

Project of implementing an intelligent system into a Raspberry Pi based on deep learning for face detection and recognition in real-time

Eduard Sulé Armengol

The School of Industrial, Aerospace and Audiovisual Engineering of Terrassa. ESEIAAT

Supervisor: Fatos Khafa

Department of Computer Science, ESEIAAT, UPC



**UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH**

June 10, 2019

Contents

1	Introduction, Scope and Objectives	9
1.1	Problem definition	9
1.2	State of art	10
1.3	Proposed solution	12
1.4	Relevance of the project	12
1.5	Goal of the project	13
1.6	Scope of the project	14
1.7	Functional requirements of the project	14
1.8	Non-functional requirements of the project	15
2	Development of the proposed solution	16
2.1	Machine learning basics	16
2.1.1	The Task, T	16
2.1.2	The Performance Measure, P	17
2.1.3	The experience, E	17
2.2	Work-flow of a deep learning project	18
2.3	The data set: images	18

2.4	Data gathering	20
2.4.1	Targeted person (positive case)	20
2.4.2	Random people (negative case)	20
2.4.3	EU General Data Protection Regulation [Cou16b]	21
2.4.4	Labelling the data	22
2.5	Data preparation	24
2.5.1	Open source computer vision (OpenCV2)	24
2.5.2	Viola-Jones algorithm	26
2.5.3	Normalisation	28
2.6	Data augmentation	30
2.7	Data split (train, valid and test sets)	32
2.8	Model selection	34
2.8.1	Why deep learning?	35
2.8.2	Multilayer perceptron neural network (MLP)	37
2.8.3	Convolutional Neural Networks	43
2.8.4	Model selected	47
2.9	Model training	49
2.9.1	TensorFlow	49
2.9.2	Training the model	49
2.10	Model evaluation	50
2.11	Model implementation into the Raspberry Pi	51
2.11.1	Raspberry Pi 3 model B	51
2.11.2	PiCamera module	52
2.11.3	Pi-Traffic lights	53

2.11.4	Input signal	55
2.11.5	Pipeline of the system	56
2.12	Experimental results and performance evaluation	57
2.12.1	Performance in terms of accuracy	58
2.12.2	Performance in terms of speed	60
3	SUMMARY OF RESULTS	62
3.1	Summary of the budget and study of economic viability	62
3.1.1	Costs of the device and its components	62
3.1.2	Cost of labor	62
3.1.3	Total cost	63
3.2	Analysis and assessment of environmental implications	63
3.3	Conclusions and recommendations for future work	63
	Bibliography	65
A	Appendix	68
A.1	Codes	68
A.1.1	VIOLA Jones data prep negative	68
A.1.2	VIOLA Jones data prep positive	69
A.1.3	Data augmentation	72
A.1.4	CNN	73
A.1.5	Raspberry	76
A.2	Experimental results	79

List of Figures

1.1	Probabilistic model of recognition for binary classification	10
1.2	Example of HOG. Source: " https://hypraptive.github.io/2017/02/02/find-the-bears-dlib.html "	11
1.3	Example of search selective. Source: J.R.R. Uijlings and al. (2012)	11
2.1	Deep learning workflow. Source: https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94	18
2.3	RGB function. Source: http://ai.stanford.edu/~syueung/cvweb/tutorial1.html 19	19
2.4	Example of RGB. Source: http://ai.stanford.edu/~syueung/cvweb/tutorial1.html 19	19
2.2	Representation of a greyscale image. Source: http://ai.stanford.edu/~syueung/cvweb/tutorial1.html	19
2.5	The model will assign the correct label depending on the source folder of each image.	23
2.6	Raw data	23
2.7	OpenCV2. Source: https://opencv.org/	24
2.8	Haar Feature that looks similar to the bridge of the nose is applied onto the face.- Source: https://en.wikipedia.org/wiki/ViolaJones	27
2.9	Haar Feature that looks similar to the eye region which is darker than the upper cheeks is applied onto a face. Source: https://en.wikipedia.org/wiki/ViolaJones	27

2.10	Data preparation pipeline for a positive case	29
2.11	Data preparation pipeline for a negative case	30
2.12	Distribution of the data	31
2.13	Data augmentation example	32
2.14	Data split	34
2.15	Illustration of a deep learning model. Source: "Deep learning" by Ian Goodfellow	36
2.16	Linear threshold unit. Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" by Aurélien Géron	38
2.17	Perceptron diagram. Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" by Aurélien Géron	38
2.18	Perceptron learning rule. Source: "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" by Frank Rosenblatt	39
2.19	Multi-Layer Perceptron. Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" by Aurélien Géron	40
2.20	ReLU vs sigmoid. ReLU is faster to compute and doesn't compromise the performance of the mode. Source: " https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 "	41
2.21	A modern MLP (including ReLU and softmax) for classification. Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" by Aurélien Géron	42
2.22	Flattening operation in a grey scale image (1 channel). Source: " https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 "	43
2.23	A RGB image. Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53	44
2.24	Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature (step1). Source: " https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 "	45

2.25	Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature (step2). Source: " https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 "	45
2.26	Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel. Source: " https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 "	46
2.27	Types of Pooling. Source: " https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 "	47
2.28	Final model	48
2.29	Parameters of the neural network	48
2.30	Train accuracy by epoch	50
2.31	Confusion matrix of the result of the test set	50
2.32	Raspberry Pi 3 Model B	52
2.33	PiCamera Module	53
2.34	Raspberry Pi with the traffic lights	54
2.35	Button circuit. Source: https://projects.raspberrypi.org/en/projects/physical-computing/2	55
2.36	The Raspberry Pi 3 model B and all the components	56
2.37	Pipeline of the process of detecting and recognising faces in real time in the Raspberry Pi	57
2.38	Experimental results from the Viola-Jones algorithm	58
2.39	Experimental results from the convolutional neural network	59
2.40	Histogram of time in seconds used for image caption and reading. Average = 4.97 seconds	60
2.41	Histogram of time in seconds used for face detection. Average = 5.02 seconds	60
2.42	Histogram of time in seconds used for face recognition. Average = 0.307 seconds	61

List of Tables

3.1	Costs of the device and its components	62
3.2	Total costs	63

Abstract

Artificial Intelligence (AI) is among most important fields of knowledge and applications in a large variety of domains. Recently however, it has become a trending research topic propelled by Cloud computing, social networks and alike. Terms like machine learning, "Big Data" and artificial neural networks very frequently appear not only in scientific media but even in the mass media.

In this project, we aim to design, implement and evaluate an AI technique, namely, deep learning, which has become very popular for face recognition. The problem is formulated from an engineering perspective: to design a small size system based on Raspberry Pi and an attached camera to it to detect and recognise human faces in real time. It should be mentioned that while for humans face recognition is a trivial task, we do it every day and with a full accuracy, for a computer, this is complex task. Recent applications from many industries show a large potential of intelligent systems that need to recognise faces with high accuracy.

The thesis is essentially structured into two main parts. In the first part we formulate the problem, analyse potential solutions and propose a solution for its resolution. In the second part of the project we develop the proposed solution into an implementation of an intelligent system for a computationally limited and physical portable device (Raspberry Pi). The solution is empirically evaluated in terms of accuracy and performance using real data sets. The relevance of using such a small size intelligent system relies in the fact that this application can be installed in other devices, such as drones, easily, at low cost and without compromising the performance and speed of the said intelligent system.

Declaration

I declare that, the work in this Degree Thesis is completely my own work, no part of this Degree Thesis is taken from other people's work without giving them credit, all references have been clearly cited. I understand that an infringement of this declaration leaves me subject to the foreseen disciplinary actions by The Universitat Politècnica de Catalunya BarcelonaTECH.

1 | Introduction, Scope and Objectives

1.1 Problem definition

In this thesis we study the problem of detecting and recognising faces in real-time for its implementation in computationally limited device of Raspberry Pi. The problem has three main components:

- We need an algorithm that given a certain image, it can detect human faces. It needs to draw a boundary box and crop them so we can input them into the recognition system.
- The recognition system, in this case, is a problem of binary classification where the system has to be able to, given certain pixel values, outputs a probability of that face belong to the person we want to recognise. In this case, for demonstration purposes, the targeted person will be Eduard Sulé Armengol the author of the project.

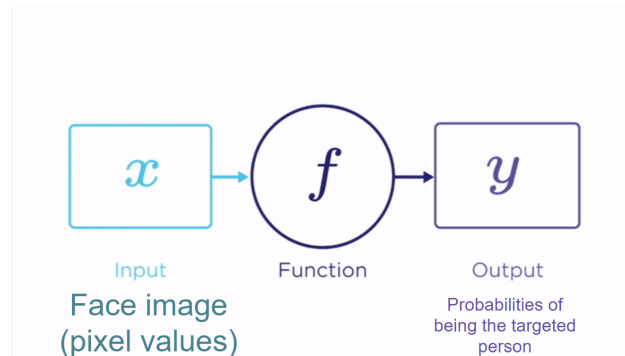


Figure 1.1: Probabilistic model of recognition for binary classification

- These two functions will be performed in real-time in a portable device, the Raspberry Pi. So this function should not be computationally expensive although, maximum values of accuracy and performance should be achieved.

1.2 State of art

The first part of the problem to be solved is the detection of faces in an image. A human can do this easily, but a computer needs precise instructions and constraints. The first algorithm that proved to be functional was the Viola-Jones algorithm proposed in 2001 by Paul Viola and Michael Jones. The Viola-Jones algorithm works by checking if parts of the image match certain hand-coded features that Viola and Jones found they conform to a human face.

The histogram of oriented gradients (HOG) was another algorithm that proved to be functional. In his core, HOG extracts the gradient of brightness of each pixel in the image for detecting the face presence. However, HOG is very sensitive to the brightness of the image, so it made this procedure only functional in controlled occasions were the image was taken without shadows or dark areas.

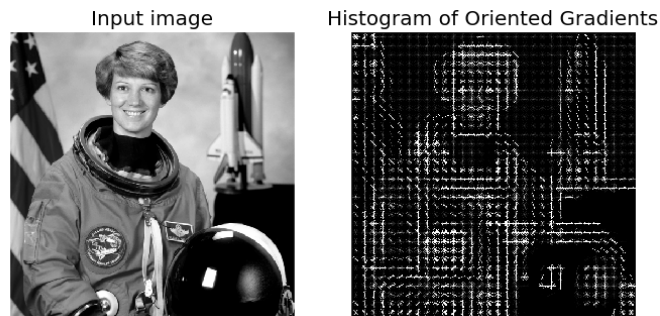


Figure 1.2: Example of HOG. Source: "<https://hypraptive.github.io/2017/02/02/find-the-bears-dlib.html>"

With the rise of artificial neural networks, which can extract the features that conform a human face automatically, there was no need for human hand-coded features and this kind of algorithm was proven to be more robust but computational more expensive. These artificial neural networks were called convolutional neural networks (CNN) since they convolute through the image searching for spatial correlations between pixels for detecting faces (more on CNN later in the project). So, the current state of art for object detection is R-CNN. R-CNN (R. Girshick et al., 2014) uses search selective (J.R.R. Uijlings and al. (2012)) to find out the regions of interests and passes them to a ConvNet. It tries to find out the areas that might be an object by combining similar pixels and textures into several rectangular boxes. The R-CNN paper uses 2,000 proposed areas (rectangular boxes) from search selective. Then, these 2,000 areas are passed to a pre-trained CNN model. Finally, the outputs (feature maps) are passed to a support vector machines (machine learning algorithm) for classification. The regression between predicted bounding boxes (bboxes) and ground-truth bboxes are computed.

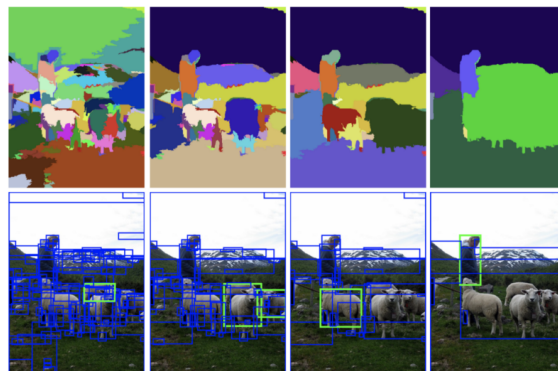


Figure 1.3: Example of search selective. Source: J.R.R. Uijlings and al. (2012)

The main problem is that the search selective algorithm is really computationally expensive since it has to find 2000 areas with similar textures (possible objects). Additionally, all of these areas need to be fed into a neural network that will output a prediction determining if said areas are the object, which the neural network was trained to recognise, or not. All of these operations can not be computed by small devices such as Raspberry Pi so this solution is not suitable for solving the need of the project.

1.3 Proposed solution

The proposed solution is to divide the problem into two parts. First, all the faces of the input image are detected, and then each face is fed to a neural network for posterior recognition (detecting a specific person). To do that we need a lighter face detection algorithm. We propose the Viola-Jones algorithm since it is specially designed for detecting faces, and the computational power required to execute is rather low compared to the R-CNN. Moreover, the Viola-Jones algorithm won't output 2000 areas to be classified, it will only focus on the possible faces on the image. Once faces are detected, we train a personalised neural network for detecting whatever person we want to recognise.

In this project, we are going to recognise the face of the author Eduard Sulé Armengol since gathering the data of a said person is easy and fast. However, it is worth pointing out, that if we wanted to train the neural network to recognise multiple persons, the procedures would be exactly the same, except the training data would need to correspond to data of each person we want to recognise and, in a single training session, the model would be able to recognise between the people we have trained on. Instead of binary classification (is the targeted person or not), we would have a multilabel classification problem.

1.4 Relevance of the project

With this project, we will design software able to do face detection and recognition in real-time that can run in a Raspberry Pi, without compromising accuracy. Raspberry Pi is easily installed in other machines such as cars or drones; so thanks to this lighter and efficient software we could enhance other machines with the said capabilities so they can interact with the real-world and make human-like decisions.

1.5 Goal of the project

The goal of the project is to design a software that will be executed by a physical portable device like a Raspberry Pi capable of detecting human faces and classifying them in real-time:

- **Face detection.** The way face detection in images is done nowadays is through an algorithm called Viola-Jones. This algorithm is based on the fact that some patterns are repeated in the pixel composition when faces appear in images. The algorithm convolutes through all the pixels of the image, finding those patterns. The main advantage of the algorithm is that it is swift compared to others while maintaining high accuracy.
- **Face recognition.** This is the part where Artificial Intelligence is needed. For humans, face recognition is a trivial task, we do it everyday and with an incommensurable accuracy. However, for a computer this task is presented as an extremely complex task. This is because it is not a task that it is possible to describe formally. More on that on the actual project documentation, but to give a brief explanation on what “describe formally” means, it is a way to describe a task that you cannot specify the exact sequence of task that the algorithm must do in order to find the solution because the high variance of inputs.

Deep learning attempts to solve this complexity by using the human approach. With deep learning a mathematical probabilistic model is built that simulates the behaviour of the human brain and it is feed with a lot of human images each one labelled, so the model is able to learn by example what a face of a young man is (for example).

- **Data collection and preparation.** In any AI project, gathering the data is a crucial aspect. With low data, the model would not learn enough, and with bad data, the model will learn poorly. So we need to train the model with enough and varied data so it can perform good predictions. Also, much preparation of this data has to be done since any model can learn by raw data.
- **An efficient implementation of the software** has to be done since any loss of efficiency can result in not meeting the real-time detection and recognition requirement.
- **Once everything implemented,** we need to perform a lot of empirical tests to see if we are meeting the desired accuracy. Any machine/deep learning algorithm is a probabilistic model, so we can't expect a hundred percent accuracy.

1.6 Scope of the project

The scope of the project is composed by the following points:

- Gather data of human faces, considering the EU GDPR restrictions.
- Implement a face detection algorithm (Viola-Jones algorithm).
- Train and test different neural networks architectures.
- Deploy the final refined and trained neural network in the Raspberry Pi. In addition, implement the face detection algorithm and all the image pre-processing in the device.
- Evaluate the system integrated on the Raspberry Pi (speed and accuracy).

1.7 Functional requirements of the project

The functional requirements of the project comprise, among others, the following:

- Image reading in real-time. For, this a camera properly attached to the Raspberry Pi is needed.
- Detecting the faces in the image. We need an algorithm that, after the image is taken, obtains the faces presents in the image, removing all other areas of certain image.
- Recognising the face.
- High accuracy. AI systems can incur in error since the task that they are given can be very broad and they are not ruled by formal instructions. However, designing the optimal neural network and the correct image pre-processing is key to enhance the performance of the algorithm. There are also other important metrics that are worth to check out, like recall and sensitivity.
- Fast response time (low latency). Since real time recognition is a paramount feature in the project, it is mandatory to optimise all stages of the procedure to ensure the desired speed.
- Small system size. It is not needed the best camera quality or the most powerful Raspberry Pi. The minimal size of device that accomplish the requirement would be enough.

1.8 Non-functional requirements of the project

Economical requirements:

- Low budget project. A Raspberry Pi with basic functions and with camera on it is needed.

Compliance with legal requirements:

- EU GDPR. Any project that involves AI of some kind, needs data. Data nowadays is easy to get but it is mandatory to consider the new legislation that regularises data gathering and use. Otherwise it might be very problematic, particularly in terms of privacy. Therefore, in this project, there will be a huge emphasis on legal requirement and the lighthouse will be the EU GDPR.

Open source software requirements:

- Using open software and libraries is considered relevant for this project. For that we will use open source libraries such as OpenCV2.

2 | Development of the proposed solution

2.1 Machine learning basics

Deep learning is a specific kind of machine learning. To understand deep learning well, one must have a solid understanding of the basic principles of machine learning. This section provides a brief introduction to the most important general principles that are applied throughout the rest of the book. A machine learning algorithm is an algorithm that can learn from data. However, what do we mean by learning? Mitchell (1997) provides the definition “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”

2.1.1 The Task, T

Machine learning allows us to tackle tasks that are too difficult to solve with fixed programs written and designed by human beings. From a scientific and philosophical point of view, machine learning is engaging because developing our understanding of machine learning entails developing our understanding of the principles that underlie intelligence. In this relatively formal definition of the word “task,” the process of learning itself is not the task. Learning is our means of attaining the ability to perform the task. For example, if we want a robot to be able to walk, then walking is the task. We could program the robot to learn to walk, or we could attempt to write a program that specifies how to walk manually directly.

Machine learning tasks are usually described in terms of how the machine learning system should process an example. An example is a collection of features that have been quanti-

tatively measured from some object or event that we want the machine learning system to process. For example, the features of an image are usually the values of the pixels in the image.

In this case, the task consists of a classification problem where the model has to be able to classify faces through the pixel values of them.

2.1.2 The Performance Measure, P

To evaluate the abilities of a machine learning algorithm, we must design a quantitative measure of its performance. Usually, this performance measure P is specific to the task T being carried out by the system. For tasks such as classification, we often measure the accuracy of the model. Accuracy is just the proportion of examples for which the model produces the correct output. We can also obtain equivalent information by measuring the error rate, the proportion of examples for which the model produces incorrect output. Usually, we are interested in how well the machine learning algorithm performs on data that it has not seen before since this determines how well it works when deployed in the real world. We, therefore, evaluate these performance measures using a test set of data that is separate from the data used for training the machine learning system.

2.1.3 The experience, E

Machine learning algorithms can be broadly categorised as unsupervised or supervised by what kind of experience they are allowed to have during the learning process.

- Unsupervised learning algorithms experience a dataset containing many features, then learn useful properties of the structure of this dataset. In the context of deep learning, we usually want to learn the entire probability distribution that generated a dataset, whether explicitly as in density estimation or implicitly for tasks like synthesis or denoising. Some other unsupervised learning algorithms perform other roles, like clustering, which consists of dividing the dataset into clusters of similar examples.
- Supervised learning algorithms experience a dataset containing features, but each example is also associated with a label or target. The term supervised learning originates from the view of the target y being provided by an instructor or teacher who shows the machine learning system what to do. In unsupervised learning, there is no instructor or teacher, and the algorithm must learn to make sense of the data without this guide.

2.2 Work-flow of a deep learning project

In this section, an overview of the work-flow of a deep learning project is presented. It is possible to divide this project into 5 stages:

1. Gathering data.
2. Data pre-processing.
3. Researching the model that suits best for the type of data
4. Training and testing the model
5. Evaluation

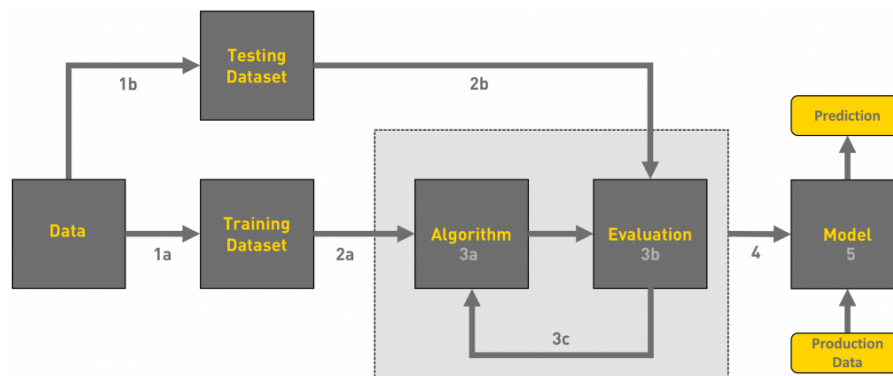


Figure 2.1: Deep learning workflow. Source: <https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94>

Given the general overview of the work-flow, in the next sections, all details and explanations are given by each stage.

2.3 The data set: images

Data can be represented in multiple ways: structured data (tables), text, sound, images, etc. In this project, images are the data structure that is used, so it is reasonable to explain the format of it to understand how the neural network is going to work.

In essence, images are matrices of numbers, each number referring to each pixel value. Each of the pixels that represent an image stored inside a computer has a pixel value which

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

Figure 2.3: RGB function. Source: <http://ai.stanford.edu/~syyeong/cvweb/tutorial1.html>

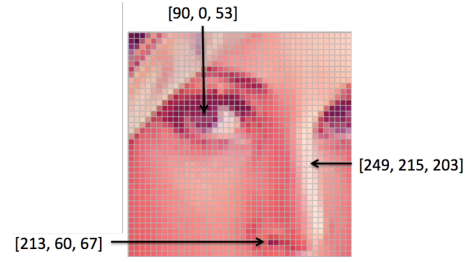


Figure 2.4: Example of RGB. Source: <http://ai.stanford.edu/~syyeong/cvweb/tutorial1.html>

describes how bright that pixel is, and what colour it should be. In the simplest case of binary images, the pixel value is a 1-bit number indicating either foreground or background. For grayscale images, the pixel value is a single number that represents the brightness of the pixel. The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to be black, and 255 is taken to be white. Values in between make up the different shades of gray.

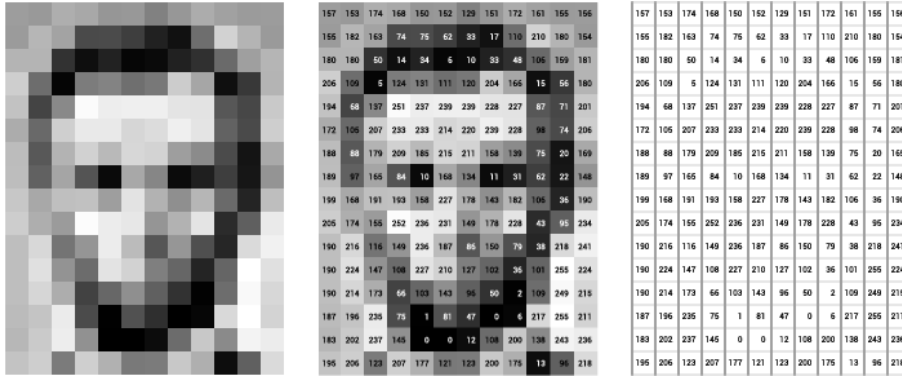


Figure 2.2: Representation of a grayscale image. Source: <http://ai.stanford.edu/~syyeong/cvweb/tutorial1.html>

To represent colour images, separate red, green and blue components must be specified for each pixel (assuming an RGB colorspace), and so the pixel ‘value’ is a vector of three numbers. Often the three different components are stored as three separate ‘grayscale’ images known as colour planes (one for each of red, green and blue), which have to be recombined when displaying or processing.

To sum up, an RGB image is defined by:

- Width: number of pixels in the x-axis
- Height: number of pixels in the y-axis
- Channels: 3 in case RGB, 1 in case grayscale.
- Pixel values: 0-255 numerical value according to the potency of color.

It is essential to understand the structure of the data since the model selected has to make decisions only based on those parameters.

2.4 Data gathering

Data gathering is the process of recollecting data to feed them into an intelligent algorithm so it can perform its task correctly. In this case, both images are needed of the targeted person (positive case) and random people (negative case).

2.4.1 Targeted person (positive case)

The targeted person is the author of this project, Eduard Sulé Armengol, since gathering images of himself is an easy and safe task. It is worth pointing out that due to the nature of this project and the capabilities of deep learning, it is straightforward to generalise and change the targeted person without changing any basic procedure. These images had been taken for a long time using several phones or cameras. Also, they have been taken in a way such, although they belong to the same person, they present different characteristics (different haircuts, shaves, backgrounds.etc) so the model will be able to generalise better. A total of 337 images had been gathered.

2.4.2 Random people (negative case)

Images of random people are needed since it is mandatory to teach the model what the face of the non-targeted person looks like; they act as the negative case. To gather all that data, a public database of images has been used, Faces in the Wild.

Faces in the Wild is a database of face photographs designed for studying the problem of unconstrained face recognition. The data set contains more than 13,000 images of faces

collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more clear photos in the data set. For the specific need, a total of 6500 images are more than necessary.

In the next section, the current regulations (EUGDPR [Cou16b]) are reviewed to verify if the use of this public database is legal.

2.4.3 EU General Data Protection Regulation [Cou16b]

At a high level, GDPR[Cou16b] requires that the data controller (or data processor, either a business, non-profit or other organisation) handles the data of users and customer (collectively called data subjects) in a responsible, secure, transparent and non-abusive fashion, thus allowing the user to be in the control of her/his data or data pertaining to her/his identity. This means that organisations can only handle your personally identifiable data at your request, with your consent and must stop using any such data at your request. In this case, the identity of the people in this dataset is under protection since the purpose of the project is not to distinguish amongst them instead of the targeted person who, being the author of the project, gives all the consent to use his data. Another important points worth of reviewing are the followings:

- Processing of personally identifiable data is lawful if the user gives his explicit consent, if it's in the public interest (not to be confused with a private interest of a group, organisation) or if it's required by applicable law. (Article 6)
- Processing of personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership, and the processing of genetic data, biometric data for the purpose of uniquely identifying a natural person, data concerning health or data concerning a natural person's sex life or sexual orientation shall be prohibited, unless the user gives explicit consent. Such explicit consent must be provable under non-repudiation. (Article 9)
- The data processor is not under any requirement to maintain, process, or acquire additional information about the user to comply with the GDPR.
- The user has the right to inquire about the information stored or processed by an organisation (Articles 15: Right of access by the data subject)
- The user has the right to correct or amend any incorrect data stored or processed by an organisation (Articles 16: Right to rectification)

- The user has the right to request the complete erasure of her/his data as stored or processed by an organisation, including but not limited to withdrawal of consent (Articles 17: Right to erasure, also known as "Right to be forgotten")
- The user has the right to request that stored data is not processed at any point, except for those purposes mandated by law (Articles 18: Right to restriction of processing)
- Organisations must notify data users of compliance with their requests of inquiry, deletion, or restriction. (Articles 19)
- The user has the right to request all information held by an organisation about her/him, "in a structured, commonly used and machine-readable format and have the right to transmit that data to another controller without hindrance from the controller (organisation processing the data)" (Article 20)
- The user has the right to object to the processing of her/his data (Article 21)
- The organisation processing personal information shall do so using state of the art methods for processing only data necessary for each specific purpose (Article 25: Data protection by design and by default)
- The organisation processing personal information shall do so using state of the art methods to "ensure the ongoing confidentiality, integrity, availability, and resilience of processing systems and services" (Article 32)

In this project, any of these points are violated, so it is plausible to conclude that it is safe to use this database. Finally, it is worth mentioning that this project is aimed for educational purposes only.

2.4.4 Labelling the data

Once collected all the data, the only thing left is to label the data. Labeling means classifying each image on the class that they belong so that the model can differentiate them in the training phase. Labeling the data is crucial since this is a supervised learning project. Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and the desired output value (also called the supervisory signal). A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario allows for the algorithm to determine the class labels for

unseen instances correctly. This requires the learning algorithm to generalise from the training data to unseen situations in a "reasonable" way.

So, in this case, the input object corresponds to the image vector (pixel values), and the output value or label is 0 if the negative case and 1 if the positive case.

The way labeling is done in this case is to separate into two folders all the images depending on if they belong in the positive or negative case. Then, when the model is trained and depending on the source folder, the model will assign the correct label (0 for the negative case and 1 for the positive case).

Este equipo > Escritorio > TFG > face_data



Nombre	Fecha de modifica...	Tipo	Tamaño
 negative	16/05/2019 15:30	Carpeta de archivos	
 positive	16/05/2019 22:37	Carpeta de archivos	

Figure 2.5: The model will assign the correct label depending on the source folder of each image.

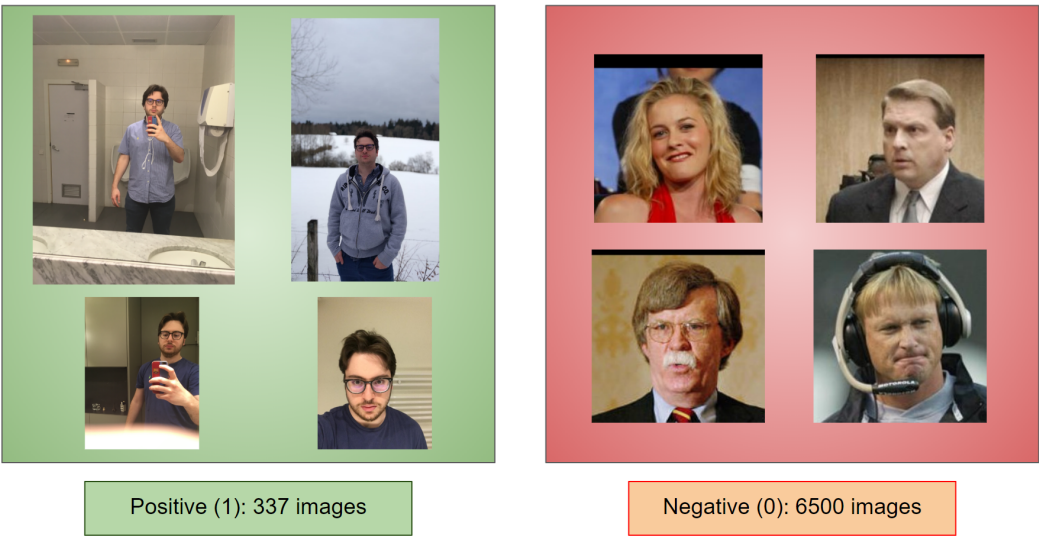


Figure 2.6: Raw data

2.5 Data preparation

Data preparation is key for ensuring a good performance of the model. Although deep learning is a technique that presents great accuracy, helping the model in tasks that are possible of automating is preferable. In this case, it is even more important to do so since images can be very complex for any deep learning model to handle. Two big procedures are necessary to apply to images before training the model:

- Detecting faces. This is paramount, so the model has to only focus on the relevant pixels for recognising the person. Selecting only the face results in getting rid of the background and other noises that result in an improvement of the performance of the model. That task is performed by the Viola-Jones algorithm, which is an algorithm designed for detecting faces in images.
- Normalisation. This includes resising the image in a standard form and re-scaling the pixel values to values between 0 and 1.

All this image pre-processing is done in Python with the open-source library OpenCV2.

2.5.1 Open source computer vision (OpenCV2)

OpenCV2 is a library of programming functions mainly aimed at real-time computer vision. The main reason we use OpenCV2 is that Opencv runs faster and the space occupied by the software is much lesser than other computer vision-oriented libraries. Also, Opencv2+python can be installed on a Raspberry Pi.

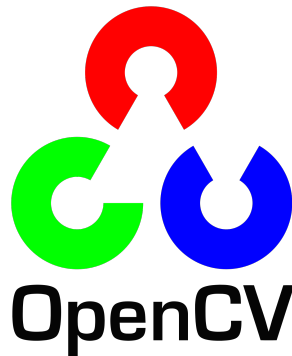


Figure 2.7: OpenCV2. Source:<https://opencv.org/>

Some of the function used in this project are the followings:

- `imread`: Loads an image from a file.
`cv2.imread(filename) → retval`
 - `filename`: Name of file to be loaded.
 - `retval`: Matrix of pixel values of the loaded image.
- `imwrite`: Saves an image to a specified file.
`cv2.imwrite(filename, img)`
 - `filename`: Name of the file.
 - `img`: Image to be saved
- `getRotationMatrix2D`: Calculates an affine matrix of 2D rotation.
`cv2.getRotationMatrix2D(center, angle, scale) → retval`
 - `center`: Center of the rotation in the source image.
 - `angle`: Rotation angle in degrees. Positive values mean counter-clockwise rotation (the coordinate origin is assumed to be the top-left corner).
 - `scale`: Isotropic scale factor.
 - `retval`: Matrix of pixel values of the rotated image.
- `resize`: Resizes an image.
`cv2.resize(src, dsize[, dst]) → retval`
 - `src`: input image.
 - `retval`: output image with desired size
 - `dsize`: output image size
- `cvtColor`: Converts an image from one color space to another.
`cv2.cvtColor(src, code[, dst[, dstCn]]) → dst`
 - `src`: input image
 - `dst`: output image
 - `code`: color space conversion code
 - `dstCn`: number of channels in the destination image
- `CascadeClassifier.detectMultiScale`: Viola Jones algorithm for detecting faces. In next section, the algorithm and its application is explained in more detail.
`cv2.CascadeClassifier.detectMultiScale(image[, scaleFactor[, minNeighbors[, minSize[, maxSize]]]]) → objects`

- image: Matrix containing an image where objects are detected.
- objects: Vector of rectangles where each rectangle contains the detected object.
- scaleFactor: Parameter specifying how much the image size is reduced at each image scale.
- minNeighbors: Parameter specifying how many neighbors each candidate rectangle should have to retain it.
- minSize: Minimum possible object size. Objects smaller than that are ignored.
- maxSize: Maximum possible object size. Objects larger than that are ignored.
- cascade: Haar classifier cascade. It can be loaded from XML or YAML file using `Load()`.

2.5.2 Viola-Jones algorithm

The Viola-Jones object detection framework is the first object detection framework to provide competitive object detection rates in real-time proposed in 2001 by Paul Viola and Michael Jones.

The problem to be solved is the detection of faces in an image. A human can do this easily, but a computer needs precise instructions and constraints. To make the task more manageable, Viola-Jones requires full view frontal upright faces. Thus to be detected, the entire face must point towards the camera and should not be tilted to either side. While it seems these constraints could diminish the algorithm's utility somewhat, because the detection step is most often followed by a recognition step, in practice these limits on pose are quite acceptable.

The algorithm maps through all the image searching for features that researchers had found that resembles a human face. That features are given the name of Haar features. The features sought by the detection framework universally involve the sums of image pixels within rectangular areas. As such, they bear some resemblance to Haar basis functions, which have been used previously in the realm of image-based object detection. However, since the features used by Viola and Jones all rely on more than one rectangular area, they are generally more complex. The value of any given feature is the sum of the pixels within clear rectangles subtracted from the sum of the pixels within shaded rectangles. Rectangular features of this sort are primitive when compared to alternatives such as steerable filters. Although they are sensitive to vertical and horizontal features, their feedback is considerably coarser.

Haar features

All human faces share some similar properties. These regularities may be matched using Haar Features.

A few properties common to human faces:

- The eye region is darker than the upper-cheeks.
- The nose bridge region is brighter than the eyes.



Figure 2.8: Haar Feature that looks similar to the bridge of the nose is applied onto the face.-
Source: <https://en.wikipedia.org/wiki/ViolaJones>



Figure 2.9: Haar Feature that looks similar to the eye region which is darker than the upper cheeks is applied onto a face.
Source: <https://en.wikipedia.org/wiki/ViolaJones>

Composition of properties forming matching facial features:

- Location and size: eyes, mouth, bridge of the nose

- Value: oriented gradients of pixel intensities

Rectangle features:

- $Value = \sum(\text{pixels in black area}) - \sum(\text{pixels in white area})$
- Three types: two-, three-, four-rectangles, Viola & Jones used two-rectangle features
- For example, the difference in brightness between the white & black rectangles over a specific area
- Each feature is related to a special location in the sub-window

The characteristics of the Viola-Jones algorithm which make it a good detection algorithm are:

- Robust – very high detection rate (true-positive rate) and little false-positive rate always.
- Real-time – For practical applications, at least 2 frames per second must be processed.
- Face detection only (not recognition) - The goal is to distinguish faces from non-faces (detection is the first step in the recognition process).

2.5.3 Normalisation

Once detected the human face in the image, the next step remaining is to crop this region to remove the noise of the background and resize this cropped image to a standard shape so the model can ingest it. All machine/deep learning model requires the same input shape for all elements, no matter if they are used for training or testing. In this case, it has opted for a 50x50x3 input shape.

This decision, although arbitrary, is adjusted through the model training in case of low accuracy or high complexity. The lower the input shape (width and height), the less information the model has at his disposal, but the faster is trained since the number of parameters of the model are lower. Also, it is advisable to re-scale the pixel values to a range of 0 and 1. The majority of machine/deep learning models prefer values in that range because they tend to spike with high values resulting in lower accuracy.

All these steps of data preparation are applied to all images. All the codes referring to these procedures can be found in the annex. More precisely, they can be found in the sections "VIOLA_JONES_data_prep_positive.py" and "VIOLA_JONES_data_prep_negative.py."



Figure 2.10: Data preparation pipeline for a positive case

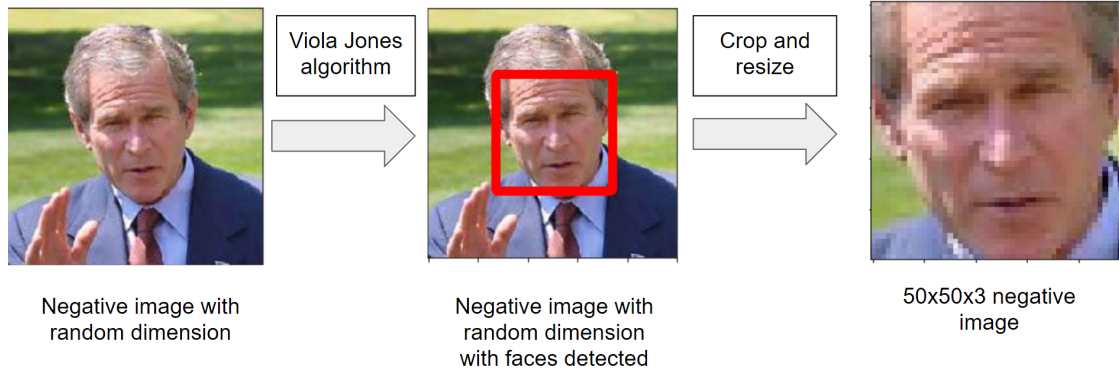


Figure 2.11: Data preparation pipeline for a negative case

2.6 Data augmentation

Before starting training the model, there is still the necessity to deal with another problem which some may already have noticed: the training set is conformed by 337 positive images and 6500 negative images. This is a massive imbalanced dataset.

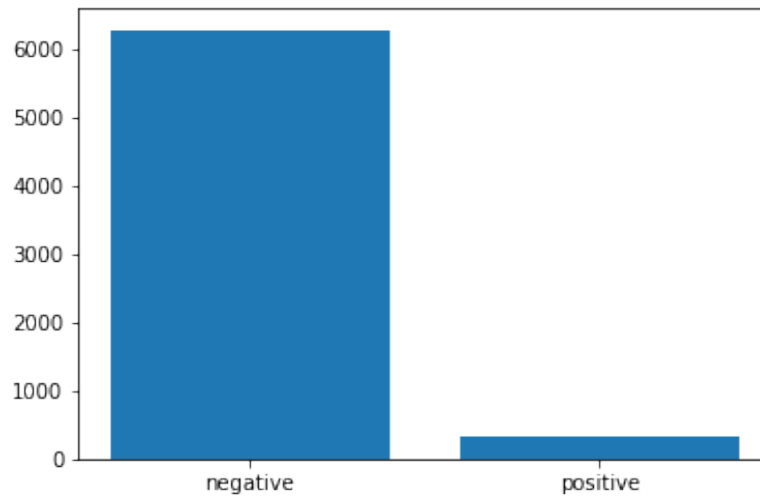


Figure 2.12: Distribution of the data

An unbalanced dataset creates a problem due to three main reasons:

- Not getting optimised results for the class which is unbalanced in real-time as the model/algorithm never gets sufficient look at the underlying class
- It creates a problem of making a validation or test sample as its difficult to have representation across classes in case number of observation for few classes is extremely less
- The model could be biased to the negative class since predicting "negative" has more chance to be correct by simple probabilities.

There are several ways to solve this issue:

- Undersampling-Randomly deletes the class which has sufficient observations so that the comparative ratio of two classes is significant in our data. Although this approach is straightforward to follow, there is a high possibility that the data that we are deleting may contain important information about the predictive class.
- Collecting more data. That's a reasonable way to proceed, but it requires much time, and it is difficult to reach the same samples as the negative case by hand.
- Data augmentation. Since images are the data used in this project, it is possible to apply transformations to the positive case, so similar images are generated. Common

transformations used in this procedure are rotations, zooms, stretches, horizontal flips, etc. All of these transformations are in such a range they don't distort the image to be completely illegible for the model. For more information on how these transformations are implemented, check the annex section "Data_augmentation.py"

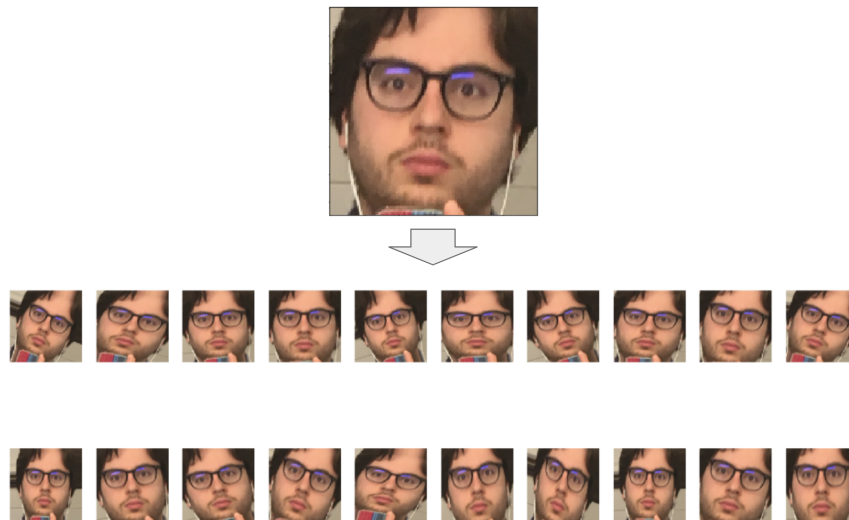


Figure 2.13: Data augmentation example

This procedure is applied in the training set of the data. (More on that in next section Data split (train, valid and test sets)). With this technique, from a single image, it is obtained 20 new images that help the model by a lot.

2.7 Data split (train, valid and test sets)

The data used to build the final model usually comes from multiple datasets. In particular, three data sets are commonly used in different stages of the creation of the model.

The model is initially fit on a training dataset, which is a set of examples used to fit the parameters (e.g., weights of connections between neurons in artificial neural networks) of the model. The model (e.g., a neural net or a naive Bayes classifier) is trained on the training dataset using a supervised learning method. In practice, the training dataset often consists of pairs of an input vector (or scalar) and the corresponding output vector (or scalar), which is the label as previously explained. The current model is run with

the training dataset and produces a result, which is then compared with the target, for each input vector in the training dataset. Based on the result of the comparison, and the specific learning algorithm being used, the parameters of the model are adjusted. The model fitting can include both variable selection and parameter estimation.

Successively, the fitted model is used to predict the responses for the observations in a second dataset called the validation dataset. The validation dataset provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters (e.g., the number of hidden units in a neural network).

Finally, the test dataset is a dataset used to provide an unbiased evaluation of a final model fit on the training dataset. If the data in the test dataset has never been used in training (for example in cross-validation), the test dataset is also called a holdout dataset.

To recap:

- Training Dataset: The sample of data used to fit the model. In this case, it is composed of 11140 images (6400 negatives and 4740 positives, which are the images resulting from the data augmentation technique).
- Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as a skill on the validation dataset is incorporated into the model configuration. In this case, it is composed of 100 images (50 negatives and 50 positives).
- Test Dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. In this case, it is composed of 100 images (50 negatives and 50 positives).

The split is performed by the codes "VIOLA_JONES_data_prep_positive.py" and "VIOLA_JONES_data_prep_negative.py", so for more details, check the annex. The next figure shows more clearly the data split made.

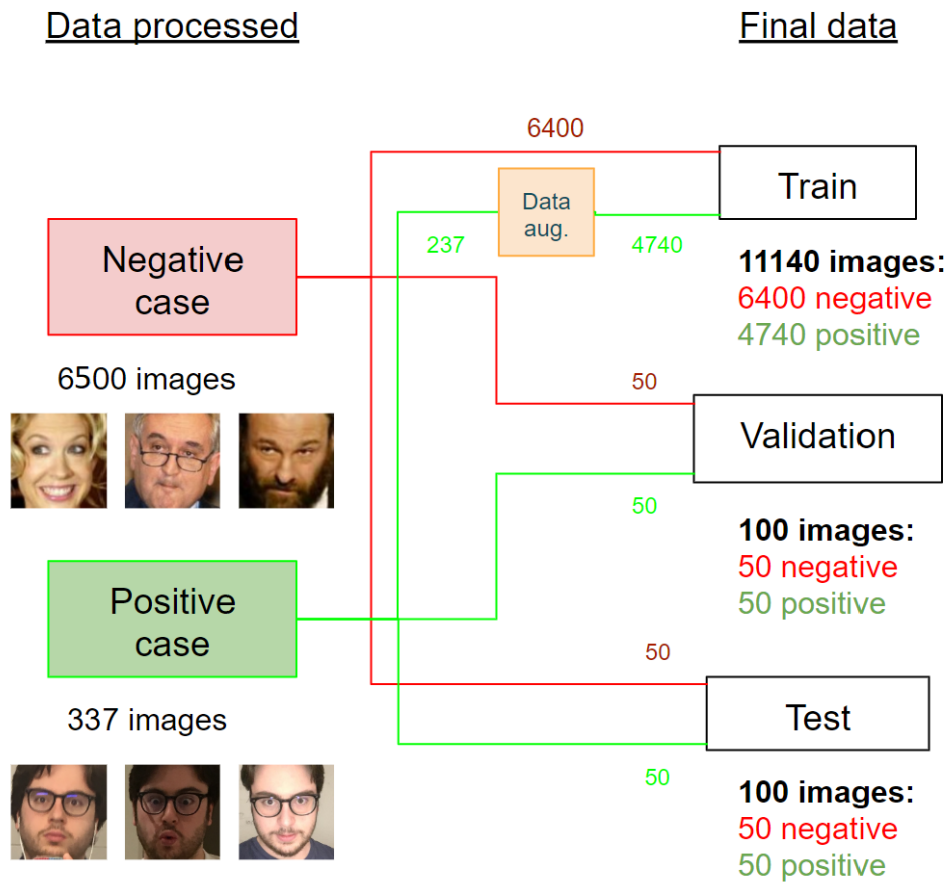


Figure 2.14: Data split

2.8 Model selection

The plan is to use deep learning, which is a type of machine learning procedure. In the next sections, it is going to be explained why deep learning is used in this kind of project and how exactly deep learning works for this case.

2.8.1 Why deep learning?

Inventors have long dreamed of creating machines that think. When programmable computers were first conceived, people wondered whether such machines might become intelligent, over a hundred years before one was built (Lovelace, 1842). Today, artificial intelligence (AI) is a thriving field with many practical applications and active research topics.

Intelligent software is used to automate routine tasks, understand speech or images, make diagnoses in medicine and support basic scientific research. In the early days of artificial intelligence, the field rapidly tackled and solved problems that are intellectually difficult for human beings but relatively straightforward for computers—problems that can be described by a list of formal, mathematical rules. The true challenge to artificial intelligence proved to be solving the tasks that are easy for people to perform but hard for people to describe formally—problems that we solve intuitively, that feel automatic, like recognising spoken words or faces in images. The solution is based in allowing computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. By gathering knowledge from experience (data), this approach avoids the need for human operators to formally specify all of the knowledge that the computer needs. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones.

The performance of these simple machine learning algorithms depends heavily on the representation of the data they are given. This dependence on representations is a general phenomenon that appears throughout computer science and even daily life. In computer science, operations such as searching a collection of data can proceed exponentially faster if the collection is structured and indexed intelligently. It is not surprising that the choice of representation has an enormous effect on the performance of machine learning algorithms. Many artificial intelligence tasks can be solved by designing the right set of features to extract for that task, then providing these features to a simple machine learning algorithm. For example, a useful feature for speaker identification from sound is an estimate of the size of speaker’s vocal tract. It therefore gives a strong clue as to whether the speaker is a man, woman, or child. However, for many tasks, it is difficult to know what features should be extracted. For example in this project, we have the urge to recognise from pixel values if the faces of the image that all this pixels values represent corresponds to the target person. We could try to analyse what conforms in terms of the pixel values the face of the targeted person but finding these patterns is very hard for the programmer since the data that is not presented in an ordered way and it can have multiple representations of itself. For example, brightness of the photo can change completely all the patterns we might be able to find studying the positive cases, or maybe the targeted person changes regularly his hairstyle so all the patterns found for some cases are going to be useless in others.

Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning allows the computer to build complex concepts out of simpler concepts. Next figure shows how a deep learning system can represent the concept of an image of a person by combining simpler concepts, such as corners and contours, which are in turn defined in terms of edges.

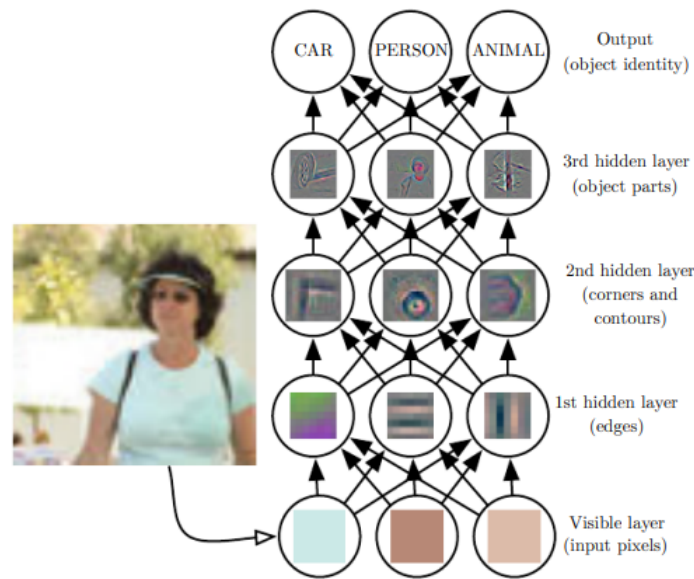


Figure 2.15: Illustration of a deep learning model. Source: "Deep learning" by Ian Goodfellow

It is difficult for a computer to understand the meaning of raw sensory input data, such as this image represented as a collection of pixel values. The function mapping from a set of pixels to an object identity is very complicated. Learning or evaluating this mapping seems insurmountable if tackled directly. Deep learning resolves this difficulty by breaking the desired complicated mapping into a series of nested simple mappings, each described by a different layer of the model. The input is presented at the visible layer, so named because it contains the variables that we are able to observe. Then a series of hidden layers extracts increasingly abstract features from the image. These layers are called “hidden” because their values are not given in the data; instead the model must determine which concepts are useful for explaining the relationships in the observed data. The images here are visualisations of the kind of feature represented by each hidden unit. Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighboring pixels. Given the first hidden layer’s description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognisable as collections

of edges. Given the second hidden layer's description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners. Finally, this description of the image in terms of the object parts it contains can be used to recognise the objects present in the image.

The quintessential example of a deep learning model is the feedforward deep network or multilayer perceptron (MLP). A multilayer perceptron is just a mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions called perceptrons which resemble the biological neurons in the human brain. We can think of each application of a different mathematical function as providing a new representation of the input. In next section, we are going to go in detail of what are those mathematical operations called perceptrons and how are they connected.

To summarise, deep learning, is an approach to AI. Specifically, it is a type of machine learning, a technique that allows computer systems to improve with experience and data. Machine learning is the only viable approach to building AI systems that can operate in complicated, real-world environments. Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

2.8.2 Multilayer perceptron neural network (MLP)

A multilayer perceptron is just a mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions called perceptrons.

The perceptron

The perceptron is the basic unit of an artificial neural network. It is based on a slightly different artificial neuron called a linear threshold unit (LTU): the inputs and output are now numbers (instead of binary on/off values) and each input connection is associated with a weight. The LTU computes a weighted sum of its inputs ($z = w_1x_1 + w_2x_2 + \dots + w_nx_n$), then applies a step function to that sum and outputs the result: $hw(\mathbf{x}) = step(z) = step(\mathbf{w}^T \mathbf{x})$

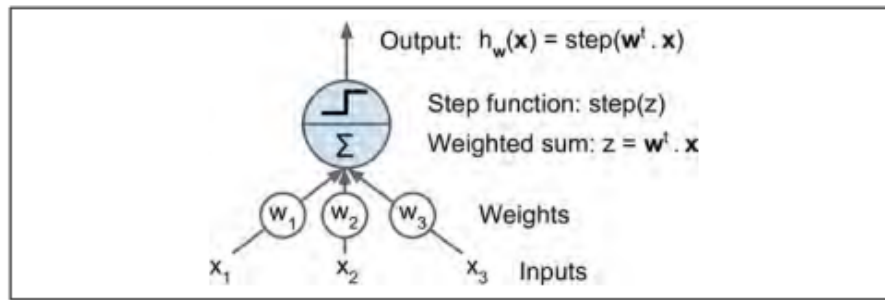


Figure 2.16: Linear threshold unit. Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" by Aurélien Géron

A single LTU can be used for simple linear binary classification. It computes a linear combination of the inputs and if the result exceeds a threshold, it outputs the positive class or else outputs the negative class. Training an LTU means finding the right values for w_0 , w_1 and w_2 (the training algorithm is discussed shortly). A Perceptron is simply composed of a single layer of LTUs, with each neuron connected to all the inputs. These connections are often represented using special passthrough neurons called input neurons: they just output whatever input they are fed.

Moreover, an extra bias feature is generally added ($w_0 = 1$). This bias feature is typically represented using a special type of neuron called a bias neuron, which just outputs 1 all the time. A Perceptron with two inputs and three outputs is represented in next figure. This Perceptron can classify instances simultaneously into three different binary classes, which makes it a multioutput classifier.

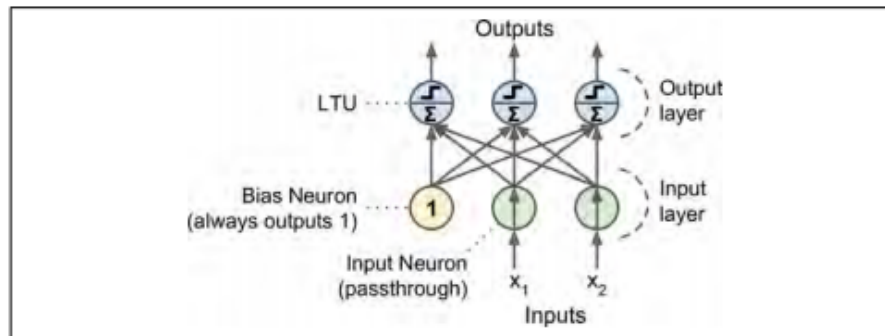


Figure 2.17: Perceptron diagram. Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" by Aurélien Géron

The Perceptron training algorithm proposed by Frank Rosenblatt was largely inspired by

Hebb's rule. In his book [HEBB], "The Organization of Behavior", published in 1949, Donald Hebb suggested that when a biological neuron often triggers another neuron, the connection between these two neurons grows stronger. This rule later became known as Hebb's rule (or Hebbian learning); that is, the connection weight between two neurons is increased whenever they have the same output. Perceptrons are trained using a variant of this rule that takes into account the error made by the network; it does not reinforce connections that lead to the wrong output. More specifically, the Perceptron is fed one training instance at a time, and for each instance it makes its predictions. For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction. Next figure contains the formula that Frank Rosenblatt proposed for training a perceptron.

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(\hat{y}_j - y_j)x_i$$

Figure 2.18: Perceptron learning rule. Source: "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" by Frank Rosenblatt

Where, in the formula:

- $w_{i,j}$ is the connection weight between the i_{th} input neuron and the j_{th} output neuron.
- x_i is the i_{th} input value of the current training instance.
- \hat{y}_j is the output of the j_{th} output neuron for the current training instance.
- y_j is the target output of the j_{th} output neuron for the current training instance.
- η is the learning rate.

In their 1969 monograph titled Perceptrons, Marvin Minsky and Seymour Papert highlighted a number of serious weaknesses of Perceptrons, in particular the fact that they are incapable of solving some trivial problems. However, it turns out that some of the limitations of Perceptrons can be eliminated by stacking multiple Perceptrons. The resulting ANN is called a Multi-Layer Perceptron (MLP).

Multi-Layer Perceptron (MLP)

An MLP is composed of one (passthrough) input layer, one or more layers of LTUs, called hidden layers, and one final layer of LTUs called the output layer. Every layer except the

output layer includes a bias neuron and is fully connected to the next layer. When an ANN has two or more hidden layers, it is called a deep neural network (DNN).

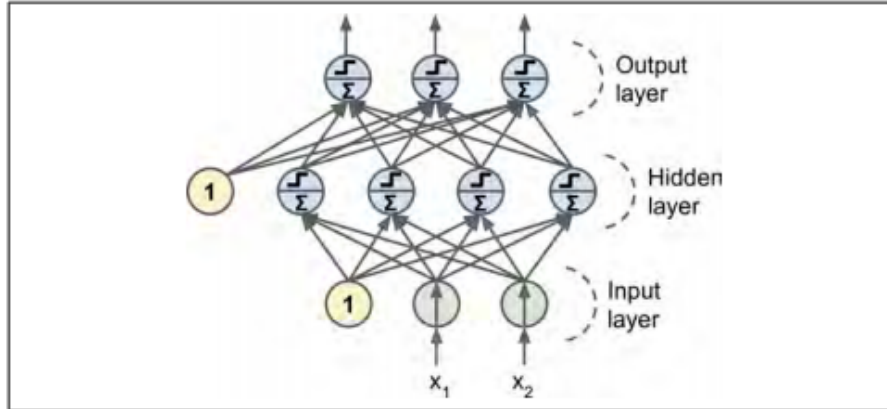


Figure 2.19: Multi-Layer Perceptron. Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" by Aurélien Géron

For many years researchers struggled to find a way to train MLPs, without success. But in 1986, D. E. Rumelhart et al. published a groundbreaking ("Learning representations by back-propagating errors") article introducing the back propagation training algorithm. Today we would describe it as Gradient Descent. For each training instance, the algorithm feeds it to the network and computes the output of every neuron in each consecutive layer (this is the forward pass, just like when making predictions). Then it measures the network's output error (i.e., the difference between the desired output and the actual output of the network), and it computes how much each neuron in the last hidden layer contributed to each output neuron's error. It then proceeds to measure how much of these error contributions came from each neuron in the previous hidden layer—and so on until the algorithm reaches the input layer. This reverse pass efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward in the network (hence the name of the algorithm).

To sum up, for each training instance the back propagation algorithm first makes a prediction (forward pass), measures the error, then goes through each layer in reverse to measure the error contribution from each connection (reverse pass), and finally slightly tweaks the connection weights to reduce the error (Gradient Descent step also called epoch).

In order for this algorithm to work properly, the authors made a key change to the MLP's architecture: they replaced the step function with the sigmoid function, $\sigma(z) = 1/(1 + e^{-z})$. This was essential because the step function contains only flat segments, so there is no gradient to work with (Gradient Descent cannot move on a flat surface), while the sigmoid

function has a well-defined nonzero derivative everywhere, allowing Gradient Descent to make some progress at every step. The back propagation algorithm may be used with other activation functions, instead of the sigmoid function.

However, the sigmoid activation function is slow to compute so recently, researchers have found a new activation function that works better, the ReLU function $ReLU(z) = \max(0, z)$. It is continuous but unfortunately not differentiable at $z = 0$ (the slope changes abruptly, which can make Gradient Descent bounce around). Although, in practice it works very well and has the advantage of being fast to compute. Additionally, the fact that it does not have a maximum output value also helps reduce some issues during Gradient Descent.

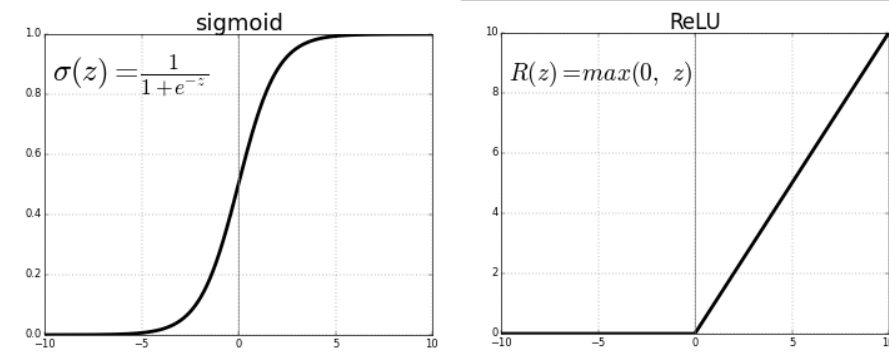


Figure 2.20: ReLU vs sigmoid. ReLU is faster to compute and doesn't compromise the performance of the model. Source: "<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>"

An MLP is often used for classification, with each output corresponding to a different binary class (face of the targeted person or not). When the classes are exclusive, the output layer is typically modified by replacing the individual activation functions by a shared softmax function. The output of each neuron corresponds to the estimated probability of the corresponding class. Note that the signal flows only in one direction (from the inputs to the outputs), so this architecture is an example of a feed forward neural network (FNN).

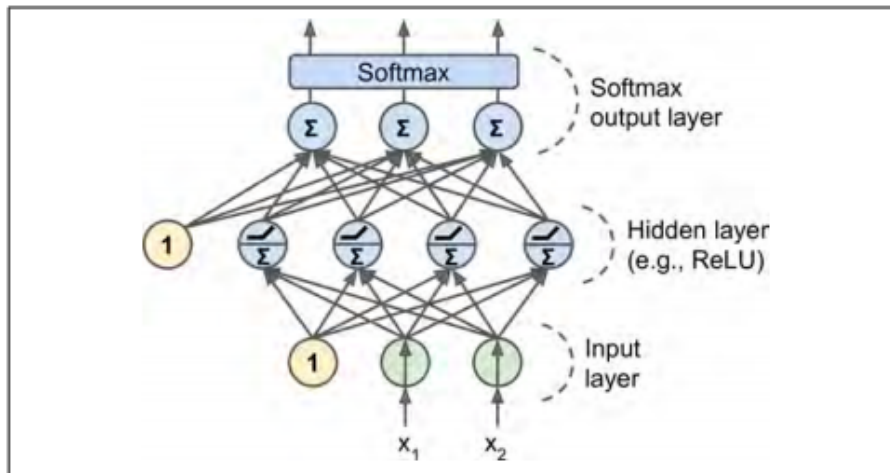


Figure 2.21: A modern MLP (including ReLU and softmax) for classification. Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow" by Aurélien Géron

In the use case of recognising a single person from the face image, the neural network has one output neuron with the probability of given specific input pixel values; they belong to the targeted person that it is needed to recognise. Therefore, the softmax function isn't needed, and it is perfectly reasonable to apply the sigmoid activation function to the output neuron to normalise the output value in the range between 0 and 1. Additionally, assigning a threshold is needed since we need to decide on which confidence we assume that the value outputted from the NN corresponds to the reality. In any probabilistic model, the decision of the value of the threshold is arbitrary since it depends on each problem. In this case, we are going to start with a value of 0.95 of threshold. So, if the model outputs a lower probability of 0.95, this prediction is considered as a negative case (value 0).

Some might say that the model is ready to train, but another important problem arises from the fact that the input data of this project are images. In all the examples above the input value was the vector of features $(x_1, x_2, x_3, \dots, x_n)$. However, since images are a matrix, as discussed in section 3, how do we transform an image into a one-dimensional vector? One possible answer is applying the flatten operation to the matrix. The flattening operation is to arrange the pixel values of the matrix by spatial position in a single vector. However, by doing that, all the spatial correlation between pixels is lost, and the model suffers a negative impact inaccuracy since, in images, the position of pixels is crucial information.

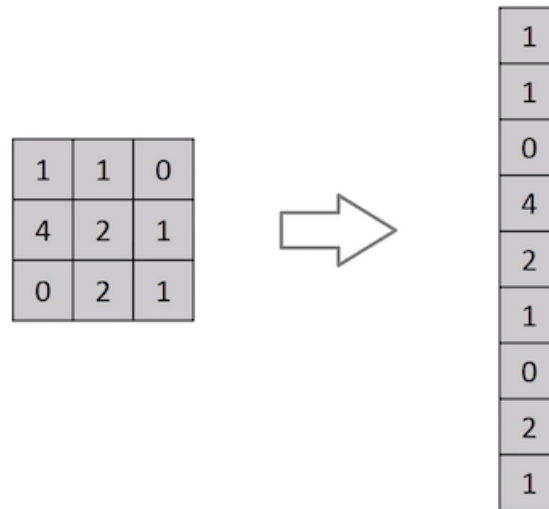


Figure 2.22: Flattening operation in a grey scale image (1 channel).
Source: "<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>"

Fortunately for us, a revolutionary paper named "Object Recognition with Gradient Based Learning" by LeCun et al. presenting a way of transforming image matrix into vector of features maintaining the spatial correlation between pixels: the convolutional operation layer.

2.8.3 Convolutional Neural Networks

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

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

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting

to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

The input image

In the figure, we have an RGB image which has been separated by its three color planes (Red, Green, and Blue).

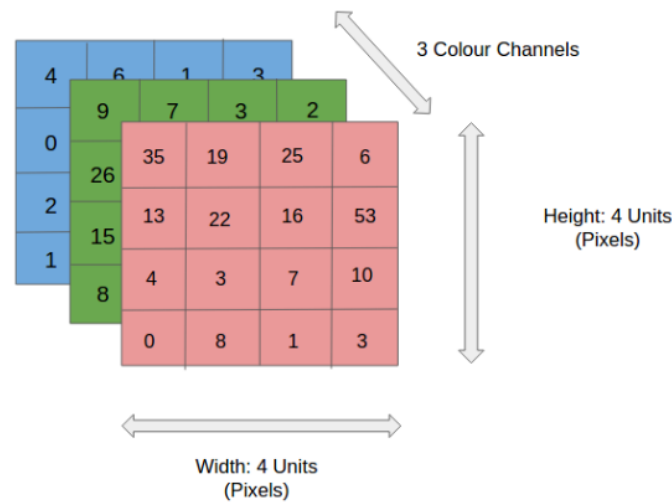


Figure 2.23: A RGB image. Source: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

Convolution Layer: The Kernel

The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow in the next figure. We have selected K as a 3x3x1 matrix with random values of filters. The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix

multiplication operation between K and the portion P of the image over which the kernel is hovering.

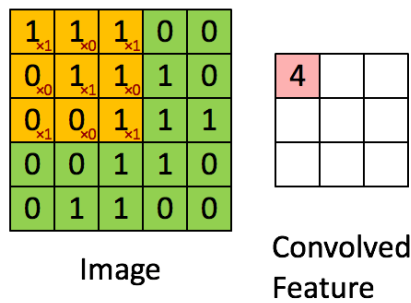


Figure 2.24: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature (step1). Source: "<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>"

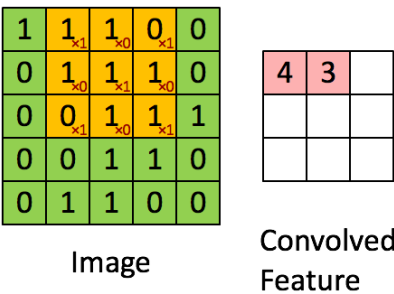


Figure 2.25: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature (step2). Source: "<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>"

In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between K_n and I_n stack ($[K_1, I_1]; [K_2, I_2]; [K_3, I_3]$) and all the results are summed with the bias to give us a squashed one-depth channel Convolved Feature Output.

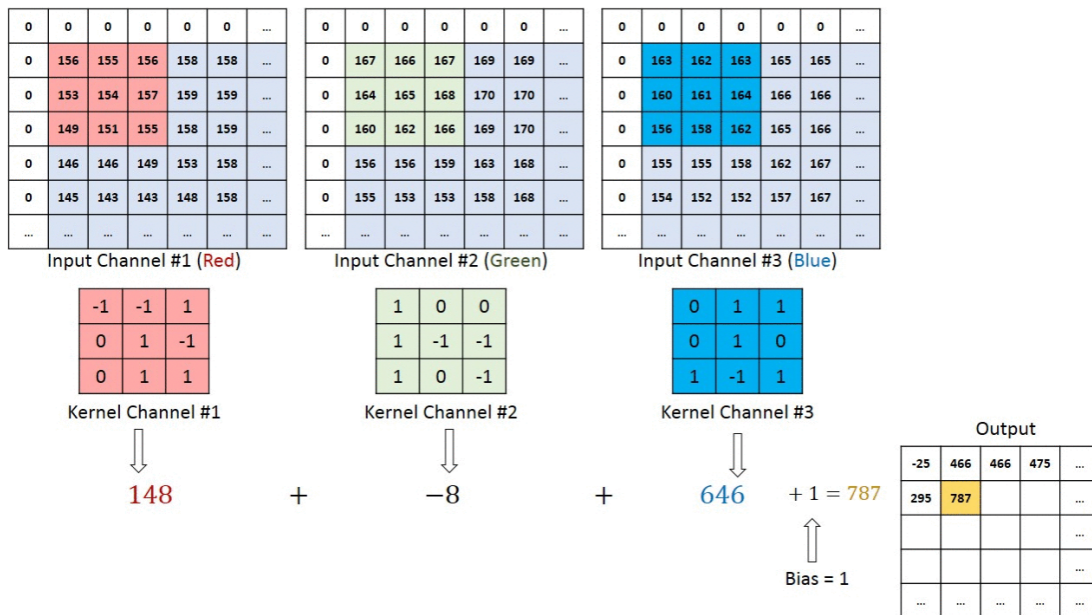


Figure 2.26: Convolution operation on a $M \times N \times 3$ image matrix with a $3 \times 3 \times 3$ Kernel. Source: "<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>"

All this filters values are going to be trained the same way the weights of the neurons are trained, by gradient descent. That's the strength of convolutional networks, they train themselves on how to extract the important features so they can feed a MLP that will output a prediction based on those features.

Pooling Layer

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

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.

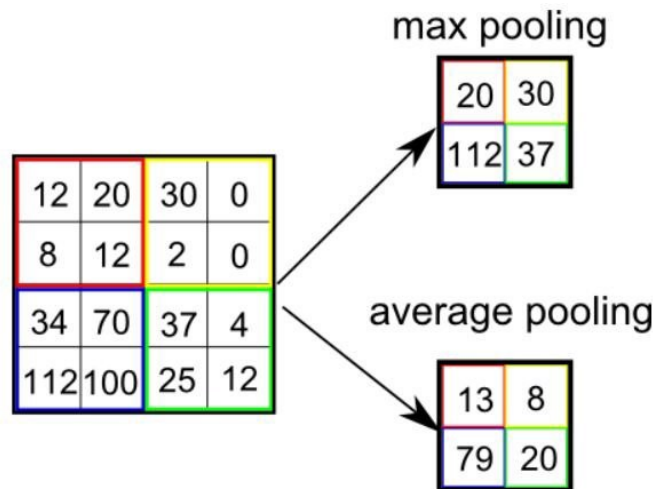


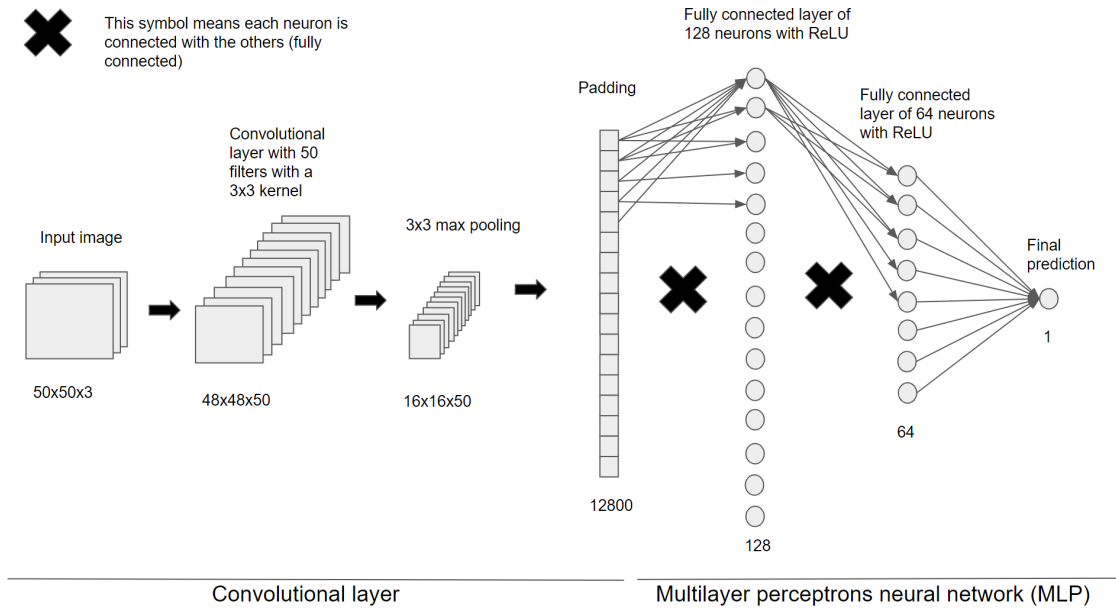
Figure 2.27: Types of Pooling. Source: "<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>"

The Convolutional Layer and the Pooling Layer, together form the i -th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes. In the next section, we are going to present how the final model looks like.

2.8.4 Model selected

The model is composed of a convolutional layer plus an MLP because of all the reasons previously discussed. The next figure shows a detailed view of the model of each operation with their parameters:

**Figure 2.28:** Final model

In the next figure, it is shown all the parameters for operation that the gradient descent will need to adjust in the training phase:

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 48, 48, 50)	1400
max_pooling2d_2 (MaxPooling2)	(None, 16, 16, 50)	0
flatten_2 (Flatten)	(None, 12800)	0
dense_4 (Dense)	(None, 128)	1638528
dense_5 (Dense)	(None, 64)	8256
dense_6 (Dense)	(None, 1)	65
Total params: 1,648,249		
Trainable params: 1,648,249		
Non-trainable params: 0		

Figure 2.29: Parameters of the neural network

For more information on why that number of neurons or why those filters in the convolu-

tional layer, check the annex. Adjusting those parameters (often called hyperparameters), it's achieved by doing a grid search (trial and error) until the models result in a good accuracy of the validation set.

2.9 Model training

In order to train the model, a very famous open source library is going to be used: TensorFlow.

2.9.1 TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 on November 9, 2015. The main reason TensorFlow is used is because it allows you to save the trained model and loaded with ease in other machine such a Raspberry Pi.

2.9.2 Training the model

As said previously, now we need to feed the model with the training images and apply gradient descent so the weights of the neurons and the values of the filters can be optimised. At first all of these values are randomised. For more information on the code of the training phase, please check "CNN.py" in the annex. Next figure, the accuracy of the training and validation set is displayed by epoch (gradient descent step). A 99.6 percent accuracy of the training set is achieved by the 5th epoch.

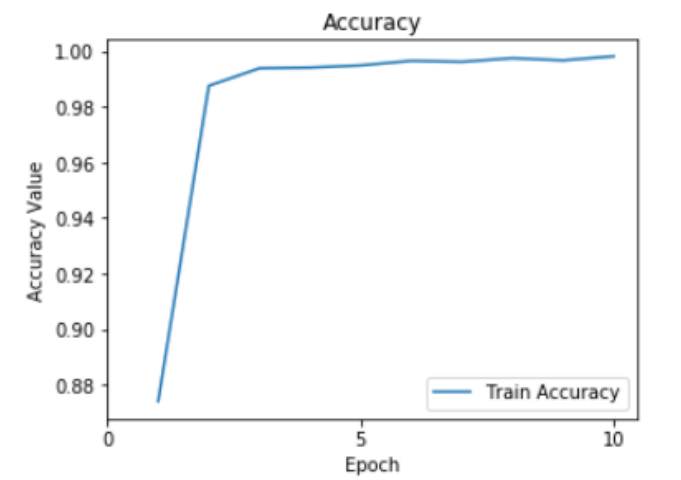


Figure 2.30: Train accuracy by epoch

2.10 Model evaluation

Let's test the model with images that it has never seen before: the test. Those are the results:

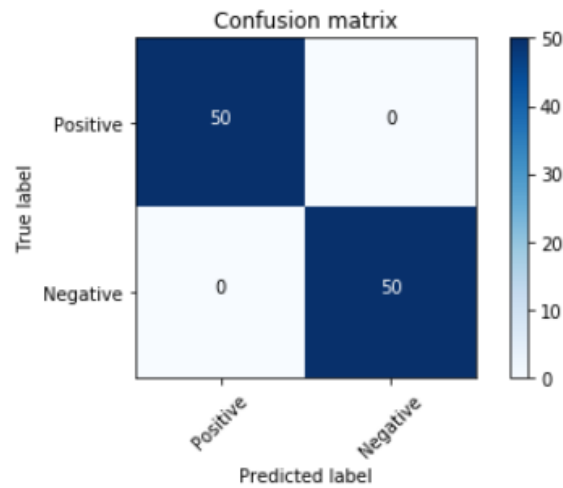


Figure 2.31: Confusion matrix of the result of the test set

Looking at the results, the model achieved 100 percent accuracy in the test set. Considering

these results, it is plausible to say that the model can recognise the targeted person. Although the model performed very well in the test set, another evaluation of the model implemented in the Raspberry Pi is done. Please check the section "Experimental results and evaluation of performance."

2.11 Model implementation into the Raspberry Pi

The next step is to implement the Viola-Jones algorithm and the model into the Raspberry Pi. First, we will present all the components that conform to the system for achieving the face detection and recognition in real-time.

2.11.1 Raspberry Pi 3 model B

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation. In this project, we are going to use the newest model the raspberry Pi 3 model B. This new model holds the following specifications:

- SoC: Broadcom BCM2837
- CPU: 4× ARM Cortex-A53, 1.2GHz
- GPU: Broadcom VideoCore IV
- RAM: 1GB LPDDR2 (900 MHz)
- Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
- Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
- Storage: microSD
- GPIO: 40-pin header, populated
- Ports: HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

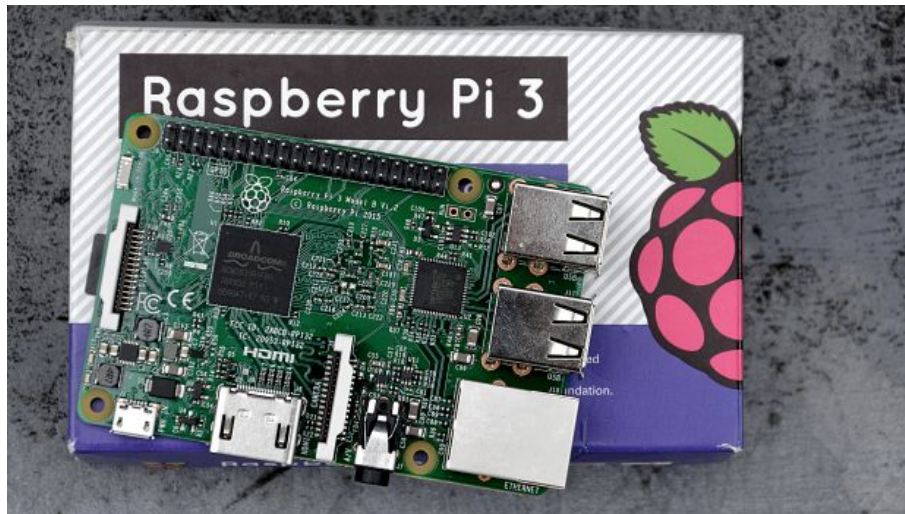


Figure 2.32: Raspberry Pi 3 Model B

The Raspberry Pi is the responsible of executing the Viola-Jones and the pre-trained convolutional neural network that we trained and saved so we can load it in the Raspberry Pi. Tensorflow saves the pre-trained model as a file (CNN.h5 file). For more information on how a pre-trained model is loaded in another device, refer to the code Raspberry in the annex. However, how the Raspberry Pi obtains the picture to analyse. For that, we use the PiCamera module.

2.11.2 PiCamera module

The Raspberry Pi Camera v2 is the new official camera board released by the Raspberry Pi Foundation. The Raspberry Pi Camera Module v2 is a high quality 8 megapixel Sony IMX219 image sensor custom designed add-on board for Raspberry Pi, featuring a fixed focus lens. It's capable of 3280 x 2464 pixel static images, and also supports 1080p30, 720p60 and 640x480p90 video. It attaches to Pi by way of one of the small sockets on the board upper surface and uses the dedicated CSI interface, designed especially for interfacing to cameras. Those are the specifications of the PiCamera module:

- Fixed focus lens on-board
- 8 megapixel native resolution sensor-capable of 3280 x 2464 pixel static images
- Supports 1080p30, 720p60 and 640x480p90 video
- Size 25mm x 23mm x 9mm

- Weight just over 3g
- Connects to the Raspberry Pi board via a short ribbon cable (supplied)
- Camera v2 is supported in the latest version of Raspbian, Raspberry Pi's preferred operating system



Figure 2.33: PiCamera Module

The PiCamera meets all the requirements in terms of image resolution, and it is easy to install and use in the Raspberry Pi. With these two components, the algorithm is up and running. However, it would be nice to see the output of it (no faces, face of the targeted person, etc.) so traffic lights connected to the GPIO are used to show the results. We use Pi-Traffic lights.

2.11.3 Pi-Traffic lights

The Pi-Traffic light provides an easy way to add three 10mm LEDs to your Raspberry Pi project. The red, yellow and green LEDs give visual feedback to the Raspberry Pi GPIO.

The Pi-Traffic light can also be used as a tool to learn about controlling the Raspberry Pi GPIO pins.

Features:

- Red, Yellow and Green 10mm LEDs
- Compatible with Pi Models A, B, A+, B+, 2, 3 and Zero
- Uses only four of the Raspberry Pi GPIO pins leaving the remaining GPIO pins for your project
- LEDs connected to GPIO pins 9,10 and 11
- Mounts vertically to the Raspberry Pi GPIO header

The Pi traffic lights will be connected in the Raspberry Pi as shown in the next picture.

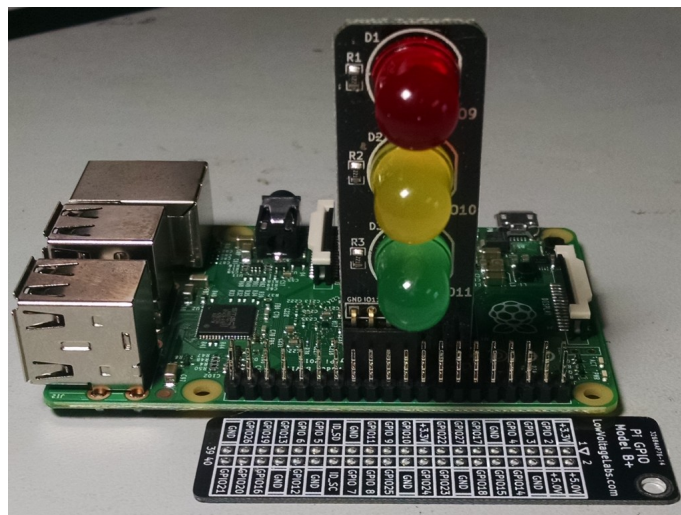


Figure 2.34: Raspberry Pi with the traffic lights

The way these traffic light works is:

- Green light if at least one face corresponds to the targeted person
- Red light is all faces detected belong to the negative class (non-targeted person)
- Yellow light if zero faces are detected

However, there is still the need for a final component. We need a way to input a signal to the Raspberry Pi, so it starts the process and executes the PiCamera to take a photo. We use a simple button structure.

2.11.4 Input signal

To give the system an input signal we will use the following subcomponents:

- A solderless breadboard
- 2 male-to-female jumper leads
- 1 male-to-male jumper lead
- A button

In order to connect the button to Raspberry Pi we will use the following circuit:

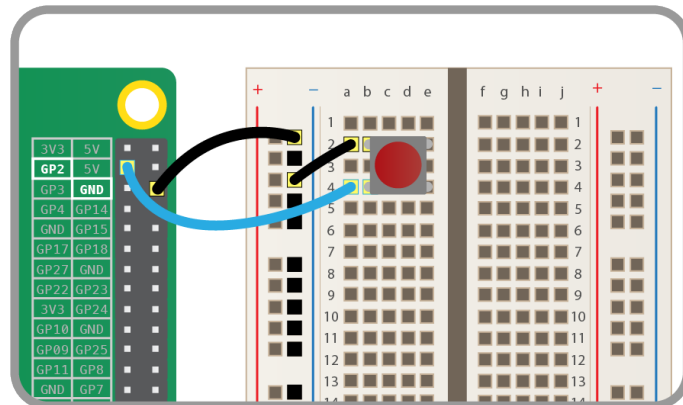


Figure 2.35: Button circuit. Source: <https://projects.raspberrypi.org/en/projects/physical-computing/2>

Finally, all the components are connected and the system is assembled and ready to function.

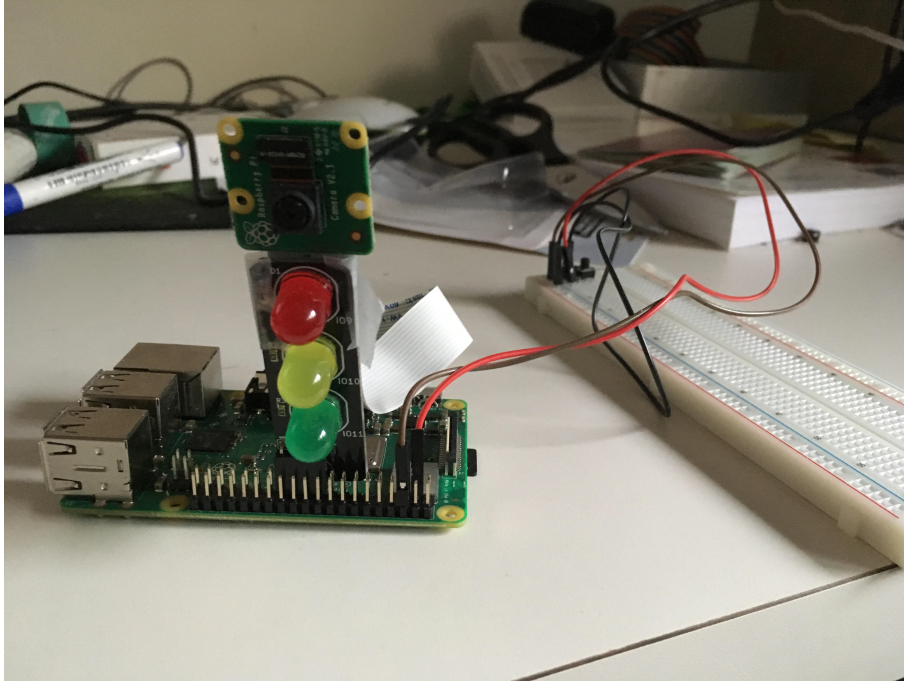


Figure 2.36: The Raspberry Pi 3 model B and all the components

2.11.5 Pipeline of the system

The pipeline of the system is very straightforward. Right after the Raspberry Pi is powered on, it executes the `Raspberry.py` (check annex to see the full code), and it remains in standby until the user presses the button. At this point, the PiCamera takes a photo, and the Viola-Jones algorithm detects all the faces of that image. For each face detected, the convolutional neural network outputs a prediction, and the traffic lights light up according to the results of those predictions.

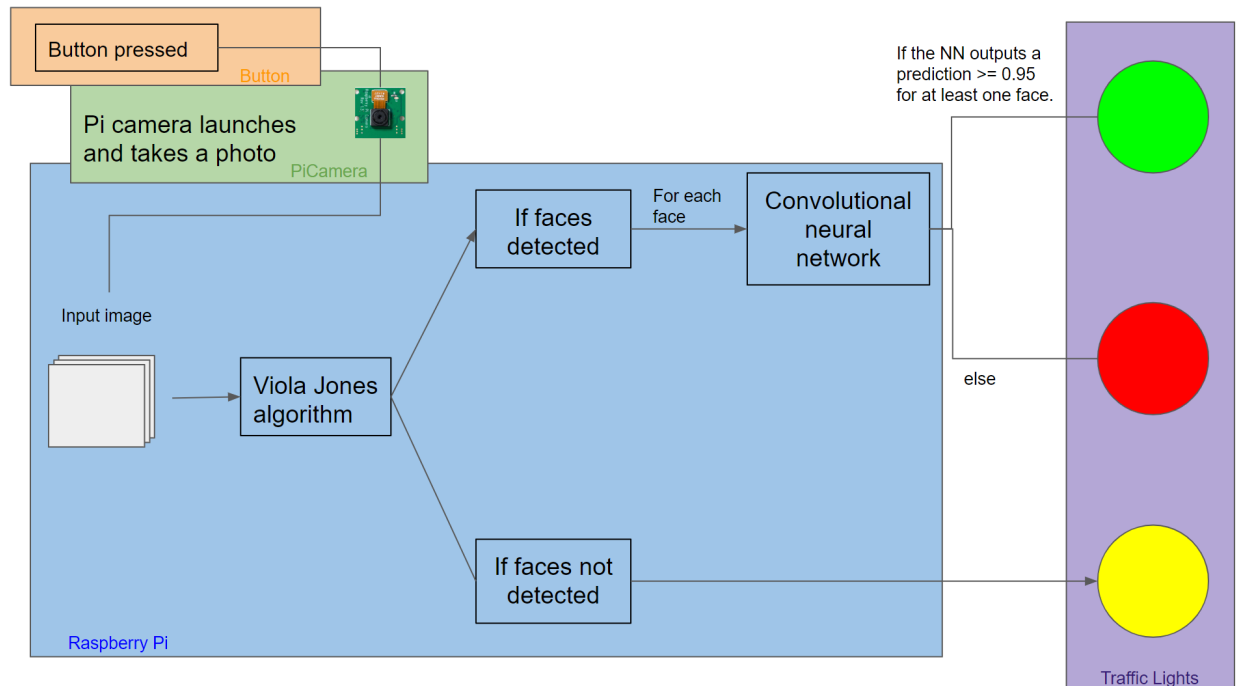


Figure 2.37: Pipeline of the process of detecting and recognising faces in real time in the Raspberry Pi

Finally, with the system functioning, we need to run tests to extract results and verify that the system is accomplishing the goal of detecting and recognising faces in real-time. In the next character, performance in terms of speed of the process and accuracy of the algorithm is evaluated.

2.12 Experimental results and performance evaluation

In this section, we are going to evaluate the performance of the project from the results taken by executing the pipeline of the system 113 times in different situations. We are going to evaluate the performance in terms of accuracy and speed. All these results can be found in the appendix, section "Experimental results."

2.12.1 Performance in terms of accuracy

Face detection

These are the results of the face detection algorithm Viola-Jones in images taken by the Raspberry Pi integrated camera (numbers in the matrix correspond to number of images. Respective percentages are computed below):

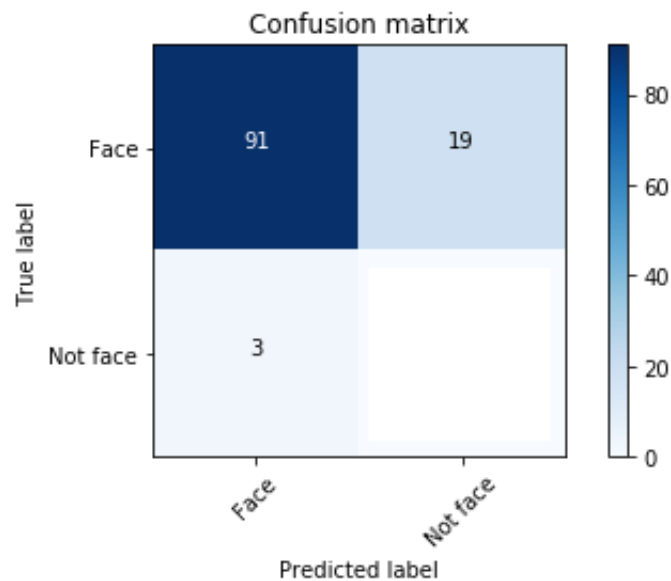


Figure 2.38: Experimental results from the Viola-Jones algorithm

- True positives (faces detected that were actual faces): 91 (83 percent of faces were detected)
- False negatives (faces not detected that were real faces): 19 (17 percent of faces were not detected). From this, 12 (63 percent) were due to the faces being rotated more than 10 degrees. This can be explained because the Haar Cascades features can't match rotated faces since they were hand-coded using not rotated images. Also, 3 faces were not detected due to low illumination, and the remaining false negatives (4) were not detected for unknown reasons.
- False positives (faces detected that were not real faces): 3. All of them for unknown reasons.
- The study of true negatives cannot be performed in a detection based algorithm.

Face recognition

These are the results from the face recognition algorithm in images taken by the Raspberry Pi integrated camera (numbers in the matrix correspond to number of images. Respective percentages are computed below):

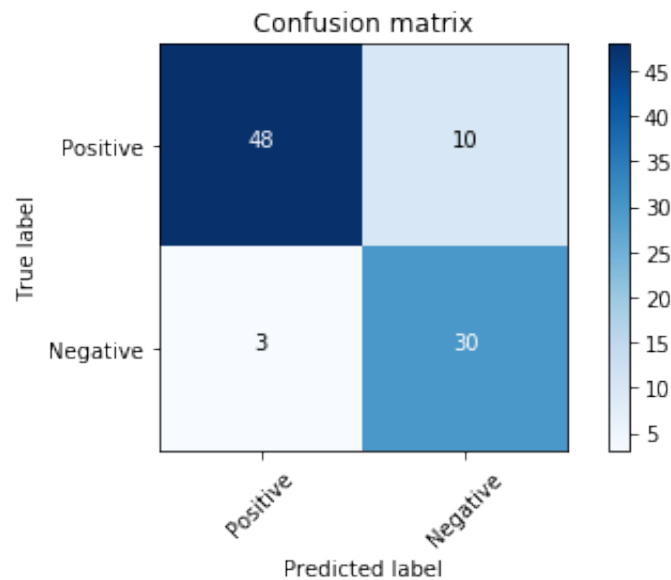


Figure 2.39: Experimental results from the convolutional neural network

- True positives: 48 (79 percent of times was the targeted person recognised)
- False negatives (the model not recognised the targeted person when it really was the actual person): 10 (20 percent of times was the targeted person not recognised)
- False positives (the model recognised the targeted person when it wasn't the actual person): 3. (1 percent of times a non targeted person was treated as the targeted person)
- True negatives: 30 (percent of times 70 percent of times a non targeted person was recognised as negative)

2.12.2 Performance in terms of speed

Image caption and reading

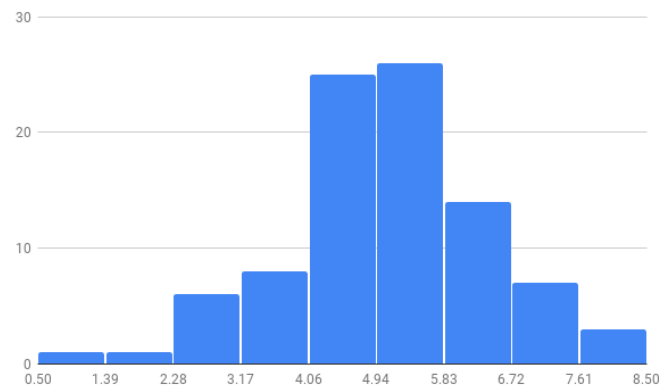


Figure 2.40: Histogram of time in seconds used for image caption and reading. Average = 4.97 seconds

Face detection

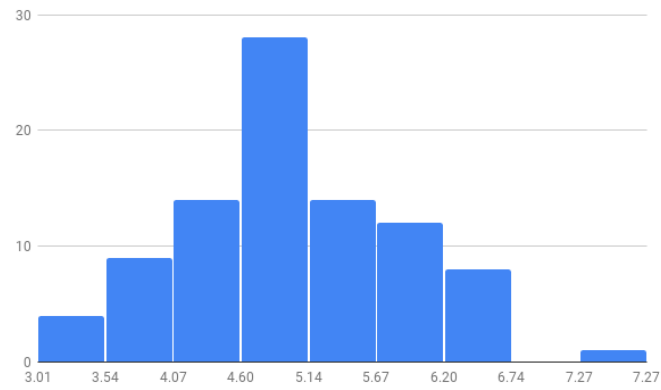


Figure 2.41: Histogram of time in seconds used for face detection. Average = 5.02 seconds

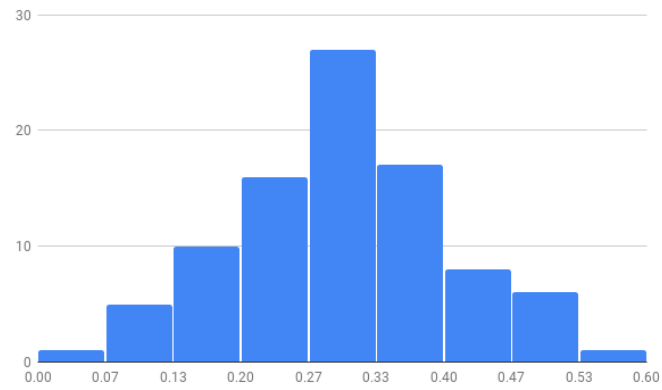
Face recognition

Figure 2.42: Histogram of time in seconds used for face recognition. Average = 0.307 seconds

Total time spent by the algorithm

On average the time the whole pipeline will take 10,297 seconds.

3 | SUMMARY OF RESULTS

3.1 Summary of the budget and study of economic viability

Next, the budget for the project is presented. This budget is divided into two main parts: the cost of the device and its components and the cost of labor.

3.1.1 Costs of the device and its components

In this section, the cost of the device that achieves face detection and recognition in real-time is presented.

Raspberry Pi	61 €
Picamera	20 €
Traffic lights	5 €
Solderless board	6 €
Power charger	20 €
Buttons and jumper leads	20 €
TOTAL	115 €

Table 3.1: Costs of the device and its components

3.1.2 Cost of labor

In this section, the cost of labor related to the realisation of the project is presented. Assuming that the worker will work 600 hours¹ at 10 €/hour, it amounts to 6000 € as total cost of labor.

¹Computed from the number of 24 ECTS of this project: 24 ECTS X 25 h = 600h.

3.1.3 Total cost

Costs of the device and its components	115 €
Cost of labor	6000 €
TOTAL	6115 €

Table 3.2: Total costs

3.2 Analysis and assessment of environmental implications

This project has very little environmental implications. However, it is worth mentioning that the Raspberry Pi is being manufactured according to the EU regulations regarding the supply of electronic/computer equipment [Cou16a]. This regulation constrains electronics manufacturers to reduce the amount of e-waste entering the refuse chain that ends up in landfill.

3.3 Conclusions and recommendations for future work

We can extract several conclusions from this project:

- The Viola-Jones algorithm performs well in most cases, except when faces suffer from big rotations. Even though the Viola-Jones is fast to compute so for machines with low computation power is the best option.
- In terms of recognition, deep learning achieved high accuracy, and therefore, we can say that it is a powerful technique at least in the field of computer vision. The only disadvantage is the data is needed to train the model, which sometimes is not easily obtained.

Looking at the results obtained in the facial recognition part, we could push the limits of the convolutional neural network and, in future work, we could train the model to recognise between several people. Instead of outputting the single probability of being the targeted person, our new model will output a vector of probability for each person we train the model to recognise. The procedures would be very similar to the ones being done in this project. However, the training phase would be done using images adequately labeled of the people we want to identify. The experimental study would have to be conducted again to conclude on multi-person case for accuracy and performance.

It would also be interesting to consider the case of a camera not attached to RPi, which would communicate remotely through wireless communication. Again, studying the real-time response would be interest to empirically study and conclude on its usefulness on real time applications that need face detection.

Bibliography

- [Heb49] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. New York: Wiley, June 1949. ISBN: 0-8058-4300-0.
- [Le+11] Quoc V Le et al. “On optimization methods for deep learning”. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress. 2011, pp. 265–272.
- [FSL15] Sachin Sudhakar Farfade, Mohammad J Saberian, and Li-Jia Li. “Multi-view face detection using deep convolutional neural networks”. In: *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. ACM. 2015, pp. 643–650.
- [HS15] Kaiming He and Jian Sun. “Convolutional neural networks at constrained time cost”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 5353–5360.
- [LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.
- [Liu+15] Ziwei Liu et al. “Deep learning face attributes in the wild”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 3730–3738.
- [Lon+15] Mingsheng Long et al. “Learning transferable features with deep adaptation networks”. In: *arXiv preprint arXiv:1502.02791* (2015).
- [Aba+16] Martin Abadi et al. “Tensorflow: A system for large-scale machine learning”. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.
- [Cou16a] Council of European Union. *Commission guidelines: Ecodesign requirements for computers and servers (June 2014)*. 2016.

- [Cou16b] Council of European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*. 2016.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [Ren+16] Shaoqing Ren et al. “Object detection networks on convolutional feature maps”. In: *IEEE transactions on pattern analysis and machine intelligence* 39.7 (2016), pp. 1476–1481.
- [Wit+16] Ian H Witten et al. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.
- [Cho17] François Chollet. *Deep Learning with Python*. Manning, 2017.
- [Gér17] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.
- [GP17] Antonio Gulli and Sujit Pal. *Deep Learning with Keras*. Packt Publishing Ltd, 2017.
- [Hua+17] Jonathan Huang et al. “Speed/accuracy trade-offs for modern convolutional object detectors”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7310–7311.
- [Ket17] Nikhil Ketkar. “Introduction to keras”. In: *Deep Learning with Python*. Springer, 2017, pp. 97–111.
- [TM18] H Taud and JF Mas. “Multilayer perceptron (mlp)”. In: *Geomatic Approaches for Modeling Land Change Scenarios*. Springer, 2018, pp. 451–455.
- [Gru19] Joel Grus. *Data science from scratch: first principles with python*. O'Reilly Media, 2019.
- [OAZ19] Ebenezer Owusu, Jamal-Deen Abdulai, and Yongzhao Zhan. “Face detection based on multilayer feed-forward neural network and Haar features”. In: *Software: Practice and Experience* 49.1 (2019), pp. 120–129.
- [Sto+19] Catalin Stoean et al. “On classifying images using Keras and Tensorflow in Python”. In: (2019).
- [ONLa] ONLINE. *A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way*. URL: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.

- [ONLb] ONLINE. *Having an Imbalanced Dataset? Here Is How You Can Fix It*. URL: <https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb>.

A | Appendix

A.1 Codes

A.1.1 VIOLA Jones data prep negative

```
1
2 # coding: utf-8
3
4 # In [1]:
5
6
7 get_ipython().run_line_magic('matplotlib', 'inline')
8
9 import numpy as np
10 import cv2
11 import os
12 import matplotlib.pyplot as plt
13 from os.path import isfile, join
14
15 DIM = (150, 150)
16
17 mypath = r"raw_data\negative"
18 onlyfiles = [f for f in os.listdir(mypath) if isfile(join(mypath, f))]
19 savepath = r"face_data\negative"
20
21 face_cascade = cv2.CascadeClassifier(r"haarcascade_frontalface_alt2.xml")
22
23 i=0
24 for image_path in onlyfiles:
25     image = cv2.imread(mypath + "\\" + image_path)
26
27     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
28
29     faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.1,
        minNeighbors = 5, minSize=(80, 80))
```

```

30     print("Found {0} faces!".format(len(faces)))
31     for (x,y,w,h) in faces:
32         roi_gray = gray[y:y+h, x:x+w]
33         roi_color = image[y:y+h, x:x+w]
34         resized_image = cv2.resize(roi_color, DIM)
35         img_item = "negative{i}.png".format(i=i)
36         cv2.imwrite(savepath + "\\" + img_item, resized_image)
37         #img = cv2.rectangle(image, (x,y), (x+w,y+w), (0,0,255), 8)
38         #img = img.astype('float32')
39         #img /= 255
40         #plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
41         #plt.show()
42         i+=1
43
44
45 # In [2]:
46
47
48 mypath = r"face_data\negative"
49 onlyfiles = [f for f in os.listdir(mypath) if isfile(join(mypath, f))]
50 savepath = r"final_data"
51 print(len(onlyfiles))
52 import random
53 number_test = 50
54 number_valid = 50
55 test_set = random.sample(onlyfiles, number_test)
56 print(len(test_set))
57 onlyfiles = [i for i in onlyfiles if i not in test_set]
58 print(len(onlyfiles))
59 valid_set = random.sample(onlyfiles, number_valid)
60 print(len(valid_set))
61 onlyfiles = [i for i in onlyfiles if i not in valid_set]
62 print(len(onlyfiles))
63
64
65 # In [3]:
66
67
68 for image_path in test_set:
69     image = cv2.imread(mypath + "\\" + image_path)
70     cv2.imwrite(savepath + "\\" + "test" + "\\" + image_path, image)
71 for image_path in valid_set:
72     image = cv2.imread(mypath + "\\" + image_path)
73     cv2.imwrite(savepath + "\\" + "validation" + "\\" + image_path, image)
74 for image_path in onlyfiles:
75     image = cv2.imread(mypath + "\\" + image_path)
76     cv2.imwrite(savepath + "\\" + "train" + "\\" + image_path, image)

```

A.1.2 VIOLA Jones data prep positive

```

1
2 # coding: utf-8
3
4 # In[2]:
5
6
7 get_ipython().run_line_magic('matplotlib', 'inline')
8
9 import numpy as np
10 import cv2
11 import os
12 import matplotlib.pyplot as plt
13 from os.path import isfile, join
14
15 DIM = (150, 150)
16 mypath = r"raw_data\positive"
17 onlyfiles = [f for f in os.listdir(mypath) if isfile(join(mypath, f))]
18 savepath = r"face_data\positive"
19
20 face_cascade = cv2.CascadeClassifier(r"haarcascade_frontalface_alt2.xml")
21
22 i=0
23 for image_path in onlyfiles:
24     image = cv2.imread(mypath + "\\ " + image_path)
25
26     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
27
28     faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.1,
29 minNeighbors = 5, minSize=(80, 80))
30     print(faces)
31     print("Found {0} faces!".format(len(faces[0])))
32     for (x,y,w,h) in faces[0]:
33         print(x,y,w,h)
34         roi_gray = gray[y:y+h, x:x+w]
35         roi_color = image[y:y+h, x:x+w]
36         #img_item = "my-image{i}.png".format(i=i)
37         #img = cv2.rectangle(image,(x,y),(x+w,y+w),(0,0,255),20)
38         #img = img.astype('float32')
39         #img /= 255
40         #plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
41         #plt.show()
42         i+=1
43         resized_image = cv2.resize(roi_color, DIM)
44         img_item = "positive{i}.png".format(i=i)
45         cv2.imwrite(savepath + "\\ " + img_item, resized_image)
46         print("_____")
47
48 # In[5]:
49

```

```

50
51 mypath = r"face_data\positive"
52 onlyfiles = [f for f in os.listdir(mypath) if isfile(join(mypath, f))]
53 savepath = r"final_data"
54 print(len(onlyfiles))
55 import random
56 number_test = 50
57 number_valid = 50
58 test_set = random.sample(onlyfiles, number_test)
59 print(len(test_set))
60 onlyfiles = [i for i in onlyfiles if i not in test_set]
61 print(len(onlyfiles))
62 valid_set = random.sample(onlyfiles, number_valid)
63 print(len(valid_set))
64
65
66 # In [6]:
67
68
69 for image_path in test_set:
70     image = cv2.imread(mypath + "\\" + image_path)
71     cv2.imwrite(savepath + "\\" + "test" + "\\" + image_path, image)
72 for image_path in valid_set:
73     image = cv2.imread(mypath + "\\" + image_path)
74     cv2.imwrite(savepath + "\\" + "validation" + "\\" + image_path, image)
75
76
77 # In [14]:
78
79
80 def plots(ims, count, figsize=(12,6), rows=1, interp=False, titles=None,
81          savepath = r"face_augmented_data\positive"):
82     if type(ims[0]) is np.ndarray:
83         ims = np.array(ims).astype(np.uint8)
84         if (ims.shape[-1] != 3):
85             ims = ims.transpose((0,2,3,1))
86     f = plt.figure(figsize=figsize)
87     cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
88     i=0
89     for i in range(len(ims)):
90         sp = f.add_subplot(rows, cols, i+1)
91         sp.axis("Off")
92         if titles is not None:
93             sp.set_title(titles[i], fontsize=16)
94         plt.imshow(ims[i], interpolation=None if interp else "none")
95         img_item = "positive{i}_{count}.png".format(i=i, count=count)
96         image_final=cv2.cvtColor(ims[i], cv2.COLOR_BGR2RGB)
97         cv2.imwrite(savepath + "\\" + img_item, image_final)
98

```

```

99 # In [ ]:
100
101
102 count = 0
103 from keras import backend as K
104 from keras.preprocessing.image import ImageDataGenerator
105 from scipy import misc, ndimage
106 for image_path in onlyfiles:
107     gen = ImageDataGenerator(rotation_range = 20, zoom_range=0.2)
108     image_path= r"face_data\positive" + "\\ " + image_path
109     image = np.expand_dims(ndimage.imread(image_path),0)
110     plt.imshow(image[0])
111     aug_iter = gen.flow(image)
112     aug_images = [next(aug_iter)[0].astype(np.uint8) for i in range(20)]
113     plots(aug_images, count, figsize=(20,7), rows = 2)
114     count = count + 1
115
116
117 # In [ ]:
118
119
120 mypath = r"face_data\positive"
121 onlyfiles = [f for f in os.listdir(mypath) if isfile(join(mypath, f))]
122 savepath = r"final_data"
123 print(len(onlyfiles))

```

A.1.3 Data augmentation

```

1
2 # coding: utf-8
3
4 # In [1]:
5
6
7 get_ipython().run_line_magic('matplotlib', 'inline')
8
9 import numpy as np
10 import cv2
11 import os
12 import matplotlib.pyplot as plt
13 from os.path import isfile, join
14 import keras
15 from keras import backend as K
16 from keras.preprocessing.image import ImageDataGenerator
17 from scipy import misc, ndimage
18
19
20 # In [2]:
21
22

```

```

23 def plots(ims, figsize=(12,6),rows=1, interp=False, titles=None,savepath = r
    "face_augmented_data\positive"):
24     if type(ims[0]) is np.ndarray:
25         ims = np.array(ims).astype(np.uint8)
26         if (ims.shape[-1] != 3):
27             ims = ims.transpose((0,2,3,1))
28     f = plt.figure(figsize=figsize)
29     cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows +1
30     i=0
31     for i in range(len(ims)):
32         sp = f.add_subplot(rows, cols, i+1)
33         sp.axis("Off")
34         if titles is not None:
35             sp.set_title(titles[i], fontsize=16)
36         plt.imshow(ims[i], interpolation=None if interp else "none")
37         img_item = "positive{i}.png".format(i=i)
38         image_final=cv2.cvtColor(ims[i], cv2.COLOR_BGR2RGB)
39         cv2.imwrite(savepath + "\\\" + img_item,image_final)
40
41
42 # In [4]:
43
44
45 gen = ImageDataGenerator(rotation_range = 20, zoom_range=0.2)
46 image_path= r"face_data\positive\positive10.png"
47 image = np.expand_dims(ndimage.imread(image_path),0)
48 plt.imshow(image[0])
49 aug_iter = gen.flow(image)
50 aug_images = [next(aug_iter)[0].astype(np.uint8) for i in range(20)]
51 plots(aug_images, figsize=(20,7),rows = 2)

```

A.1.4 CNN

```

1
2 # coding: utf-8
3
4 # In [9]:
5
6
7 import glob
8 import numpy as np
9 import os
10 import shutil
11 np.random.seed(42)
12 import numpy as np
13 import matplotlib.pyplot as plt
14 from keras.preprocessing.image import ImageDataGenerator, load_img,
    img_to_array, array_to_img
15
16

```

```

17 # In[10]:
18
19
20 files = glob.glob('final_data/train/*')
21
22 positive_files = [fn for fn in files if 'pos' in fn]
23 negative_files = [fn for fn in files if 'neg' in fn]
24 len(positive_files), len(negative_files)
25
26
27 # In[11]:
28
29
30 IMG_DIM = (50,50)
31
32 train_files = glob.glob('final_data/train/*')
33 train_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in
    train_files]
34 train_imgs = np.array(train_imgs)
35 train_labels = [fn.split('\\')[1].split('.')[0].strip()[0] for fn in
    train_files]
36
37 validation_files = glob.glob('final_data/validation/*')
38 validation_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img
    in validation_files]
39 validation_imgs = np.array(validation_imgs)
40 validation_labels = [fn.split('\\')[1].split('.')[0].strip()[0] for fn in
    validation_files]
41
42 print('Train dataset shape:', train_imgs.shape,
43       '\tValidation dataset shape:', validation_imgs.shape)
44
45
46 # In[12]:
47
48
49 train_imgs_scaled = train_imgs.astype('float32')
50 validation_imgs_scaled = validation_imgs.astype('float32')
51 train_imgs_scaled /= 255
52 validation_imgs_scaled /= 255
53
54
55 # In[13]:
56
57
58 batch_size = 100
59 num_classes = 2
60 epochs = 10
61 input_shape = (50, 50, 3)
62

```

```

63 # encode text category labels
64 from sklearn.preprocessing import LabelEncoder
65
66 le = LabelEncoder()
67 le.fit(train_labels)
68 train_labels_enc = le.transform(train_labels)
69 validation_labels_enc = le.transform(validation_labels)
70
71
72 # In [14]:
73
74
75 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
76 from keras.models import Sequential
77 from keras import optimizers
78
79 model = Sequential()
80
81 model.add(Conv2D(50, kernel_size=(3, 3), activation='relu',
82                 input_shape=input_shape))
83 model.add(MaxPooling2D(pool_size=(3, 3)))
84 model.add(Flatten())
85 model.add(Dense(128, activation='relu'))
86 model.add(Dense(64, activation='relu'))
87 model.add(Dense(1, activation='sigmoid'))
88
89
90 model.compile(loss='binary_crossentropy',
91               optimizer=optimizers.RMSprop(),
92               metrics=['accuracy'])
93
94 model.summary()
95
96
97 # In [15]:
98
99
100 history = model.fit(x=train_imgs_scaled, y=train_labels_enc,
101                    validation_data=(validation_imgs_scaled,
102                                    validation_labels_enc),
103                    batch_size=batch_size,
104                    epochs=epochs,
105                    verbose=1)
106
107 # In [20]:
108
109
110 f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
111 t = f.suptitle('CNN Performance', fontsize=12)

```



```

112 f.subplots_adjust(top=0.85, wspace=0.3)
113
114 epoch_list = list(range(1,11))
115 ax1.plot(epoch_list, history.history['acc'], label='Train Accuracy')
116 ax1.set_xticks(np.arange(0, 11, 5))
117 ax1.set_ylabel('Accuracy Value')
118 ax1.set_xlabel('Epoch')
119 ax1.set_title('Accuracy')
120 l1 = ax1.legend(loc="best")
121
122 ax2.plot(epoch_list, history.history['loss'], label='Train Loss')
123 ax2.plot(epoch_list, history.history['val_loss'], label='Validation Loss')
124 ax2.set_xticks(np.arange(0, 11, 5))
125 ax2.set_ylabel('Loss Value')
126 ax2.set_xlabel('Epoch')
127 ax2.set_title('Loss')
128 l2 = ax2.legend(loc="best")
129
130
131 # In[ ]:
132
133
134 get_ipython().run_line_magic('matplotlib', 'inline')
135 import cv2
136 test_files = glob.glob('final_data/test/*')
137 test_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in
138               test_files]
139 test_imgs_display = [img_to_array(load_img(img, target_size=(200,200))) for
140                       img in test_files]
141 test_imgs_display = np.array(test_imgs_display)
142 test_imgs_display = test_imgs_display.astype('float32')
143 test_imgs_display /=255
144 test_imgs = np.array(test_imgs)
145 test_imgs_scaled = test_imgs.astype('float32')
146 test_imgs_scaled /= 255
147
148 for i in range(len(test_imgs)):
149     print(model.predict(np.expand_dims(test_imgs_scaled[i], axis=0)))
150     plt.imshow(test_imgs_display[i])
151     plt.show()
152
153 # In[ ]:
154
155 model.save("models/simpleCNN.h5")

```

A.1.5 Raspberry

```

1 import RPi.GPIO as GPIO

```

```

2 import time
3 import cv2
4 import matplotlib.pyplot as plt
5 from picamera import PiCamera
6 from time import sleep
7 import datetime
8 import os
9 import keras
10 from keras.models import load_model
11 import numpy as np
12 from gpiozero import Button
13 import time
14 import subprocess
15 import RPi.GPIO as GPIO
16
17 button = Button(2)
18
19 def rotate_image(image, angle):
20     if angle == 0: return image
21     height, width = image.shape[:2]
22     rot_mat = cv2.getRotationMatrix2D((width/2, height/2), angle, 0.9)
23     result = cv2.warpAffine(image, rot_mat, (width, height), flags=cv2.
INTER_LINEAR)
24     return result
25
26 #GPIO.setwarnings(False)
27 # Pin Setup:
28 GPIO.setmode(GPIO.BCM) # Broadcom pin-numbering scheme. This uses the pin
    numbers that
29 # match the pin numbers on the Pi Traffic light.
30
31 GPIO.setup(9, GPIO.OUT) # Red LED pin set as output
32 GPIO.setup(10, GPIO.OUT) # Yellow LED pin set as output
33 GPIO.setup(11, GPIO.OUT) # Green LED pin set as output
34
35 GPIO.output(9, True) # Turns on the Red LED
36 GPIO.output(10, True) # Turns on the Red LED
37 GPIO.output(11, True) # Turns on the Red LED
38
39 time.sleep(1)
40
41 GPIO.output(9, False) # Turns on the Red LED
42 GPIO.output(10, False) # Turns on the Red LED
43 GPIO.output(11, False) # Turns on the Red LED
44 model = load_model(r"/home/pi/Desktop/simpleCNN.h5")
45
46 camera = PiCamera()
47
48 face_cascade = cv2.CascadeClassifier(r"/home/pi/Desktop/
    haarcascade_frontalface_alt2.xml")

```

```

49 settings = {
50     'scaleFactor': 1.1,
51     'minNeighbors': 5,
52     'minSize': (20, 20)
53 }
54
55 #setup GPIO using Broadcom SOC channel numbering
56 for times in range(30):
57     print("System Ready")
58
59     button.wait_for_press()
60     camera.start_preview()
61     sleep(3)
62     now = datetime.datetime.now().strftime("%Y_%m_%d%H:%M:%S")
63     newpath = r'/home/pi/Data/{now}'.format(now=now)
64     if not os.path.exists(newpath):
65         os.makedirs(newpath)
66
67     print(newpath)
68     img_path = r'/home/pi/Data/{now}/originalphoto.png'.format(now=now)
69     camera.capture(img_path)
70     camera.stop_preview()
71
72
73
74     i=0
75     image = cv2.imread(img_path)
76     image2 = cv2.imread(img_path)
77     is_positive = 0
78     no_faces = 0
79
80     for angle in range(0,1):
81         image_ro = rotate_image(image, angle)
82         cv2.imwrite(r"/home/pi/Data/{now}/image_{angle}.png".format(angle=
angle,now=now),image_ro)
83         gray = cv2.cvtColor(image_ro, cv2.COLOR_BGR2GRAY)
84
85         DIM = (50,50)
86         faces = face_cascade.detectMultiScale(gray, **settings)
87         print("Found {0} faces!".format(len(faces)))
88
89
90         for (x,y,w,h) in faces:
91             no_faces = 1
92             roi_gray = gray[y:y+h, x:x+w]
93             roi_color = image_ro[y:y+h, x:x+w]
94             resized_image = cv2.resize(roi_color, DIM)
95             img = resized_image.astype('float32')
96             img /= 255
97             prediction = model.predict(np.expand_dims(img, axis=0))

```

```

98         cv2.imwrite(r"/home/pi/Data/{now}/face_{i}_{prediction}.png".
format(i=i,now=now,prediction=prediction),roi_color)
99         print(prediction)
100         #plt.imshow(cv2.cvtColor(resized_image, cv2.
COLOR_BGR2RGB))
101         #plt.show()
102         i+=1
103         if prediction >= 0.9:
104             is_positive = 1
105             cv2.rectangle(image2, (x, y), (x+w, y+h), (0,255,0), 2)
106         else:
107             cv2.rectangle(image2, (x, y), (x+w, y+h), (0,0,255), 2)
108
109     cv2.imwrite(r"/home/pi/Data/{now}/experiment.png".format(now=now),image2
)
110     if is_positive == 1:
111         GPIO.output(11, True) # Turns on the Red LED
112         #GPIO.output(11, True) # Turns on the Red LED
113
114         time.sleep(10)
115         # Set the pin LOW
116
117         GPIO.output(11, False) # Turns on the Red LED
118         #GPIO.output(11, False) # Turns on the Red LED
119
120     elif no_faces == 0:
121         GPIO.output(10, True) # Turns on the Red LED
122         time.sleep(10)
123         GPIO.output(10, False)
124
125     else:
126         GPIO.output(9, True) # Turns on the Red LED
127         #GPIO.output(11, True) # Turns on the Red LED
128
129         time.sleep(10)
130         # Set the pin LOW
131
132         GPIO.output(9, False) # Turns on the Red LED
133         #GPIO.output(11, False) # Turns on the Red LED

```

A.2 Experimental results

Viola Jones			
Experiment number	Ground truth	Algorithm prediction	Reason
experiemnt1	TRUE	TRUE	
experiemnt2	TRUE	TRUE	
experiemnt3	TRUE	TRUE	
experiemnt4	TRUE	FALSE	Too dark
experiemnt5	TRUE	TRUE	
experiemnt6	TRUE	TRUE	
experiemnt7	TRUE	TRUE	
experiemnt8	TRUE	FALSE	Rotation
experiemnt9	TRUE	TRUE	
experiemnt10	FALSE	TRUE	
experiemnt11	TRUE	TRUE	
experiemnt12	TRUE	TRUE	
experiemnt13	TRUE	FALSE	Rotation
experiemnt14	TRUE	TRUE	
experiemnt15	TRUE	TRUE	
experiemnt16	TRUE	TRUE	
experiemnt17	TRUE	TRUE	
experiemnt18	TRUE	TRUE	
experiemnt19	TRUE	TRUE	
experiemnt20	TRUE	TRUE	
experiemnt21	TRUE	TRUE	
experiemnt22	FALSE	TRUE	
experiemnt23	TRUE	FALSE	Rotation
experiemnt24	TRUE	TRUE	
experiemnt25	TRUE	FALSE	Rotation
experiemnt26	TRUE	TRUE	
experiemnt27	TRUE	TRUE	
experiemnt28	TRUE	FALSE	NA

experiemnt29	TRUE	TRUE	
experiemnt30	TRUE	TRUE	
experiemnt31	TRUE	TRUE	
experiemnt32	TRUE	TRUE	
experiemnt33	TRUE	TRUE	
experiemnt34	TRUE	TRUE	
experiemnt35	TRUE	TRUE	
experiemnt36	TRUE	TRUE	
experiemnt37	TRUE	TRUE	
experiemnt38	TRUE	TRUE	
experiemnt39	TRUE	FALSE	Rotation
experiemnt40	TRUE	TRUE	
experiemnt41	TRUE	TRUE	
experiemnt42	TRUE	TRUE	
experiemnt43	TRUE	TRUE	
experiemnt44	TRUE	TRUE	
experiemnt45	TRUE	FALSE	NA
experiemnt46	TRUE	TRUE	
experiemnt47	TRUE	TRUE	
experiemnt48	TRUE	FALSE	Too dark
experiemnt49	TRUE	TRUE	
experiemnt50	TRUE	TRUE	
experiemnt51	TRUE	TRUE	
experiemnt52	TRUE	TRUE	
experiemnt53	TRUE	TRUE	
experiemnt54	TRUE	FALSE	Rotation
experiemnt55	TRUE	TRUE	
experiemnt56	TRUE	TRUE	
experiemnt57	TRUE	TRUE	
experiemnt58	TRUE	TRUE	
experiemnt59	TRUE	TRUE	

experiemnt60	TRUE	TRUE	
experiemnt61	TRUE	TRUE	
experiemnt62	TRUE	FALSE	Rotation
experiemnt63	TRUE	TRUE	
experiemnt64	TRUE	TRUE	
experiemnt65	TRUE	TRUE	
experiemnt66	TRUE	FALSE	Rotation
experiemnt67	TRUE	TRUE	
experiemnt68	TRUE	TRUE	
experiemnt69	TRUE	FALSE	NA
experiemnt70	TRUE	TRUE	
experiemnt71	TRUE	TRUE	
experiemnt72	TRUE	TRUE	
experiemnt73	TRUE	TRUE	
experiemnt74	TRUE	FALSE	Rotation
experiemnt75	TRUE	TRUE	
experiemnt76	TRUE	TRUE	
experiemnt77	TRUE	TRUE	
experiemnt78	TRUE	TRUE	
experiemnt79	TRUE	TRUE	
experiemnt80	TRUE	TRUE	
experiemnt81	TRUE	TRUE	
experiemnt82	TRUE	TRUE	
experiemnt83	FALSE	TRUE	
experiemnt84	TRUE	TRUE	
experiemnt85	TRUE	TRUE	
experiemnt86	TRUE	FALSE	Rotation
experiemnt87	TRUE	TRUE	
experiemnt88	TRUE	FALSE	Too dark
experiemnt89	TRUE	TRUE	
experiemnt90	TRUE	TRUE	

experiemnt91	TRUE	TRUE	
experiemnt92	TRUE	TRUE	
experiemnt93	TRUE	TRUE	
experiemnt94	TRUE	TRUE	
experiemnt95	TRUE	TRUE	
experiemnt96	TRUE	TRUE	
experiemnt97	TRUE	TRUE	
experiemnt98	TRUE	TRUE	
experiemnt99	TRUE	FALSE	NA
experiemnt100	TRUE	TRUE	
experiemnt101	TRUE	TRUE	
experiemnt102	TRUE	TRUE	
experiemnt103	TRUE	TRUE	
experiemnt104	TRUE	TRUE	
experiemnt105	TRUE	TRUE	
experiemnt106	TRUE	TRUE	
experiemnt107	TRUE	TRUE	
experiemnt108	TRUE	TRUE	
experiemnt109	TRUE	TRUE	
experiemnt110	TRUE	FALSE	Rotation
experiemnt111	TRUE	TRUE	
experiemnt112	TRUE	TRUE	
experiemnt113	TRUE	FALSE	Rotation
experiemnt114	TRUE	TRUE	

Convolutional Neural Network		
Experiment num	Ground truth	Algorithm prediction
experiemnt1	TRUE	TRUE
experiemnt2	TRUE	FALSE
experiemnt3	TRUE	TRUE
experiemnt4	TRUE	TRUE
experiemnt5	TRUE	TRUE
experiemnt6	FALSE	FALSE
experiemnt7	FALSE	FALSE
experiemnt8	FALSE	FALSE
experiemnt9	FALSE	FALSE
experiemnt10	FALSE	FALSE
experiemnt11	FALSE	FALSE
experiemnt12	FALSE	FALSE
experiemnt13	FALSE	FALSE
experiemnt14	FALSE	FALSE
experiemnt15	FALSE	FALSE
experiemnt16	FALSE	FALSE
experiemnt17	FALSE	FALSE
experiemnt18	FALSE	FALSE
experiemnt19	FALSE	FALSE
experiemnt20	FALSE	FALSE
experiemnt21	FALSE	FALSE
experiemnt22	FALSE	FALSE
experiemnt23	FALSE	FALSE
experiemnt24	FALSE	FALSE
experiemnt25	FALSE	FALSE
experiemnt26	FALSE	FALSE
experiemnt27	TRUE	TRUE
experiemnt28	TRUE	TRUE

experiemnt29	TRUE	TRUE
experiemnt30	TRUE	TRUE
experiemnt31	TRUE	FALSE
experiemnt32	TRUE	TRUE
experiemnt33	TRUE	TRUE
experiemnt34	TRUE	TRUE
experiemnt35	TRUE	TRUE
experiemnt36	TRUE	TRUE
experiemnt37	TRUE	TRUE
experiemnt38	TRUE	TRUE
experiemnt39	FALSE	FALSE
experiemnt40	FALSE	TRUE
experiemnt41	FALSE	TRUE
experiemnt42	FALSE	FALSE
experiemnt43	TRUE	TRUE
experiemnt44	TRUE	TRUE
experiemnt45	TRUE	TRUE
experiemnt46	TRUE	TRUE
experiemnt47	TRUE	TRUE
experiemnt48	TRUE	TRUE
experiemnt49	TRUE	TRUE
experiemnt50	TRUE	TRUE
experiemnt51	TRUE	FALSE
experiemnt52	TRUE	TRUE
experiemnt53	TRUE	TRUE
experiemnt54	TRUE	TRUE
experiemnt55	TRUE	TRUE
experiemnt56	TRUE	TRUE
experiemnt57	TRUE	TRUE
experiemnt58	TRUE	TRUE
experiemnt59	FALSE	FALSE

experiemnt60	FALSE	FALSE
experiemnt61	FALSE	FALSE
experiemnt62	FALSE	FALSE
experiemnt63	FALSE	TRUE
experiemnt64	FALSE	FALSE
experiemnt65	FALSE	FALSE
experiemnt66	TRUE	TRUE
experiemnt67	TRUE	FALSE
experiemnt68	TRUE	TRUE
experiemnt69	TRUE	TRUE
experiemnt70	TRUE	FALSE
experiemnt71	TRUE	TRUE
experiemnt72	TRUE	TRUE
experiemnt73	TRUE	TRUE
experiemnt74	TRUE	FALSE
experiemnt75	TRUE	TRUE
experiemnt76	TRUE	FALSE
experiemnt77	TRUE	TRUE
experiemnt78	FALSE	FALSE
experiemnt79	TRUE	TRUE
experiemnt80	TRUE	TRUE
experiemnt81	TRUE	TRUE
experiemnt82	TRUE	TRUE
experiemnt83	TRUE	TRUE
experiemnt84	TRUE	TRUE
experiemnt85	TRUE	TRUE
experiemnt86	TRUE	FALSE
experiemnt87	TRUE	TRUE
experiemnt88	TRUE	TRUE
experiemnt89	TRUE	FALSE
experiemnt90	TRUE	TRUE

experimnt91	TRUE	FALSE
-------------	------	-------

Speed			
Experiment num	Image	Detection	Recognition
experiemnt1	4.95	4.79	0.38
experiemnt2	5.36	5.19	0.15
experiemnt3	5.65	5.52	0.15
experiemnt4	5.11	3.43	0.28
experiemnt5	4.41	5.49	0.16
experiemnt6	6.03	4.99	0.16
experiemnt7	6.92	4.09	0.29
experiemnt8	6.05	5.36	0.29
experiemnt9	7.29	6.22	0.62
experiemnt10	6.20	4.84	0.24
experiemnt11	5.40	6.35	0.24
experiemnt12	4.13	3.23	0.26
experiemnt13	5.44	5.55	0.30
experiemnt14	5.22	5.86	0.27
experiemnt15	5.96	5.03	0.38
experiemnt16	4.16	5.23	0.12
experiemnt17	3.25	5.25	-0.01
experiemnt18	5.82	5.18	0.25
experiemnt19	4.24	5.11	0.33
experiemnt20	3.32	5.03	0.31
experiemnt21	4.02	4.71	0.34
experiemnt22	5.84	3.73	0.35
experiemnt23	3.28	4.36	0.37
experiemnt24	6.87	5.88	0.28
experiemnt25	3.81	4.57	0.14
experiemnt26	3.90	5.13	0.21
experiemnt27	4.49	4.57	0.34
experiemnt28	3.49	5.20	0.29

experiemnt29	5.65	4.68	0.23
experiemnt30	5.78	5.81	0.36
experiemnt31	4.54	4.07	0.41
experiemnt32	5.53	4.37	0.37
experiemnt33	5.50	5.27	0.33
experiemnt34	4.51	5.48	0.33
experiemnt35	2.62	6.09	0.45
experiemnt36	0.23	5.83	0.40
experiemnt37	3.72	5.78	0.17
experiemnt38	4.39	5.30	0.38
experiemnt39	5.48	3.64	0.27
experiemnt40	5.45	6.25	0.16
experiemnt41	4.58	6.31	0.23
experiemnt42	8.70	4.55	0.21
experiemnt43	4.48	4.88	0.11
experiemnt44	5.20	4.72	0.26
experiemnt45	4.54	4.31	0.35
experiemnt46	6.61	3.43	0.20
experiemnt47	5.12	3.34	0.52
experiemnt48	4.49	6.23	0.35
experiemnt49	9.67	3.63	0.16
experiemnt50	6.05	5.00	0.34
experiemnt51	5.19	5.02	0.34
experiemnt52	3.64	6.68	0.25
experiemnt53	3.88	6.53	0.33
experiemnt54	4.53	4.71	0.22
experiemnt55	4.53	5.79	0.43
experiemnt56	4.73	3.60	0.15
experiemnt57	4.83	5.62	0.28
experiemnt58	5.92	6.56	0.25
experiemnt59	4.50	4.58	0.22

experiemnt60	4.77	4.22	0.28
experiemnt61	3.54	3.93	0.35
experiemnt62	4.25	4.23	0.20
experiemnt63	8.16	4.56	0.20
experiemnt64	4.27	4.28	0.41
experiemnt65	5.42	4.36	0.20
experiemnt66	7.43	4.35	0.23
experiemnt67	5.36	3.71	0.38
experiemnt68	5.27	5.15	0.28
experiemnt69	5.96	5.25	0.31
experiemnt70	4.10	3.99	0.38
experiemnt71	6.14	4.54	0.49
experiemnt72	5.65	4.99	0.52
experiemnt73	5.69	5.26	0.41
experiemnt74	5.53	4.26	0.34
experiemnt75	4.38	5.28	0.28
experiemnt76	5.88	6.19	0.37
experiemnt77	5.97	6.41	0.53
experiemnt78	6.72	3.81	0.28
experiemnt79	4.94	3.24	0.27
experiemnt80	5.53	3.75	0.32
experiemnt81	2.75	3.39	0.17
experiemnt82	4.51	3.46	0.31
experiemnt83	7.54	3.46	0.22
experiemnt84	6.65	4.68	0.47
experiemnt85	3.63	4.16	0.26
experiemnt86	6.87	4.36	0.30
experiemnt87	5.65	4.30	0.32
experiemnt88	7.23	4.99	0.33
experiemnt89	6.45	4.22	0.20
experiemnt90	5.05	3.22	0.29

experimnt91	4.42	4.66	0.33
-------------	------	------	------