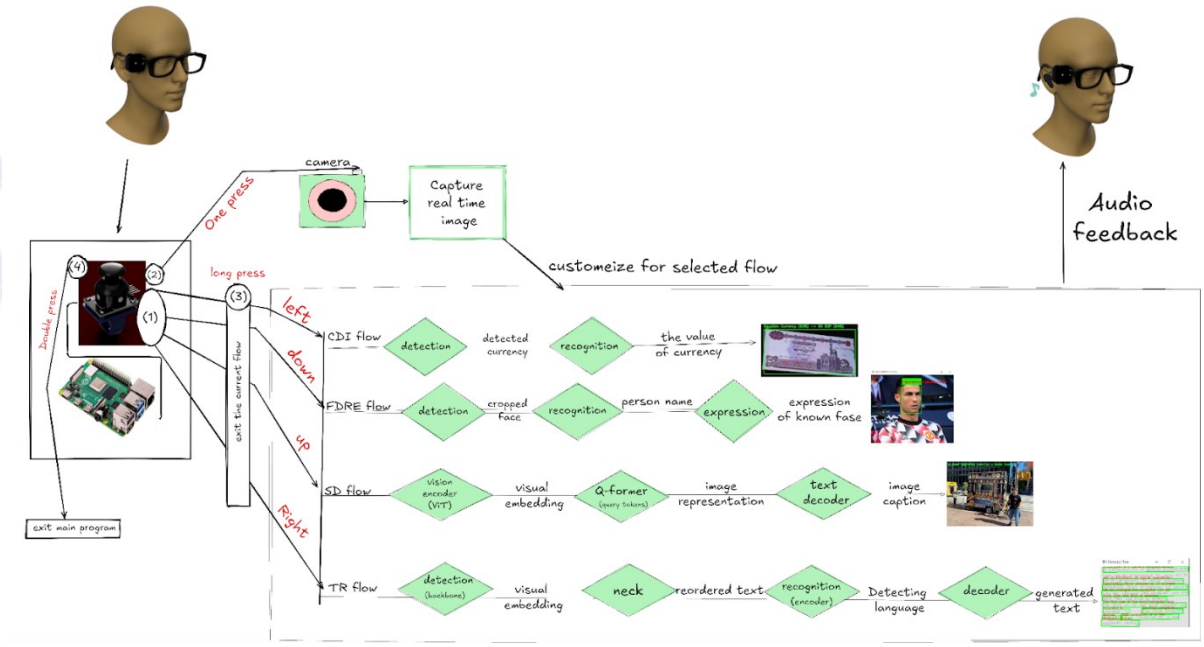


Figure 3-1 System architecture

This system presents a smart glasses system designed to assist visually impaired individuals by processing real-time camera input through a Raspberry Pi and providing audio feedback via a bone-conduction speaker. The system is operated using intuitive joystick controls, allowing the user to easily navigate between different functionalities. A left press on the joystick activates the currency recognition mode to detect and announce the denomination of Egyptian currency, while a right press enables the text reading mode, using OCR to extract and vocalize printed text. Pressing the joystick upward triggers the scene description mode, where the system analyzes and describes the surrounding environment through image captioning. A downward press initiates the face and emotion

recognition mode to identify known individuals and interpret their facial expressions. For system management, a long press exits the current mode and returns to standby, a double press quits the main program, and a single press in any mode captures a real-time image for processing. This control mechanism ensures a hands-free, user-friendly experience tailored to enhance accessibility and independence for visually impaired users in real-world scenarios.

3.5 Challenges

One of the primary challenges in this project was achieving real-time performance on a resource-constrained device like the Raspberry Pi. Processing tasks such as object detection, face recognition, OCR, and scene description require significant computational power, which had to be carefully optimized to maintain responsiveness and minimize latency. Integrating multiple AI models and ensuring they worked seamlessly without overloading the system presented additional complexity. Ensuring accurate recognition of currency, faces, and text under varying lighting conditions and camera angles was also a major challenge, as environmental inconsistencies can significantly affect performance. Furthermore, designing an intuitive and accessible control mechanism for visually impaired users required thoughtful hardware and software integration, particularly in mapping joystick inputs to complex functionalities without causing confusion.

Balancing usability, speed, and accuracy while maintaining low power consumption and compact form factor posed continuous technical and design hurdles throughout the development process

Chapter 4
Implementation and Testing

4.1 Detailed Description of All the Functions in The System

The system mainly consists of groups of features divided into four flows:

1- CDI flow that contains two models

- currency detection
- currency identification

2- FDRE flow that contains three models

- mtcnn detection
- face recognition
- facial expression

3- SD flow that contains BLIP transformer which is contains three parts

- image caption
- Image-Text Matching (ITM)
- Image-Text Contrastive Learning (ITC)

4- TR flow that contains paddleOCR which is contains two parts

- Differentiable Binarization Net for text detection
- API Net for text recognition

4.2 Description of All the Techniques and Algorithms Implemented in Each Flow

4.2.1 CDI Flow:

- **Currency Detection:**

Model

We utilized the **YOLOv8** (You Only Look Once version 8) model for the task of detecting Egyptian currency. YOLOv8 is the latest iteration in the YOLO family of real-time object detectors developed by Ultralytics, known for its exceptional speed, accuracy, and modularity.

The model is designed for maximum flexibility and deploy ability across a range of platforms, including edge devices, making it ideal for our smart glasses system tailored to assist visually impaired users.

The model was fine-tuned using a **custom dataset** composed of annotated images of Egyptian currency denominations: 5, 10, 20, 50, 100, and 200 Egyptian pounds.

These images were collected under varied lighting, angles, and backgrounds to ensure robustness in real-world conditions.

YOLOv8 operates through a single forward pass of a deep convolutional neural network that outputs bounding boxes and corresponding class labels in real-time.

Its streamlined architecture allows for fast inference even on constrained hardware like Raspberry Pi, especially after optimization using formats like ONNX or frameworks like TensorRT. ensuring smooth performance on edge devices such as Raspberry Pi when optimized (e.g., using ONNX or TensorRT).

Architecture Overview

YOLOv8 builds upon the strong foundation established by its predecessors in the YOLO family, integrating cutting-edge advancements in neural network design and training methodologies.

Like earlier versions, YOLOv8 unifies object localization and classification tasks within a single, end-to-end differentiable neural network framework, maintaining the balance between speed and accuracy.

The architecture of YOLOv8 is structured around three core components

1. **Backbone:** YOLOv8 employs a sophisticated convolutional neural network (CNN) backbone designed to extract multi-scale features from input images. This backbone, possibly an advanced version of CSPDarknet or another efficient architecture, captures hierarchical feature maps that represent both low-level textures and high-level semantic information crucial for accurate object detection. The backbone is optimized for both speed and accuracy, incorporating depth wise separable convolutions or other efficient layers to minimize computational overhead while retaining representational power.
2. **Neck:** The neck module in YOLOv8 refines and fuses the multi-scale features extracted by the backbone. It leverages an optimized version of the Path Aggregation Network (PANet), which is enhanced to improve the flow of information across different feature levels. This multi-scale feature integration is critical for detecting objects of varying sizes and scales, and the enhanced PANet design in YOLOv8 likely includes modifications to the original PANet to further optimize memory usage and computational efficiency.

3. **head:** The head module is responsible for generating the final predictions, including bounding box coordinates, object confidence scores, and class labels, from the refined features. YOLOv8 introduces an anchor-free approach to bounding box prediction, moving away from the anchor-based methods.

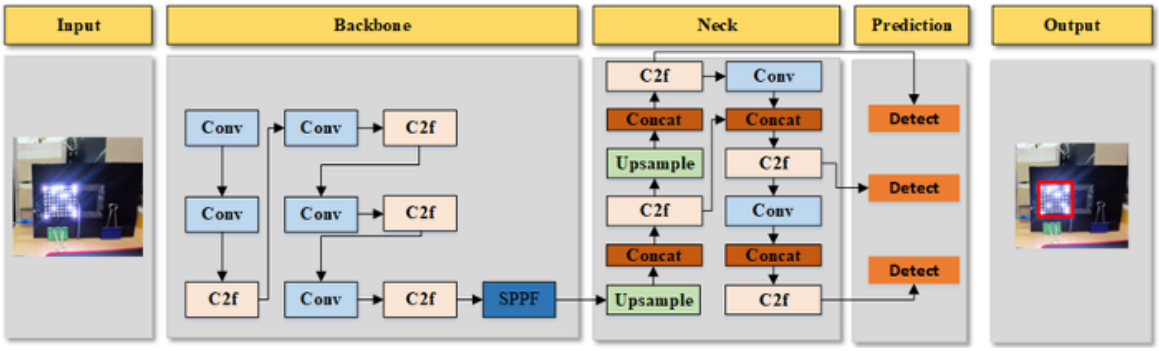


Figure 4-1 Object detection process

YOLOv8 introduces significant architectural and performance improvements over previous YOLO versions:

- **Anchor-Free Detection:** YOLOv8 adopts an anchor-free approach, simplifying training and improving the detection of objects of various sizes and aspect ratios.
- **Unified Model Format:** It provides seamless integration across detection, segmentation, and classification tasks using a single model structure.
- **Modular Design:** The architecture allows easy customization and scaling based on hardware capabilities or accuracy-speed tradeoffs.
- **Decoupled Heads:** The classification and regression heads are separated, enhancing precision in both object localization and classification.

- **Transformer-enhanced Backbone:** YOLOv8 supports improved feature extraction through a robust backbone and can optionally integrate transformer blocks for further performance boosts.

In our application, the backbone extracts spatial features from input images, while the head produces bounding boxes and associated class probabilities for each currency note in the scene.

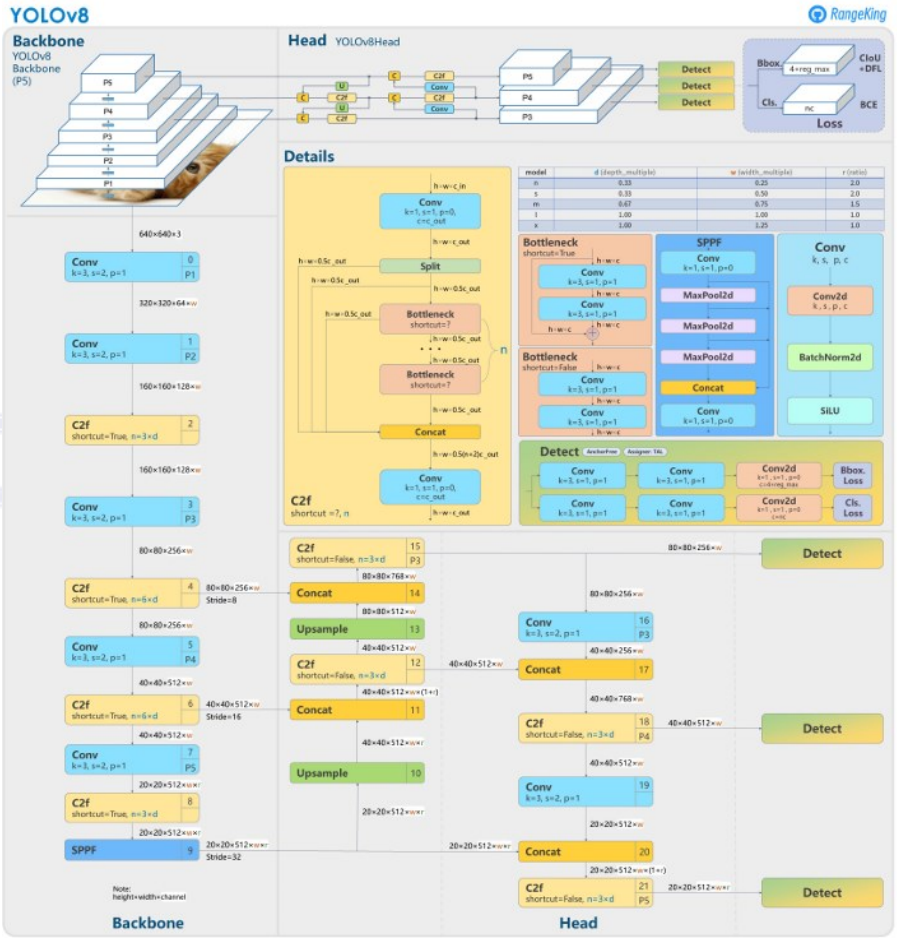


Figure 4-2 Yolov8 structure

Custom Training Details

The YOLOv8 model was trained from scratch using a manually collected dataset of labeled Egyptian currency.

The key training parameters were as follows:

- **Image Resolution:** 640×640 pixels
- **Optimizer:** Stochastic Gradient Descent (SGD) with momentum
- **Loss Functions:** Combination of IoU loss, classification loss, and objectness loss
- **Training Epochs:** 100
- **Batch Size:** 32
- YOLO format (class_id x_center y_center width height).

Training was conducted using the official Ultralytics YOLOv8 training pipeline and codebase, with validation and testing phases using held-out portions of the dataset.

Deployment After training

The model was exported and optimized for real-time inference on resource-constrained devices such as the Raspberry Pi.

Key steps included:

- Conversion to ONNX format for compatibility across platforms and frameworks

- 8-bit Quantization using post-training optimization to reduce size and improve speed
- Integration with PiCamera for live image capture
- Inference Pipeline: Real-time video stream processed to detect and vocalize the detected currency denomination to assist visually impaired users

The deployment achieves high responsiveness while maintaining robust detection in various practical environments.

Dataset

Egyptian Currency Dataset Preparation.

- We began by collecting images of Egyptian banknotes across all denominations, including 5, 10, 20, 50, 100, and 200 EGP.
- After gathering the dataset, we split the images into training and validation sets, **with 250 images for training** and **120 images for validation.**
- Next, **we manually annotated each image** using **bounding boxes** to identify the banknotes.

This annotation process resulted in two types of data for each image: The original image and A corresponding text file containing the bounding box coordinates and class labels.

The final dataset structure is organized under the main directory named **annotated_currency_dataset,** which contains two subfolders:

- images/

  train/: Contains the training images

  val/: Contains the validation images

- labels/

  train/: Contains the annotation text files for the training set

  val/: Contains the annotation text files for the validation set

This structure ensures the dataset is well-prepared and ready for training an object detection model on YOLOv8.

Figure 4-3 Annotated Currency dataset

Image:

Figure 4-4 Currency detection dataset examples

Labels:

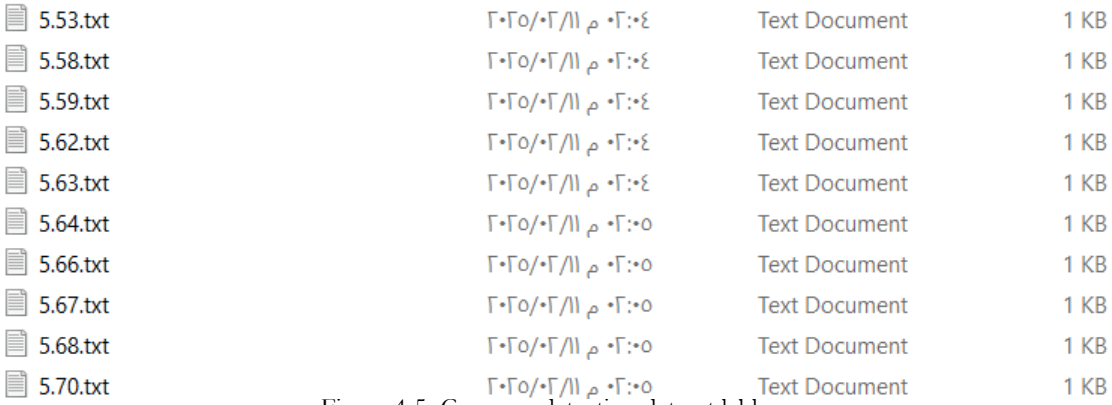


Figure 4-5 Currency detection dataset lables

Evaluation Metrics

The Percentage of accuracy metric is ~98 %, and the visualization of another metrics is:

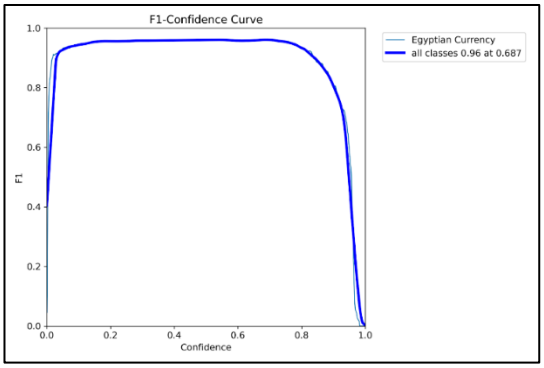


Figure 4-6 Currency detection F1-curve

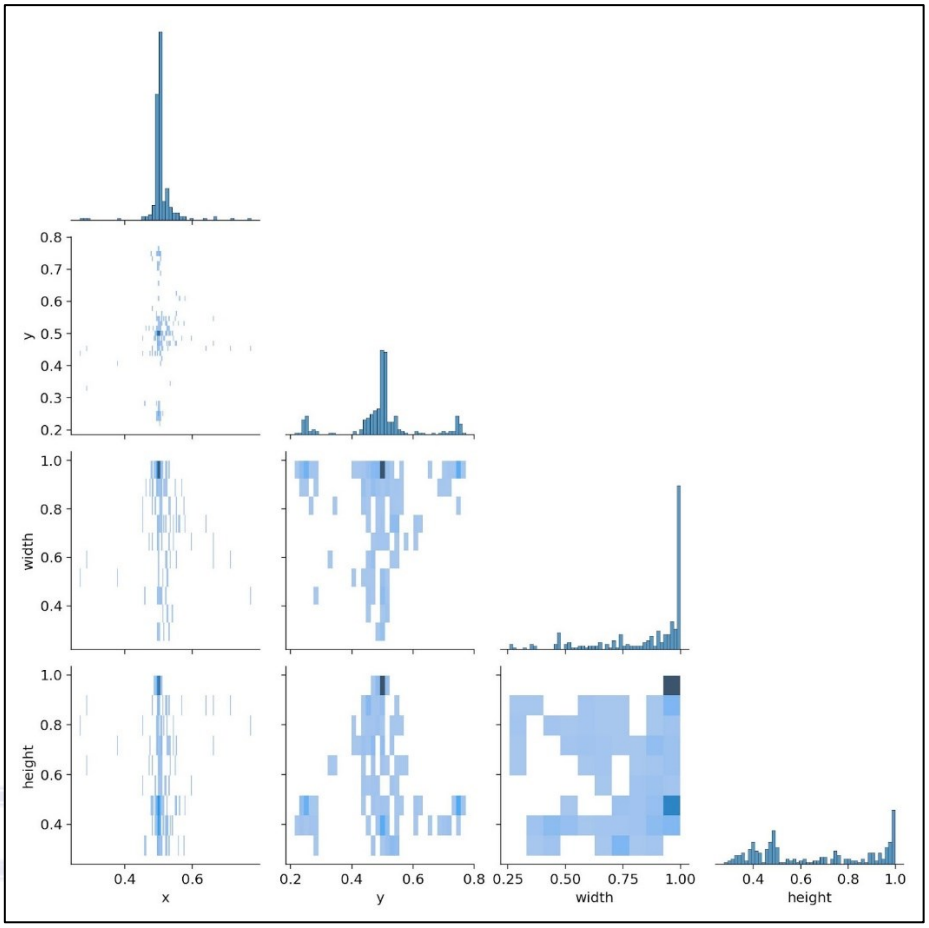


Figure 4-7 Currency detection Intersection over Union

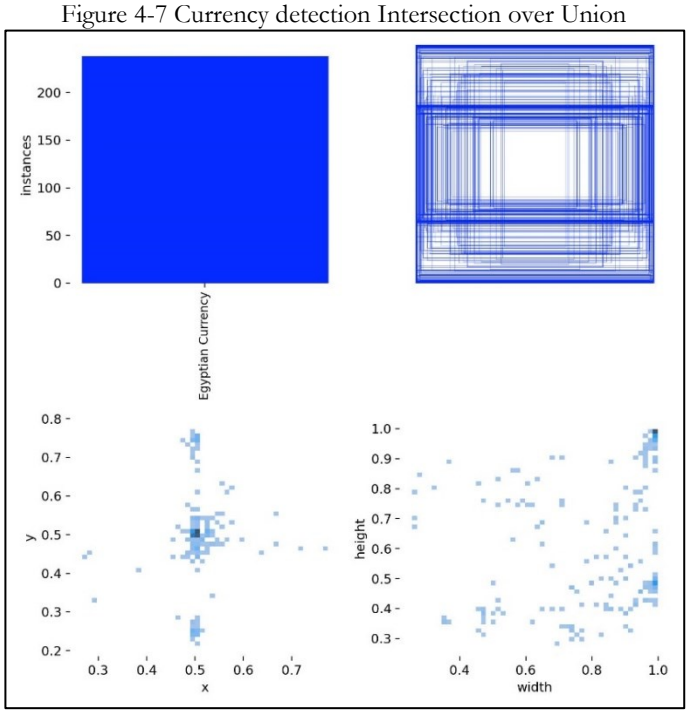


Figure 4-8 Currency detection Mean Average Precision

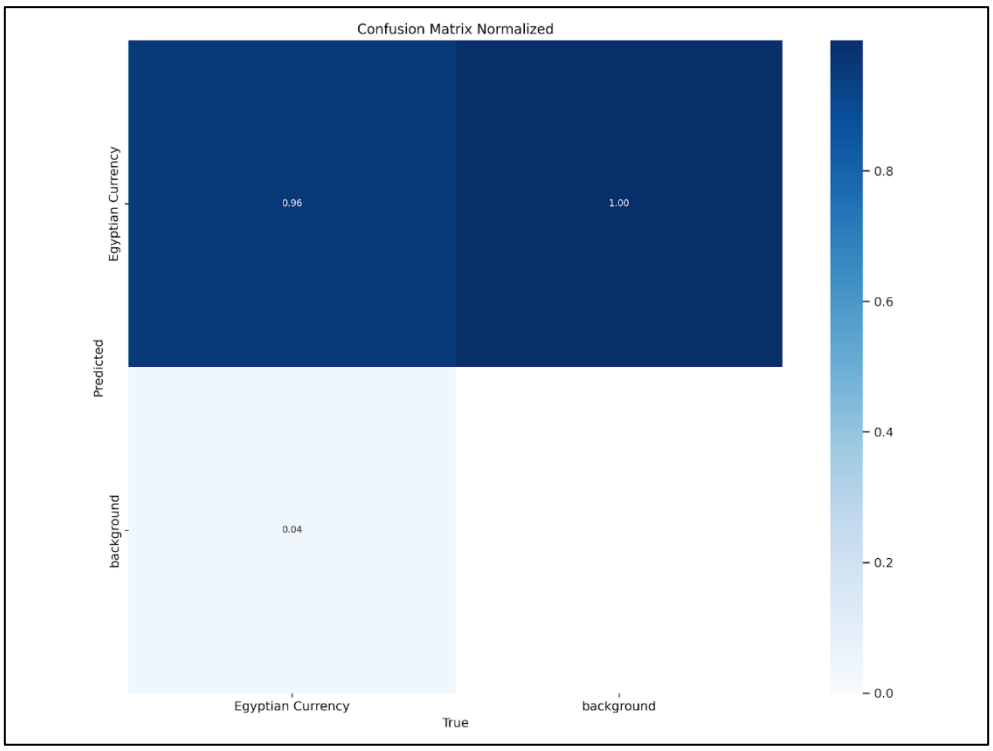


Figure 4-9 Currency detection Confusion matrix

Figure 4-10 Currency detection output examples

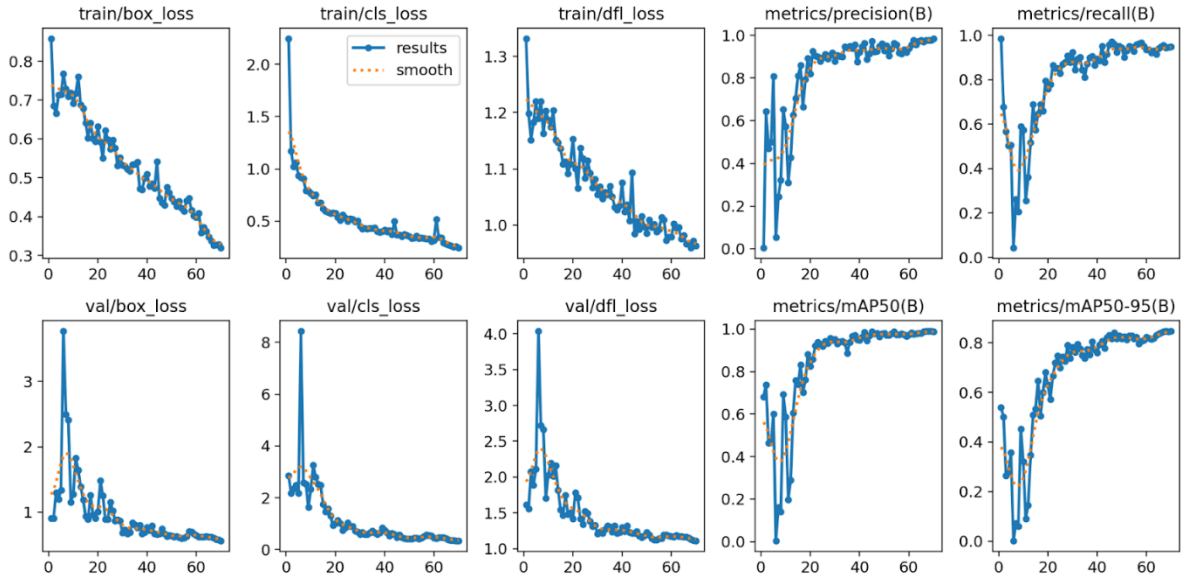


Figure 4-11 Currency detection Mean Average Precision Specifically

▪ CCN Currency Identification

The primary function of the system is to classify Egyptian currency in real-time using a lightweight convolutional neural network (CNN) optimized for edge devices.

This functionality is critical for supporting visually impaired individuals to recognize different banknotes independently and accurately.

The core capabilities of the system include:

- Currency classification (5, 10, 10_new, 20, 20_new, 50, 100, 200 EGP).
- Preprocessing input frames from the camera
- Real-time inference on embedded hardware like Raspberry Pi to predict the currency class

Model

We utilized the **MobileNetV2** architecture as the backbone for the currency classification task, it is a lightweight and efficient deep convolutional neural network, specifically designed for mobile and embedded vision applications.

The architecture of MobileNetV2 consists of an initial fully convolutional layer with 32 filters, followed by 19 residual bottleneck layers, as described in the Table. We use ReLU6 as the non-linearity because of its robustness when used with low-precision computation. We always use kernel size 3*3 as is standard for modern networks and utilize dropout and batch normalization during training. With the exception of the first layer, we use a constant expansion rate throughout the network. In experiments, it was found that expansion rates between 5 and 10 result in nearly identical performance curves, with smaller networks being better off with slightly smaller expansion rates and larger networks having slightly better performance with larger expansion rates. For all experiments, an expansion factor of 6 was used, applied to the size of the input tensor. For example, for a bottleneck layer that takes a 64-channel input tensor and produces a tensor with 128 channels, the intermediate expansion layer is then 64 * 6 = 384 channels.

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 4-12 MobileNetV2 layers

All layers in the same sequence have the same number of output channels. The first layer of each sequence has a stride, and all others use stride 1. All spatial convolutions use 3*3 kernels. The expansion factor is always applied to the input size as described in the Table.

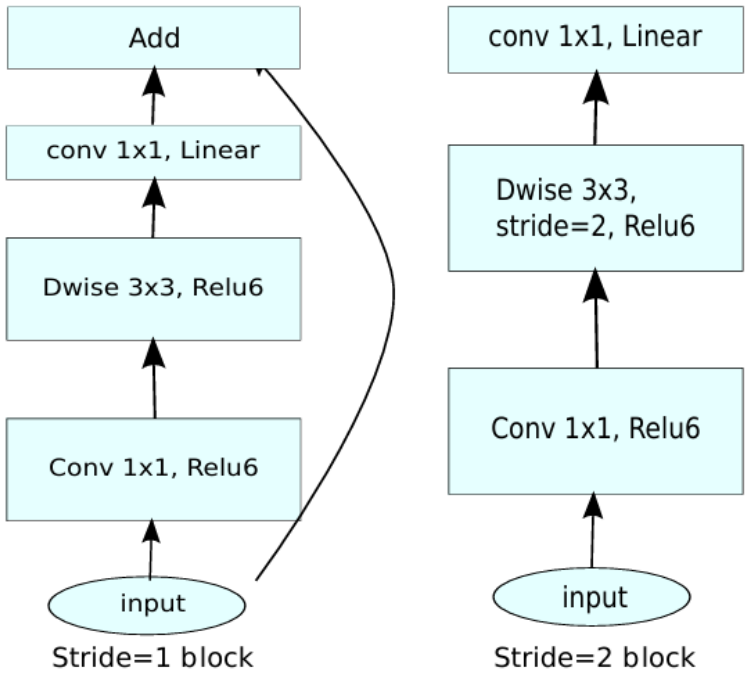


Figure 4-13 MobileNetV2

Advantages:

- Fast and efficient — ideal for real-time classification on Raspberry Pi.
- Low model size (~3.4M params) — easy to deploy and optimize.
- Accurate — competitive with larger models when fine-tuned.

Architecture Overview

We fine-tuned MobileNetV2 using a custom dataset consisting of eight Egyptian currency classes: 5 EGP, 10 EGP, 10_new EGP, 20 EGP, 20_new EGP, 50 EGP, 100 EGP, 200 EGP. These images were collected under diverse lighting conditions, different angles, and backgrounds to ensure the robustness of the classifier in real-world scenarios to build a CNN Architecture.

The overall model architecture follows the standard MobileNetV2 layout with the following structure:

- **Input Layer:** 224x224x3 RGB images
- **Base Network:** Pretrained MobileNetV2 layers (excluding the classifier) Global Average Pooling Layer
- **Fully Connected Layer:** Dense layer with 8 output neurons (for 8 classes)
- **Activation:** Softmax for multi-class classification

```
# MobileNetV2
from tensorflow.keras.applications import MobileNetV2

base_model = MobileNetV2(alpha=1.0, weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Figure 4-14 currency MobileNetV2 hands on

block_16_expand_re… (ReLU)	(None, 7, 7, 960)	0	block_16_expand_…
block_16_depthwise (DepthwiseConv2D)	(None, 7, 7, 960)	8,640	block_16_expand_…
block_16_depthwise… (BatchNormalizatio…	(None, 7, 7, 960)	3,840	block_16_depthwi…
block_16_depthwise… (ReLU)	(None, 7, 7, 960)	0	block_16_depthwi…
block_16_project (Conv2D)	(None, 7, 7, 320)	307,200	block_16_depthwi…
block_16_project_BN (BatchNormalizatio…	(None, 7, 7, 320)	1,280	block_16_project…
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409,600	block_16_project…
Conv_1_bn (BatchNormalizatio…	(None, 7, 7, 1280)	5,120	Conv_1[0][0]
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn[0][0]
global_average_poo… (GlobalAveragePool…	(None, 1280)	0	out_relu[0][0]
predictions (Dense)	(None, 8)	10,248	global_average_p…

Total params: 2,268,232 (8.65 MB)

Trainable params: 2,234,120 (8.52 MB)

Figure 4-15 currency recognition layers

Custom Training Details

The training setup was optimized to balance accuracy and performance for deployment on embedded hardware:

- **Image Resolution:** 224x224x3
- **Loss Function:** Categorical Cross-Entropy
- **Optimizer:** Adam
- **Learning Rate:** 0.0001
- **Batch Size:** 32
- **Epochs:** 100
- **Data Augmentation:** Random rotation, flipping, width_shift ,height_shift,shear,zoom adjustment to increase model generalization
- **Train/Validation/test:** 72/20/8
- **Label Format:** One-hot encoded vectors corresponding to the 8 currency classes

```
Epoch 90/100
81/81 ━━━━━━━━━━━━━━━━━━━━ 24s 275ms/step - accuracy: 0.9535 - loss: 0.1540 - val_accuracy: 0.9568 - val_loss: 0.1390
Epoch 91/100
81/81 ━━━━━━━━━━━━━━━━━━━━ 25s 278ms/step - accuracy: 0.9553 - loss: 0.1419 - val_accuracy: 0.9500 - val_loss: 0.1642
Epoch 92/100
81/81 ━━━━━━━━━━━━━━━━━━━━ 25s 279ms/step - accuracy: 0.9575 - loss: 0.1288 - val_accuracy: 0.9662 - val_loss: 0.1217
Epoch 93/100
81/81 ━━━━━━━━━━━━━━━━━━━━ 24s 277ms/step - accuracy: 0.9459 - loss: 0.1747 - val_accuracy: 0.9595 - val_loss: 0.1217
Epoch 94/100
81/81 ━━━━━━━━━━━━━━━━━━━━ 24s 273ms/step - accuracy: 0.9571 - loss: 0.1342 - val_accuracy: 0.9541 - val_loss: 0.1290
Epoch 95/100
81/81 ━━━━━━━━━━━━━━━━━━━━ 24s 276ms/step - accuracy: 0.9606 - loss: 0.1203 - val_accuracy: 0.9595 - val_loss: 0.1219
Epoch 96/100
81/81 ━━━━━━━━━━━━━━━━━━━━ 24s 277ms/step - accuracy: 0.9537 - loss: 0.1442 - val_accuracy: 0.9541 - val_loss: 0.1396
Epoch 97/100
81/81 ━━━━━━━━━━━━━━━━━━━━ 24s 275ms/step - accuracy: 0.9540 - loss: 0.1421 - val_accuracy: 0.9595 - val_loss: 0.1240
Epoch 98/100
81/81 ━━━━━━━━━━━━━━━━━━━━ 24s 276ms/step - accuracy: 0.9651 - loss: 0.1215 - val_accuracy: 0.9527 - val_loss: 0.1600
Epoch 98: early stopping
Restoring model weights from the end of the best epoch: 68.
```

Figure 4-16 currency recognition classes accuracy

```
[22]: plot_history(history)
```

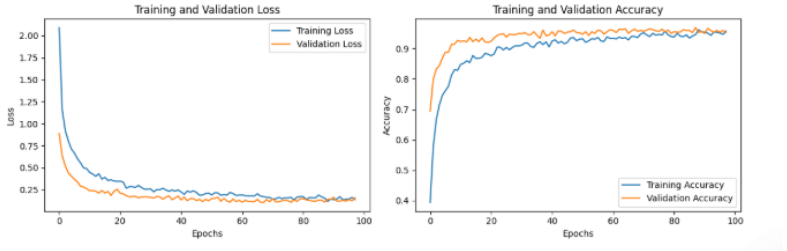


Figure 4-17 currency recognition history plotting

Deployment After training

After training, the model was optimized for real-time inference and deployed on a Raspberry Pi.

The following optimizations were applied:

- **Model Quantization:** Converted to TensorFlow Lite with float32 quantization for faster inference and reduced size
- **Input Pipeline:** Captures frames from a camera, resizes to 224x224, and normalizes pixel values
- **Real-Time Feedback**: Based on the predicted class.

Dataset

Egyptian Currency Classification Dataset for CNN. To build a CNN-based classification model, we first collected images of Egyptian banknotes across various categories.

This classification step is intended to work after the detection stage, where the detected banknote is passed to the CNN for final classification.

The dataset includes the following classes: 5 EGP 10 EGP (old design) 10 EGP (new design) 20 EGP 20 EGP (new design) 50 EGP 100 EGP 200 EGP Each class is assigned a label from 0 to 7.

The dataset is organized in a folder named currency_dataset, which contains three main subfolders: train/ val/ test/ Inside each of these folders, there are eight subfolders, one for each class. Each subfolder is named according to its class label (from 0 to 7) and contains the corresponding images.

This structured format is ideal for training and evaluating deep learning models for banknote classification.

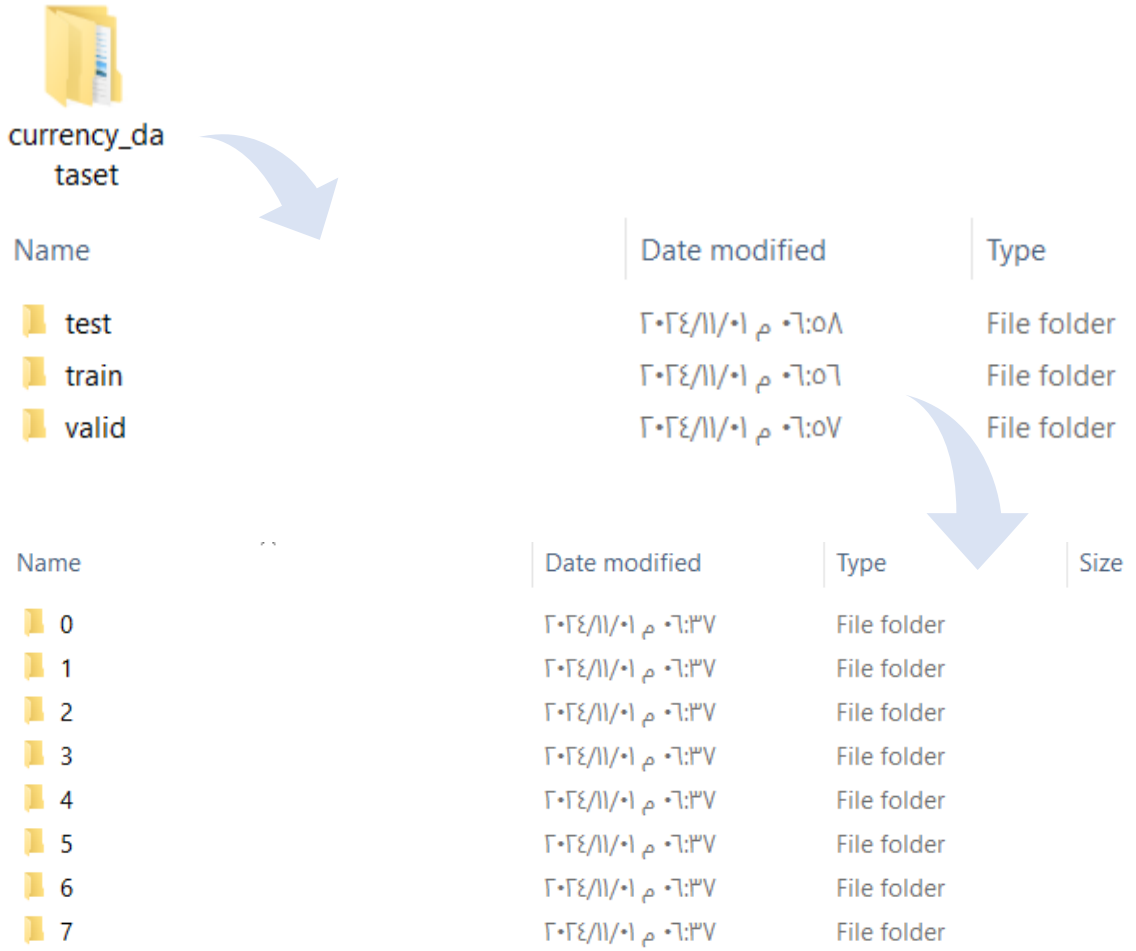


Figure 4-18 Currency Recognition dataset

```
[23]:
train_images = train_images.astype('float32') / 255.0
valid_images = valid_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0

# Image statistics
def image_statistics(images, dataset_name):
    print(f"Dataset: {dataset_name}")
    print(f"Number of images: {len(images)}")
    print(f"Image shapes: {images.shape}")
    print("------------------------------")

image_statistics(train_images, "Train")
image_statistics(valid_images, "Validation")
image_statistics(test_images, "Test")
```

```
Dataset: Train
Number of images: 2577
Image shapes: (2577, 224, 224, 3)
------------------------------
Dataset: Validation
Number of images: 740
Image shapes: (740, 224, 224, 3)
------------------------------
Dataset: Test
Number of images: 270
Image shapes: (270, 224, 224, 3)
------------------------------
```

Figure 4-19 Currency Recognition dataset deviding

```
# Plot training data class distribution
plt.figure(figsize=(12, 6))
pd.DataFrame(train_labels).value_counts().sort_index().plot(kind='bar')
plt.title('Training Data Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```

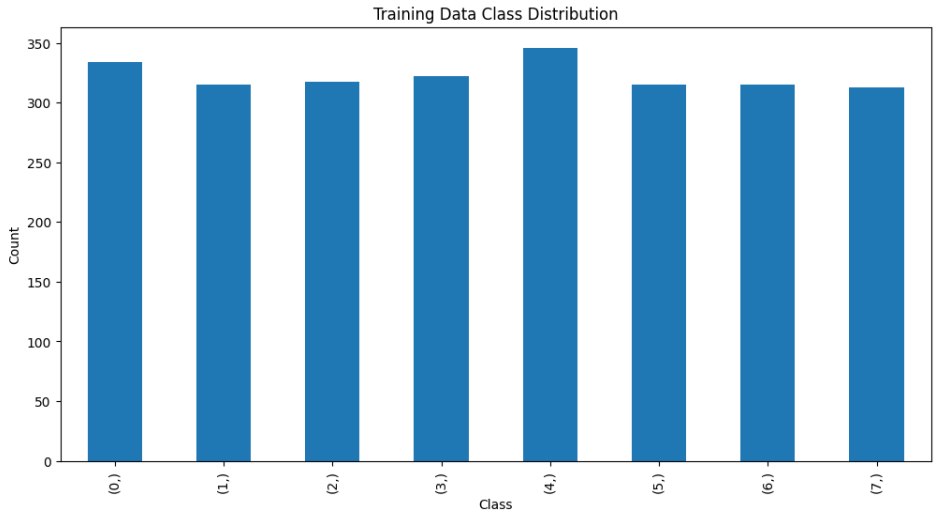


Figure 4-20 Training Currency Recognition distribution

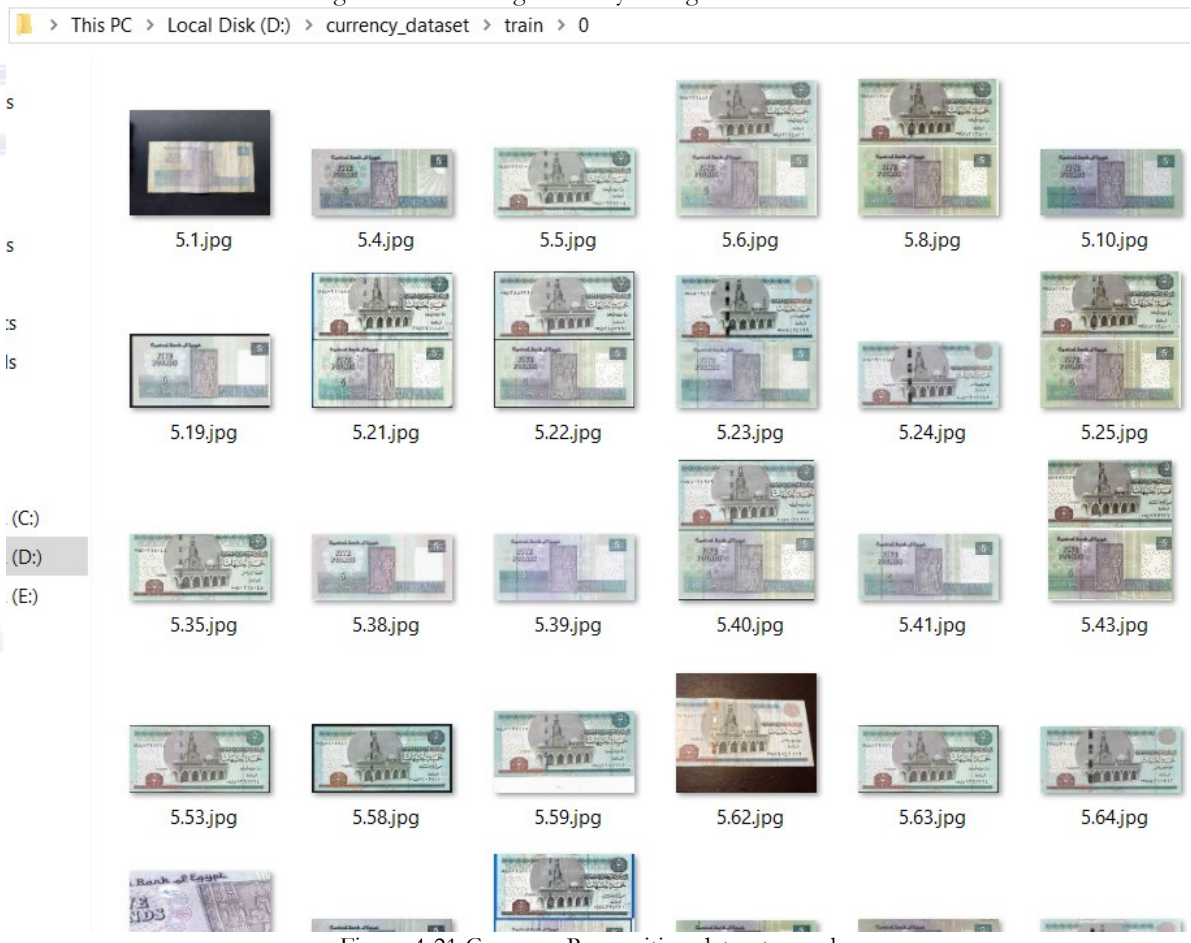


Figure 4-21 Currency Recognition dataset examles

Evaluation Metrics

The Percentage of accuracy metric is ~96 % , and the visualization of another metrics is :

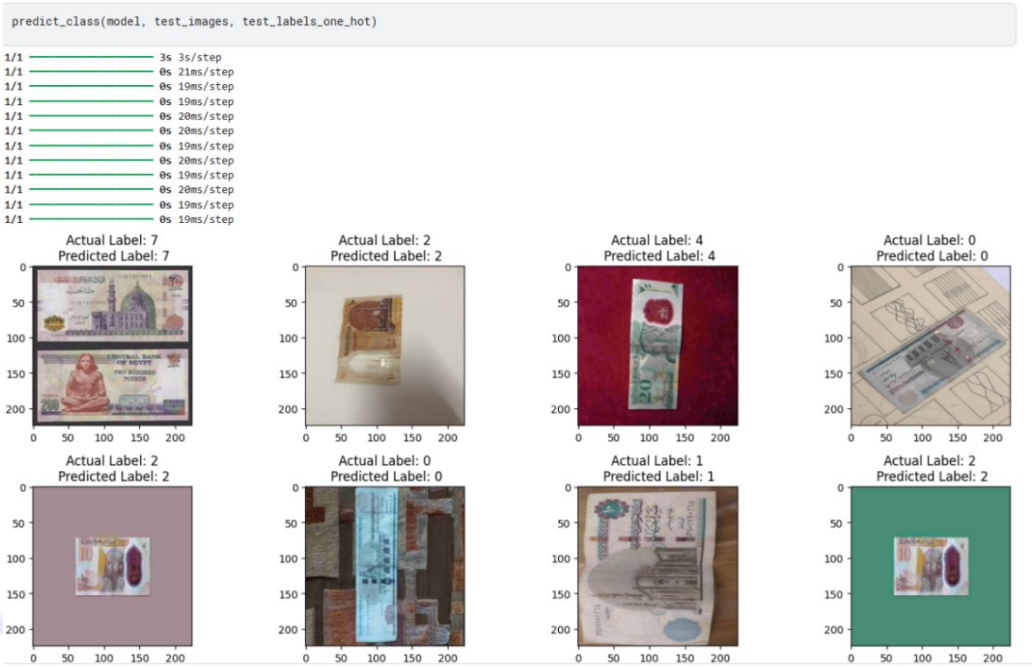


Figure 4-22 Currency Recognition prediction examles

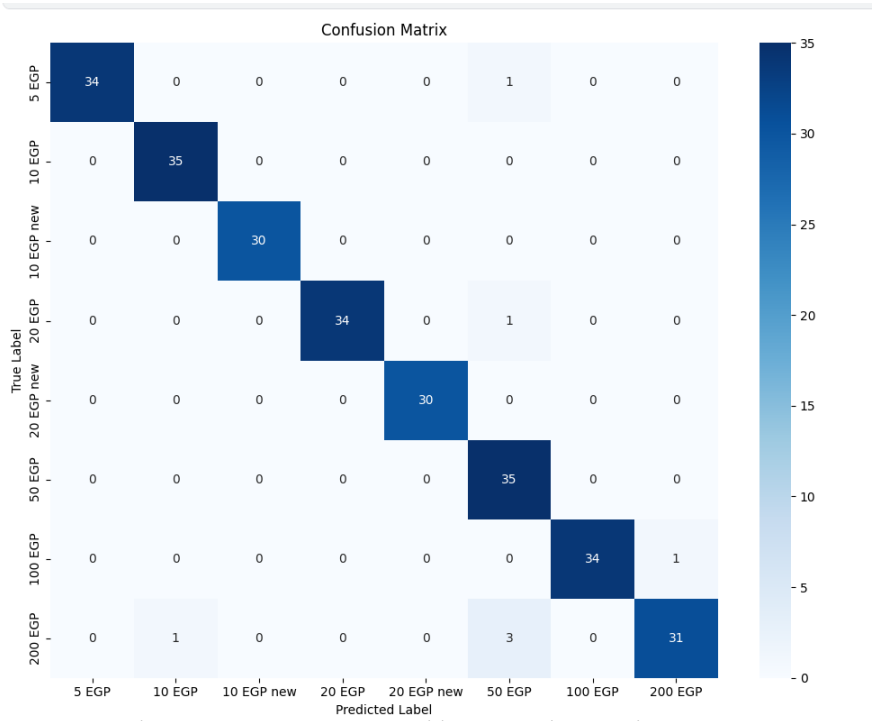


Figure 4-23 Currency Recognition confusion matrix

```
Classification Report:
5 EGP        => Precision: 1.00, Recall: 0.97, F1-score: 0.99, Support: 35
10 EGP       => Precision: 0.97, Recall: 1.00, F1-score: 0.99, Support: 35
10 EGP new   => Precision: 1.00, Recall: 1.00, F1-score: 1.00, Support: 30
20 EGP       => Precision: 1.00, Recall: 0.97, F1-score: 0.99, Support: 35
20 EGP new   => Precision: 1.00, Recall: 1.00, F1-score: 1.00, Support: 30
50 EGP       => Precision: 0.88, Recall: 1.00, F1-score: 0.93, Support: 35
100 EGP      => Precision: 1.00, Recall: 0.97, F1-score: 0.99, Support: 35
200 EGP      => Precision: 0.97, Recall: 0.89, F1-score: 0.93, Support: 35
accuracy     => 0.97
macro avg    => {'precision': 0.98, 'recall': 0.98, 'f1-score': 0.98, 'support': 270}
weighted avg => {'precision': 0.98, 'recall': 0.97, 'f1-score': 0.97, 'support': 270}
```

Figure 4-24 Currency Recognition classification report

4.2.2 FDRE Flow:

- **Face Detection**

Computer vision technology is used to identify and locate human faces in digital images or video frames. Face detection algorithms analyze visual data to find patterns that resemble human facial features (like eyes, nose, and mouth) and determine the coordinates of the face(s) within the frame.

Computer vision technology is used to identify and locate human faces in digital images or video frames. Face detection algorithms analyze visual data to find patterns that resemble human facial features (like eyes, nose, and mouth) and determine the coordinates of the face(s) within the frame.

For face detection, we chose Haar Cascade because it's lightweight and efficient for real-time applications on devices with limited resources like the Raspberry Pi. Other common methods include HOG (Histogram of Oriented Gradients), Dlib CNN-based detector, MTCNN (Multi-task Cascaded Convolutional Networks), and Retina Face.

Model

The Haar Cascade Algorithm, developed by Paul Viola and Michael Jones, are well-known for their significant contribution to object detection. Rapid object detection using a boosted cascade of simple features by introducing efficient methods for feature computation, such as the "integral image "representation, and utilizing learning algorithms like AdaBoost. According to the study "Use of Haar Cascade Classifier for Face Tracking System in Real Time Video. they introduced a comprehensive face detection and tracking system designed for real-time video inputs, emphasizing its application in security contexts. Incorporating the Haar-Cascade method and OpenCV libraries, the system's initial phase encompasses face recognition and detection. Subsequently, face tracking is performed using a Face clustering algorithm. The system primarily targets security purposes, operating on video recordings from public areas to identify and track individuals or suspicious activities.

Architecture

It introduced a fast and accurate approach by combining three key innovations, Haar-like features, Integral images and AdaBoost learning with cascade architecture:

Haar-like Features

- These are rectangular features that resemble Haar wavelets (white and black rectangles).
- They capture patterns like edges, lines, and changes in texture.
- Types include: two-rectangle, three-rectangle, and four-rectangle features.
- Compared to pixel intensity, these features are more robust and computationally simple.

Integral Image

- A summed-area table used to rapidly calculate Haar-like feature values.
- Each pixel in the integral image contains the sum of all pixels above and to the left.
- This allows any rectangular sum to be computed in constant time (O (1)).
- Equation: ii (x, y) = sum_{x' ≤ x, y' ≤ y} i(x', y')

AdaBoost for Feature Selection

- From over 180,000 features in a 24x24 window, **AdaBoost** selects a small subset (~200) that are most discriminative.
- Each selected feature is associated with a **weak classifier**.
- AdaBoost combines them into a **strong classifier** using a weighted vote.
- Helps reduce computation by ignoring less informative features.

Cascade of Classifiers

- A **multi-stage decision pipeline** where each stage contains a strong classifier.
- Early stages quickly reject negative windows (non-face).
- Only promising regions move forward to later stages for detailed evaluation.

- This dramatically speeds up detection (~10x faster than sliding window alone).

▪ CNN Face Recognition

The primary function of the system is to classify faces in real-time using a lightweight convolutional neural network (CNN) optimized for edge devices. This functionality is critical for supporting visually impaired individuals to recognize faces independently and accurately. The core capabilities of the system include Face Recognition (Nada, Nooran, Cristiano Ronaldo, Leonardo DiCaprio, Magdy Yacoub, Mohamed Salah), preprocessing input frames from the camera, and real-time inference on embedded hardware like Raspberry Pi to predict the face class.

Model

We utilized the **MobileNetV2** architecture as the backbone for the face recognition task. MobileNetV2 is a lightweight and efficient deep convolutional neural network, specifically designed for mobile and embedded vision applications. All information about the MobileNetV2 architecture was mentioned previously.

Architecture Overview

We fine-tuned MobileNetV2 using a custom dataset consisting of 6 face classes: Nada, Nooran, Cristiano Ronaldo, Leonardo DiCaprio, Magdy Yacoub Mohamed Salah. These images were collected under diverse lighting conditions, different angles, and backgrounds to ensure the robustness of the classifier in real-world scenarios to build a CNN Architecture.

The overall model architecture follows the standard MobileNetV2 layout with the following structure:

- **Input Layer:** 224x224x3 RGB images
- **Base Network:** Pretrained MobileNetV2 layers (excluding the classifier) Global Average Pooling Layer
- **Fully Connected Layer:** Dense layer with 6 output neurons (for 6 classes)
- **Activation:** SoftMax for multi-class classification

```
45]: mobile = tf.keras.applications.mobilenet.MobileNet()
     mobile.summary

45]: <bound method Model.summary of <Functional name=mobilenet_1.00_224, built=True>>

49]: x = mobile.layers[-6].output
     x

49]: <KerasTensor shape=(None, 7, 7, 1024), dtype=float32, sparse=False, name=keras_tensor_85>

67]: from tensorflow.keras.layers import GlobalAveragePooling2D
     x = GlobalAveragePooling2D()(x)  # Convert 7x7x6 to 6

69]: output = Dense(units=6 , activation='softmax')(x)

71]: model = Model(inputs = mobile.input , outputs=output)

73]: for layer in model.layers[:-23]:
         layer.trainable = False
```

Figure 4-25 Face recognition MobileNetV2 hands on

conv_dw_11_relu (ReLU)	(None, 14, 14, 512)	0
conv_pw_11 (Conv2D)	(None, 14, 14, 512)	262,144
conv_pw_11_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_11_relu (ReLU)	(None, 14, 14, 512)	0
conv_pad_12 (ZeroPadding2D)	(None, 15, 15, 512)	0
conv_dw_12 (DepthwiseConv2D)	(None, 7, 7, 512)	4,608
conv_dw_12_bn (BatchNormalization)	(None, 7, 7, 512)	2,048
conv_dw_12_relu (ReLU)	(None, 7, 7, 512)	0
conv_pw_12 (Conv2D)	(None, 7, 7, 1024)	524,288
conv_pw_12_bn (BatchNormalization)	(None, 7, 7, 1024)	4,096
conv_pw_12_relu (ReLU)	(None, 7, 7, 1024)	0
conv_dw_13 (DepthwiseConv2D)	(None, 7, 7, 1024)	9,216
conv_dw_13_bn (BatchNormalization)	(None, 7, 7, 1024)	4,096
conv_dw_13_relu (ReLU)	(None, 7, 7, 1024)	0
conv_pw_13 (Conv2D)	(None, 7, 7, 1024)	1,048,576
conv_pw_13_bn (BatchNormalization)	(None, 7, 7, 1024)	4,096
conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1024)	0
dense_1 (Dense)	(None, 6)	6,150

Total params: 3,235,014 (12.34 MB)

Trainable params: 1,869,830 (7.13 MB)

Non-trainable params: 1,365,184 (5.21 MB)

Figure 4-26 Face Recognition layers

Custom Training Details

The training setup was optimized to balance accuracy and performance for deployment on embedded hardware:

- **Image Resolution:** 224x224x3
- **Loss Function:** Categorical Cross-Entropy
- **Optimizer:** Adam
- **Learning Rate:** 0.0001
- **Batch Size:** 10
- **Epochs:** 30
- **Data Augmentation:** rotation up to 30, flipping, width_shift, height_shift, shear, zoom adjustment to increase model generalization
- **Train/Validation/test:** 71/20/9
- **Label Format:** One-hot encoded vectors corresponding to the 6 currency classes.

```
Epoch 6/30
89/89 - 16s - 178ms/step - accuracy: 0.9966 - loss: 0.0170 - val_accuracy: 0.9792 - val_loss: 0.0423
Epoch 7/30
89/89 - 16s - 177ms/step - accuracy: 0.9989 - loss: 0.0104 - val_accuracy: 0.9875 - val_loss: 0.0377
Epoch 8/30
89/89 - 16s - 176ms/step - accuracy: 0.9989 - loss: 0.0104 - val_accuracy: 0.9875 - val_loss: 0.0405
Epoch 9/30
89/89 - 16s - 176ms/step - accuracy: 0.9989 - loss: 0.0090 - val_accuracy: 0.9833 - val_loss: 0.0483
Epoch 10/30
89/89 - 15s - 174ms/step - accuracy: 0.9989 - loss: 0.0109 - val_accuracy: 0.9875 - val_loss: 0.0495
Epoch 11/30
89/89 - 16s - 175ms/step - accuracy: 0.9977 - loss: 0.0105 - val_accuracy: 0.9917 - val_loss: 0.0430
Epoch 12/30
89/89 - 16s - 175ms/step - accuracy: 0.9966 - loss: 0.0145 - val_accuracy: 0.9750 - val_loss: 0.0759
Epoch 13/30
89/89 - 15s - 174ms/step - accuracy: 0.9989 - loss: 0.0103 - val_accuracy: 0.9917 - val_loss: 0.0432
Epoch 14/30
89/89 - 16s - 182ms/step - accuracy: 0.9989 - loss: 0.0073 - val_accuracy: 0.9917 - val_loss: 0.0286
Epoch 15/30
89/89 - 15s - 173ms/step - accuracy: 0.9977 - loss: 0.0105 - val_accuracy: 0.9917 - val_loss: 0.0253
Epoch 16/30
89/89 - 16s - 180ms/step - accuracy: 0.9989 - loss: 0.0087 - val_accuracy: 0.9833 - val_loss: 0.0401
Epoch 17/30
89/89 - 16s - 175ms/step - accuracy: 0.9989 - loss: 0.0069 - val_accuracy: 0.9875 - val_loss: 0.0387
Epoch 18/30
89/89 - 16s - 175ms/step - accuracy: 0.9989 - loss: 0.0053 - val_accuracy: 0.9875 - val_loss: 0.0423
Epoch 19/30
89/89 - 15s - 173ms/step - accuracy: 1.0000 - loss: 0.0062 - val_accuracy: 0.9875 - val_loss: 0.0386
Epoch 20/30
89/89 - 15s - 172ms/step - accuracy: 0.9989 - loss: 0.0046 - val_accuracy: 0.9917 - val_loss: 0.0352
Epoch 21/30
89/89 - 17s - 189ms/step - accuracy: 0.9989 - loss: 0.0048 - val_accuracy: 0.9875 - val_loss: 0.0315
Epoch 22/30
89/89 - 16s - 177ms/step - accuracy: 0.9977 - loss: 0.0081 - val_accuracy: 0.9875 - val_loss: 0.0397
Epoch 23/30
89/89 - 15s - 173ms/step - accuracy: 1.0000 - loss: 0.0047 - val_accuracy: 0.9792 - val_loss: 0.0402
Epoch 24/30
89/89 - 16s - 179ms/step - accuracy: 0.9977 - loss: 0.0074 - val_accuracy: 0.9792 - val_loss: 0.0500
Epoch 25/30
89/89 - 15s - 173ms/step - accuracy: 1.0000 - loss: 0.0032 - val_accuracy: 0.9875 - val_loss: 0.0295
Epoch 26/30
89/89 - 16s - 176ms/step - accuracy: 1.0000 - loss: 0.0047 - val_accuracy: 0.9917 - val_loss: 0.0220
Epoch 27/30
89/89 - 15s - 174ms/step - accuracy: 0.9989 - loss: 0.0057 - val_accuracy: 0.9917 - val_loss: 0.0267
Epoch 28/30
89/89 - 16s - 175ms/step - accuracy: 0.9989 - loss: 0.0032 - val_accuracy: 0.9917 - val_loss: 0.0259
Epoch 29/30
89/89 - 15s - 173ms/step - accuracy: 1.0000 - loss: 0.0041 - val_accuracy: 0.9917 - val_loss: 0.0227
Epoch 30/30
89/89 - 15s - 174ms/step - accuracy: 0.9977 - loss: 0.0080 - val_accuracy: 0.9875 - val_loss: 0.0299
80]:  <keras.src.callbacks.history.History at 0x1d461e447c0>
```

Figure 4-27 Face Recognition training

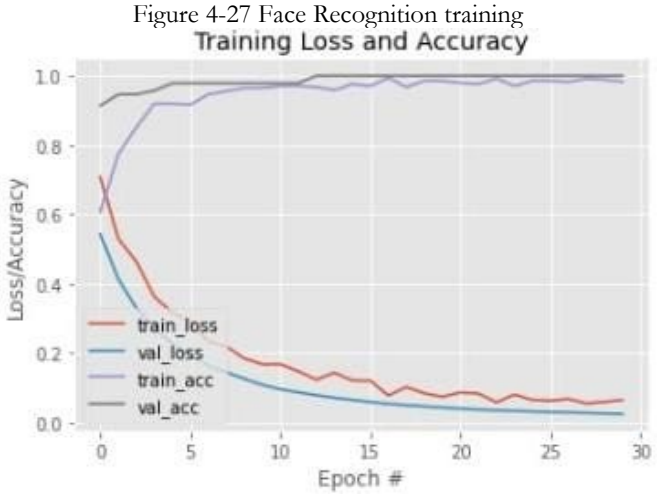


Figure 4-28 Face Recognition training visualization

Deployment After training

After training, the model was optimized for real-time inference and deployed on a Raspberry Pi.

The following optimizations were applied:

- **Model Quantization:** Converted to TensorFlow Lite with float32 quantization for faster inference and reduced size
- **Input Pipeline:** Captures frames from a camera, resizes to 224x224, and normalizes pixel values
- **Real-Time Feedback:** Based on the predicted class.

Dataset

To build a CNN-based classification model, we first collected images of faces across various categories. This classification step is intended to work after the detection stage, where the detected face is passed to the CNN for final classification.

The dataset includes the following classes: Nada, Nooran, Cristiano Ronaldo, Leonardo DiCaprio, Magdy Yacoub, and Mohamed Salah. Each class is assigned a label from 0 to 5.

The dataset is organized in a folder named Final_FaceRECOGNTIOnn, which contains three main subfolders: train/, val/, and test/. Inside each of these folders, there are six subfolders, one for each class. Each subfolder is named according to

its class label (from 0 to 5) and contains the corresponding images. This structured format is ideal for training and evaluating deep learning models for face classification.

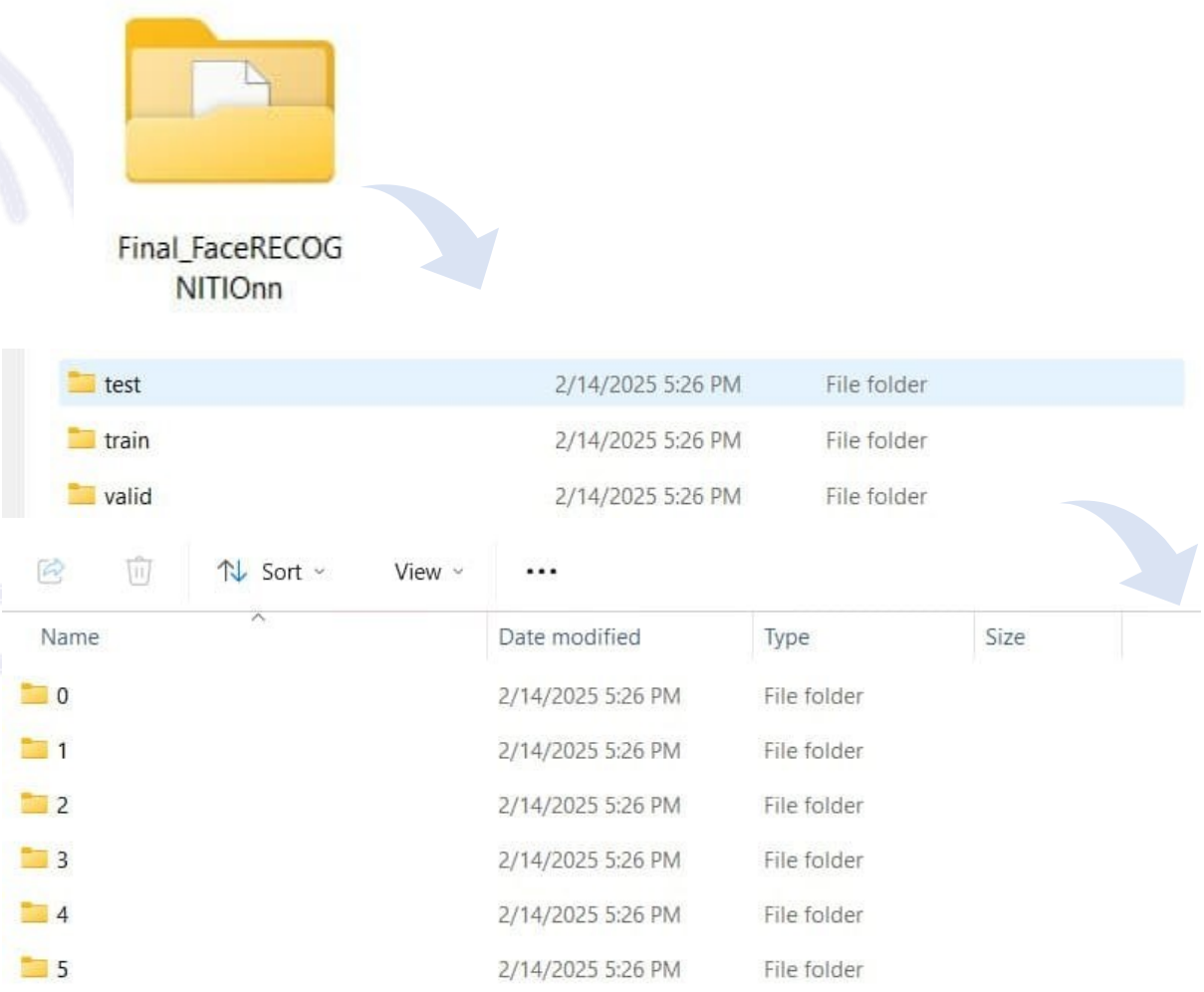


Figure 4-29 Face Recognition dataset

```
[40]: train_path= r'E:\Final_FaceRECOGNITIOnn\train'
valid_path= r'E:\Final_FaceRECOGNITIOnn\valid'
test_path= r'E:\Final_FaceRECOGNITIOnn\test'

train_batches = ImageDataGenerator(preprocessing_function = tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
    directory = train_path , target_size=(224,224) , batch_size=10)

valid_batches = ImageDataGenerator(preprocessing_function = tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
    directory = valid_path , target_size=(224,224) , batch_size=10)

test_batches = ImageDataGenerator(preprocessing_function = tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(
    directory = test_path , target_size=(224,224) , batch_size=10 , shuffle = False)

Found 881 images belonging to 6 classes.
Found 240 images belonging to 6 classes.
Found 120 images belonging to 6 classes.
```

Figure 4-30 Face Recognition dataset deviding

```
plt.figure(figsize=(12, 6))
pd.DataFrame(class_labels).value_counts().sort_index().plot(kind='bar')
plt.title('Training Data Class Distribution')
plt.xlabel('Class (Person)')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 4-31 Training Face Recognition distribution

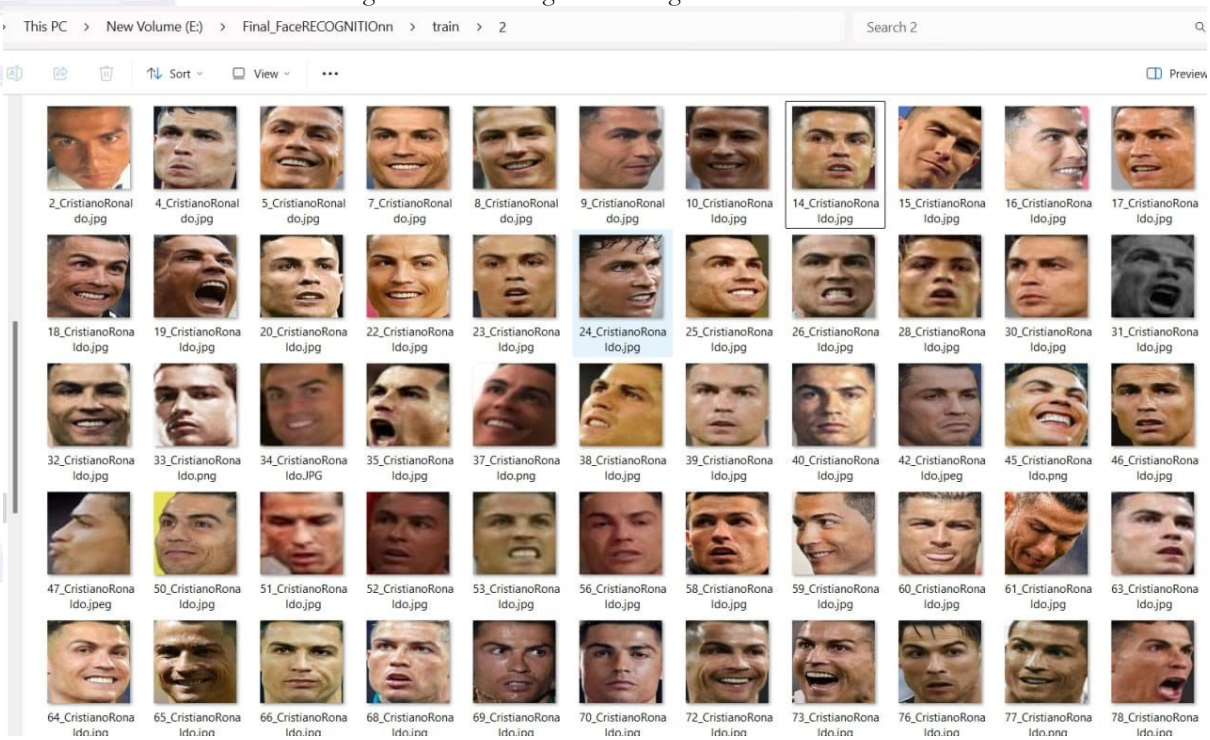


Figure 4-32 Face Recognition dataset examles

Evaluation Metrics

The Percentage of accuracy metric is ~99 %, and the visualization of another metrics is:

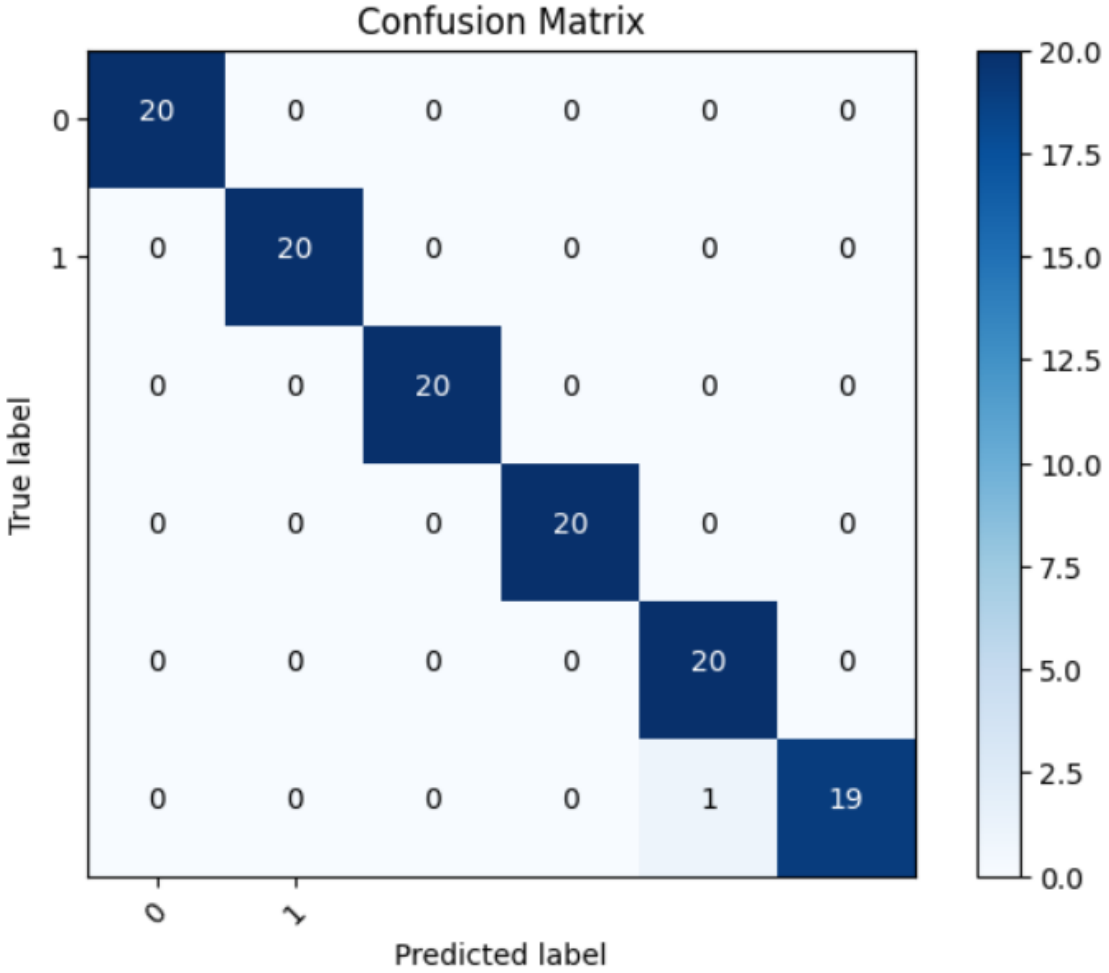


Figure 4-33 Face Recognition confusion matrix

```
]: print("Classification Report:\n", classification_report(test_labels, predicted_classes, target_names=class_labels))

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        20
           2       1.00      1.00      1.00        20
           3       1.00      1.00      1.00        20
           4       0.95      1.00      0.98        20
           5       1.00      0.95      0.97        20

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120
```

Figure 4-34 Face Recognition classification report

Figure 4-35 Face Recognition prediction examles

▪ CNN Expression Recognition

The primary function of the system is to classify expressions of faces in real-time using a lightweight convolutional neural network (CNN) optimized for edge devices. This functionality is critical for supporting visually impaired individuals to recognize expressions of faces independently and accurately. The core capabilities of the system include Expression Recognition (Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral), preprocessing input frames from the camera, and real-time inference on embedded hardware like Raspberry Pi to predict the expression class.

Model

We utilized the MobileNetV2 architecture as the backbone for the expression recognition task. It is a lightweight and efficient deep convolutional neural network, specifically designed for mobile and embedded vision applications. All information about the MobileNetV2 architecture was mentioned previously.

Architecture Overview

We fine-tuned MobileNetV2 using a FER2013 dataset consisting of 7 expression classes: Angry, Disgust, Fear, Happy, Sad, Surprise Neutral.

The overall model architecture follows the standard MobileNetV2 layout with the following structure:

- **Input Layer:** 224x224x3 RGB images
- **Base Network:** Pretrained MobileNetV2 layers (excluding the classifier), Global Average Pooling Layer
- **Fully Connected Layer:** Dense layer with 7 output neurons (for 6 classes)
- **Activation:** SoftMax for multi-class classification

```
# Build MobileNetV2 model
base_model = MobileNetV2(
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    include_top=False,
    weights='imagenet'
)
base_model.trainable = False

# Add custom classification head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
output = Dense(train_generator.num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=output)

# Compile model
model.compile(
    optimizer=Adam(),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
I0000 00:00:1747095726.021184      31 gpu_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB memory:  -> device: 0, name: Tesla T4, pci bus
id: 0000:00:04.0, compute capability: 7.5
I0000 00:00:1747095726.021881      31 gpu_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB memory:  -> device: 1, name: Tesla T4, pci bus
id: 0000:00:05.0, compute capability: 7.5
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Figure 4-36 Facial expression MobileNetV2 hands on

block_15_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_15_depthwise_BN…
block_15_project (Conv2D)	(None, 7, 7, 160)	153,600	block_15_depthwise_re…
block_15_project_BN (BatchNormalization)	(None, 7, 7, 160)	640	block_15_project[0][0]
block_15_add (Add)	(None, 7, 7, 160)	0	block_14_add[0][0], block_15_project_BN[0…
block_16_expand (Conv2D)	(None, 7, 7, 960)	153,600	block_15_add[0][0]
block_16_expand_BN (BatchNormalization)	(None, 7, 7, 960)	3,840	block_16_expand[0][0]
block_16_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_16_expand_BN[0]…
block_16_depthwise (DepthwiseConv2D)	(None, 7, 7, 960)	8,640	block_16_expand_relu[…
block_16_depthwise_BN (BatchNormalization)	(None, 7, 7, 960)	3,840	block_16_depthwise[0]…
block_16_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_16_depthwise_BN…
block_16_project (Conv2D)	(None, 7, 7, 320)	307,200	block_16_depthwise_re…
block_16_project_BN (BatchNormalization)	(None, 7, 7, 320)	1,280	block_16_project[0][0]
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409,600	block_16_project_BN[0…
Conv_1_bn (BatchNormalization)	(None, 7, 7, 1280)	5,120	Conv_1[0][0]
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn[0][0]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0	out_relu[0][0]
dropout (Dropout)	(None, 1280)	0	global_average_poolin…
dense (Dense)	(None, 7)	8,967	dropout[0][0]

Total params: 2,266,951 (8.65 MB)

Trainable params: 8,967 (35.03 KB)

Non-trainable params: 2,257,984 (8.61 MB)

Figure 4-37 Facial expression layers

Custom Training Details

The training setup was optimized to balance accuracy and performance for deployment on embedded hardware:

- **Image Resolution:** 224x224x3
- **Loss Function:** Sparse Categorical Cross Entropy
- **Optimizer:** Adam
- **Learning Rate**: automatic
- **Batch Size:** 64
- **Epochs:** 150
- **Train/val/test:** 80% / 20% from the main training dataset, 20% from the testing dataset as it is.
- **Label Format**: One-hot encoded vectors corresponding to the 7 currency classes

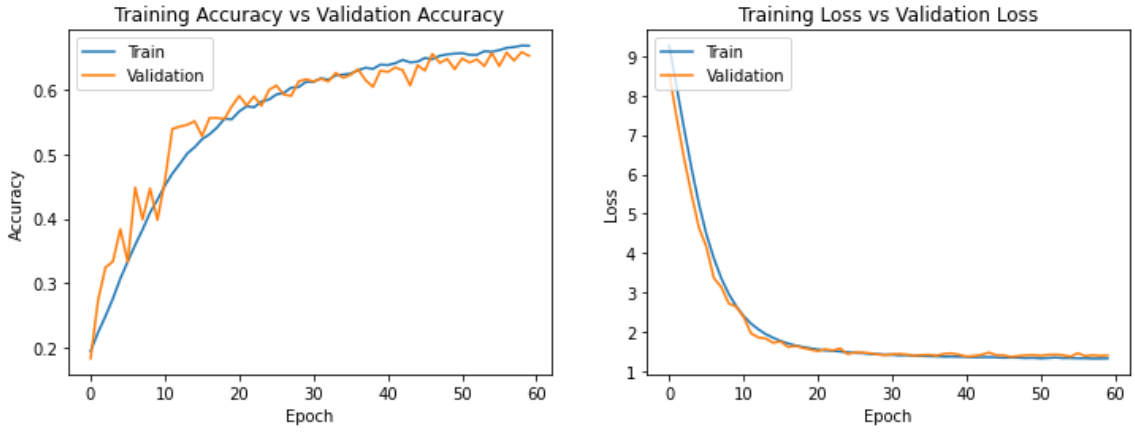


Figure 4-38 Facial expression training visualization

Deployment After training

After training, the model was optimized for real-time inference and deployed on a Raspberry Pi.

The following optimizations were applied:

- **Model Quantization:** Converted to TensorFlow Lite with float32 quantization for faster inference and reduced size
- **Input Pipeline:** Captures frames from a camera, resizes to 224x224, and normalizes pixel values
- **Real-Time Feedback:** Based on the predicted class.

Dataset

We use the FER2013 Dataset to build a CNN-based classification model. This classification step is intended to work after the detection stage, where the detected face is passed to the CNN for final classification. The dataset includes the following classes: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. Each class is assigned a label from 0 to 6. The dataset is organized in a folder named FER_2013_dataset, which contains two main subfolders: train/ and test/. Inside each of these folders, there are seven subfolders, one for each class. Each subfolder is named according to its class label (from 0 to 6) and contains the corresponding images. This structured format is ideal for training and evaluating deep learning models for face classification.

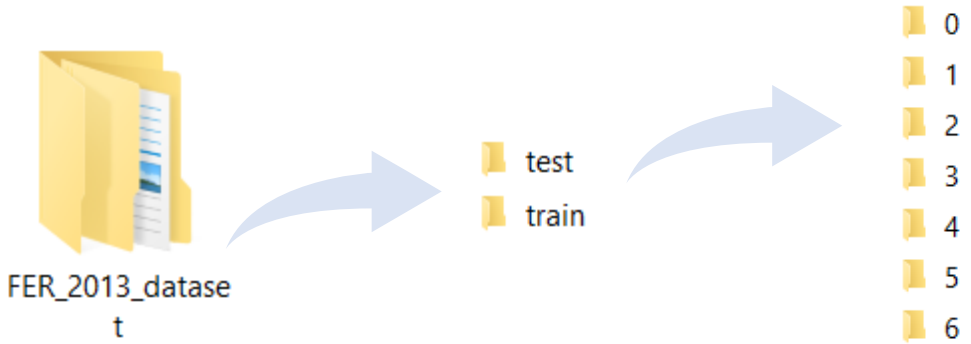


Figure 4-39 FER dataset

[6]:
```
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0

# Image statistics
def image_statistics(images, dataset_name):
    print(f"Dataset: {dataset_name}")
    print(f"Number of images: {len(images)}")
    print(f"Image shapes: {images.shape}")
    print("-----------------------------")

image_statistics(train_images, "Train")

image_statistics(test_images, "Test")
```

```
Dataset: Train
Number of images: 28709
Image shapes: (28709, 224, 224, 3)
-----------------------------
Dataset: Test
Number of images: 7178
Image shapes: (7178, 224, 224, 3)
-----------------------------
```

Figure 4-40 FER dataset deviding

[5]:
```
# Plot training data class distribution
plt.figure(figsize=(12, 6))
pd.DataFrame(train_labels).value_counts().sort_index().plot(kind='bar')
plt.title('Training Data Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```

Figure 4-41 Training Facial expression distribution

Figure 4-42 FER dataset examles

Evaluation Metrics

The Percentage of accuracy metric is ~60 %, and the visualization of another metrics is:

Figure 4-43 Facial expression confusion matrix

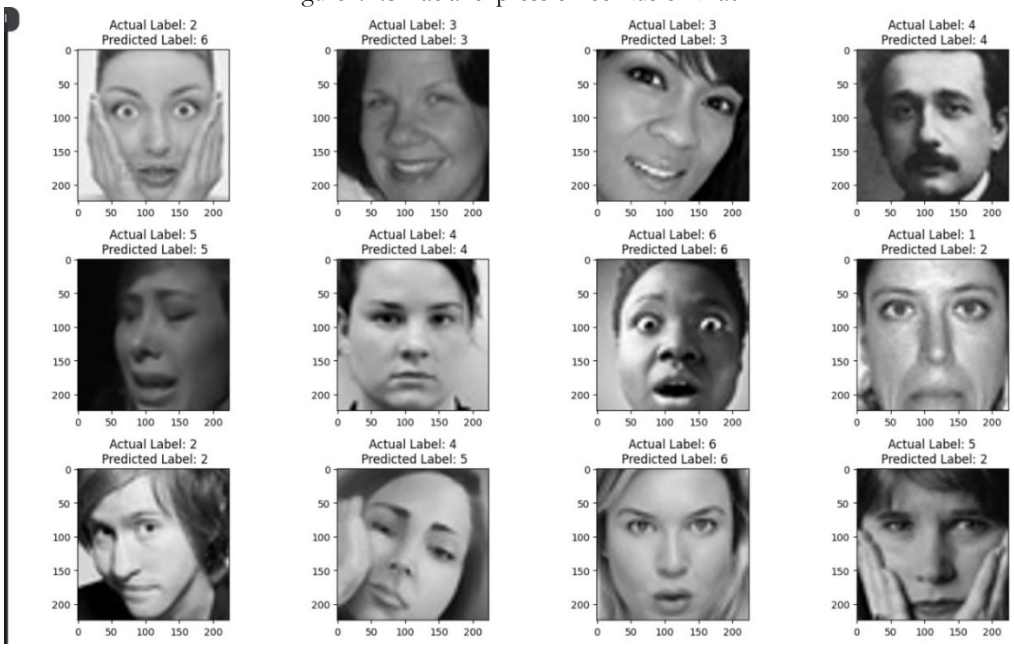


Figure 4-44 Facial expression prediction examles

4.2.3 SD Flow:

Model

Image captioning is the task of automatically generating a natural language description for a given image. It is a complex problem that lies at the intersection of computer vision and natural language processing (NLP). In this project, we used a pre-trained vision-language model called BLIP (Bootstrapped Language-Image Pretraining) to perform image captioning.

BLIP is a state-of-the-art model for vision-language tasks, including image captioning. It is trained with a combination of contrastive and language modeling objectives, making it highly accurate and effective, Due to hardware limitations on our deployment platform (e.g., no dedicated GPU on a Raspberry Pi or local machine), the BLIP model was executed entirely on CPU, While this results in slower inference (a few seconds per image), it proves the feasibility of running state-of-the-art AI models on modest hardware, and Supports zero-shot and fine-tuned captioning Performs well on general domain images Is available via Hugging Face Transformers.

Architecture Overview

The BLIP model consists of a vision transformer (ViT) as the visual encoder and a transformer-based language model as the decoder.

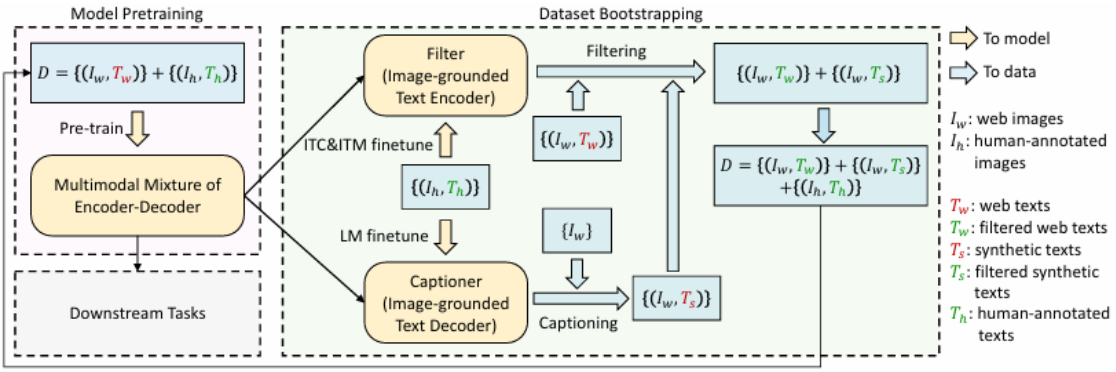


Figure 4-45 Bootstrapping Language-Image Pretraining flow

During inference, the image is first passed through the visual encoder to obtain a feature embedding, which is then used to generate a caption using the language model.

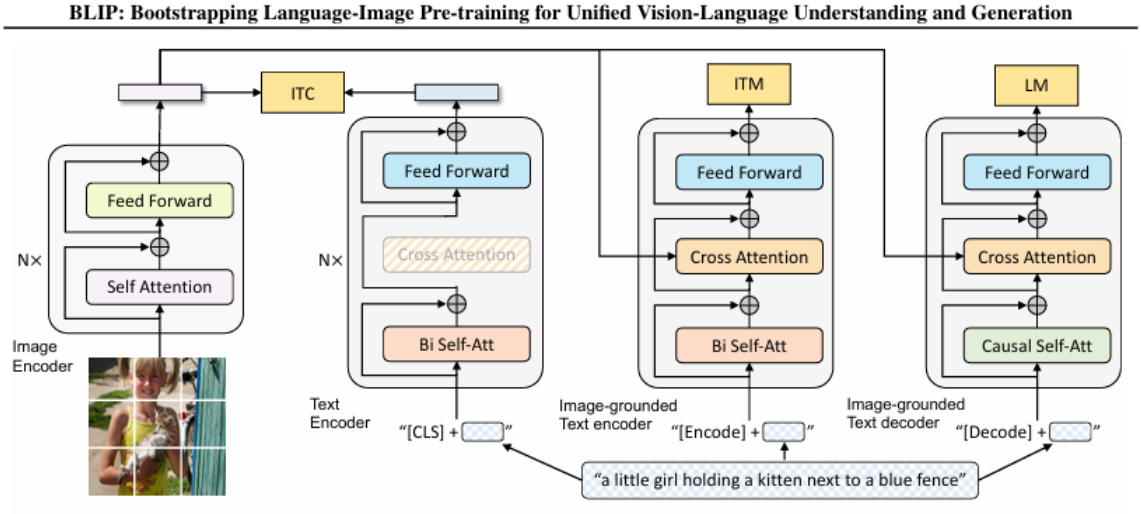


Figure 4-46 Bootstrapping Language-Image Pretraining architecture

Pre-training model architecture and objectives of BLIP (same parameters have the same color).

We propose a multimodal mixture of encoder-decoder, a unified vision-language model which can operate in one of the three functionalities:

1. Unimodal encoder is trained with an image-text contrastive (ITC) loss to align the vision and language representations.
2. Image-grounded text encoder uses additional cross-attention layers to model vision-language interactions. It is trained with an image-text matching (ITM) loss to distinguish between positive and negative image-text pairs.
3. Image-grounded text decoder replaces the bi-directional self-attention layers with causal self-attention layers and shares the same cross-attention layers and feed forward networks as the encoder.

The decoder is trained with a language modeling (LM) loss to generate captions given images.

Details Implementation

To enable image captioning functionality on a hardware-constrained device such as a Raspberry Pi, we deployed the BLIP model on a local server using FastAPI and made it accessible via the internet using ngrok. This approach allows the Raspberry Pi to send an image to the server and receive a generated caption in response, all without running the heavy model directly on the device.

Environment Setup and Project Initialization

We created a virtual environment named blip_env to run an image captioning project using a vision-language model such as BLIP. Within this environment, we installed all the required libraries to ensure proper isolation and compatibility.

The following Python libraries were installed:

```
!pip install numpy
!pip install pandas
!pip install pillow
!pip install tensorflow
!pip install torch
!pip install torchvision
!pip install transformers
```

Figure 4-47 Scene Description dependencies

These installations were performed inside the virtual environment to avoid conflicts and ensure smooth operation of the image captioning pipeline.

Image Captioning API using FastAPI and BLIP

caption_api.py 2

C: > Users > Taha > Desktop > blip_env > caption_api.py > generate_caption

```python
from fastapi import FastAPI, File, UploadFile
from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image
import io

app = FastAPI()

# Load the model and processor once
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")

@app.post("/caption/")
async def generate_caption(file: UploadFile = File(...)):
    contents = await file.read()
    image = Image.open(io.BytesIO(contents)).convert("RGB")

    inputs = processor(image, return_tensors="pt")
    out = model.generate(**inputs)
    caption = processor.decode(out[0], skip_special_tokens=True)

    return {"caption": caption}
```

Figure 4-48 Scene Description API

We built an image captioning API using **FastAPI** and the **Salesforce BLIP** (Bootstrapping Language-Image Pretraining) model.

The API utilizes the **blip-image-captioning-base** checkpoint to generate descriptive captions for input images. The model and processor are loaded once at startup using the **Transformers** library.

A **POST** endpoint /caption/ is defined, where users can upload an image file. The image is then read, processed, and passed through the model to generate a caption.

This allows users to send image files via an HTTP request and receive a text description automatically generated by the BLIP model.

Running the API

We navigated to the project directory and started the FastAPI application using Uvicorn which is start the image captioning API on http://0.0.0.0:8000/.

Once the message Application startup complete appeared, it confirmed the server was running and ready to receive image uploads for caption generation.

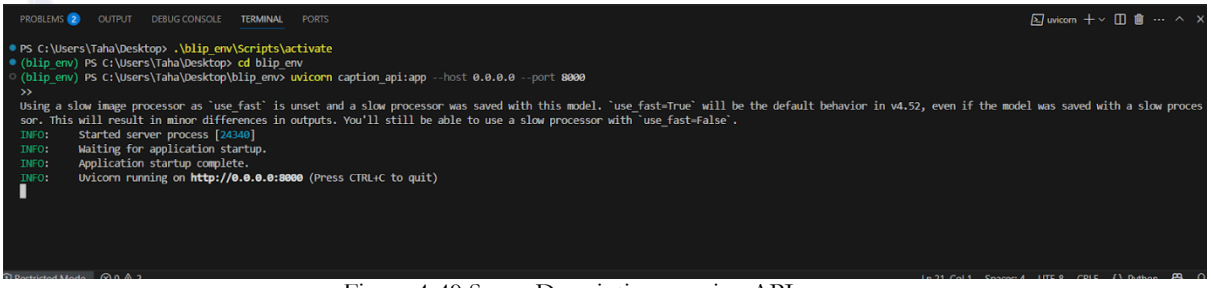


Figure 4-49 Scene Description running API

ngrok Session Overview Explanation

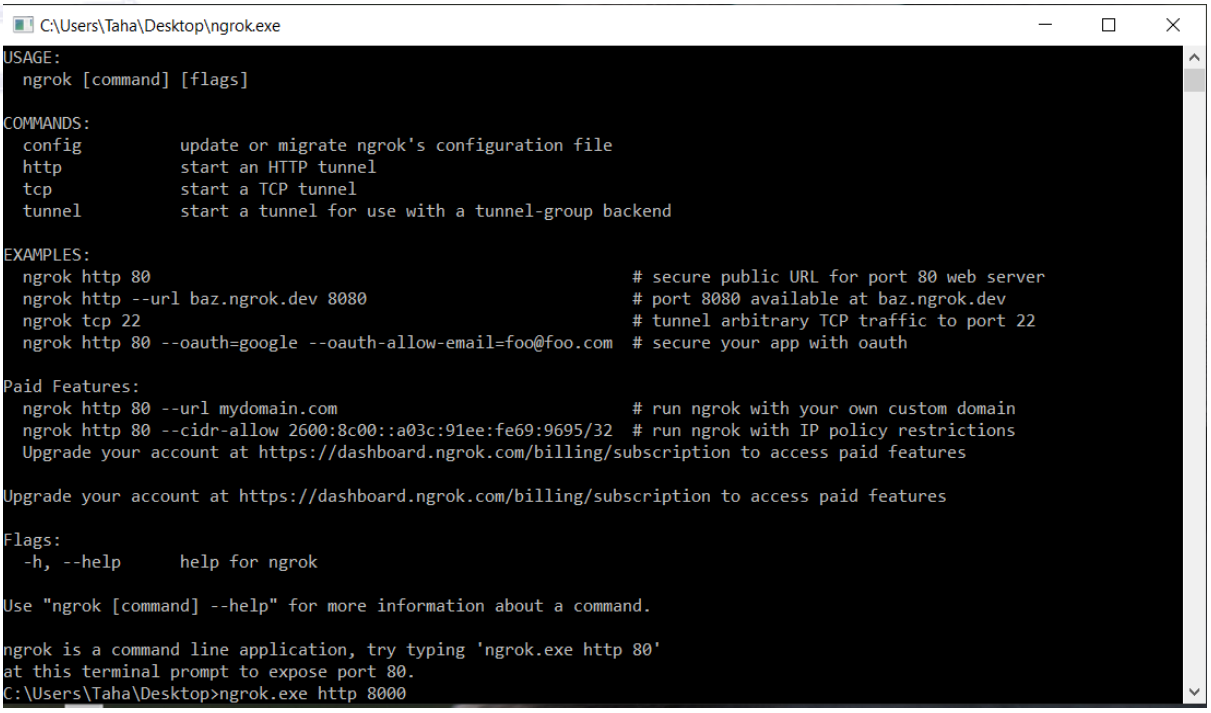


Figure 4-50 connect ngrok with Scene Description API

The Ngrok session displays key information:

- **Account:** Shows email and plan type.
- **Version & Region:** Current Ngrok version and server region.
- **Web Interface:** Local dashboard URL (e.g., 127.0.0.1:4040).
- **Forwarding:** Public URL forwarding to your local server (e.g., localhost:8000). Metrics: Total/open connections and response times.

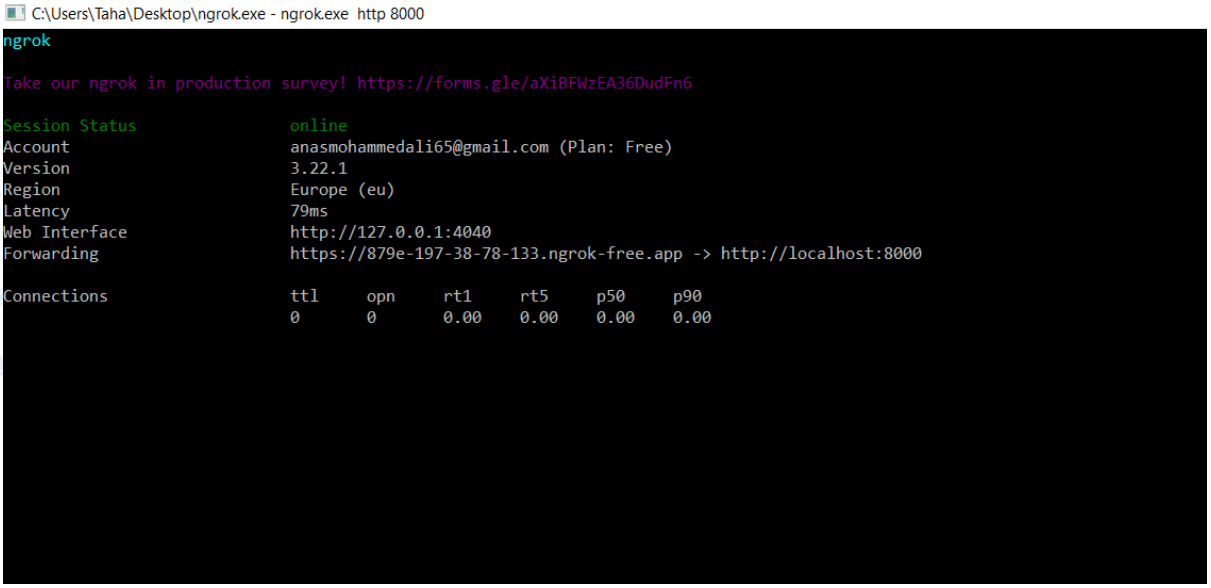


Figure 4-51 Scene Description running server

Image Captioning Client Script

C: > Users > Taha > Desktop > blip_env > test_remotly.py > ...

```python
import requests
import cv2

# Define the server URL and image path
url = "https://879e-197-38-78-133.ngrok-free.app/caption/"
image_path = "1.jpg"

# Read the image using OpenCV
image = cv2.imread(image_path)

# Send the image to the server
with open(image_path, "rb") as img:
    files = {"file": img}
    response = requests.post(url, files=files)

# Extract the caption
caption = response.json().get("caption", "No caption received")
print("Caption:", caption)

# Add caption text on the image
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(image, caption, (10, 30), font, 0.8, (0, 255, 0), 2, cv2.LINE_AA)

# Show the image in a window
cv2.imshow("Image with Caption", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 4-52 request to Scene Description server

This script allows you to send an image to a FastAPI-based BLIP captioning server and display the image with the generated caption overlaid by:

- Set the server URL and load the image using OpenCV.
- Send a POST request to the API using requests, attaching the image as binary data.
- Receive the caption from the server's JSON response.
- Draw the caption on the image using cv2.putText.
- Display the final image with caption in an OpenCV window.

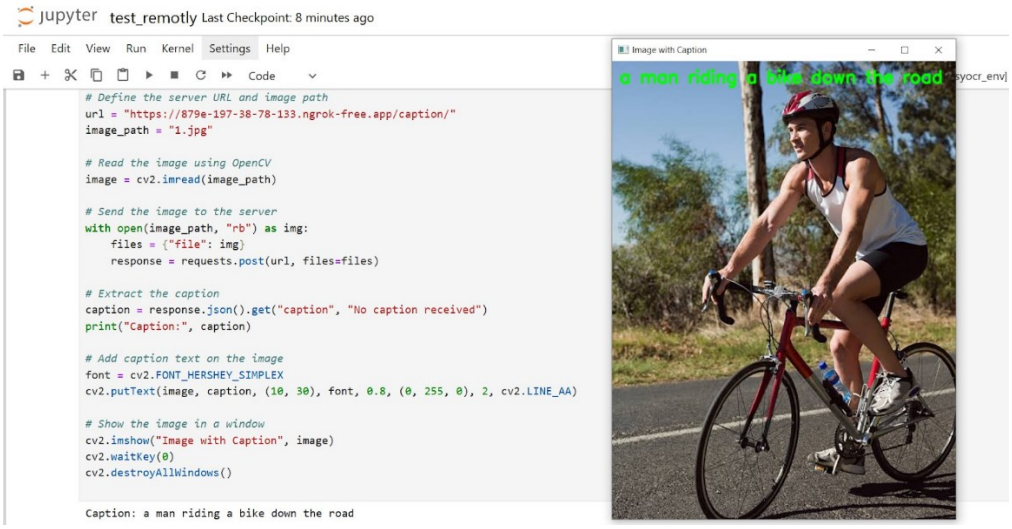


Figure 4-53 test-zero Scene Description server

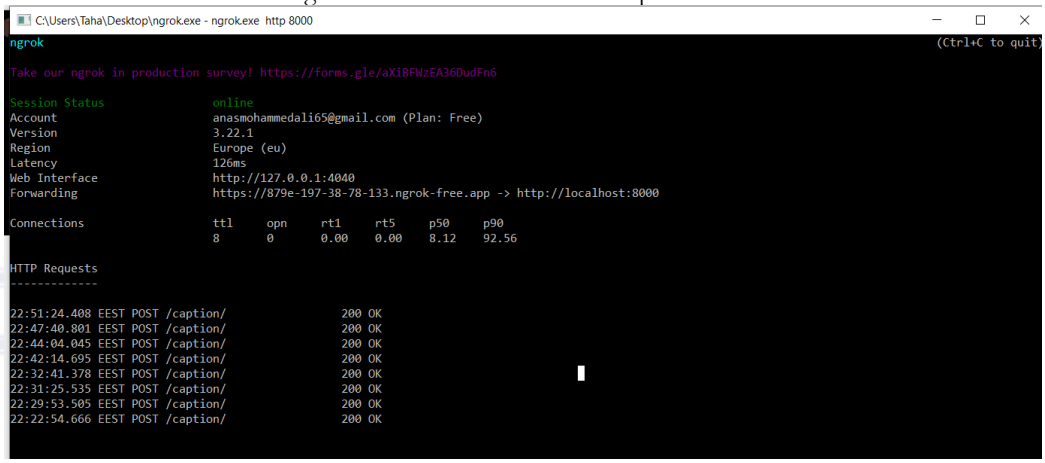


Figure 4-54 response of Scene Description server

Figure 4-55 test-one Scene Description server

4.2.4 TR Flow:

Since OCR is computationally heavy for the Raspberry Pi, we built a lightweight API on a more powerful machine to handle the OCR processing. We exposed the API using Ngrok, allowing the Raspberry Pi to send images remotely via HTTP requests. The server processes the image using PaddleOCR and returns the extracted text as a response.

Model

The Optical Character Recognition (OCR) systems have been widely used in various of application scenarios, such as office automation (OA) systems, factory automations, online educations, map productions etc. However, OCR is still a challenging task due to the various of text appearances and the demand of computational efficiency. we use a practical ultra lightweight OCR system, i.e., PPOCR. The overall model size of the PP-OCR is only 3.5M for recognizing 6622 Chinese characters and 2.8M for recognizing 63 alphanumeric symbols, respectively. it introduces a bag of strategies to either enhance the model ability or reduce the model size. The corresponding ablation experiments with the real data are also provided. Meanwhile, several pre-trained models for the Chinese and English recognition are released, including a text detector (97K images are used), a direction classifier (600K images are used) as well as a text recognizer (17.9M images are used). Besides, the proposed PP-OCR are also verified in several other language recognition tasks, including French, Korean, Japanese and German.

Architecture Overview

considering the cost. In particular, the OCR system need to be run on embedded devices in many scenarios, such as cell phones, which makes it necessary to consider the model size. Trade off model size and performance is difficult but of great value. it proposes a practical ultra lightweight OCR system, named as PP-OCR, which consists of three parts, text detection, detected boxes rectification and text recognition as shown in Figure.

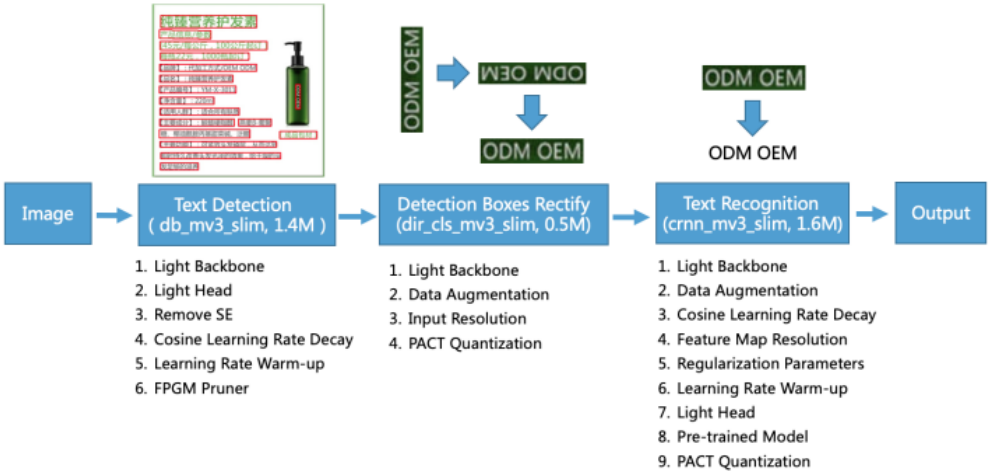


Figure 4-56 Text Reading flow

- **Text Detection**

The purpose of text detection is to locate the text area in the image. In PP-OCR, it uses Differentiable Binarization (DB) (Liao et al. 2020) as text detector which is based on a simple segmentation network. The simple postprocessing of DB makes it very efficient. To further improve its effectiveness and efficiency, the following six strategies are used: light backbone, light head, remove SE module, cosine learning rate decay, learning rate warm-up, and FPGM pruner. Finally, the model size of the text detector is reduced to 1.4M.

- **Detection Boxes Rectify**

  Before recognizing the detected text, the text box needs to be transformed into a horizontal rectangle box for subsequent text recognition, which is easy to be achieved by geometric transformation as the detection frame is composed of four points. However, the rectified boxes may be reversed. Thus, a classifier is needed to determine the text direction. If a box is determined reversed, further flipping is required. Training a text direction classifier is a simple image classification task. the following four strategies to enhance the model ability and reduce the model size: light backbone, data augmentation, input resolution and PACT quantization. Finally, the model size of the text direction classifier is 500KB.

- **Text Recognition**

  In PP-OCR, it is using CRNN (Shi, Bai, and Yao 2016) as text recognizer, which is widely used and practical for text recognition. CRNN integrates feature extraction and sequence modeling. It adopts the Connectionist Temporal Classification (CTC) loss to avoid the inconsistency between prediction and label. To enhance the model ability and reduce the model size of a text recognizer, the following nine strategies are used: light backbone, data augmentation, cosine learning rate decay, feature map resolution, regularization parameters, learning rate warm-up, light head, pre-trained model and PACT quantization. Finally, the model size of the text recognizer is only 1.6M for Chinese and English recognition and 900KB for alphanumeric symbols recognition.

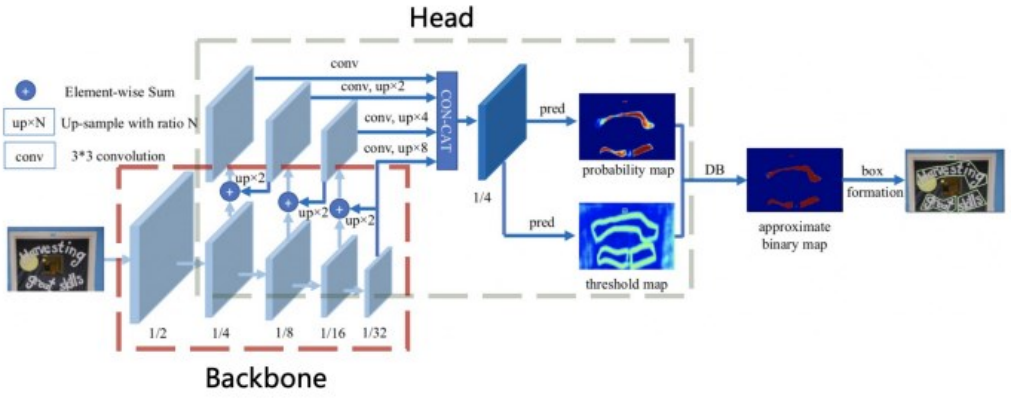


Figure 4-57 Paddle2Paddle architecture

Details Implementation

To enable text reading functionality on a hardware-constrained device such as a Raspberry Pi, we deployed the PP-OCR model on a local server using FastAPI and made it accessible via the internet using ngrok. This approach allows the Raspberry Pi to send an image to the server and receive a recognized text in response, all without running the heavy model directly on the device.

Environment Setup and Project Initialization

We created a virtual environment named paddleocr_env to run a text reading project using a PP-OCR model. Within this environment, we installed all the required libraries to ensure proper isolation and compatibility.

The following Python libraries were installed:

```
pip install paddlepaddle -f https://www.paddlepaddle.org.cn/whl/simple
pip install paddleocr
pip install fastapi
pip install uvicorn
pip install opencv-python
pip install pillow
pip install numpy
```

Figure 4-58 Text Reading dependencies

These installations were performed inside the virtual environment to avoid conflicts and ensure smooth operation of the text reading pipeline.

Text Reading API using FastAPI and Paddle-OCR

TextRead_test_fastapi.py 3

C: > Users > Taha > Desktop > paddleocr_env > TextRead_test_fastapi.py > ...

```python
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import JSONResponse
import cv2
import numpy as np
import os
import logging
import warnings
from paddleocr import PaddleOCR

# Suppress warnings and logs
warnings.filterwarnings("ignore", category=UserWarning)
os.environ["PADDLE_NO_GPU"] = "0"
os.environ["FLAGS_allocator_strategy"] = "auto_growth"
logging.basicConfig(level=logging.ERROR)

# Initialize FastAPI app and OCR model
app = FastAPI()
ocr = PaddleOCR(use_angle_cls=True, lang='en', show_log=False)

@app.post("/paddleocr/")
async def paddleocr(file: UploadFile = File(...)):
    try:
        contents = await file.read()
        np_arr = np.frombuffer(contents, np.uint8)
        image = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)

        result = ocr.ocr(image, cls=True)

        if not result or result[0] is None:
            return JSONResponse(content={"message": "No text detected."}, status_code=200)

        detected_texts = [line[1][0] for line in result[0]]
        return {"detected_texts": detected_texts}

    except Exception as e:
        return JSONResponse(content={"error": str(e)}, status_code=500)
```

Figure 4-59 Text Reading API

We built a text reading API using FastAPI and the PaddleOCR (Optical Character Recognition) model. The API is designed to extract textual content from uploaded images by users. The PaddleOCR model is initialized at the beginning, with specific environment settings to optimize performance and suppress unnecessary warnings. The system uses OpenCV to decode the uploaded image and convert it into a suitable format for OCR processing.

A POST endpoint /paddleocr/ is defined, where users can send an image file. Upon receiving the image, the API reads and decodes it, then applies the OCR model to extract any detected text. If no text is found, a proper response is returned indicating that no text was detected. Otherwise, the detected words are extracted and sent back as a JSON response. This setup allows for a fast and efficient way to process image files and retrieve any embedded text using PaddleOCR through a simple HTTP request.

Running the API

We navigated to the project directory and started the FastAPI application using Uvicorn, which starts the text reading API on http://0.0.0.0:8000/. Once the message "Application startup complete" appeared, it confirmed the server was running and ready to receive image uploads for text extraction using PaddleOCR.

Figure 4-60 Text Reading running API

ngrok Session Overview Explanation

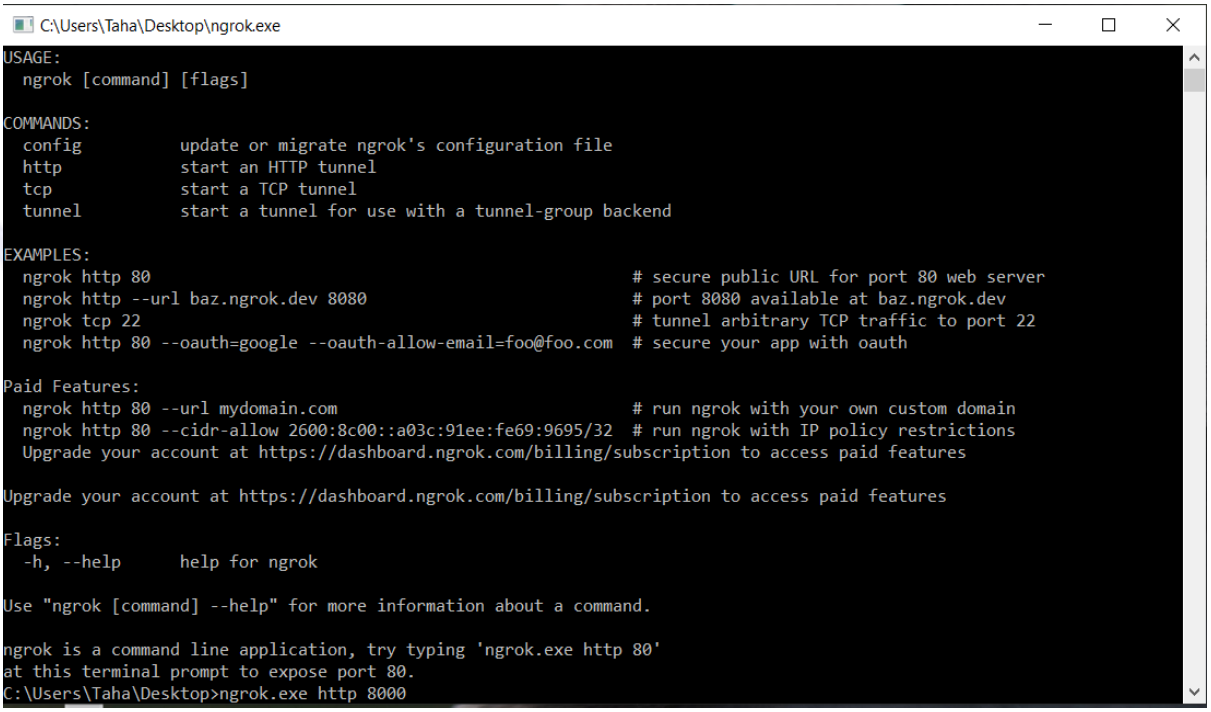


Figure 4-61 connect ngrok with Text Reading API

The Ngrok session displays key information:

- **Account:** Shows email and plan type.
- **Version & Region:** Current Ngrok version and server region.
- **Web Interface:** Local dashboard URL (e.g., 127.0.0.1:4040).
- **Forwarding:** Public URL forwarding to your local server (e.g., localhost:8000).
- **Metrics:** Total/open connections and response times.

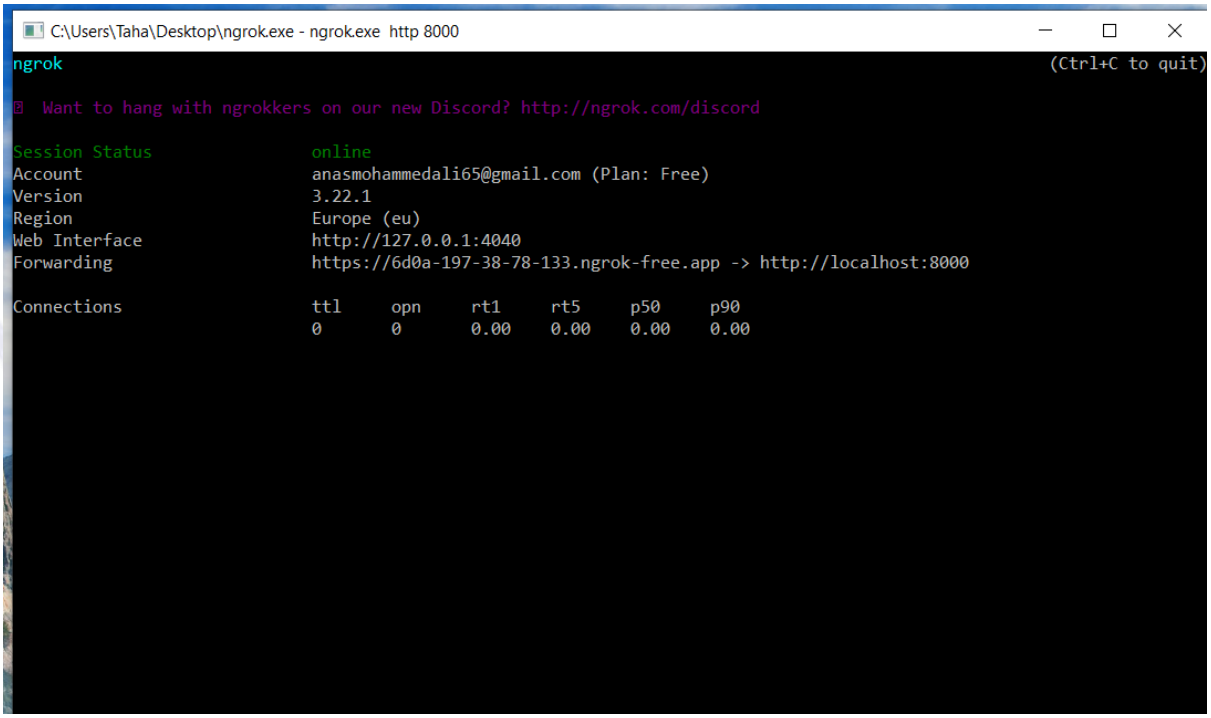


Figure 4-62 Text Reading running server

Text Reading Client Script

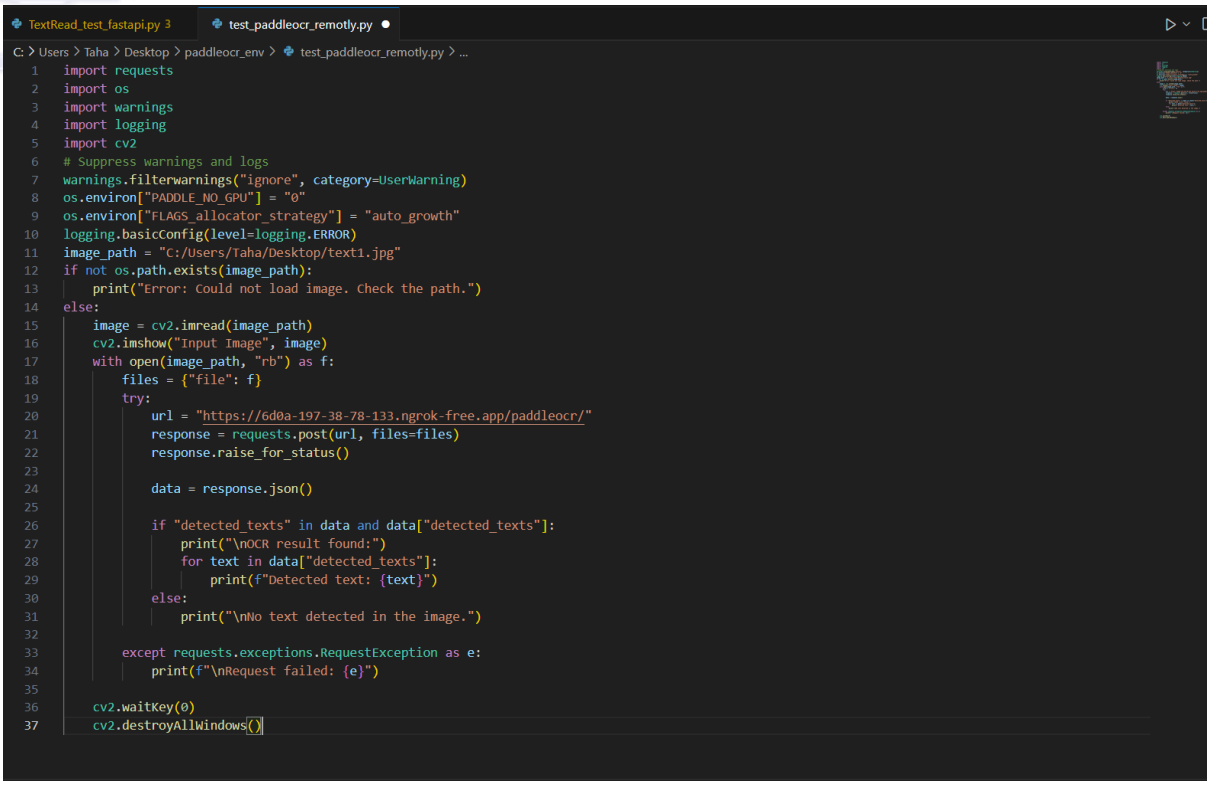


Figure 4-63 request to Text Reading server

This Python script sends an image to a FastAPI-based OCR server and displays both the image and any detected text results. It begins by suppressing unnecessary warnings and logs, sets environment variables for optimal memory usage with PaddleOCR, and verifies that the specified image path is valid. If the image is found, it is loaded using OpenCV and displayed in a GUI window. The script then sends the image as binary data in a POST request to a remote OCR API endpoint hosted via FastAPI. The server analyzes the image and returns any recognized text in a JSON response. The script checks for detected text and prints it to the terminal. Finally, OpenCV functions are used to keep the image window open until a key is pressed, allowing the user to view the image before the window is closed.

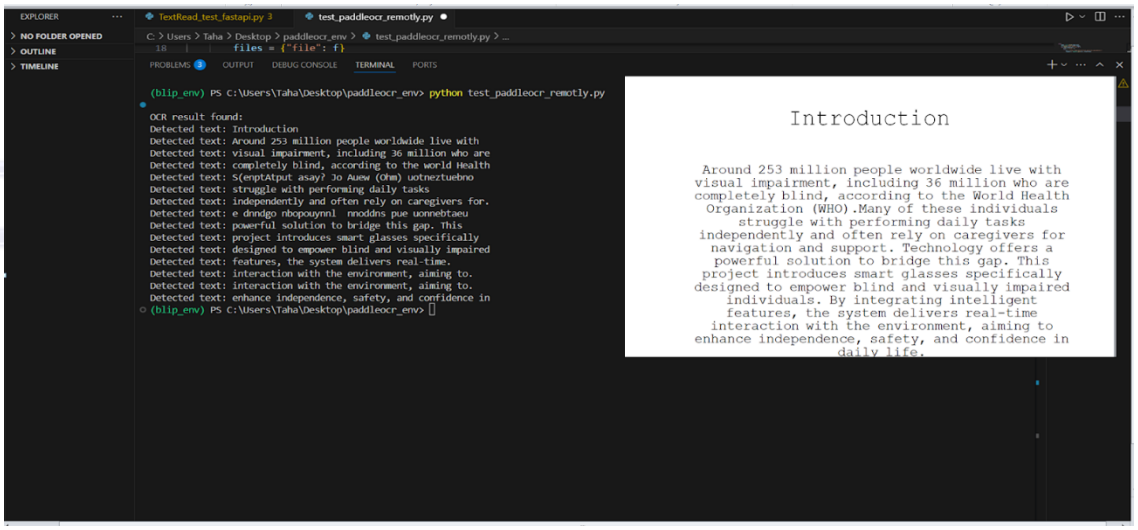


Figure 4-64 test-zero Text Reading server

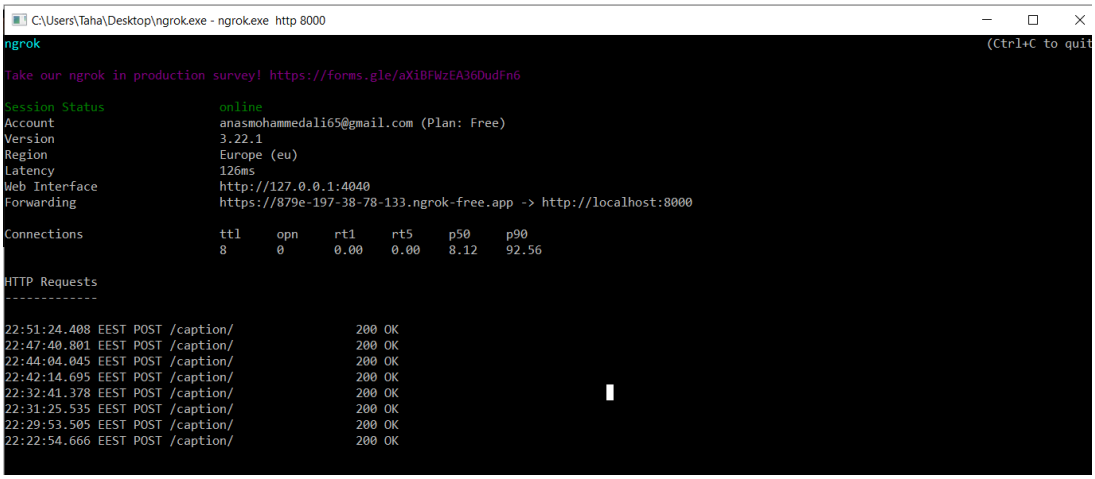


Figure 4-65 response of Text Reading server

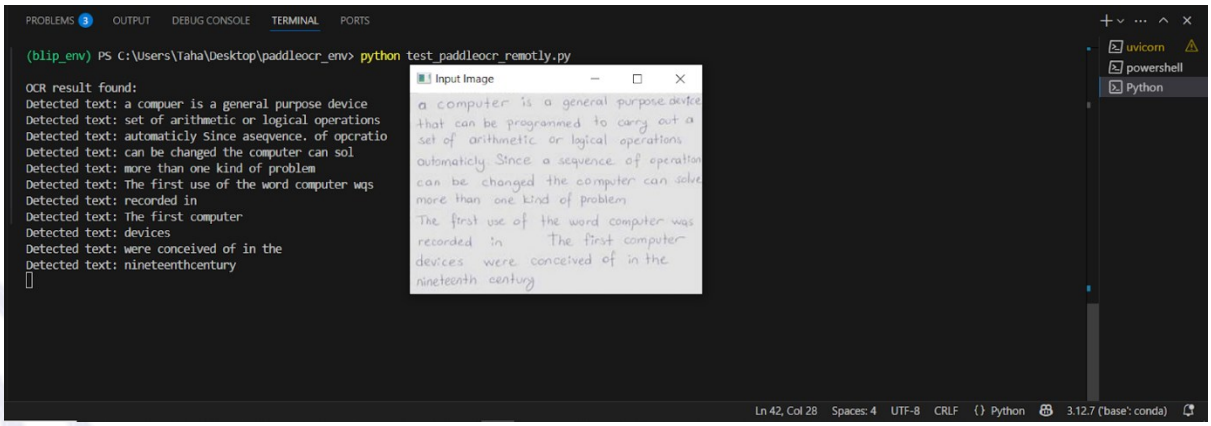


Figure 4-66 test-one Text Reading server

4.3 Training Challenges

Due to the large size of the training images, it wasn't easy to perform the training phase with a normal computer. Therefore, Google Colab, Google Drive, and Kaggle were used to overcome this problem; however, we faced several issues with them.

1. We encountered a problem while reading and processing the data on both Kaggle and Colab due to limited available memory. As a result, we had to reduce the size of the dataset used for training to avoid memory errors. This solution helped us continue running the models without unexpected interruptions during execution.

2. Google Colab has a daily GPU usage limit of 12 hours per day; if it is exceeded, the run will stop. It was solved by saving checkpoints and continuing from the last saved checkpoint.

4.4 Description of Any New Technologies Used in the Implementation

4.4.1 Used Environment

In this section, we will mention the used environments in our project with their specifications.

- **Google Colab:**

  Google Colab has several drawbacks, however, especially when it comes to limitations on the free plan. Colab's free GPU instances (most frequently K80 GPUs released in 2014) are often underpowered. Connectivity can be unreliable as instances will disconnect frequently or can be preempted by other users during inactivity. And instances often do not come with enough RAM, particularly when working with larger datasets. Our dataset was large, so we needed a faster GPU and Large storage.

- **Kaggle Notebooks:**

  are cloud-based Jupyter notebooks provided by Kaggle, a platform for data science competitions and collaboration. They allow users to write and execute Python code directly in the browser without any local setup. Kaggle provides free access to GPUs, TPUs, and a variety of pre-installed data science libraries, making it ideal for machine learning and deep learning projects. Users can import datasets directly from Kaggle's large public dataset repository. Notebooks can be easily shared, forked (copied), and collaborated on with others in the community. The key advantages include ease of use, zero setup, free compute resources, and a strong community for learning and support and powerful of GPUs Which allowed us to use it.

- **PyCharm:**

  PyCharm is an IDE used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains.

- **VS Code:**

  Visual Studio Code (VS Code) is a source code editor used in computer programming for various languages, including Python. It is developed by Microsoft.

- **Jupyter:**

  Jupyter is an open-source web application used in data science and machine learning for creating and sharing documents that contain live code, equations, visualizations, and narrative text. It is developed by Project Jupyter.

4.4.2 Used Technologies

- **TensorFlow:**

  An open-source machine learning framework that enables building and training models for a wide range of tasks, from research experiments to production-level applications.

- **TensorFlow Lite:**

  A lightweight version of TensorFlow designed for deploying machine learning models on mobile and embedded devices with low latency and minimal resource usage.

- **PyTorch:**

  An open-source machine learning framework that accelerates the path from research prototyping to production deployment.

- **RoboFlow:**

  A platform that streamlines the process of collecting, annotating, training, and deploying computer vision models, making it easier to build and manage vision AI projects.

- **REST API:**

  A software architectural style that allows interaction between client and server over HTTP, enabling access to and manipulation of web resources using standard HTTP methods like GET, POST, PUT, and DELETE.

Chapter 5
Hardware

5.1 introduction

Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and developing countries. The original model became far more popular than anticipated, selling outside its target market for uses such as robotics. It does not include peripherals such as keyboards and mice or cases. However, some accessories have been included in several official and unofficial bundles.

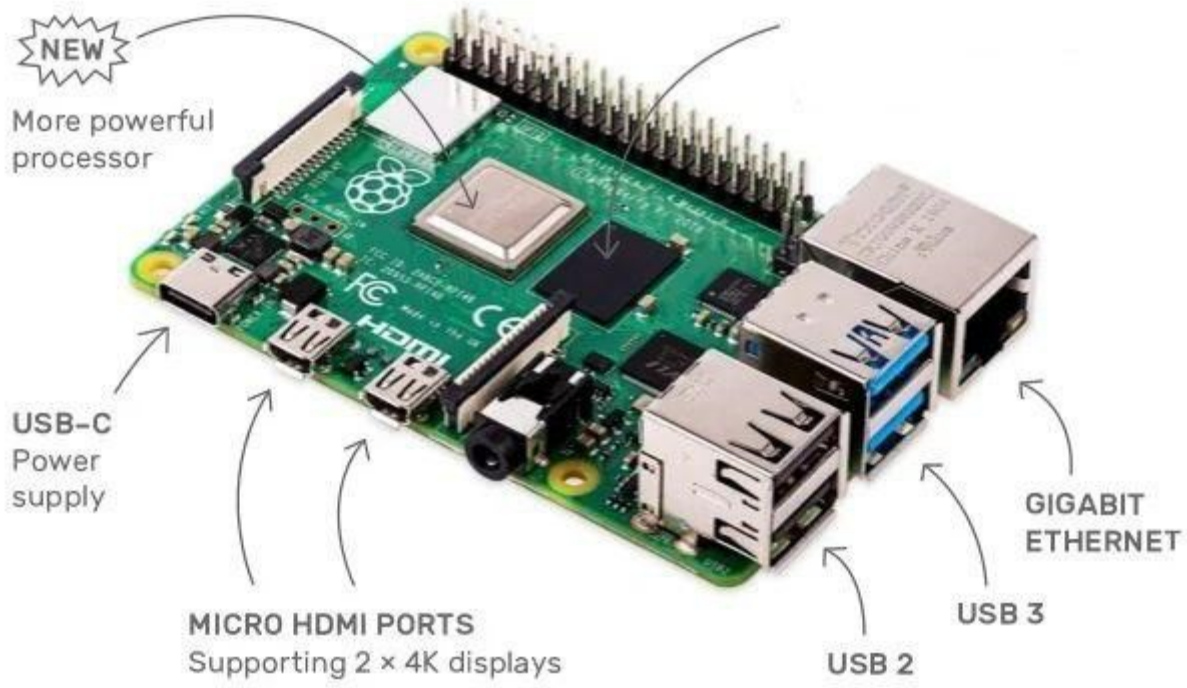


Figure 5-1 Raspberry Pi 4 Model B

Components:

- Processor: Broadcom BCM2711 64-bit SoC 1.5 GHz.
- 8GB LPDDR4-3200 SDRAM.
- Raspberry Pi standard 40 pin GPIO header
- Connectivity: Fast data transfer with USB 3.0, 2 USB 2.0 ports and Gigabit
- Ethernet.
- GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0.
- Supports two 4k HDMI displays.
- Storage: Micro-SD card slot for loading operating system and data storage.
- 2-lane MIPI DSI display port.
- 2-lane MIPI CSI camera port (for the 8 MP camera).
- 4-pole stereo audio and composite video port (3.5 mm Headphone Jack).
- 5V DC via USB-C connector (minimum 3A*).
- 5V DC via GPIO header (minimum 3A*).

5.2 Specifications of components

5.2.1 Processor

Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz.
BCM2711: this is the Broadcom chip (Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz) used in the Raspberry Pi 4 Model B, the Raspberry Pi 400, and the Raspberry Pi Compute Module 3. The architecture of the BCM2711 is a considerable upgrade on that used by the SoCs in earlier Raspberry Pi models. It continues the quad-core CPU design of the BCM2837 but uses the more powerful ARM A72 core. It has a greatly improved GPU feature set with much faster input/output, due to the

incorporation of a PCIe link that connects the USB 2 and USB 3 ports, and a natively attached Ethernet controller. It is also capable of addressing more memory than the SoCs used before. The ARM cores can run at up to 1.5 GHz, making the Raspberry Pi 4 about 50% faster than the Raspberry Pi 3B+. The new VideoCore VI 3D unit now runs at up to 500 MHz The ARM cores are 64-bit, and while the VideoCore is 32-bit, there is a new Memory Management Unit, which means it can access more memory than previous versions. The BCM2711 chip continues to use the heat-spreading technology that started with the BCM2837B0, which provides better thermal management.

5.2.2 Memory

Accesses up to 8GB LPDDR4-2400 SDRAM (depending on model)

5.2.3 Caches

32kB data + 48kB instruction L1 cache per core. 1MB L2 cache.

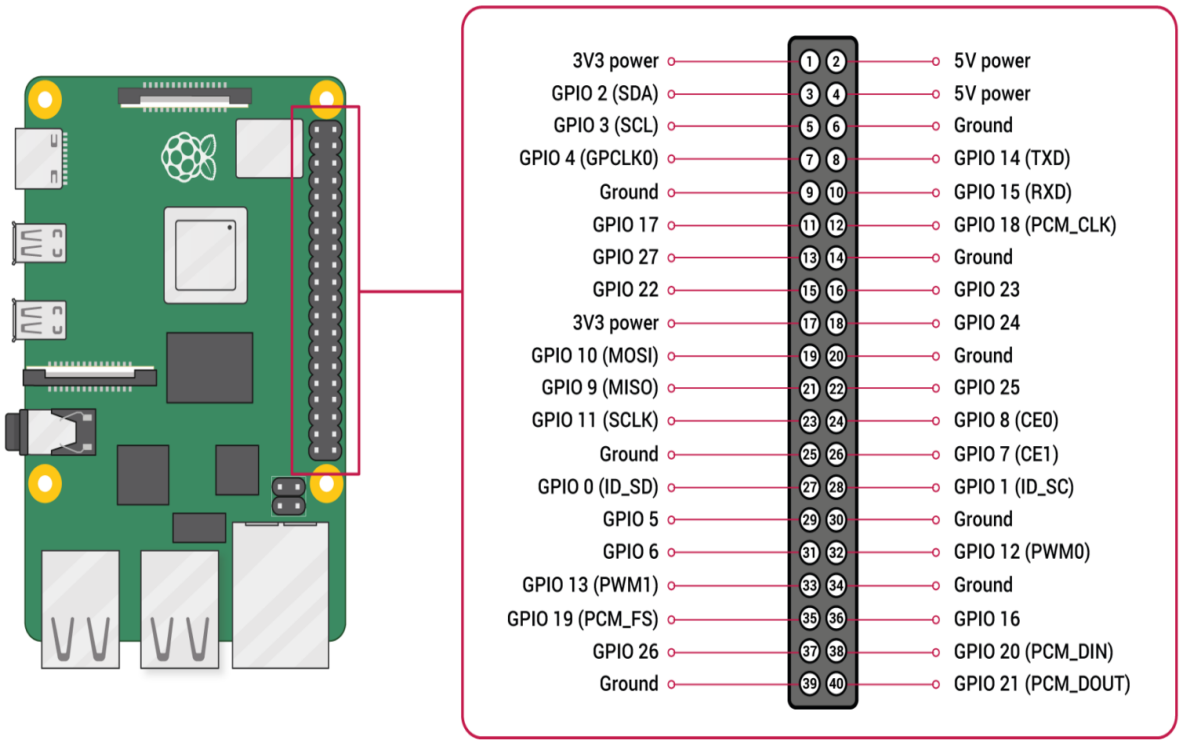
5.2.4 Multimedia:

H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics I/O: PCIe bus, onboard Ethernet port, 2 × DSI ports (only one exposed on Raspberry Pi 4B), 2 × CSI ports (only one exposed on Raspberry Pi 4B), up to 6 × I2C, up to 6 × UART (mixed with I2C), up to 6 × SPI (only five exposed on Raspberry Pi 4B), dual HDMI video output, composite video output.

5.2.5 Raspberry pi (GPIO):

A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Raspberry Pi Zero, Raspberry Pi Zero W and Raspberry Pi Zero 2 W). Prior to the Raspberry Pi 1

Model B+ (2014), boards comprised a shorter 26-pin header. The GPIO header on all boards (including the Raspberry Pi 400) have a 0.1" (2.54mm) pin pitch.



Any of the GPIO pins can be designated (in software) as an input or output pin and used for a wide range of purposes.

Figure 5-2 GPIO header

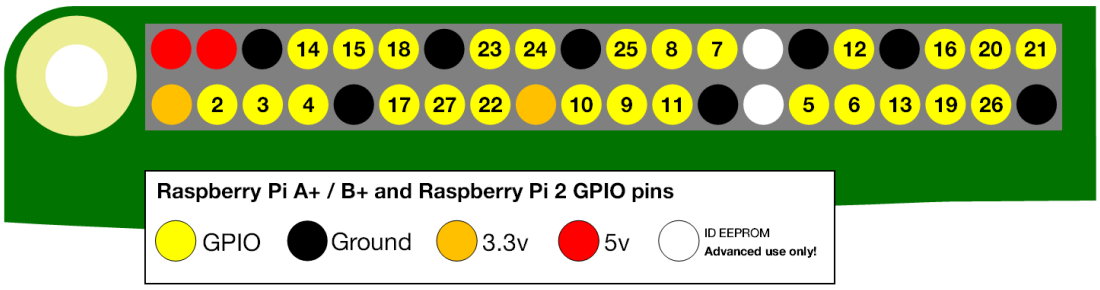


Figure 5-3 GPIO pins

- **Voltages**

  Two 5V pins and two 3.3V pins are present on the board, as well as several ground pins (0V), which are unconfigurable. The remaining pins are all general purpose 3.3V pins, meaning outputs are set to 3.3V and inputs are 3.3V-tolerant.

- **Outputs**
  A GPIO pin designated as an output pin can be set to high (3.3V) or low (0V).

- **Inputs**
  A GPIO pin designated as an input pin can be read as high (3.3V) or low (0V). This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins, this can be configured in software. (4.3.4 More.)

  As well as simple input and output devices, the GPIO pins can be used with a variety of alternative functions, some are available on all pins, others on specific pins.

  - **PWM (pulse-width modulation)** o Software PWM available on all pins o Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
  - **SPI o SPI0:** MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7) o SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
  - **I2C o Data:** (GPIO2); Clock (GPIO3) o EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)

5.2.6 Raspberry Pi Camera Module 3

Raspberry Pi Camera Module 3 is a compact camera from Raspberry Pi. It offers an IMX708 12-megapixel sensor with HDR, and features phase detection autofocus. Camera Module 3 is available in standard and wide-angle variants, both of which are available with and without an infrared cut filter.

Figure 5-4 Raspberry Pi Camera Module 3

- Back-illuminated, stacked CMOS 12-megapixel Sony IMX708 image sensor
- High signal-to-noise ratio (SNR)
- Built-in 2D Dynamic Defect Pixel Correction (DPC)
- Phase Detection Autofocus (PDAF) for rapid autofocus
- QBC Re-mosaic function
- HDR mode (up to 3-megapixel output)
- CSI-2 serial data output
- 2-wire serial communication (supports I2C fast mode and fast mode plus)
- 2-wire serial control of focus mechanism
- Resolution: 11.9 megapixels
- Sensor size: 7.4mm sensor diagonal
- Pixel size: 1.4µm × 1.4µm
- Horizontal/vertical: 4608 × 2592 pixels Diagonal field of view: 75 degrees (Camera Module 3,

Camera Module 3 NoIR), 120 degrees (Camera Module 3 Wide, Camera Module 3 NoIR Wide)

- Common video modes: 1080p50, 720p100, 480p120
- Output: RAW10
- IR cut filter: Integrated in standard variants; not present in NoIR variants
- Dimensions: 25 × 24 × 11.5mm (12.4mm height for Wide variants)
- Ribbon cable length: 200mm
- Cable connector: 15 × 1mm FPC

5.2.7 Joystick

joystick very similar to the 'analog' joysticks on PS2 (PlayStation 2) controllers. Directional movements are simply two potentiometers – one for each axis. Pots are ~10k each. This joystick also has a select button that is actuated when the joystick is pressed down. This is a Joystick module that has X, Y analog outputs and has a switch that detects button press. The joystick is made up of two passive potentio-meters (variable resistors) and a push button and therefore can be used at any standard voltage levels. Lots of robotic projects need a joystick. This module offers an affordable solution to that. The Joystick module is like analog joysticks found in gamepads. It is made by mounting two potentiometers at a 90 degrees angle. The potentiometers are connected to a short stick centered by springs. This module produces an output of around 2.5V from X and Y when it is in resting position. Moving the joystick will cause the output to vary from 0v to 5V depending on its direction. If you connect this module to a microcontroller, you can expect to read a value of around 512 in its resting position (expect small variations due to tiny imprecisions of the springs and mechanism) When you move the joystick you should see the values change from 0 to 1023 depending on its position.

Figure 5-5 Joystick

Joystick that can be interfaced with Arduino, Raspberry Pi, and other Microcontrollers and Embedded Boards Joystick for custom designed/experimental RC planes, RC Cars. Joystick for CNC Machines or other industrial controls electronic games pads like PlayStation. Using the board to connect the joystick is reduced to the following pins:

- VCC – supply voltage (5 V).
- GND– ground of the system.
- VRx – the analog output for the vertical axis (in the middle position, the value is equal to half of the supply voltage VCC/2)
- VRy– Analog output for the horizontal axis (in the middle position, the value is equal to half of the supply voltage VCC/2).
- SW– Digital signal button, in case of activation, it reaches the value of the GND, deactivated – is in the floating status – so you need to add a pulling-up resistor to power.

5.2.8 Power Bank

The Raspberry Pi needs an input voltage of around 5 Volts, which is supplied through the USB type C port. In fact, the input voltage should be a little bit higher than 5 Volts.

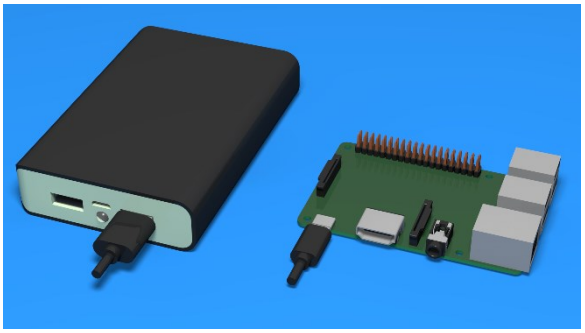


Figure 5-6 Power Bank

5.3 hardware setup

Step 1: Download Raspberry Pi Imager

- from the official Raspberry Pi site and download the **Raspberry Pi Imager**

Step 2: Flash the OS to the SD Card.

Step 3: Boot the Raspberry Pi.

Step 4: First-Time Setup.

- Open a terminal and run ‘’ sudo apt update && sudo apt upgrade -y ‘‘

Step 5: Install All Dependencies

Chapter 6
User Manual

6.1 Introduction

This chapter provides a comprehensive guide to using the smart glasses system, which is designed to assist visually impaired users by enabling hands-free operation through real-time AI processing and audio feedback. It covers system setup, joystick controls, mode functionalities, audio output, and troubleshooting to ensure smooth operation and accessibility.

6.2 System Boot and Initialization

To begin using the system, users must first power on the Raspberry Pi via an external battery or switch. Upon successful boot, an audio message confirms that the system is ready for interaction. It is important to ensure that the camera is facing forward, and that the joystick module is securely attached. The bone-conduction speaker should also be checked to confirm it is placed correctly on the user's cheekbone to provide effective audio transmission. Once all components are verified, the user can proceed to interact with the device using the joystick.

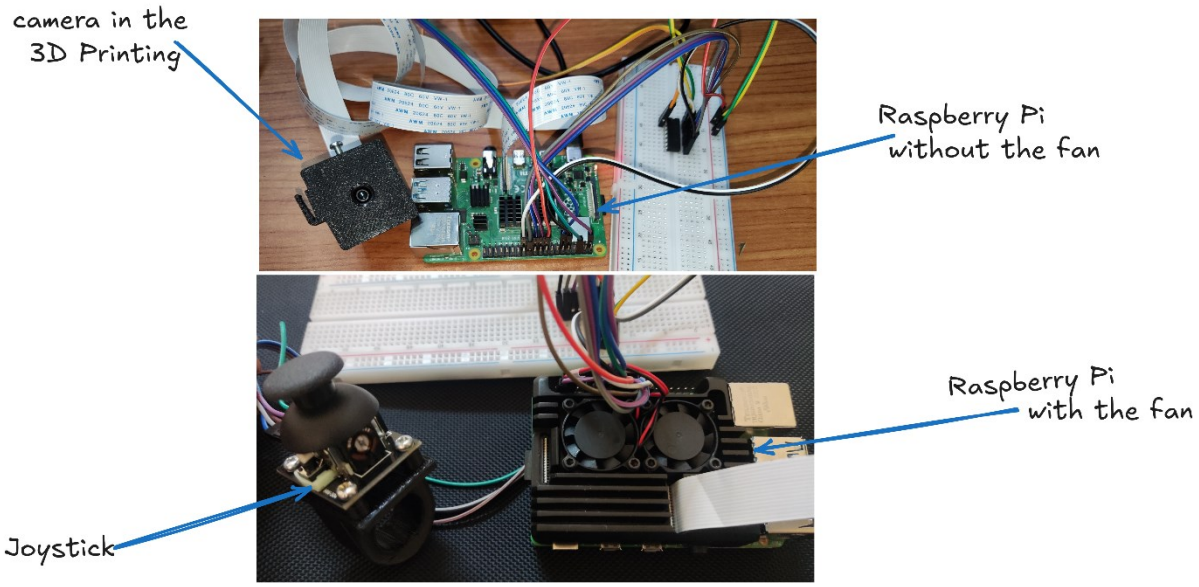


Figure 6-1 System components

6.3 Joystick Control Guide

The system is controlled entirely through the joystick, offering a hands-free and intuitive interface. A press in the left direction activates the currency recognition mode, which identifies and announces the denomination of any Egyptian banknote captured by the camera. A press to the right triggers the text reading mode, where the system detects printed or handwritten text in the environment and reads it aloud. Pressing the joystick up initiates scene description mode, which captures the surroundings and describes them through AI-generated captions. Pressing the joystick down enables the face and emotion recognition function, where the system identifies people from a pre-trained database and interprets their emotional expression.

In addition to directional controls, other joystick actions manage system control. A long press exits the currently active mode and returns the system to standby, ensuring that the user can switch tasks easily. A double press shuts down the main program completely, while a single press (the center click) captures an image in the currently active mode for immediate processing. These controls provide a responsive and fluid interaction model suited for real-time applications.

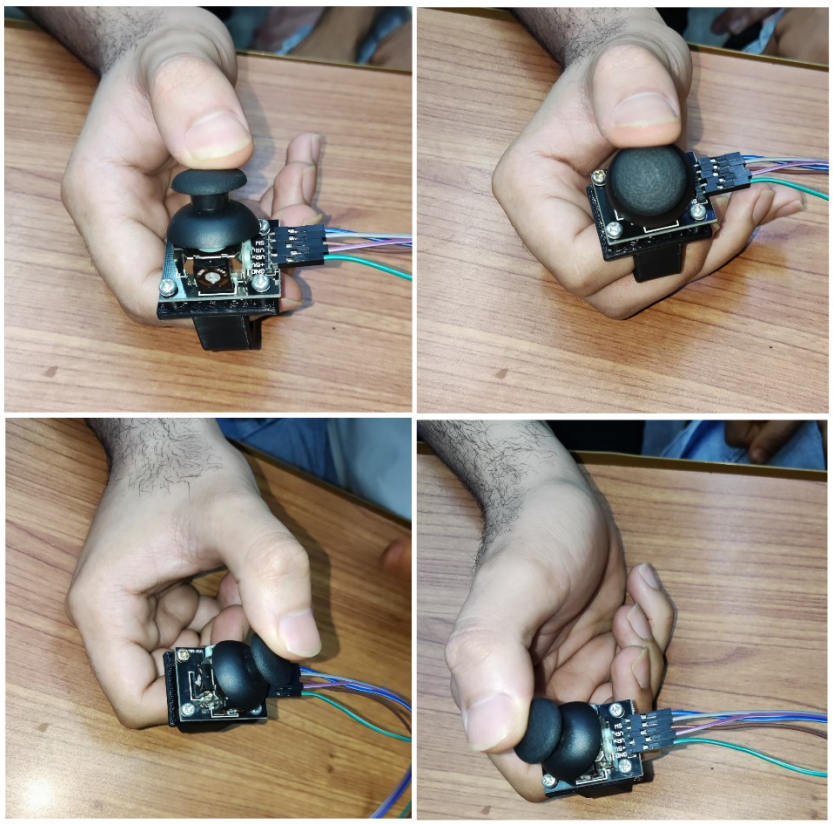


Figure 6-2 Joystick control guide

6.4 Mode Descriptions

6.4.1 Currency Detection and Identification Mode

To use this mode, the user should press the joystick to the left. The system will activate the currency detection pipeline, first identifying the presence of a currency note and then classifying its denomination using a trained model. Once detected, an audio message is played to inform the user of the note's value, such as “This is a fifty-pound note.”

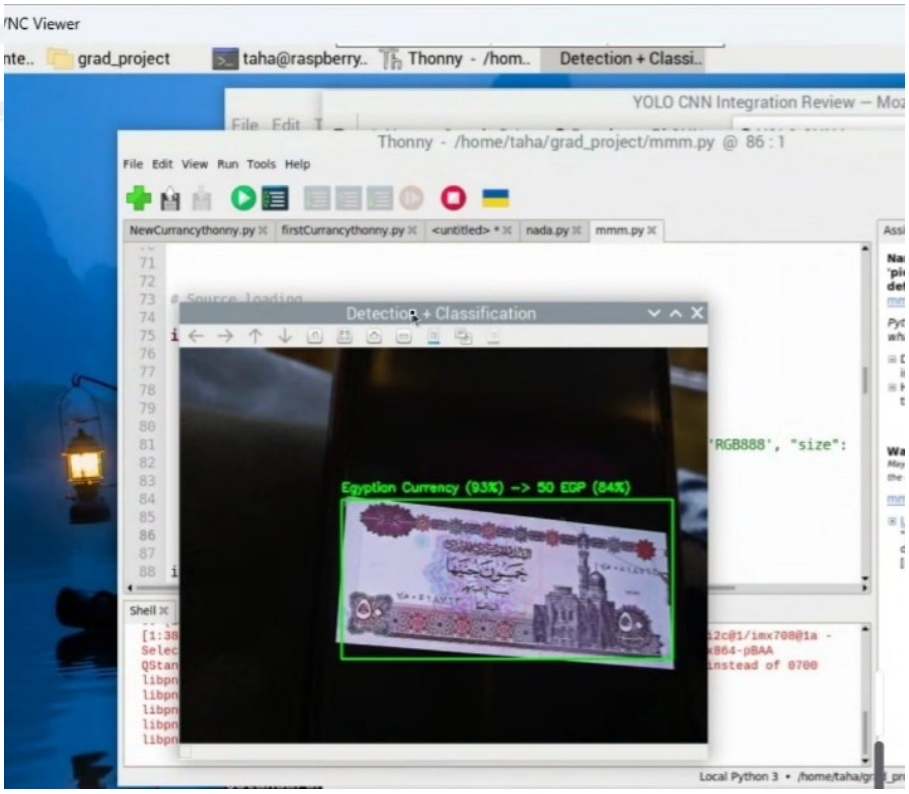


Figure 6-3 Currency pipeline guide

6.4.2 Text Reading Mode

Triggered by pressing the joystick to the right, this mode allows the user to recognize written content from books, signs, menus, or other printed sources. The system uses an OCR (Optical Character Recognition) engine to process the captured image and extract the text. Once extracted, the text is vocalized through the speaker. For example, if the camera captures a sign that says, “Innovation Nation” the system will announce this to the user.

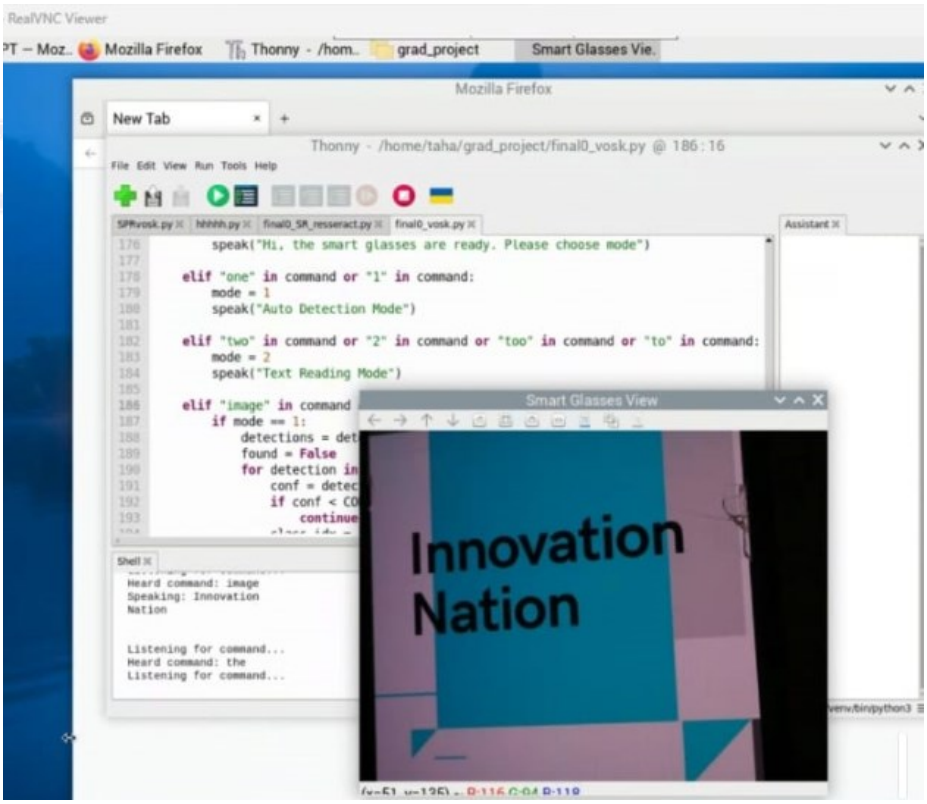


Figure 6-4 Text Reading pipeline guide

6.4.3 Scene Description Mode

When the joystick is pressed upward, the system activates the scene description mode. This function uses a vision-language transformer model to generate a natural language description of the scene. For example, it may say, “A person riding a motorcycle on a road.”

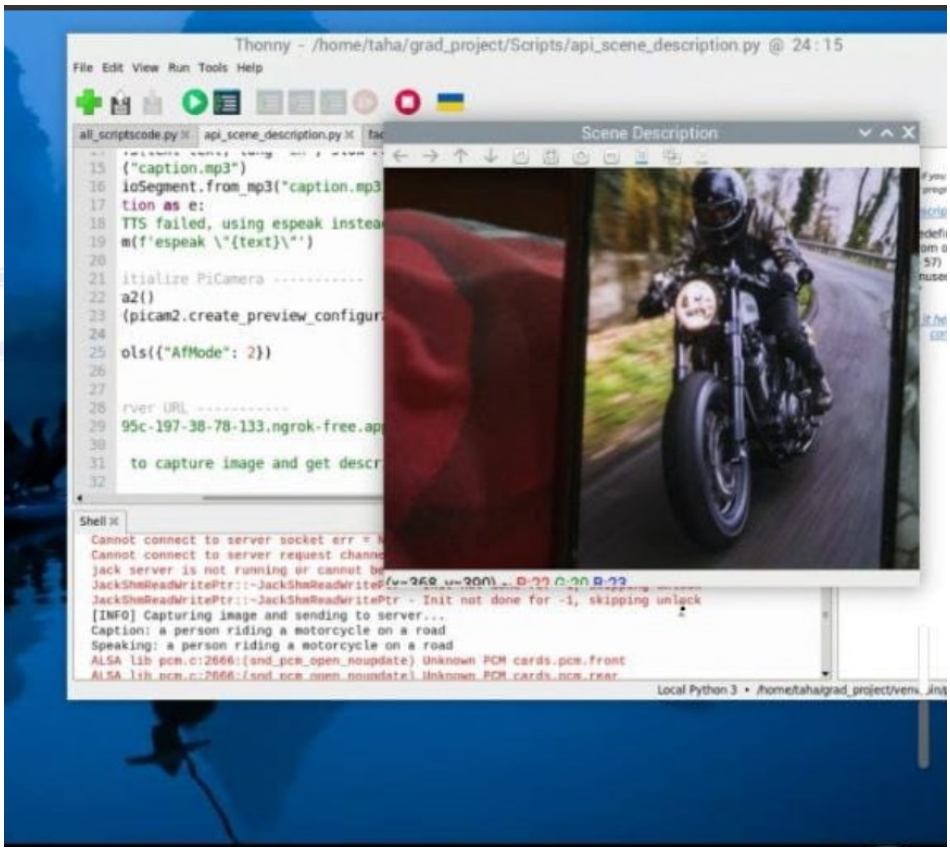


Figure 6-5 Scene Description pipeline guide

6.4.4 Face and Emotion Recognition Mode

By pressing the joystick downward, the face and emotion recognition mode is launched. This function detects their faces using a facial landmark detector and compares them against a pre-saved dataset to identify known people. Once a face is recognized, the system uses another model to estimate the person's emotional state and announces both pieces of information. For instance, it might say, "Cristiano Ronaldo looks Neutral."

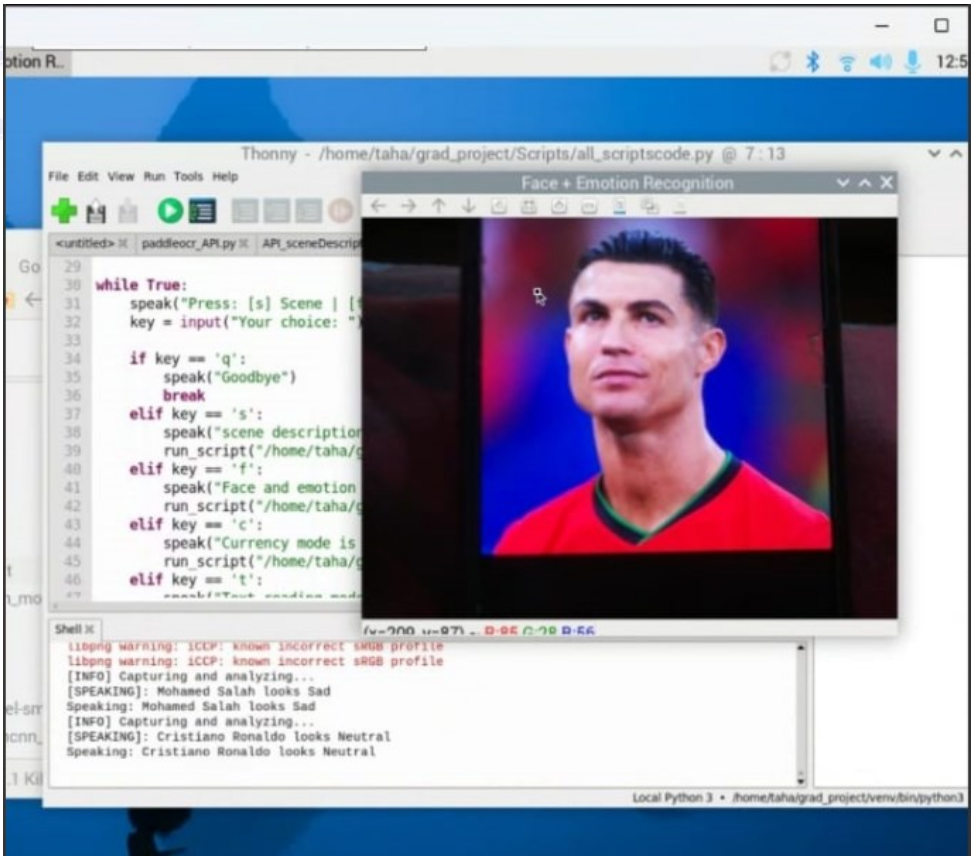


Figure 6-6 Face Recognition & Emotion detection

6.5 Exiting Modes and Shutting Down

To exit any mode and return to standby, users can perform a long press on the joystick. This ensures that only one functionality is active at a time, minimizing confusion and conserving processing power. When the user wants to completely shut down the system, a double press on the joystick is required. This action stops all processes and safely turns off the application. During active modes, a single press on the joystick captures a new image for analysis, giving the user precise control over when to trigger a ta

Chapter 7
Conclusion and Future Work

7.1 Conclusion

This project presents the successful design and implementation of a real-time assistive smart glasses system tailored for visually impaired individuals. The system operates on a lightweight multi-task architecture and runs on a Raspberry Pi, leveraging four parallel processing flows to provide contextual feedback via a bone-conduction speaker. These flows include: (1) the Currency Detection and Identification (CDI) flow, which uses two models to detect and recognize Egyptian currency with evaluation accuracies of approximately 95% and 96% respectively; (2) the Face Detection, Recognition, and Emotion (FDRE) flow, composed of Haar and MTCNN-based detection (accuracy ~95%), face recognition (accuracy ~99%), and facial expression recognition (accuracy ~60%); (3) the Scene Description (SD) flow, utilizing a BLIP transformer consisting of Text-Image Retrieval (TTIR) with ~71% accuracy, image classification (~82–88%), and zero-shot captioning (~90%); and (4) the Text Reading (TR) flow, based on PaddleOCR, which includes Differentiable Binarization Net for text detection (~80%) and an API Net for text recognition (~50%).

The system integrates a joystick-based control scheme that allows users to activate each flow intuitively with directional presses and simple actions such as long-press, double-press, and single-tap to manage operational modes or capture images. This hands-free, accessible interaction makes it practical for real-world deployment. Additionally, this work contributes to the broader field of computer vision by introducing annotated and labeled datasets for Egyptian currency detection and supporting the development of deep learning-based accessibility tools.

7.2 Future Work

Looking ahead, several enhancements can be explored to elevate both the performance and the functionality of the system. Expanding the dataset—particularly for currency, facial expressions, and printed text—will be vital for improving model accuracy and generalization. Developing an AI-based auto-annotation tool could significantly reduce the manual labor involved in dataset preparation. On the hardware side, upgrading to more powerful edge AI platforms like the NVIDIA Jetson Nano or Coral Dev Board would enable faster inference and support for more sophisticated models.

In terms of added features, several advanced functionalities are envisioned: **gesture recognition** for touchless interaction, **audio scene classification** to recognize environmental sounds (e.g., car horns or doorbells), and **monocular depth estimation** for improved spatial awareness. Combining **facial emotion recognition with voice tone analysis** could enhance the emotional understanding capabilities of the system. Implementing **low-light and blur enhancement** using GAN-based models will address challenges in poor lighting conditions. A **personalized AI assistant**, running locally, could provide users with interactive support, while optional **cloud synchronization and remote monitoring** would allow caregivers to track the user's location and receive real-time alerts—further enhancing safety and usability.

These improvements will not only broaden the scope of the project but also push it toward becoming a comprehensive, intelligent companion for the visually impaired, enabling greater autonomy and connection with the world.

7.3 References

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021, March).

Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data, 8*(1), 53. https://doi.org/10.1186/s40537-021-00444-8

BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation. (2022, January 28). *Hugging Face Transformers Documentation*. https://huggingface.co/docs/transformers/en/model_doc/blip

Du, Y., Li, C., Guo, R., Yin, X., Liu, W., Zhou, J., ... & Wang, H. (2020). Pp-ocr: A practical ultra lightweight ocr system. *arXiv preprint arXiv:2009.09941*.

Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc.

GitHub. (2008, April 10). *Build software better, together*. https://github.com

Google. (2018, November 7). *Google Colab*. https://colab.research.google.com/

IBM. (2021, October 6). *What are convolutional neural networks?* https://www.ibm.com/think/topics/convolutional-neural-networks

IBM. (2021, October 6). *What is a neural network?* https://www.ibm.com/think/topics/neural-networks

Kaggle. (2020, June 15). *Your machine learning and data science community*. https://www.kaggle.com/

Krishna, D. (2021, January 5). The components of a neural network. *Towards Data Science*. https://towardsdatascience.com/the-components-of-a-neural-network-af6244493b5b/

Lang, N. (2024, December 12). Activation functions in neural networks: How to choose the right one. *Towards Data Science*. https://towardsdatascience.com/activation-functions-in-neural-networks-how-to-choose-the-right-one-cb20414c04e5/

Mishra, M. (2020, August 26). Convolutional neural networks, explained. *Towards Data Science*. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939/

MistyRain. (2016, May 17, 2025). *imistyrain/MTCNN [Repository]*. GitHub. https://github.com/imistyrain/MTCNN

ngrok. (2013, March 20). *ngrok [Repository]*. GitHub. https://github.com/ngrok

Paszke, A. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

PaddlePaddle. (2025, June 5). *PaddlePaddle/PaddleOCR [Repository]*. GitHub. https://github.com/PaddlePaddle/PaddleOCR

Ramírez, S. (2018, April 7; updated June 9, 2025). *fastapi [Repository]*. GitHub. https://github.com/fastapi/fastapi

Roboflow. (2020). *Computer vision tools for developers and enterprises*. https://roboflow.com

ScienceDirect. *Classification neural network - an overview.* https://www.sciencedirect.com/topics/engineering/classification-neural-network

Supervisely. (2021). *Curate, label and build production models in one platform*. https://supervisely.com/

Ultralytics. (2023, January 10). *Docs/En/Models/Yolov8.md at main*. GitHub. https://github.com/ultralytics/ultralytics/blob/main/docs/en/models/yolov8.md

Viola, P., & Jones, M. (2001, December). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* (Vol. 1, pp. I-I). IEEE.