MICROPROCESSOR SYSTEMS
LABORATORY PROJECT

Facial recognition

STUDENTS:
Alexandru RACOVAN
Stefania SANTIMBREAN
3rd year, CTI ENG

Index

I. Requirements

Create an IoT application using Raspberry Pi and Python.

II. Introduction

The project presented is a client-server application which detects the face and eyes through a webcam.

The raspberry pi has a webcam connected to it. When the application starts, video streaming is captured by the webcam and the data is sent to the server, which processes it and send tweets and logs off the user from the computer, if it recognizes the object to be a face.

III. Python - programming language

A. Why use it?

The programming language which was used for the app is Python because community provides many introductory resources and it is a very accessible language.

B. About

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

C. Portable

Due to its open-source nature, Python has been ported to (i.e. changed to make it work on) many platforms. All your Python programs can work on any of these platforms without requiring any changes at all if you are careful enough to avoid any system-dependent features.

You can use Python on GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC!
You can even use a platform like Kivy to create games for your computer and for iPhone, iPad, and Android.

IV. Raspberry Pi

A Raspberry Pi is a credit card-sized computer originally designed for education, inspired by the 1981 BBC Micro.

The Raspberry Pi Foundation was established in 2008 as a UK-based charity with the purpose to "further the advancement of education of adults and children, particularly in the field of computers, computer science and related subjects".

The catalyst for starting Raspberry Pi was a drop in applications for the Cambridge University undergraduate computer science degree in the early 2000s. That was a symptom of a much broader challenge. For decades, schools had taught how to use computer programs, not how to make them. At the same time, digital technologies had become less hackable. The result was that, whether at school or at home, too many of us had become digital consumers rather than digital makers.

Their co-founders were inspired to create a product: a low-cost programmable computer that would introduce young people to computing. Between launching their first product in February 2012 and December 2017, they sold over 17 million Raspberry Pi computers.

The Raspberry Pi was designed for the Linux operating system, and many Linux distributions now have a version optimized for the Raspberry Pi.

Two of the most popular options are Raspbian, which is based on the Debian operating system, and Pidora, which is based on the Fedora operating system. For beginners, either of these two work well; which one you choose to use is a matter of personal preference. A good practice might be to go with the one which most closely resembles an operating system you're familiar with, in either a desktop or server environment.

V. Client-Server Model

The client-server model describes how a server provides resources and services to one or more clients. Examples of servers include web servers, mail servers, and file servers. Each of these servers provide resources to client devices, such as desktop computers, laptops, tablets, and smartphones. Most servers have a one-to-many relationship with clients, meaning a single server can provide resources to multiple clients at one time.

When a client requests a connection to a server, the server can either accept or reject the connection. If the connection is accepted, the server establishes and maintains a connection with the client over a specific protocol. For example, an email client may request an SMTP (Simple Mail Transfer Protocol) connection to a mail server in order to send a message. The SMTP application on the mail server will then request authentication from the client, such as the email address and password. If these credentials match an account on the mail server, the server will send the email to the intended recipient.

Online multiplayer gaming also uses the client-server model. One example is Blizzard's Battle.net service, which hosts online games for World of Warcraft, StarCraft, Overwatch, and others. When players open a Blizzard application, the game client automatically connects to a Battle.net server. Once players log in to Battle.net, they can see who else is online, chat with other players, and play matches with or against other gamers.

While Internet servers typically provide connections to multiple clients at a time, each physical machine can only handle so much traffic. Therefore, popular online services distribute clients across multiple physical servers, using a technique called distributed computing. In most cases, it does not matter which specific machine users are connected to, since the servers all provide the same service.

VI. Network Socket

A. About

A socket is one of the most fundamental technologies of computer network programming. Sockets allow network software applications to communicate using standard mechanisms built into network hardware and operating systems.

Although it might sound like just another feature of Internet software development, socket technology existed long before the Web. And, many of today's most popular network software applications rely on sockets.

B. What Sockets can do for your network

A socket represents a single connection between exactly two pieces of software (a so-called point-to-point connection). More than two pieces of software can communicate with client/server or distributed systems by using multiple sockets. For example, many Web browsers can simultaneously communicate with a single Web server via a group of sockets made on the server.

Socket-based software usually runs on two separate computers on the network, but sockets can also be used to communicate locally (interprocess) on a single computer. Sockets are bidirectional, meaning that either side of the connection is capable of both sending and receiving data. Sometimes the one application that initiates communication is termed the "client" and the other application the "server", but this terminology leads to confusion in peer to peer networking and should generally be avoided.

C. Socket APIs and Libraries

Several libraries that implement standard application programming interfaces (APIs) exist on the Internet. The first mainstream package - the Berkeley Socket Library is still widely in use on UNIX systems. Another very common API is the Windows Sockets (WinSock) library for Microsoft operating systems.

Relative to other computer technologies, socket APIs are quite mature: WinSock has been in use since 1993 and Berkeley sockets since 1982.

The socket APIs are relatively small and simple. Many of the functions are similar to those used in file input/output routines such as <tt>read()</tt>, <tt>write()</tt>, and <tt>close()</tt>. The actual function calls to use depend on the programming language and socket library chosen.

D. Socket interface Types

Socket interfaces can be divided into three categories:

- Stream sockets, the most common type, requires that the two communicating parties first establish a socket connection, after which any data passed through that connection will be guaranteed to arrive in the same order in which it was sent - so-called connection-oriented programming model.
- Datagram sockets offer "connection-less" semantics. With datagrams, connections are implicit rather than explicit as with streams. Either party simply sends datagrams as needed and waits for the other to respond; messages can be lost in transmission or received out of order, but it is the application's responsibility and not the sockets to deal with these problems. Implementing datagram sockets can give some applications a performance boost and additional flexibility

compared to using stream sockets, justifying their use in some situations.
- The third type of socket -- the raw socket -- bypasses the library's built-in support for standard protocols like TCP and UDP. Raw sockets are used for custom low-level protocol development.

E. Socket Support in Network Protocols

Modern network sockets are typically used in conjunction with the Internet protocols -- IP, TCP, and UDP. Libraries implementing sockets for Internet Protocol use TCP for streams, UDP for datagrams, and IP itself for raw sockets.

To communicate over the Internet, IP socket libraries use the IP address to identify specific computers. Many parts of the Internet work with naming services, so that the users and socket programmers can work with computers by name (e.g., "thiscomputer.wireless.about.com") instead of by address (e.g., 208.185.127.40).

Stream and datagram sockets also use IP port numbers to distinguish multiple applications from each other. For example, Web browsers on the Internet know to use port 80 as the default for socket communications with Web servers.

VII. Transmission Control Protocol/Internet Protocol (TCP/IP)

A. About

TCP/IP, or the Transmission Control Protocol/Internet Protocol, is a suite of communication protocols used to interconnect network devices on the internet. TCP/IP can also be used as a communications protocol in a private network (an intranet or an extranet).
The entire internet protocol suite -- a set of rules and procedures -- is commonly referred to as TCP/IP, though others are included in the suite.

TCP/IP specifies how data is exchanged over the internet by providing end-to-end communications that identify how it should be broken into packets, addressed, transmitted, routed and received at the destination. TCP/IP requires little central management, and it is designed to make networks reliable, with the ability to recover automatically from the failure of any device on the network.

The two main protocols in the internet protocol suite serve specific functions. TCP defines how applications can create channels of communication across a network. It also manages how a message is assembled into smaller packets before they are then transmitted over the internet and reassembled in the right order at the destination address.

IP defines how to address and route each packet to make sure it reaches the right destination. Each gateway computer on the network checks this IP address to determine where to forward the message.

B. The history of TCP/IP

The Defense Advanced Research Projects Agency (DARPA), the research branch of the U.S. Department of Defense, created the TCP/IP model in the 1970s for use in ARPANET, a wide area network that preceded the internet. TCP/IP was originally designed for the Unix operating system, and it has been built into all of the operating systems that came after it.

The TCP/IP model and its related protocols are now maintained by the Internet Engineering Task Force.

C. How TCP/IP works

TCP/IP uses the client/server model of communication in which a user or machine (a client) is provided a service (like sending a webpage) by another computer (a server) in the network.
Collectively, the TCP/IP suite of protocols is classified as stateless, which means each client request is considered new because it is unrelated to previous requests. Being stateless frees up network paths so they can be used continuously.

The transport layer itself, however, is stateful. It transmits a single message, and its connection remains in place until all the packets in a message have been received and reassembled at the destination.
The TCP/IP model differs slightly from the seven-layer Open Systems Interconnection (OSI) networking model designed after it, which defines how applications can communicate over a network.

D. TCP/IP model layers

TCP/IP functionality is divided into four layers, each of which include specific protocols.

- The application layer provides applications with standardized data exchange. Its protocols include the Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Post Office Protocol 3 (POP3), Simple Mail Transfer Protocol (SMTP) and Simple Network Management Protocol (SNMP).
- The transport layer is responsible for maintaining end-to-end communications across the network. TCP handles communications between hosts and provides flow control, multiplexing and reliability. The transport protocols include TCP and User Datagram Protocol (UDP), which is sometimes used instead of TCP for special purposes.
- The network layer, also called the internet layer, deals with packets and connects independent networks to transport the packets across network boundaries. The

network layer protocols are the IP and the Internet Control Message Protocol (ICMP), which is used for error reporting.
- The physical layer consists of protocols that operate only on a link -- the network component that interconnects nodes or hosts in the network. The protocols in this layer include Ethernet for local area networks (LANs) and the Address Resolution Protocol (ARP).

E. Advantages of TCP/IP

TCP/IP is nonproprietary and, as a result, is not controlled by any single company. Therefore, the internet protocol suite can be modified easily. It is compatible with all operating systems, so it can communicate with any other system. The internet protocol suite is also compatible with all types of computer hardware and networks.

VIII. NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things: a powerful N-dimensional array object, sophisticated (broadcasting) functions, tools for integrating C/C++ and Fortran code, useful linear algebra, Fourier transform, and random number capabilities.
Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.
NumPy is licensed under the BSD license, enabling reuse with few restrictions.

IX. OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

X. Face Detection using Haar Featured-based Cascade Classifiers

Haar Features (Fig 1.)

- Edge features (first row from Fig. 1)
- Line features (second row from Fig. 1)
- Center surround features (third row from Fig. 1)

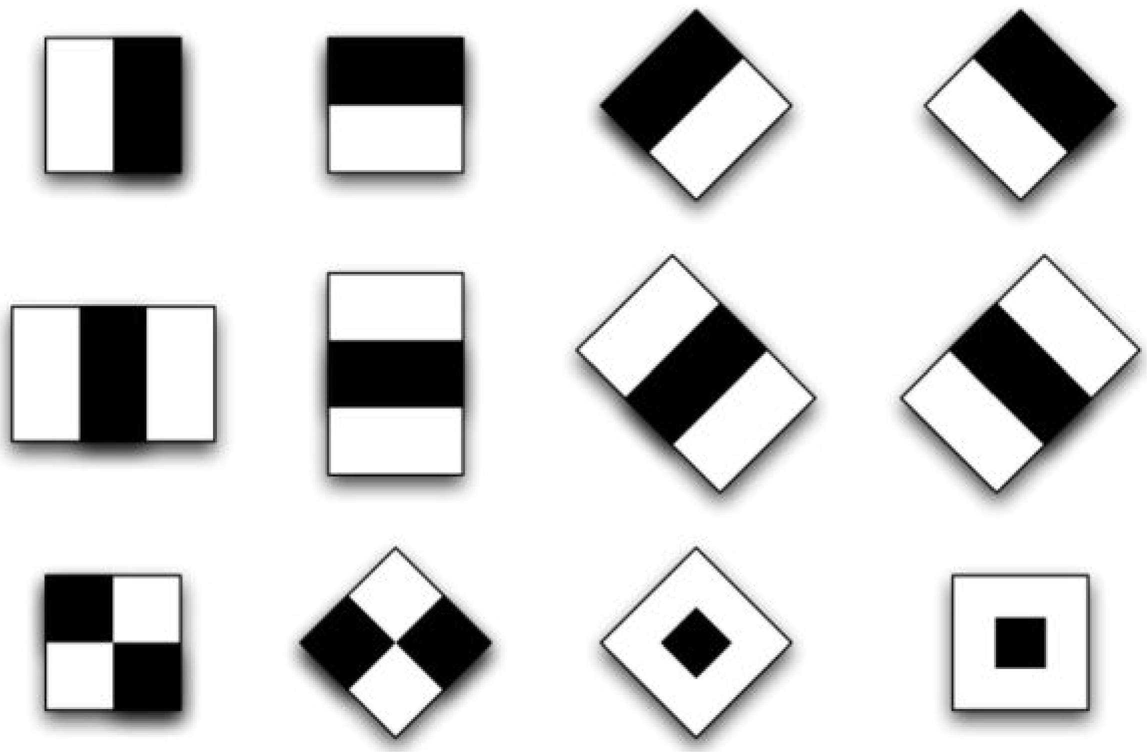

Fig. 1

Basics

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, haar features shown in Fig 1 are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.
Now all possible sizes and locations of each kernel is used to calculate plenty of features. (Just imagine how much computation it needs? Even a 24x24 window

results over 160000 features). For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, they introduced the integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by Adaboost.

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. Authors have a good solution for that.

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don't process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

For this they introduced the concept of Cascade of Classifiers. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. (Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is the plan !!!

Authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in first five stages. (Two features in the above image is actually obtained as the best two features from Adaboost). According to authors, on an average, 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works. Read paper for more details or check out the references in Additional Resources section.

XI. Pickle - Python object serialization

We use Pickle to send the data received from the Webcam to the server.
The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” or “flattening”, however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

XII. Conclusions

The advantage of using the client-server model in this case is that the processing of the image happens on the server, where you have the cascade files, so basically if you want to detect other objects you have to put the files and the necessary code only on the server and all the clients can use it, without any modification to the clients.

XIII. Resources

https://python.swaroopch.com/
https://techterms.com/definition/client-server_model
http://searchnetworking.techtarget.com/definition/TCP
http://searchnetworking.techtarget.com/definition/TCP-IP
https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html
https://www.lifewire.com/socket-programming-for-computer-networking-4056385
https://opencv.org
http://www.numpy.org/
https://www.youtube.com/watch?v=MDaZtJPv3Ik
https://www.youtube.com/watch?v=88HdqNDQsEk
https://docs.python.org/3/library/socket.html

https://docs.python.org/2/library/pickle.html
https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html
https://docs.python.org/3/howto/sockets.html
https://www.python.org/