TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Diploma Thesis

Development of Robot Tutor for Kids

Efstathia-Maria Chatziathanasiou

Thesis Committee
Professor Michail G. Lagoudakis (School of ECE, TUC)
Professor Aikaterini Mania (School of ECE, TUC)
Assistant Professor Nikolaos I. Spanoudakis (Dept of EE, HMU)

Chania, October 2025

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

Ανάπτυξη Ρομποτικού Δασκάλου για Παιδιά

Ευσταθία-Μαρία Χατζηαθανασίου

Εξεταστική Επιτροπή
Καθηγητής Μιχαήλ Γ. Λαγουδάκης (Σχολή ΗΜΜΥ, ΠολΚρ)
Καθηγήτρια Αικατερίνη Μανιά (Σχολή ΗΜΜΥ, ΠολΚρ)
Επίκουρος Καθηγητής Νικόλαος Σπανουδάκης (Τμήμα ΗΜ, ΕΛΜΕΠΑ)

Χανιά, Οκτώβριος 2025

Abstract

In our days, artificial intelligence has significantly contributed to the advancement of technology. It has introduced new scientific fields, one of the most important being machine learning, which focuses on the study and development of algorithms capable of learning from data and making predictions based on it. Machine learning has played a major role in improving people's daily lives, directly influencing their social life and, more specifically, their education. Motivated by the above, we implemented this diploma thesis, which focuses on developing a simple robot tutor for children, using the humanoid robot NAO. In the proposed setup, the robot interacts through two simple games with a child writing on a board. In the first game, the robot helps the child learn the correct spelling of words that describe everyday objects. In the second game, the robot helps the child correctly evaluate simple math expressions, which include addition, subtraction, multiplication and division. During the execution of this work, we tackled the problem of visual object recognition and the problem of visual recognition of handwritten text. With the help of the OpenCV library, we successfully retrieved real-time images from the robot camera, and by using the YOLOv3 model, we completed the object recognition process. Next, we faced the challenge of recognizing handwritten text, where we applied appropriate image processing algorithms from OpenCV to improve the image quality. Finally, using properly trained neural networks, we successfully resolved the text recognition task. Our robot tutor was evaluated by a small group of pupils just finishing the elementary school. This project was positively evaluated, as the NAO responded as expected, making most of the time the correct decisions based on each child's input. The results demonstrate the effective interaction between NAO and the children, indicating the significant impact of integrating robots into human education.

Περίληψη

Στις μέρες μας, η τεχνητή νοημοσύνη έχει συμβάλλει σημαντικά στην εξέλιξη της τεχνολογίας. Έχει εντάξει νέους επιστημονικούς κλάδους, με ιδιαίτερα σημαντική την προσθήκη της μηχανικής μάθησης, η οποία διερευνά τη μελέτη και την κατασκευή αλγορίθμων που μπορούν να μαθαίνουν από τα δεδομένα και να κάνουν προβλέψεις σχετικά με αυτά. Η μηχανική μάθηση έχει συντελέσει στην βελτίωση της καθημερινότητας του ανθρώπου, επηρεάζοντας άμεσα τον κοινωνικό τομέα της ζωής του και πιο συγκεκριμένα την εκπαίδευσή του. Με αφορμή τα παραπάνω, υλοποιήσαμε τη παρούσα διπλωματική εργασία, η οποία έχει ως θέμα την ανάπτυξη ενός ρομπότ δασκάλου για παιδιά, χρησιμοποιώντας το ανθρωποειδές ρομπότ NAO. Στην προτεινόμενη διάταξη, το ρομπότ αλληλεπιδρά μέσω δύο απλών παιχνιδιών με κάποιο παιδί που γράφει σε έναν πίνακα. Στο πρώτο παιχνίδι, το ρομπότ βοηθά το παιδί να μάθει τη σωστή ορθογραφία λέξεων που περιγράφουν καθημερινά αντικείμενα. Στο δεύτερο παιχνίδι, το ρομπότ βοηθά το παιδί να υπολογίζει σωστά απλές μαθηματικές εκφράσεις, που περιλαμβάνουν πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση. Κατά την εκπόνηση της εργασίας, αντιμετωπίσαμε το πρόβλημα της οπτικής αναγνώρισης αντικειμένων και το πρόβλημα οπτικής αναγνώρισης χειρόγραφου κειμένου. Με τη βοήθεια της βιβλιοθήκης OpenCV επιτύχαμε την λήψη εικόνας από την κάμερα του ρομπότ και με τη χρήση του μοντέλου YOLOv3 ολοκληρώσαμε την αναγνώριση των αντικειμένων. Στη συνέχεια, αντιμετωπίσαμε το πρόβλημα της οπτικής αναγνώρισης χειρόγραφου κειμένου, όπου εφαρμόσαμε κατάλληλους αλγορίθμους επεξεργασίας εικόνας της OpenCV για τη βελτίωση της ποιότητας των εικόνων. Τέλος, χρησιμοποιώντας κατάλληλα εκπαιδευμένα νευρωνικά δίκτυα επιλύσαμε το πρόβλημα, αναγνωρίζοντας πλέον με επιτυχία το χειρόγραφο κείμενο. Ο ρομποτικός μας δάσκαλος αξιολογήθηκε από μια μικρή ομάδα μαθητών που μόλις τελείωσαν το δημοτικό σχολείο. Η εργασία αυτή αξιολογήθηκε θετικά, καθώς το NAO ανταποκρινόταν όπως αναμενόταν, λαμβάνοντας κατά κανόνα τις σωστές αποφάσεις, ανάλογα με την απόκριση του κάθε παιδιού. Τα αποτελέσματα αποδεικνύουν την επιτυχή αλληλεπίδραση του NAO με τα παιδιά, υποδεικνύοντας τον σημαντικό αντίκτυπο της ενσωμάτωσης των ρομπότ στην εκπαίδευση του ανθρώπου.

Acknowledgments

With the completion of this thesis, I would like to express my deepest gratitude to the people who supported me throughout this journey. First and foremost, I sincerely thank my supervisor, Prof. Michail G. Lagoudakis, for his guidance, patience, and encouragement throughout the development of this thesis. I am also thankful to Prof. Aikaterini Mania and Assistant Prof. Nikolaos I. Spanoudakis for their participation in the examination committee. I would also like to thank our School's graduate Dimitris Kavroulakis, whose idea inspired this thesis and who guided me on how to implement it successfully. Finally, I am especially grateful to my parents and my friends for their constant support, understanding and help during this journey.

Contents

List of Figures

Chapter 1

Introduction

In today's world, technology has advanced significantly, playing an important role in improving people's lives. One of the most significant advancements is Artificial Intelligence (AI), which is used both in software development and in the creation of robots and machines, with autonomy being their key feature. The integration of this technology into everyday life can become a powerful tool for both problem-solving and enhancing the educational process.

Alongside artificial intelligence, other fields have also emerged, such as machine learning, which is now considered one of the most important and rapidly growing areas of modern science. It provides valuable knowledge and tools — with neural networks being a prime example, offering the autonomy we seek in intelligent systems.

Recognizing the importance of incorporating AI into education and taking into account the rapid progress in machine learning the idea to develop a robot teacher using the NAO humanoid robot was born. More specifically, NAO will interact with children by playing two different educational games. The first game focuses on visual object recognition, where the robot will identify an object chosen by the child and check the spelling of the corresponding word written by the child on a board. The second game involves handwritten equation recognition, where children write a math equation, and the robot provides the correct answer. In the following chapters, we will present in detail the implementation of this project.

1.1 Thesis Contribution

The project faced the challenges of on visual object recognition and visual handwritten text recognition using the humanoid robot NAO. To succesfully train our

system to recognize handwritten text, we used the help of OpenCV's libraries to capture and process the image taken by the robot and Convolutional Neural Networks (CNN) to recognize the written words, digits and operators. For the visual object recognition, we used the model YOLOv3 for detecting and identifying the object. During model training, we observed high accuracy, and in practice, the robot behaved as expected. It wasn't only recognizing correctly, but it was also responding and interacting with the children in accordance with their actions. These results give us great optimism for the future role of technology in education.

1.2 Thesis Overview

The thesis is divided in six chapters, each covering a specific part of the project:

In Chapter 2 we have the theoretical background of the thesis, where we explain in depth all the tools and methods we used throughout the project.

In Chapter 3 we have the problem statement, where we describe more analytically our project and some related work done in the past.

In Chapter 4 we have our approach to the problem, outlining the challenges we faced and the solutions we applied.

In Chapter 5 we have the results of our project, where with pictures and graphs we analyze the outcomes of our work.

Finally, in Chapter 6 we have the discussion, where we summarize our conclusions and propose some future directions that can be taken based on our project.

Chapter 2

Background

In this chapter, we present the basic knowledge related to topics that will be discussed later in this thesis. The goal is to make it easier for the reader to follow the content and understand the methods and tools used to implement the project.

2.1 Humanoid Robot NAO

Figure 2.1: Humanoid Robot NAO [1]

2.1.1 Historical Background of NAO

NAO (Figure 2.1) is a humanoid robot developed by Aldebaran Robotics (then SoftBank Robotics [2], now Maxvision Technology [3]) starting in 2004 in Paris. It is an autonomous and programmable robot that was first used as the official platform of the RoboCup Standard Platform League [4], replacing Sony's Aibo robot. Later, NAO entered the academic field for reasearch and education. Between 2005

and 2007 six prototypes were built, and after several versions in 2018 the NAO 6 was released with upgraded computing power and sensors.

2.1.2 Description of NAO

The version of NAO we used in this thesis is NAO EVOLUTION (V5). It has height 57.40 centimeters with depth of 31.1 centimeters and width of 27.5 centimeters. It weights 5.3 kilograms and it has battery autonomy for 90 minutes of active use. This robot can speak in 20 different languages, walks, dances and recognizes speech, objects and faces. NAO V5 is equipped with 25 degrees of freedom, powered by high-precision electric motors (actuators) that enable smooth and natural movement of the head, arms, legs, and torso as shown in Figure 2.3. Its sensor suite includes tactile sensors on the head, hands, and feet as shown in Figure 2.2, as well as an inertial measurement unit (IMU) for balance and orientation. For vision, NAO has two high-definition cameras—one on the forehead and one in the mouth area—providing depth perception and object recognition capabilities. The robot's speech system supports text-to-speech in multiple languages, while its four directional microphones allow it to detect sound sources, recognize voice commands, and interact naturally with users. Together, these features enable NAO to perceive its environment, move accurately, and communicate effectively with people.

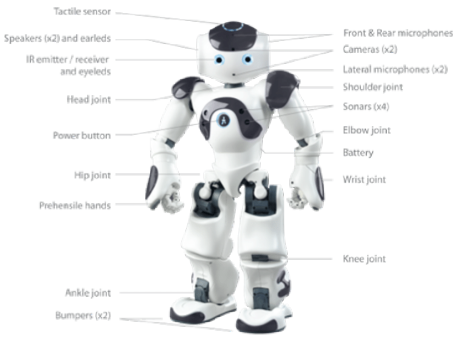


Figure 2.2: Sensors of NAO [5]

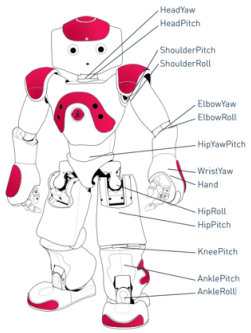


Figure 2.3: Actuators of NAO [6].

2.1.3 Technical Characteristics

The NAO v5 is powered by an Intel Atom Z530 at 1.6 GHz CPU with 1 GB RAM, 2 GB Flash memory and 8 GB Micro SDHC storage. It is compatible with LINUX operating system and the connectivity includes Ethernet and Wi-Fi. It also has an IEEE 802.11g network card, that can connect wirellessly or wired with RJ45 Ethernet.

2.1.4 The Operating System NAOqi

NAOqi is the main operating system that runs on NAO robots. It has been developed by Aldebaran Robotics [7] and acts as the core framework. It manages all handware components, such as motors, sensors, cameras and microphones, while providing high-level APIs for motion control, vision processing, speech synthesis, and sensor data handling. It also enables the user to program NAO in various languages, including Python, Java, C++ and JavaScript. Overall, NAOqi simplifies development and allows seamless interaction between hardware and software.

2.2 OpenCV

OpenCV (Open Source Computer Vision Library) [8] [9] was initially developed by Intel and is a free, cross-platform, computer vision library for real-time image processing. It has become a standard tool for all problems related to Computer Vision. The OpenCV library contains over 2500 algorithms, extensive documentation, source code, and sample code for real-time computer vision. These algorithms can be used for tasks, such as face detection and recognition, object identification, human action classification in videos, camera motion tracking, and moving object tracking, among others. OpenCV provides interfaces for C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. It is optimized for real-time applications and can take advantage of hardware acceleration technologies, such as MMX and SSE, when available.

2.2.1 OpenCV's Functions and Algorithms

Cvtcolor Function

The `cv2.cvtcolor()` [10] method in OpenCV is a function used for converting the color space of an image. It takes an input image in one color space and transforms it into another color space, such as converting from RGB to Grayscale, RGB (red, green, bule) to HSV (hue, saturation, value), or vice versa. This method is crucial in various image processing tasks and computer vision applications, allowing users to manipulate and extract useful information from images in different color representations. The function uses as parameters the `src`, `the input image` (NumPy array) and the `code`, that specifies the type of color space conversion to be performed. This is an integer value representing the color space conversion, e.g., `cv2.COLOR_BGR2GRAY`, `cv2.COLOR_BGR2HSV`, etc. Then, we have the parameter `dst`, which is optional and it stores the result of the color space conversion and lastly we have the optional parameter `dstCn`, which defines the number

of channels in the destination image. Below is an example (Figure 2.4, Figure 2.5) of an image with the `cvtcolor()` method.

Figure 2.4: Original Image [11]

Figure 2.5: Image after `cvtcolor()` [11].

Canny Edge Detection Function

Canny edge detection [12] is a widely used edge detection algorithm developed by John F. Canny. It is a multi-stage process. First, it performs noise reduction by applying a 5×5 Gaussian filter. Then, it calculates the intensity gradient of the image by filtering the smoothed image with a Sobel kernel in both the horizontal (G_x) and vertical (G_y) directions to obtain the first derivatives. From these two results, we can compute the edge gradient and direction for each pixel as follows:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}, \quad Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Then, non-maximum suppression is applied to thin the detected edges by retaining only the local maxima in the gradient direction, producing accurate and crisp edge representations. Finally, there is the Hysteresis Thresholding, that decides which are all edges are really edges and which are not. For this, two threshold values are needed, `minVal` and `maxVal`. Any edges with intensity gradient more than `maxVal` are sure to be edges and those below `minVal` are sure to be non-edges, so discarded. Those that lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Below is an example (Figure 2.6, Figure 2.5) of an original image and the canny edge detector method apllied to the image.

Hough Line Transform Algorithm

The algorithm of Hough Line Transform [14] is a transform used to detect straight lines and to apply the transform, first an edge detection pre-processing is desirable.

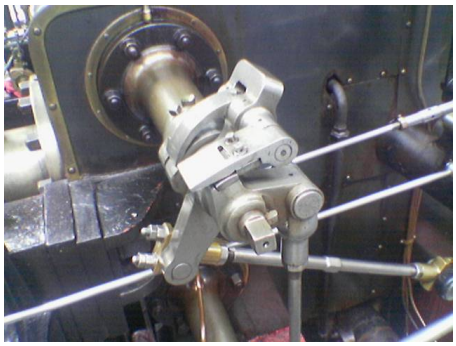


Figure 2.6: Original Image [13]

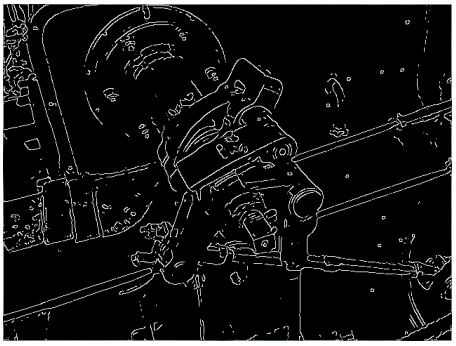


Figure 2.7: Image after Canny Edge Detection Function [13].

For Hough Transforms, we express lines in the Polar system. A line can be represented as: $r = x\cos\theta + y\sin\theta$, where r is the vertical distance from the origin to the line, and θ is the angle of the normal with respect to the x-axis. For each point (x_0, y_0), we can define the family of lines that goes through that point as:

$$r_\theta = x_0\cos\theta + y_0\sin\theta$$

Each pair (r_θ, θ) represents each line that passes by (x_0, y_0). In general, a line can be detected by finding the number of intersections between curves. The more curves intersecting means that the line represented by that intersection have more points. A threshold can be defined for the minimum number of intersections required to detect a line. The algorithm keeps track of the intersection between curves of every point in the image. If the number of intersections is above some threshold, then it declares it as a line with the parameters $(\theta, r\theta)$ of the intersection point. For instance, we see in Figure 2.8 the plot for $x_0 = 8$ and $y_0 = 6$ and in Figure 2.9 we observe two additional points ($x_1 = 4$ and $y_1 = 9$) and ($x_2 = 12$ and $y_2 = 3$) in the same plot as shown in Figure 2.8

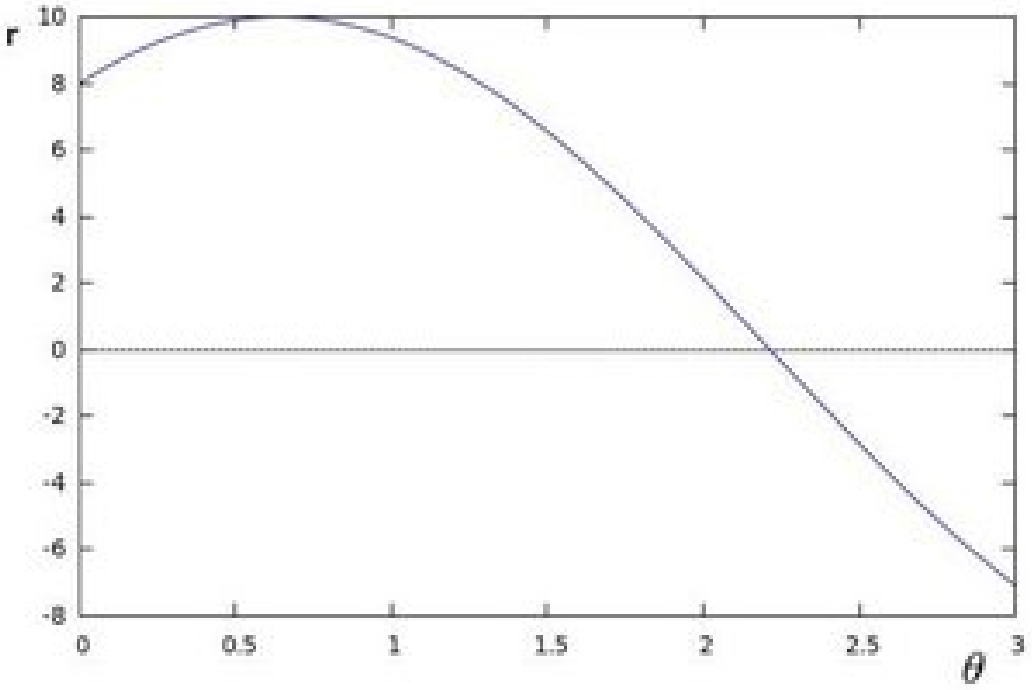


Figure 2.8: Hough Line Transform Example [14].

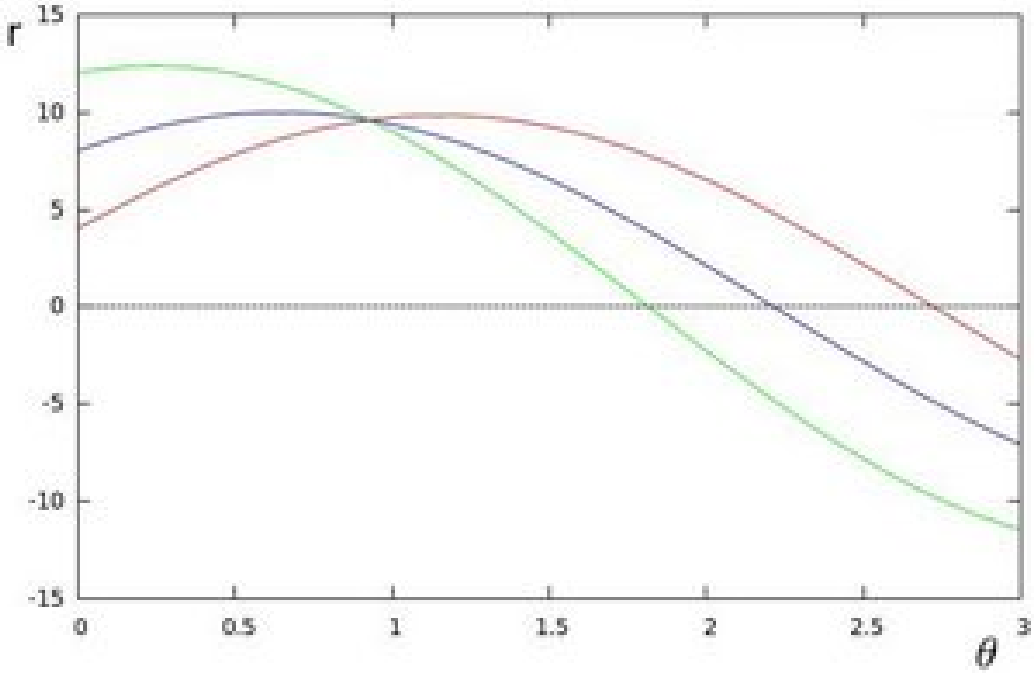


Figure 2.9: Hough Line Transform Example [14].

The three plots intersect in one single point $(0.925, 9.6)$. These coordinates are the parameters (θ, r) of the line on which (x_0, y_0), (x_1, y_1) and (x_2, y_2) lie.

Line Function

The `cv2.line()` method [15] in OpenCV is used to draw a line on an image. The parameters of the function are the `image`, which is the target image, the `start_point` that, specifies the coordinates where the line begins, and the `end_point`, that specifies where it ends, both given as tuples in the form (x, y). The color parameter defines the line color as an RGB tuple (e.g., (255, 0, 0) for blue), while thickness determines the line width in pixels. The function returns the image with the drawn line.

PyrMeanShiftFiltering Function

The function [16] implements the filtering stage of meanshift segmentation, that is, the output of the function is the filtered "posterized" image with color gradients and fine-grain texture flattened. At every pixel (X, Y) of the input image the function executes meanshift iterations, that is, the pixel (X, Y) neighborhood in the joint space-color hyperspace is considered:

$$(x, y) : X - sp \leq x \leq X + sp, \quad Y - sp \leq y \leq Y + sp, \quad \|(R, G, B) - (r, g, b)\| \leq sr$$

where (R, G, B) and (r, g, b) are the vectors of color components at (X, Y) and (x, y), respectively. The average spatial value (X', Y') and average color vector (R', G', B') are found and they act as the neighbourhood center on the next iteration:

$$(X, Y) \, (X', Y'), \; (R, G, B) \, (R', G', B')$$

The color components of the initial pixel, or the pixel from which the iterations began, are set to the final value (average color at the last iteration) once the iterations are complete:

$$I(X, Y) \leftarrow (R^*, G^*, B^*)$$

When `maxLevel` > 0, where the parameter `maxLevel` represents the maximum level of the pyramid for the segmentation, the Gaussian pyramid of `maxLevel`+1 levels is built, and the above procedure is run on the smallest layer first. The outcomes are then transferred to the higher layer, and iterations are repeated only on pixels where the colors of the layer deviate from the pyramid's lower-resolution layer by more than sr. That makes boundaries of color regions sharper. The results will be actually different from the ones obtained by running the meanshift procedure on the whole original image (i.e. when `maxLevel` $== 0$).

Threshold Function

The function `cv.threshold` [17] is used to apply the thresholding. If the pixel value is smaller than or equal to the threshold, it is set to 0, otherwise it is set to a maximum value. The first argument is the `source image`, which should be a grayscale image. The second argument is the `threshold value` which is used to classify the pixel values. The third argument is the `maximum value`, which is assigned to pixel values exceeding the threshold. OpenCV provides different types of thresholding which is given by the fourth parameter of the function. Basic thresholding as described above is done by using the type `cv.THRESH_BINARY`. All simple thresholding types are:

1. `cv.THRESH_BINARY`
2. `cv.THRESH_BINARY_INV`

3. `cv.THRESH_TRUNC`
4. `cv.THRESH_TOZERO`
5. `cv.THRESH_TOZERO_INV`

Below in Figure 2.10 we observe an example of the thresholding types, that are refered above:

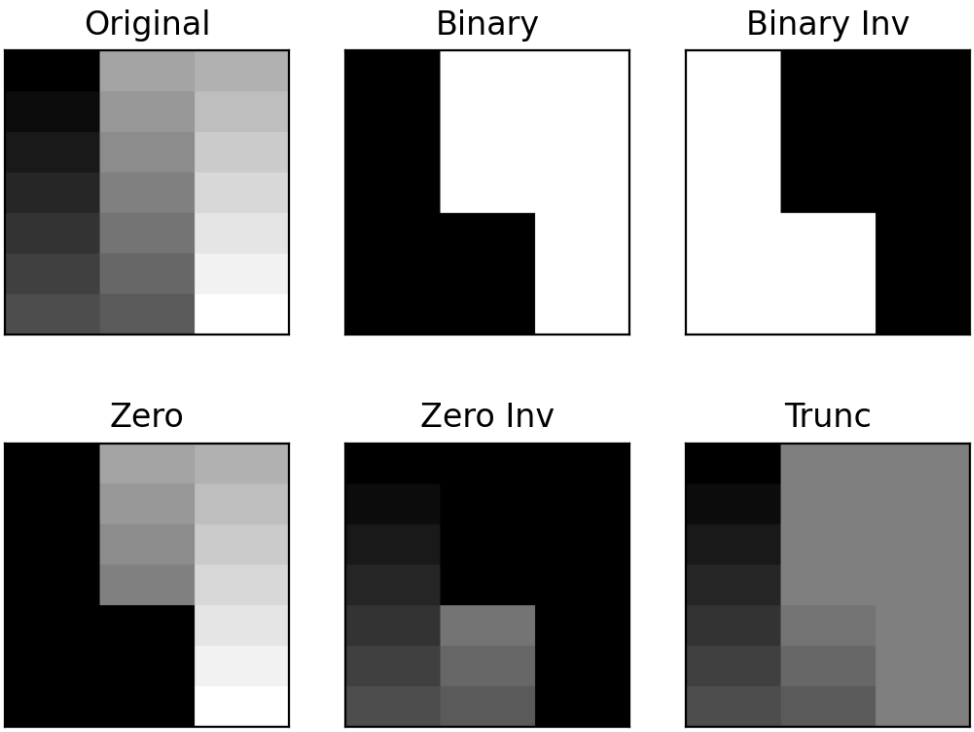


Figure 2.10: Threshold Example [18].

FindContours Function

Contours are edges or outline of objects in a image and is used in image processing to identify shapes, detect objects or measure their size. We use OpenCV's `findContours()` function [19] that works best for binary images.

There are three important arguments of this function:

1. **Source Image:** This is the image from which we want to find the contours.

2. **Contour Retrieval Mode:** This determines how contours are retrieved.

3. **Contour Approximation Method:** This decides how much detail to keep when storing the contours.

The function gives us three outputs:

1. **Image:** The image with contours found in it.

2. **Contours:** A list of contours. Each contour is made up of the (x, y) coordinates that outline a shape in the image.

3. **Hierarchy:** This gives extra information about the contours like which ones are inside others.

2.3 Neural Networks

Neural networks [25] are machine learning programs, or models, that use methods that resemble how biological neurons collaborate to recognize occurrences, evaluate possibilities, and reach conclusions. This allows them to make judgments in a way that is comparable to that of the human brain.

An input layer, one or more hidden layers, an output layer, and layers of nodes or artificial neurons make up every neural network, as we see in Figure 2.11. Each node is linked to other nodes and has a weight and threshold of its own. If any node's output surpasses the designated threshold value, it becomes active and transmits data to the network's next layer. Otherwise, no data is sent to the following layer of the network.

Input layer is the layer where the input is fed to the network. Output layer uses the output of a particular Hidden layer, where most of the computation takes place. This is associated with some numeric value called bias, which is then added to the input sum, which is passed to a threshold function called activation function. Activation function determines whether a particular neuron should be activated or not. Activated neuron transmits data to the next layer. A Channel is a connection between these neurons.

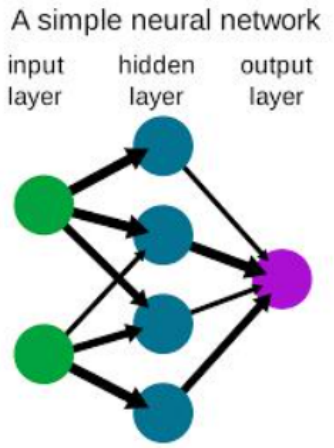


Figure 2.11: Structure of a neural network [26].

Training data are necessary for neural networks to learn and gradually increase their accuracy. These are effective techniques in computer science and artificial intelligence that enable us to quickly classify and cluster data, once they are adjusted for accuracy. When compared to manual identification by human experts, tasks in image or audio recognition can take hours instead of minutes. The search algorithm used by Google is among the most well-known instances of a neural network.

2.4 Deep Neural Network

Deep learning [27] is a kind of machine learning that mimics the intricate decision-making process of the human brain by using multilayered neural networks, or deep neural networks. The majority of artificial intelligence (AI) apps in our daily lives are powered by deep learning in one way or another. Digital assistants, voice-activated TV remote controls, credit card fraud detection, self-driving cars, and generative AI are just a few of the commonplace goods and services made possible by this.

The structure of the underlying neural network architecture is the primary difference between machine learning and deep learning. Simple neural networks with one or two computational layers are used in “nondeep”, conventional machine learning models. Three or more layers are used in deep learning models, however hundreds or thousands of layers are usually used for training. Deep learning models can use unsupervised learning, while supervised learning models need organized, labeled input data to produce reliable results. Unsupervised learning enables deep learning models to extract the properties, attributes, and connections

required to provide precise results from unstructured, raw data. For greater accuracy, these models may even assess and improve their outputs.

Deep neural networks consist of multiple layers of interconnected nodes, each building on the previous layer to refine and optimize the prediction or categorization (Figure 2.12). This progression of computations through the network is called forward propagation. Visible layers are a deep neural network's input and output layers. The deep learning model processes the data in the input layer before making the final classification or prediction in the output layer.

Another method, known as backpropagation, calculates prediction errors using techniques like gradient descent and then trains the model by going backwards through the layers and adjusting the function's weights and biases.

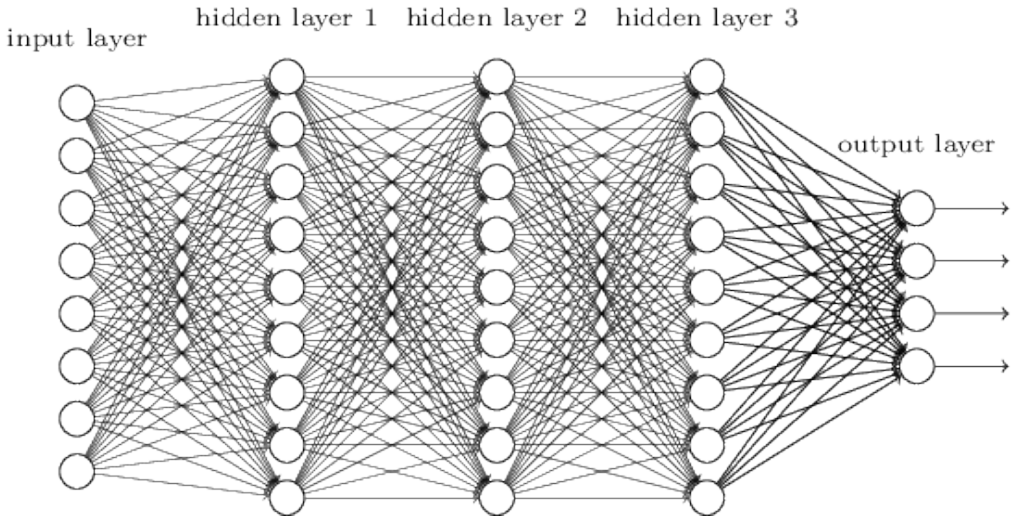


Figure 2.12: Structure of a deep neural networks [28].

2.5 Convolutional Neural Networks

Computer vision and image classification applications are the main uses for convolutional neural networks, also known as CNNs or ConvNets [29] [31] [30]. Tasks, like object detection, image recognition, pattern recognition, and face recognition, are made possible by their ability to identify characteristics and patterns in pictures and videos. These networks use matrix multiplication and other linear and non-linear algebraic concepts to find patterns in an image.

The node layers that make up CNNs, a particular kind of neural network, include an input layer, one or more hidden layers, and an output layer. There are three primary kinds of layers in convolutional neural networks, and they are:

1. Convolutional layer
2. Pooling layer
3. Fully-connected (FC) layer

A convolutional network's initial layers are typically called the convolutional layers. The fully-connected layer is typically the last layer. Between the initial and the last layers may exist other convolutional or pooling layers (Figure 2.13). The CNN becomes more complicated with each layer, recognizing larger areas of the image. Earlier layers focus on basic elements, like borders and colors. As the image input passes through its layers, the CNN starts to recognize the main elements or shapes of the object, and eventually recognizes the target item.

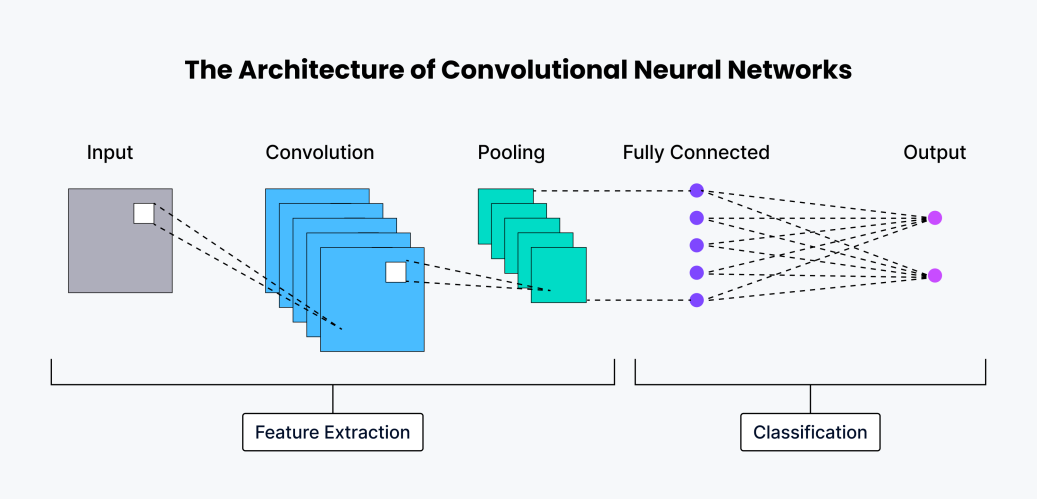


Figure 2.13: CNN Architecture [33].

2.5.1 Convolutional Layer

The majority of computation takes place in a convolutional layer, which is the fundamental component of a CNN. A feature map, a filter, and input data are the basic things it needs. Assume that a color image, consisting of a 3D pixel matrix, will be the input. This indicates that the input will have three dimensions: height, width, and depth. Additionally, we have a feature detector, sometimes referred to as a kernel or filter, that will traverse the image's receptive fields to determine whether the feature is present. We call this process a convolution.

Part of the image is represented by a two-dimensional (2-D) array of weights that make up the feature detector. The size of the receptive field is determined by the filter size, which is usually a 3×3 matrix, however they can vary in size. After applying the filter to a portion of the image, the dot product between the input pixels and the filter is computed. After that, an output array receives this dot product. The filter then moves by a stride, and so on, until the kernel has swept over the whole image. Activation maps, convolved features, or feature maps are the ultimate results of the series of dot products from the input and the filter.

It should be noted that the feature detector's weights, sometimes referred to as parameter sharing, stay constant as it analyzes the image. During training, some parameters, including the weight values, change via the backpropagation process and gradient descent. However, before the neural network training process starts, three hyperparameters that impact the output volume size must be established. These include:

1. The output's depth is affected by the number of filters. For instance, a depth of three would be produced by three different feature maps produced by three different filters.

2. The kernel's stride is the amount of pixels it travels over the matrix of input. A larger stride results in a smaller output, despite that stride values of two or higher are uncommon.

3. Zero-padding is usually used when the filters do not fit the input image. This creates a larger or equal-sized output by setting all elements outside of the input matrix to zero. There are three types of padding.

Valid padding: This is also known as no padding. In this case, the last convolution is dropped, if dimensions do not align.
Same padding: This padding ensures that the output layer has the same size as the input layer.
Full padding: This type of padding increases the size of the output by adding

zeros to the border of the input.

The model becomes nonlinear after a CNN applies a Rectified Linear Unit (ReLU) adjustment to the feature map following each convolution operation.

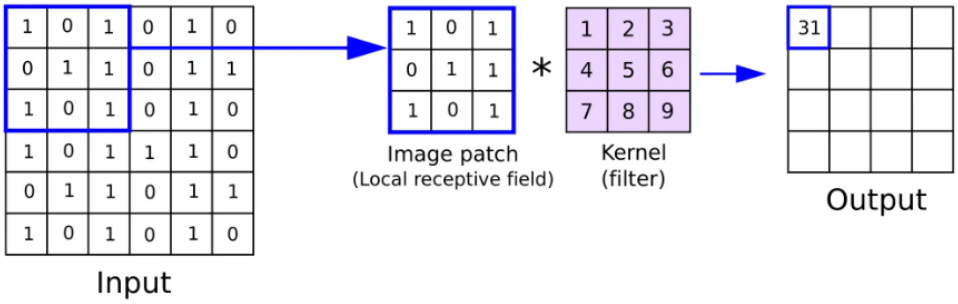


Figure 2.14: An example of a convolutional layer [32].

The first convolution layer may be followed by another convolution layer. Because the later layers may view the pixels inside the receptive fields of the earlier layers, the CNN's structure may become hierarchical as a result. Let's take the example of attempting to identify whether a bicycle is included in an image (Figure 2.15. The bicycle can be viewed as the culmination of its components. It is made up of a frame, wheels, pedals, handlebars, and other parts. In the neural net, each component of the bicycle represents a lower-level pattern, while the parts taken together represent a higher-level pattern, forming a feature hierarchy within CNN. The neural network will then understand and extract relevant patterns once the convolutional layer converts the image into numerical information.

Figure 2.15: Hierarchy of convolutional layer [29].

2.5.2 Pooling Layer

Pooling layers [29], sometimes referred to as downsampling, reduce the number of parameters in the input by performing dimensionality reduction. The pooling operation applies a filter to the entire input, unlike convolution, this kernel does not have weights. The kernel uses an aggregation function to fill the output array with the values in the receptive field. There are two main types of pooling:

1.**Max pooling:** The filter chooses the pixel with the highest value to send to the output array, as it passes over the input. In addition, this method usually gets used more frequently than normal pooling.
2.**Average pooling:** The filter determines the average value within the receptive field to send to the output array, as it moves across the input.

Although the pooling layer loses a lot of information, they reduce the chance of overfitting, increase efficiency, and simplify things. In Figure 2.16, we see an example of Max Pooling and one of Average Pooling.

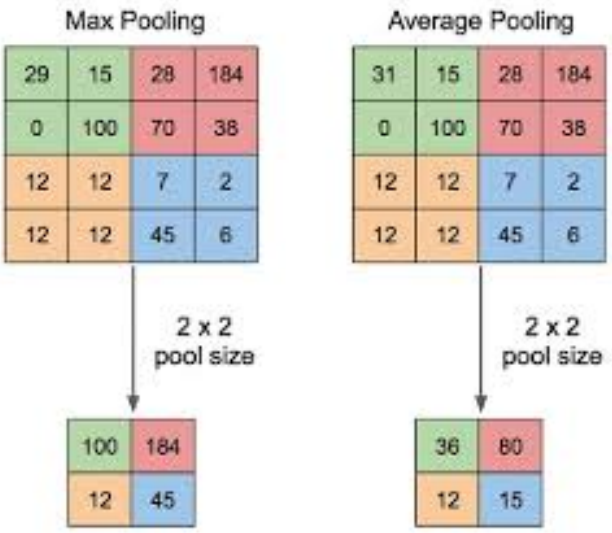


Figure 2.16: Examples of Pooling Layers [34].

2.5.3 Fully-Connected Layer

In the fully-connected (FC) layer [29], every node in the output layer is directly connected to a node in the previous layer. Every neuron in these layers is linked to every activation in the layer before it, forming a network that is extremely interconnected. FC layers typically utilize a softmax activation function to categorize inputs appropriately, generating a probability ranging from 0 to 1.

2.5.4 Neural Network Activation Function

The Activation Function [35] is a mathematical function applied to the output of a neuron. By adding non-linearity to the model, it enables the network to recognize and depict complex patterns in the data. No matter how many layers a neural network has, without this non-linearity property, it will act as a linear regression model. Below, we are going to analyze two specific activation functions in Deep Learning:

ReLU (Rectified Linear Unit) Function

The ReLU (Rectified Linear Unit) activation function is defined as follows:

$$A(x) = \max(0, x)$$

This means that, if the input x is positive, ReLU returns x; otherwise, if the input is negative, it returns 0, as we observe also in Figure 2.17

Value Range: $[0, \infty)$, indicating that the function outputs are only non-negative values.
Nature: ReLU is a non-linear activation function, which allows neural networks to learn complex patterns and improves the efficiency of backpropagation.
Advantage over Other Activations: ReLU is computationally less expensive compared to `tanh` and `sigmoid`, because it involves simpler mathematical operations. Additionally, since only a few neurons are activated at a time, the network remains sparse, making computation faster and more efficient.

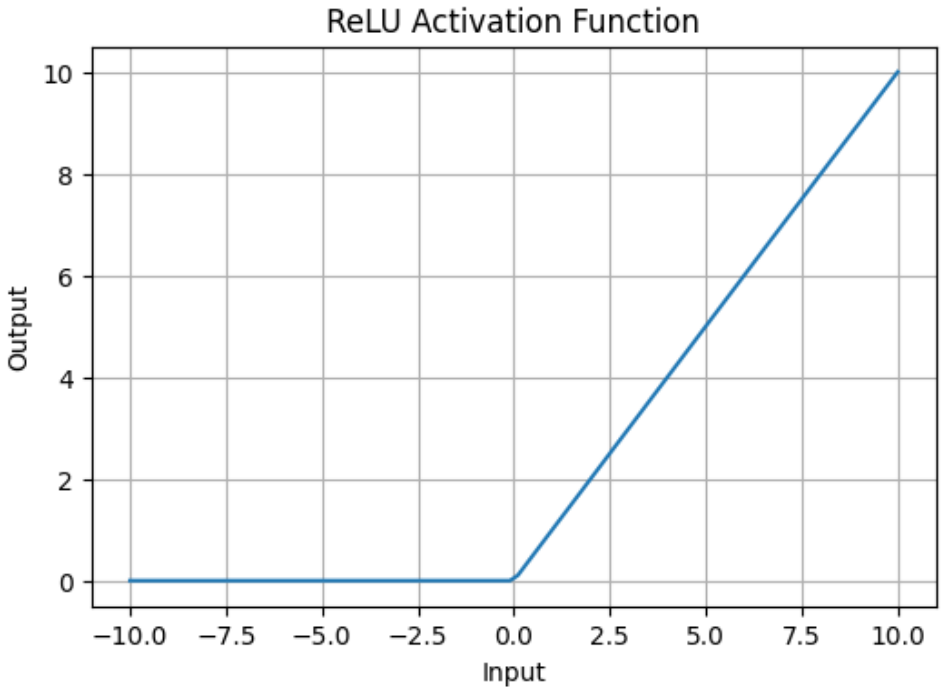


Figure 2.17: Relu Activation Function [35].

Softmax Function

Softmax function [36] is designed to handle multi-class classification problems. It transforms raw output scores from a neural network into probabilities. These probabilities are distributed across different classes such that their sum equals 1. Softmax is useful for classification tasks since it essentially assists in converting output values into a format that can be understood as probabilities.
The Softmax function is defined as:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where:

- z_i is the logit (the output of the previous layer in the network) for the i^{th} class,
- K is the total number of classes,
- e^{z_i} represents the exponential of the logit,
- $\sum_{j=1}^{K} e^{z_j}$ is the sum of exponentials across all classes.

Below in Figure 2.18 we observe the Softmax Function Curve.

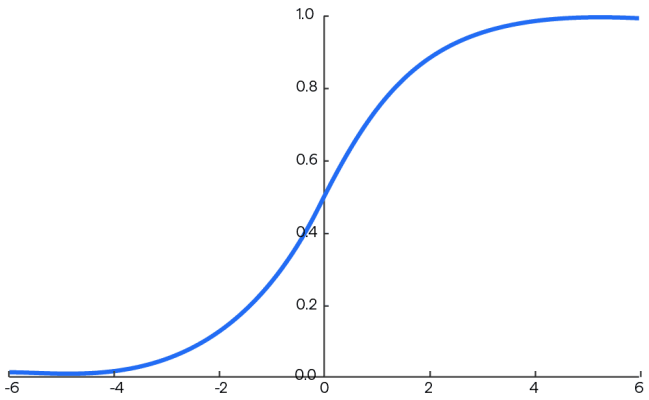


Figure 2.18: Softmax Activation Function [37].

2.6 Tensorflow

TensorFlow [20] [21] is an open-source framework for machine learning with data flow graphs that is widely used by academics, software developers, and data scientists. The framework was first created by the Google Brain Team to support research on deep neural networks (DNNs) and machine learning, but it is versatile enough to be used in many other fields as well. The TensorFlow workflow is comprised of three independent steps: data preprocessing, model construction, and prediction training. The framework runs in two different ways and receives data as a multidimensional array called tensors. The main technique involves creating a computational graph that specifies a dataflow for model training. The second, and frequently more natural, approach is eager execution, which evaluates operations instantly and adheres to imperative programming principles. Training often takes place in a data center or on a desktop using the TensorFlow architecture. Tensors are placed on the GPU to speed up the process in both scenarios. After that, trained models can operate on a variety of platforms, including desktop, mobile, and cloud. In Figure 2.19 below we have the Tensorflow workflow.

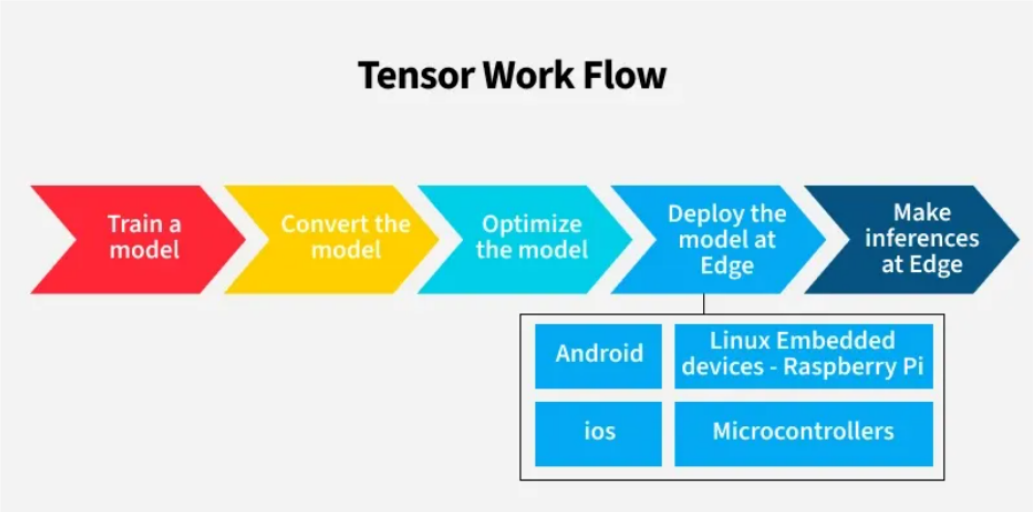


Figure 2.19: Tensorflow workflow [22].

2.7 Keras

Designing and implementing deep learning models in Python requires the use of the Keras Layers API [23]. By joining layers that carry out particular computing tasks, it provides a means of building networks. The Layers API optimizes implementation, while offering crucial capabilities for creating reliable models across a range of data formats, such as text, time series, and images. In Keras, a layer describes data transformation. It takes as input tensors, computes them, and then outputs the results. Because of this abstraction, developers can think of models as a series of precise mathematical processes. Keras allows for the definition of bespoke layers in addition to predefined standard layers. Every layer keeps its own settings, weights, and parameters. In order for the framework to test the dimensions for upcoming calculations, the input shape for multiple layers needs to be provided during setup.

2.8 YOLOv3

YOLOv3 (You Only Look Once) [24] is a real-time object detection algorithm, that can identify particular items in pictures and videos. The machine learning system YOLO quickly recognizes objects in a picture by using features extracted from a deep convolutional neural network.The third version of YOLO is the one with the highest accuracy among the various versions, which were written by Joseph Redmon and Ali Farhadi. The majority of object detectors use a classifier to make detection predictions based on the features that the convolutional layers have learned. On the other hand, YOLO's prediction relies on a unique convolutional layer that employs 1×1 convolutions. This indicates that the prediction map's dimensions match those of the feature map that came before it. The fully connected layer may employ a compact representation of features, while producing detection predictions thanks to this clever usage of 1×1 convolutions, which speeds up the prediction process.

After receiving a picture as input, the YOLOv3 algorithm applies a CNN known as Darknet-53 to identify objects inside the image. Based on the ResNet architecture, Darknet-53 is specifically designed for object detection tasks. It has 53 convolutional layers and performs exceptionally well on a variety of object detection benchmarks (Figure 2.21). Darknet-53's deep structure, which enables it to directly learn complex patterns and representations from unprocessed picture input, is one of its primary characteristics. Improved object detection performance results from the network's ability to catch fine features and sensitivities found in images thanks to its deep architecture. Additionally, Darknet-53 uses remnant

connections, which are similar to ResNet architectures. By enabling information flow throughout the network, these links help to mitigate the vanishing gradient issue and make it possible to train deeper networks more effectively. Darknet-53 provides a strong feature extraction backbone, which is essential to the YOLOv3 algorithm. Accurate object detection in real-time scenarios is made possible by the algorithm's deep architecture and powerful feature extraction capabilities.

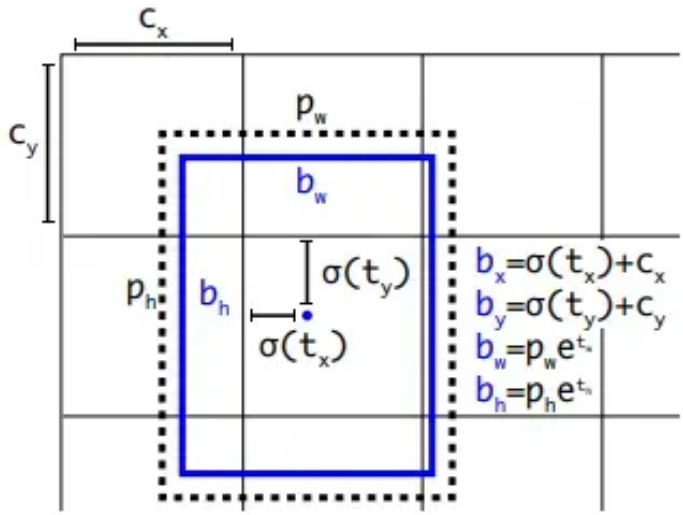


Figure 2.20: Scaled anchor boxes with varied aspect ratios [24].

By using scaled anchor boxes with different aspect ratios, YOLOv3 improves the algorithm's ability to identify objects of different shapes and sizes (Figure 2.20). Additionally, YOLOv3 presents the idea of "Feature Pyramid Networks" (FPN), a CNN architecture intended for multi-scale object detection (Figure 2.22). By building a hierarchical pyramid of feature maps, FPNs enable the model to simultaneously detect objects at various scales. Due to the model's ability to analyze items at different scales, this augmentation greatly improves detection performance for small objects. Lastly, YOLOv3 is more accurate and resilient than its predecessors due to its enhanced ability to handle a larger range of item sizes and aspect ratios.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.21: Darknet-53 architecture [24].

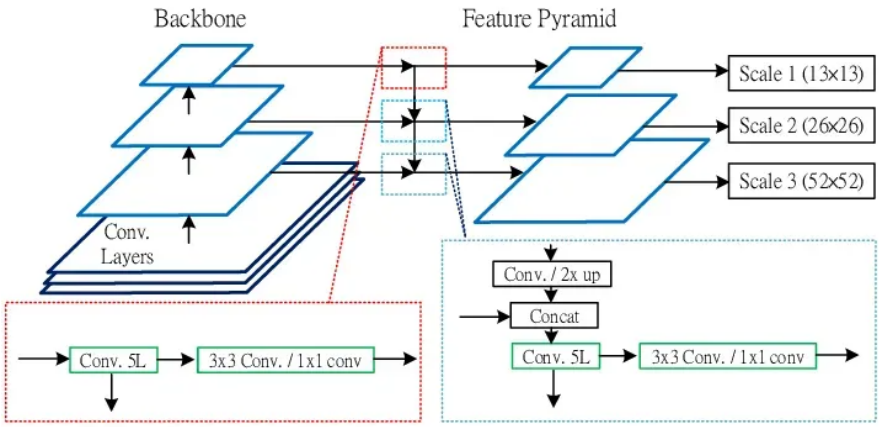


Figure 2.22: Feature Pyramid Network adopted in YOLOv3 [24].

Chapter 3

Problem Statement

3.1 Detailed Problem Statement

In today's world, artificial inteligence has advanced significally and contributed to many aspects of human life - one of them being education. Our thesis focuses on children's learning with the use of the humanoid robot NAO. More spesifically, two educational games we aim to develop, a grammar game and a math game. In the grammar game NAO asks the child to present an object and after identifying it, it says its name. Then it asks the child to write on the board the object's name and it gives three attemps to write the word correctly. In case the child fails after three attemps, the robot provides hints to the child by indicating the positions of the incorrect letters until the child writes the word correctly. In the math game NAO asks the child to write on the board a simple mathematical equation and then provides the correct result. To achieve these educational interactions, we will have to face and solve the problems of visual object recognition and visual recognition of handwritten text using the humanoid robot NAO.

For the visual object recognition subproblem, we have to solve two main challenges. The first is the image capturing problem, where the NAO robot needs to take photos of objects in real-time. The second one is the problem of the image proccessing, where NAO processes the acquired image, detects the object and with the help of the YOLOv3 model identifies it.

For the visual recognition of handwritten text, we have again to face two difficulties. First, we have the image capturing problem like in the visual object recognition, where NAO has to take a picture of handwritten text in real time. Then, we have to face the image processing problem, where NAO has to process the image to detect the text, so we can use it along with the recognition part.

Finally, we need to program a simple behavior on the robot for interacting with the child through verbal instruction and feedback.

3.2 Related Work

Next, we present some completed projects that are similar to ours and are focusing on NAO's role in education.

3.2.1 Students with Autism in Classroom with NAO Robot

Qin Yang, Huan Lu, Dandan Liang, Shengrong Gong and Huanghao Feng published a paper [38] describing a group experiment in a collective classroom using the NAO humanoid robot. Specifically, they used NAO alongside teachers in a special education classroom with autistic children, to do interactive learning activities such as roll call, command listening, picture description, and group interaction, combined with small games and performances to maintain interest. Using video content from multiple perspectives they compared the classroom performance of children with autism in both-assisted and regular classroom settings based on attention, communication, assessment and emotion. The results were encouraging, showing that kids with Autism Spectrum Disorder (ASD) were more focused and socially responsive, when NAO was involved. In Figure 3.1, we see an experiment, where the students meet the robot and get familiar with it.

Figure 3.1: Preparatory lesson for robot-assisted courses [38].

3.2.2 Teaching NAO to Play a Human-Robot Game

Cen Li, Ebosehon Imeokparia, Michael Ketzner and Tsega Tsahai developed a project with the use of a humanoid robot NAO to play the game of "Simon Says" with human players [39]. In the game the robot asks the human player(s) to make a pose and it compares the player's pose to the pre-defined poses to decide if it is correct. To succesfully do this project, they used Choregraphe, OpenPose and OpenCV for the pose detection from NAO's camera and Keras and Convolutional Neural Network to recognize the player's poses. In Figure 3.2, we see the robot asking the players to lie down.

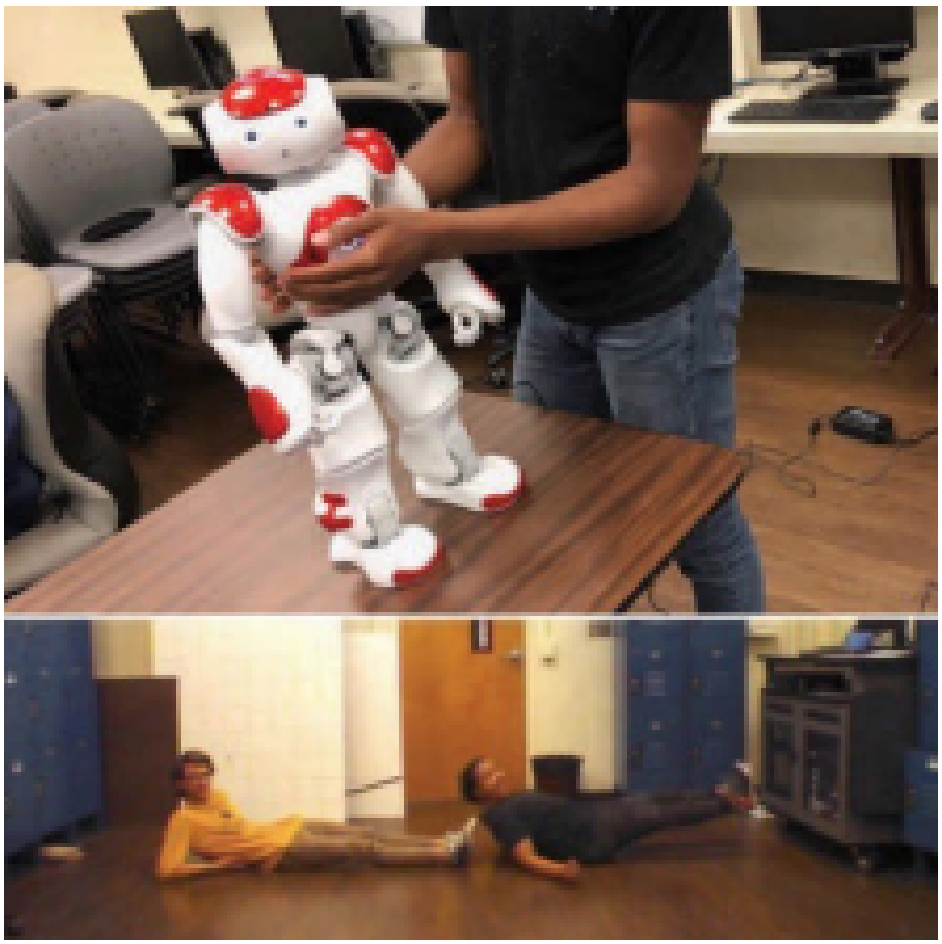


Figure 3.2: "Simon" the robot asks the player to lie down [39].

3.2.3 Impact of Multi-Modal Robots on Learning Engagement

This study by Fun Ka Yan Fung, Kwong Chiu Fung, Tze Leung Rick Lui, Kuen Fung Sin, Lik Hang Lee, Huamin Qu and Shenghui Song [40] introduces an inclusive learning system using two multi-modal robots, Kebbi and Minibo, which provide interactive features such as gestures, animations, songs, and touch to support diverse learning needs. A comparative study was carried out with 73 pupils in a teacher-led control group and 64 students in a five-day robot-led program. While non-dyslexic students reacted well to Minibo's more straightforward elements, the results indicated that dyslexic students gained more from Kebbi's sophisticated interactivity. By striking a balance between novelty, emotional connection, and cognitive load management, the results imply that multi-modal robots can improve motivation and engagement. In Figure 3.3, we see a student was learning with Kebbi in classroom setting.

Figure 3.3: A student was learning with Kebbi in classroom setting [40].

3.2.4 Humanoid Robots as Assistants in Teaching

Pepper, an educational assistant, is the subject of a study by Akshara Pande and Deepti Mishra [41] on incorporating humanoid robots into the classroom. They emphasize the ways in which technology—particularly humanoid robots—can enhance instruction by fusing interaction with a human-like presence. They use a voice transcription technology that shows text on Pepper's screen to aid comprehension because accents and other auditory problems can make speech hard to follow. Both online and offline teaching modalities were compared and evaluated. Results indicate that independent of participant gender, Pepper's speech recognition system performs well in both contexts. According to the study's findings, humanoid robots can improve learning outcomes by increasing flexibility, effectiveness, and student involvement in the classroom. In Figure 3.4, we see an overview of the working of the proposed system with humanoid robot in offline and online settings.

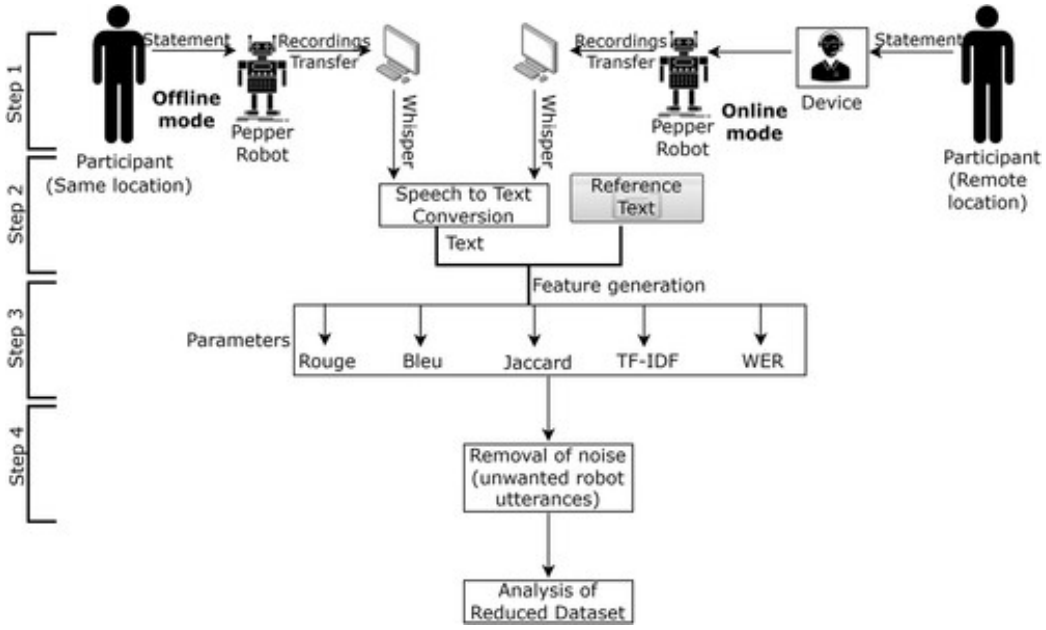


Figure 3.4: Overview of the working of the proposed system with humanoid robot in offline and online settings [41].

Chapter 4

Our Approach

Our project is divided into two small subprojects, as we have developed two educational games for children: one focused on object recognition and spelling correction, and the other on arithmetics. However, for both games, we had to implement solutions to almost the same problems.

For the spelling game we had to implement the visual object recognition using the humanoid robot NAO and for both games we had to achieve the visual recognition of handwritten text. Since both games involve the same underlying problem — handwritten text recognition — from this point on, when referring to "letter recognition", we also include symbols and digits as part of the process.

4.1 Visual Object Recognition

To achieve object recognition we had to address first the subproblem of acquiring the image and then the subproblem of image proccessing.

4.1.1 Image Acquisition for Object Recognition

For NAO to proccess the object and identify it using the YOLOv3 model, it has at first to take a picture of the object so it can run it through the proccess part of our project. At first we connected via pynaoqi and the `ALProxy` method the NAO with our computer and then with the help of `subscribeCamera` we adjusted the parameters we wanted for the camera of the robot (we choose 640×480 analysis, 0 for the top camera and RGB format). Then, using the method `getImageRemote`, which helps the user see what the robot is seeing at real time, we placed the object we wanted infront of the camera and captured the image we wanted.

4.1.2 Image Proccess for Object Recognition

In this part we have to proccess the image using the OpenCV's DNN module and the YOLOv3 model. We prepare the image for YOLOv3 model by converting it into a blob using OpenCV's `blobFromImage()` function. This function resizes, normalizes and formats the image for compatibility with the YOLOv3 network. Then, this blob is loaded and passed through the network with the help of `cv2.dd.readNet()` function (which uses the .cfg and .weights files). Yolo, then using the `getUnconnectedOutLayersNames()` function to identify its output layers, produces a set of detected objects and by choosing the object with the highest confidence score above a threshold, identifies succesfully the object we have given NAO to recognize.

4.2 Visual Recognition of Handwritten Text

This problem had to be addressed for both games. In the game of object recognition and spelling we had to recognize the letters and in the math game we had to recognize the digits and symbols. For the handwritten text recognition from the kids we had first to succesfully obtain the image of the handwritten text then proccess the image and lastly achieve the recognition of the letters (/digits/symbols).

4.2.1 Image Acquisition for Handwritten Text

The image-acquisition process is identical to the one used in object recognition. The only differences here are that we added a blue rectangular frame using the OpenCV library, in order to help the user position the board correctly within the frame, also now the NAO uses a better analysis (1280×960 pixel), because it helped us a lot with the image proccessing. Lastly, it sets the color space to BGR.

4.2.2 Image Proccess for Handwritten Text

In this part, we had to help the robot recognize the text by proccessing the image and at the end get each of the letters, digits or symbols as a picture. To do that we used a lot of available methods of OpenCV's library. At first, we loaded the captured image from the previous step (Figure 4.1) and converted it to grayscale using the `cvtColor()` method. Then, we detected first the edges with `Canny()` and used Hough Line Transform to define the straight lines. Then, to get the text as clearly as we could, we used the `line()` method to darken the lines of each letter and saved this result in an new image (Figure 4.2). Because our dataset was in white background with black letters, we had to proccess the new image

and turn into the same background as our dataset. This new image was created by using the `pyrMeanShiftFiltering`, which smooths regions while preserving edges making it more reliable and with less noise to succesfully seperate the letters. Before saving the new image, we also cut out the bounding box from the first image (Figure 4.3). Having the new image now we had to use `findCountours()` to find the letters, because it detects the curve that connects all the continuous points with the same color or intensity, allowing us this way to detect each letter. To do that we had first to convert the image to white letter with black background using the method `cvtColor()` (Figure 4.4). After detecting contours, we placed each recognized letter to a bounding box, using the `boundingRect()` method with a fixed padding, to ensure that they weren't cropped too tightly and to match the dataset pictures (Figure 4.5).

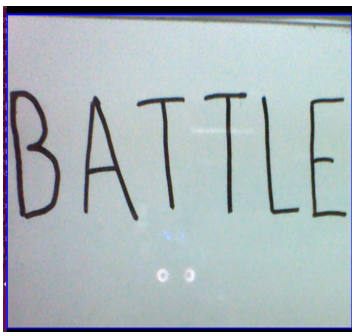


Figure 4.1: Original picture taken by NAO.

Figure 4.2: The picture with darker lines to define the letters.

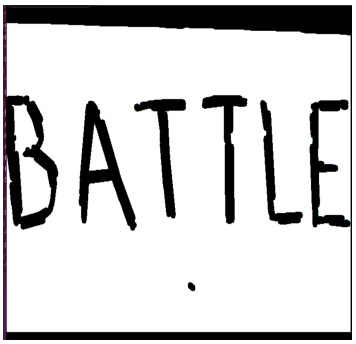


Figure 4.3: The picture converted to white background with black letters.

Figure 4.4: The picture converted to white letters with black background.

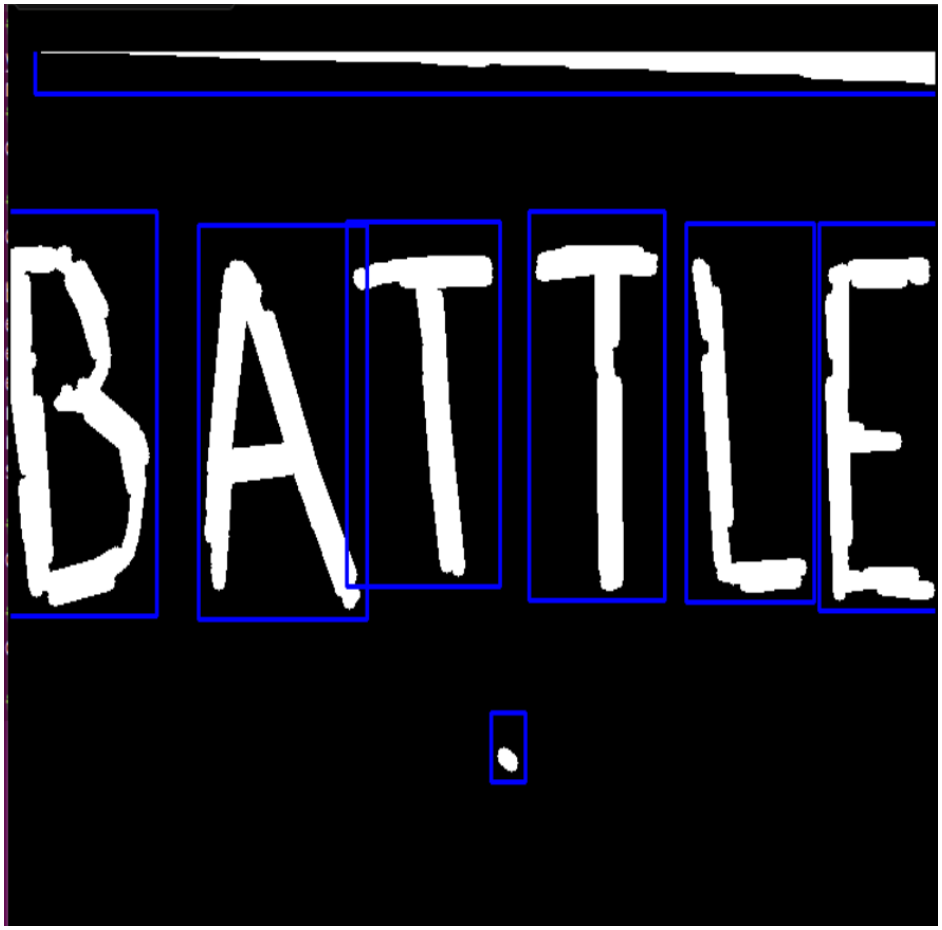


Figure 4.5: The picture with the bounding boxes on the letters.

Then we saved with the `imwrite()` method every letter to a specific folder and gave each picture the name of its maximum x variable, so it can be recognized in order. Of course to avoid any noise, we filtered every contour detected either in the edges or where the difference between the minimum and maximum x variable was smaller then 50. Finally, the saved images with each letter were all taken with their cropped dimensions, that are standard, from their detection part by the thresholded image, so it could be matched with the dataset's images and be identified. (Figures 4.6, 4.7, 4.8, 4.9, 4.10, 4.11)

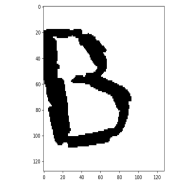


Figure 4.6: First Letter B.

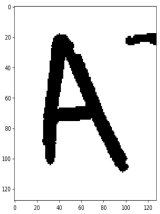


Figure 4.7: Second Letter A.

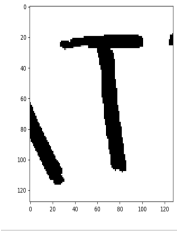


Figure 4.8: Third Letter T.

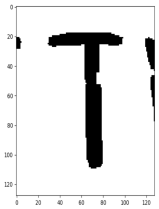


Figure 4.9: Fourth Letter T.

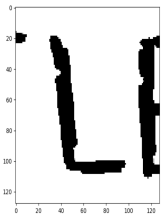


Figure 4.10: Fifth Letter L.

Figure 4.11: Sixth Letter E.

4.3 Text Recognition

To solve the subproblem of letter recognition and make NAO recognize each letter, we had to train our system using a Deep Neural Network and for that we needed a dataset.

The dataset we used was also used by Dimitrios Kavroulakis' diploma thesis titled "Visual Recognition and Writing with the NAO Humanoid Robot" [46]. He used a dataset [42], that contains many images of handwritten uppercase letters, lowercase letters, and numbers, written by 3.600 authors. However, to match his system's needs, Dimitrios Kavroulakis modified the original dataset to include only uppercase handwritten letters with only the most clearly images, creating the dataset that was highly suitable for our system. For the math game, we used a similar dataset [43] [44], that included handwritten operators and digits.

4.3.1 Recognition System Training

For our system to recognize succesfully the handwritten letters, we used our dataset and Fully connected layers in a Convolutional Neural Network (CNN). We started by loading our dataset and then proccessing our images inside the dataset to get better results. More specifically, we converted every image of the dataset to grayscale and resize it to 128×128 analysis. Then, we normalized our data (divided by 255), modified the input of every image to make it compatible with the CNN input and split randomly the images with the help of train `test_split to train_set` and `test_set` (10% of the dataset). Finally, before we started our model we used to `categorical()` to turn numbers into vectors, because we have a classification model.

With the use of the math library Tensorflow and Keras methods, we built a Sequential model. The architecture started with a 2 Convolutional layer, followed by an Activation layer ReLu and a MaxPooling layer 2×2. Then, we added 3 Convolutional layer same with the first one, but the only difference is we added in each one a Dropout layer to be safe that our model will not overfitt. We also added 1 Flatten layer, because we had 3D feature maps to convert it into 1D feature vectors. After that, we used 3 Dense layers and in the first two of those layers was added an Activation layer ReLu with a Dropout layer. In the last Dense layer an Activation layer Softmax was added with 26 classes for multi-class classification. The choice of the parameters and the number of layers was based on educational sources [45] .After training, the model achieved an accuracy of 96%, which was considered satisfactory, so no further tuning was performed.

The Sequential class has helped us not only with the add() method used above, but also with the use of `summary()`, `compile()`, `fit()`, `evaluate()` and `save()`. The `summary()` helped us see a detailed overview of our neural network architec-

ture. The `compile()` setted up the parameters for the training, such as $loss = categorical\_crossentropy$ for classification, $optimizer = adam$ and $metrics = [accuracy]$ to track how the model is going. The `fit()` function is used to train the model, the `evaluate()` to test the trained model and finaly the `save()` to save the trained model.

Figure 4.12 and Figure 4.13 represent the architectures of the two Convolutional Neural Network (CNN) models used in this project. The activation (ReLU) and max pooling layers, that help in feature extraction and dimensionality reduction, come after several convolutional layers in each model. To avoid overfitting, dropout layers are included. Following the flattening of the feature maps, classification is done using fully connected (dense) layers, ending in a final softmax output layer that is equal to the number of classes. Both models use a similar deep learning pipeline designed for character recognition in handwritten photos, and they have about 167,000 trainable parameters.

```
Using TensorFlow backend.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 64)      640
_________________________________________________________________
activation_1 (Activation)    (None, 126, 126, 64)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 64)        36928
_________________________________________________________________
activation_2 (Activation)    (None, 61, 61, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 12, 12, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
activation_5 (Activation)    (None, 4, 4, 64)          0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                16448
_________________________________________________________________
activation_6 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
activation_7 (Activation)    (None, 32)                0
_________________________________________________________________
dropout_6 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 26)                858
_________________________________________________________________
activation_8 (Activation)    (None, 26)                0
=================================================================
Total params: 167,738
Trainable params: 167,738
Non-trainable params: 0
_________________________________________________________________
Train on 1165 samples, validate on 130 samples
```

Figure 4.12: Letter model architecture.

```
Using TensorFlow backend.
('num classis are ', 14)
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 64)      640
_________________________________________________________________
activation_1 (Activation)    (None, 126, 126, 64)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 64)        36928
_________________________________________________________________
activation_2 (Activation)    (None, 61, 61, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 12, 12, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
activation_5 (Activation)    (None, 4, 4, 64)          0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                16448
_________________________________________________________________
activation_6 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
activation_7 (Activation)    (None, 32)                0
_________________________________________________________________
dropout_6 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 14)                462
_________________________________________________________________
activation_8 (Activation)    (None, 14)                0
=================================================================
Total params: 167,342
Trainable params: 167,342
Non-trainable params: 0
_________________________________________________________________
Train on 3493 samples, validate on 389 samples
```

Figure 4.13: Digits and operators model architecture.

4.4 Connecting Modules for Handwritten Text Recognition

At last we had to connect all the above together to one Python module. To do that we created one Python module for the spelling game and one called for the math game. Each one for the handwritten text recognition part begins with the module takephoto, which captures an image from the NAO and then process it with the module `segmentetion_code`. Finally, we do the recognition part with the images, which are passed through the corresponding trained models.

4.5 Implementation

To implement our project we had to work with specific versions because of the compatibility problems with the NAO version 2.1. We specifically worked with Ubuntu 16.04 in a Linux enviroment and to run our code we used Visual Code Studio with Python 2.7, which is the last compatible version of Python with NAO. Also, we used the library Tensorflow 1.5.0 and Keras 2.2.5.

Chapter 5

Results

5.1 Results from Training the Networks

Below are some graphs that are showing the accuracy and the loss from training of the two convolutional neural networks during a specific number of epochs. The graphs were created with the help of matplotlib, a low level graph plotting library in Python that serves as a visualization utility. More spesifically, in Figure 5.1 and Figure 5.2 we see the results from training the model for the handwritten letter recognition and in Figure 5.3 and Figure 5.4 we see the results from training the model for the digits and operators recognition.

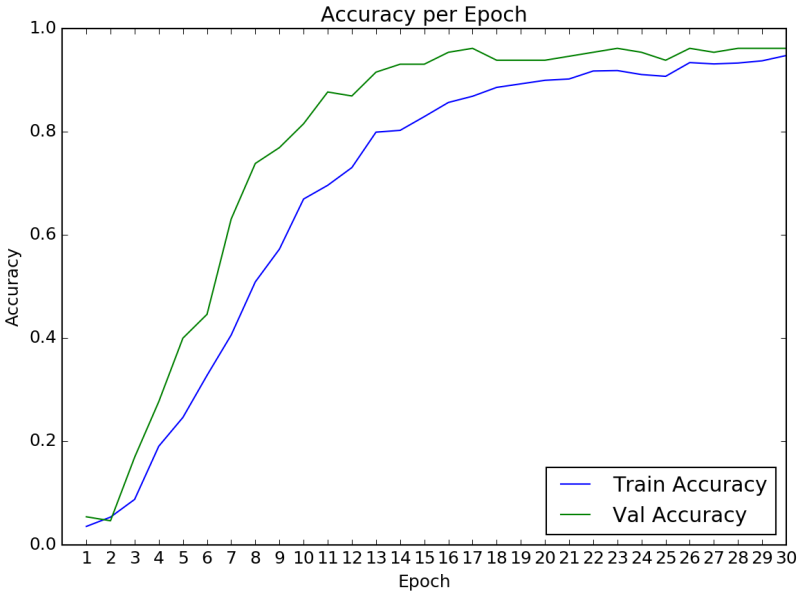


Figure 5.1: Train accuracy and Validation accuracy of the handwritten letter model.

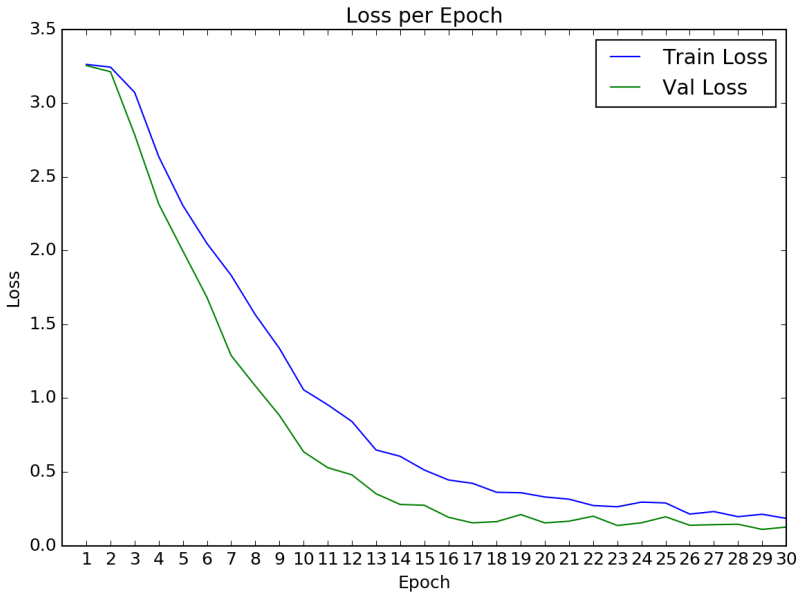


Figure 5.2: Train loss and Validation loss of the handwritten letter model.

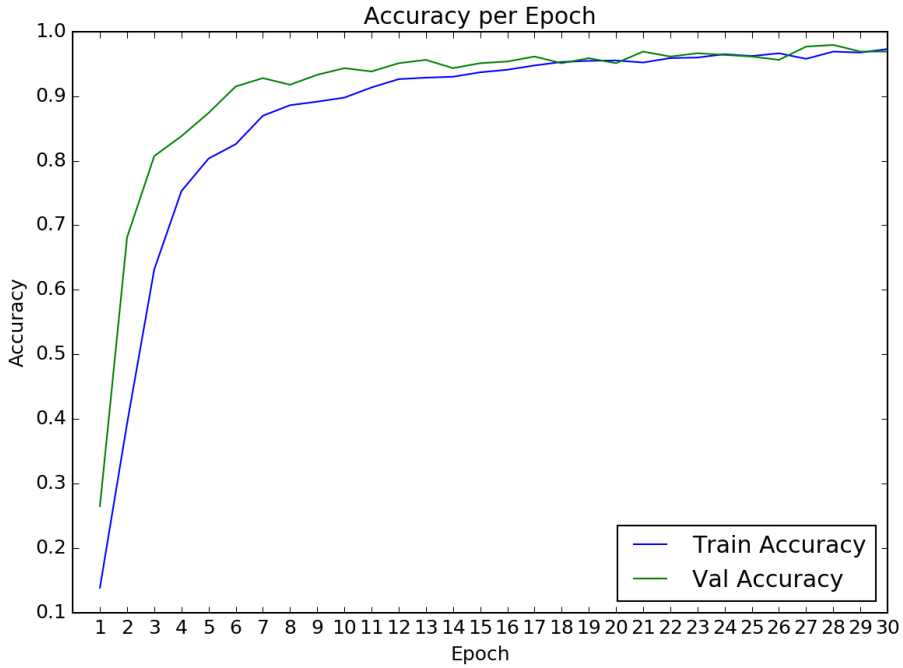


Figure 5.3: Train accuracy and Validation accuracy of the handwritten digits and operators model.

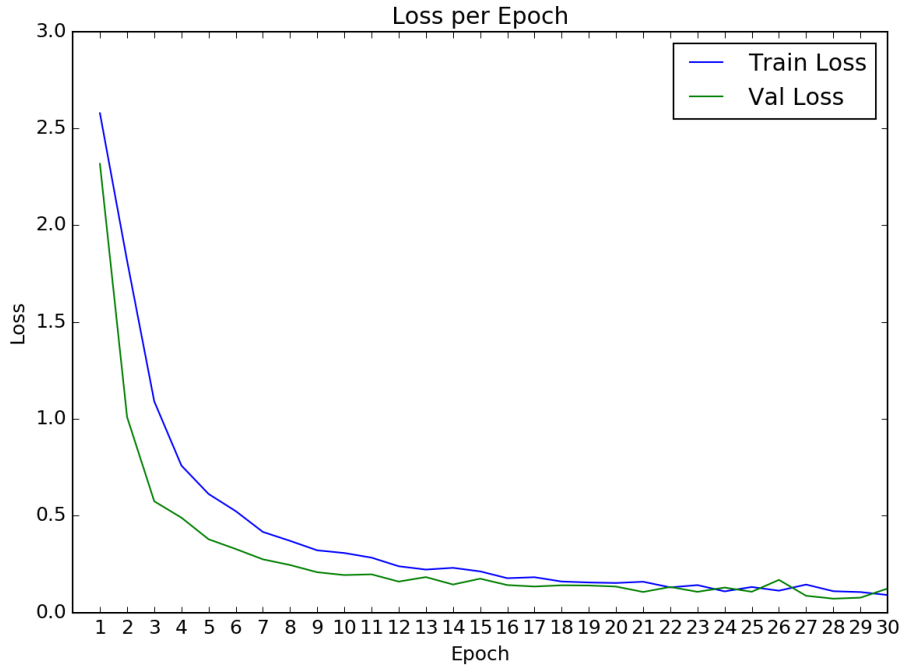


Figure 5.4: Train loss and Validation loss of the handwritten digits and operators model.

We observe that for both models the training accuracy increases steadily, while the validation accuracy improves rapidly and reaches nearly 100%. Simirarly, both training and validation losses decrease, with validation loss remaining even lower than training loss throughout the epochs.

From the Figures (Figure 5.5 and Figure 5.6) below we also see the final results of the accuracy and the loss of each model and the results from the test part of the training. We can see that the letter model achieved a final test accuracy of ~96% and test loss ~16% and the digits model a test accuracy of ~96% and test loss ~13% again.

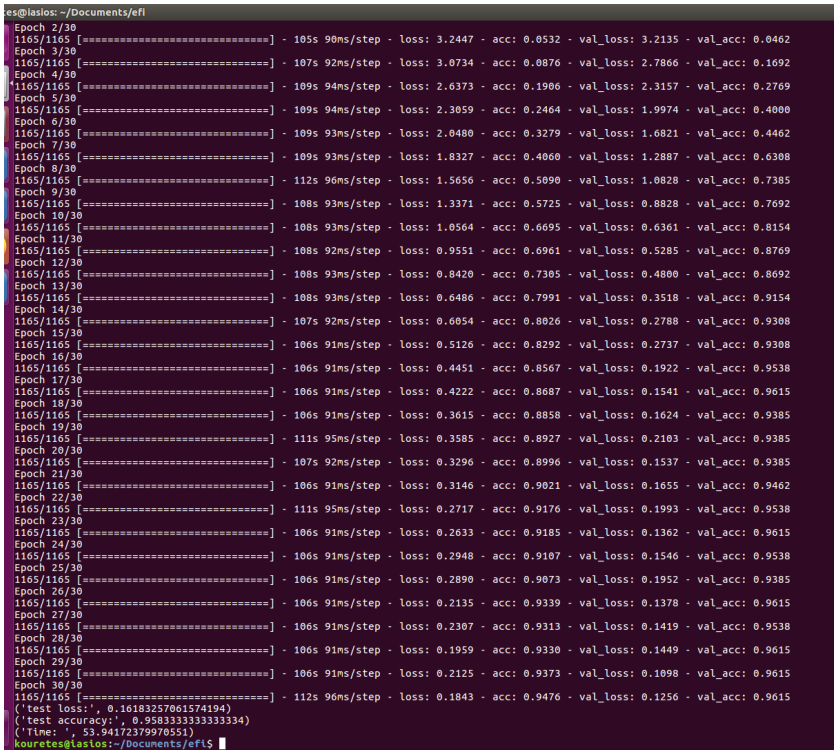


Figure 5.5: Test results from the letter model.

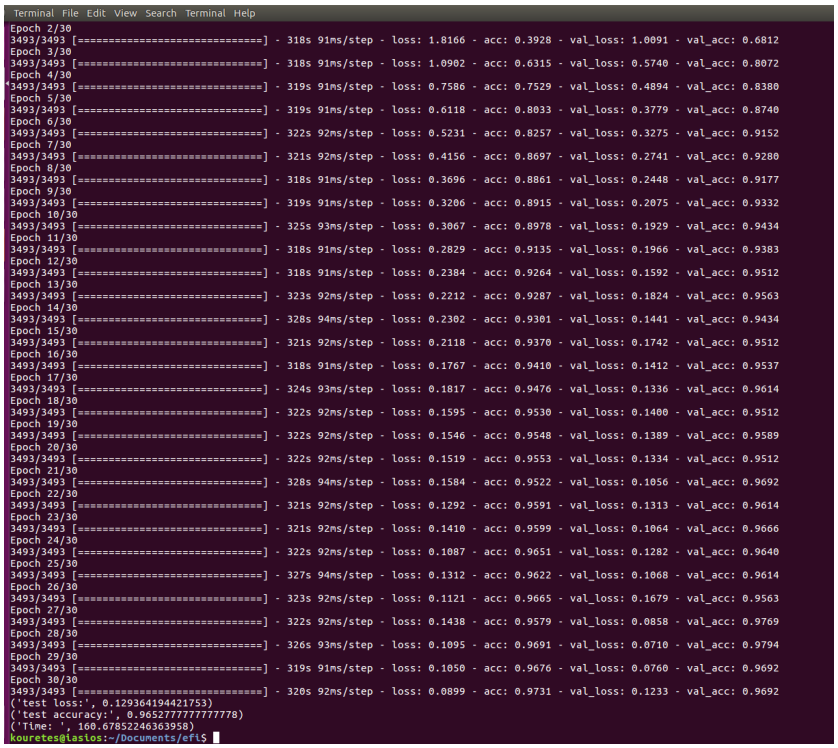


Figure 5.6: Test results from the digits and operators model.

In conclusion, the models produced impressive outcomes. Over epochs, the accuracy of both training and validation increased steadily, with test and validation accuracy approaching 96%. Additionally, the loss steadily declined approaching 13% and suggesting successful learning. There were enough layers in the well-balanced architectures to capture features without overfitting. These findings indicate the CNNs' high accuracy and generalization in handwritten character recognition, which makes them ideal for integration into our system.

5.2 Results from Image Processing

Bellow we will see some examples of pictures taken by the humanoid robot NAO, when processing images during the games and how the model in the end identifies the objects and the handwritten text.

5.2.1 Example of the word "CHAIR"

At first we start by identifying the object given to the robot to capture.

Figure 5.7: Picture of the object "CHAIR".

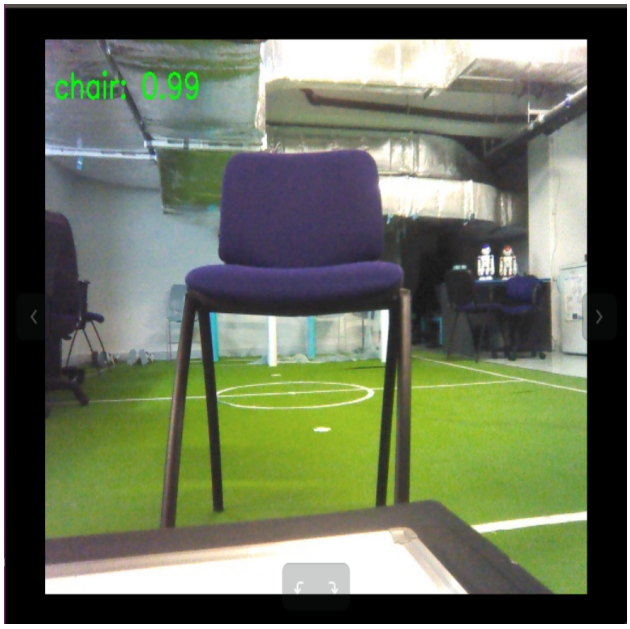


Figure 5.8: “CHAIR”: detected object with the percentage of accuracy.

Then, we continue with capturing the handwritten text and processing the image. Figures 5.9 - 5.19 illustrate the process, from the segmentation of the captured picture to the predictions of the trained model for each detected letter.

Figure 5.9: Original picture taken by NAO.

Figure 5.10: The picture with darker lines to define the letters.

Figure 5.11: The picture converted to white background with black letters.

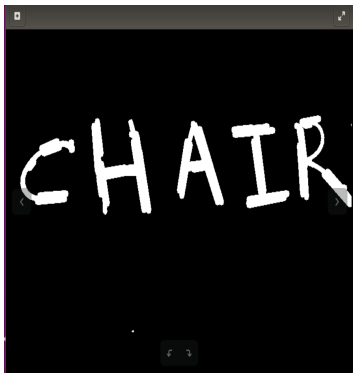


Figure 5.12: The picture converted to white letters with black background.

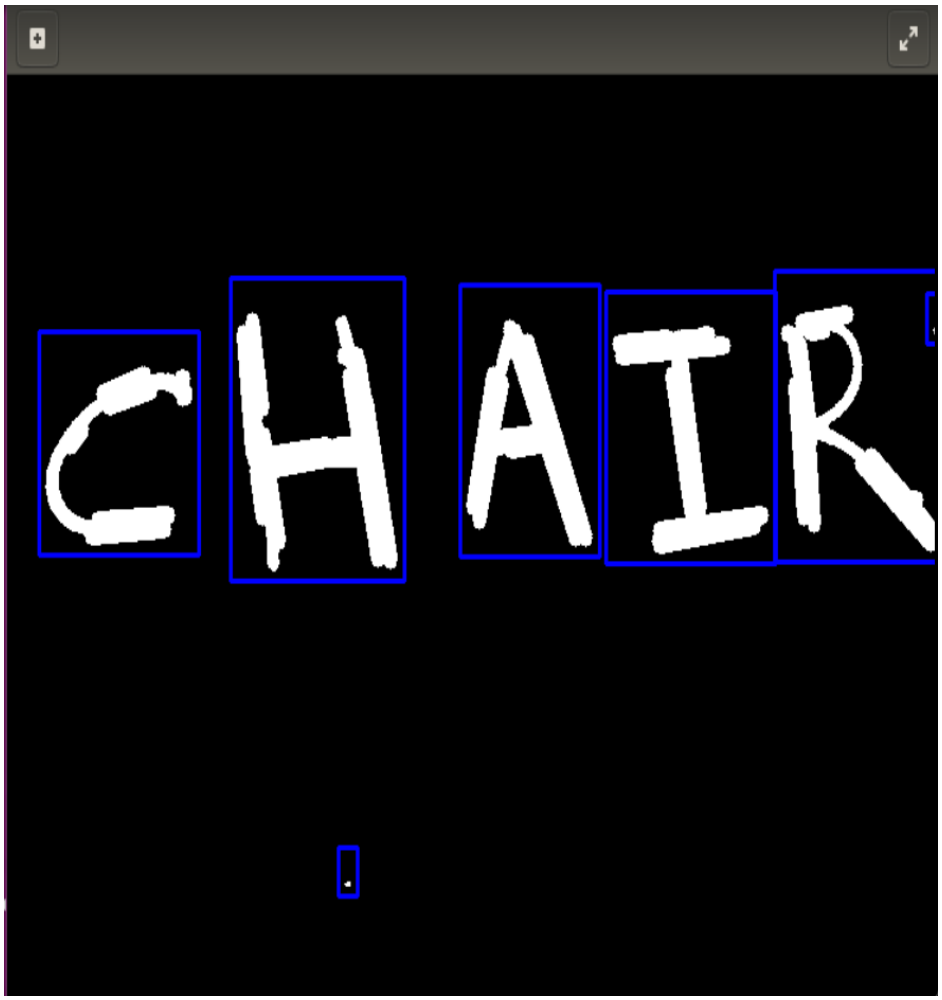


Figure 5.13: The picture with the bounding boxes on the letters.

Below are the distinct letters detected from processing the image.

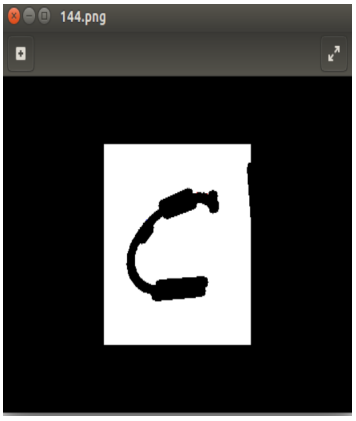


Figure 5.14: First Letter C.

Figure 5.15: Second Letter H.

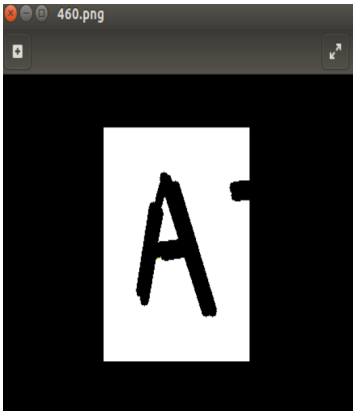


Figure 5.16: Third Letter A.

Figure 5.17: Fourth Letter I.

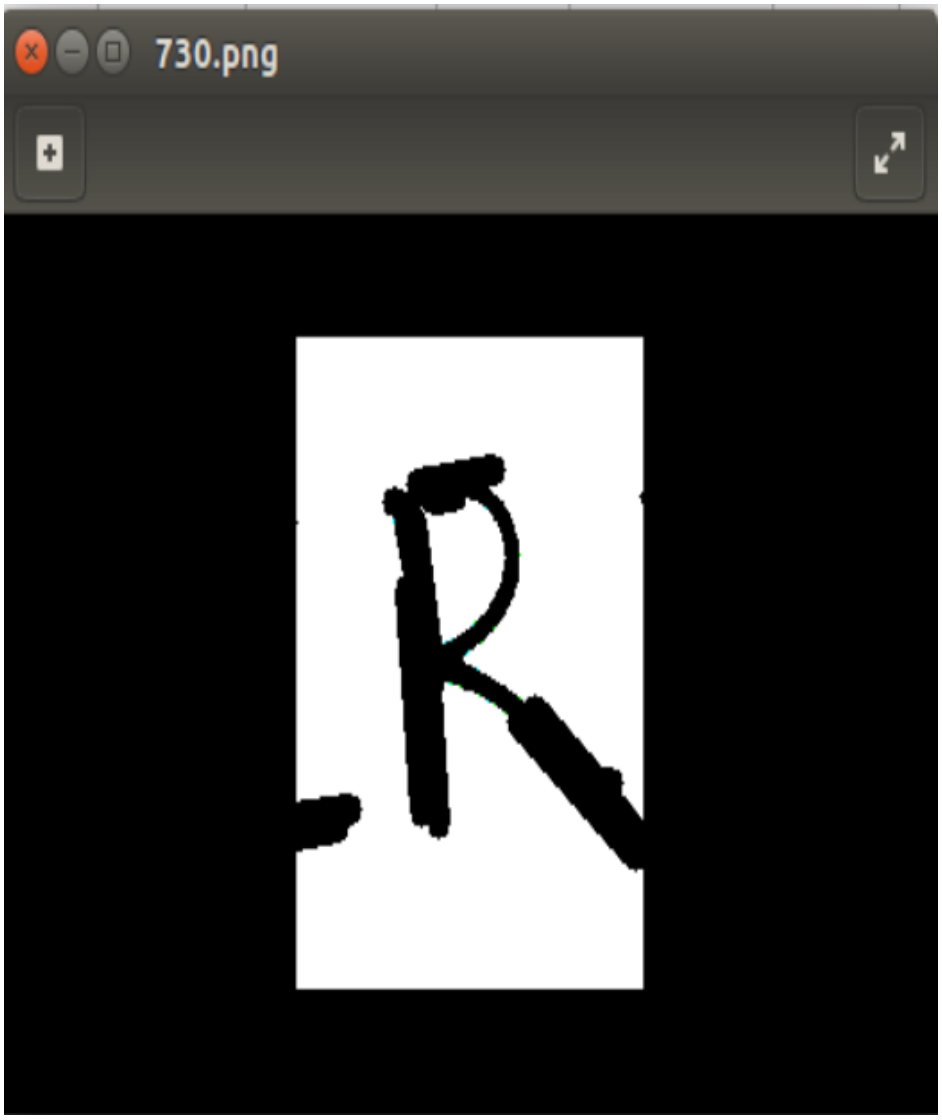


Figure 5.18: Fifth Letter R.

Finally, those are the final results from the predictions of every letter after the model detected them.

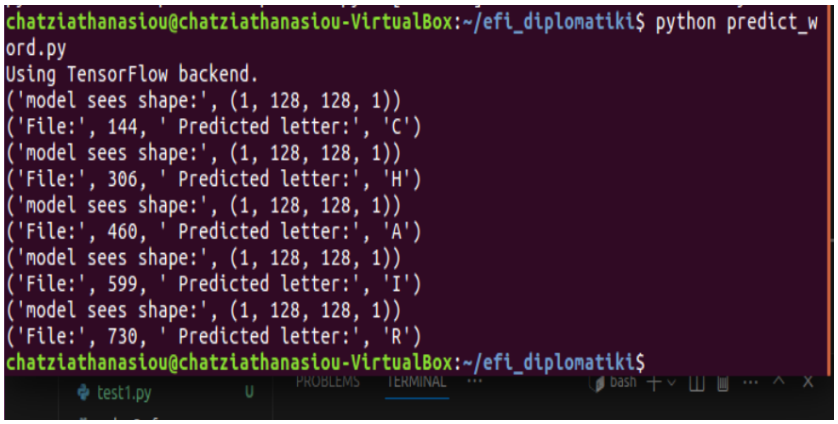


Figure 5.19: “CHAIR”: Results of prediction.

We observe that, despite the noise in the captured image, the model succesfully ignored it and isolated only the detected letters as expected from its training. Then, we took the highest possibillity of every prediction from each letter and the result matched the original handwritten given text.

5.2.2 Example of the word “SCISSORS”

Initially, the object is captured by the robot.

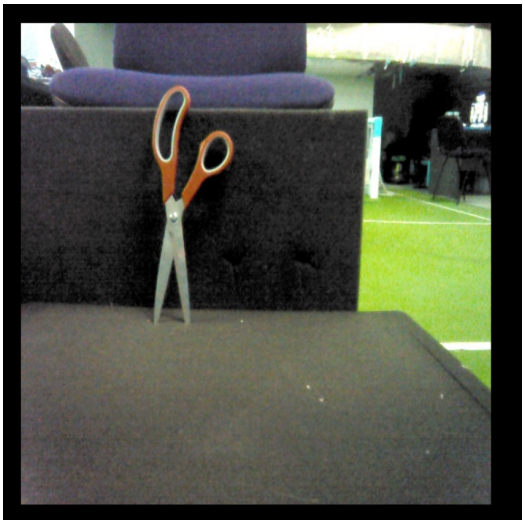


Figure 5.20: Picture of the object “SCISSORS”.

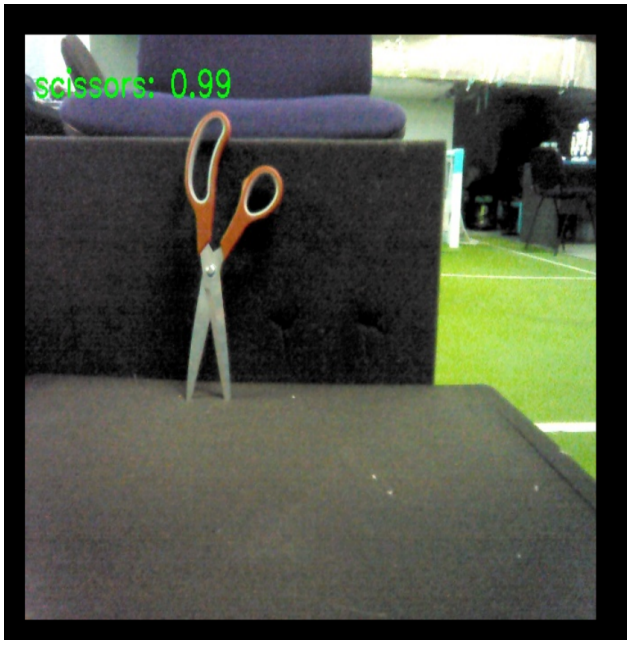


Figure 5.21: “SCISSORS”: detected object with the percentage of accuracy.

Then, again we continue with capturing the handwritten text and processing the image. Figures 5.22 - 5.28 represent the process, from the segmentation of the captured picture to the predictions for each letter.

Figure 5.22: Original picture taken by NAO.

Figure 5.23: The picture with darker lines to define the letters.

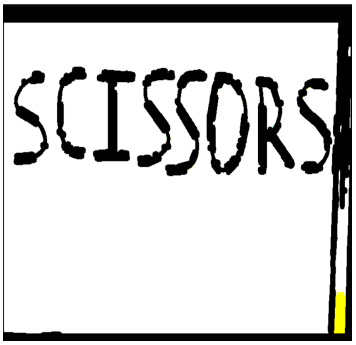


Figure 5.24: The picture converted to white background with black letters.

Figure 5.25: The picture converted to white letters with black background.

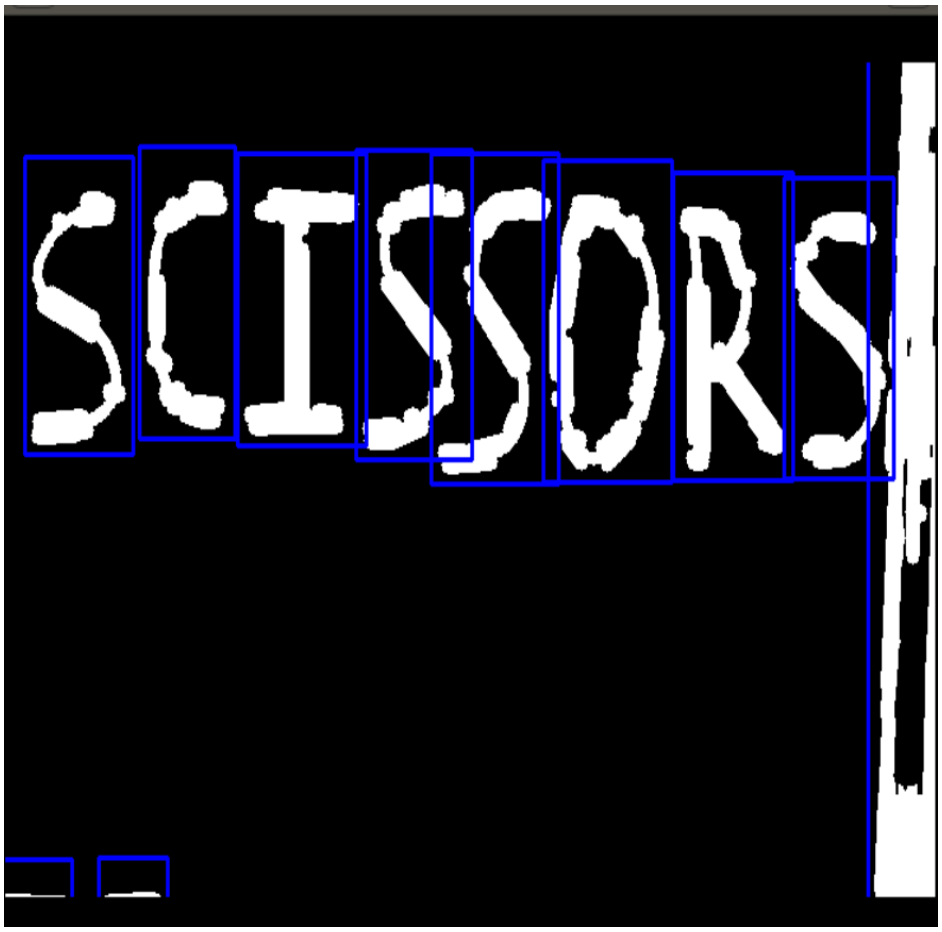
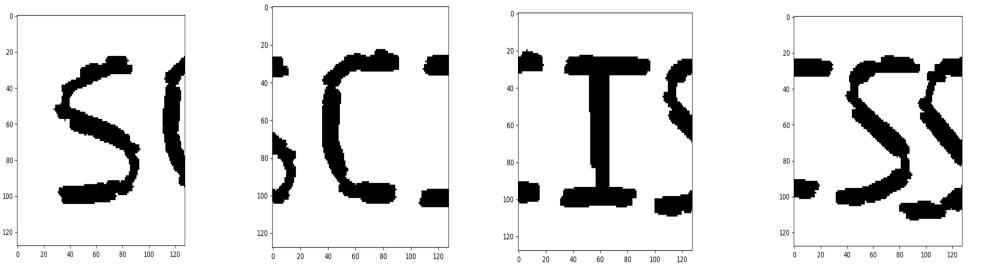
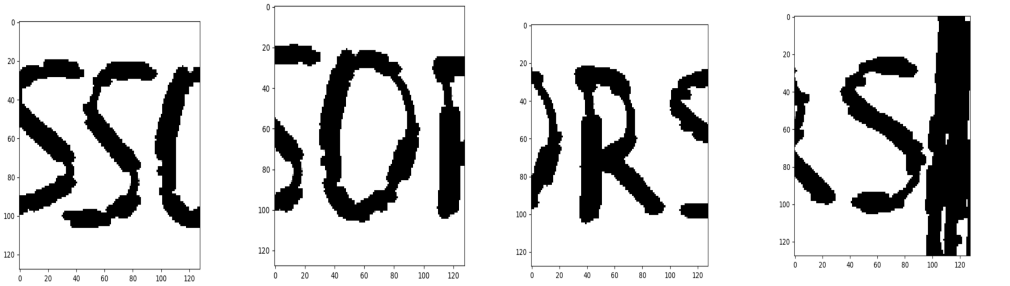


Figure 5.26: The picture with the bounding boxes on the letters.

Below are the distinct letters detected from processing the image.



(a) First Letter S. (b) Second Letter C. (c) Third Letter I. (d) Fourth Letter S.



(e) Fifth Letter S. (f) Fifth Letter O. (g) Fifth Letter R. (h) Fifth Letter S.

Figure 5.27: Extracted letters from the word.

Finally, those are the final results from the predictions of every letter after the model detected them.

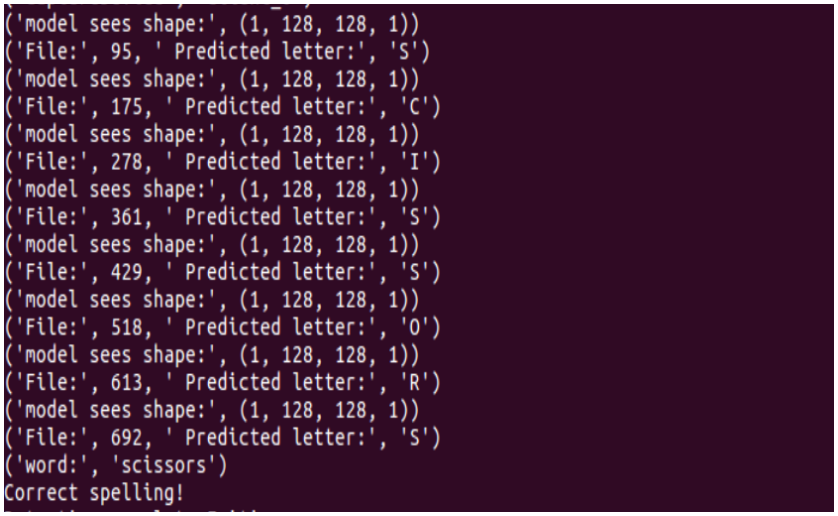


Figure 5.28: “SCISSORS”: Results of prediction.

In this example, we observe again that, despite the noise in the captured image, the model successfully isolated only the detected letters as expected from its training and proceeded with the prediction of each letter.

5.2.3 Example of the math equation “7 + 4”

After the spelling game, we switch to the math game and took several pictures to thoroughly observe how the images were processed with numbers and operators, and how the model detected them again successfully.

At first we have the process of the captured image.

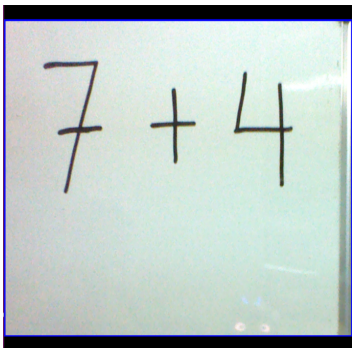


Figure 5.29: Original picture taken by NAO.

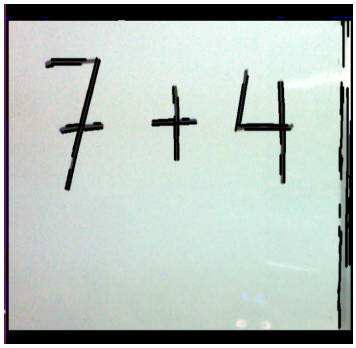


Figure 5.30: The picture with darker lines to define the letters.

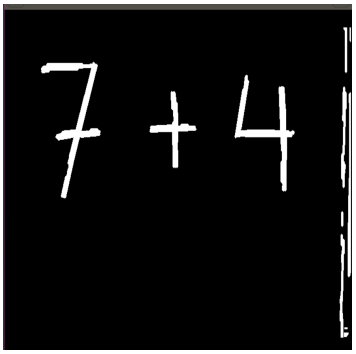


Figure 5.31: The picture converted to black background with white letters.

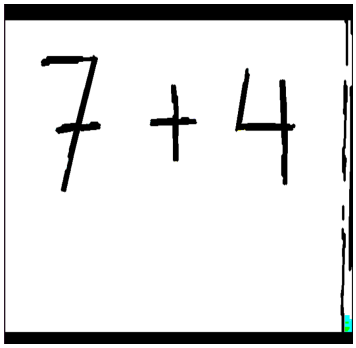


Figure 5.32: The picture converted to black background with white letters.

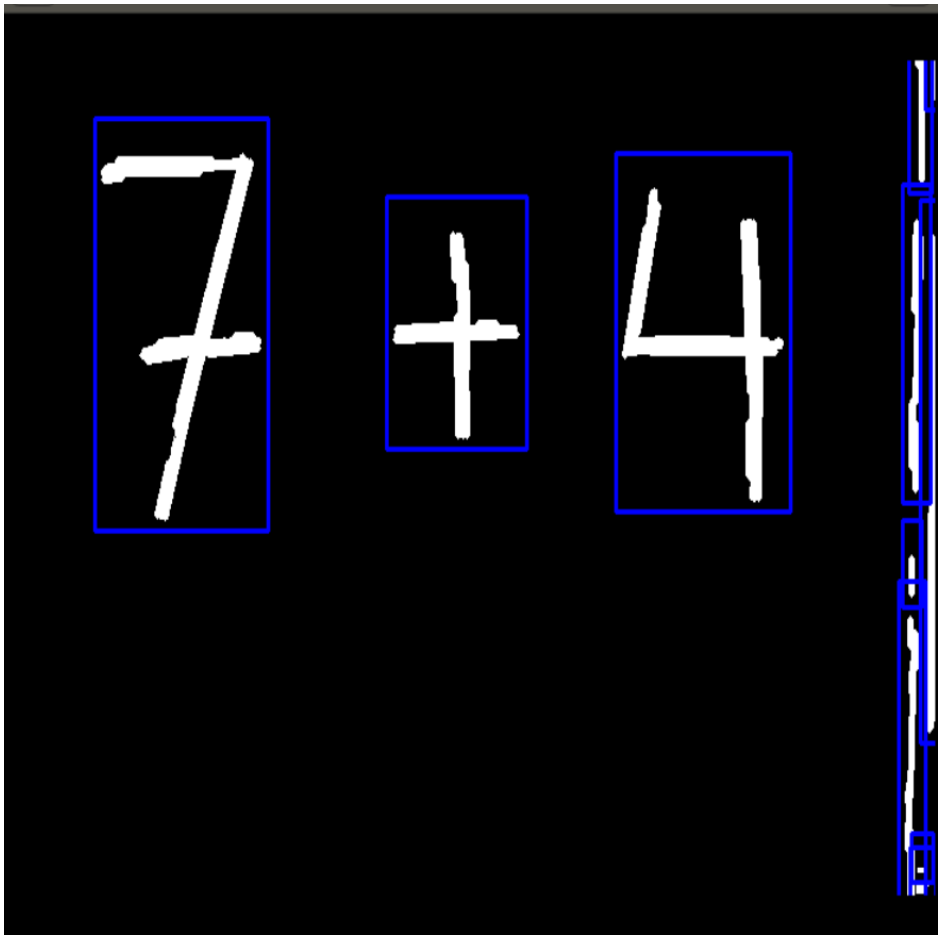


Figure 5.33: The picture with the bounding boxes on the letters.

Below are the detected digits and operators detected from the processing above.

Figure 5.34: First Number 7.

Figure 5.35: Add Operator.

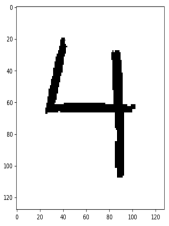


Figure 5.36: Third Number 4.

Finally, we have the results of the predictions for each digit and operator, as well as the final result of the math equation.

```
['201.png', '404.png', '611.png']
('model sees shape:', (1, 128, 128, 1))
('File:', '201.png', ' Predicted symbol:', '7')
('model sees shape:', (1, 128, 128, 1))
('File:', '404.png', ' Predicted symbol:', 'add')
('model sees shape:', (1, 128, 128, 1))
('File:', '611.png', ' Predicted symbol:', '4')
('Equation:', '7+4')
('Result:', 11)
```

Figure 5.37: “7 + 4”: Predictions of each digit and operator, and the final result of the equation.

5.2.4 Example of the math equation “8 / 2”

First, we have the captured image and the corresponding processing steps.

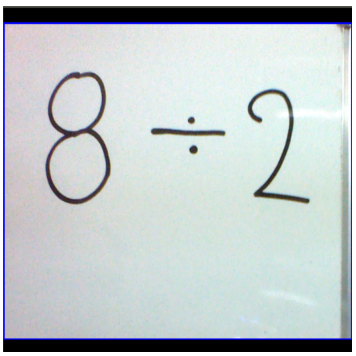


Figure 5.38: Original picture taken by NAO.

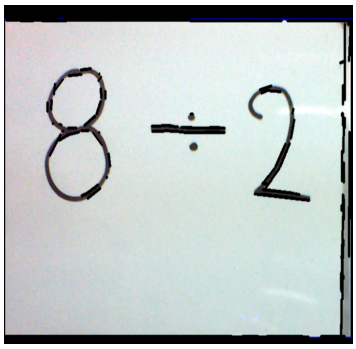


Figure 5.39: The picture with darker lines to define the letters.

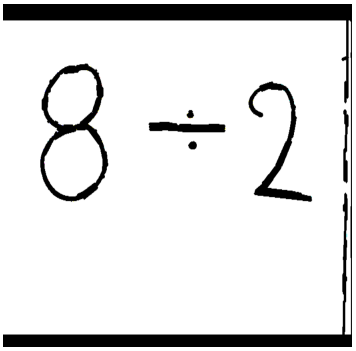


Figure 5.40: The picture converted to black background with white letters.

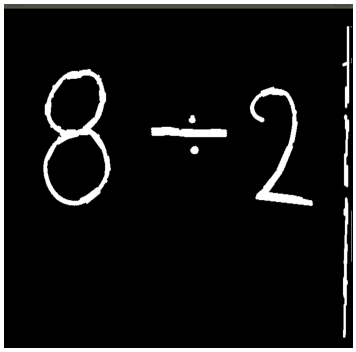


Figure 5.41: The picture converted to black background with white letters.

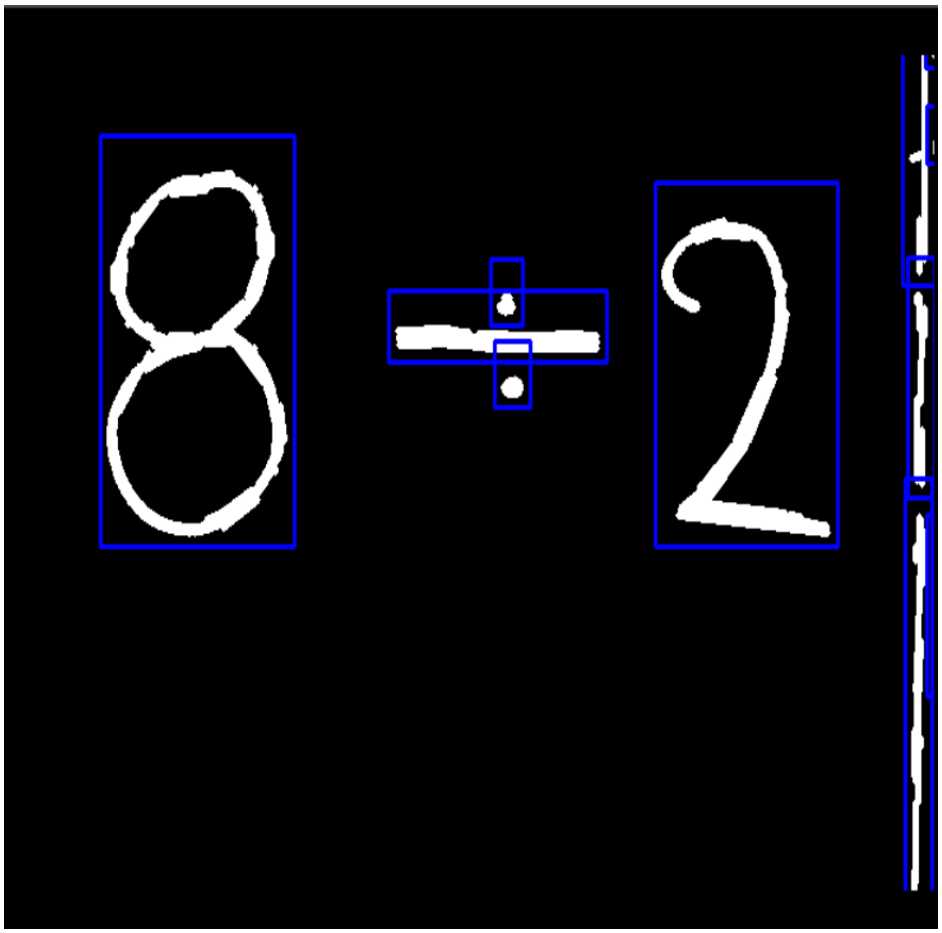


Figure 5.42: The picture with the bounding boxes on the letters.

Below are the detected digits and operators detected from the processing above.

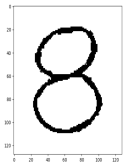


Figure 5.43: First Number 8.

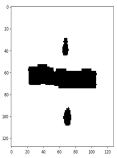


Figure 5.44: Divide Operator.

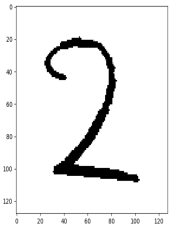


Figure 5.45: Third Number 2.

Finally, we have the results of the predictions for each digit and operator, as well as the final result of the math equation.

```
['223.png', '468.png', '649.png']
('model sees shape:', (1, 128, 128, 1))
('File:', '223.png', ' Predicted symbol:', '8')
('model sees shape:', (1, 128, 128, 1))
('File:', '468.png', ' Predicted symbol:', 'div')
('model sees shape:', (1, 128, 128, 1))
('File:', '649.png', ' Predicted symbol:', '2')
('Equation:', '8/2')
('Result:', 4)
```

Figure 5.46: “8/2”: Predictions of each digit and operator, and the final result of the equation.

5.3 NAO Playing the Games

In this section we will present the completed project with some pictures of NAO playing the games, a video of the completed behavior with some examples and a video of NAO interacting and playing the games with children.

Below are some images showcasing the setup used for the NAO robot tutor to play the games.

Figure 5.47: NAO playing the spelling game

Figure 5.48: Photo of the setup for NAO

Figure 5.49: Photo of the setup for NAO

To assess our robot tutor, we ran a small user evaluation with five children (12 years old) just finishing elementary school. Participation in the experimental sessions took place with parental consent for photographing and video-recording the sessions. Each session was conducted with one child at a time and lasted about 15 minutes. Before each session, we briefed each child about the games, explaining how to interact with NAO, and set a few simple rules to keep things consistent, such as clear handwritting and not moving the robot or the board. After they played, we asked for a short feedback. Overall, the kids found NAO fun and engaging, and most said they'd like to play again. They also pointed out a few problems. The system felt slow between capturing and image and giving an answer, the recognition part worked best with very clear handwriting, and the robot didn't feel expressive enough in its gestures and speech. We had anticipated robot's physical interaction would be limited, due to motor constraints and overheating issues during prolonged use. Therefore, we couldn't safely use full-body movements and kept interaction mostly to head motion and speech. Children also noted the robot's speech was often hard to understand, which we resolved after the evaluation. Figures 5.50 – 5.54 present selected moments from the gameplay sessions.

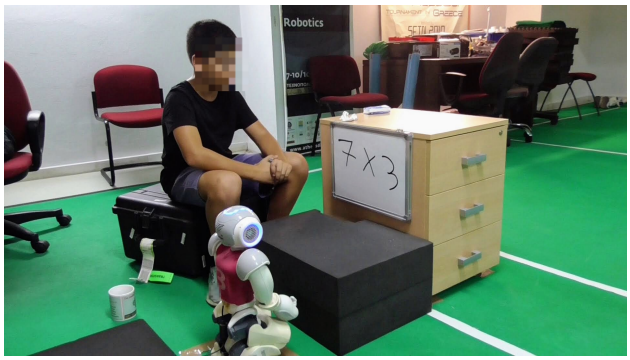


Figure 5.50: Participant playing the math game with NAO.

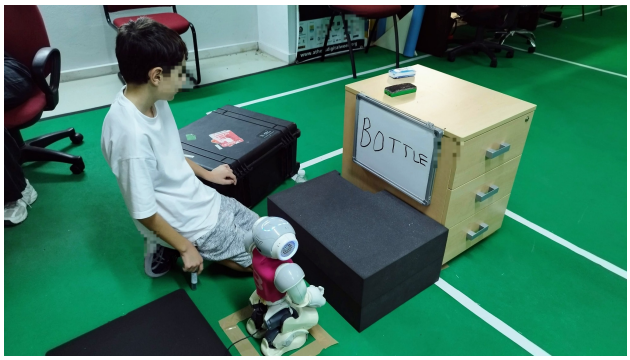


Figure 5.51: Participant playing the spelling game with NAO.

Figure 5.52: Participant writing a word on the board.

Figure 5.53: Participant playing the games with NAO.

Figure 5.54: Participant presenting a book to the robot.

Our own explanatory demos of NAO playing the games can be found here: Demo 1, Demo 2
Short clips of NAO playing the games with the children here: Demo 3, Demo 4, Demo 5

Chapter 6

Conclusions

6.1 Discussion

This diploma thesis was succesfully completed with the humanoid robot NAO playing two interactive educational games with children. Most specifically, NAO is able to recognize the objects, the letters, the digits and the math operators given by the children throughout the games.

The project was implemented with the help of many tools, such like OPENCV's library, Keras and Tensorflow, to manage the camera capture from the robot and to process correctly the image. We used YOLOv3, labeled datasets and Convolutional Neural Networks to train the system, achieving this way results with high acccuracy and validating the effectiveness of our approach. Implementing this project, we came to the conclusion, that working with a humanoid robot is a complicated and demanding process, requiring knowledge in both software development and hardware aspects.

A small user evaluation with children showed strong engagement—most children wanted to play again—indicating good acceptance. Practical limitations were also revealed by the study, including a noticeable latency between capture and response, sensitivity to handwriting clarity, limited physical expressiveness due to motor overheating constraints, and problems with speech intelligibility (which improved after the study conducted).

Finally, this project gave us valuable insight into how robotics, computer vision, and AI can be integrated into a real-time system and the results of this are confirming the potential of AI and robotics to enhance learning experiences, especially in early education.

6.2 Future Work

For future development, several enhancements could improve the educational impact and user experience. Such future work could be developing a multilingual text recognition, allowing the robot to recognize handwritten text in various languages. This would also make NAO a more inclusing educational tool, suitable for different countries and educational systems. Another future direction could be the integration of voice interaction and speech recognition, allowing this way the children to communicate with NAO and have a discussion about either the spelling or the math tasks. Finally, implementing a real-time feedback and correction system would allow the games to be even more interactive, as the robot could point out mistakes while the children are writing or guide them during the game.

6.3 Lessons

This thesis project helped us become familiar with the humanoid robot NAO and improve our knowledge in robotics, computer science, and neural networks. Through its implementation, we proved that the integration of robots into the educational system is both feasible and highly effective, potentially serving as a first step toward the partial replacement of humans in certain educational processes.

Bibliography

1. *NAO Picture*. https://robotsguide.com/robots/nao
2. *SoftBank Robotics*. https://www.softbankrobotics.com/
3. *Maxvision Secures Core Robot Assets of Aldebaran*. https://en.maxvision.com.cn/newsDetail/68779D9B4CEDFD0001443874
4. *RoboCup Standard Platform League*. https://spl.robocup.org/
5. *NAO Sensors*. https://www.researchgate.net/figure/Aldebaran-NAO-H25-18_fig3_339532880
6. *NAO Actuators*. http://doc.aldebaran.com/2-1/family/robots/motors_robot.html
7. *NAOqi*. http://doc.aldebaran.com/2-5/index_dev_guide.html
8. *OpenCV*. https://opencv.org/about/
9. *Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach, 4th US edition", 2020.*
10. *cvtcolor() function*. https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html
11. *cvtcolor() example*. https://www.geeksforgeeks.org/python/python-opencv-cv2-cvtco
12. *Canny Edges Detector*. https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
13. *Canny Edges Detector Example*. https://en.wikipedia.org/wiki/Canny_edge_detector
14. *Hough Line Transform*. https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

15. *Line Method*. https://www.geeksforgeeks.org/python/python-opencv-cv2-line-meth
16. *PyrMeanShiftFiltering*. https://docs.opencv.org/4.x/d4/d86/group_
_imgproc__filter.html#ga9fabdce9543bd602445f5db3827e4cc0
17. *Threshold Function*. https://docs.opencv.org/4.x/d7/d4d/tutorial_
py_thresholding.html
18. *Threshold Example*. https://www.geeksforgeeks.org/python/python-opencv-cv2-line
19. *FindContours Function*. https://www.geeksforgeeks.org/python/find-and-draw-conto
20. *Tensorflow*. https://www.nvidia.com/en-us/glossary/tensorflow/
21. *Adrian Kaebler and Gary Bradski, "Learning OpenCV", 2008*.
22. *Tensorflow workflow*. https://www.geeksforgeeks.org/python/introduction-to-tenso
23. *Keras*. https://www.geeksforgeeks.org/deep-learning/keras-layers-api/
24. *Keras*. https://blog.roboflow.com/what-is-yolov3/
25. *Neural network*. https://www.ibm.com/think/topics/neural-networks
26. *Neural netwrork structure picture*. https://www.specbee.com/blogs/
neural-networks-what-does-future-artificial-intelligence
27. *Deep neural network*. https://www.ibm.com/think/topics/deep-learning
28. *Deep neural netwrork structure picture*. http://neuralnetworksanddeeplearning.
com/chap5.html
29. *Convolutional Neural Networks*. https://www.ibm.com/think/topics/
convolutional-neural-networks
30. *Charu C. Aggarwal, "Neural Networks and Deep Learning: A Textbook", 2018*.
31. *Aurelien Geron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow second edition Concepts, Tools, and Techniques to Build Intelligent Systems", 2017*.
32. *Convolutional Neural Networks Example*. https://www.superannotate.
com/blog/guide-to-convolutional-neural-networks
33. *Convolutional Neural Networks Architecture picture*. https://zilliz.
com/glossary/convolutional-neural-network

34. *Pooling layer*. https://www.researchgate.net/figure/llustration-of-Max-Pooling-a fig2_333593451

35. *Activation functions in Neural Networks*. https://www.geeksforgeeks.org/machine-learning/activation-functions-neural-networks/

36. *Softmax Function*. https://www.geeksforgeeks.org/deep-learning/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applicatio

37. *Softmax Function Curve*. https://botpenguin.com/glossary/softmax-function

38. *Qin Yang, Huan Lu, Dandan Liang, Shengrong Gong, Huanghao Feng, "Surprising Performances of Students with Autism in Classroom with NAO Robot", June 2024,*. https://arxiv.org/pdf/2407.12014

39. *Cen Li, Ebosehon Imeokparia, Michael Ketzner, Tsega Tsahai, "Teaching the NAO Robot to Play a Human-Robot Interactive Game",International Conference on Computational Science and Computational Intelligence (CSCI), 2019*. https://par.nsf.gov/servlets/purl/10209917

40. *Fung, K.Y., Fung, K.C., Lui, T.L.R. et al. Exploring the impact of robot interaction on learning engagement: a comparative study of two multi-modal robots. Smart Learn. Environ. 12, 12 (2025)*. https://slejournal.springeropen.com/articles/10.1186/s40561-024-00362-1

41. *Pande, A., and Mishra, D. (2024). Humanoid robot as an educational assistant – insights of speech recognition for online and offline mode of teaching. Behaviour and Information Technology, 44(5), 975–992*. https://doi.org/10.1080/0144929X.2024.2344726

42. *EMNIST Handwritten Characters Dataset*. https://www.nist.gov/srd/nist-special-database-19

43. *Digits Dataset*. https://github.com/sumit-kothari/AlphaNum-HASYv2/blob/master/README.md

44. *Operators Dataset*. https://www.kaggle.com/datasets/sagyamthapa/handwritten-math-symbols/data

45. *Educational online source for training a model*. https://www.youtube.com/watch?v=u3FLVbNn9Os

46. *Dimitrios Kavroulakis, "Visual recognition and writing with the NAO humanoid robot", Diploma Thesis, School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece, 2020*. https://doi.org/10.26233/heallink.tuc.87831