Menoufia University
Faculty of Electronic Engineering
Department of Computer Science and Engineering

Graduation Project

A. Eye

Artificial Eye

Supervisor
Assoc. Prof. Marwa A. Shouman

2021/2022

Menoufia University
Faculty of Electronic Engineering
Department of Computer Science and Engineering

Graduation Project

A. Eye

Artificial Eye

Supervisor

Assoc. Prof. Marwa A. Shouman

Team Members

AbdElaziz Ahmed Elsayed Elahwal
Ahmed Mohamed Abouzid El-Brawany
Ahmad Tharwat Mustafa Abd El-Hameed

2021/2022

Supervisor Head of the Department Dean

Acknowledgments

First and foremost, all praises is due to **Allah**, the Entirely Merciful

We would like to thank our supervisor, prof. Marwa Shouman. For her support over this long year, her patience with us in the tight deadlines, and her kind behavior when failures happen.

We want to thank Eng. Murtaza Hassan, owner and founder of the Murtaza Workshop YouTube channel that helped us throughout the year by providing top class tutorials that formed the seed for our ideas of the project.

We also want to thank Eng. Adam Geitgey, a Machine-Learning instructor, whom we learned from the core of face and speech recognition technologies.

All our thanks to Joseph Redmon and Ali Farhadi, the original authors of the YOLO algorithm that we used in both object detection and money recognition, and Alexey Bochoknovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao for who released version 4 of YOLO which is the version we used. Thank you for providing such a model with high accuracy.

Certainly we can not forget to thank the Raspberry Pi foundation for making such a powerful and useful microcomputer board that made such a project possible.

We thank our friends that helped us in collecting the Egyptian currency data, provide technical help in fields outside of our specialty, and made the project logo for us. Thank you Abdullah, Hamdi, Ahmed Shaaban, Ahmed Eyad, and Ahmed El-Sebaai.

In the end, we want to thank our parents and family for their continues support all the way to infinity, providing us with love, creative ideas, and more than we can ever mention.

Preface

Targeted Users and Technology

In this book, we discuss our graduation project, which is a device that helps blind and visually impaired individuals in their daily lives. We focused on implementing artificial intelligence technologies based on the computer vision and image processing fields to provide several services to the targeted set of users.

How to Use this Book

This book is structured in a bottom-up manner. We start with the feasibility of the project, moving on to the hardware and system configurations which are the backbone of the device, then we talk about the first class to run at boot which is the voice recognition, and we then dive deep into the services provided by our device to understand their theory.

To get the most out of this book, you can skip the feasibility study part, except for the costs, since the project is feasible (otherwise we wouldn't have made it). You can get back to it whenever you have extra time. Read the hardware chapter carefully to understand what you will work with. If you are using a different hardware, you might still want to read this chapter to get a general idea of the requirements you will need. As for the rest of the chapters, we suggest you go through each chapter carefully one at a time while reading the appendix related to it to understand the code we use. You can go this way or the opposite, depending on your strength point (programming or theory). And in the end, we suggest reading through the references whenever they are pointed out to deepen your understanding of the subject.

Table of Contents

Chapter 1: Feasibility Study

1.1 Introduction

In this chapter, we talk about the operational feasibility of our product, what scenarios we have searched for, what problems were urgent to solve, and we discuss a survey we have performed on doctors in a local hospital.

We talk about the software libraries available for public use, hardware suitable for our use case and its peripherals, and give a simple block diagram to visualize the process we need to start our work on the device.

1.2 Operational Feasibility

1.2.1 Applications Scenarios

We had a general idea of using a device based on artificial intelligence technology to help the public in one way or another. Therefore we searched for some possible scenarios to implement our idea through. The ideas we thought of were:

- Traffic control.
- Object counting in any possible application.
- Attendance system through face recognition.
- **Text reading.**
- **Assisting blind and visually impaired people.**

Through researching some more, we found a significant need for technology to help the visually impaired/blind individuals. So we have chosen to help this group by providing a standalone product to work as a voice assistant to help them navigate their environment and interact with it more easily without needing help and we provide an additional feature of text reading since it's a significant help to this group.

1.2.2 Problem Urgency

Vision Impairment

There are about 2.2 billion people who are diagnosed with near or distant vision impairment worldwide. One billion of them have severe vision impairment or blindness[1]. In Egypt there are 3 millions visually impaired individual and one million blind[2].

Based on some statistics, many people develop vision impairment over the age of 50. However, the loss of vision can affect people of all ages. Causes of vision loss vary, some examples are cataract, uncorrected refractive errors, age, corneal opacity, glaucoma, trachoma,.. etc[1].

Prosopagnosia (Face Blindness)

Prosopagnosia is the inability to recognize people's faces. There are 2 types of prosopagnosia: developmental prosopagnosia and acquired prosopagnosia[3].

Such a disease is thought to be rare but based on a researches and tests made by Ken Nakayama and Richard Russell[4][5], about 2 percent of the population are affected with that disease.

In the early days, most of prosopagnosia cases were thought to be due to a brain injury. Later on researchers have discovered that there are many people that have prosopagnosia without the cause of brain injury[3].

Dyslexia (Reading Disorder)

It is a known learning difficulty where the patient has problems with reading, writing, and sometimes spelling. People with dyslexia have difficulty recognizing different sounds that make up the words. It is not related to the person intelligence[6].

Statistics in 2017 suggest that there are approximately about 5% to 10% of the worldwide population suffering from dyslexia[7]. The cause of dyslexia is still unknown. But researchers found that it often shows to run in families[6].

1.2.3 Solutions

The solutions we came up with is to make a device that helps the user to:

- Detect objects and obstacles that s/he might encounter.
- Recognize faces of acquaintances, friends, and family members.
- Read text.

1.2.4 Survey

We have visited eye hospital in Menouf. We interviewed the hospital manager and several doctors and asked them several questions (see appendix A for the questionnaire). From the results we have concluded that:

- The project is very useful and will make a huge difference in the lifestyle of the end user.
- According to the specialists we interviewed, the expected price for a device to perform this function is expected to be very expensive from the commercial point of view. However, we are aiming to design and implement a low-cost device to serve the MENA region.
- If such a device is made in the market it would be recommended to the patients by doctors.
- They have no proposal for additional features.

- If the device is mass produced and it has spread in the market, many charity foundations would gladly provide such a device for free for the incapable patients.
- Nearly all the specialists agreed to participate in a clinical trial of this device.

1.3 Technical Feasibility

1.3.1 Labor

The labor of the project consist of its three team members:

- One hardware engineer.
- Two programmers.

1.3.2 Location

According to our chosen application scenarios, the project is multi-purposed and doesn't depend on a specific location in most cases; to assist a blind person in everyday tasks, the project needs to function in the general environment and to read text for the client, the device can’t have a location dependency for such a task.

1.3.3 Technology

The project consists of 3 parts. One part is concerned with the software and two parts are concerned with hardware.

Software

Our software consists mainly of python libraries that are used in object detection, face recognition, OCR, and text-to-speech functions. Those libraries are downloaded on a RasPian OS that is installed on the device.

Main Apparatus

Consists of the main parts responsible for the input, processing, and output of the project. Which are consecutively:

- Raspberry Pi Camera (for image input).
- Raspberry Pi (computer for processing and the main device).
- Earphone (for voice output).

Raspberry Pi Facilities

These components are connected to the raspberry pi to provide either needed or optional functionalities. They are categorized into two sections:

- Primary facilities:
  - Power bank (as the device power source).

- Raspberry Pi case (for protection).
- Heat sinks (for heat dissipation).
- MicroSD card (for installing OS and any software component on it).

- Secondary facilities:
  - HDMI cable (for Raspberry Pi screen viewing while programming).
  - I/O facilities: keyboard, mouse,.. etc.

1.3.4 Block Diagram

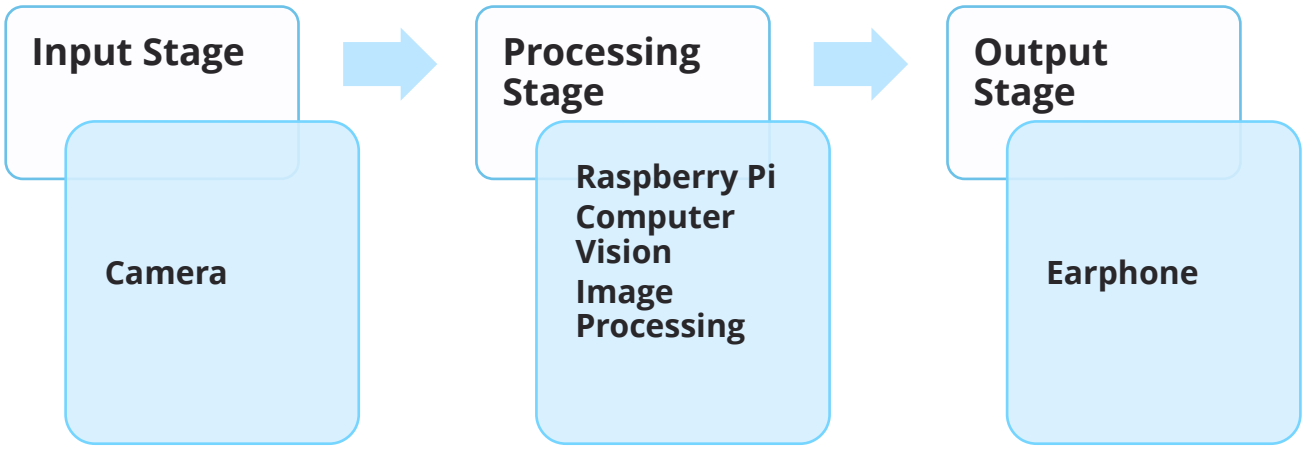


Figure 1-1: Block Diagram

1. At the input stage: the camera takes an image of the text/environment/object depending on the mode of operation that the user is using.
2. The raspberry pi fetches the image and starts to perform its operations, whether it's object detection, face recognition, or text recognition, using its corresponding python class and translates the text output to an audio file using a text-to-speech class to send to the output device (earphones) connected to it.
3. The sound file is then played on the earphones for the customer to listen to.

Chapter 2: Market Research

2.1 Introduction

In this chapter, we provide marketing related information. We talk about the needs for our customer base, discuss the competition and similar products in the market, performing a head to head comparison between some of them and our product, give an idea of the success rate expected from our product and success measurements we provide to visualize our vision, and then we explain in numbers how economically feasible our product is.

2.2 Customer Base

Our customer-base consists of visually impaired people and blind people and their needs are:

- Moving freely without the need for human help.
- Recognizing surrounding objects.
- Avoiding accidents usually caused by the impairment.
- Reading text.
- Being able to detect and recognize known acquaintances and family members.

2.3 Competition

Low-Vision Aids – المعينات البصرية

They are a governmental agency that provides three classes of services to the visually impaired patients:

- Classical vision aids such as magnifying glasses, telescopes, eye contacts, and glasses.
- Non-vision aids such as reading lamps, painted contacts, display devices, and bigger-fonts books.
- Electronic devices that might help blind individuals.

Mubser – مُبْصِر[8]

1. A 3D camera on the belt takes images of the environment and sends it to the pocket computer for processing.
2. The pocket computer performs its functions to navigate the patients.

3. The patient is navigated using audio on a Bluetooth headset or vibrations on vibration motors on the belt.

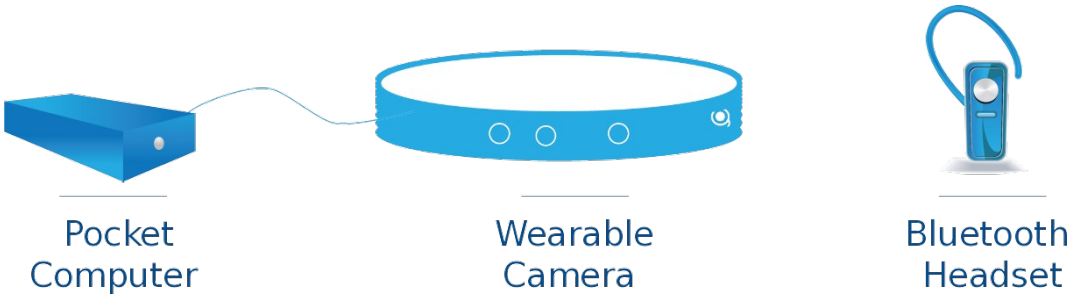


Figure 2-1: Mubser prject

Services:

- Calculating the number of people in a room.
- Detecting obstacles.
- Notification of obstacles existence using built-in vibration motors inside a belt.

Comparison:

	A. Eye	مُبْصِر
Processing unit	Raspberry Pi	Pocket computer
i/p	Raspberry Pi camera	3D camera
o/p	Voice	Vibrations and voice
no. of objects	80 objects	3 objects
Main technology	Python software	Embedded system

Arx *[9]*

This device consists of two main parts:

- ArxWear Headset:
  - Bone conduction speakers.
  - Microphone.
  - USB-C connection with phone.
- ArxApp:
  - Must be installed.
  - Detects and reads text.
  - Cloud text-to-speech.

Figure 2-2: ARx headphones

- Detects and remembers faces.
- Detects emotions.
- Detects objects.
- Describe scenes.
- Used with voice and/or touchscreen.

Comparison:

	A. Eye	ARx
Dependency	Standalone device	Depends on phone
Battery	Long battery	7 hours battery
Connectivity	Offline device	Requires internet
Earphones	External earphone	Bone conduction speakers

OrCam[10]

OrCam has two main products:

- OrCam MyEye. Features:
  - Read text.
  - Recognize faces.
  - Identify products.
  - Recognize barcodes.
  - Recognize money.
  - Identify colors.
- OrCam Read. Features:
  - Full page capture.
  - Laser guidance.
  - Read text.
  - Bright LED light.
  - Smart reading.
  - Point and click.

Figure 2-3: OrCam devices

2.4 Project Success Rate

Success rate for wearable visual assistants have been in continuous increase for the last decade ranging from old technology, of using simple sensors and vibrators which is hardware focused, to using new and smart technologies like computer vision and artificial intelligence software embedded in their respective hardware components or installed on third-party systems like smartphones.

2.5 Success Measurement

- Battery: device needs no more than one full charge per day.
- High performance: close to zero system failure downtime.
- Response time: below 1 second for usual use and bellow 200ms for danger alerts.
- Cost: the final product does not exceed 5,000 L.E to be suitable for the Egyptian market for clients with above-average income.
- Average user rating of more than 70% after the mass production stage.

2.6 Economic Feasibility

2.6.1 Cost-Benefit Analysis

Costs	
Software Costs	
Open source python libraries	0EGP
Hardware Costs	
Raspberry Pi 4 8GB Model B with 1.5GHz 64-bit quad-core CPU (8GB RAM)	1880EGP (Full Kit)[11]
32GB Samsung EVO+ Micro SD Card (Class 10) Pre-loaded with NOOBS	
USB MicroSD Card Reader	
CanaKit Premium High-Gloss Raspberry Pi 4 Case with Integrated Fan Mount	
CanaKit Low Noise Bearing System Fan	
CanaKit 3.5A USB-C Raspberry Pi 4 Power Supply with Noise Filter	
Set of Heat Sinks	
Micro HDMI to HDMI Cable - 6 foot (Supports up to 4K 60p)	
CanaKit USB-C PiSwitch (On/Off Power Switch for Raspberry Pi 4)	
Raspberry Pi Camera Board – V2 (8MP)	800EGP[12]

Xiaomi Powerbank 10000mA	400EGP[13]
USB Sound Card	20EGP
Earphone Splitter	15EGP
3D Printed Case	400EGP
Earphones	External
Total Cost	**3515EGP**
Benefits	
Providing cheaper alternative to similar solutions	
Providing mobility to the user without affecting his/her movement with mechanical parts	

2.6.2 Project Lifecycle

We are working on our project using sprints in Agile principles but with every sprint taking one month to complete due to our busy schedule throughout the year. The timeline was as follows:

- First two months: project feasibility study.
- Two months: searching for suitable software libraries.
- One month: hardware and linux installations and testing.
- Last two months: implementing and documenting the project software.

This should be a full-lifetime product that needs minimum maintenance since no software system updates/upgrades are required. Only hardware fixes might be needed and such problems should occur rarely.

Chapter 3: Hardware and Configurations

3.1 Introduction

In this chapter, we will discuss the Raspberry Pi device, Raspberry Pi 4 Model B in specific, why we chose Raspberry Pi and not another device, and talk about Raspberry Pi components, ports, and peripherals. Then we'll discuss how to install NOOBS to an empty microSD card, moving to setting up the hardware by the case and all peripherals we use for initial configuration.

After that we will get to know the Welcome Wizard, the Raspberry Pi configuration tool, and how to shut down the Raspberry Pi properly to not cause any crashing in the OS. We will recognize that we can open the Raspberry pi configuration tool either from the Raspbian menu (GUI) or from the terminal (CLI) as indicated in running program section.

After installing NOOBS and performing all initial configurations, like SSH service, we will learn that we mustn't access the Raspberry Pi in the same way; connecting it to a monitor via HDMI. We will access the Raspberry Pi using SSH service and VNC viewer. Then we will move to an essential section in our chapter, how to run our project software directly after booting the Raspberry Pi, then take a deep perception on the Raspberry Pi camera, our Artificial Eye (A.Eye), passing through how to install the camera module and adjust focus.

We will then talk about voice issue and its solution using USB sound card and earphone splitter. Then how the user interacts or uses the project functions that will be via either buttons connecting to GPIO of the Raspberry Pi or voice input or commands that inform the Raspberry pi to run certain function.

3.2 Raspberry Pi[33]

Raspberry Pi is known as a single-board computer, which means exactly what it sounds like: it's a computer, just like a desktop, laptop, or smartphone, but built on a single printed circuit board. Like most single-board computers, Raspberry Pi is small, roughly the same footprint as a credit card, but that doesn't mean it's not powerful; a Raspberry Pi can do anything a bigger and more power-hungry computer can do, though not necessarily as quickly.

Various models of Raspberry Pi have been released since the original Model B, each bringing either improved specifications or features specific to a particular use-case. The Raspberry Pi Zero family, for example, is a tiny version of the full-size Raspberry Pi which drops a few features, in particular the multiple USB ports and wired network port, in favor of a significantly smaller layout and reduced power requirements.

All Raspberry Pi models have one thing in common, though: they're compatible, meaning that software written for one model will run on any other model. It's even possible to take the very latest version of Raspberry Pi's operating system and run it on an original pre-launch Model B prototype. It will run more slowly, it's true, but it will still run. Throughout this chapter you'll be learning about Raspberry Pi 4 Model B, the latest and most powerful version of Raspberry Pi as of writing this book. What you learn, though, can be easily applied to other models in the Raspberry Pi family, so don't worry if you're using a different version.

Figure 3-1 shows a Raspberry Pi 4 Model B. If you're using a Raspberry Pi of the same or similar model with the guidance of this book, try to keep it turned the same way as in the picture; if it's turned around it can get confusing when it comes to using things like the GPIO header. While it may look like there's a lot packed into this tiny board, Raspberry Pi is very simple to understand.

We'll explain to you why we chose this model before diving deeper into its components, the inner workings that make the device tick.

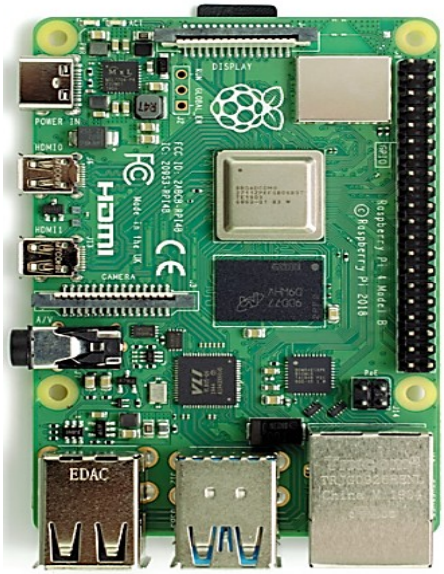


Figure 3-1: Raspberry Pi 4 Model B

3.2.1 Why Raspberry Pi[35]

The Raspberry Pi is a full Linux desktop computer which just happens to have access to the GPIO thanks to the Broadcom SoC (System on Chip). Connecting the Raspberry Pi to a monitor, keyboard and mouse provides us with a user experience not too different from a typical computer.

Because the Raspberry Pi runs Linux, it has access to many different programming languages, some of which can also be used with the GPIO. Python and Scratch are two obvious examples of languages that can work with the GPIO, but there are many others including Node-RED, Ruby and C.

The Raspberry Pi is a fully featured Linux desktop computer that can be used for day to day activities or as a server, but it also provides the GPIO being used in projects great and small. From simply flashing an LED, to computer vision, machine learning and robotics the Raspberry Pi packs lots of functionality into a credit card sized board.

Raspberry Pi has different models all powered by an ARM CPU. From the original single core 700MHz model in 2012, to the quad-core 1.5GHz model of now. The main difference between the Raspberry Pi 4 and its predecessors is:

- High speed (Faster processor)
- More RAM

3.2.2 Raspberry Pi Components

Like any computer, Raspberry Pi is made up of various components, each of which has a role to play in making it work. The first, and arguably most important, of them can be found just above the center point on the top side of the board (Figure 3-2), covered in a metal cap: the system-on-chip (SoC).

System-on-Chip (SoC)

This includes the central processing unit (CPU), commonly thought of as the 'brain' of a computer, and the graphics processing unit (GPU), which handles the visual side of things.

Figure 3-2: Raspberry Pi system-on-chip

Random Access Memory (RAM)

Another chip, which looks like a small, black, plastic square (Figure 3-3) is the Raspberry Pi random access memory (RAM). When you're working on Raspberry Pi, it's the RAM that holds what you're doing; only when you save your work will it be written to the microSD card. Together, these components form Raspberry Pi's volatile and non-volatile memories: the volatile RAM loses its contents whenever Raspberry Pi is powered off, while the non-volatile microSD card keeps its contents.

Figure 3-3: Raspberry Pi random acces memory (RAM)

Radio Module

At the top right of the board you'll find another metal lid (Figure 3-4). This metal lid is covering the radio, the component which gives Raspberry Pi the ability to communicate with devices wirelessly, using Wi-Fi or Bluetooth.

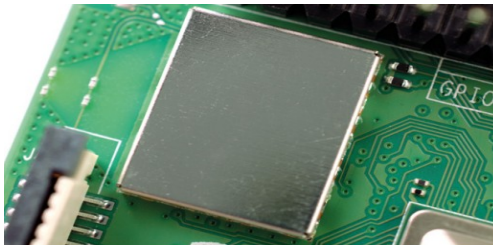


Figure 3-4: Raspberry Pi radio module

Power Management Integrated Circuit (PMIC)

Another black plastic-covered chip can be seen at the bottom edge of the board, just behind the middle set of USB ports. This is the USB controller, and is responsible for running the four USB ports in Model B. Next to this is an even smaller chip, the network controller, which handles Raspberry Pi's Ethernet network port. A final black chip, smaller than the rest, can be found a little bit above the USB Type-C power connector to the upper-left of the board (Figure 3-5), this is known as a Power Management Integrated Circuit (***PMIC*** for short). It handles turning the power that comes in from the micro USB port into the power Raspberry Pi needs to run.

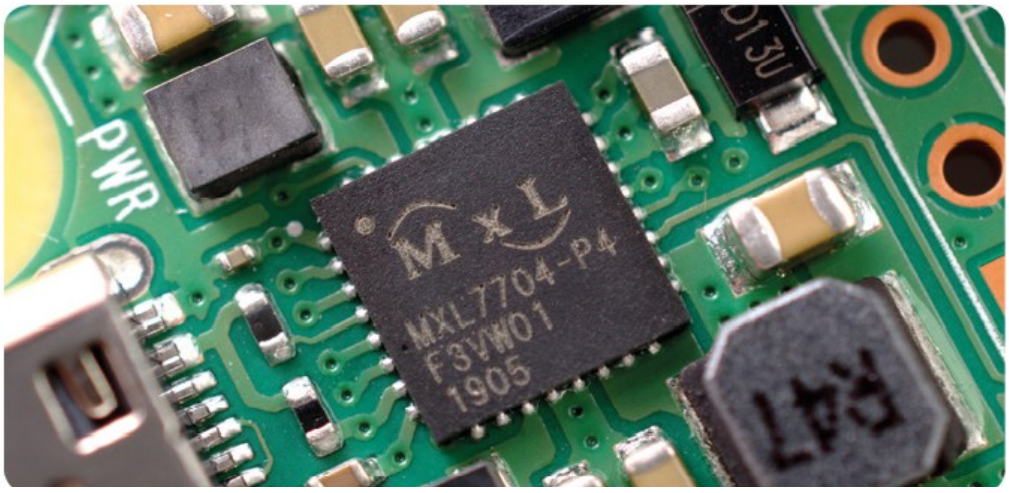


Figure 3-5: Raspberry Pi power management integrated circuit (PMIC)

3.2.3 Raspberry Pi Ports

Universal Serial Bus (USB)

Raspberry Pi 4 Model B has a range of ports, starting with four Universal Serial Bus (USB) ports (Figure 3-6) to the middle and right-hand side of the bottom edge. These ports let you connect any USB-compatible peripheral, from keyboards and mice to digital cameras and flash drives, to another Raspberry Pi even. Speaking technically, there are two types of USB ports: the ones with black parts inside are USB 2.0 ports, based on version two of the Universal Serial Bus standard; the ones with blue parts are faster USB 3.0 ports, based on the newer third version.

Figure 3-6: Raspberry Pi USB and Ethernet ports

Ethernet Port

To the right of the USB ports is an Ethernet port, also known as a network port (Figure 3-6). You can use this port to connect Raspberry Pi to a wired computer network using an Ethernet cable.

Audio-Visual (AV) Jack

Just behind the USB ports, on the left-hand edge of Raspberry Pi, is a 3.5 mm audio-visual (AV) jack (Figure 3-7). This is also known as the headphone jack, and it can be used for that exact purpose.

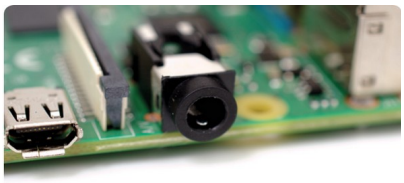


Figure 3-7: Raspberry Pi AV jack

Camera Serial Interface (CSI)

Directly next to the 3.5 mm AV jack is a strange-looking connector with a plastic flap which can be pulled up; this is the camera connector, also known as the Camera Serial Interface (CSI) (Figure 3-8). This allows you to use the specially designed Raspberry Pi Camera Module.

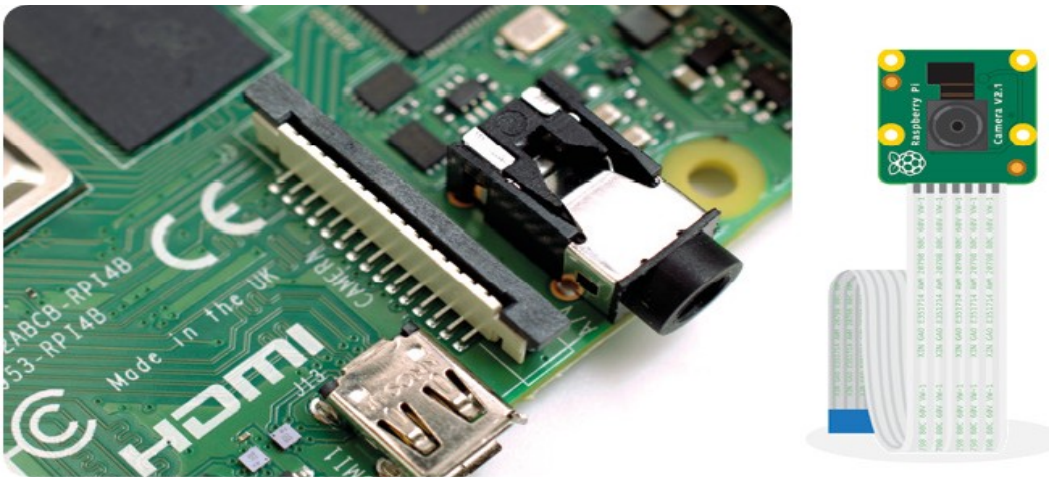


Figure 3-8: Raspberry Pi's camera connector

micro High Definition Multimedia Interface (micro-HDMI)

Above the CSI are the micro High Definition Multimedia Interface (micro-HDMI) ports (Figure 3-9), which are a smaller version of the connectors you'll find on a games console, set-top box, or TV. The multimedia part of its name tells you that it carries both audio and video signals, while high-definition tells you that you can expect excellent quality. You'll use these to connect Raspberry Pi to one or two display devices: a computer monitor, TV, or projector.

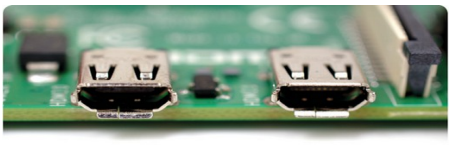


Figure 3-9: Raspberry Pi's micro-HDMI ports

USB Type-C Power Port

Next to the HDMI ports is a USB Type-C power port (Figure 3-10), which you'll use to connect Raspberry Pi to a power source. The USB Type-C port is a common sight on smartphones, tablets, and other portable devices. While you could use a standard mobile charger to power Raspberry Pi, for best results you should use the official Raspberry Pi USB Type-C Power Supply.

Figure 3-10: Raspberry Pi's USB Type-C power port

General-Purpose Input-Output (GPIO)

At the right-hand edge of the Model B board you'll find 40 metal pins, split into two rows of 20 pins (Figure 3-11). This is the GPIO (general-purpose input/output) header, a feature of Raspberry Pi used to talk to additional hardware from LEDs and buttons all the way to temperature sensors, joysticks, and pulse-rate monitors.

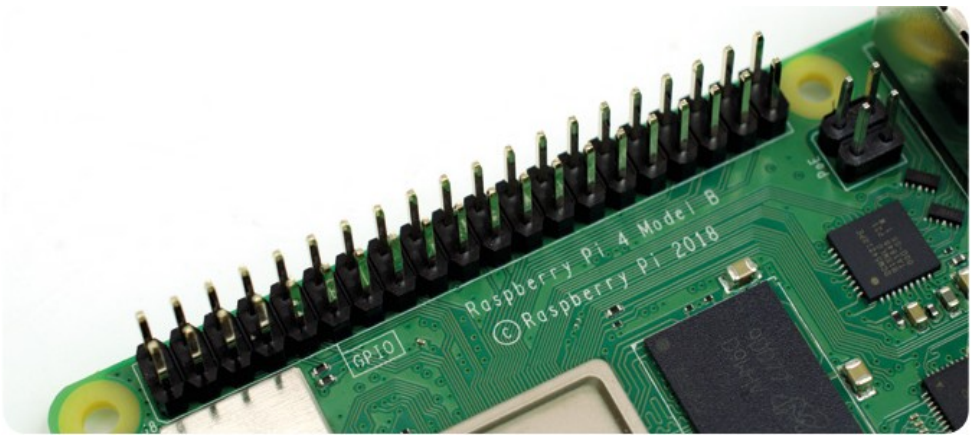


Figure 3-11: Raspberry Pi's GPIO header

MicroSD Card Connector

There's one final port on Raspberry Pi, but you won't see it on the top. Turn the board over and you'll find a microSD card connector on the opposite side of the board to the display connector (Figure 3-12). This is Raspberry Pi's storage: the microSD card inserted in here contains all the files you save, all the software you install, and the operating system that makes Raspberry Pi run.

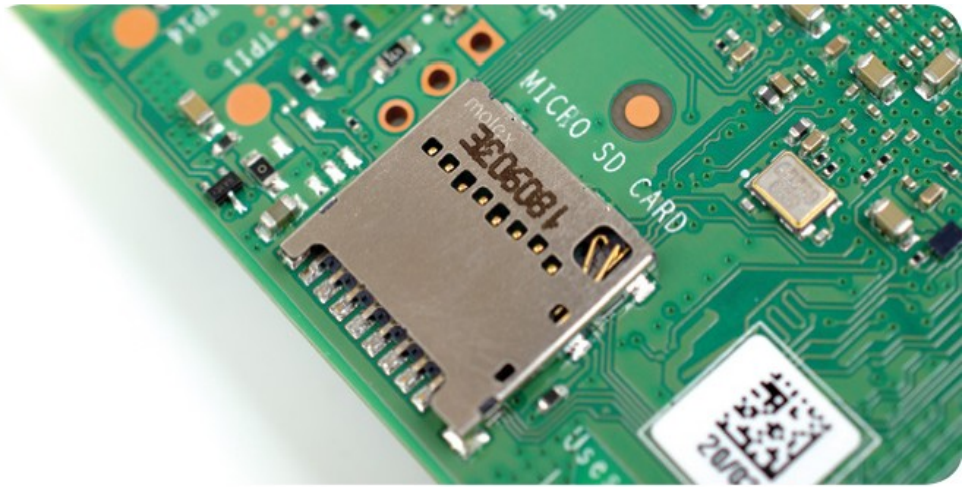


Figure 3-12: Raspberry Pi's microSD card connector

3.3 Raspberry Pi Peripherals[33]

A Raspberry Pi by itself can't do very much, just the same as a desktop computer on its own is a little more than a door-stop. To work, Raspberry Pi needs peripherals: at the minimum, you'll need a microSD card for storage; a monitor or TV so you can see what you're doing; a keyboard and mouse to tell Raspberry Pi what to do; and a 5 volt (5 V) USB Type-C power supply rated at 3 amps (3A) or better. With those, you've got yourself a fully functional computer. We'll tell you how to connect all these peripherals to your Raspberry Pi in the next sections.

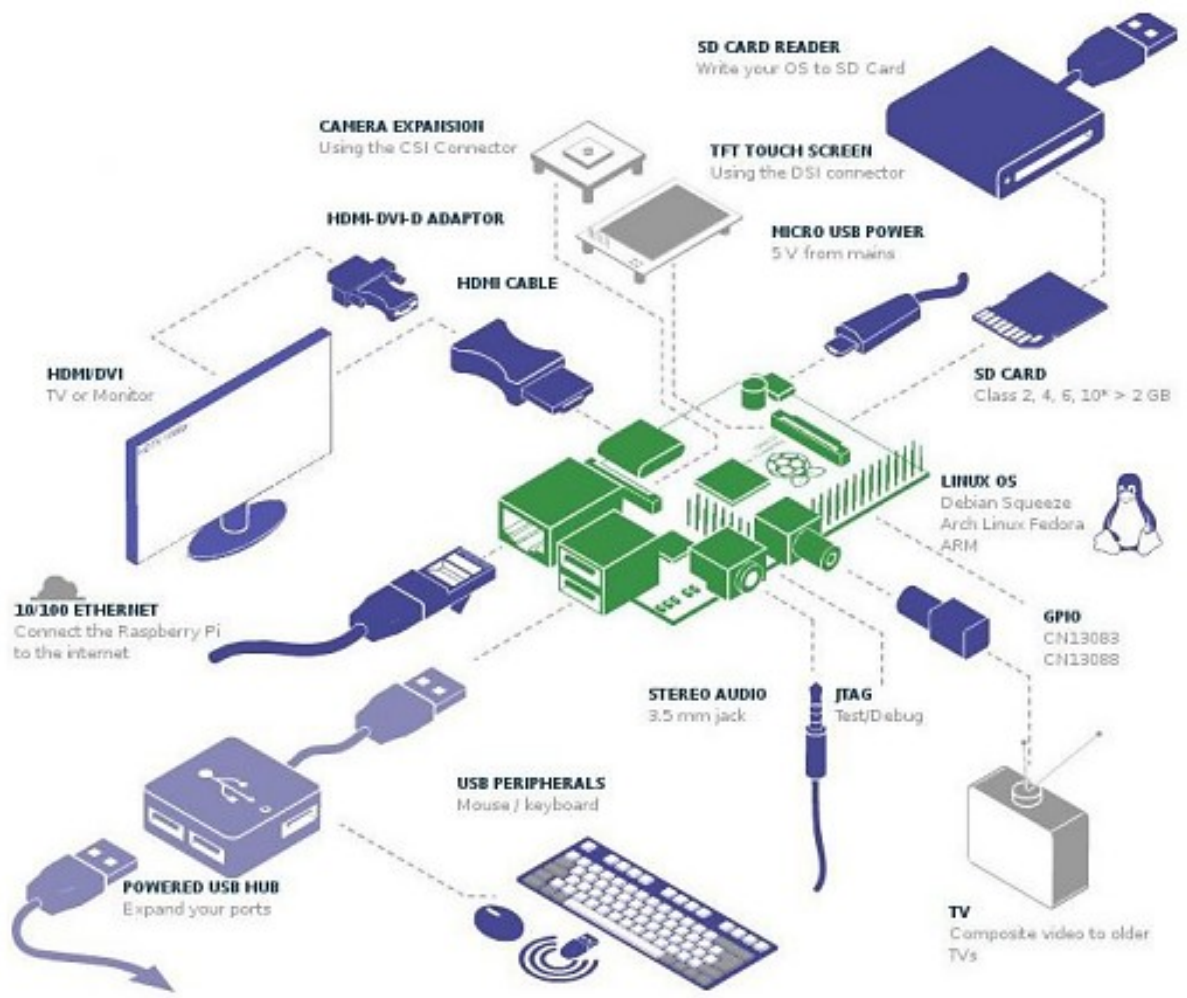


Figure 3-13: Raspberry Pi and its peripherals

3.3.1 Installing NOOBS to a microSD Card

The New Out Of the Box Software (NOOBS) is designed to make it as easy as possible to install and set up operating systems on your Raspberry Pi. You can use the following instructions to do that on your own microSD card.

When installing NOOBS onto a new blank or previously used microSD card, you'll first need to download it from the Raspberry Pi website. On a computer with a microSD card reader, or a full-size SD card reader and a microSD card adapter, open the web browser and type `rpf.io/downloads` into its address bar. From the page that loads, click NOOBS – marked with a raspberry icon – then click 'Download ZIP' under 'NOOBS Offline and network install' as shown in Figure 3-14 below.

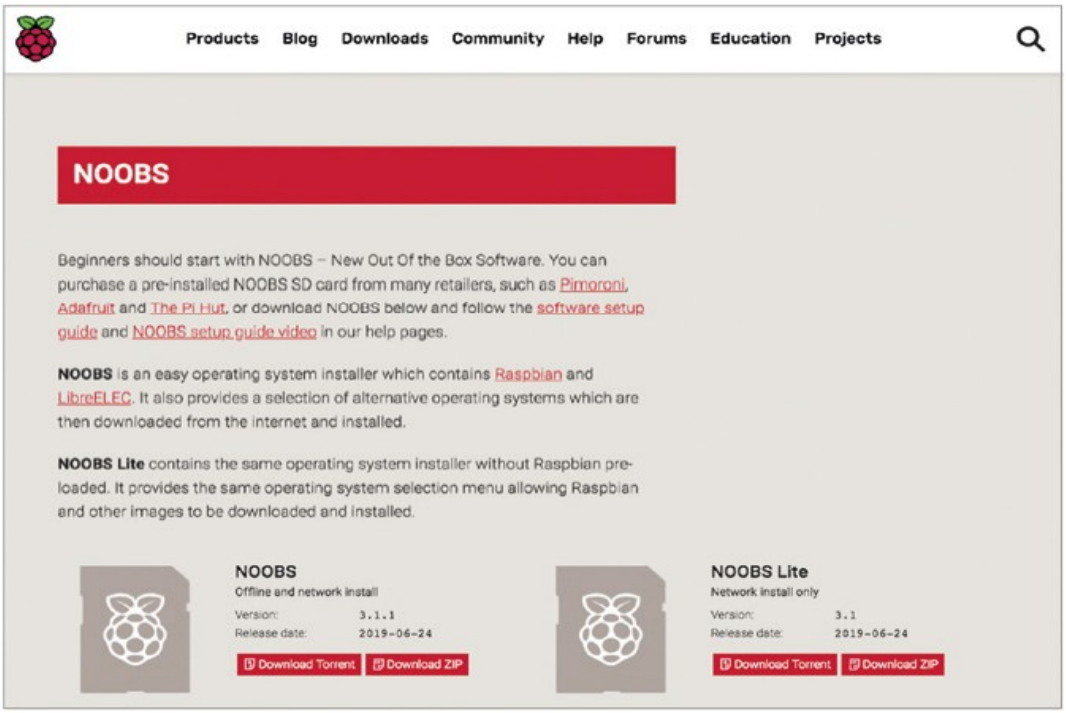


Figure 3-14: Downloading NOOBS

The NOOBS ZIP file is quite large, and can take a while to download on a slower internet connection. When the download has finished, plug your microSD card into your PC as shown in Figure 3-15. It should show up on your desktop/laptop as a single removable drive; if it doesn't, it may need to be formatted first.

Figure 3-15: Plug your microSD card into your PC

To format a previously used microSD card to be ready for installing NOOBS, Windows, and macOS users should download the SD Card Association SD Memory Card Formatter tool from `rpf.io/sdcard`, then double-click to install it. Linux users should use their distribution's disk management tool to delete any existing partitions on the disk, such as the dd utility found in most linux distributions, create a single partition, and format it as VFAT, then move onto the next section of this guide.

Insert your microSD card into your card reader, if you haven't done so already, and load the SD Card Formatter tool as shown in Figure 3-16 below. Look for your microSD card in the 'Select card' list; if you're reformatting a microSD card that has already been used with a Raspberry Pi, you may find it has more than one entry. Just select any one of them and double-check that you selected the correct drive by looking at the 'Card information' section: it should report the size and type of the microSD card you inserted. If the information is wrong, select a different entry from the 'Select card' list and check again.

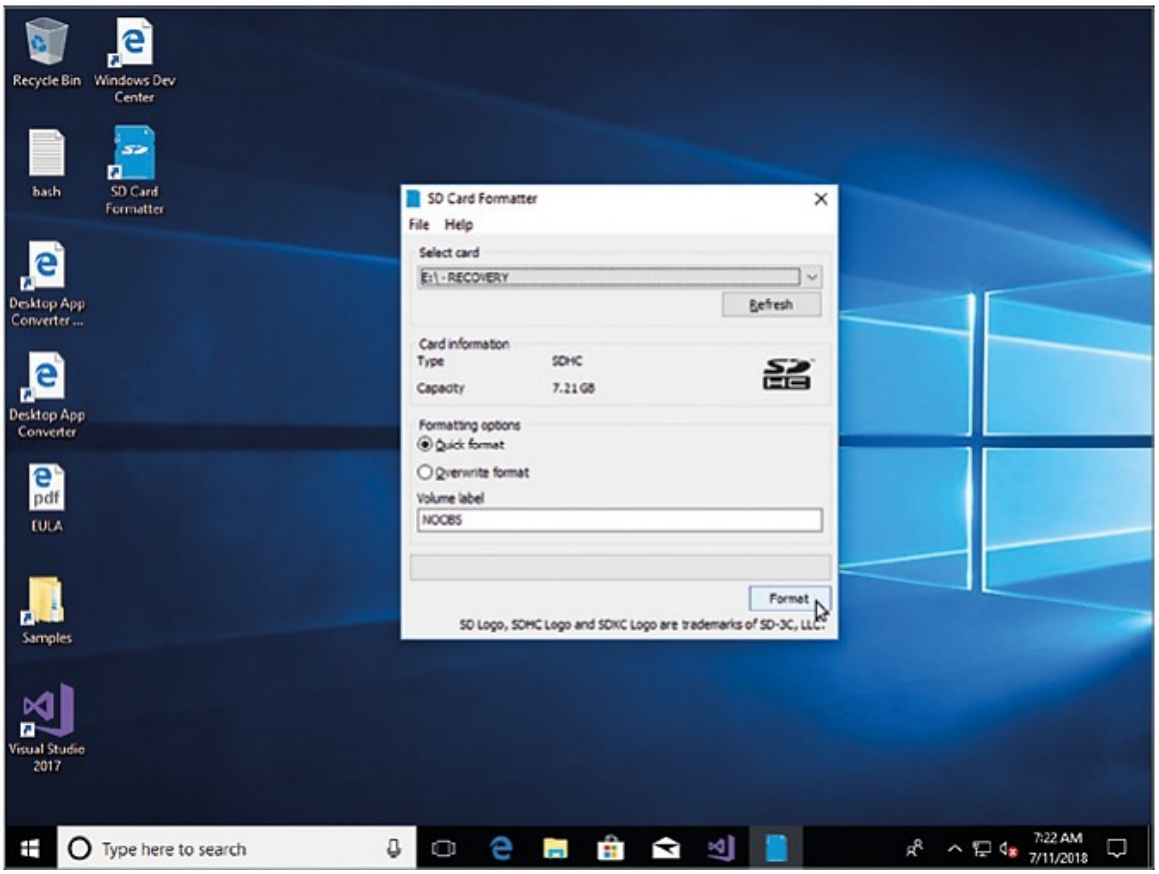


Figure 3-16: SD Card Formatter tool

When you're absolutely sure you've picked the correct microSD card, and you've backed up any files you want to keep, in case it was previously used, type 'NOOBS' into the 'Volume label' box, click the Format button, and confirm you want to overwrite the card. The default 'quick format' mode should only take a few seconds to complete, after which you can close the SD Card Formatter.

Installing NOOBS is as simple as drag-and-drop. Start by finding the NOOBS archive file, which should be in your Downloads folder. Double-click on the archive to open it, then press CTRL+A on your keyboard to select all the files in the archive. Click on one of the files with the left mouse button, and drag them to the removable drive representing your microSD card. Let go of the mouse button to drop the files, and wait for them to be copied to the microSD, this can take a few minutes. When the files are successfully copied, eject the microSD card from the computer and insert it into your Raspberry Pi.

3.3.2 Setting up Hardware[33]

Begin by unpacking your Raspberry Pi from its box. Raspberry Pi is a robust piece of hardware, but that doesn't mean it's indestructible: try to get into the habit of holding the board by the edges, rather than on its flat sides, and be extra careful around the raised metal pins. If these pins are bent, at best it'll make using add-on boards and other extra hardware difficult and, at worst, can cause a short circuit that will damage your Raspberry Pi.

Assembling the case

Raspberry Pi casing should be your first step. If you're using the Official Raspberry Pi Case, begin by splitting it into its two individual pieces: the red base and white lid. Take the base and hold it so that the raised end is to your left and the lower end to your right as shown in Figure 3-17 below.

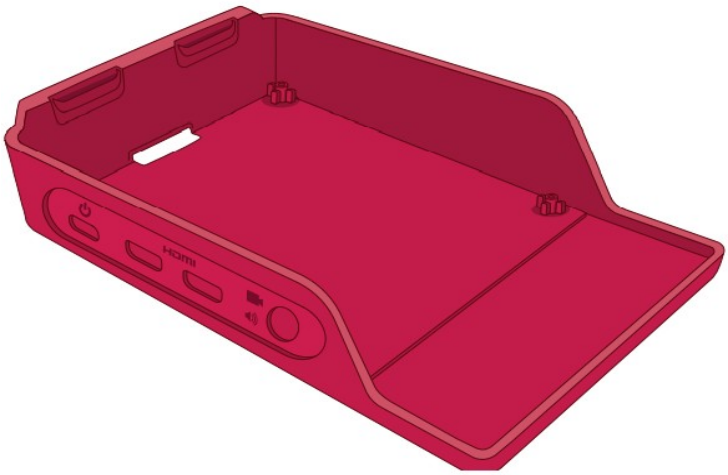


Figure 3-17: The base case

Holding your Raspberry Pi (with no microSD card inserted) by its USB and Ethernet ports, at a slight angle, slot its connectors (USB Type-C, 2 × micro-HDMI, and 3.5 mm) into their holes in the side of the base, then gently lower the other side down so it sits flat as shown in Figure 3-18.

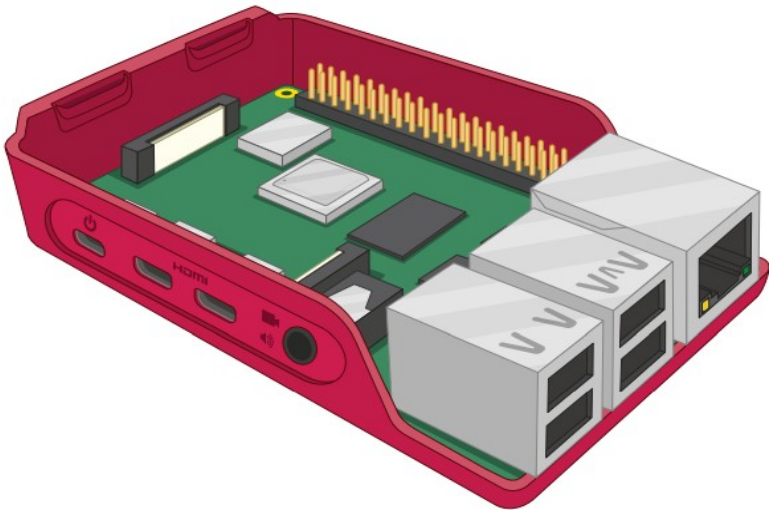


Figure 3-18: Raspberry Pi inside the base case

Take the white lid and place the two clips at the left into the matching holes on the left of the base, above the microSD card slot. When they're in place, push the right-hand side (above the USB ports) down, as shown in Figure 3-19 below, until you hear a click.

Figure 3-19: Raspberry Pi inside the whole case

Installing the microSD Card

To install the microSD card, which is Raspberry Pi's storage, turn Raspberry Pi over and slide the card into the microSD slot as shown in Figure 3-20 with the label facing away from Raspberry Pi. It can only go in one way, and should slide in the slot without too much pressure. The microSD card will slide into the connector as shown in below, then stop without a click.

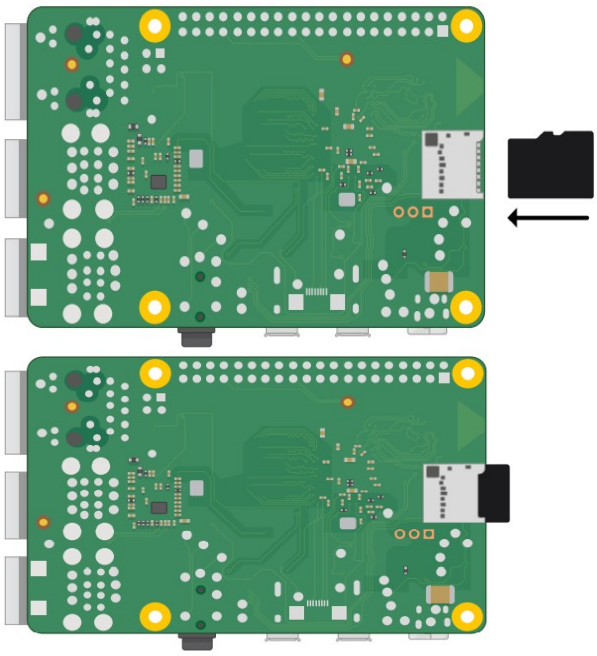


Figure 3-20: Slide the card into the microSD slot

If you want to remove it again in the future, simply grip the end of the card and pull it gently out. If you're using an older model of Raspberry Pi, you'll need to give the card a gentle push first to unlock it; this isn't necessary with a Raspberry Pi 3 or 4.

Connecting other peripherals

You can connect a keyboard and mouse as shown in Figure 3-21 below.

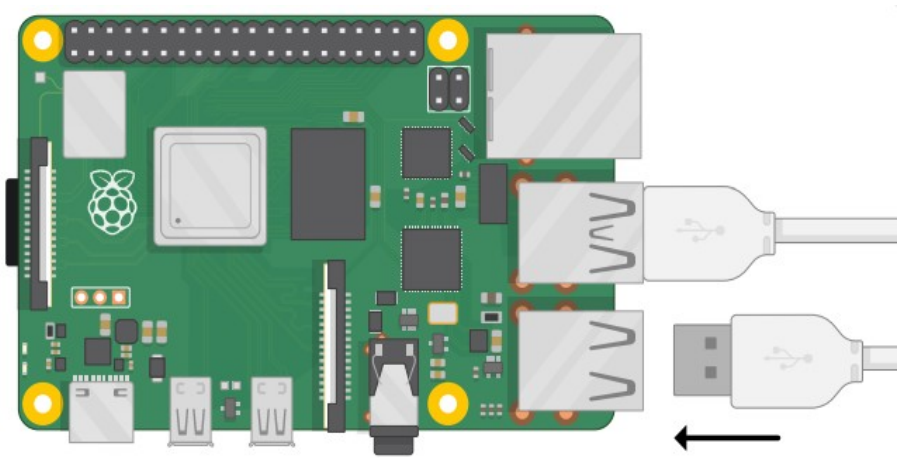


Figure 3-21: Connecting keyboard and mouse

To connect a display, take the micro-HDMI cable and connect the smaller end to the micro-HDMI port closest to the USB Type-C port on your Raspberry Pi as shown below, and the other end to your display or monitor.

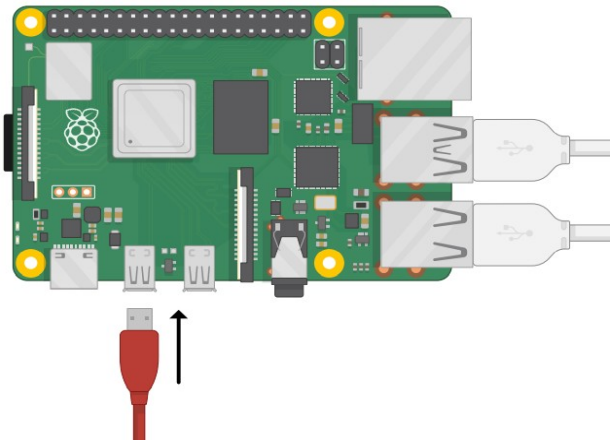


Figure 3-22: Connecting the micro-HDMI port

Connecting a network cable (if needed) is shown in Figure 3-23 below.

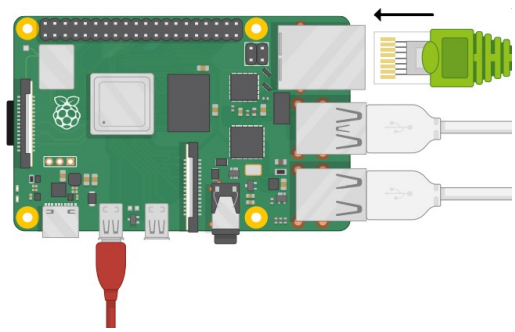


Figure 3-23: Network cable

Connecting a power supply, as shown in Figure 3-24 below, is the very last step in the hardware setup process, and it's one you should do only when you're ready to set up its software: Raspberry Pi does not have a power switch and will turn on as soon as it is connected to a live power supply. Connect the power supply to a main socket and switch the socket on; your Raspberry Pi will immediately start running.

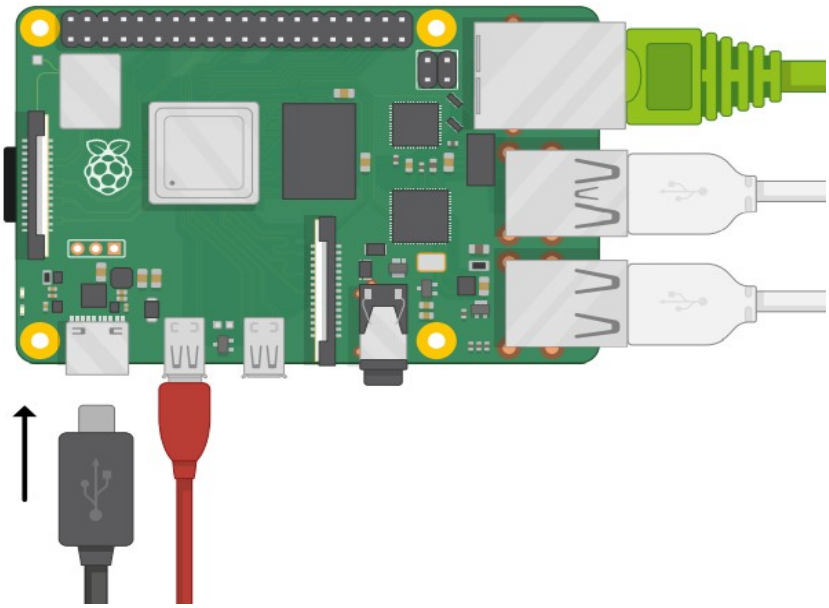


Figure 3-24: Connect a power supply

Your Raspberry Pi is switched on, NOOBS will load and ask you to choose your operating system.

Figure 3-25: Choose your operating system

This is the NOOBS menu, a system which lets you choose the operating system to run on your Raspberry Pi. Two operating systems are included with NOOBS as standard: Raspbian, a version of the Debian Linux operating system tailored specifically for Raspberry Pi; and LibreELEC, a version of the Kodi Entertainment Centre software.

To begin installing an operating system, use the mouse to put a cross in the box to the left of Raspbian Full. When you've done so, you'll see that the 'Install (i)' menu icon, is no longer greyed-out; this lets you know that your operating system is ready to install.

Click the 'Install (i)' icon once with the left mouse button and you'll see a warning message telling you that installing the operating system will overwrite any data currently stored on the microSD card – not counting NOOBS itself, which stays intact. Click 'Yes' and the installation process will begin as shown in Figure 3-26 below.

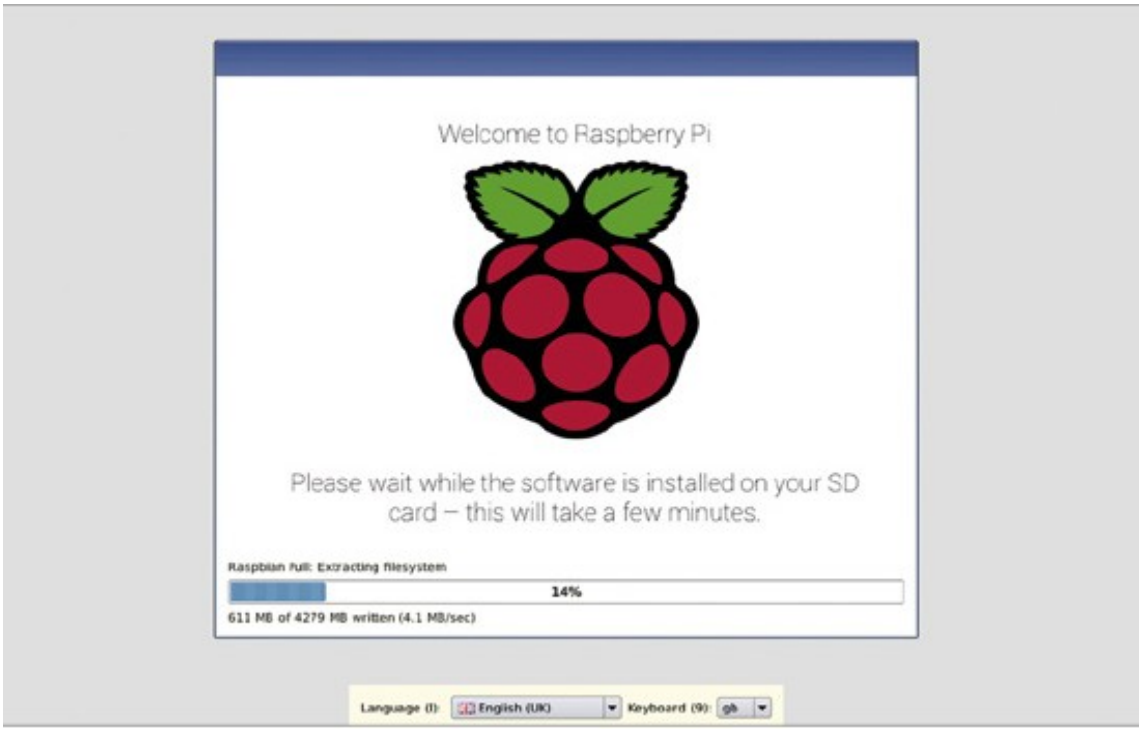


Figure 3-26: The installation process

The installation process can take anything from 10 to 30 minutes, depending on the speed of your microSD card. When the installation has finished, a window will pop up with an 'OK' button; click this and Raspberry Pi will restart into its freshly installed operating system. Note that the first time you boot into Raspbian, it can take a minute or two as it adjusts itself to make the best use of the free space on your microSD card. The next time you boot, things will go more quickly.

3.4 RaspianOS

3.4.1 The Welcome Wizard

The first time you run Raspbian, you'll see the Welcome Wizard as shown in Figure 3-27 below. This helpful tool will walk you through changing some settings in Raspbian, known as the configuration, to match how and where you will be using Raspberry Pi.

Figure 3-27: Welcome Wizard

Click the Next button, then choose your country, language, and time zone by clicking on each drop-down box in turn and selecting your answer from the list as shown in Figure 3-28. If you want the desktop and programs to appear in English, regardless of your country's native language, click on the 'Use English language' checkbox to tick it. When you're finished, click Next.

Figure 3-28: Choose your country, language, and time zone

The next screen will ask you to change the password for the 'pi' user (from the default 'raspberry') for security purposes, it's a very good idea to create a new one. Enter it in the boxes.

Figure 3-29: Change the password for the 'pi' user

The next screen will allow you to choose your WiFi network from a list. Scroll through the list of networks with the mouse or keyboard, find your network's name, click on it, then click Next. Click Next to connect to the network. If you don't want to connect to a wireless network, just click Skip as shown in Figure 3-30 below.

Figure 3-30: Select WiFi Network

The next screen will allow you to check for and install updates for Raspbian and the other software on Raspberry Pi. Raspbian is regularly updated to fix bugs, add new features, and improve performance. To install these updates, click Next; otherwise, click Skip. Downloading the updates can take several minutes, so be patient. When the updates are installed, a window saying 'System is up to date' will appear; click the OK button.

The final screen of the Welcome Wizard, as shown in Figure 3-31 below, has a simple task to do: certain changes made will only take effect when you reboot your Raspberry Pi. If prompted to do so, click the Reboot button and Raspberry Pi will restart. This time the Welcome Wizard won't appear; its job is done, and your Raspberry Pi is ready to be used.

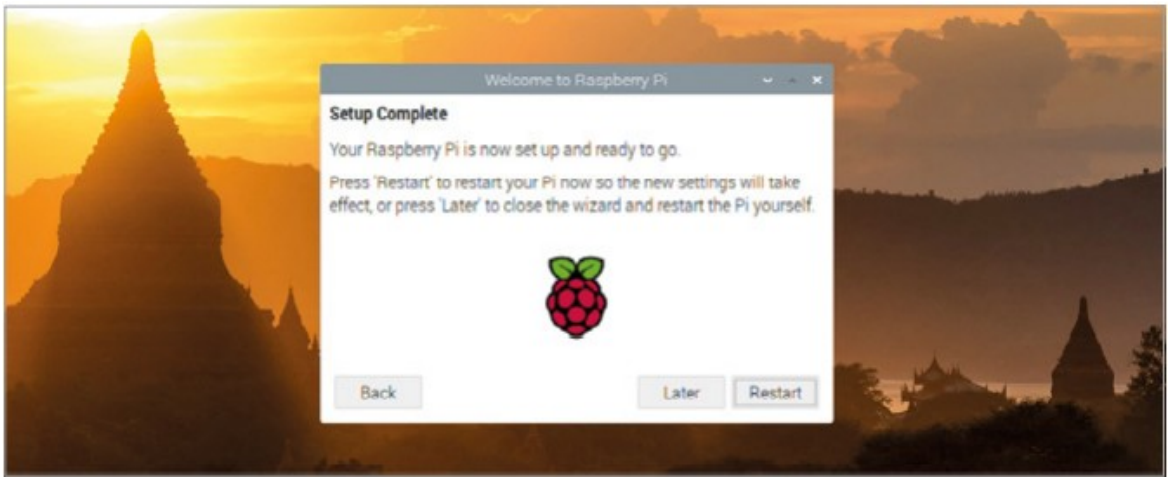


Figure 3-31: Setup complete

3.4.2 Raspberry Pi Configuration tool

It's a lot like the Welcome Wizard you used at the start: it allows you to change various settings in Raspbian. Click on the raspberry icon at the far left, move your mouse pointer to select the Preferences category, then click on Raspberry Pi Configuration to load as shown in Figure 3-32.

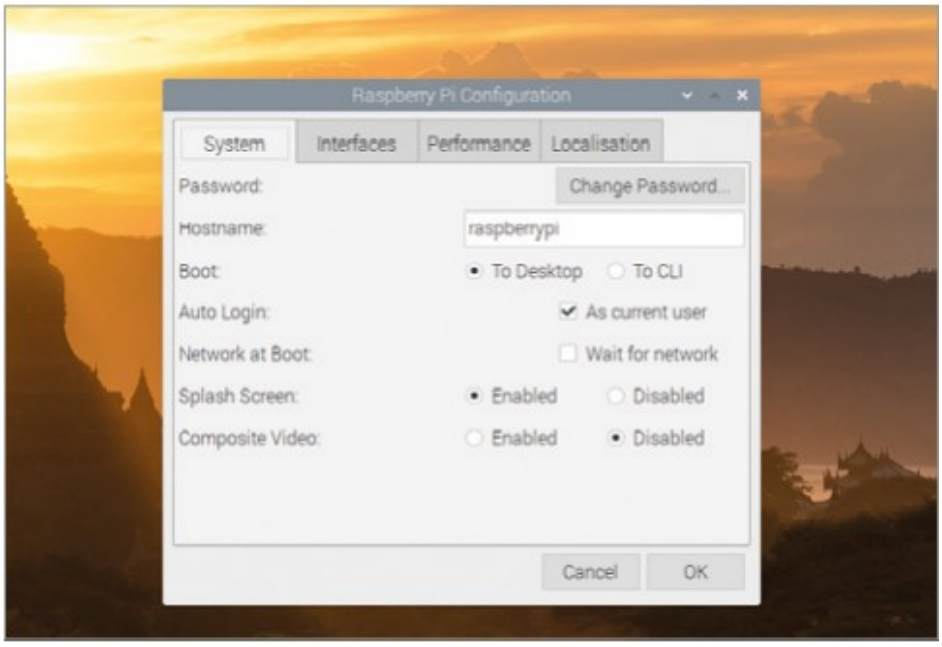


Figure 3-32: Raspberry Pi Configuration tool

The tool is split into four tabs, each of which can control a particular aspect of Raspbian. The first of these, which you see when the tool is first loaded, is System: this allows you to change the password of your account, set a host name – the name Raspberry Pi uses on your local wireless or wired network – and change a range of other settings. The majority of these, though, shouldn't need changing.

Click on the Interfaces tab to bring up the next category. Here you'll find a range of settings, all of which start off disabled. These settings should only be changed if you're adding new hardware, such as the Raspberry Pi Camera Module, and then only if instructed by the hardware's manufacturer. The exceptions to this rule are: SSH, which enables a 'Secure Shell' and lets you log into Raspberry Pi from another computer on your network using an SSH client; VNC, which enables a 'Virtual Network Computer' and lets you see and control the Raspbian desktop from another computer on your network using a VNC client.

Click on the Performance tab to see the third category. Here you can set the amount of memory used by Raspberry Pi's graphics processing unit (GPU) and, for some models, increase the performance of Raspberry Pi through a process known as overclocking. As before, though, it's best to leave these settings alone unless you know you need to change them. Finally, click on the Localisation tab to see the last category. Here you can change your locale, which controls things like the language used in Raspbian and how numbers are displayed, change the time zone, change the

keyboard layout, and set your country for WiFi purposes. For now, though, just click on Cancel to close the tool without making any changes.

3.4.3 Shutting down

Now you've explored the Raspbian desktop, it's time to learn a very important skill: safely shutting your Raspberry Pi down. The documents you're working on aren't the only files open, though. Raspbian itself keeps a number of files open while it's running, and pulling the power cable from your Raspberry Pi while these are still open can result in the operating system becoming corrupt and needing to be reinstalled. To prevent this from happening, you need to make sure you tell Raspbian to save all its files and make itself ready for being powered off – a process known as shutting down the operating system.

Click on the raspberry icon at the top left of the desktop and then click on Shutdown. A window will appear with three options: Shutdown, Reboot, and Logout. Shutdown is the option you'll use most: clicking on this will tell Raspbian to close all open software and files, then shut Raspberry Pi down. Once the display has gone black, wait a few seconds until the flashing green light on Raspberry Pi goes off; then it's safe to turn off the power supply. To turn Raspberry Pi back on, simply disconnect then reconnect the power cable, or toggle the power at the wall socket.

Reboot goes through a similar process to Shutdown, closing everything down, but instead of turning Raspberry Pi's power off, it restarts Raspberry Pi. You'll need to use Reboot if you make certain changes which require a restart of the operating system such as installing certain updates to its core software.

3.4.4 Running programs

Some programs can only be run at the command line, while others have both graphical and command-line interfaces. An example of the latter is the Raspberry Pi Software Configuration Tool, which you would normally load from the terminal as shown below.

Type:

```
raspi-config
```

You'll be given an error telling you that the software can only be run as root, the superuser account on your Raspberry Pi. It will also tell you how to do that, by typing:

```
sudo raspi-config
```

The **sudo** part of the command means switch-user do, and tells Raspbian to run the command as the root user.

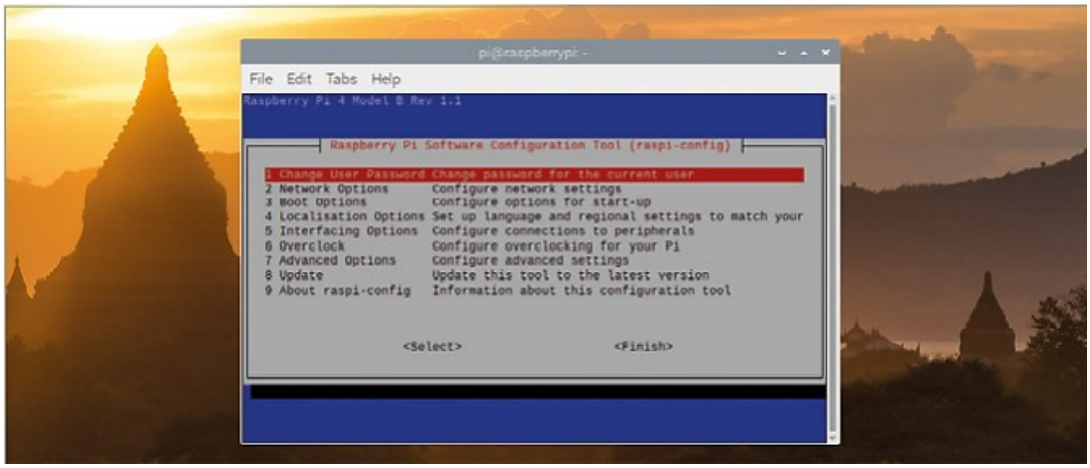


Figure 3-33: Load the software configuration tool from terminal

Press the **TAB** key twice to select Finish and press **ENTER** to quit the Raspberry Pi Software Configuration Tool and return to the command-line interface. Finally, type:

```
exit
```

This will end your command-line interface session and close the Terminal app.

3.5 Methods of accessing Raspberry pi

There are a lot of methods to connect to the Raspberry pi, we can be content with two.

1. You can access the Raspberry pi using HDMI (between projector or monitor and the Raspberry pi).
2. You can access the Raspberry pi using SSH and VNC services.

Accessing Raspberry pi via SSH and VNC services

To be able to open a command line interface on Raspberry Pi from another computer on your network using an SSH client. We must enable SSH service on Raspberry pi via the Raspberry Pi Configuration Tool as shown in Figure 3-34 below.

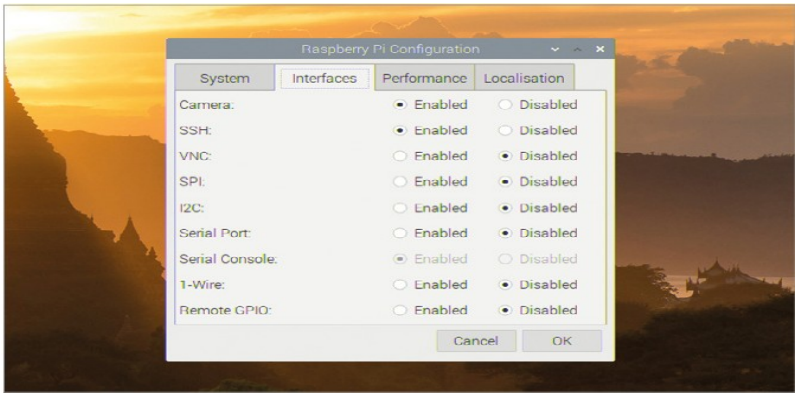


Figure 3-33: Enable SSH service on Raspberry pi

Next, you need to download and install mainly two software on your laptop and they are:

1. PuTTY(Optional).
2. VNC Viewer.

Our next task is to find the exact IP address of the Raspberry Pi 4. For this, you can create a local network and connect the Raspberry pi to it. Note the laptop and the Raspberry pi must be in the same subnetwork.

Then, open the putty and enter the IP of the Raspberry pi, with SSH connection type, port 22 as shown in Figure 3-34 below.

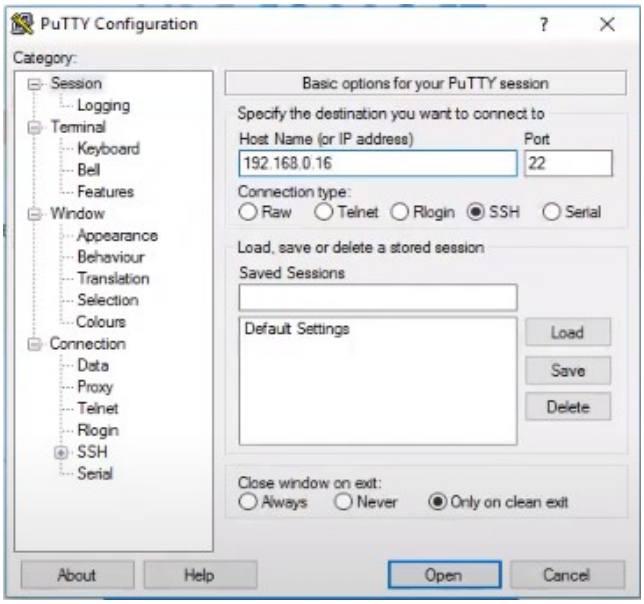


Figure 3-34: PuTTy configuration

Then click Open, a Cmd window will appear for authentication as shown in (Fig.35) below.

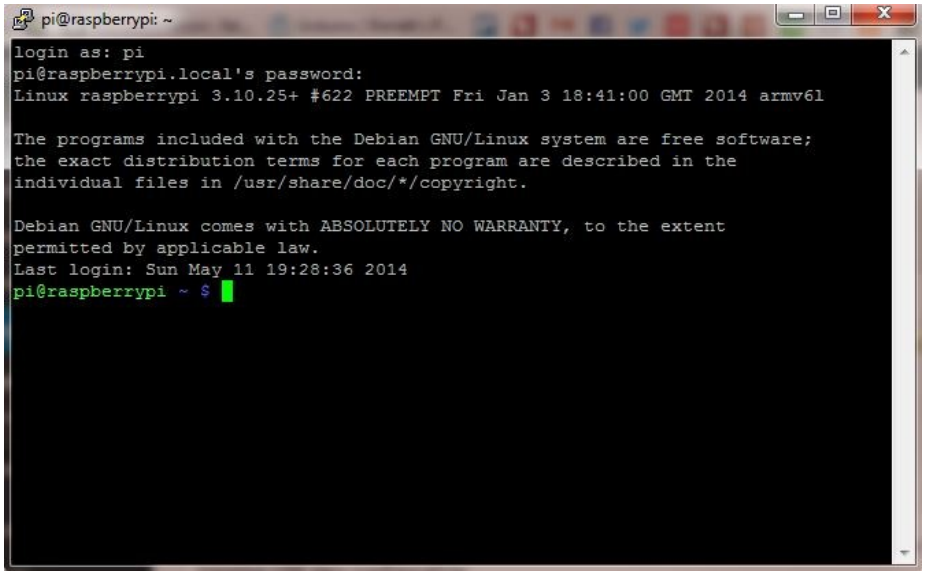


Figure 3-35: Connect to the Raspberry pi

Now, you need to setup a VNC Server as shown in (Fig.36) below

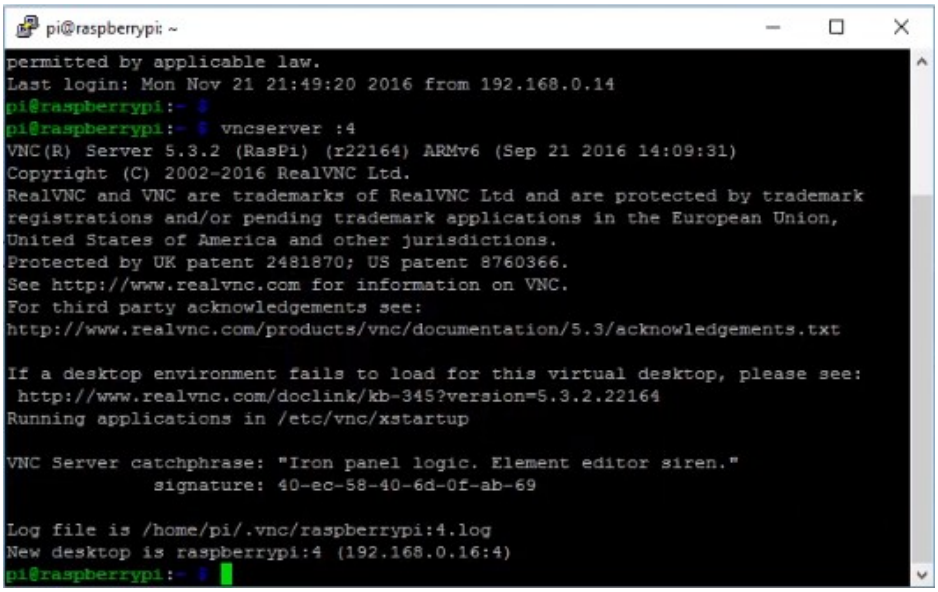


Figure 3-36: Setup a VNC Server

Then, you open the VNC Viewer and type 192.168.0.16:4 in the upper tab as shown in (Fig.37) below.

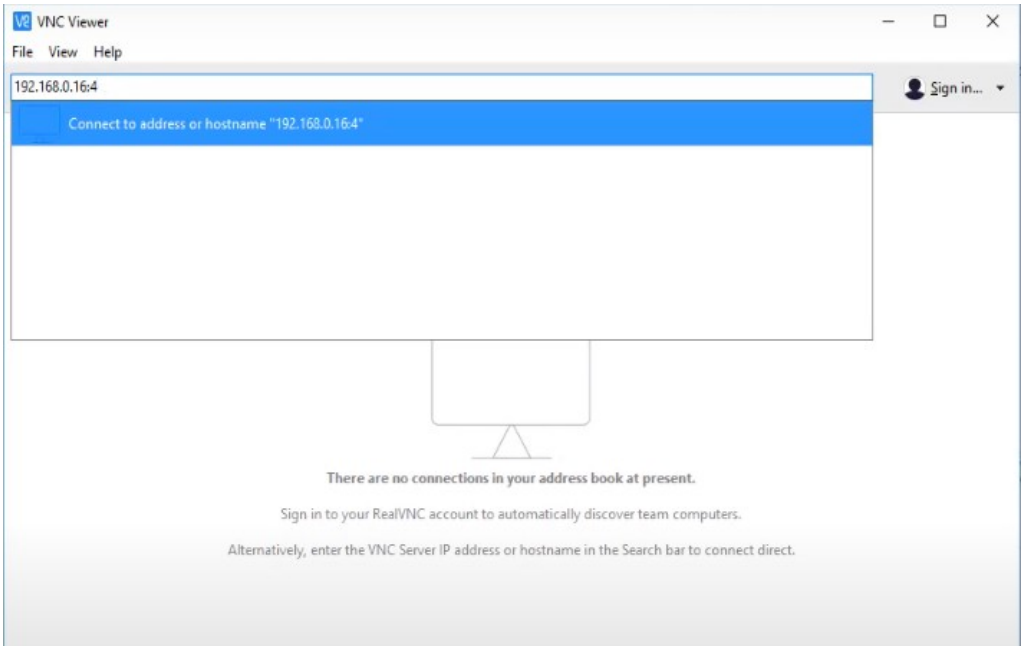


Figure 3-37: Connect to address or hostname

Then, click **Enter** key, a new window appears for authentication shown in (Fig.38) below.

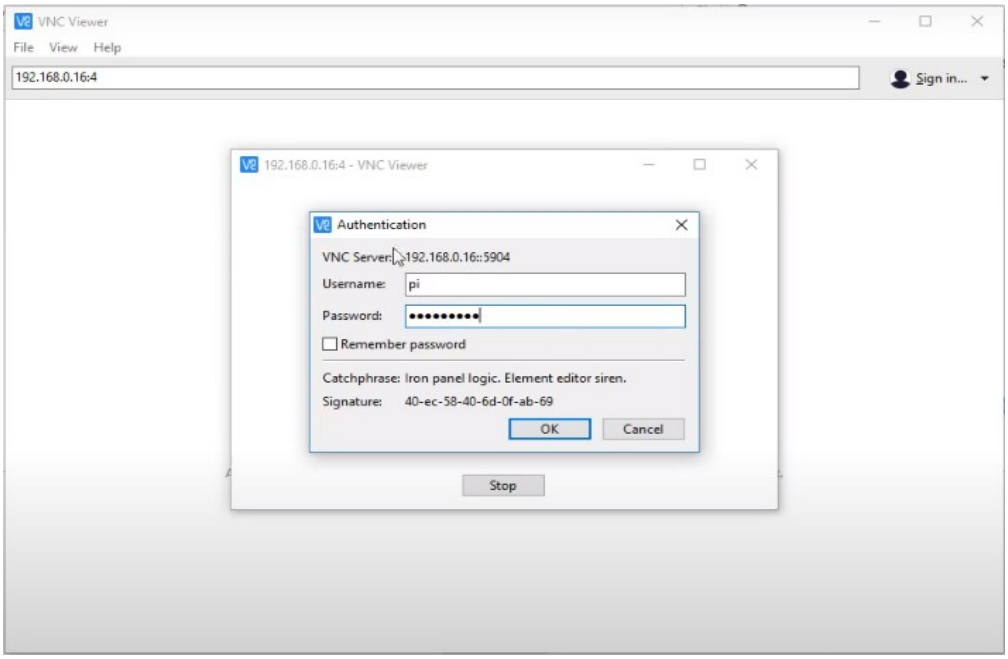


Figure 3-38: Authentication

Type the username and password of the Raspberry pi for authentication and click OK, then a new window appears to access the Raspberry pi resources as shown in (Fig.39) below.

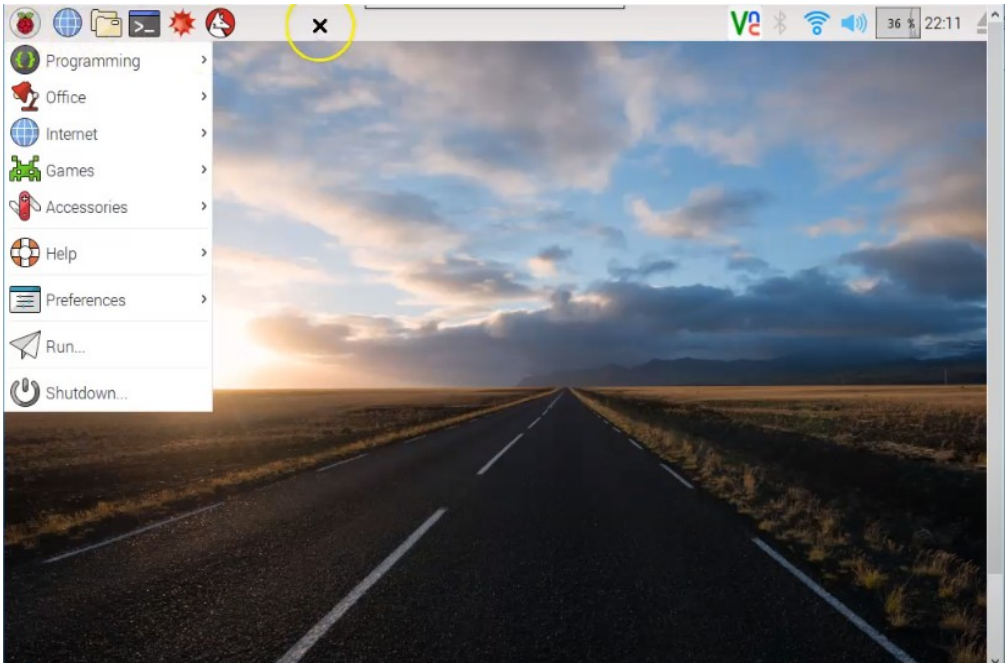


Figure 3-39: Access Raspberry pi

3.6 Launch Python Script on Startup[34]

We will need the project run once Raspberry Pi starts up, this will show you how to setup your Raspberry Pi to automatically launch a Python script upon startup.

Step 1: Make a Launcher Script

Python script is called: **Main.py** and lives in a directory called **Main** that is in the root directory as shown in (Fig.40) below.

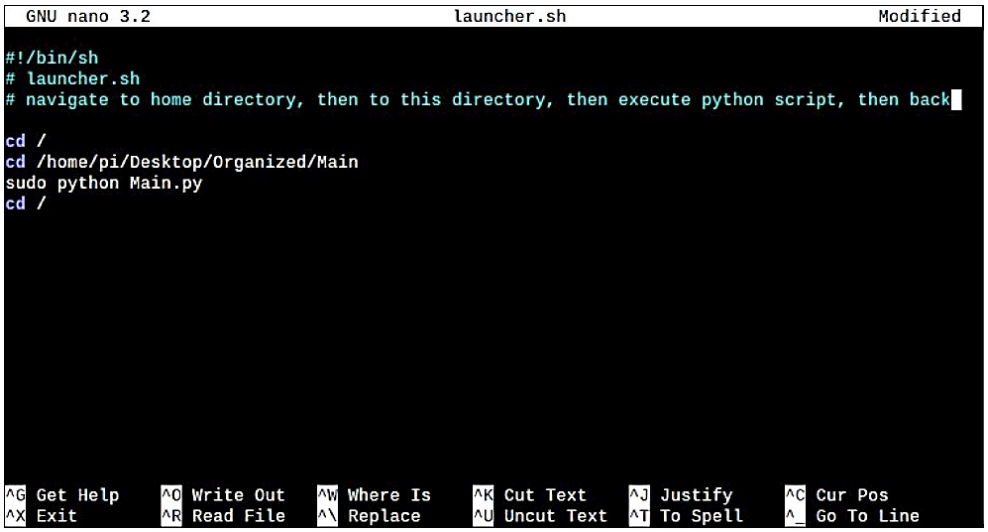


Figure 3-40: Make a Launcher Script

We will use the Linux crontab to run the Python script. Solution is to make a shell script, which always navigates to the proper directory and will launch my **Main.py** Python script. Let's create the shell script!

I'm using SSH to access to Raspberry Pi. My IP address for the SD card for this is 10.0.1.68 Open the Terminal window and on the command line, type:

```
ssh pi@10.0.1.68
```

If you are running directly hooked into the monitor, you can skip this step. Type:

```
cd Main
```

then:

```
nano launcher.sh
```

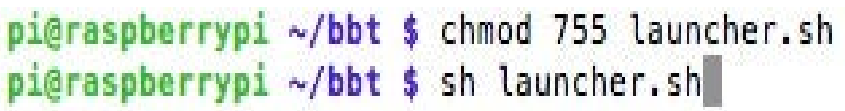
Will launch your editor, type in this script:

```
#!/bin/sh
# launcher.sh
# navigate to home directory , then to this directory , then execute python script , then back home.
cd /
cd home/pi/Desktop/Organized/Main
sudo python Main.py
cd /
```

Ctrl-X, Yes, Return to save.

This script will do is to navigate to the root directory, then to the **Organized** directory, launch the Python script and then return to the root directory.

Step 2: Make It Executable



We need to make the launcher script an executable, which we do with this command:

```
chmod 755 launcher.sh
```

Now test it, by typing in:

```
sh launcher.sh
```

This should run your Python code.

Step 3: Add Logs Directory

We will get to use crontab in a minute, but first we need to make a directory for the any errors in crontab to go. Navigate back to your home directory:

```
cd
```

Create a logs directory:

```
mkdir logs
```

Step 4: Add to Your Crontab

crontab is a background (daemon) process that lets you execute scripts at specific times. It's essential to Python and Raspberry Pi. The details are confusing, as is often the case with Linux. Type:

```
sudo crontab -e
```

This will brings up a crontab window as shown in (Fig.41) below.

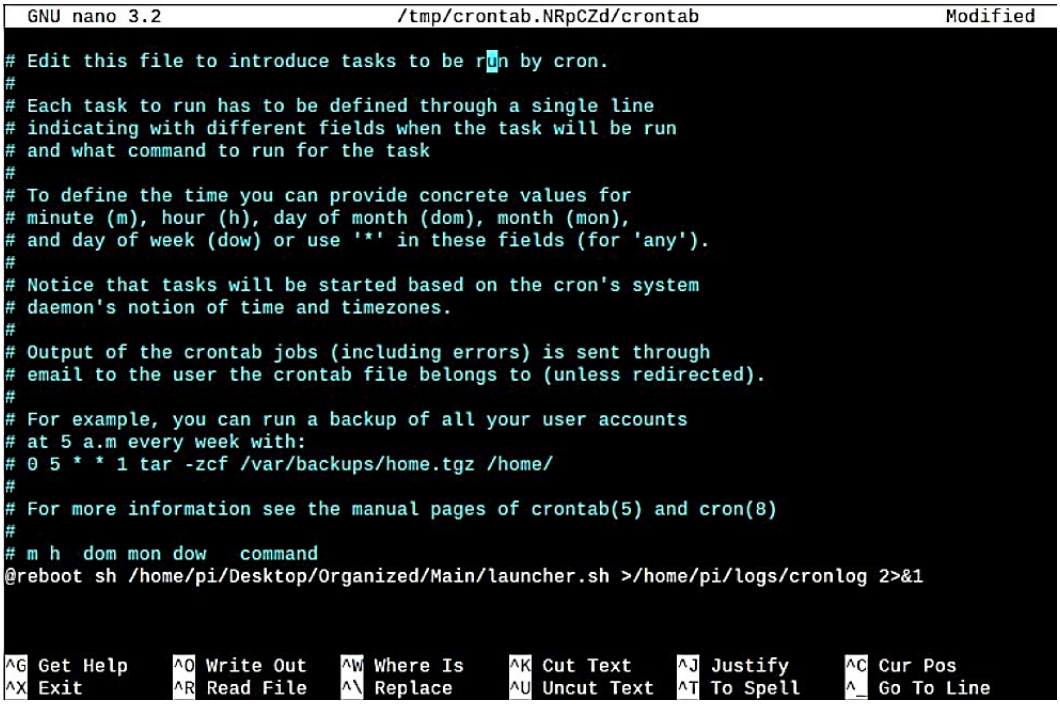


Figure 3-41: Crontab window

Now, enter the line:

```
@reboot sh /home/pi/Desktop/Organized/Main/launcher.sh  >/home/pi/logs/cronlog 2>&1
```

This does is rather than executing the launcher script at a specific time, it will execute it once upon startup.

Step 5: Reboot and See If It Works

Unplug the power or just type in:

```
sudo reboot
```

Wait for startup and see if your script automatically launches. If it doesn't work, check out the log file. This will show you any errors that you might have.

```
cd logs
cat cronlog
```

3.7 Raspberry Pi Camera Module[33]

If you've ever wanted to build something that can see then Raspberry Pi's optional Camera Module is a great starting place. A small square circuit board with a thin ribbon cable, the Camera Module connects to the Camera Serial Interface (CSI) port on your Raspberry Pi and provides high-resolution still images and moving video signals which can be used as is or integrated into your own programs.

Camera Types

There are two versions of the Raspberry Pi Camera Module available: the normal version and the 'NoIR' version. You can easily tell the difference: the normal version has a green circuit board, while the NoIR version has a black circuit board. If you want to take normal pictures and video in well-lit environments, you should always use the normal version for best image quality. The NoIR version – so called because it has no infrared, or IR, filter – is designed for use with infrared light sources to take pictures and video in total darkness. If you're building a nest box, security camera, or other project involving night vision, you want the NoIR version – but remember to buy an infrared light source at the same time!

At the time of writing, the current version of the Raspberry Pi Camera Module, known as the 'v2' module or 'Version 2.1', is based on a high-quality Sony IMX219 image sensor – the same type of sensor you might find on the back of your smartphone or tablet. This is an 8 megapixel sensor, which means it can take pictures with up to 8 million pixels in them. It does this by capturing images up to 3280 pixels wide by 2464 tall: multiply those two numbers together and you get a total of just over 8 million individual pixels!

As well as still images, the Camera Module can capture video footage at Full HD resolution – the same resolution as most TVs – at a rate of 30 frames per second (30 fps). For smoother motion or even to create a slow-motion effect, the camera can be set to capture at a higher frame rate by lowering the resolution: 60fps for 720p video footage, and up to 90 fps for 480p – or 'VGA' resolution – footage.

Installing the Camera Module

Like any hardware add-on, the Camera Module should only be connected to or disconnected from Raspberry Pi when the power is off and the power cable unplugged. If your Raspberry Pi is on, choose Shutdown from the raspberry menu, wait for it to power off, and unplug it. Unpack your Camera Module: you'll find a small circuit board, which is the Camera Module itself, and a flat ribbon cable. In most cases, the ribbon cable will already be connected to the Camera Module; if it isn't, turn your Module upside-down so the camera lens is on the bottom and look for a flat plastic

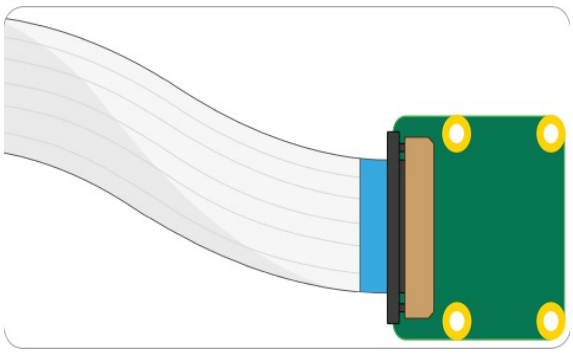


Figure 3-42: The Raspberry pi camera

connector. Carefully hook your fingernails around the sticking-out edges and pull outwards until the connector pulls part-way out. Slide the ribbon cable, with the silver edges downwards and the blue plastic facing upwards, under the flap you just pulled out, then push the flap gently back into place with a click as shown in (Fig.42) below; it doesn't matter which end of the cable you use. If the cable is installed properly, it will be straight and won't come out if you give it a gentle tug; if not, pull the flap out and try again.

Install the other end of the cable the same way. Find the Camera (or CSI) port on Raspberry Pi and pull the flap gently upwards. If your Raspberry Pi is installed in a case, you might find it easier to remove it first. With Raspberry Pi positioned so the HDMI port is facing you, slide the ribbon cable in so the silver edges are to your left and the blue plastic to your right, then gently push the flap back into place as shown in (Fig.43) below. If the cable is installed properly, it'll be straight and won't come out if you give it a gentle tug; if not, pull the flap out and try again.

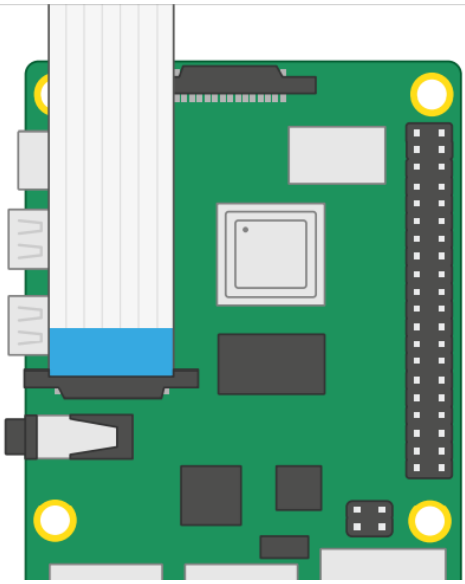


Figure 3-43: Install the ribbon cable inside the camera module interface

The Camera Module comes with a small piece of blue plastic covering the lens, in order to protect it from scratches during manufacturing, shipping, and installation. Find the small flap of plastic and pull it gently off the lens to get the camera ready for use.

ADJUSTING FOCUS

The Raspberry Pi Camera Module is usually supplied with a small plastic wheel. This is designed for adjusting the focus of the lens. While the factory-set focus is usually perfect, if you're using your camera for very close-up work you can slide the wheel over the lens and gently twist it to adjust the focus manually.

Connect the power supply back to Raspberry Pi and let it load Raspbian. Before you can use the camera, you'll need to tell Raspberry Pi it has one connected: click the raspberry icon to load the menu, choose the Preferences category, and click Raspberry Pi Configuration. When the tool has loaded, click the Interfaces tab, find the Camera entry in the list, and click on the round radio button to the left of 'Enabled' to switch it on as shown in (Fig.44) below. Click OK, and the tool will prompt you to reboot your Raspberry Pi. Do so and your camera will be ready to use.

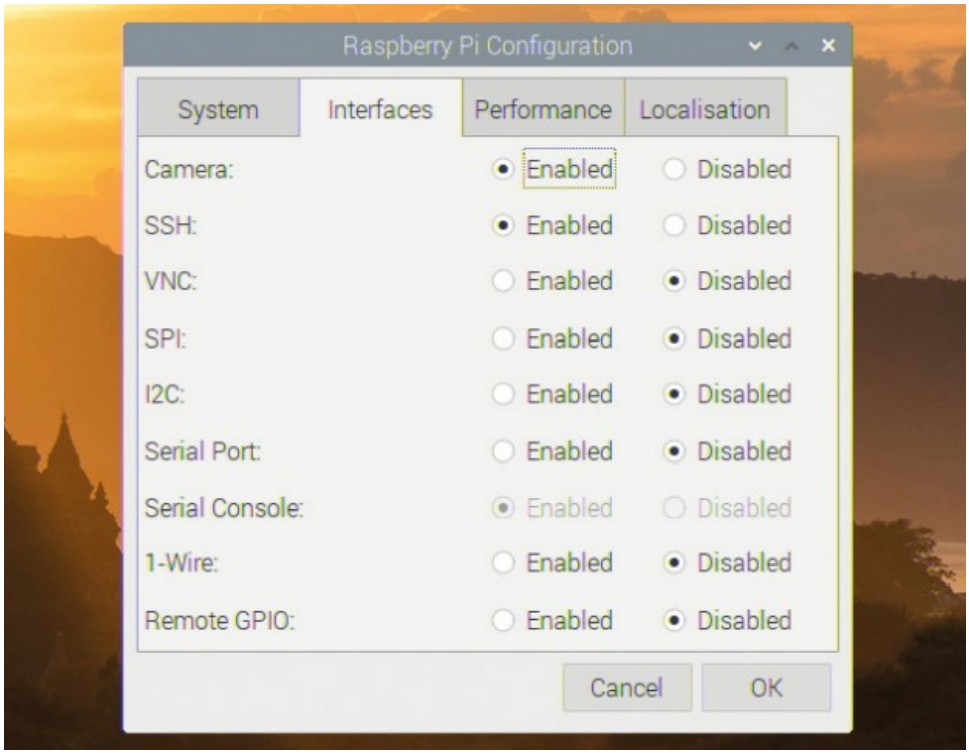


Figure 3-44: Enable camera interface

Testing the Camera Module

To confirm that your Camera Module is properly installed, and that you've enabled the interface in the Raspberry Pi Configuration Tool, you can use the **raspistill** tool. This, along with **raspivid** for videos, is designed to capture images from the camera using Raspberry Pi's command-line interface (CLI).

Unlike the programs you've been using so far, you won't find raspistill in the menu. Instead, click on the raspberry icon to load the menu, choose the Accessories category, and click on Terminal. A black window with green and blue writing in it will appear as shown in (Fig.45) below: this is the terminal, which allows you to access the command-line interface.

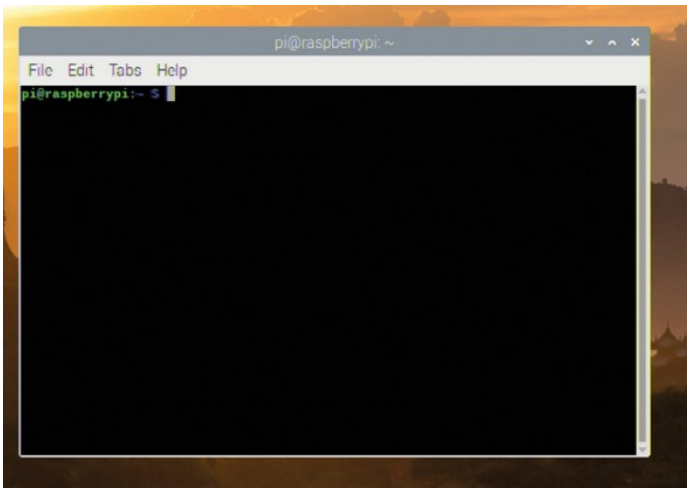


Figure 3-45: Terminal

To test the camera, type the following into the Terminal:

```
raspistill -o test.jpg
```

As soon as you hit **ENTER**, you'll see a large picture of what the camera sees appear on screen. This is called the live preview as shown in (Fig.46) below and, unless you tell raspistill otherwise, it will last for 5 seconds. After those 5 seconds are up, the camera will capture a single still picture and save it in your home folder under the name **test.jpg**. If you want to capture another, type the same command again – but make sure to change the output file name, after the **-o**, or you'll save over the top of your first picture!

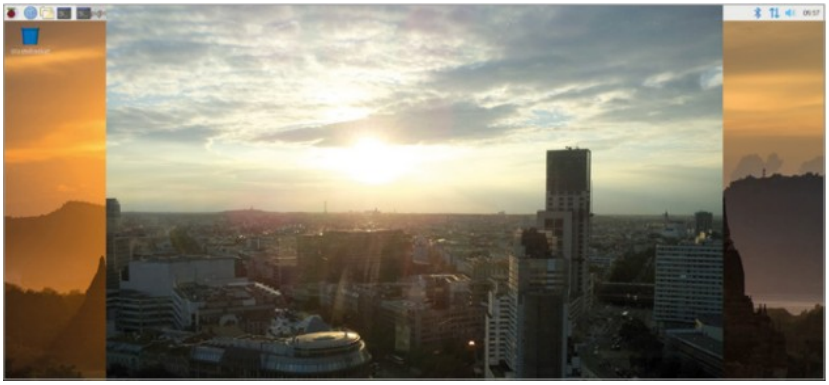


Figure 46: Live preview

If the live preview was upside-down, you'll need to tell raspistill that the camera is rotated. The Camera Module is designed to have the ribbon cable coming out of the bottom edge; if it's coming out of the sides or the top, as with some third-party camera mount accessories, you can rotate the image by 90, 180, or 270 degrees using the **-rot** switch. For a camera mounted with the cable coming out of the top, simply use the following command:

```
raspistill -rot 180 -o test.jpg
```

To see your picture, open the File Manager from the Accessories category of the raspberry menu: the image you've taken, called test.jpg, will be in your home/pi folder. Find it in the list of files, then double-click it to load it in an image viewer as shown in (Fig.47) below. You can also attach the image to emails, upload it to websites via the browser, or drag it to an external storage device.

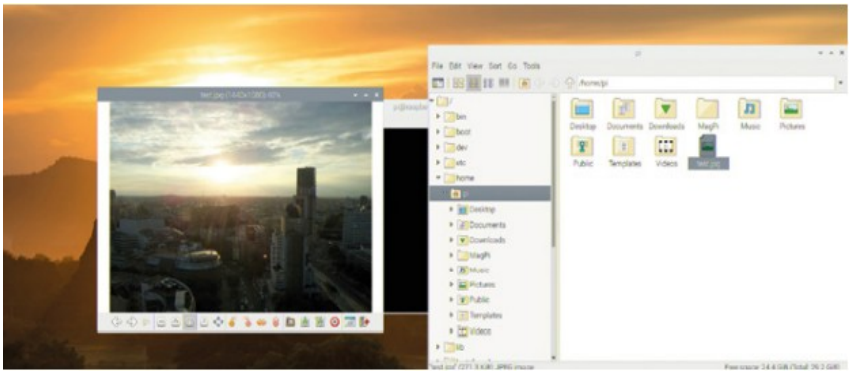


Figure 3-47: The picture taken after 5 seconds

3.8 Voice Issue

We find out that the AVI jack port can only output the voice, but cannot work as input voice. To solve that problem we will use USB sound card and headphone splitter shown in (Fig.55) below.

Figure 3-48: USB sound card and headphone splitter

We will connect the USB end of the USB sound card to one of the USB ports of the Raspberry Pi, then connect the two male headphone and microphone of the splitter to the two female headphone and microphone of the USB sound card, and eventually connect the female of the splitter to your headphone.

3.9 User interaction

Users interact with the options the project provides such as text recognition, object detection, money recognition, face detection or any new utility we need to add. We can achieve that via:

- Using buttons.
- Using voice input (commands).

Using buttons

Using the Raspberry pi GPIO, we will connect buttons to them, each button to run certain command mode.

Voice input (commands)

You can use the mic of the headphone to send commands to the Raspberry pi which receive and process them and then execute the command you send.

3.10 Project installation

In project installation, we talk about how to install the product on the user. There are multiple options for project installation that got out interest before choosing one , here are some of them:

Option 1: Spy Glasses

Using a spy wireless camera installed on the center of the glass, and the Raspberry pi and power bank in the pocket of user. Coming out from the Raspberry pi, the headphone. This option was our first intention but was out of our reach due to restrictions on imported products in Egypt.

Figure 3-49: Spy camera

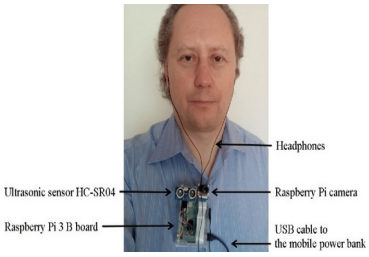
Option 2: Laptop Webcam

Using a wired camera laptop (connected to the USB port of the Raspberry pi via USB cable) installed in the middle of the glass and the Raspberry pi and power bank in the pocket of user. Coming out from the Raspberry pi, the headphone.

Figure 3-50: Laptop camera

Option 3: Pocket Camera

Suppressing the Raspberry pi with Raspbian camera using a chain or installing the total product on the pocket.



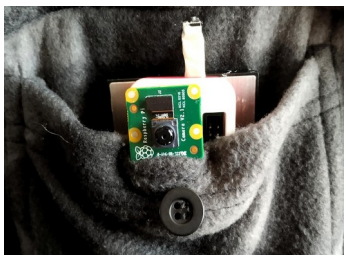


Figure 3-51: Pocket camera

Option 4

Using Raspbian camera connected to camera module interface via ribbon cable, but in that option we need extension to the ribbon cable to enable us to install the Raspbian camera on the side arm of the glass and the Raspberry pi and power bank in the pocket of user, coming out from the Raspberry pi the headphone.

Option 5

Using the Raspberry Pi as a handwatch, with earphones connected under the user shirt. The user should be pointing the watch in a certain direction to take the pictures. This is the option we chose to implement. The watch is a 3D printed case, shown in Figure 3-52, which have the hardware inside with the camera put in a slider above the cover.

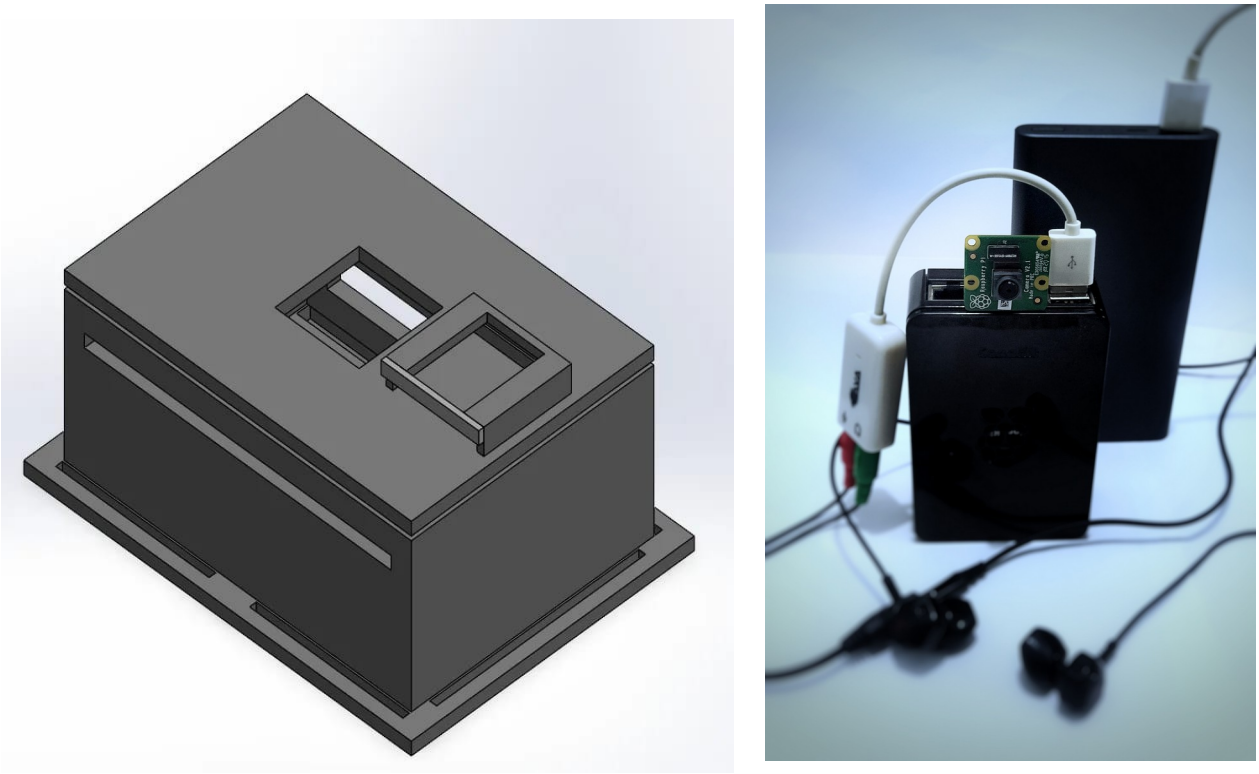


Figure 3-52: 3D Raspberry Pi case

Chapter 4: Voice Recognition

4.1 Introduction[40]

Alternatively referred to as speech recognition, voice recognition is a computer software program or hardware device with the ability to decode the human voice. Voice recognition is commonly used to operate a device, perform commands, or write without having to use a keyboard, mouse, or press any buttons. Today, this is done on a computer with ASR (automatic speech recognition) software programs. Many ASR programs require the user to "train" the ASR program to recognize their voice so that it can more accurately convert the speech to text. For example, you could say "open Internet" and the computer would open the Internet browser.

The first ASR device was used in 1952 and recognized single digits spoken by a user (it was not computer driven). Today, ASR programs are used in many industries, including healthcare, military (e.g., F-16 fighter jets), telecommunications, and personal computing (i.e., hands-free computing).

4.2 Voice Recognition System Components[45]

Speech recognition is an alternative to traditional methods of interacting with a computer, such as textual input through a keyboard. An effective system can replace, or reduce the reliability on, standard keyboard and mouse input. A speech recognition system consists of:

- A microphone, for the person to speak into.
- Speech recognition software.
- A computer to take and interpret the speech.
- A good quality soundcard for input and/or output.

4.3 Voice Recognition Usage Examples[40]

As voice recognition improves, it is being implemented in more places and it's very likely you have already used it.

- **Automated phone systems** - Many companies today use phone systems that help direct the caller to the correct department. If you have ever been asked something like "Say or press number 2 for support" and you say "two," you used voice recognition.
- **Google Voice** - Google voice is a service that allows you to search and ask questions on your computer, tablet, and phone.

- **Digital assistant** - Amazon Echo, Apple's Siri, and Google Assistant use voice recognition to interact with digital assistants that helps answer questions.
- **Car Bluetooth** - For cars with Bluetooth or Handsfree phone pairing, you can use voice recognition to make commands, such as "call my wife" to make calls without taking your eyes off the road.

4.4 Types of Voice Recognition Systems[40]

Automatic speech recognition is one example of voice recognition. Below are other examples of voice recognition systems:

- **Speaker dependent system** - The voice recognition requires training before it can be used, which requires you to read several words and phrases.
- **Speaker independent system** - The voice recognition software recognizes most users' voices with no training.
- **Discrete speech recognition** - The user must pause between each word so that the speech recognition can identify each separate word.
- **Continuous speech recognition** - The voice recognition can understand a normal rate of speaking.
- **Natural language** - The speech recognition not only can understand the voice, but can also return answers to questions or other queries that are being asked.

4.5 How It Works[41][43]

Speech recognition software works by breaking down the audio of a speech recording into individual sounds, analyzing each sound, using algorithms to find the most probable word fit in that language, and transcribing those sounds into text.

Speech recognition software uses natural language processing (NLP) and deep learning neural networks. This means that the software breaks the speech down into bits it can interpret, converts it into a digital format, and analyzes the pieces of content.

From there, the software makes determinations based on programming and speech patterns, making hypotheses about what the user is actually saying. After determining what the users most likely said, the software transcribes the conversation into text.

This all sounds simple enough, but the advances in technology mean these multiple, intricate processes are happening at lightning speed. Machines can actually transcribe human speech more accurately, correctly, and quickly than humans can.

Voice recognition systems have to go through certain steps to figure out what we're saying. When your device's microphone picks up your audio, it's converted into an electrical current which travels down to the Analog to Digital Converter (ADC). As the name suggests, the ADC converts the electric current (the analog signal) into a digital binary signal.

As the current flows to the ADC, it takes samples of the current and deciphers its voltage at certain points in time. The voltage at a given point in time is called a sample. Each sample is only several thousandths of a second long. Based on the sample's voltage, the ADC will assign a series of eight binary digits (one byte of data).

In order for the device to better understand the speaker, the audio needs to be processed to improve clarity. The device is sometimes tasked with deciphering speech in a noisy environment; thus, certain filters are placed on the audio to help eliminate background noise. For some voice recognition systems, frequencies that are higher and lower than the human's hearing range are filtered out.

The system doesn't only get rid of unwanted frequencies; certain frequencies in the audio are also emphasized so that the computer can better recognize the voice and separate it from background noise. Some voice recognition systems actually split the audio up into several discrete frequencies.

Other aspects, such as the speed and volume of the audio, are adjusted to better match the references audio samples that the voice recognition system uses to compare. These filtration and denoising processes really help improve the overall accuracy. Then, the Voice Recognition System Starts Making Words.

There are two popular ways that voice recognition systems analyze speech. One is called the hidden Markov model, and the other method is through neural networks (that we will use). We'll understand the second method in more details.

4.6 Voice Preprocessing and RNN Recognition[47]

It works by simply feeding sound recordings into a neural network and train it to produce text. A major problem in speech recognition is the speech speed which can vary in many situations. For example, a person could say "hello" very quickly while another says "heeeeellllooooo" very slowly, This result in different sound files. But both files should be recognized as the same text "hello". The automatic aligning of audio files which varies in lengths into a fixed-length piece of text is difficult. To work around this, some additional processing steps are required.

4.6.1 Turning sounds into bits

The first step is obvious, we need to feed sound waves into a computer. Like how images is translated into array of numbers, we need to convert sound waves into the form of numbers (bits). Look at the Figure below showing the waveform of saying hello.

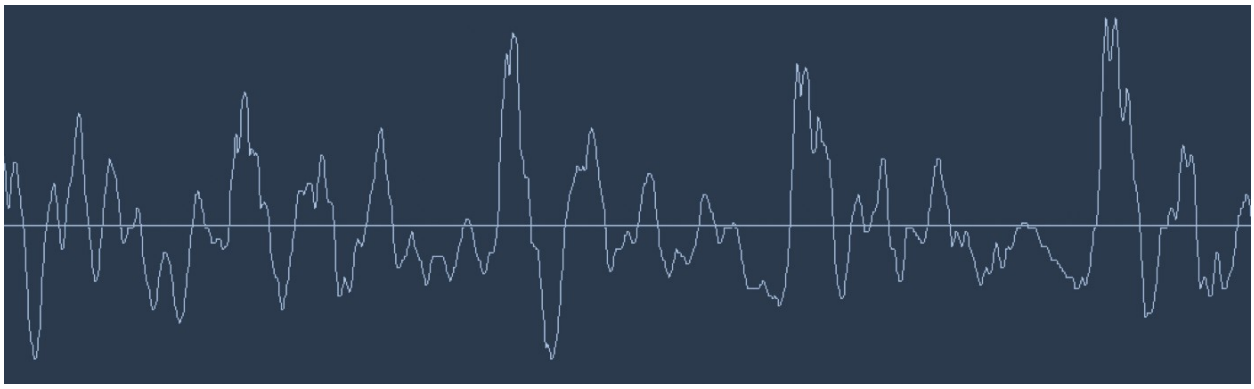
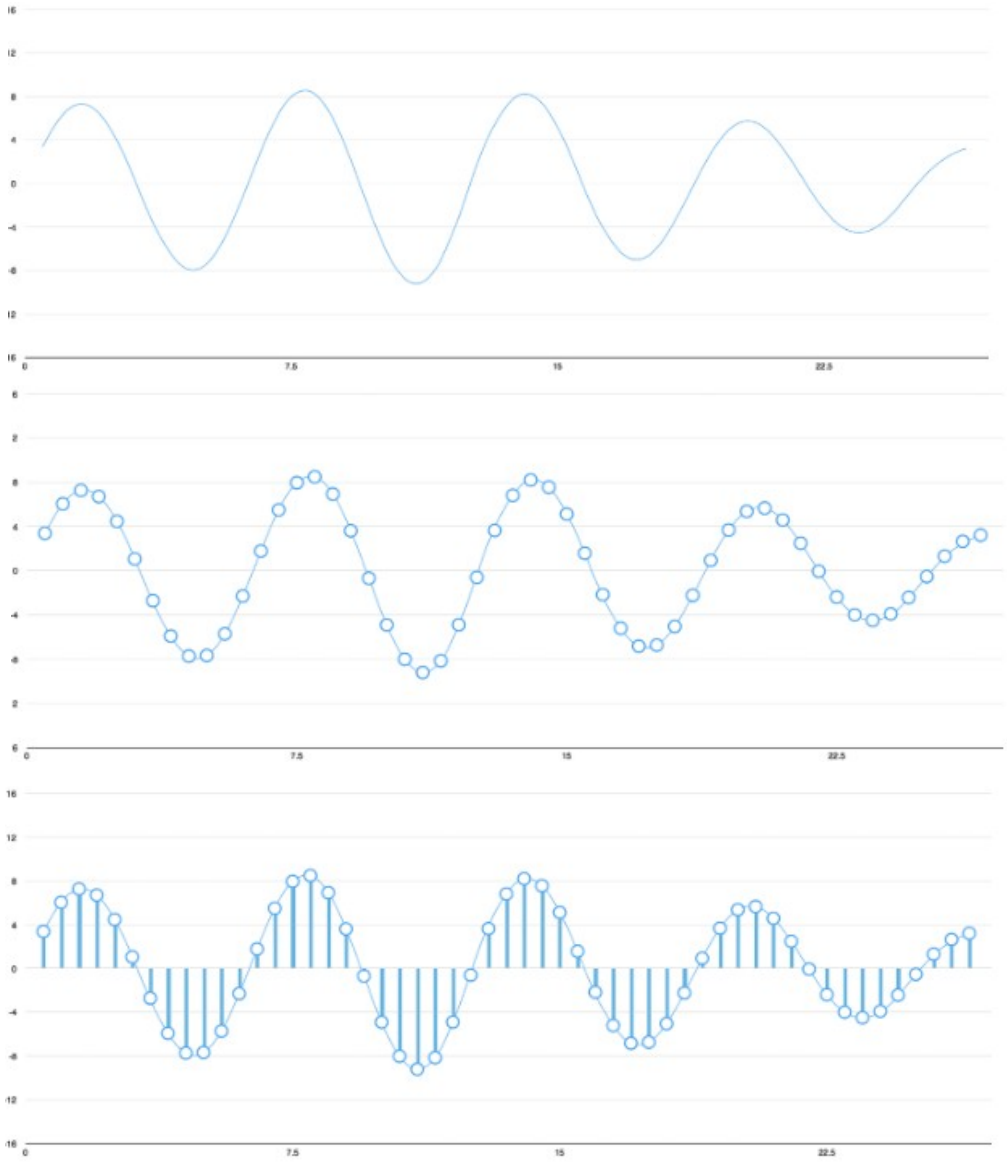


Figure 7-1: Waveform of saying "Hello"

Sound waves are one-dimensional. At any given time, there is only a single value based on the height of the wave. To turn sound wave into numbers, we need to record the height of the wave at equal-spaced points (sampling). Look at the figures below showing the sequence of sampling operation.



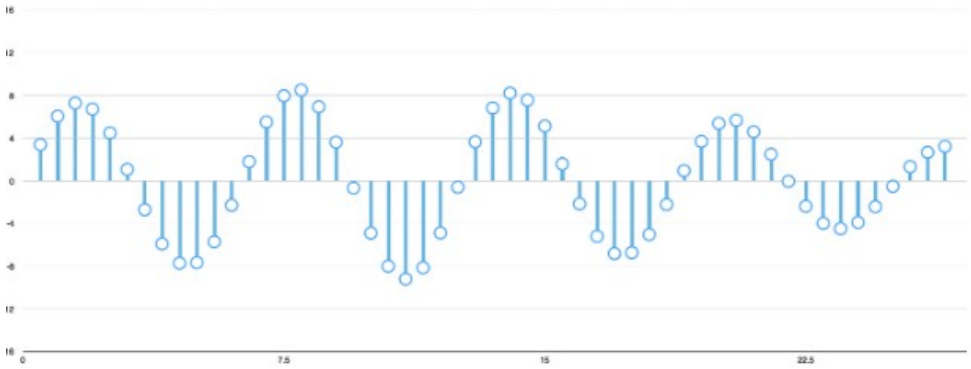


Figure 7-2: Sampling a sound wave

For speech recognition, a sampling rate of 16Khz (16000 samples per second) is enough to cover the frequency of human speech. In sampling you might think that there is data lost in the process but thanks to Nyquist theorem, we can use math to perfectly reconstruct the original sound wave from the spaced-out samples. Based on the theorem as long as we sample at least twice as fast as the highest frequency we want to record there will be no data loss. This also means sampling at higher rates than that it won't increase audio quality.

4.6.2 Pre-processing our sampled sound data

We now have an array of numbers; each number represents a sound wave amplitude at 1/16000th of a second intervals. Feeding these numbers to a neural network to recognize speech patterns by processing these samples directly is hard. Instead, we do some pre-processing on the audio data to make it easier.

We start by grouping our sampled audio into 20-millisecond-long chunks. Plotting these numbers as a simple line graph gives an approximation of the original sound wave for that 20-millisecond period of time look at the figure below:

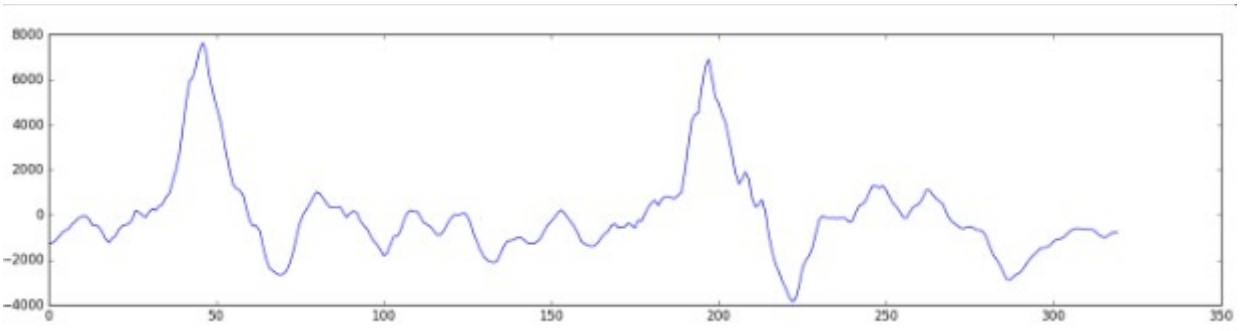


Figure 7-3: Simple line graph of 20millisecond period

This recording is only *1/50th of a second long*. But even this short recording is a complex mish-mash of different frequencies of sound. There's some low sounds, some mid-range sounds, and even some high-pitched sounds sprinkled in. But taken all together, these different frequencies mix together to make up the complex sound of human speech.

To make this data easier for a neural network to process, we are going to break apart this complex sound wave into it's component parts. We'll break out the low-pitched parts, the next-lowest-pitched-parts, and so on. Then by adding up how much energy is in each of those frequency bands (from low to high), we create a *fingerprint* of sorts for this audio snippet.

Imagine you had a recording of someone playing a C Major chord on a piano. That sound is the combination of three musical notes— C, E and G — all mixed together into one complex sound. We want to break apart that complex sound into the individual notes to discover that they were C, E and G. This is the exact same idea.

We do this using a mathematic operation called a *Fourier transform*. It breaks apart the complex sound wave into the simple sound waves that make it up. Once we have those individual sound waves, we add up how much energy is contained in each one.

The end result is a score of how important each frequency range is, from low pitch (i.e., bass notes) to high pitch. Each number will represents how much energy was in each 50hz band of our 20-millisecond audio clip. By looking at the figure below showing our 20-millisecond sound snippet has a lot of low-frequency energy and not much energy in the higher frequencies (male voices).

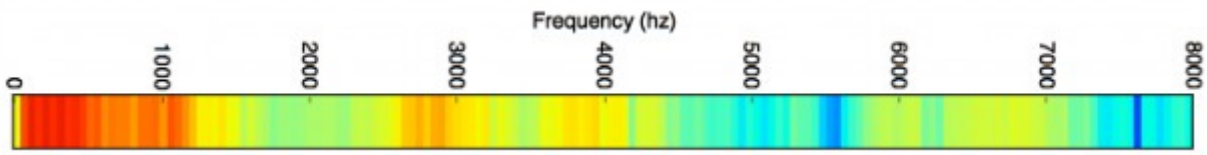


Figure 7-4: 20milisecond sound snippet

If the process is repeated on every 20millisecond chunk of audio, we end up with a spectrogram (each column from left-to-right is one 20ms chunk):

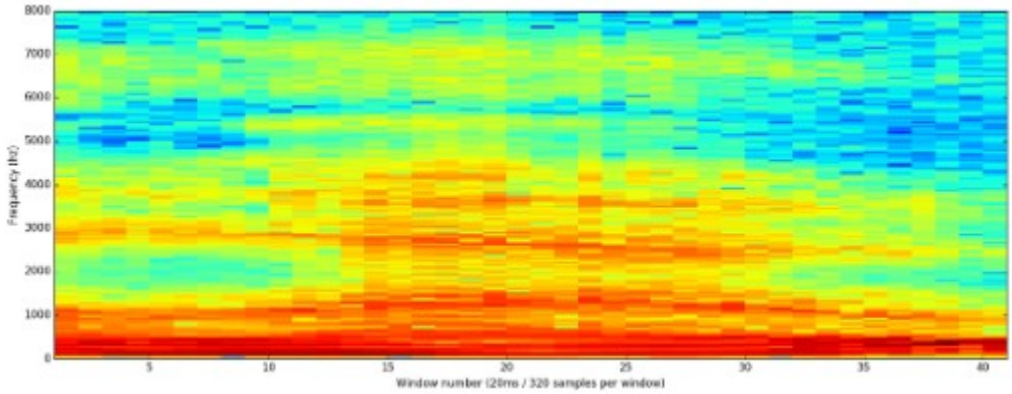


Figure 7-5: Full spectrogram of the "Hello" sound clip

A neural network can find patterns in this kind of data more easily than raw sound waves. So this is the data representation we'll actually feed into our neural network.

4.6.3 Recognizing characters from short sounds

The input to the neural network will be 20millisecond audio chunks. For each little audio slice, it will try to figure out the letter that corresponds the sound currently being spoken see the figure below showing a recurrent neural network taking an audio slice.

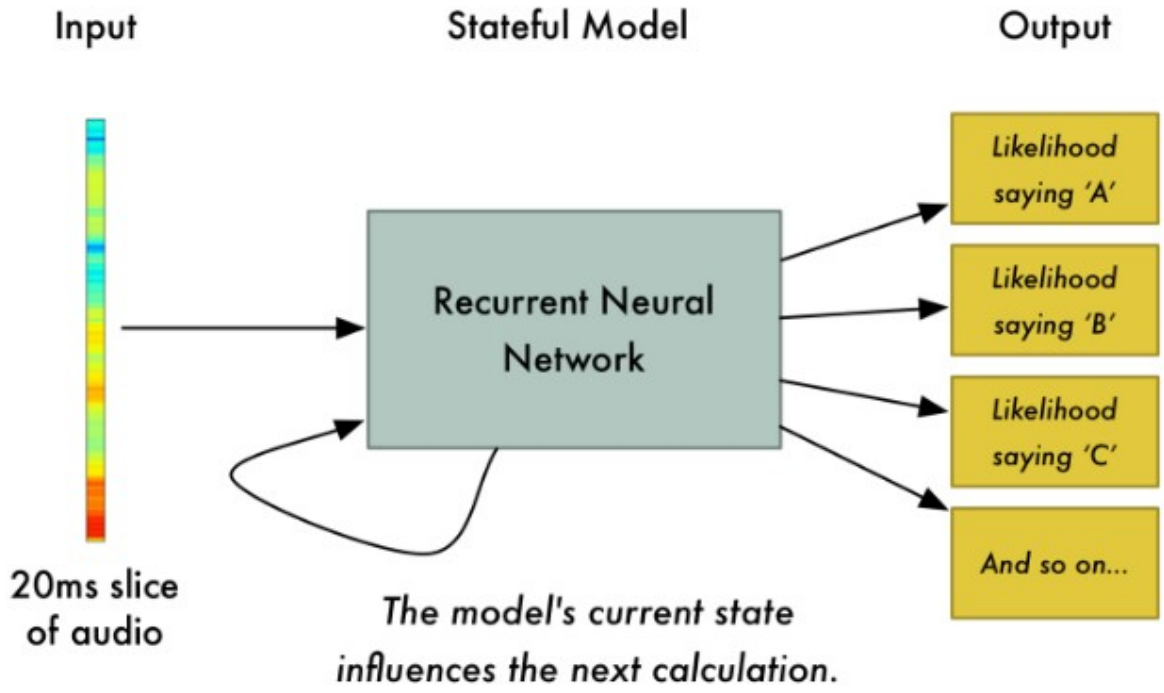


Figure 7-6: Recurrent Neural Network

We'll use a recurrent neural network. This type of neural networks has a memory that influences future predictions. Because each letter predicted should affect the likehood of the next letter. For example, if we have said "HEL" so far, it's very likely we will say "LO" next to finish out the word "Hello". It's much less likely that we will say something unpronounceable next like "XYZ". So having memory of previous predictions helps the neural network make more accurate predictions going forward.

After we run our entire audio clip through the neural network (one chunk at a time), we'll end up with a mapping of each audio chunk to the letters most likely spoken during that chunk. Here's what that mapping looks like for saying "Hello" look at the figure below:

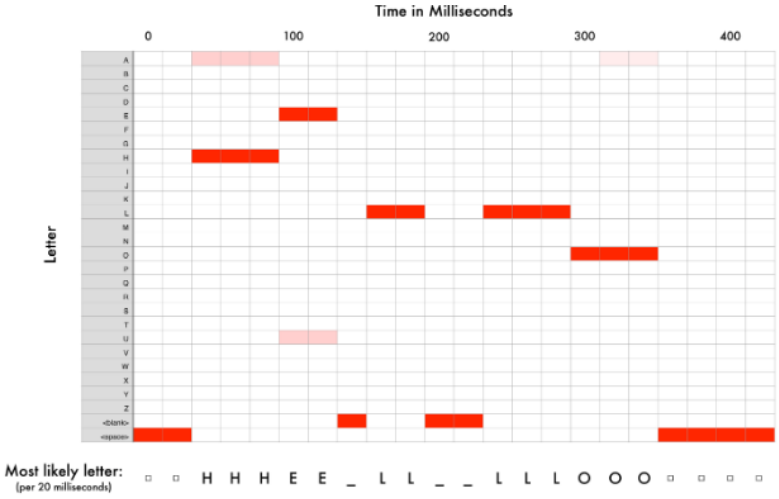


Figure 7-7: Mapping of saying "Hello"

Our neural net is predicting that one likely thing I said was “HHHEE_LL_LLLOOO”. But it also thinks that it was possible that I said “HHHUU_LL_LLLOOO” or even “AAAUU_LL_LLLOOO”.

We have some steps we follow to clean up this output. First, we’ll replace any repeated characters a single character:

- HHHEE_LL_LLLOOO becomes HE_L_LO
- HHHUU_LL_LLLOOO becomes HU_L_LO
- AAAUU_LL_LLLOOO becomes AU_L_LO

Then we’ll remove any blanks:

- HE_L_LO becomes HELLO
- HU_L_LO becomes HULLO
- AU_L_LO becomes AULLO

That leaves us with three possible transcriptions — “Hello”, “Hullo” and “Aullo”. If you say them out loud, all of these sound similar to “Hello”. Because it’s predicting one character at a time, the neural network will come up with these very sounded-out transcriptions. For example, if you say “He would not go”, it might give one possible transcription as “He wud net go”.

The trick is to combine these pronunciation-based predictions with likelihood scores based on large database of written text (books, news articles, etc). You throw out transcriptions that seem the least likely to be real and keep the transcription that seems the most realistic.

Of our possible transcriptions “Hello”, “Hullo” and “Aullo”, obviously “Hello” will appear more frequently in a database of text (not to mention in our original audio-based training data) and thus is probably correct. So, we’ll pick “Hello” as our final transcription instead of the others.

4.7 Speech Recognition Tools[42]

If you don't want to build your speech recognition system, there are various open-source tools. Among them are:

- **CMU Sphinx** — A speaker-independent, continuous-speech recognition system developed at Carnegie Mellon University. CMU Sphinx Includes a group of products designed for different purposes. It supports many popular programming languages, such as C/C++, C#, Java, and Python;
- **HTK Toolkit** — A toolkit for working with Hidden Markov Models. Developed at Cambridge University by Machine Intelligence Laboratory, it is primarily used for speech recognition research. It's not fully open source. Supported programming languages are C and Python;
- **Kaldi** — This one is an open-source toolkit for speech recognition and signal processing. Supported programming languages are C++ and Python. Kaldi is our adopted recognizer.

4.8 Kaldi[44]

Kaldi is a toolkit for speech recognition written in C++ and licensed under the Apache License v2.0. Kaldi is intended for use by speech recognition researchers. According to legend, Kaldi was the Ethiopian goat herder who discovered the coffee plant.

Kaldi is an automatic speech recognition toolkit that supports linear transforms, MMI, boosted MMI (Maximum Mutual Information) and MCE (Minimum Classification Error) discriminative training, feature-space discriminative training, and deep neural networks.

Kaldi is similar in aims and scope to HTK. The goal is to have modern and flexible code, written in C++, that is easy to modify and extend. Important features include:

- Code-level integration with Finite State Transducers (FSTs)
- Extensive linear algebra support
- Extensible design
- Linux or Windows environment
- Open license
- Complete recipes

The goal of releasing complete recipes is an important aspect of Kaldi. Since the code is publicly available under a license that permits modifications and re-release.

- Kaldi code is thoroughly tested.
- Kaldi code is easy to understand.

- Kaldi code is easy to reuse and refactor.
- Kaldi recognizer is one of the classes used in vosk library

4.9 Vosk[46]

- Offline speech recognition API for Android, iOS, Raspberry Pi and servers with Python, Java, C# and Node.
- Vosk models are small (50 Mb) but provide continuous large vocabulary transcription, zero-latency response with streaming API, reconfigurable vocabulary and speaker identification.
- Vosk supplies speech recognition for chatbots, smart home appliances, virtual assistants. It can also create subtitles for movies, transcription for lectures and interviews.
- Vosk scales from small devices like Raspberry Pi or Android smartphone to big clusters.

Chapter 5: Object Detection

5.1 Introduction

In this chapter, we are going to talk about object detection. It is considered as a computer vision technique used for identifying and localizing objects in an image or video. This allows us to identify, count, and track the precise locations of the objects while accurately labeling them [14].

Object detection approaches can be broken down into machine learning based approaches and deep learning based approaches. The later is the most popular and common for making an object detection models [14].

There are many different deep learning-based algorithms used and the one we are going to use in our project is called You Only Look Once (***YOLO***) which is considered one of the fastest algorithms out there. But before we dive deep into this model, we need to learn more about other models to give us a general idea of why we used YOLO.

5.2 How Deep Learning Models Work

The basic structure of deep learning-based object detection models typically has 2 parts[14]:

- Encoder that takes an image as input, then this image goes through a series of blocks and layers for statistical features extraction to locate and label the objects.
- Decoder that takes the output from the encoder and predicts the bounding boxes and labels for each object.

The simplest decoder is a pure regressor. It is used to predict the location and size of each bounding box by providing the X and Y coordinates of the object, that represent the center of the object, and width and height of the object. But the problem with pure regressor is that the required number of objects must be specified ahead of time in each image[14].

The solution for this problem is a region proposal network which is an extension of regressor approach. Figure 5-1 shows a simple diagram of region proposal network. In this decoder, the model proposes regions of an image where it believes an object might reside. The pixels belonging to these regions are then fed into

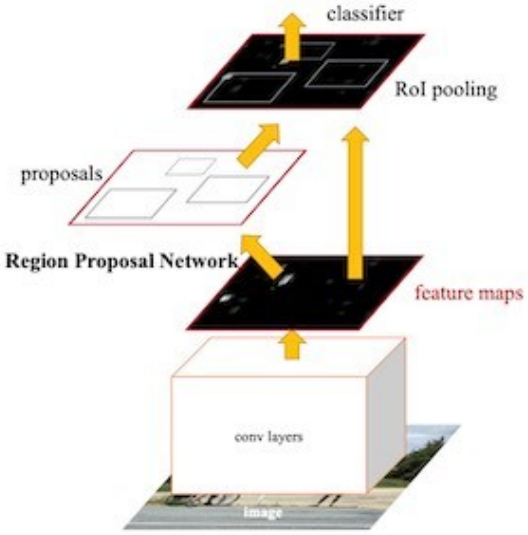


Figure 5-1: Simple diagram of region proposal network

a classification subnetwork to determine a label (or reject the proposal). It then runs the pixels containing those regions through a classification network. The benefit of this method is a more accurate, flexible model that can propose arbitrary numbers of regions that may contain a bounding box. The added accuracy, though, comes at the cost of computational efficiency[14].

5.3 Deep learning-based Model Architectures

R-CNN, Faster R-CNN, and Mask R-CNN

These some popular models that belong to the R-CNN family (an abbreviation for Region Convolutional Neural Network). These architectures use the region proposal network approach discussed earlier. These models have become more accurate and more computationally efficient over the years. They were developed by researchers at Facebook[14].

YOLO, MobileNet + SSD, SqueezeDet

These models belong to the Single Shot Detector (SSD) family. SSDs seek a middle ground. This means that rather than using a subnetwork to propose regions, SSDs rely on a set of predetermined regions.

A grid of anchor points is laid over the input image, and at each anchor point, boxes of multiple shapes and sizes serve as regions, see Figure 5-2 below. For each box at each anchor point, the model outputs a prediction of whether an object exists within the region and modifications to the box's location and size are done to make it fit the object more closely.

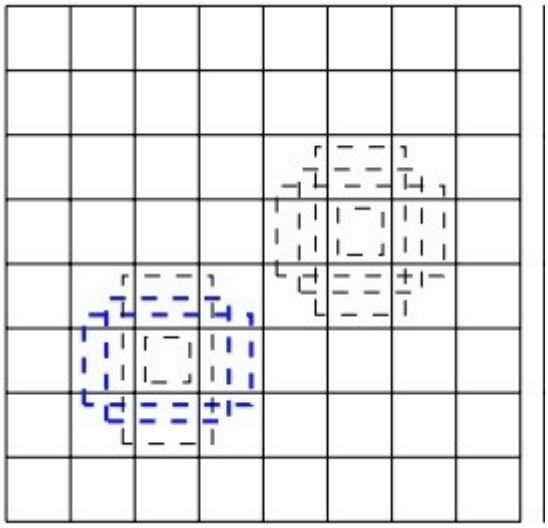


Figure 5-2: Anchor points boxes

Because there are multiple boxes at each anchor point and anchor points may be close together, SSDs produce many potential detections that overlap. Post-processing must be applied to SSD outputs in order to prune away most of these predictions and pick the best one. The most popular post-processing technique is known as non-maximum suppression. SSDs make great choices for models destined for mobile or embedded devices[14].

5.4 Adopted Algorithm :YOLO[15]

In this project the YOLO algorithm has been used. It uses CNN (Convolutional Neural Networks) to detect objects in real-time. It only requires a single forward propagation through the neural network to detect objects. This means that the predictions made in an image is done in a

single algorithm run. That's why it's named "You Only Look Once" (YOLO). YOLO algorithm has various variants: YOLO, YOLOv2, YOLOv3, YOLOv4, …etc.

5.4.1 Why YOLO?

1. Speed: this algorithm improves the speed of detection and can predict objects in real-time.
2. High accuracy: it's a predictive technique which gives an accurate result with minimal background errors.
3. Learning capabilities: it has excellent capabilities that enables it to learn representations of objects and apply them in object detection.

5.4.2 How YOLO works?

The following figure demonstrates how the YOLO algorithm works:

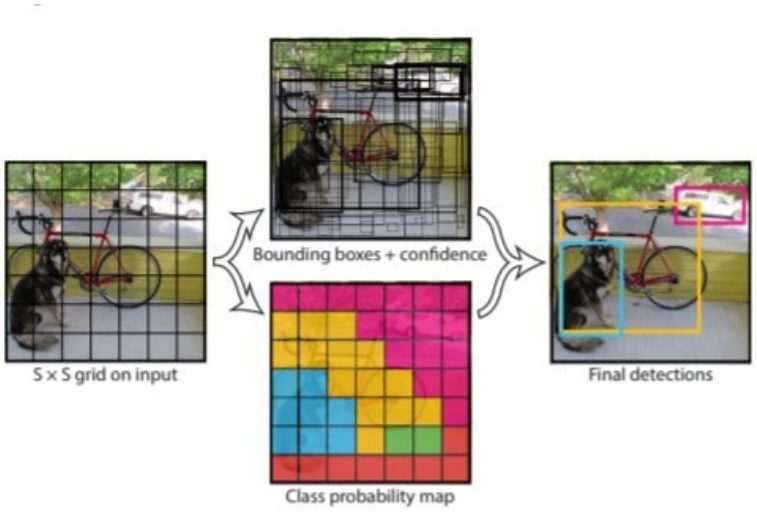


Figure 5-3: YOLO techniques overview

YOLO works using three techniques:

- Residual blocks.
- Bounding box regression.
- Intersection Over Union (IOU).

Residual Blocks

The image is divided into various SxS grids. All cells in the grid have equal dimension and each of these cells is responsible for detecting the object that appear within them. Figure 5-4 illustrates the process.

Figure 5-4: Dividing the image into SxS grid

Bounding Box Regression

A bounding box is an outline that highlights an object in an image. Every bounding box has the following attributes: Width (bw), Height (bh), Class (c), center (bx, by), probability (pc). See Figure 5-5 below.

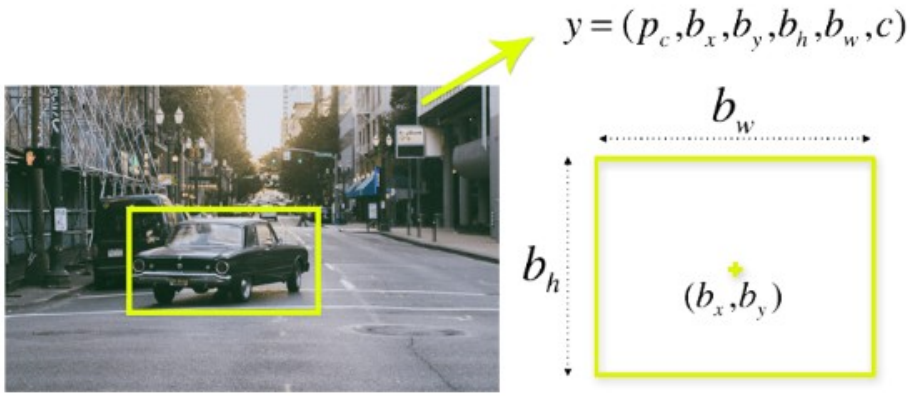


Figure 5-5: Bounding box representation

Intersection Over Union (IOU) and Non-maximum suppression (NMS)

The Intersection over Union (IoU) metric, also referred to as the Jaccard index, is essentially a method used usually to quantify the percent overlap between the ground truth BBox (Bounding Box) and the prediction BBox. However, in NMS, we find IoU between two predictions BBoxes instead. IoU in mathematical terms can be represented by the following expression:

$$\text{Intersection Over Union}\,(IoU) = (Target \cap Prediction)/(Target\ U\ Prediction)$$

In our case using BBoxes, it can be modified to:

$$IOU\,(Box\,1, Box\,2) = \text{Intersection\_Size}\,(Box\,1, Box\,2)/\text{Union\_Size}\,(Box\,1, Box\,2)$$

Consider the the two BBoxes in the following figure:

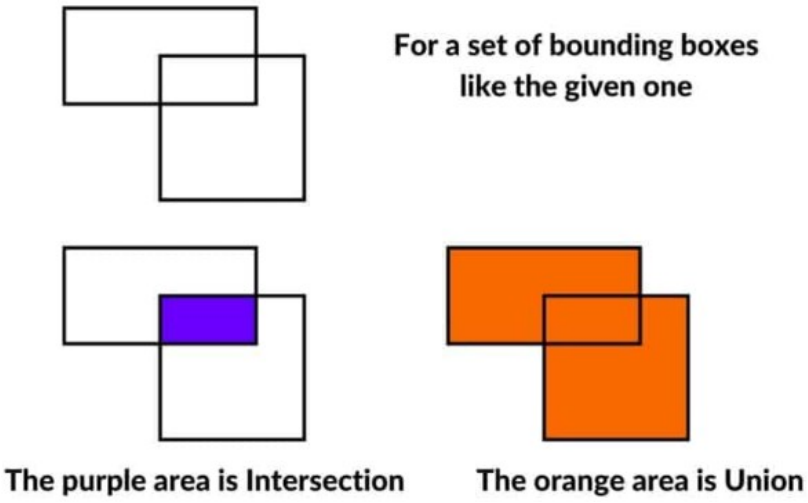


Figure 5-6: IOU with bounding boxes

NMS is used to discard all the overlapping BBoxes and leave only one bounding box around the object. It works as follows:

1. Get the BBox with the highest Pc (let's name it box A).
2. Apply IOU between box A and all other boxes around it.
3. Remove all BBoxes that have high IOU with box A (higher IOU indicates that the other box is overlapped on the real box A where both boxes refer to the same object).

For example, by Looking at the figure below you can see two guys detected with multiple overlapping BBoxes. If we take the area on the left, there are many overlapped boxes, the box in green has the highest Pc. When applying IOU you'll find that IOU between the blue boxes and the green box is high while the IOU between the green box and the yellow boxes or red box is low. Therefore, we can say that the overlapping blue boxes and the green box refer to the same object (the guy on the left) thus removed while the yellow boxes and the red box is left as it is. Once non-max suppression is applied on the right area you will find that the yellow boxes will be removed as well.

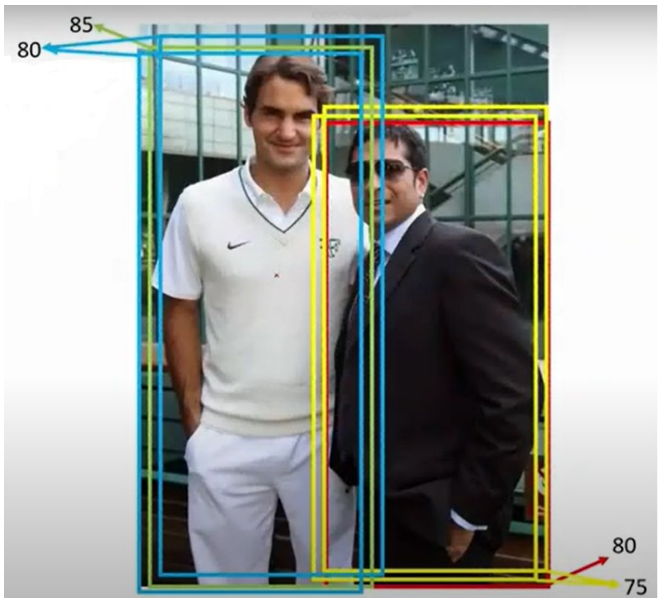


Figure 5-7: Overlapped bounding boxes

5.5 Distance Calculation[32]

In our project, we need to calculate the distances of the objects detected by the user from camera to be able to help him/her estimate where the objects are and how far they are.

How to calculate the distance?

The only thing we have is the image taken from camera and to be able to determine the distance from the camera to known objects, we are going to utilize the triangle similarity.

Triangle similarity goes like this: let's say that we have an object with a known width W, placed at a distance D from the camera. When an image of the object is taken form the camera, we measure the apparent width of the object in pixels P. This allows us to derive the perceived focal length F of our camera, see Equation below:

$$F = \frac{P * D}{W}$$

Now all we need to do is to swap the variables in the equation above to be in the form we need as shown here:

$$D = \frac{W * F}{P}$$

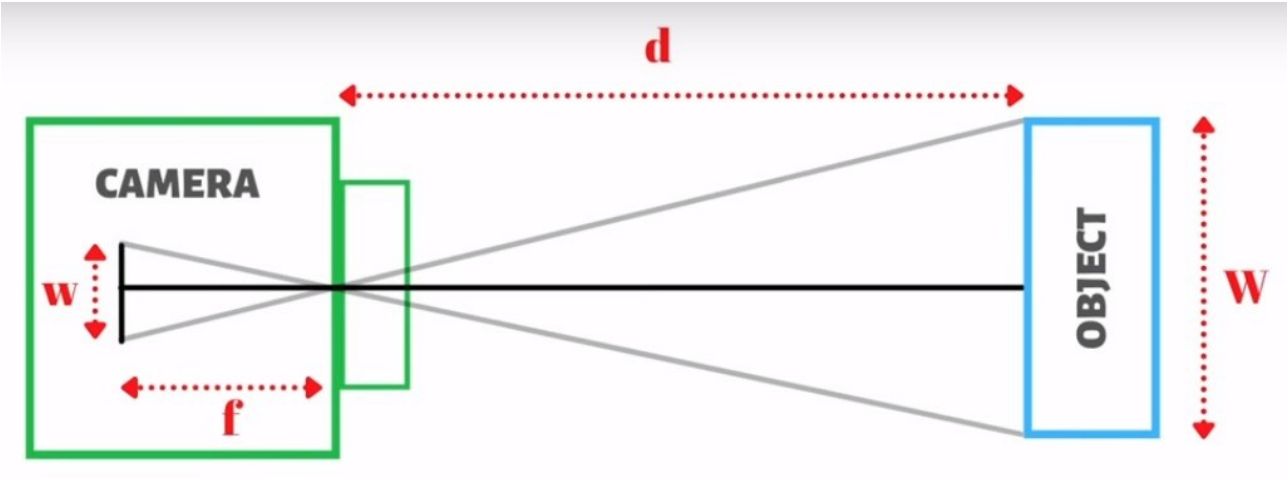


Figure 5-8: Distance parameters

There will be a problem regarding the actual width of the object (W) and how to determine the focal length of the camera (F). The first problem is determining the actual width of the object. A bottle for example, there are many bottles with different widths and heights. So which bottle width should we calculate to be used in our equation? There are way too many variations. The only way to minimize the error resulting from the variations between bottles sizes is to use the average width of the object. So for each object available for detection, we need to find and store its average width.

The second problem is easy to solve. All we need to do is get a certain object, the bottle for example, measure its width (W), place it at a known distance (D) from the camera, then take an image using the camera and calculate width of the object in pixels in the image taken. Now all we have to do is use the first equation we mentioned and calculate focal length (F) of the camera.

5.6 Our Model Results

For code, look at Appendix F

Figure 5-9: Objects detected

Chapter 6 : Face Recognition[16]

6.1 Introduction

You have probably noticed how common face recognition is among the softwares of big tech companies such as Facebook, Snapchat, Twitter, or any other social media software out there. It is also used in security systems to control users access. An example of that can be closer than you think; if you have a modern smartphone, you might be using face recognition technology to be allowed to open your screen.

Many other uses can be mentioned but as far as we are concerned in our project, we want to help the visually impaired people be able to detect the people they know as any other person would do. We do this by providing our A. Eye device with a face recognition model (see Appendix G for code) that keeps running and recognizing faces in real-time. This way is the most natural way to help our users feel close to the people that don't have the same circumstances.

The technology of face recognition has gone through several updates and many researches have been made to improve its accuracy and speed over the last decades. We didn't need to search much to find the best face recognition software to use in our A. Eye[31]. It appears that the face_recognition library[30] made by Adam Geitgey[29] has all our needs. It has the speed needed for the real-time nature of the system, accuracy that is high enough to not get mistaken between faces, and simple syntax that helps getting the work done in the fastest way possible.

In this chapter, we'll look at how the face_recognition library works behind the scenes by utilizing OpenFace and dlib libraries in Python and the theory in the heart of all that. Generally speaking, face recognition is really a series of several related problems:

1. First, look at a picture and find all the faces in it.
2. Second, focus on each face and be able to understand that even if a face is turned in a weird direction or in bad lighting, it is still the same person.
3. Third, be able to pick out unique features of the face that you can use to tell it apart from other people; like how big the eyes are, how long the face is, etc.
4. Finally, compare the unique features of that face to all the people you already know to determine the person's name.

All of those problems can be solved by choosing one machine learning/deep learning algorithm, feeding in data, and getting the result. As a human, your brain is wired to do all of this automatically and instantly. In fact, humans are too good at recognizing faces that they end up seeing faces in everyday objects. Computers are not that smart though.

We need to build a *pipeline* where we solve each step of face recognition separately and pass the result of the current step to the next step. In other words, we will chain together several machine learning algorithms as explained in the next sections.

6.2 Step 1: Face Detection

Face detection is a great feature that is commonly used for cameras. When the camera can automatically pick out faces, it can make sure that all the faces are in focus before it takes the picture. But we'll use it for a different purpose; finding the areas of the image we want to pass on to the next step in our pipeline.

Face detection went mainstream in the early 2000's when Paul Viola and Michael Jones invented a way to detect faces[17] that was fast enough to run on cheap cameras. However, much more reliable solutions exist now. We're going to use a method invented in 2005 called Histogram of Oriented Gradients[18] — or just ***HOG*** for short.

Histogram of Oriented Gradients (HOG)

To find faces in an image, we'll start by making our image black and white because we don't need color data to find faces. Then we'll look at every single pixel in our image one at a time. For every single pixel, we want to look at the pixels that directly surround it as shown in Figure 6-1.

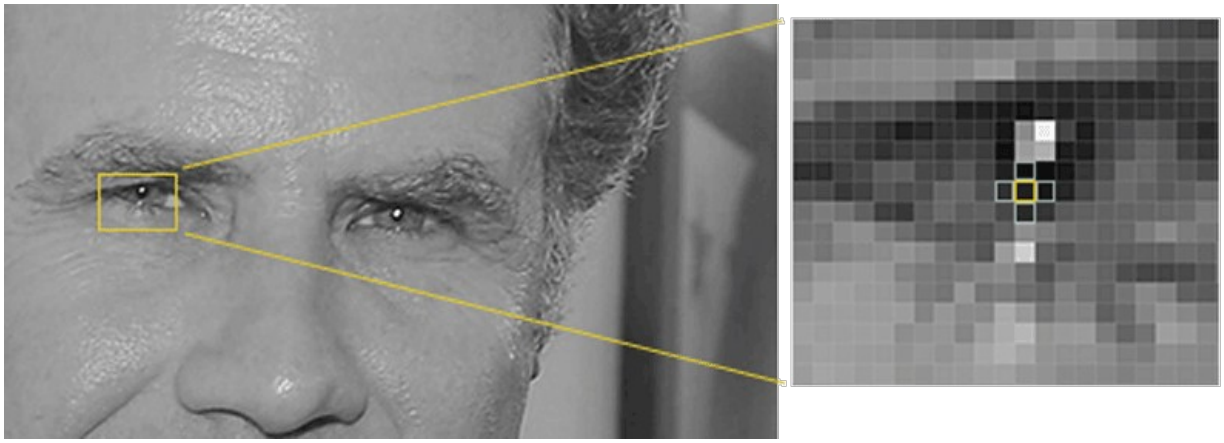


Figure 6-1: Checking surrounding pixels for every pixel

Then we need to figure out how dark the current pixel is compared to the pixels directly surrounding it. Afterwards, we want to draw an arrow showing in which direction the image is getting darker as shown in Figure 6-2.

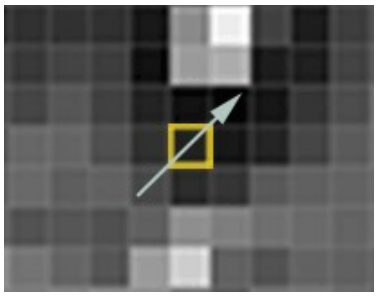


Figure 6-2: Image is getting darker towards the upper right

When repeating that process for every single pixel in the image, we end up with every pixel being replaced by an arrow. These arrows are called *gradients* and they show the flow from light to dark across the entire image. This is done on several levels but Figure 6-3 shows the final form of it.

This might seem like a random thing to do, but there's a really good reason for replacing the pixels with gradients. If we analyze pixels directly, really dark images and really light images of the same person will have totally different pixel values. But by only considering the *direction* at which the brightness changes, both really dark images and really bright images will end up with the same exact representation. That makes the problem a lot easier to solve.

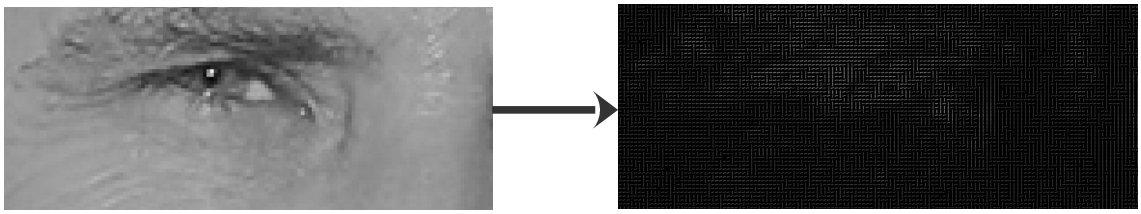


Figure 6-3: Pixels converted to gradients

But saving the gradient for every single pixel gives us way too much detail. We end focusing too much on small pixels while the real concern is the picture as a whole. It would be better if we could just see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image.

To do this, we break up the image into small squares of 16x16 pixels each. In each square, we'll count up how many gradients point in each major direction (how many point up, point up-right, point right, etc…). Then we'll replace that square in the image with the arrow directions that were the strongest. The end result is we turn the original image into a very simple representation that captures the basic structure of a face in a simple way as shown in Figure 6-4.

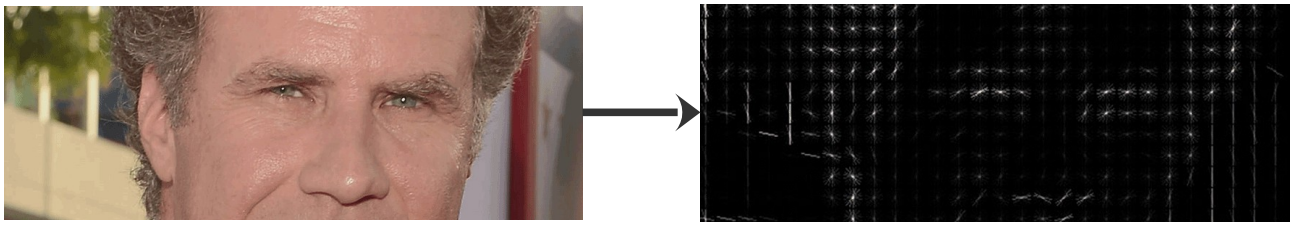


Figure 6-4: Original image turned into a HOG representation

To find faces in this HOG image, all we have to do is find the part of our image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces as shown in Figure 6-5. Using this technique, we can now easily find faces in any image.

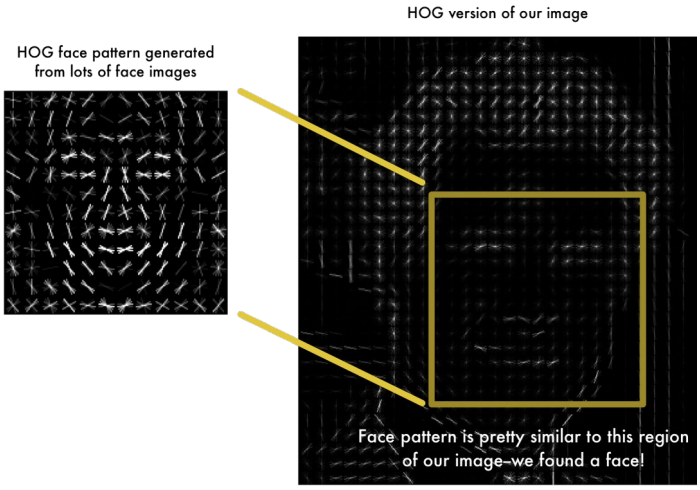


Figure 6-5: HOG patterns compared to each other

6.3 Step 2: Posing and Projecting Faces

After isolating the faces in our image, we now have to deal with another problem; faces turned to different directions look totally different to a computer. Humans can easily recognize if two images taken at different angles are of a person or not, but computers would see these pictures as two completely different people.

To account for this, we will try to warp each picture so that the eyes and lips are always in the sample place in the image. This will make it a lot easier for us to compare faces in the next steps. To do this, we are going to use an algorithm called ***face landmark estimation***. There are lots of ways to do this, but we are going to use an approach invented in 2014 by Vahid Kazemi and Josephine Sullivan[20].

Face Landmark Estimation

The basic idea is that we will come up with 68 specific points (called *landmarks*) that exist on every face; the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. All the 68 landmarks are represented in Figure 6-6. Then we will train a machine learning algorithm to be able to find these 68 specific points on any face. You might have noticed by now that using this technique makes it easy to computationally recognize faces represented in different angles as the same person without much processing.

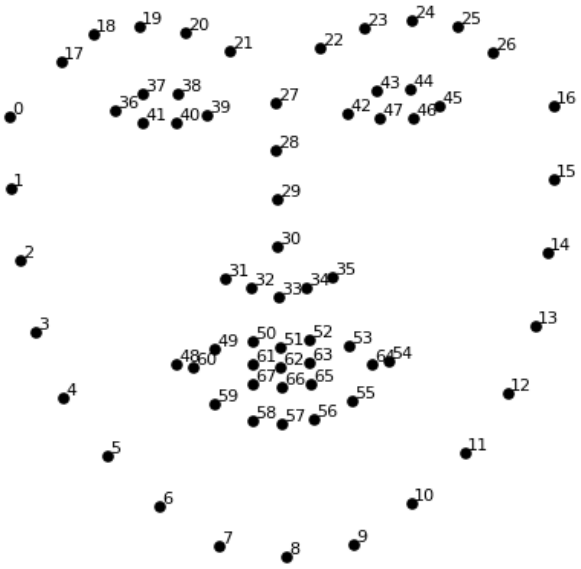


Figure 6-6: 68 face landmarks

Now that we know where the eyes and mouth are, we'll simply rotate, scale and shear[21] the image so that the eyes and mouth are centered as best as possible. We are only going to use basic image transformations like rotation and scale that preserve parallel lines (called affine transformations[22]). This is illustrated below in Figure 6-7. Now no matter how the face is turned, we are able to center the eyes and mouth in roughly the same position as in the image. This will make our next step a lot more accurate.

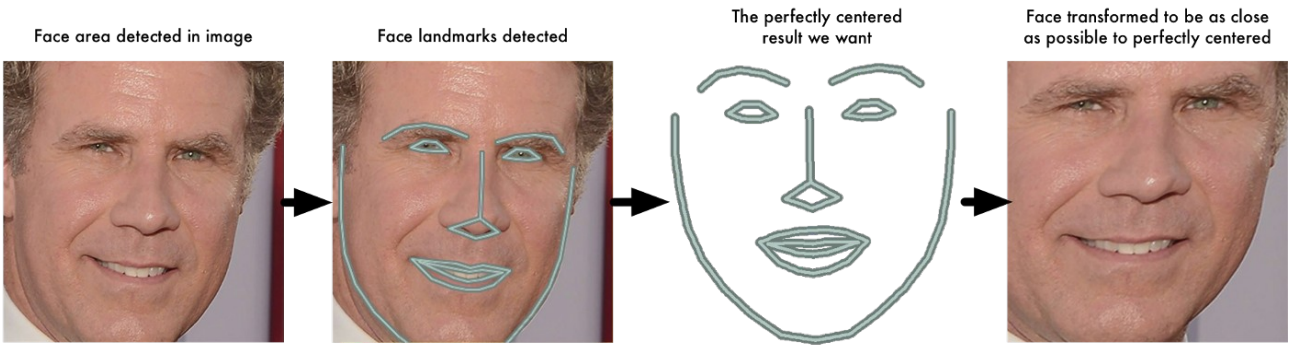


Figure 6-7: Face landmarks detection and face transformation

6.4 Step 3: Encoding Faces

At this step, we need to know how to tell faces apart. The simplest approach to face recognition is to directly compare the unknown face we found in Step 2 with all the pictures we have of people that have already been tagged. When we find a previously tagged face that looks very similar to our unknown face, it must be the same person.

This seems like a reasonably good idea at first glance, but there's actually a huge problem with that approach. A person walking around with the A. Eye and moving his/her head frequently can generate tens of photose per second that can contain multiple faces in each photo. The algorithm can't possibly loop through every previously tagged face to compare it to every newly uploaded picture. That would take way too long. It needs to be able to recognize faces in milliseconds, not minutes.

What we need is a way to extract a few basic measurements from each face. Then we could measure our unknown face the same way and find the known face with the closest measurements. For example, we might measure the size of each ear, the spacing between the eyes, the length of the nose, etc.

Now you might ask: which measurements should we collect from each face to build our known face database? It turns out that the measurements that seem obvious to us humans (like eye color) don't really make sense to a computer looking at individual pixels in an image. Researchers have discovered that the most accurate approach is to let the computer figure out the measurements to collect itself. Deep learning does a better job than humans at figuring out which parts of a face are important to measure.

The solution is to train a Deep Convolutional Neural Network (more on that can be found at reference point [19]). But instead of training the network to recognize pictures objects like we did last time, we are going to train it to generate 128 measurements for each face. The training process works by looking at 3 face images at a time:

1. Load a training face image of a known person.
2. Load another picture of the same known person.
3. Load a picture of a totally different person.

Then the algorithm looks at the measurements it is currently generating for each of those three images. It then tweaks the neural network slightly so that it makes sure the measurements it generates for #1 and #2 are slightly closer while making sure the measurements for #2 and #3 are slightly further apart. This is illustrated in Figure 6-8.

After repeating this step millions of times for millions of images of thousands of different people, the neural network learns to reliably generate 128 measurements for each person. Any ten different pictures of the same person should give roughly the same measurements.

Machine learning people call the 128 measurements of each face an embedding. The idea of reducing complicated raw data like a picture into a list of computer-generated numbers comes up a lot in machine learning (especially in language translation). The exact approach for faces we are using was invented in 2015 by researchers at Google[23] but many similar approaches exist.

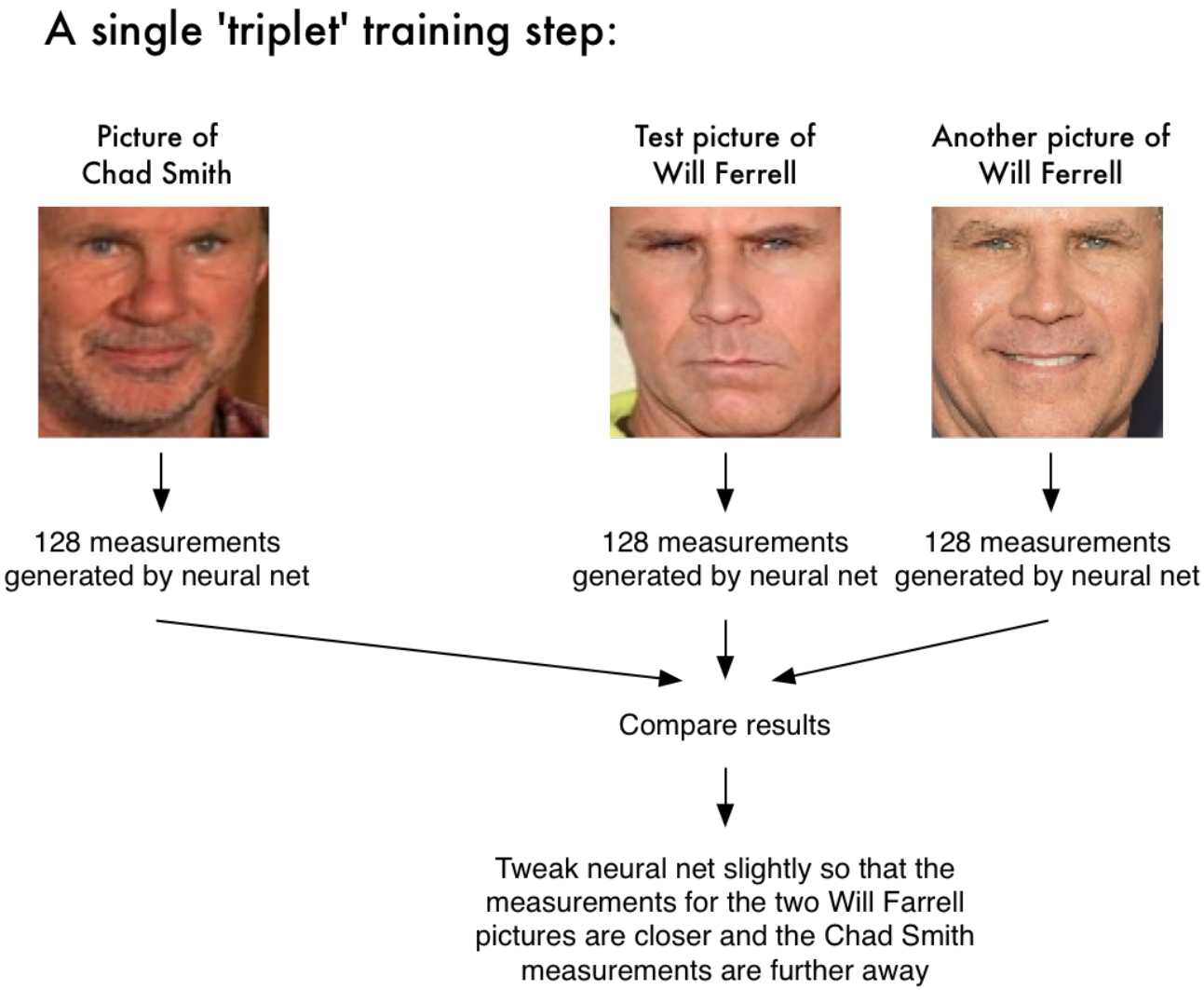


Figure 6-8: Triplet training step for CNN

This process of training a convolutional neural network to output face embeddings requires a lot of data and computer power. Even with an expensive NVidia Telsa video card[24], it takes about 24 hours[25] of continuous training to get good accuracy. But once the network has been trained, it can generate measurements for any face, even ones it has never seen before. So this step only needs to be done once. Fortunately, this has been already taken care of by people at OpenFace[26] already did this and they published several trained networks[27] which we can directly use thankfully. So all we need to do ourselves is run our face images through their pre-trained network to get the 128 measurements for each face. The measurements for our test image is shown in Figure 6-9.

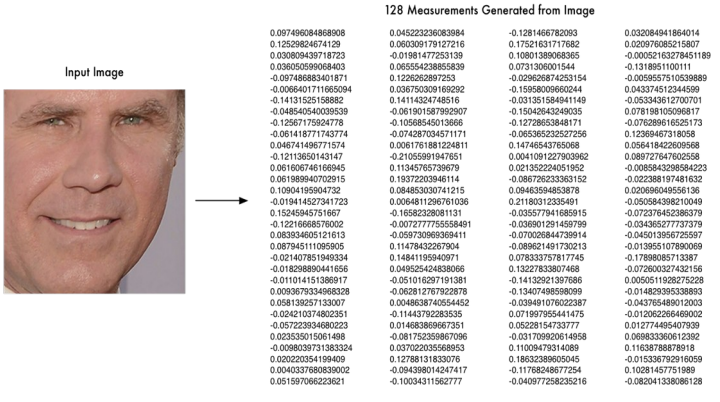


Figure 6-9: Example of the 128 measurements generated from the OpenFace network on a test image

So what parts of the face are these 128 numbers measuring exactly? It turns out that we have no idea. It doesn't really matter to us. All that we care for is that the network generates nearly the same numbers when looking at two different pictures of the same person.

6.5 Step 4: Finding the Person's Name from the Encoding

This last step is actually the easiest step in the whole process. All we have to do is find the person in our database of known people who has the closest measurements to our test image. You can do that by using any basic machine learning classification algorithm. No deep learning tricks are needed. We'll use a simple linear SVM classifier[28], but lots of classification algorithms could work.

All we need to do is train a classifier that can take in the measurements from a new test image and tells which known person is the closest match. Running this classifier takes milliseconds. The result of the classifier is the name of the person.

If you want to test this out, here's the steps to follow:

1. Train a classifier with the embeddings of about 20 pictures of each person you want your system to know.
2. Run the classifier on either testing pictures saved in a dataset or on every frame of a video containing the face of the person you want to recognize. It depends on your needs. In our project, we make the testing on a live video frames the are taken by the Raspberry Pi camera.

6.6 Our Model Results

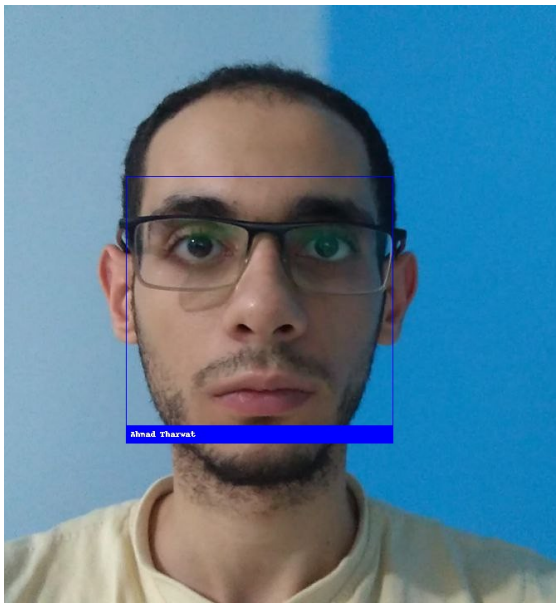


Figure 6-10: Person recognized

Chapter 7: Money Recognition

7.1 Introduction

In this chapter, we are going to add money recognition functionality. First, we need to choose which currency to be recognized (in our case it will be the Egyptian currency EGP), then collecting data and labeling them. Second, train a model on the data collected until you are satisfied with its accuracy. In chapter 5, we talked about object detection using YOLO. We are going to use it for training custom our objects (EGP currencies) and explain it in detail. Third, deploying the model in code and use it.

7.2 Data Collection and Labeling

For collecting the data there are many options: collect data from the internet, making the data yourself by taking photos, or making videos of the data and extract frames using code. Due to the lack of data on EGP currency on the internet we collected data using the last two options. To get the data we have collected refer to reference point [48].

Now after the data has been collected, we need to label each photo of the data collected using the following steps:

1. Download labelImg GitHub repository[49] and place the labelImg folder in your project file.
2. Open command prompt or anaconda prompt and change directory to the labelImg folder.
3. Install pyqt5: using `conda install pyqt=5` (in case anaconda is used) otherwise use `pip install pyqt5==5`.
4. Install lxml: using `conda install lxml=4.6.1` or `pip install lxml==4.6.1`
5. After installation: open the labelImg folder move these two files: resources.py and resources.qrc into the libs file.
6. Then return to the prompt write the command:
   `pyrcc5 -o libs/resources.py resources.qrc`
7. Go to the data folder in the labelImg folder and change the content of predefined_classes.txt to the following:

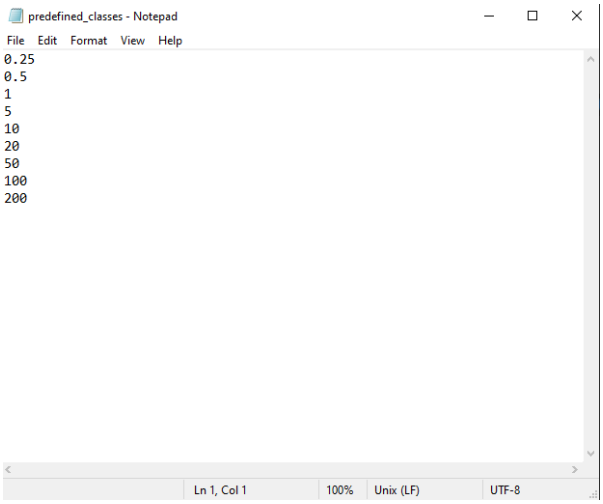


Figure 7-1: Predefined_classes update

8. Create a new folder (let's name it image data) and place all data images inside it.
9. In prompt write the command: python labelImg.py. the following window will appear:

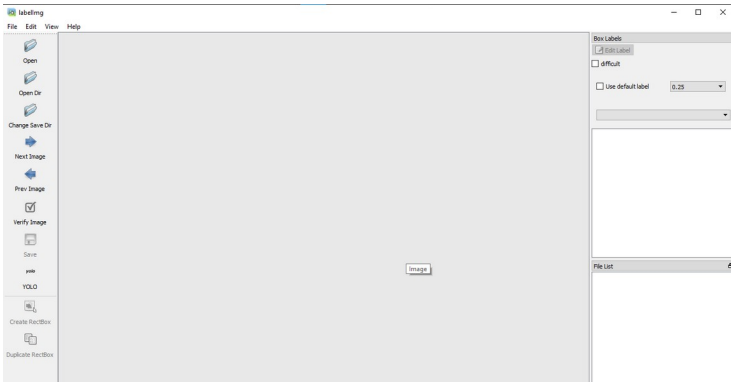


Figure 7-2: LabelImg program

10. Click on open Dir and select image data folder the program will open the first image inside the folder like the following:

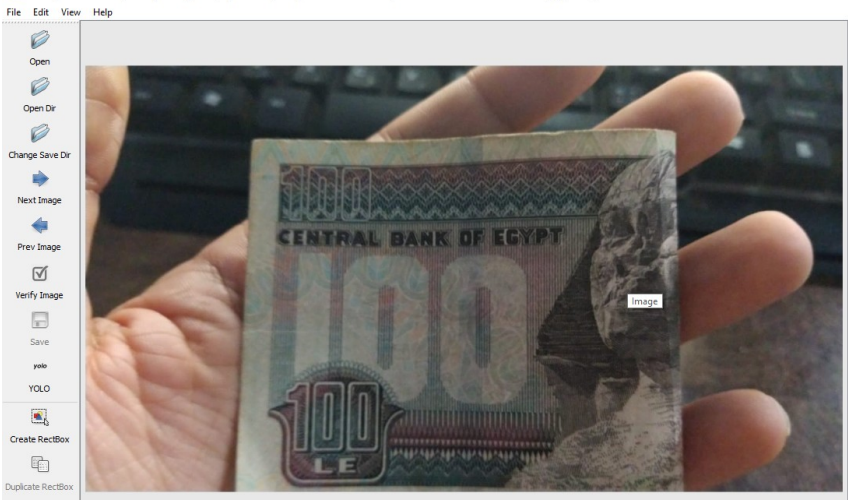


Figure 7-3: LabelImg displaying an image

11. Click on create RectBox on the left side and mark the object (currency) and try to make the box round the object as much as possible a small window will appear asking which class does this object belong to select the class and click ok. Like the following:

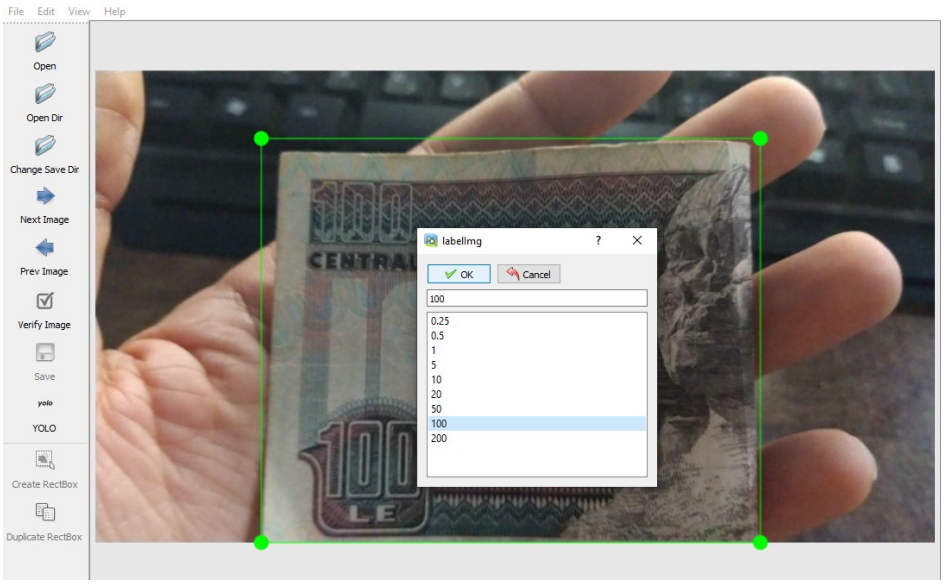


Figure 7-4: Labeling the currency

12. Click on save then click on next image and repeat until you finish labeling all the images. After you finish, You will find in the image data folder for each image a text file containing labeling information and classes.txt for the classes information.

7.3 Model Training

Before we start training the model, you will need a computer with GPU, prepare some installations for proper environment for training the model. refer to these two YouTube videos for preparations at reference points [50] and [51]. When the environment is ready, you should have a darknet folder. Now follow these steps:

1. Go to this path in the darknet folder darknet/darknet-master/build/x64/data create a new folder name it obj.
2. Now go to the image data folder copy all the files in it except for the classes.txt and paste it inside the obj folder.
3. Return back to the data folder make a copy of coco.data and coco.names files inside the same folder (data) rename the copies into obj.data and obj.names.
4. Open obj.names with any text editor and change its content like in Figure 7-1.

5. Open obj.data change it to the following:

```
classes= 9
train  = data/train.txt
names = data/obj.names
backup = backup/
```

Figure 7-5: obj.data update

6. Go back to the x64 folder, then go inside cfg folder copy a file named yolov4-custom.cfg into the same folder, then name the copy yolov4-obj.cfg.

7. Open that file (yolov4-obj.cfg) with any text editor and change the following:

   Subdivisions = 64 or 32, width = 416, height = 416, max_batches = 18000,
   steps = 14400, 16200, in each yolo layer: classes = 9 and each filtersparameter before the yolo layer: filters = 42

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=64
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 18000
policy=steps
steps=14400,16200
scales=.1,.1

```

```
963 filters=42
964 activation=linear
965
966
967 [yolo]
968 mask = 0,1,2
969 anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
970 classes=9
971 num=9
972 jitter=.3
973 ignore_thresh = .7
974 truth_thresh = 1
975 scale_x_y = 1.2
976 iou_thresh=0.213
977 cls_normalizer=1.0
978 iou_normalizer=0.07
979 iou_loss=ciou
980 nms_kind=greedynms
981 beta_nms=0.6
982 max_delta=5
983
```

```
1051 filters=42
1052 activation=linear
1053
1054
1055 [yolo]
1056 mask = 3,4,5
1057 anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
1058 classes=9
1059 num=9
1060 jitter=.3
1061 ignore_thresh = .7
1062 truth_thresh = 1
1063 scale_x_y = 1.1
1064 iou_thresh=0.213
1065 cls_normalizer=1.0
1066 iou_normalizer=0.07
1067 iou_loss=ciou
1068 nms_kind=greedynms
1069 beta_nms=0.6
1070 max_delta=5
1071
```

```
1139 filters=42
1140 activation=linear
1141
1142
1143 [yolo]
1144 mask = 6,7,8
1145 anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
1146 classes=9
1147 num=9
1148 jitter=.3
1149 ignore_thresh = .7
1150 truth_thresh = 1
1151 random=1
1152 scale_x_y = 1.05
1153 iou_thresh=0.213
1154 cls_normalizer=1.0
1155 iou_normalizer=0.07
1156 iou_loss=ciou
1157 nms_kind=greedynms
1158 beta_nms=0.6
1159 max_delta=5
1160
```

Figure 7-6: yolov4-obj.cfg updates

8. Refer to the GitHub link at reference point [52] go to the README.md part click on yolov4.conv.137 and download it then download the repository.
9. Place yolov4.conv.137 and create_list_of_images.py file (from the repository) in the x64 folder.
10. Open create_list_of_images and run it or run it from prompt using the command python create_list_of_images.
11. To ensure that the script ran well go to the data folder you should find a text file called train open it and you shall see all paths to all the images in the obj folder.
12. To train the model: go to the prompt make sure that the directory is in x64 folder and run the command: `darknet.exe detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv.137`
13. Now the training should have started, and a loss graph window have popped up. Wait until you are satisfied with AVG loss value shown under the graph (note the lower the loss the better the model accuracy). This should take a lot of time.
14. Once you are satisfied with the avg loss go to the running prompt click ctrl C to stop.
15. Go to the x64 folder you should find an image (named chart_yolov4-obj.png) of the graph. And if you go to the backup folder you should find a yolov4-obj_last.weights file.

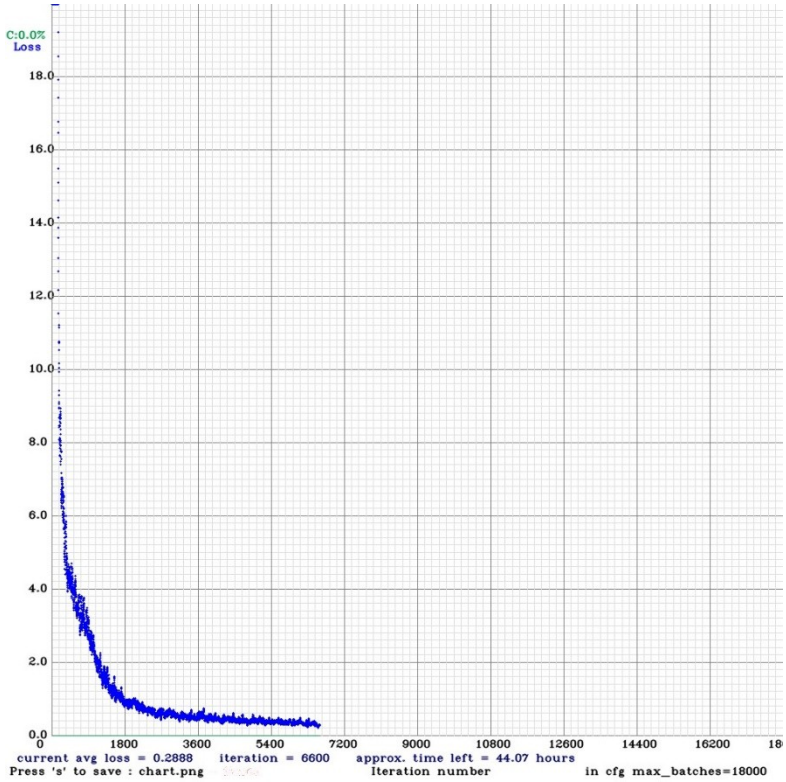


Figure 7-7: chart_yolov4-obj.png

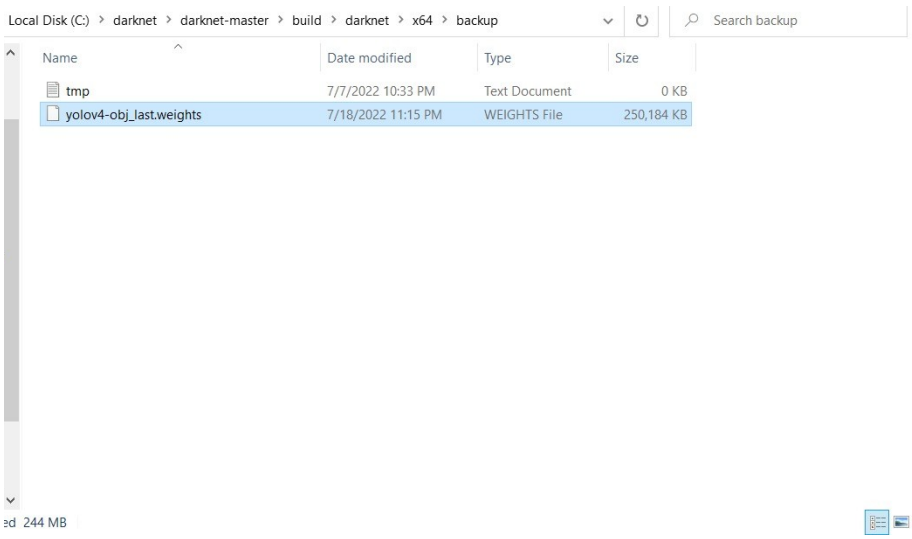


Figure 7-8: Training Outputs

16. If you want to resume training of the model after it has stopped or you wanted to put new data and train it on the new data, write the following command:
    `darknet.exe detector train data/obj.data cfg/yolov4-obj.cfg backup/yolov4-obj_last.weights`
17. To deploy or use the model in code refer to Appendix H

7.4 Our Model Results

Figure 7-9: Egyptian Currency Recognized

Chapter 8: Text Recognition[39]

8.1 Introduction[36][37]

Optical Character Recognition or (***OCR***) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast).

Widely used as a form of data entry from printed paper data records – whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation – it is a common method of digitizing printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

Early versions needed to be trained with images of each character, and worked on one font at a time. Advanced systems capable of producing a high degree of recognition accuracy for most fonts are now common, and with support for a variety of digital image file format inputs. Some systems are capable of reproducing formatted output that closely approximates the original page including images, columns, and other non-textual components.

OCR is generally an "offline" process, which analyses a static document. There are cloud based services which provide an online OCR API service. Handwriting movement analysis can be used as input to handwriting recognition. Instead of merely using the shapes of glyphs and words, this technique is able to capture motions, such as the order in which segments are drawn, the direction, and the pattern of putting the pen down and lifting it. This additional information can make the end-to-end process more accurate. This technology is also known as "on-line character recognition", "dynamic character recognition", "real-time character recognition", and "intelligent character recognition".

Out of many OCR engines, tesseract was our engine of choice. Some reasons behind that choice is that tesseract is one of the top OCR engines in terms of accuracy, it is available for linux, which is our OS of choice, it can detect whether text is monospaced or proportionally spaced, and it can process right-to-left text, so Arabic language can be read. Throughout this chapter, you'll learn how tesseract works step by step to read text from our camera. In Appendix I, you can read the code we made for our A. Eye using tesseract.

8.2 Overview of Tesseract[38]

Tesseract is an open source text recognition (OCR) Engine, available under the Apache 2.0 license. It can be used directly, or (for programmers) using an API to extract text from images or even extract text from PDFs. It supports a wide variety of languages. Tesseract doesn't have a built-in GUI, but there are several available from the 3rd parties. Tesseract is compatible with many programming languages and frameworks through wrappers. It can be used with the existing layout analysis to recognize text within a large document, or it can be used in conjunction with an external text detector to recognize text from an image of a single text line.

Processing in Tesseract follows a traditional step-by-step pipeline, but some of the stages were unusual in their day, and possibly remain so even now. The first step is a connected component analysis in which outlines of the components are stored. This was a computationally expensive design decision at the time, but had a significant advantage: by inspection of the nesting of outlines, and the number of child and grandchild outlines, it is simple to detect inverse text and recognize it as easily as black-on-white text. Tesseract was probably the first OCR engine able to handle white-on-black text so trivially. At this stage, outlines are gathered together, purely by nesting, into Blobs.

Blobs are organized into text lines, and the lines and regions are analyzed for fixed pitch or proportional text. Text lines are broken into words differently according to the kind of character spacing. Fixed pitch text is chopped immediately by character cells. Proportional text is broken into words using definite spaces and fuzzy spaces.

Recognition then proceeds as a two-pass process. In the first pass, an attempt is made to recognize each word in turn. Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to more accurately recognize text lower down the page.

Since the adaptive classifier may have learned something useful too late to make a contribution near the top of the page, a second pass is run over the page, in which words that were not recognized well enough are recognized again.

A final phase resolves fuzzy spaces, and checks alternative hypotheses for the x-height to locate small- cap text.

Tesseract 4.00 includes a new neural network subsystem configured as a text line recognizer. It has its origins in OCRopus' Python-based LSTM implementation but has been redesigned for Tesseract in C++. The neural network system in Tesseract pre-dates TensorFlow but is compatible with it, as there is a network description language called Variable Graph Specification Language (VGSL), that is also available for TensorFlow.

Following in the next sections, we will expand this overview and learn about the steps of processing that happen inside tesseract phase by phase.

8.3 Phase 0: Image Preprocessing

Tesseract seems not to work very well with some images in their original form. Therefore, we have made some field testing to search for the best accuracy we can get out of an image taken from the Raspberry Pi camera. These tests were on the best practices of image preprocessing we can perform on our images to get the best results. The final parameters are written in the code in Appendix I. Here, we will explain in details about the effects of each step in the preprocessing pipeline.

8.3.1 RGB to Gray Conversion

This step is straightforward and is done in a single line of code. The importance of this step is not as simple as its implementation though. Converting an RGB image to Gray scaled one skyrockets the processing speed required on the image threefold. If you notice, working on a single channel of colors is always faster than working on three.

This also removes unnecessary information from the image. Since colors are not directly affecting the accuracy of detecting and recognizing text, it is unnecessary in such a system.

8.3.2 Resizing[53]

The standard recommended resolution for OCR is between 300 and 600 DPI (Dots Per Inch). It seems that if the image was below 300 DPI, the accuracy of the system decreases and less text is detected. While if it was above 600 DPI, the processing time increases without any actual gain in text reading accuracy.

This is directly related to the image resolution, if the concept of DPI was not clear enough. What we did was scaling our images taken from the Raspberry Pi camera by 1.5x1.5. That way, even if the image was taken from a distance further than recommended, the image would get scaled to normal DPI values and accuracy would be barely affected.

8.3.3 CLAHE and Binarization[54]

Contrast Limiting Adaptive Histogram Equalization (CLAHE) is is a variant of adaptive histogram equalization in which the contrast amplification is limited, so as to reduce this problem of noise amplification. In simple words, CLAHE does histogram equalization in small patches or in small tiles with high accuracy and contrast limiting[56].

But what is histogram equalization and why do we need it in the first place? Well, an image histogram is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image[55]. The reason we use it is that when we collect images that are washed out or images with low contrast, we can stretch the histogram to span the entire range.

In layman's terms *Binarization* means converting a colored image into an image which consists of only black and white pixels (Black pixel value = 0 and White pixel value = 255). As a basic rule, this can be done by fixing a *threshold* (normally threshold = 127, as it is exactly half of

the pixel range 0–255). If the pixel value is greater than the threshold, it is considered as a white pixel, else considered as a black pixel.

But this strategy may not always give us desired results. In the cases where lighting conditions are not uniform in the image, this method fails. So, the crucial part of binarization is determining the *threshold*. This can be done by using various techniques:

Otsu's Binarization

This method gives a threshold for the whole image considering the various characteristics of the whole image (like lighting conditions, contrast, sharpness etc) and that threshold is used for Binarizing image.

Adaptive Thresholding

This method gives a threshold for a small part of the image depending on the characteristics of its locality and neighbours i.e there is no single fixed threshold for the whole image but every small part of the image has a different threshold depending upon the locality and also gives smooth transition.

We have tested out three thresholding methods and came to the conclusion of using a combination of normal binary thresholding and Otsu's binarization.

8.3.4 Skewing[54]

While scanning a document, it might be slightly skewed (image aligned at a certain angle with horizontal angle) sometimes. While extracting the information from the scanned image, detecting & correcting the skew is crucial. There are several techniques used for skewing. But we chose the projection profile method which is considered the easiest and most widely used method. We will explain how the method works in simple steps:

1. Take the binary image.
2. Take the sum of pixels along rows of the image matrix to get the count of foreground pixels for every row.
3. Now the image is rotated at various angles (at a small interval of angles called *Delta*) and the difference between the peaks will be calculated. The angle at which the maximum difference between peaks is found, that corresponding angle will be the *Skew angle* for the image.
4. After finding the Skew angle, we can correct the skewness by rotating the image through an angle equal to the skew angle in the *opposite direction* of skew. You can visualize the process as in Figure 8-1.

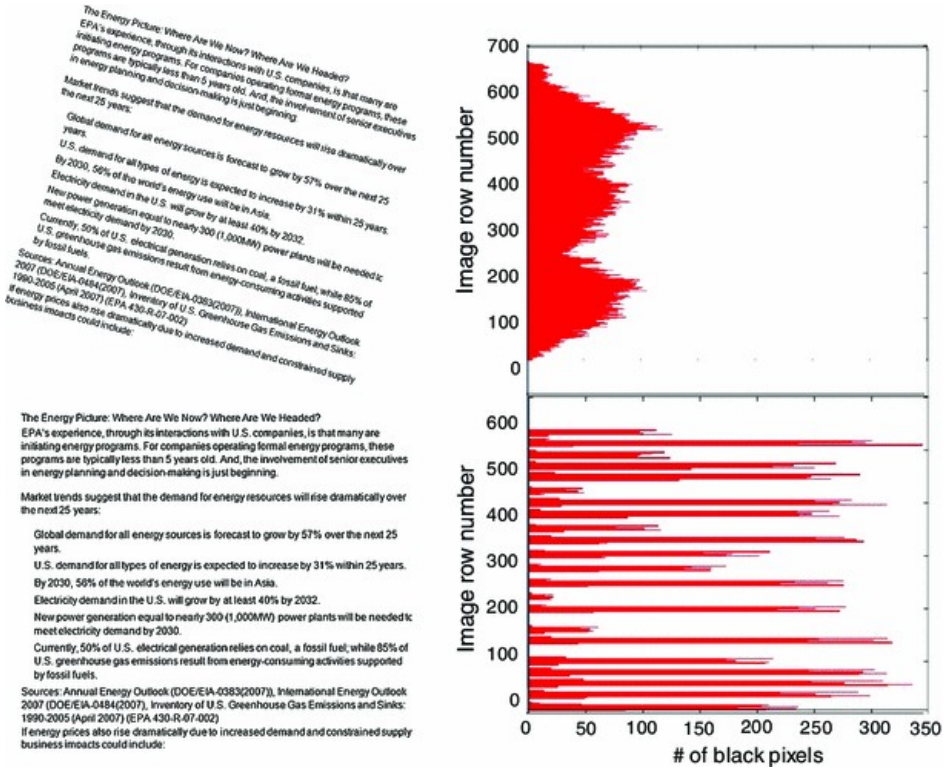


Figure 8-1: Correcting skew using the Projection Profile method

8.3.5 Noise Removal[57]

The main objective of the *Noise removal* stage is to smoothen the image by removing small dots/patches which have high intensity than the rest of the image. Noise removal can be performed for both *Coloured* and *Binary images*.

There is a fair number of smoothing filters (and the number is increasing). It is a very interesting topic which also affects the model accuracy greatly. You can read more about noise removal filters at reference point [57]. We will talk here about the median filter which we found to give the best results for our use case.

Median Filter

The median filter calculates the median of the pixel intensities that surround the center pixel in a *n x n* kernel. The median then replaces the pixel intensity of the center pixel. The median filter does a better job of removing salt and pepper noise than mean and Gaussian filters. The median filter preserves the edges of an image but it does not deal with speckle noise. This functionality of preserving edges is why we chose this filter. This way, we would save the edges for characters to not perform further unnecessary edge detection techniques like needed when using mean or Gaussian filters that result in blurry images. Another advantage is that median filter can remove some of the salt and pepper noise while retaining the edges of the image in case it is applied to grayed images.

A lot of other preprocessing steps can be performed on an image. Those were the ones we chose for our use case. Depending on your use case, you are free to make your own preprocessing pipeline. Ranging from just using different parameters than we used to adding or removing complete preprocessing steps in the pipeline.

8.4 Phase 1: Connected Component Analysis

8.4.1 Line Finding

The line finding algorithm is designed so that a skewed page can be recognized without having to de-skew, thus saving loss of image quality. The key parts of the process are blob filtering and line construction.

Assuming that page layout analysis has already provided text regions of a roughly uniform text size, a simple percentile height filter removes drop-caps and vertically touching characters. The median height approximates the text size in the region, so it is safe to filter out blobs that are smaller than some fraction of the median height, being most likely punctuation, diacritical marks and noise.

The filtered blobs are more likely to fit a model of non-overlapping, parallel, but sloping lines. Sorting and processing the blobs by x-coordinate makes it possible to assign blobs to a unique text line, while tracking the slope across the page, with greatly reduced danger of assigning to an incorrect text line in the presence of skew. Once the filtered blobs have been assigned to lines, a least median of squares fit is used to estimate the baselines, and the filtered-out blobs are fitted back into the appropriate lines.

The final step of the line creation process merges blobs that overlap by at least half horizontally, putting diacritical marks together with the correct base and correctly associating parts of some broken characters.

8.4.2 Baseline Fitting

Once the text lines have been found, the baselines are fitted more precisely using a quadratic spline. This was another first for an OCR system, and enabled Tesseract to handle pages with curved baselines, which are a common artifact in scanning, and not just at book bindings.

The baselines are fitted by partitioning the blobs into groups with a reasonably continuous displacement for the original straight baseline. A quadratic spline is fitted to the most populous partition, (assumed to be the baseline) by a least squares fit. The quadratic spline has the advantage that this calculation is reasonably stable, but the disadvantage that discontinuities can arise when multiple spline segments are required.

8.5 Phase 2: Blobs Analysis and Word Recognition

Part of the recognition process for any character recognition engine is to identify how a word should be segmented into characters. The initial segmentation output from line finding is classified first. The rest of the word recognition step applies only to non-fixed-pitch text.

8.5.1 Fixed Pitch Detection and Chopping

Tesseract tests the text lines to determine whether they are fixed pitch. Where it finds fixed pitch text, Tesseract chops the words into characters using the pitch, and disables the chopper and associator on these words for the word recognition step. Fig. 8-1 shows a typical example of a fixed-pitch word.

Figure 8-2: A fixed-pitch chopped word.

8.5.2 Proportional Word Finding

Non-fixed-pitch or proportional text spacing is highly non-trivial task. Fig. 8-2 illustrates some typical problems. The gap between the tens and units of '11.9%' is a similar size to the general space, and is certainly larger than the kerned space between 'erated and 'junk'. There is no horizontal gap at all between the bounding boxes of 'of' and 'financial'. Tesseract solves most of these problems by measuring gaps in limited vertical range between the baseline and mean line. Spaces that are close to the threshold at this stage are made fuzzy, so that a final decision can be made after word recognition.

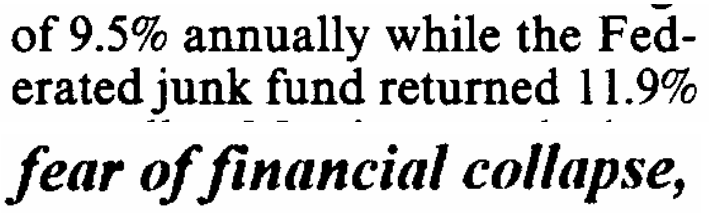


Figure 8-3: Some difficult word spacing.

8.5.3 Chopping Joined Characters

While the result from a word is unsatisfactory, Tesseract attempts to improve the result by chopping the blob with worst confidence from the character classifier. Candidate chop points are found from concave vertices of a polygonal approximation of the outline, and may have either another concave vertex opposite, or a line segment. It may take up to 3 pairs of chop points to successfully separate joined characters from the ASCII set.

Fig. 8-3 shows a set of candidate chop points with arrows and the selected chop as a line across the outline where the ‘r’ touches the ‘m’. Chops are executed in priority order. Any chop that fails to improve the confidence of the result is undone but not completely discarded so that the chop can be re-used later by the associator if needed.

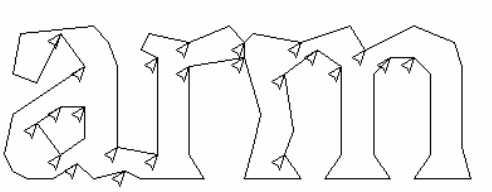


Figure 8-4: Candidate chop points and chop.

8.5.4 Associating Broken Characters

When the potential chops have been exhausted, if the word is still not good enough, it is given to the associator. The associator makes an A* (best first) search of the segmentation graph of possible combinations of the maximally chopped blobs into candidate characters. It does this without actually building the segmentation graph, but instead maintains a hash table of visited states. The A* search proceeds by pulling candidate new states from a priority queue and evaluating them by classifying unclassified combinations of fragments.

8.6 Phase 3: Static Character Classifier

Classification proceeds as a two-step process. In the first step, a class pruner creates a shortlist of character classes that the unknown might match. Each feature fetches, from a coarsely quantized 3-dimensional look- up table, a bit-vector of classes that it might match, and the bit-vectors are summed over all the features. The classes with the highest counts (after correcting for expected number of features) become the short-list for the next step.

Each feature of the unknown looks up a bit vector of prototypes of the given class that it might match, and then the actual similarity between them is computed. Each prototype character class is represented by a logical sum-of-product expression with each term called a configuration, so the distance calculation process keeps a record of the total similarity evidence of each feature in each configuration, as well as of each prototype. The best combined distance, which is calculated from the summed feature and prototype evidences, is the best over all the stored configurations of the class.

8.7 Limitations of Tesseract[38]

Tesseract works best when there is a clean segmentation of the foreground text from the background. In practice, it can be extremely challenging to guarantee these types of setup. There are a variety of reasons you might not get good quality output from Tesseract like if the image has noise on the background. The better the image quality (size, contrast, lightning) the better the recognition result. It requires a bit of preprocessing to improve the OCR results, like you saw in phase 0, images need to be scaled appropriately, have as much image contrast as possible, and the text must be horizontally aligned.

8.8 Our Model Results

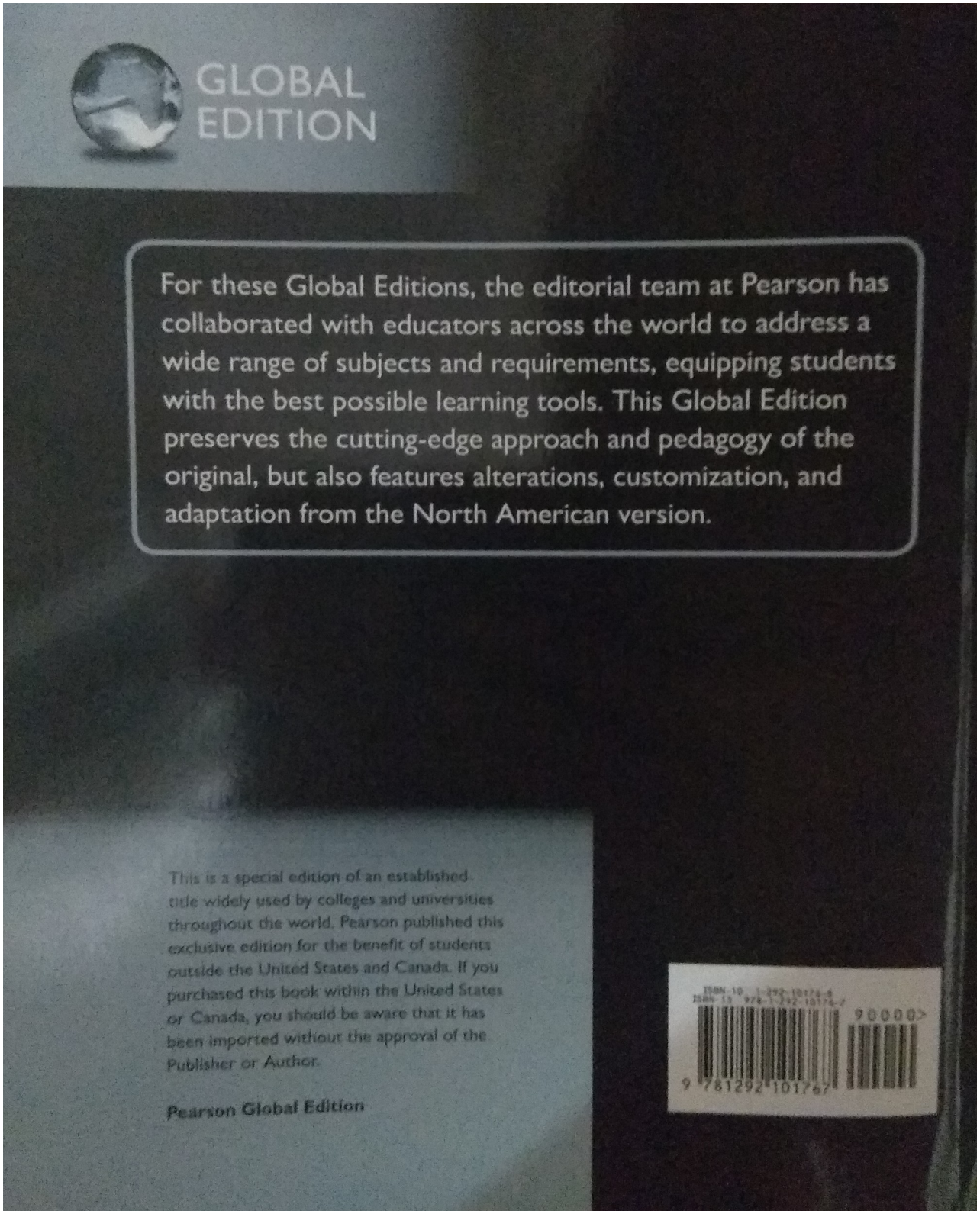


Figure 8-5: OCR test image from
Computer Systems: A Programmer's Perspective
Third Global Edition Book

Result

For these Global Editions, the editorial team at Pearson has
ated with educators across the world to address a
CLs and requirements, equipping students
isle lomrni od tools. This Global Edition
won pens edison of an estabbshed
sje widely used by colleges and univer sivies
rhaoughout the wea ld Pearson published vhis
exclusive edition for (he benefit of students
outside the United States and Canada If you
. purchased this beak witha the United brates
or Carada, yau sheild be aware that it has
been imported wittiour the approval of the
Publisher or Author. a

As you can see, the model is not as accurate as we hoped for. After all, the OCR technology is not mature enough to give us an all-round solution that suits every type of images taken with different lighting scenarios, different resolutions, blurring behavior, or typefaces. Still, the result we got is the best we could reach with the current technology of tesseract which is considered, as far as we know, the best solution currently in the market. It would be up to the next generations of this project to improve those results. We wish you good luck my friend. May Allah help you.

Future Work

- Adding Arabic support for the users in our region.
- Training more objects to the YOLO network should improve on the user experience, giving him/her more vision over the world.
- Flash light can be added to improve the illumination of pictures taken at night.
- Making and improving a navigational model that can help the user walk freely in the streets.
- Making mobile/web applications to monitor the device and help adding more new features more easily.
- Improving accuracy is a main and continues maintenance in the system.
- Increasing money recognition model speed is needed.

References

[1] "Vision Impairment and blindness," World Health Organization. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment. [Accessed: 01-Aug-2021].

[2] "Eastern Mediterranean Region," World Health Organization. [Online]. Available: http://www.emro.who.int/egy/egypt-events/world-sight-day.html. [Accessed: 01-Aug-2021].

[3] "Face Blindness," NHS choices. [Online]. Available: https://www.nhs.uk/conditions/face-blindness/. [Accessed: 01-Aug-2021].

[4] S. Bradt, "Face-blindness disorder may not be so rare", 1-June-2006. [Online]. Available: news.harvard.edu/gazette/story/2006/06/face-blindness-disorder-may-not-be-so-rare/. [Accessed: 01-Aug-2021].

[5] K. Samuelson, "Prosopagnosia and face blindness: Why you may be face blind," Time, 14-Jul-2017. [Online]. Available: https://time.com/4838661/prosopagnosia-face-blindness/. [Accessed: 01-Aug-2021].

[6] "Dyslexia," NHS choices. [Online]. Available: https://www.nhs.uk/conditions/dyslexia/. [Accessed: 02-Aug-2021].

[7] "Sector specialisms," NCFE. [Online]. Available: https://www.cache.org.uk/news-media/dyslexia-the-facts. [Accessed: 02-Aug-2021].

[8] *بوابة الأهرام*. [Online]. Available: https://gate.ahram.org.eg/News/417438.aspx. [Accessed: 02-Aug-2021].

[9] *ARxVision*. [Online]. Available: Arx.vision. [Accessed: 02-Aug-2021].

[10] "StackPath", [Online]. Available: www.orcam.com/en/myeye2. [Accessed: 02-Aug-2021].

[11] "CanaKit Raspberry 8GB Starter Kit", [Online]. Available: www.amazon.com/CanaKit-Raspberry-8GB-Starter-Kit/dp/B08956GVXN/ref=sr_1_3. [Accessed: 03-Aug-2021].

[12] "raspberry pi camera board v2 8mp", [Online]. Available: makerselectronics.com/product/raspberry-pi-camera-board-v2-8mp. [Accessed: 03-Aug-2021].

[13] "Power Bank", [Online]. Available: egypt.souq.com/eg-ar/powerbank/s/. [Accessed: 03-Aug-2021].

[14] "Object Detection Guide", *Fritz AI*. [Online]. Available: www.fritz.ai/object-detection. [Accessed: 04-July-2022].

[15] "Introduction to YOLO Algorithm for Object Detection", *Section*. [Online]. Available: www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection. [Accessed: 04-July-2022].

[16] A. Geitgey, "Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning", *Behave Your Business*. 24-July-2016. [Online]. Available: medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78. [Accessed: 10-July-2022].

[17] "ViolaJones object detection framework", *Wikipedia*. 12-Dec.-2007. [Online]. Available: en.wikipedia.org/wiki/Viola-Jones_object_detection_framework. [Accessed: 10-July-2022].

[18] N. Dalal και B. Triggs, 'Histograms of oriented gradients for human detection', *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, 2005, τ. 1. 886–893.

[19] https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

[20] V. Kazemi και J. Sullivan, 'One millisecond face alignment with an ensemble of regression trees', *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014. 1867–1874.

[21] “Shear mapping”, *Wikipedia*. 9-July-2004. [Online]. Available: en.wikipedia.org/wiki/Shear_mapping. [Accessed: 10-July-2022].
[22] “Affine transformation”, *Wikipedia*. 25-Feb.-2002. [Online]. Available: en.wikipedia.org/wiki/Affine_transformation. [Accessed: 10-July-2022].
[23] F. Schroff, D. Kalenichenko, και J. Philbin, ‘Facenet: A unified embedding for face recognition and clustering’, *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015. 815–823.
[24] “High Performance Computing Products and Solutions”, *NVIDIA*. [Online]. Available: www.nvidia.com/object/tesla-supercomputing-solutions.html. [Accessed: 11-July-2022].
[25] “JavaScript is not available.”, [Online]. Available: twitter.com/brandondamos/status/757959518433243136. [Accessed: 11-July-2022].
[26] “OpenFace”, [Online]. Available: cmusatyalab.github.io/openface/. [Accessed: 11-July-2022].
[27] “Openface/Models/Openface At Master Cmusatyalab/Openface”, [Online]. Available: github.com/cmusatyalab/openface/tree/master/models/openface. [Accessed: 11-July-2022].
[28] “Support-vector machine”, *Wikipedia*. 27-July-2002. [Online]. Available: en.wikipedia.org/wiki/Support_vector_machine. [Accessed: 11-July-2022].
[29] “ageitgeyOverview”, 11-July-2011. [Online]. Available: github.com/ageitgey/. [Accessed: 11-July-2022].
[30] “GitHub - ageitgey/face_recognition: The world's simplest facial recognition api for Python and the command line”, *Ageitgey/face_recognition: The world's simplest fa*. [Online]. Available: github.com/ageitgey/face_recognition. [Accessed: 11-July-2022].
[31] Murtaza's Workshop - Robotics and AI. *FACE RECOGNITION + ATTENDANCE PROJECT | OpenCV Python | Computer Vision*. (June 11, 2020). Accessed: July 11, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=sz25xxF_AVE
[32] A. Rosebrock, “Find distance from camera to object using Python and OpenCV”, 19-Jan.-2015. [Online]. Available: pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/. [Accessed: 08-July-2022].
[33] G. Halfacree, *The Official Raspberry Pi Beginner’s Guide: How to Use Your New Computer*. Raspberry pi PRESS, 2019.
[34] “Raspberry Pi: Launch Python Script on Startup”, *Instructables*. 28-Apr.-2014. [Online]. Available: www.instructables.com/Raspberry-Pi-Launch-Python-script-on-startup/. [Accessed: 14-July-2022].
[35] L. Pounder, “Raspberry Pi vs Arduino: Which Board is Best?”, *Tom's Hardware*. 10-July-2020. [Online]. Available: www.tomshardware.com/features/raspberry-pi-vs-arduino [Accessed: 14-July-2022].
[36] “Optical character recognition”, *Wikipedia*. 14-Apr.-2002. [Online]. Available: en.wikipedia.org/wiki/Optical_character_recognition. [Accessed: 14-July-2022].
[37] “Tesseract (software)”, *Wikipedia*. 7-Sept.-2006. [Online]. Available: en.wikipedia.org/wiki/Tesseract_(software). [Accessed: 14-July-2022].
[38] S. Thiyagaraj, “A comprehensive guide to OCR with Tesseract, OpenCV and Python”, *NanoNets*. 22-Dec.-2020. [Online]. Available: medium.com/nanonets/a-comprehensive-guide-to-ocr-with-tesseract-opencv-and-python-fd42f69e8ca8. [Accessed: 14-July-2022].
[39] R. Smith, ‘An overview of the Tesseract OCR engine’, *Ninth international conference on document analysis and recognition (ICDAR 2007)*, 2007, 629–633.
[40] C. Hope, “What is Voice Recognition?”, 6-July-2021. [Online]. Available: www.computerhope.com/jargon/v/voicreco.htm. [Accessed: 14-July-2022].
[41] C. Lyden, “What is Speech Recognition Software?”, *CallRail*. [Online]. Available: www.callrail.com/blog/speech-recognition-software/. [Accessed: 14-July-2022].

[42] “What is Voice Recognition? Voice & Speech Recognition Overview RecFaces”, 15-June-2021. [Online]. Available: recfaces.com/articles/what-is-voice-recognition. [Accessed: 14-July-2022].
[43] [Online]. Available: www.makeuseof.com/how-does-voice-recognition-work/. [Accessed: 14-July-2022].
[44] “Kaldi: Kaldi”, [Online]. Available: kaldi-asr.org/doc/index.html. [Accessed: 14-July-2022].
[45] V. C. Rajyaguru, ‘Different Methods Used In Voice Recognition Techniques’, *Int. Res. J. Eng. Technol*, 699–703, 2016
[46] “vosk: Docs, Tutorials, Reviews |”, *Openbase*. [Online]. Available: openbase.com/js/vosk. [Accessed: 14-July-2022].
[47] A. Geitgey, “Machine Learning is Fun Part 6: How to do Speech Recognition with Deep Learning”, *Medium*. 24-Dec.-2016. [Online]. Available: medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a. [Accessed: 15-July-2022].
[48] “Money Images3”, *Google Drive*. [Online]. Available: drive.google.com/drive/folders/1umUl9hc--LNT9Vb-BsKqcDilovNxk9dL. [Accessed: 19-July-2022]
[49] “GitHub - heartexlabs/labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images”, *Heartexlabs/labelImg: LabelImg is a graphical ima*. [Online]. Available: github.com/heartexlabs/labelImg. [Accessed: 19-July-2022]
[50] TheCodingBug. *Darknet YOLOv4 Object Detection Tutorial for Windows 10 on Images, Videos, and Webcams*. (Sept. 23, 2020). Accessed: July 19, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=FE2GBeKuqpc
[51] TheCodingBug. *Build and Install OpenCV With CUDA GPU Support on Windows 10 | OpenCV 4.5.1 | 2021*. (Jan. 21, 2021). Accessed: July 19, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=YsmhKar8oOc
[52] “GitHub - haroonshakeel/yolov4_darknet_custom_object_detection”, *Haroonshakeel/Yolov4_Darknet_Custom_Object_Detecti*. [Online]. Available: github.com/haroonshakeel/yolov4_darknet_custom_object_detection. [Accessed: 19-July-2022].
[53] “OCR Pre-Processing Techniques | Image processing for OCR |”, *Technovators*. 6-Jan.-2021. [Online]. Available: medium.com/technovators/survey-on-image-preprocessing-techniques-to-improve-ocr-accuracy-616ddb931b76. [Accessed: 20-July-2022].
[54] “Pre-Processing in OCR”, *Towards Data Science*. 25-Mar.-2019. [Online]. Available: towardsdatascience.com/pre-processing-in-ocr-fc231c6035a7. [Accessed: 20-July-2022].
[55] “Image histogram”, *Wikipedia*. 10-Sept.-2005. [Online]. Available: en.wikipedia.org/wiki/Image_histogram. [Accessed: 20-July-2022].
[56] “CLAHE and Thresholding in Python”, *Towards Data Science*. 3-July-2020. [Online]. Available: towardsdatascience.com/clahe-and-thresholding-in-python-3bf690303e40. [Accessed: 20-July-2022].
[57] “Image Filters in Python”, *Towards Data Science*. 10-Aug.-2019. [Online]. Available: towardsdatascience.com/image-filters-in-python-26ee938e57d2. [Accessed: 20-July-2022].

Appendices

Appendix A: Questionnaire

1. In your point of view do you think the device is useful?
2. Do you know any company that produce a similar device?
3. What is the expected price for such a device?
4. Would you recommend the device if it is already produced and available in the market?
5. Do you want to add new features to the device?
6. Do you know any institution that would provide the device for free for the incapable patients?
7. In case the device is made by our team would you allow us to test it on your patients?

Appendix B: Main Class

```
import sys
sys.path.append('../')
sys.path.append(r'/home/pi/.local/lib/python3.7/site-packages/')
from voice_recognition.voice_recognition import *
from Factory.Factory import *
from voice_recognition.voice_recognition import *
from text_to_speech.text_to_speech import text_to_speech as tts
from camera.camera import *
import cv2 as cv
import os


def get_name(vr):
    """
    this method is made to interact with the user to get the name of the person to be
registered.

    :param vr: voice recognition object
    :return: name of the person to be registered
    """
    # output voice asking about the person name.
    speak("what is the person name?", "register_error")
    # getting name from microphone
    name = vr.name()
    # conforming the person name
    speak(f"the name you entered {name} for conformation say yes to try again say
no", "test_name")
    # getting conformation answer
    conform = vr.conform()
    # if the conformation answer is yes return name of the person
    if conform == 'yes':
        return name
    # otherwise loop again till we get the correct name.
    else:
        return get_name(vr)

def speak(text, status):
    """
    this method is used to speak to the user through the earphones or speakers.

    :param text: text to be said in earphones
    :param status: name of the audio to be saved in the outputs_mp3 folder
    :return:
    """
    # creating a text to speech object
    speech = tts(text, status)
    # saving mp3 audio in outputs_mp3 folder
    speech.generate_mp3()
    # play the audio file
    speech.play_mp3()
    # deleting the object after usage
    del speech
```

```
def main():
    # creating object from factory class
    factory = Factory.get_instance()
    # creating object form voice recognition class
    vr = voice_recognition()
    # creating object from the camera class
    cam = camera.get_instance()


    while True:
        # waiting and listening until hot keyword (Be my eye) is said by the user to
start.
        vr.start()
        # output voice welcoming the user.
        speak("Welcome to A.EYE", "welcome")
        while True:
            # output voice asking the user to choose a command
            speak("Choose a mode.", "welcome")

            # getting command from the user
            # commands available(object,read,people,money,sleep now, shutdown)
            mode = vr.mode()
            mode = mode.lower()

            # if the command wasn't recognized, inform the user then loop again.
            if "try again" in mode:
                speak(f"I do not know what{mode.split(':')[1]} means", "mode_error")
                continue

            # if the command is sleep now break from the inner loop and wait for the
hot keyword again.
            if "sleep now" in mode:
                speak("bye for now", "sleep")
                break

            # if the command is shutdown then the os system (linux in our case) shall
shutdown.
            if "shutdown" in mode:
                speak("bye bye", "shutdown")
                os.system("sudo shutdown -h now")

            # otherwise send mode to the factory to get the object required.
            obj = factory.get_objects(mode)


            # take a photo from the camera
            im = cam.snapshot()

            # if the mode was register
            # interact with the user to get the person name and save this person.
            if 'register' in mode:
                # start registering and saving person's face
                success = obj.register.train(cv.cvtColor(im, cv.COLOR_BGR2RGB))
```

```
                # inform user about the registration status (successful or not)
                if not success:
                    speak("there is no face detected in the picture registration
failed", "register_error")
                else:
                    # getting person's name
                    name = get_name(vr)
                    line = f'{len(obj.names)+1}, {name} \n'
                    obj.names.append(line)
                    # saving name of the person
                    with
open('/home/pi/Desktop/Organized/Face_Recognition/people/people.txt', 'a') as file:
                        file.write(line)

            # in case of register mode is successful it will inform the user of the
person name and id for endurance
            # otherwise input the image to the object and get output text.
            obj.get_input(im)
            t = obj.output()
            # output voice of the object result.
            speak(t, obj.name)


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

Appendix C: Factory Pattern

```
import sys

sys.path.append("../")
from Object_Recognition.object_recognition import *
from text_recognition.OCR import *
from Face_Recognition.FaceRecognition import *
from Money_Recognition.MR import *

class Factory:
    __instance = None  # singleton instance

    def __init__(self):
        """
        constructor of the Factory class
        """
        if Factory.__instance != None:
            raise Exception("Singleton can not be instantiated more than once!")
        else:
            Factory.__instance = self
            self.objects = None

    @staticmethod
    def get_instance():
        """
        this method is used to return a singleton instance of the factory class, and
ensure that only one instance is created.
        :return: Singleton instance of the Factory class
        """
        if Factory.__instance == None:
            Factory()
        return Factory.__instance

    def get_objects(self, inp):
        """
        This method is the essence of the factory class where it returns an instances
of other classes (classes in concern) based on its input.
        :param inp: data provided to the factory for it to decide which instance to
create.
        :return: returns a list of objects required based on the input provided
        """
        # deciding which object to create based on the data provided in inp variable
        if "object" in inp:
            self.objects=object_detection.get_instance()

        elif "read" in inp:
            self.objects=OCR.get_instance()

        elif "people" in inp or "register" in inp:
            self.objects=FaceRecognition.get_instance()

        elif "money" in inp:
            self.objects= money_recognition.get_instance()
        return self.objects
```

Appendix D: Voice Recognition Model

```
from vosk import Model, KaldiRecognizer
import pyaudio

class voice_recognition():
    def __init__(self):
        """
        constructor of the voice_recognition class
        """
        #load vosk model
        model = Model(r"/home/pi/Desktop/Organized/voice_recognition/vosk-model-
small-en-us-0.15")
        frequency = 16000
        # define recognizer to use the model and pass the frequency used in the audio
        self.recognizer = KaldiRecognizer(model, frequency)
        # create pyaudio object
        self.cap = pyaudio.PyAudio()
        # define audio parameters and connect to microphone device
        self.stream = self.cap.open(format=pyaudio.paInt16, channels=1,
rate=frequency, input=True, frames_per_buffer=8192)


    def name(self):
        """
        this method is used to capture name of the person to be registered from the
microphone
        :return: name of the person to be registered
        """
        # start stream (listening)
        self.stream.start_stream()
        while True:
            # read stream data
            data = self.stream.read(4096, exception_on_overflow=False)
            name = ""
            # use recognizer to convert data to text
            # if there is a result pass it to name variable
            if self.recognizer.AcceptWaveform(data):
                text = self.recognizer.Result()
                text = eval(text)
                name = text['text']
                # stop stream
                self.stream.stop_stream()
                return name
```

```
    def conform(self):
        """
        this method is used to get conformation from the user about the person to be
registered name
        if the name is correct the user should say yes otherwise the user should say
no
        :return: conformation result (yes or no)
        """
        # start stream (listening)
        self.stream.start_stream()
        while True:
            # read stream data
            data = self.stream.read(4096, exception_on_overflow=False)
            conform = ""
            # use recognizer to convert data to text
            # if there is a result pass it to conform variable
            if self.recognizer.AcceptWaveform(data):
                text = self.recognizer.Result()
                text = eval(text)
                conform = text['text']
                conform.lower()
            # ensuring that conform variable is either yes or no otherwise loop again
            if "yes" in conform:
                conform='yes'
            elif "no" in conform:
                conform = 'no'
            else:
                conform = ''
            # once conform is either yes or no return the conformation result
            if conform:
                #stop stream
                self.stream.stop_stream()
                return conform

    def start(self):
        """
        this method is used at the beginning to listen for the hot keyword (be my
eye) to start mode operations
        :return: None
        """
        self.stream.start_stream()
        while True:
            data = self.stream.read(4096, exception_on_overflow=False)
            text =""

            if self.recognizer.AcceptWaveform(data):
                text = self.recognizer.Result()
                text = eval(text)
                text = text['text'].lower()

            # if hot keyword (be my eye) detected then return
            if text=="be my eye":
                self.stream.stop_stream()
                return
```

```
    def mode(self):
        """
        this method is used to get which mode operation the user wants from the
microphone
        :return: required mode (modes available:  object, people, read, money)
        """
        self.stream.start_stream()
        while True:
            data = self.stream.read(4096, exception_on_overflow=False)
            modes_detected = ""
            if self.recognizer.AcceptWaveform(data):
                text = self.recognizer.Result()
                text = eval(text)
                text = text['text'].lower()
                # check which mode the user choose
                if "object" in text:
                    modes_detected += "object"

                elif "people" in text:
                    modes_detected += "people"

                elif "read" in text:
                    modes_detected += "read"

                elif "register" in text:
                    modes_detected += "register"

                elif "money" in text:
                    modes_detected += "money"

                elif "sleep now" in text:
                    modes_detected += "sleep now"

                elif "shutdown" in text.replace(" ", ""):
                    modes_detected += "shutdown"

                # if no mode has been recognized. return try again with text of what
the user has said
                else:
                    if not text == "":
                        modes_detected += "Try again:" + text

            # to ensure that something (a command) has been said by the user. (to
prevent empty text)
            if modes_detected:
                #stop stream
                self.stream.stop_stream()

                return modes_detected
```

Appendix E: Camera Class

```
import cv2 as cv
import numpy as np
import os


class camera:
    __instance = None  # singleton instance

    def __init__(self):
        """
        constructor of the camera class
        """
        if camera.__instance != None:
            raise Exception("Singleton can not be instantiated more than once!")
        else:
            self.frames = []
            camera.__instance = self

    @staticmethod
    def get_instance():
        """
        this method is used to return a singleton instance of the camera class, and
ensure that only one instance is created.
        :return: Singleton instance of the Factory class
        """
        if camera.__instance == None:
            camera.__instance = camera()
        return camera.__instance

    def snapshot(self):
        """
        this method is used to take a photo from camera
        :return: image (the least blurry frame)
        """
        # clear frames list from previous use
        self.frames.clear()
        # creating a video capture object
        self.vid = cv.VideoCapture(0)  # 0 means reading from default camera
connected to the system

        # setting frame width, height, and frame per second parameters
        self.vid.set(cv.CAP_PROP_FRAME_WIDTH, 640)
        self.vid.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
        self.vid.set(cv.CAP_PROP_FPS, 20)

        while (True):

            # capturing frames
            ret, frame = self.vid.read()

            # if frame captured successfully store it in the frame list
            if ret:
                self.frames.append(frame)
```

```
            # if number of frames stored is 10 break and stop  capturing frames
            if len(self.frames) == 10:
                break

        # After the loop release the video capture object
        self.vid.release()

        # calculating blur values and return the least blurry frame
        blur_values = []
        for frame in self.frames:
            blur_values.append(self.blur_calculations(frame))

        return self.frames[blur_values.index(max(blur_values))]

    def blur_calculations(self, img):
        """
        this method is used to calculate blur values (Laplacian variance)
        :param img: frame taken from camera
        :return: blur value of the frame
        """
        lap_var = cv.Laplacian(img, cv.CV_64F).var()
        return lap_var
```

Appendix F: Object Detection Model

```
import cv2 as cv
import numpy as np
import sys

sys.path.append('../')
from Object_Recognition.Object_Distance import Object_Distance as od

class object_detection:
    # names of the objects available for detection (coco dataset).
    classes_names = ['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus',
'train',
                     'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign',
'parking meter',
                     'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
'elephant', 'bear',
                     'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie',
'suitcase',
                     'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball
bat',
                     'baseball glove', 'skateboard', 'surfboard', 'tennis racket',
'bottle',
                     'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana',
'apple',
                     'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
'donut',
                     'cake', 'chair', 'sofa', 'pottedplant', 'bed', 'diningtable',
'toilet',
                     'tvmonitor', 'laptop', 'mouse', 'remote', 'keyboard', 'cell
phone',
                     'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book',
'clock',
                     'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']

    __instance = None  # Singleton instance

    def __init__(self):
        """
        constructor of the object_recognition class.
        """
        if object_detection.__instance != None:
            raise Exception("Singleton can not be instantiated more then once!")
        else:
            object_detection.__instance = self
            self.name = 'object'

            # list of tuples which contains objects detected and their distance from
the camera
            self.objects_detected = []
            # list of objects center x coordinates in the image to specify the object
in which direction (left, front, right).
            self.objects_center = []

            self.confThreshold = 0.5  # confidence threshold
            self.nmsThreshold = 0.5  # non-maximum suppression threshold
```

```
            # path to the model configuration
            model_configuration =
r"/home/pi/Desktop/Organized/Object_Recognition/yolov4-tiny.cfg"
            # path to model weights
            model_weights = r"/home/pi/Desktop/Organized/Object_Recognition/yolov4-
tiny.weights"

            # construct a network from model configuration and weights
            self.network = cv.dnn.readNetFromDarknet(model_configuration,
model_weights)

            # declaring that we are going to use opencv as the backend
            self.network.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
            # declaring that we are going to use CPU. (CPU or GPU)
            self.network.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)

    @staticmethod
    def get_instance():
        """
        this method is used to return a singleton instance of the factory class, and
ensure that only one instance is created.
        :return: Singleton instance of the object_recognition class
        """
        if object_detection.__instance == None:
            object_detection()
        return object_detection.__instance

    def get_input(self, img):
        """
        This method is responsible for accepting the input image and prepare it for
output processing
        :param img: image taken from camera.
        :return: None
        """
        # loading image in our class.
        self.img = img

        # convert the image into a Blob image format for the network input
        blob_img = cv.dnn.blobFromImage(self.img, 1 / 255, (416, 416), [0, 0, 0], 1,
crop=False)

        # pass the image into the network input
        self.network.setInput(blob_img)

    def output(self):
        """
        this method is responsible for processing the image in the network and gives
an info text about the objects
        detected in the image and their distances from the camera.
        :return: text info of objects detected and their distances from the camera
        """
        # getting names of all network layers.
        layers_names = self.network.getLayerNames()
```

```
        # getting names of output layers.
        outputNames = [layers_names[i[0] - 1] for i in
self.network.getUnconnectedOutLayers()]
        # getting the result of each output layer
        outputs = self.network.forward(outputNames)

        self.__find_objects(outputs)

        text = self.__get_text()
        # clearing lists for the next process
        self.objects_detected.clear()
        self.objects_center.clear()
        if not text:
            text += "no objects detected."

        text = "Object result: " + text

        return text


    def __find_objects(self, outputs):
        """
        This method is responsible for detecting objects and their x coordinate of
their center,
         and provide object width in pixels for object distance calculation
        :param outputs: network output information
        :return: None
        """
        # extracting image shape hT height, wT width, c channels
        hT, wT, c = self.img.shape
        # list to contain bounding boxes (x, y, w, h)
        bbox = []
        # list to contain indices of the max confidence values
        classIds = []
        # list of the max confidence values that exceeded the confidence threshold
        confs = []

        for output in outputs:
            for det in output:
                # getting confidence values for each object
                scores = det[5:]
                # getting index of the max confidence value
                classId = np.argmax(scores)
                # storing the confidence value
                confidence = scores[classId]
                # if the max confidence value exceeds the threshold, it is worth
mentioning
```

```
                if confidence > self.confThreshold:
                    # getting width w, height h, and center x,y coordinates
                    # note that the values are in percentage therefore we multiply it
by width and height to get the actual value
                    w, h = int(det[2] * wT), int(det[3] * hT)
                    x, y = int(det[0] * wT), int(det[1] * hT)
                    # you can uncomment the below lines to move the center point to
the left upper corner for drawing purposes.
                    # x  = int(x - w/2)
                    # y= int(y-h/2)
                    bbox.append([x, y, w, h])
                    classIds.append(classId)
                    confs.append(float(confidence))

        #applying Non-maximum suppression
        indices = cv.dnn.NMSBoxes(bbox, confs, self.confThreshold, self.nmsThreshold)
        for i in indices:
            i = i[0]
            box = bbox[i]
            x, y, w, h = box[0], box[1], box[2], box[3]
            # you can draw bounding boxes in the image here for testing purposes
            #
            #

            #calculating object distance from the camera.
            obdist = od(w, self.classes_names[classIds[i]])
            dist = obdist.calculate()

            # storing objects detected and there distance from the camera, and x
coordinate of object center
            self.objects_detected.append((self.classes_names[classIds[i]], dist))
            self.objects_center.append(x)

        # you can show image here after the bounding boxes have been drawn
        #
        #

    def __get_text(self):
        """
        this method is used to generate text containing info about objects detected,
their directions
        and distances from the camera
        :return: info text about objects detected, their directions and distances
from the camera
        """
        # text responsible for the right area of the image
        string_right = ""
        # text responsible for the center area of the image
        string_cen = ""
        # text responsible for the left area of the image
        string_left = ""

        # left area boundaries
        x_left = img_shape[1] / 4
```

```
        # center area boundaries
        x_cen = x_left + x_left * 2

        # checking if there was any objects detected first. if there is no objects
return empty text.
        if not objects_detected:
            text = ""
            return text

        # getting each object name and x coordinate of its center.
        for obj, x in zip(objects_detected, objects_center):
            # if x coordinate is larger than the center boundaries then the object at
the right area.
            if x > x_cen:
                string_right += f"{obj[0]} at distance {obj[1]} meters, "
            # if it's less than the center and larger than left boundaries then the
object in the center area.
            elif x > x_left:
                string_cen += f"{obj[0]} at distance {obj[1]} meters, "
            # otherwise the object will be in the left area
            else:
                string_left += f"{obj[0]} at distance {obj[1]} meters, "

        # constructing text of the right area.
        if string_right:
            string_right += "on your right"
        # constructing text of the center area.
        if string_cen:
            if string_right:
                string_right += " ,and "
            string_cen += "front of you"
        # constructing text of the left area.
        if string_left:
            if string_cen:
                string_cen += " ,and "
            string_left += "on your left"

        # constructing text to be returned
        text = "there is a " + string_right + string_cen + string_left
        return text
```

```python
import math

class Object_Distance:
    # focal length of camera
    F = 877.88

    # average width of all objects in concern (will need some updates)
    ObjectWidth = {'person':45,'bicycle':180,'car':175,'motorbike':271,
    'bus':250,'train':320,'truck':260,'bottle': 9,'traffic light':35,
    'fire hydrant':18,'stop sign':76,'boat':563,'book':14,'aeroplane':6440,
    'apple':8,'parking meter':240,'bench':45,'bird':25,'cat':47.5,
    'dog':64.5,'horse':190,'sheep':117,'cow':245,'elephant':500 ,
    'bear':305,'zebra':200,'giraffe':240,'backpack':23,'umbrella':110,
    'handbag':18,'tie':9.5,'suitcase':42,'frisbee':22.5,'skis':160,
    'snowboard':24.5,'sports ball':19,'kite':80,'baseball bat':86.4,
    'baseball glove':3.75,'skateboard':20,'surfboard':50,'fork':20,
    'tennis racket':27,'wine glass':10.8,'cup':11,'knife':33,'sofa':227,
    'spoon':16,'bowl':25,'banana':13,'sandwich':12,'orange':8,'pizza':33,
    'broccoli':46,'carrot':20,'hot dog':15,'donut':9,'cake':27.5,
    'chair':50,'pottedplant':18.5,'bed':76,'diningtable':206,'toilet':51,
    'tvmonitor':132,'laptop':39,'mouse':10,'remote':4,'keyboard':40,
    'cell phone':7,'microwave':50,'oven':60,'toaster':40,'sink':82.5,
    'refrigerator':85,'clock':40,'vase':20,'scissors':22,'teddy bear':25,
    'hair drier':23,'toothbrush':13}


    def __init__(self, P , obj):
        """
        constructor of Object_Distance class
        :param P: width of object in pixels.
        :param obj: name of object detected in image.
        """
        self.P = P
        self.obj = obj
        # getting average width of the object
        self.W = self.ObjectWidth[obj]

    def calculate(self):
        """
        this method is used to calculate the distance of the object from the camera.
        :return: distance between the object detected and the camera.
        """
        # calculating distance in cm
        self.D = (self.W  * self.F) / (self.P)

        # returning distance in m
        if self.D >=100:
            return str(round(self.D/100,1))
        else:
            return str(round(self.D/100,2))
```

Appendix G: Face Recognition Model

```
import numpy as np
import face_recognition as frc
import cv2 as cv
import sys
sys.path.append('../')
from Face_Recognition.Face_Register import *

class FaceRecognition:

    __instance = None

    def __init__(self):
        if FaceRecognition.__instance != None:
            raise Exception("Singleton can not be instantiated more than once!")
        else:
            try:
                self.people =
np.load('/home/pi/Desktop/Organized/Face_Recognition/people/people.npy',
allow_pickle=False)
            except:
                print("error 1")
                self.people = np.array([])

            self.names = []
            with
open('/home/pi/Desktop/Organized/Face_Recognition/people/people.txt', 'r+') as file:
                x = file.readlines()
                for line in x:
                    self.names.append(line)

            self.name = "people"
            self.register = register(self.people)
            FaceRecognition.__instance = self

    @staticmethod
    def get_instance():
        if FaceRecognition.__instance == None:
            FaceRecognition.__instance = FaceRecognition()
        return FaceRecognition.__instance

    def get_input(self, image):
        """Method to get image as input

        :param image: numpy image used as input to the model
        :type image: ndarray
        :returns: None
        """
        self.numpyImage = image
        self.numpyImage = cv.cvtColor(image, cv.COLOR_BGR2RGB)
        self.people =
np.load('/home/pi/Desktop/Organized/Face_Recognition/people/people.npy',
allow_pickle=False)
        self.register.people = self.people
```

```
def output(self):
    """Test method used to recognize known personnals

    :returns: text: text describing the result of the recognition model
    """
    faces = frc.face_locations(self.numpyImage)
    encodings = frc.face_encodings(self.numpyImage, faces)

    matches = None
    text = ''
    for codec, face in zip(encodings, faces):
        matches = frc.compare_faces(self.people, codec) #threshold
        print(matches)

        for i in range(len(matches)):
            if matches[i]:
                text += self.names[i]
    if text == '':
        text = 'no face has been recognized'

    text = "people result: " + text
    print(text)
    return text
```

```
import numpy as np
import face_recognition as frc


class register:

    def __init__(self, people):
        self.people = people  #np.load('people/people.npy', allow_pickle=False)

    def train(self, numpyImage):
        """Train method used to recognize and add encodings of new faces

        :param numpyImage: the numpy array equivalent of an RGB camera snapshot
        :type numpyImage: ndarray
        :returns: None
        """
        faces = frc.face_locations(numpyImage)
        try:
            encodings = frc.face_encodings(numpyImage, faces)[0]
        except:
            return 0

        encodings = np.array(encodings).reshape(1, 128)
        print(encodings.shape)
        #print(self.people.shape)
        print(self.people)

        if np.size(self.people) == 0:
            print("emptyyyyyy")
            self.people = encodings
        else:
            print("not emptyyyyyy")
            self.people = np.vstack((self.people, encodings)) # When resetting
people.npy file, comment out this file

        np.save('/home/pi/Desktop/Organized/Face_Recognition/people/people.npy',
self.people, allow_pickle=False)
        return 1
```

Appendix H: Money Recognition Model

```
import cv2 as cv
import numpy as np
import sys

sys.path.append('../')

class money_recognition:
    # names of the money objects.
    classes_names = ['0.25', '0.5', '1', '5', '10', '20', '50','100', '200']

    __instance = None  # Singleton instance

    def __init__(self):
        """
        constructor of the money_recognition class.
        """

        if money_recognition.__instance != None:
            raise Exception("Singleton can not be instantiated more then once!")
        else:
            money_recognition.__instance = self
            self.name = 'money'

            self.confThreshold = 0.5  # confidence threshold
            self.nmsThreshold = 0.5  # non-maximum suppression threshold

            # path to the model configuration
            model_configuration =
r"/home/pi/Desktop/Organized/Money_Recognition/yolov4-obj.cfg"
            # path to model weights
            model_weights = r"/home/pi/Desktop/Organized/Money_Recognition/yolov4-
obj_last.weights"

            # construct a network from model configuration and weights
            self.network = cv.dnn.readNetFromDarknet(model_configuration,
model_weights)

            # declaring that we are going to use opencv as the backend
            self.network.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
            # declaring that we are going to use CPU. (CPU or GPU)
            self.network.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)

    @staticmethod
    def get_instance():
        """
        this method is used to return a singleton instance of the money_recognition
class, and ensure that only one instance is created.
        :return: Singleton instance of the money_recognition class
        """
        if money_recognition.__instance == None:
            money_recognition()
        return money_recognition.__instance
```

```
    def get_input(self, img):
        """
        This method is responsible for accepting the input image and prepare it for
output processing
        :param img: image taken from camera.
        :return: None
        """
        # loading image in our class.
        self.img = img

        # convert the image into a Blob image format for the network input
        blob_img = cv.dnn.blobFromImage(self.img, 1 / 255, (416, 416), [0, 0, 0], 1,
crop=False)

        # pass the image into the network input
        self.network.setInput(blob_img)

    def output(self):
        """
        this method is responsible for processing the image in the network and gives
an info text about the money
        detected in the image.
        :return: text of the money detected in the image
        """
        # getting names of all network layers.
        layers_names = self.network.getLayerNames()

        # getting names of output layers.
        outputNames = [layers_names[i[0] - 1] for i in
self.network.getUnconnectedOutLayers()]
        # getting the result of each output layer
        outputs = self.network.forward(outputNames)

        text = self.__get_text(self.__find_money(outputs))
        if not text:
            text += "no money detected."

        text = "Money result: " + text

        return text
```

```
    def __find_money(self, outputs):
        """
        This method is responsible money detection in the image
        :param outputs: network output information
        :return: list of money detected in the image
        """
        # extracting image shape hT height, wT width, c channels
        hT, wT, c = self.img.shape
        # list to contain bounding boxes (x, y, w, h)
        bbox = []
        # list to contain indices of the max confidence values
        classIds = []
        # list of the max confidence values that exceeded the confidence threshold
        confs = []

        for output in outputs:
            for det in output:
                # getting confidence values for each object
                scores = det[5:]
                # getting index of the max confidence value
                classId = np.argmax(scores)
                # storing the confidence value
                confidence = scores[classId]
                # if the max confidence value exceeds the threshold, it is worth
mentioning
                if confidence > self.confThreshold:
                    # getting width w, height h, and center x,y coordinates
                    # note that the values are in percentage therefore we multiply it
by width and height to get the actual value
                    w, h = int(det[2] * wT), int(det[3] * hT)
                    x, y = int(det[0] * wT), int(det[1] * hT)
                    # you can uncomment the below lines to move the center point to
the left upper corner for drawing purposes.
                    # x  = int(x - w/2)
                    # y= int(y-h/2)
                    bbox.append([x, y, w, h])
                    classIds.append(classId)
                    confs.append(float(confidence))

        #applying Non-maximum suppression
        indices = cv.dnn.NMSBoxes(bbox, confs, self.confThreshold, self.nmsThreshold)
        #list to contain money detected in an image
        money_detected = []
        # filling money_detected list
        for i in indices:
            i = i[0]
            money_detected.append(self.classes_names[classIds[i]])

        return money_detected
```

```python
def __get_text(self, money_list):
    """
    this method is used to construct a text of the money detected in an image.
    :param money_list: money detected list
    :return: text containing information about the money detected in the image
    """
    text = ''
    for money in money_list:
        text += f'{money} EGP, '

    if money_list:
        text = 'I have recognized ' + text

    return text
```

Appendix I: Text Recognition Model

```
import cv2
import pytesseract
import numpy as np
from scipy.ndimage import interpolation as inter

class OCR:

    __instance = None

    def __init__(self):
        if OCR.__instance !=None:
            raise Exception("Singleton can not be instantiated more than once!")
        else:
            OCR.__instance = self

    @staticmethod
    def get_instance():
        if OCR.__instance == None:
            OCR.__instance = OCR()
        return OCR.__instance


    def resize(self, img):
        """Method to grayscale the input image and resize it over 300 DPI

        :param img: image taken as input
        :type img: ndarray
        :returns: img: grayscaled and resized image
        """
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_LINEAR)
        return img

    def binarize(self, img):
        """Method to binarize gray image using thresholding

        :param img: image from the previous pipeline phase
        :type img: ndarray
        :returns: img: binarized image using CLAHE histogram and thresholding
        """
        clahe = cv2.createCLAHE(clipLimit =1.5, tileGridSize=(8,8))
        img = clahe.apply(img)
        # Play in the parameters here. A LOT!!!
        # img = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 7, 7)
        ret, img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
        return img
```

```
def __find_score(self, arr, angle):
    """Method to find rotation score used by the skewing method

    :param arr: image from previous pipeline stage in array format
    :type arr: ndarray
    :param angle: slope angle of the image
    :type angle: ndarray
    :returns:
        - hist
        - score
    """
    data = inter.rotate(arr, angle, reshape=False, order=0)
    hist = np.sum(data, axis=1)
    score = np.sum((hist[1:] - hist[:-1]) ** 2)
    return hist, score

def skew(self, img):
    """Method to skew sloped images

    :param img: image taken as input from previous pipeline stage
    :type img: ndarray
    :returns: data: image after being skewed
    """
    delta = 1
    limit = 5
    angles = np.arange(-limit, limit+delta, delta)
    scores = []
    for angle in angles:
        hist, score = self.__find_score(img, angle)
        scores.append(score)
    best_score = max(scores)
    best_angle = angles[scores.index(best_score)]
    # correct skew
    data = inter.rotate(img, best_angle, reshape=False, order=0)
    return data

def removeNoise(self, img):
    """Method to remove noise by blurring filters

    :param img: image taken as input from pipeline
    :type img: ndarray
    :returns: img: image after being blurred and denoised
    """
    kernel = np.ones((1, 1), np.uint8)
    img = cv2.filter2D(img, -1, kernel)
    img = cv2.dilate(img, kernel, iterations=1)
    img = cv2.erode(img, kernel, iterations=1)
    img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
    img = cv2.medianBlur(img, 1)
    return img
```

```
    def remove_borders(self, image):
        """Method to remove unnecessary borders from image

        :param image: image taken as input from pipeline
        :type img: ndarray
        :returns: crop: image after being borders-croped
        """
        contours, hierarchy = cv2.findContours(image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        contours = sorted(contours, key = lambda x:cv2.contourArea(x))
        cnt = contours[-1]
        x,y,w,h = cv2.boundingRect(cnt)
        crop = image[y:y+h, x:x+w]
        return (crop)


    def get_input(self, img):
        """Method to call from object to get input to the class

        :param img: image taken as input
        :type img: ndarray
        :returns: None
        """
        self.img = img

    def output(self):
        """Pipeline used to recognize and extract text from image

        :returns: text: text recognized by tesseract
        """
        self.img = self.resize(self.img)
        self.img = self.binarize(self.img)
        self.img = self.skew(self.img)
        self.img = self.removeNoise(self.img)
        self.img = self.remove_borders(self.img)
        custom_config = r'--oem {} --psm {}'.format(1, 3)
        text = pytesseract.image_to_string(self.img, lang='eng',
config=custom_config)
        text = "Read result: " + text
        return text
```

Appendix J: Text-to-Speech Model

```
from gtts import gTTS
import os, subprocess


class text_to_speech:

    def __init__(self, text, name, language='en'):
        """
        constructor of text to speech class
        :param text: text to be converted to audio
        :param name: name of the audio file to be saved
        :param language: language of the text (English by default)
        """
        self.text = text
        self.language = language
        self.output_filename = f'../output_mp3/output_{name}.mp3'

    def generate_mp3(self):
        """
        generate an audio file of the text and save the audio as an mp3 file in the
outputs_mp3 folder
        :return: None
        """"""
        self.output = gTTS(text=self.text, lang=self.language, slow=False)
        self.output.save(self.output_filename)

    def play_mp3(self):
        """
        play mp3 audio file using mpv player.
        :return:
        """
        os.system(f'mpv {self.output_filename}')
```