Pict2Text 2.0. Identifying and classifying the pictograms contained in an image

Pict2Text 2.0. Identificando y clasificando los pictogramas contenidos en una imagen

Final project
Course 2020–2021

Authors:
Gasan Mohamad Nazer
and
Veronika Borislavova Yankova

Director:
Virginia Francisco Gilmartín

Co-director:
Susana Bautista Blasco

Bachelor's Degree in Software Engineering
Faculty of Informatics
Complutense University of Madrid

Pict2Text 2.0. Identifying and classifying the pictograms contained in an image

Pict2Text 2.0. Identificando y clasificando los pictogramas contenidos en una imagen

Bachelor's Degree Final Project in Software Engineering

Authors:
Gasan Mohamad Nazer
and
Veronika Borislavova Yankova

Director:
Virginia Francisco Gilmartín

Co-director:
Susana Bautista Blasco

Convocation: *June 2021*

Bachelor's Degree in Software Engineering
Faculty of Informatics
Complutense University of Madrid

June 15, 2021

Abstract

Nowadays, communication is a basic need in our society. However, some people cannot use the typical methods of communication for reasons that don't depend on them. A way of communication for those people is using pictograms. However, for people without specific training, understanding sentences formed by those pictograms is not easy, if not impossible. That's why tools that translate sentences written with pictograms into natural language are essential.

Pict2Text 1.0 is the only existing tool that translates messages written with pictograms to natural language (Spanish). Unfortunately, it still has to be improved. One of the biggest flaws of the tool is the fact that the message with pictograms has to be created manually by looking for each pictogram in the search engine provided by the application. At this current state, the people who most need the tool can't use it because they would not be able to type the words to compound the message to select the pictograms. For that reason, in this final project, we have focused our efforts on improving that feature by giving people the option to upload a picture of a sentence written with pictograms instead of creating the message in the above-mentioned way. To do that, we have implemented and tested two Machine Learning models: one to detect the pictograms in a picture (YOLO) and the other to identify the word associated with each one of the pictograms in an image (One-shot learning algorithm). We trained and tested several versions of each of them. We have managed to recognize more than one pictogram in a zoomed-in picture. The classification algorithm is able to recognize around 70% of the pictograms correctly. Even though there is work left, the results we obtained are encouraging enough to believe that if we increase the training set, the prediction accuracy will be more than satisfactory.

Keywords

- Pictogram.
- ARAASAC.
- Machine Learning.
- Computer vision.
- One-shot learning.
- YOLO.

Resumen

Hoy en día, la comunicación es una necesidad básica en nuestra sociedad. Sin embargo, algunas personas no pueden utilizar los métodos típicos de comunicación por razones que no dependen de ellos. Una forma de que esas personas se comuniquen es utilizando pictogramas. Sin embargo, para las personas sin formación específica, entender las frases formadas por esos pictogramas no es fácil, y a veces puede llegar a ser imposible. Por eso, las herramientas que traducen al lenguaje natural las frases escritas con pictogramas, son esenciales.

Pict2Text 1.0 es la única herramienta existente que traduce mensajes escritos con pictogramas al lenguaje natural (español). Por desgracia, todavía tiene que ser mejorada. Uno de los mayores defectos de la herramienta es el hecho de que el mensaje con pictogramas tiene que crearse manualmente buscando cada pictograma en el buscador que proporciona la aplicación. En este estado actual, las personas que más necesitan la herramienta no pueden utilizarla ya que no son capaces de escribir las palabras que componen el mensaje para seleccionar los pictogramas. Por eso, en este proyecto final, hemos centrado nuestros esfuerzos en mejorar esa función dando a las personas la opción de subir una imagen de una frase escrita con pictogramas en lugar de crear el mensaje de la forma mencionada. Para ello, hemos implementado y probado dos modelos de Machine Learning: uno para detectar los pictogramas de una imagen (YOLO) y otro para identificar la palabra asociada a cada uno de los pictogramas de una imagen (algoritmo One-shot learning). Hemos entrenado y probado varias versiones de cada uno de ellos. Hemos conseguido reconocer más de un pictograma en una imagen ampliada. El algoritmo de clasificación es capaz de reconocer correctamente alrededor del 70% de los pictogramas. Aunque queda trabajo por hacer, los resultados que hemos obtenido son lo suficientemente alentadores como para creer que si aumentamos el conjunto de entrenamiento, la precisión de la predicción será más que satisfactoria.

Palabras clave

- Pictogramas.
- ARAASAC.
- Machine Learning.
- Visión artificial.
- One-shot learning.
- YOLO.

Contents

List of figures

List of tables

Chapter 1

Introduction

This chapter will display the motivation behind this project and its objectives. At the end of the chapter, we present the document structure with the different chapters and a brief description of them.

1.1 Motivation

Communication is one of the pillars of interpersonal relationships, a fundamental need in our society. However, for some people, communication requires a lot of effort. Their differences are an uncrossable barrier, and it is almost impossible to have a human connection using the traditional way of communication. To remove this barrier, an alternative approach should be used, for example, the use of alternative ways of communication as pictograms. These graphic images, representing an object or a concept, have helped people with special needs to communicate. However, the majority of the population does not understand pictograms.

To include pictograms into the communication between people with disabilities and those without, we can use modern technology and create software to be a mediator between the two sides.

Currently, multiple tools allow transforming natural language into pictograms, but there is only one tool that converts a message written with pictograms into text: Pict2Text 1.0 (González Álvarez and López Pulido, 2019). It translates pictograms into natural language, but to provide an input message, it must be created by searching for every single pictogram that compounds the message in the search engine of the application. This is not good enough for people who want to communicate with people with special needs, as people with disabilities who need pictograms cannot think about the words that compose the sentence and search for them in the search engine. This problem can be solved by giving those people the option to upload

a file with the message written with pictograms or take a picture of the sentence. We aim to build a new version of Pict2Text that allows uploading messages written with pictograms more efficiently and improves the translator coverage.

The beneficiaries of this software are people who don't understand pictograms but want to interact with people who need them to communicate. This project will help people understand each other better, creating a more equal society.

1.2 Goals

The main goal of our project is to improve Pict2Text. To do that, we have two main objectives:

1. To change the form in which Pict2Text 1.0 receives the text written with pictograms. The application should allow the user to upload a picture with the message written with pictograms instead of searching for each pictogram in the integrated search engine.

2. To improve the translations provided by Pict2Text 1.0. At the moment, Pict2Text 1.0 can translate only simple phrases containing one subject, one object, and one verb. We aim to provide the opportunity to translate more complex sentences.

During the implementation, we will follow a Services-Oriented Architecture (SOA), constructing web services that implement each of the functionalities added. This will increase the maintainability of the application and will allow all our functionalities to be integrated into other developments.

In addition, we would like this work to allow us to consolidate and expand the knowledge acquired during the Software Engineering Degree.

1.3 Document structure

The document is structured as follows. Chapter 2 (State of the Art) covers the Augmentative and Alternative Systems of Communication, the state of Pict2Text 1.0, the general idea of the machine learning algorithms and models we have used as well as the main tools we needed for the implementation. Chapter 3 (Software development methodology) describes the methodology we have chosen to develop our application, the type of tests we have done to ensure the correct performance of our application and the continuous deployment we have implemented. Chapter 4 (Pict2Text 2.0) provides detailed information on everything we have developed ourselves. The next chapter (Individual work) describes the work each of us has done for the project. In

the final chapter (Conclusions and Future Work), we will present the conclusions we have obtained from our project, as well as the future work that has to be done to complete it.

Chapter 2

State of the Art

In this chapter, we briefly define Augmentative and Alternative Systems of Communication (section 2.1) and pictograms (section 2.2). Also, in section 2.3, we review Pict2Text 1.0, which is the tool that serves as the basis for our work. In section 2.4 we analyze how machine learning can be used for image processing. Last, in section 2.5 we present the tools we have used in the implementation of our project.

2.1 Augmentative and Alternative Systems of Communication (AASC)

The Augmentative and Alternative Systems of Communication (AASC)[1] (Beukelman and Light, 2013) are communication systems, alternative to the natural language, that don't use spoken or written words but can transmit information. They are used by people, who cannot use natural language, or it is not sufficient for them to express themselves. They are created to increase the communication capabilities of the people who use them.

The AASCs do not arise naturally, but they need previous knowledge. There are two types of AASCs - those who need additional equipment such as objects, pictures, pictograms, etcetera, and those that do not need any equipment.

AASC includes different systems of symbols: graphic and gestural. The gestural symbols can vary widely from mimics to hand signs. The graphic symbols can be used by both people with an intellectual disability or with a motor disability. Examples of graphic symbols are drawings and pictures, as well as pictograms, which will be better explained in the next chapter.

Those systems provide various benefits for their users. They prevent or

[1] https://arasaac.org/aac

decrease the isolation of people with disabilities, helping with the improvement of social and communication abilities. Also, AASCs are relatively easy to learn and apply and adapted for modern technology.

2.2 Pictograms

Pictograms are AASCs that need additional equipment. They are written signs representing objects from the real world, as shown in Figure 2.1 where a pictogram of ice cream is shown. Pictograms are used in the day to day life at hospitals, malls, airports, etcetera. They are also widely used by people with special needs, to help with communication and social integration.

Figure 2.1: Pictogram representing an icecream.

2.2.1 Pictogram systems

As pictograms are not universal, various systems exist, such as Blissymbolics, CSUP, Minspeak, and ARASAAC, which we will see in more detail in the following subsections.

2.2.1.1 Blissymbolics

Blissymbolics[2] was created in 1949. It is an ideographic language consisting of several hundred basic symbols, each representing a concept. Each symbol is represented by basic forms (circles, triangles) and universal signs (numbers, punctuation signs) and uses colour codification to mark the grammar category. They can be combined to generate new symbols that represent new concepts. Figure 2.2 shows Blissymbolics pictograms for a house, person, love, etcetera. Blissymbolics characters do not correspond to the sounds

[2]https://www.blissymbolics.org/

of any spoken language and have their use in the education of people with communication difficulties.

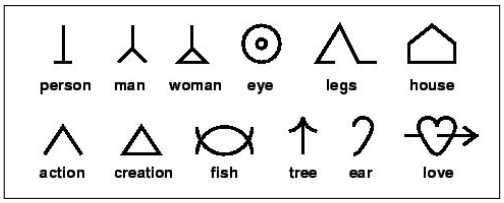

Figure 2.2: Example of Blissymbolics

2.2.1.2 CSUP

The Communication System Using Pictograms (CSUP)[3], developed in 1981 by Mayer-Johnson, is one of the systems that use pictograms to support interactive non-verbal communication. This AASC can be used with a physical or digital board. As shown in Figure 2.3 CSUP uses pictograms for physical objects: school and mother, for events like talk, draw, and also for descriptions as big, cold, close, etcetera. It is designed in a way that it can be used between a person with a disability and a non-disabled person, child and adult, people speaking different languages, and so on.

2.2.1.3 Minspeak

Minspeak[4] is a pictographic system created by Bruce Baker in 1992. Unlike other systems, this system is based on multi-meaning icons whose meaning is determined by the speech therapist and the user. Two or three icons can combine, determined by rule-driven patterns, to code vocabulary. An example of this can be seen in Figure 2.4, where the icon for apple can mean not only apple but also food or eat. In the same figure, we can observe how combining the apple icon with house means grocery, and combining it with a rainbow means the colour red.

2.2.1.4 ARASAAC

The ARASAAC[5] system is the most used pictogram system in Spain. The ARASAAC project was created in 2007, and it currently consists of more than 30.000 pictograms, including complex pictograms with already constructed phrases, in more than 20 languages. The pictograms are separated

[3]http://masquemayores.com/magazine/tipos-de-sistemas-alternativos-y-aumentativos-de-la-comunicacion-sistemas-pictograficos-de-comunicacion/

[4]http://www.minspeak.com/

[5]https://arasaac.org/

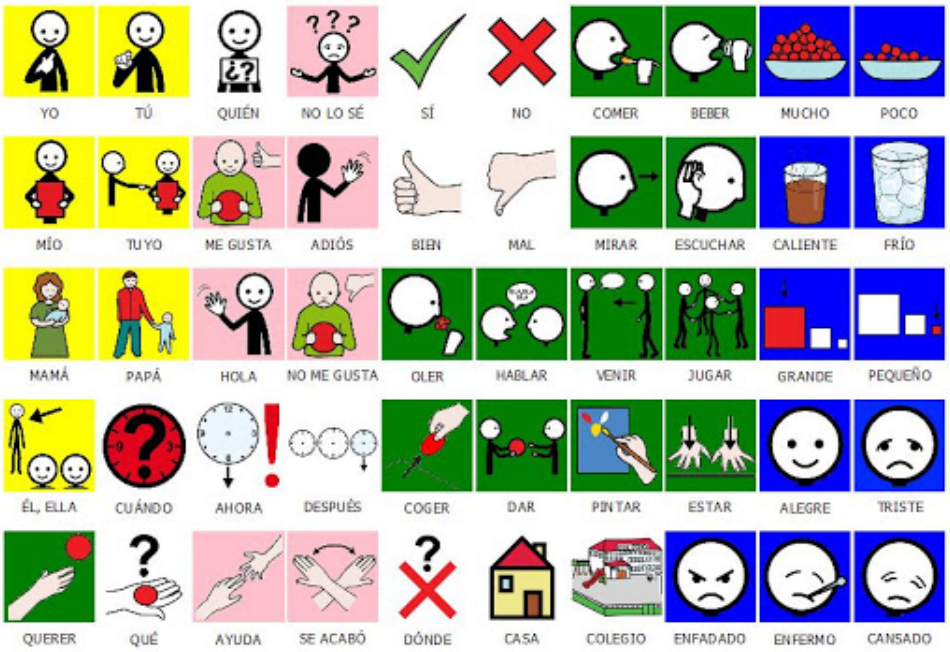

Figure 2.3: Example of Communication System Using Pictograms (CSUP)

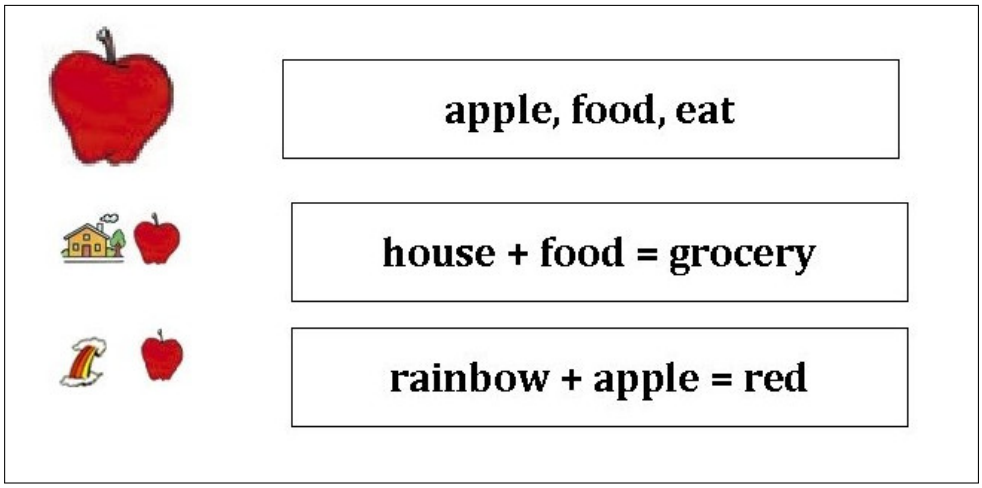

Figure 2.4: Construction of symbologic in Minspeak.

into five groups: coloured pictograms, black and white, photographs, and sign language videos and pictures. Unlike other pictogram systems, ARASAAC makes a difference between singular and plural and genders. For example, in Figure 2.5 can be seen the difference between the pictograms for male and

female teachers.

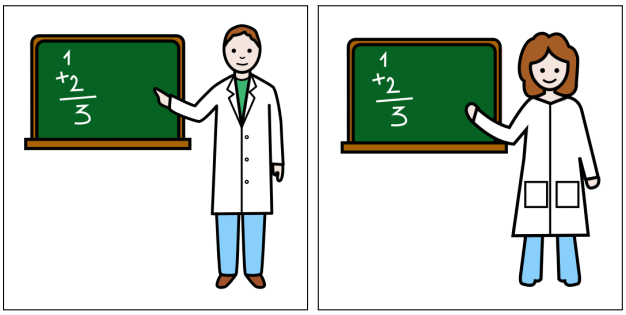

Figure 2.5: Example for ARASAAC differentiation for the word ‘teacher‘.

In ARASAAC one word can be represented by various pictograms. In Figure 2.6 we can observe the different pictograms for the word ‘teacher‘.

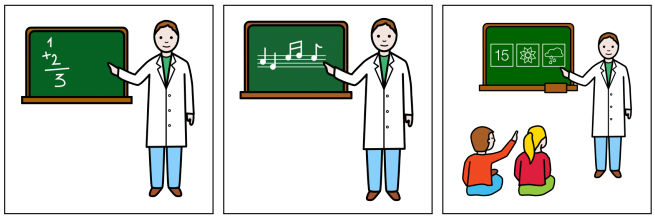

Figure 2.6: Pictograms associated with the word ‘teacher‘ in ARASAAC.

Verbs are not conjugated in ARASAAC, there is only a pictogram for the infinitive of the verb. The tense of the sentence must be determined by pictograms representing yesterday, today, and tomorrow.

ARASAAC is free to use under the Creative Commons license. ARASAAC also provides a web page[6] from where you can search or/and download the pictograms. For developers an API[7] is provided that gives functionalities such as:

- obtaining a pictogram corresponding to a certain word[8]: returns a JSON with the type of the word, the plural forms, and the id among other attributes. In Figure 2.7 can be seen the result from calling the ARASAAC API searching for the word "Spiderman".
- obtaining a pictogram given the id[9]: returns a JSON with the information about the pictogram with that id. In Figure 2.8 you can observe

[6] https://arasaac.org/pictograms/search
[7] https://arasaac.org/developers/api
[8] https://api.arasaac.org/api/pictograms/es/search/'namePictoram'
[9] https://api.arasaac.org/api/pictograms/es/idPictogram

```
[
    {
        "_id": 8224,
        "created": "2009-02-20T13:23:06.000Z",
        "downloads": 0,
        "tags": [
            "character",
            "movie character",
            "cinema"
        ],
        "synsets": [],
        "sex": false,
        "lastUpdated": "2020-06-03T08:54:45.997Z",
        "schematic": false,
        "keywords": [
            {
                "keyword": "Spiderman",
                "type": 1,
                "meaning": "Spider-Man es un personaje de ficción, un
superhéroe creado por Stan Lee y Steve Ditko en agosto de 1962 para una
historieta aparecida en el número 15 de la revista Amazing Fantasy. Su
identidad secreta es Peter Parker un joven neoyorquino que adquiere
asombrosos poderes tras ser picado por una araña radiactiva.",
                "plural": "",
                "hasLocution": true
            }
        ],
        "desc": "",
        "categories": [
            "movie character"
        ],
        "violence": false,
        "hair": false,
        "skin": false,
        "aac": false,
        "aacColor": false,
        "score": 11
    }
]
```


Figure 2.7: ARASAAC API response for 'Spiderman'.

the API result from searching a pictogram with id 8224.

2.2.2 Communication based on pictograms

Communication via pictograms happens with the help of a board or a communication book. Figure 2.9 shows an ACC communication book, where the person points to the pictograms one by one to form a sentence. The complexity of the phrases with pictograms is limited, usually consisting only of subject, verb, and object. Often, only the most significant words are used, although ARASAAC has pictograms for determinatives and prepositions.

As it was explained above, in ARASAAC, the pictograms for verbs do not have conjugations. That means that no matter the tense, number, and person we want to construct the sentence for, we have to use the same pictogram - the one for the infinitive of the verb. In a phrase, past, present, and future are expressed by the pictograms for yesterday, today, and tomorrow respectively.

2.3 Pict2Text 1.0

As described previously, Pict2Text 1.0 is the base of our project. The first version of this project is a web application that allows the translation of

```
{
schematic: false,
sex: false,
violence: false,
aac: false,
aacColor: false,
skin: false,
hair: false,
downloads: 0,
categories:
[
"movie character"
],
synsets: [ ],
tags:
[
"character",
"movie character",
"cinema"
],
_id: 8224,
created: "2009-02-20T13:23:06.000Z",
lastUpdated: "2020-06-03T08:54:45.997Z",
keywords:
[
{
keyword: "Spiderman",
type: 1,
meaning: "Spider-Man es un personaje de ficción, un superhéroe creado
por Stan Lee y Steve Ditko en agosto de 1962 para una historieta
aparecida en el número 15 de la revista Amazing Fantasy. Su identidad
secreta es Peter Parker un joven neoyorquino que adquiere asombrosos
poderes tras ser picado por una araña radiactiva.",
plural: "",
hasLocution: true
}
],
desc: ""
}
```


Figure 2.8: Example of ARASAAC API response when called for pictogram id 8224.

Figure 2.9: Example of ACC communication book.

sentences written using pictograms to natural language (Spanish).

When entering the website[10] the user can see on the left part, the pictogram sentence panel, with a caption 'Pictograms' above it, and a button

[10] https://holstein.fdi.ucm.es/tfg-pict2text

'Traducir' below it. On the right part, an input box with the caption 'Nombre del picto', and a button 'Buscar' on the left of it. The user can write and search a specific word from the ARASAAC pictogram database and display it in a panel on the right part of the web page. After that, they can include the chosen one into the pictogram sentence panel, from where later the message is translated into natural language.

To search for a specific pictogram, the user should write the world they are looking for in the input box on the right side and click the button 'Buscar'. In Figure 2.10 it can be seen in the search for the word "Hombre".

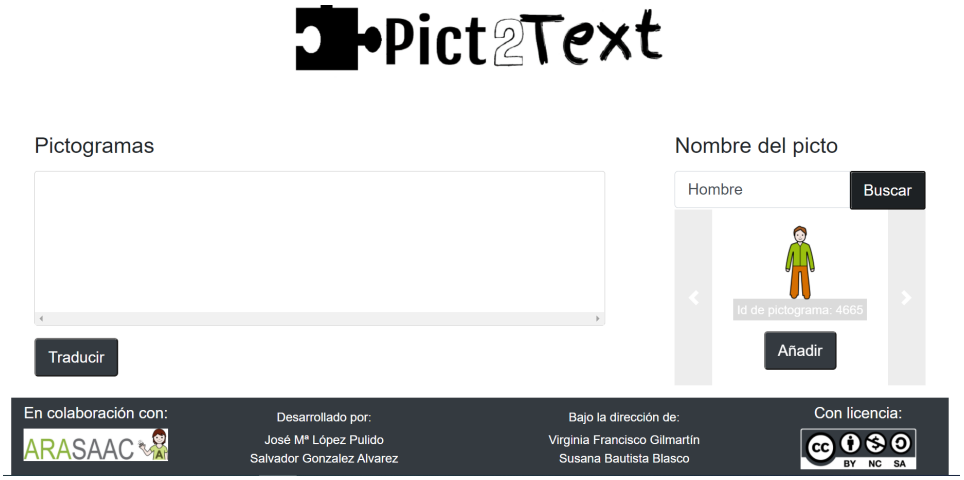

Figure 2.10: Searching for the word "Hombre" in PICT2Text 1.0.

After searching the pictogram, the user needs to include it in the left panel with pictograms. This is done by clicking the button "Añadir". In Figure 2.11, the pictogram corresponding to the word "Hombre" is included in the pictogram sentence panel.

The user can form a sentence by repeating the previous actions with other words. Figure 2.12 displays a translation of a sentence written with the pictograms corresponding to the words "Hombre", "Comer", "pizza". As we can see, the sentence is translated to "El hombre come una pizza" ("The man is eating a pizza").

2.3.1 Implementation

For the front-end of the project, Angular[11] was used. As the website itself is a SPA (Single-Page Applications), which needs to respond fast, a framework like Angular fulfills this performance requirement.

[11] https://angular.io/

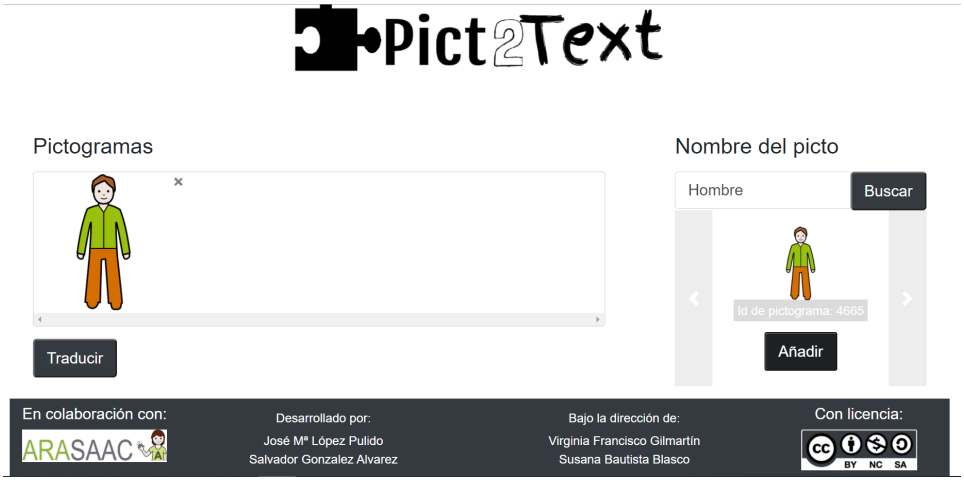

Figure 2.11: Adding the pictogram "Hombre" to the pictogram sentence panel.

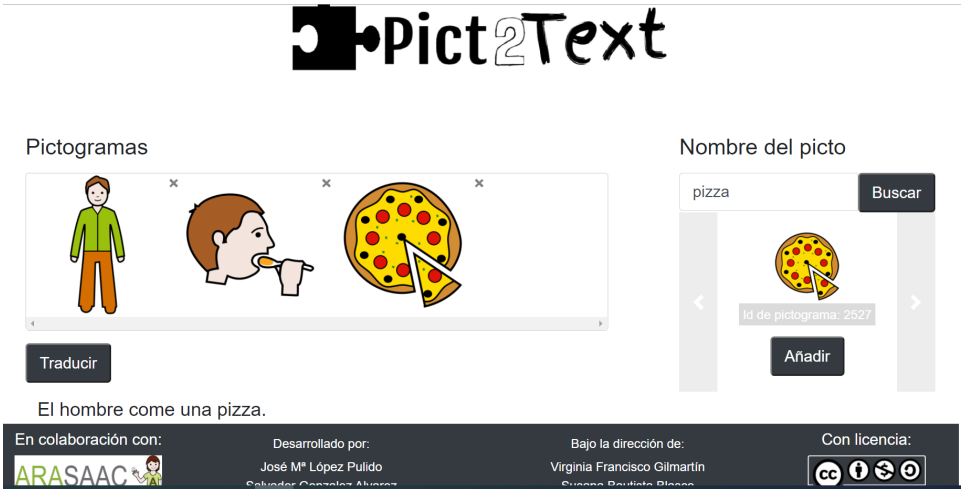

Figure 2.12: Translating the sentence "El hombre come una pizza."

The framework Django[12] was used for integration and intercommunication between the implemented web services.

The API of ARASAAC[13] provides the searching mechanism used to match words to pictograms, the graphical images of pictograms, and additional information about them.

When the pictograms are selected, Pict2Text 1.0 constructs the sentence

[12] https://www.djangoproject.com/
[13] https://arasaac.org/developers/api

by looking for the subject, verb, and proper tense, object, etcetera. To do that Spacy[14], a Python library for advanced natural language processing with high accuracy was used.

2.3.2 Conclusions

Although Pict2Text 1.0 translates messages with pictograms into natural language, it requires the user to manually select the pictograms they want to use in the construction of the sentence with pictograms. But constructing the sentence in this way is not possible for end-users of the application.

In addition, although Pict2Text 1.0 provides a good translation of simple sentences, having only one subject, verb, and object, some aspects should be improved in the translation to increase its coverage.

2.4 Machine Learning for image processing

To reach the first goal we have established in section 1.2, we need to process the image with the message written with pictograms uploaded by the user. The processing consists of two parts:

1. To detect how many pictograms are in the picture and their location.

2. To identify which ARASAAC pictogram corresponds to each of the pictograms detected on the image.

As the sequences of pictograms contain different elements and they could come from different sources, a black or whiteboard, a desk or a table, or else, we cannot assume that they will always have a specific background. That increases the complexity of the problem because we cannot pass through the pixels of the image and separate the pictograms based on the color of the background.

As the ARASAAC pictograms by default do not have a frame, a separation of the pictograms using it would not satisfy our use case. On the other hand, due to different light, angle, or colouring the image of the pictogram may appear different from the original digital version of it, which makes the comparison pixel by pixel impossible.

For those reasons, we need machine learning algorithms for each of the tasks described above (detection of pictograms, and the identification of each pictogram detected). The concept of machine learning and some possible algorithms will be explained in the following subsections.

[14] https://spacy.io/

2.4.1 Machine Learning

Machine learning (ML) (Mitchell, 1997) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of Artificial Intelligence. Machine learning algorithms build a model based on sample data, known as "training data", to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

A machine learning model[15] is a file that has been trained to recognize certain types of patterns. You train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data.

Once you have trained the model, you can use it to reason over data that it hasn't seen before and make predictions about that data.

Deep learning is one of the many machine learning algorithms, which can learn complex relationships and can solve problems like predictions and classification. For this, deep learning uses neural networks, which will be explained in the following subsection.

2.4.1.1 Neural Network

Artificial neural networks (ANNs) (Bhadeshia , 1999), usually called neural networks (NNs), are computing systems vaguely inspired by the biological neural networks that constitute animal brains.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear functions of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, as presented in Figure 2.13, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly after traversing the layers multiple times.

Supervised learning is learning through pre-labelled inputs, which act as targets. For each training example there will be a set of input values (vectors) and one or more associated designated output values. The goal of this form

[15] https://docs.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model

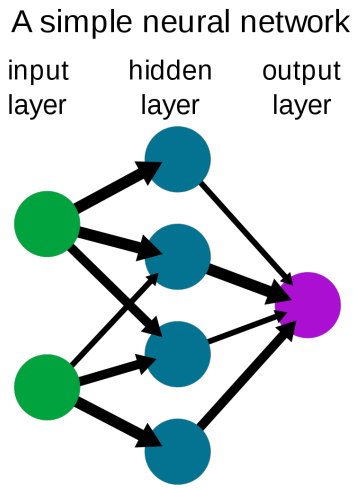

Figure 2.13: A simple neural network architecture.

of training is to reduce the models overall classification error, through correct calculation of the output value of training example by training.

Unsupervised learning differs in that the training set does not include any labels. Success is usually determined by whether the network can reduce or increase an associated cost function. However, it is important to note that most image-focused pattern-recognition tasks usually depend on classification using supervised learning (O'Shea and Ryan Nash, 2015).

One of the largest limitations of traditional forms of ANN is that they tend to struggle with the computational complexity required to compute image data. Common machine learning benchmarking datasets such as the MNIST[16] database of handwritten digits are suitable for most forms of ANN, due to its relatively small image dimensionality of just 28 x 28. With this dataset, a single neuron in the first hidden layer will contain 784 weights (28x28x1 where 1 bear in mind that MNIST is normalised to just black and white values), which is manageable for most forms of ANN. If you consider a more substantial coloured image input of 64 x 64, the number of weights on just a single neuron of the first layer increases substantially to 12.288 (64x64x3- Height, Width, RGB[17] channels). Also, it must be taken into account that to deal with this scale of input, the network will also need to be a lot larger than one used to classify colour-normalised MNIST digits, then you will understand the drawbacks of using such models.

[16] http://yann.lecun.com/exdb/mnist/
[17] https://en.wikipedia.org/wiki/RGB_color_model

2.4.1.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are analogous to traditional ANNs in that they are composed of neurons that self-optimize through learning. Each neuron will still receive input and perform an operation, such as a scalar product followed by a nonlinear function - the basis of countless ANNs. From the input raw image vectors to the final output of the class score, the entire network will still express a single perceptive score function (the weight). The last layer will contain loss functions associated with the classes.

One notable difference between CNNs and traditional ANNs is that CNNs are primarily used in the field of pattern recognition within images. This allows us to encode image-specific features into the architecture, making the network more suited for image-focused tasks - whilst further reducing the parameters required to set up the model.

One of the fundamental building blocks of a Convolutional Neural Network is the convolution operation[18], which is explained in the next subsection.

Convolution operation

In mathematics (in particular, functional analysis), convolution is a mathematical operation on two functions (f and g) that produces a third function that expresses how the shape of one is modified by the other. In computer vision, however, this term has a slightly different meaning: a linear operation that involves the multiplication of a set of weights with the input, much like in a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.

As we can see in Figure 2.14, the filter is smaller than the input data, and the type of multiplication is applied between a filter-sized patch of the input. The filter is a dot product - element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the 'scalar product'.

Using a filter smaller than the input is intentional as it allows the same filter (set of weights) to be multiplied by the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom. If the filter is designed to detect a specific type of feature in the input, then the application of that filter systematically across the entire input image allows the filter an opportunity to discover that feature anywhere in the image.

[18] https://en.wikipedia.org/wiki/Convolution

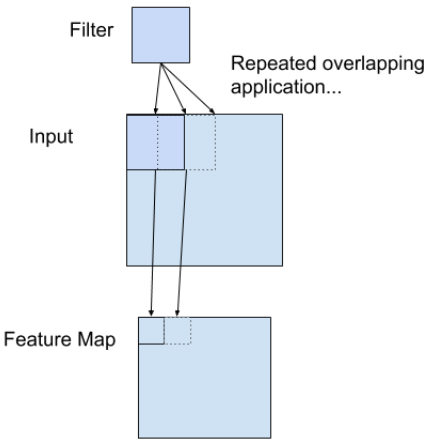

Figure 2.14: Filtering over the input image and constructing the feature map.

The output from multiplying the filter with the input array one time is a single value. As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent the filtering of the input. As such, the two-dimensional output array from this operation is called a 'feature map' (Brownlee, 2019).

In Figure 2.15 it can be observed a 3 by 3 filter for vertical edge detection applied to an image with height 6 and width 6 with stride 1 and no padding, and the calculations that need to be done in order to obtain the result for the first cell of the output.

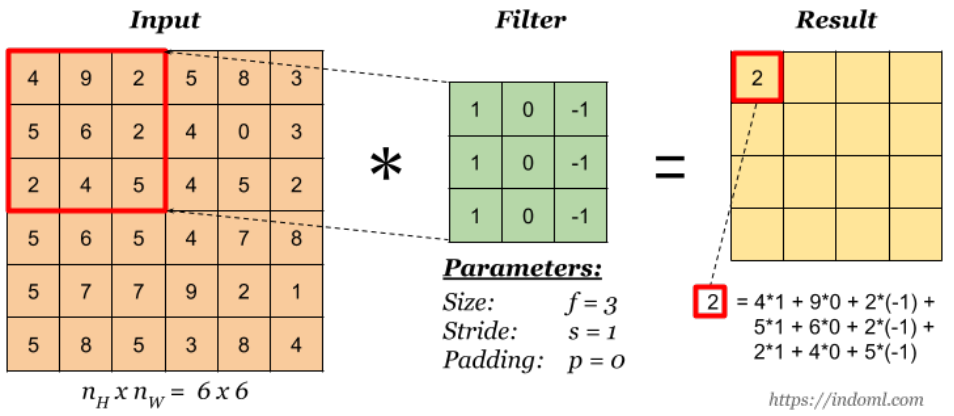

Figure 2.15: Filtering over an input image, representing the convolution operation in CNN.

Convolutional Neural Network Architecture

A typical CNN starts with the input image, sent to a network, formed by convolutional and pooling layers where the learning of the features of the image is performed. Later the final result of these is flattened, forming the input of the fully connected layers (neural network layers) with a classification (activation) function in the end. This way, the characteristics of the input image are detected in the beginning, and later, the picture is classified according to them. In Figure 2.16 a typical CNN architecture can be seen. The input image is converted into a matrix, where it goes through a series of convolution and pooling layers. The output is flattened and given as an input to a fully connected neural network that outputs the final result.

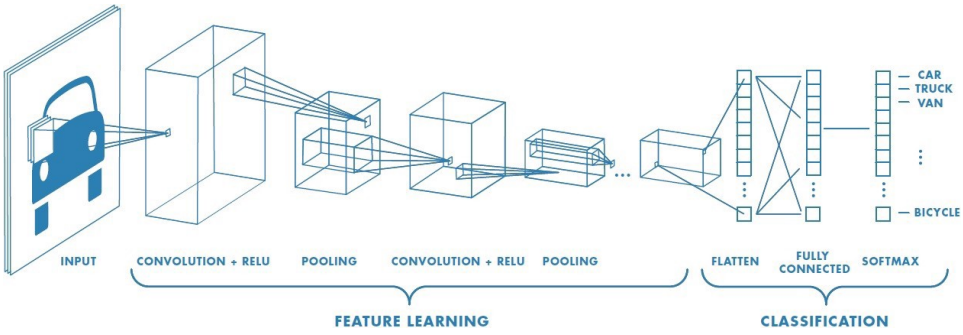

Figure 2.16: A convolutional neural network formed by several features layers followed by classifications.

This type of Neural Network works best with images, although it can be applied to any type of data. For that reason, CNN is widely used in object detection, object localization, and image classification algorithms, as will be explained in the following subsection.

2.4.2 Image processing

Nowadays, image processing is needed in a variety of different areas such as medical visualization, gaming, self-driving cars, and many more. To fulfill those needs, a completely new field of machine learning emerges - computer vision. Computer vision tasks include analysis, image restoration, pattern recognition, etcetera. In that group of algorithms are also: image detection, image localization, and object detection, which are of interest for our project.

In Figure 2.17 you can see the difference between object detection, instance segmentation, object localization, and image classification. First is the image classification, where you get a single object, such as an image, as an input, and the algorithm labels it. Object localization locates the presence of objects in an image and indicates their location with a bounding box.

Object detection locates the presence of an object, puts a bounding box around it, and labels that object. One further extension of object detection is object segmentation, also called "object instance segmentation" or "semantic segmentation", where instances of recognized objects are indicated by highlighting the specific pixels of the object instead of a coarse bounding box.

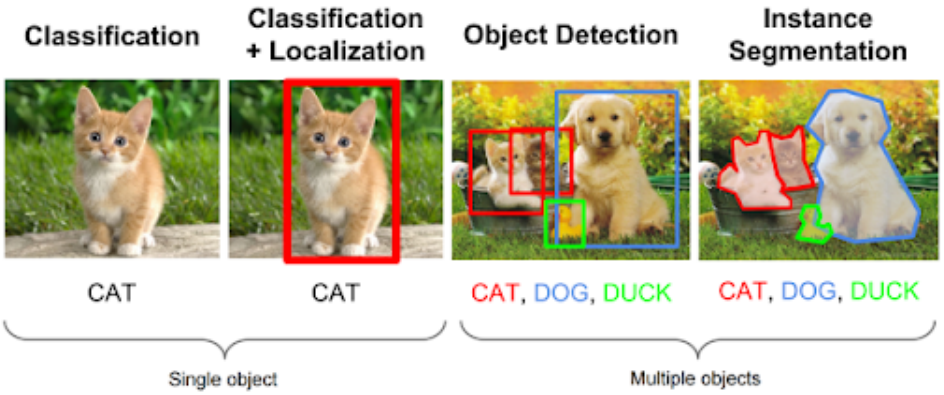

Figure 2.17: Difference between image classification, object localization, object detection and instance segmentation.

Object localization[19] is usually used to localize one object in a picture. Since we will have one or more pictograms in a picture, this algorithm doesn't work for us. We can't use an object detection algorithm to identify the classes because those algorithms are designed to work with a few classes and need a lot of data representing each one. In our case, we have more than 12.000 classes (the pictograms) and only one example of each. Therefore, to solve the problems, we need the following two machine learning algorithms:

1. An object detection algorithm to detect the pictograms in an image.
2. An image classification algorithm that predicts which class the pictogram belongs to.

In the following sections, we will explain how each of those two problems can be faced.

2.4.3 Object detection algorithm

Object detection (Brownlee, 2019) is a technology related to computer vision and image processing that deals with detecting instances of objects of a certain class (humans, buildings, cars, animals) in digital images or videos.

[19] https://towardsdatascience.com/object-localization-in-overfeat-5bb2f7328b62

Every object class has its special features that help in classifying the class (for example, all circles are round). Object class detection uses these special features when looking for a specific class (for example, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed). A similar approach is used for face identification where eyes, nose, and lips can be found and features like skin color and distance between eyes.

Object detection methods generally are separated into two types: neural network-based and non-neural approaches. For the second type, it is necessary to define features using a method like: Viola-Jones object detection framework (Viola and Jones, 2001A) and (Viola and Jones, 2001B) based on Haar features[20], Scale-invariant feature transform (SIFT) (Lowe, 1999) or Histogram of oriented gradients (HOG)[21], and then using a technique like a support vector machine (SVM) (Cortes and Vapnik, 1995). The neural network-based methods are able to do end-to-end object detection without the need to specifically define the features. They are typically based on convolutional neural networks. Common neural network approaches for object detections are Region Proposals (R-CNN (Girshick, Donahue, Darrell, Malik and Berkeley, 2014), Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren, He, Girshick and Sun, 2016), cascade R-CNN (Cai and Vasconcelos, 2017)), Single Shot MultiBox Detector(SSD) (Liu, Anguelov, Erhan, Szegedy, Reed, Fu and Berg, 2016), You Only Look Once (YOLO), Single-Shot Refinement Neural Network for Object Detection (RefineDet) (Zhang, Wen, Bian, Lei and Li, 2018), Retina-Net[22] and Deformable convolutional networks (Dai, Qi, Xiong, Li, Zhang, Hu and Wei, 2017).

In our project, we need a fast algorithm that can detect generalized objects (as our pictograms). For those reasons, we have chosen the YOLO algorithm, explained in continuation, since it suits our needs perfectly.

2.4.3.1 YOLO algorithm

You Only Look Once (YOLO) (Redmon, Divvala, Girshick and Farhadi, 2016) is a popular family of object detection models. The approach involves a single neural network trained end to end that takes a photograph as input and predicts bounding boxes and class labels for each bounding box directly.

YOLO achieves high accuracy while also being able to run in real-time. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes.

[20] https://en.wikipedia.org/wiki/Haar-like_feature
[21] https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
[22] https://developers.arcgis.com/python/guide/how-retinanet-works/

In Figure 2.18 it can be seen the flow of execution of the YOLO model. Starting from the input image, which is divided into a grid of S by S dimensions. Every cell of the grid predicts two bounding boxes. The class probabilities map and the bounding boxes with confidences are then combined into a final set of bounding boxes and the class labels.

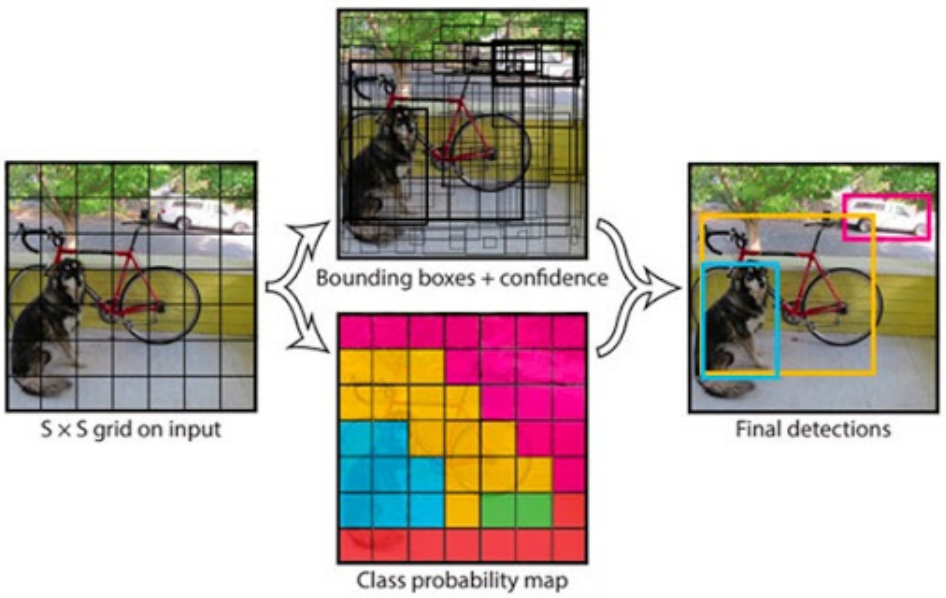

Figure 2.18: Summary of Predictions made by YOLO Model (Redmon, Divvala, Girshick and Farhadi, 2016).

With YOLO, a single CNN simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This model has several benefits over other object detection methods:

- It is extremely fast.
- It sees the entire image during training and test time, so it implicitly encodes contextual information about classes as well as their appearance.
- It learns generalizable representations of objects so that when trained on natural images and tested on the artwork, the algorithm outperforms other top detection methods.

Further research (Redmon and Farhadi, 2016) provided several improvements to the YOLO detection method including the detection of over 9.000 object categories by jointly optimizing detection and classification.

In (Redmon and Farhadi, 2018) progress with evolving YOLO is presented, with code available on a GitHub repository. In pursuit of better results for YOLOv3, the team tried many different ideas, but many of them did not work. A few of the modifications worth mentioning are a new network for performing feature extraction consisting of 53 convolutional layers, a new detection metric, predicting an "objectness" score for each bounding box using logistic regression, and using binary cross-entropy loss for the class predictions during training. The final result is that YOLOv3 runs significantly faster than other detection methods with comparable performance. In addition, YOLO no longer struggles with small objects.

The main implementation of Redmon's YOLO is based on Darknet[23], which is an open-source neural network framework written in C and CUDA. Darknet sets the underlying architecture of the network and is used as the framework for training YOLO. This implementation is fast, easy to install, and supports CPU and GPU computation.

Although the original creator of YOLO Redmon withdrew from the project, it was not its end, and in April 2020, a new version was released. The 4th generation of YOLO has been introduced in (Bochkovskiy, Wang and Liao, 2020).

YOLO v4 takes influence from BoF (bag of freebies) and several BoS (bag of specials). BoF improves detection accuracy without increasing inference time. They only increase the training cost. On the other hand, BoS increases the inference cost by a small amount. However, they significantly improve the accuracy of object detection.

YOLO v4 also based on the Darknet and has obtained an AP (Average Precision)[24] value of 43.5%t on the COCO dataset along with a real-time speed of 65 FPS on the Tesla V100, beating the fastest and most accurate detectors in terms of both speed and accuracy. The YOLO v4 has been considered the fastest, and most accurate real-time model for object detection.

When compared with YOLO v3, the AP and FPS have increased by 10 percent and 12 percent, respectively. In Figure 2.19 it is shown a comparison between YOLO v3 and v4 models where could be seen that the Average precision and FPS of version 4 is higher than the one from version 3.

2.4.4 Image classification algorithm

Image classification algorithms[25] consist of labeling an image into one of a fixed set of categories. For example, we can build an image classification algorithm to recognize various objects, such as cars, pedestrians, and signs. Some of the challenges from a computer vision perspective that we will face

[23] https://github.com/pjreddie/darknet

[24] https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

[25] https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

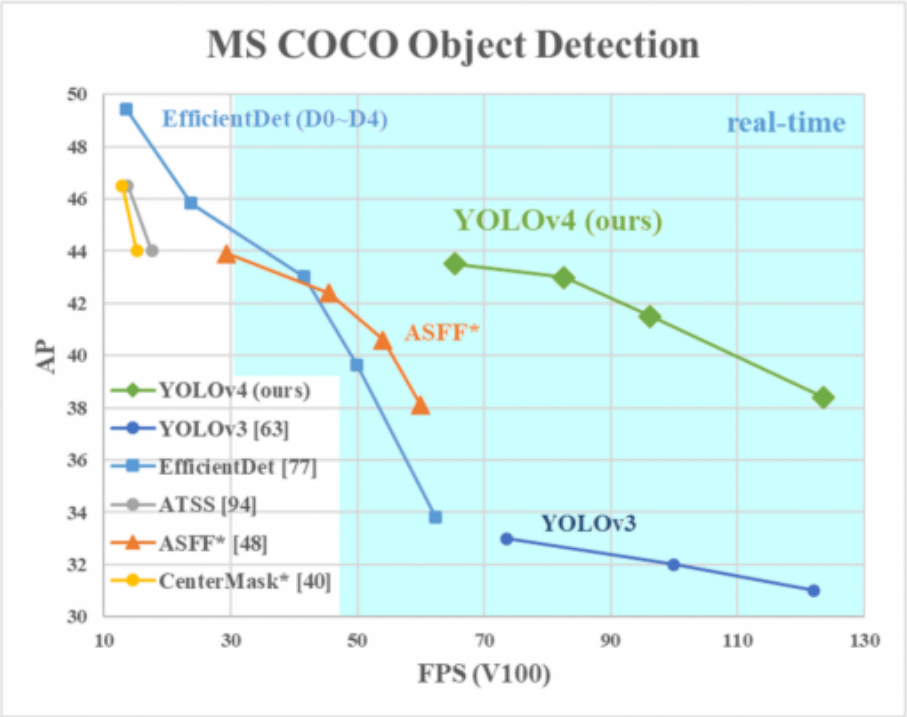

Figure 2.19: The speed and accuracy of YOLO v4[24].

in our project are viewpoint variation, scale variation, illumination conditions. Usually, those algorithms, as most neural network algorithms, require a large amount of training data to work well. Since, in our case, we don't have that much data, only one example per class, we decided to look at algorithms that can be trained even with a significantly smaller dataset. One of those algorithms is the One-shot learning algorithm using the Siamese neural network, which we will present next.

2.4.4.1 Siamese Neural Network

Usually, neural networks learn to predict a certain amount of classes. In that case, if we add a new class, we have to retrain the neural network with the new classes and the new data. Also, the traditional neural network requires a large volume of data to train on. On the other hand, the Siamese Neural Networks learns how to find similarity functions[26], patterns that bring classes together, and that enables us to add new classes without training the model again.

[26]https://en.wikipedia.org/wiki/Similarity_measure

In Figure 2.20 we can see the architecture of a Siamese neural network. It consists of two identical twin subnetworks joined at their outputs. The subnetworks have the same configuration with the same parameters and weights. We feed each input that we want to compare into one of the identical models, usually composed of various convolutional layers. Each model outputs an n-dimensional embedding where the dimensions represent a pattern found in the input. Those embeddings are given to the similarity function. This is a real-valued function that quantifies the similarity between two objects by their feature vectors[27] (vectors that represent numeric or symbolic characteristics in a way that is easier to analyze). In the end, the siamese neural network returns a similarity score that represents the level of similarity between the inputs. The smaller the similarity score is, the more similar the pictures are.

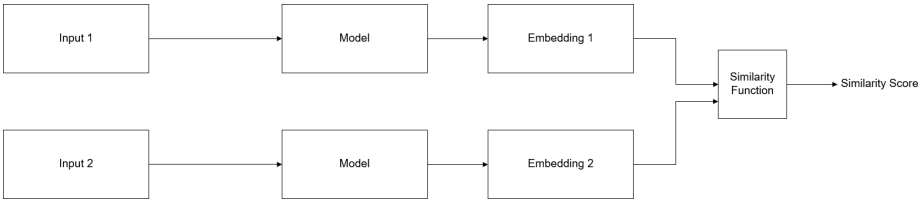

Figure 2.20: Architecture of the Siamese Neural Network.

On the other hand, this type of Neural Network requires much more training which takes more time than the traditional networks.

In the following section, we will explain SNN based algorithm: One-shot learning algorithm and implementation of this algorithm.

2.4.4.2 One-shot learning algorithm

One-shot learning[28] categorizes objects with one, or only a few, training examples. After the algorithm is trained, it takes two inputs and returns a value that shows the similarity between the two pictures. If the two inputs are similar, the algorithm returns a value smaller than a specific threshold, for example, 0.001, and if they are not the same object, the returned value is higher than the threshold. To do this the algorithm uses the Siamese neural network.

During the training phase one-shot learning trains to be able to measure the distance between the features in two inputs. To achieve that, the algorithm uses a function called triplet loss. This function trains the network giving it 3 inputs: an anchor, a positive example, and a negative one. The network should adjust its parameters in a way that the feature values for the

[27] https://brilliant.org/wiki/feature-vector/
[28] https://en.wikipedia.org/wiki/One-shot_learning

anchor and the positive example are very close, while those of the anchor and negative example are very different.

An implementation for one-shot learning is the one presented in (Lamba, 2019). The problem that this implementation solves is letter recognition. The authors use the Omniglot dataset[29], which consists of 1.623 characters from 50 alphabets. The images are of handwritten characters in grayscale with 105x105 resolution, with only 20 examples of each of the 1.623 classes.

The problem is a supervised learning task[30], a function that maps an input to an output based on given input-output pair examples, where we have pairs of (Xi, Yi). Xi itself is a randomly generated pair of two images of letters, and Yi is a value between 0 and 1, indicating the similarity of the two examples given in Xi. Yi is 1 if the two images are similar and 0 if they are not.

Each of the two subnetworks consists of a convolution layer and three MaxPooling layers, each with a ReLU as an activation function. The output of the third MaxPooling layer is flattened and given to the last layer - a regular densely connected NN layer with sigmoid as an activation function.

The model uses Adam optimizer (Kingma and Lei Ba, 2015), an algorithm for first-order gradient-based optimization, binary cross-entropy loss function[31], and low learning rate - 0,00006. The model was trained with 20.000 iterations and used 32 pairs of images per iteration.

2.5 Tools

In this section, we will explain the tools we have used to develop our project. In section 2.5.1 we will explain one of the most used machine learning platforms: TensorFlow. In section 2.5.2 we will explain Keras, an open-source Python library that works as an interface for the TensorFlow library. In section 2.5.3 we will present LabelImg, the labeling tool we used to annotate our images. Finally, Flask and GitHub Actions will be presented in sections 2.5.4 and 2.5.5 respectively.

2.5.1 TensorFlow

TensorFlow[32] is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

[29] https://github.com/brendenlake/omniglot

[30] https://en.wikipedia.org/wiki/Supervised_learning

[31] https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[32] https://www.tensorflow.org/

TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization to conduct machine learning and deep learning research. The system is general enough to be applicable in a wide variety of other domains, as well.

2.5.2 Keras

Keras[33] is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with the aim of enabling fast experimentation.

Keras is the high-level API of TensorFlow 2: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration speed.

The data preprocessing module is one of the essential modules in Keras. It provides different utilities for image, time-series, and text preprocessing. The image preprocessing submodule includes the following functions:

- image_dataset_from_directory. It provides the functionality to load in memory images from a specific directory. In addition to that, it can automatically generate the labels of the images, change their size, separate the loaded images into batches or in train and validation sets.
- load_img. It provides the functionality to load a single image in memory. It also allows the change of the size of the image, among other things.
- img_to_array: converts a PIL Image (Pillow[34] library format) instance to a NumPy array. Depending on the image color channels, the returned array will be 2D for grayscale, 3D for RGB, or 4D for RGB-A.

Also, Keras provides the ImageDataGenerator class, which allows the users to perform image augmentation on the fly. The class contains three methods: flow(), flow_from_directory() and flow_from_dataframe() to read images from NumPy arrays and folders.

2.5.3 LabelImg

LabelImg [35] is a graphical image annotation tool. It is written in Python and uses Qt[36] for its graphical interface. It permits manual labeling of images,

[33] https://keras.io/
[34] https://pillow.readthedocs.io/en/stable/
[35] https://github.com/tzutalin/labelImg
[36] https://www.qt.io/qt-for-python

which by default are saved as XML files in PASCAL VOC format (a format used by ImageNet[37]). LabelImg also supports YOLO and CreateML formats. Figure 2.21 shows the labeling of an object "person" from an image of a football game.

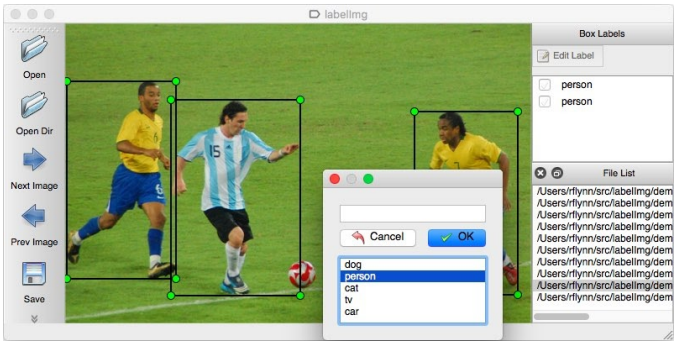

Figure 2.21: Labeling an object from class "person" from an image of a football game using the LabelImg tool.

2.5.4 Flask

Flask[38] is one of the most popular Python web application frameworks. It is lightweight, open-source, and it is designed to start quickly and easily, although it can scale to complex applications. The framework provides various tools (object-relational mappers, open authentication systems, etcetera) as well as REST[39] API support, and allows you to use multiple databases.

Representational state transfer (REST) (Fielding, 2000) is a software architecture style for providing standards between computer systems on the web. It's used to create interactive applications that use web services. A web service that follows these guidelines is called RESTful (Richardson and Ruby, 2007). REST-compliant systems are stateless (the server does not need to know anything about what state the client is in and vice versa) and separate the concerns of client and server.

2.5.5 GitHub Actions

GitHub Actions[40] is an API that allows you to automate, customize, and execute your software development workflows (a configurable automated process made up of one or more jobs) in your GitHub repository. The API allows

[37] https://image-net.org/
[38] https://flask.palletsprojects.com/en/2.0.x/
[39] https://www.codecademy.com/articles/what-is-rest
[40] https://github.com/features/actions

you to perform any job you want, including continuous integration and continuous deployment. You can also combine actions while GitHub manages the execution, providing feedback and secures all the steps.

Chapter 3

Software development methodology

Software development methodologies (Javanmard and Alian, 2015) are used to provide better team performance and to get better results. The two main types of software development methodologies are traditional and agile. Traditional methodologies, also known as heavyweight methodologies, are predictive and used to follow a linear approach. They require defining and documenting all the requirements at the beginning of the process. Agile methodologies are adaptive and open to change, which makes them more flexible. Some of the most popular agile methodologies are Scrum, Kanban, and Extreme Programming (XP). For this project, we have decided to use the agile framework Kanban. In the following sections, we will explain it, why we have selected it, how we have applied it, and what tests we have made.

3.1 Kanban

Kanban is an agile methodology oriented towards visualizing the work, making it flow, reducing waste, and maximizing the product value. The main rules that Kanban follows are visualization, usually via dashboards, limit the work in progress (WIP), which reduces the number of open tasks, and pull value through the system - a task is started only when it's needed.

We chose Kanban as the methodology for our project because the product is delivered continuously, and changes are allowed during the whole process, and no estimation of the tasks is needed. That will increase our flexibility and productivity.

Every few weeks, we have meetings with the tutors. During every meeting, the tutors will give us the tasks - code and memory- that we have to finish until the next meeting, including their priority. The date for the next meeting also will be decided during the current one.

We created a dashboard[1] to visualize the tasks we have. Tasks can be of two types: coding tasks or report tasks. We decided to have the following columns on our board:

- **To do.** Tasks given to us by the tutors are ordered according to the priority given by the tutors to each task (from higher to lower priority).
- **In progress.** Tasks we are currently working on. All tasks in that column will have a particular person assigned. As we are using Kanban, we have limited the WIP (Work In Progress) of the column to 4 to avoid doing more tasks than we can reasonably manage. The completed tasks will be moved to 'Testing'.
- **On hold.** Activities that we can't continue at the moment. The reason behind this is that they are waiting for another task to finish. When the task could be continued, it is returned to the column 'In progress'.
- **Testing.** In the case of a coding task, the testing will include code review and automated and/or manual testing (this will be explained in the following section). In the case of a report task, the person who didn't write the particular part will read and correct it. If a problem is spotted, the task will be moved to the column 'In progress'. If not, it will go to the column 'Ready for review'.
- **Ready for review.** Tasks we have finished but are not yet reviewed by our tutors. The tasks in that column will be reviewed during the next meeting and the tutors will decide which of them will be moved to the column "Done" and which ones are not finished and must be taken up again.
- **Done.** Tasks that are finished, tested, reviewed, and approved by the tutors.

If the WIP of any of the columns does not allow more tasks, any additional ones will be moved to the column "Hold on". In Figure 3.1 you can see an example of our board with the columns created and some tasks assigned.

3.2 Testing

To test Pict2Text 2.0 we have two main groups of tests: machine learning tests and the API tests.

The machine learning tests verify the correctness of the two machine learning models we have implemented: pictogram detection and pictogram recognition. To validate the algorithms, we created some ad-hoc tests. Mainly

[1] https://pic2text2.atlassian.net/jira/software/projects/PICT2TEXT/boards/1

Figure 3.1: Kanban board used for the Pict2Text 2.0 project.

we compared the expected results from the algorithms with the returned outputs.

For our project, we decided to implement continuous deployment using GitHub Actions, as will be explained in continuation.

3.3 Continuous Deployment

Continuous Integration, Continuous Deployment, and Continuous Delivery are commonly used terms in modern development practices and DevOps[2].

Continuous Integration (CI)[3] (Booch, 1991) is a practice in software engineering where members of a team integrate their work to a shared mainline (the base of a project) frequently. The term was first proposed by Grady Booch in 1991, later the idea was adopted by Extreme programming (XP). CI is supposed to work with various automated tests, as those are run before committing to the mainline. Apart from the tests, organizations usually use a build server to apply quality control in general. The build server compiles the code periodically or with every commit and sends a report to the developers.

Continuous Delivery (CD)[4] is an extension of continuous integration with automatic deployment of the changes to a wanted environment. On top of the automated testing, there is an automated release process from the source code repository to the test or/and production environment by clicking one button.

Continuous Deployment (CD)[5] is a software release process where every change is validated by automated testing and, if it passes all stages of your

[2] https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[3] https://martinfowler.com/articles/continuousIntegration.html

[4] https://www.atlassian.com/continuous-delivery

[5] https://www.atlassian.com/continuous-delivery/continuous-deployment

production pipeline, is released to the customers. The whole deployment method is automated, so only failed tests can prevent a new change to be deployed to a production environment.

To integrate and deploy as fast as possible all changes we make to the API, we create a Continuous Deployment pipeline.

Initially, we wanted to use a configuration like Jenkins-Nomad-Docker for the CI/CD, but as the provided environment from the university was a Linux container, we were not able to install and use Docker. Despite this, we created a CD pipeline using GitHub Actions, execution scripts, and system service.

When exiting the running virtual container, it closes the current terminal session and stops the execution of our application. As we needed the API running continuously without interruption, we had to run it either as a job in the background, a separate session using tmux[6] (an open-source terminal multiplexer for Unix-like operating systems), or screen[7] (a full-screen window manager that multiplexes a physical terminal between several processes), or as a system service. Initially, we started with a tmux session but as it was giving problems using it in the Github Actions, we changed to a system service.

In Appendix A you can observe the Continuous Deployment configuration we are using to deploy our API. In it, we had specified the name of the workflow, when it will be executed (when a push event on the master branch is realized), and the jobs and steps the pipeline will do: connect to the virtual container, pull the latest changes of the repository, and restart the "pict2text2" system service.

To define our "pict2text2" service, we create the configuration file pict2text2.service in the directory /etc/systemd/system in the container located in holstein.fdi.ucm.es server.

In the service section, we have defined the execution of the script "run.sh", which starts the API from the directory of the project. In Figure 3.2 is shown the system service configuration file for Pict2Text 2.0 API.

```
[Unit]
Description=Pict2Text2.0 flask API
After=network.target

[Service]
User=gnazer
WorkingDirectory=/home/gnazer/TFG/TFG-2021-Pict2Text2.0/API
ExecStart=/home/gnazer/TFG/TFG-2021-Pict2Text2.0/API/run.sh
Restart=always

[Install]
WantedBy=multi-user.target
```

Figure 3.2: System service configuration file for Pict2Text 2.0 API.

[6] https://github.com/tmux/tmux/wiki
[7] https://www.gnu.org/software/screen/

Chapter 4

Pict2Text 2.0

As we have already explained in section 1.2, one of the goals of our project is recognizing pictograms from an uploaded picture. To achieve this, we have to identify the different pictograms in an image, and then classify each one of them. For that, we need two Machine Learning algorithms: one to detect the pictograms that appear in an image (section 4.1), and another to identify the word associated with each of the pictograms detected by the previous algorithm (section 4.2). To connect the two models, we implemented an API, presented in section 4.3. A web application was also implemented (section 4.4), which serves as a visual representation of the obtained results.

4.1 Pictograms detection using YOLO Algorithm

Detecting how many pictograms are in a picture and their location is the first issue we have to solve if we want to determine which pictograms compose the message in the image. Therefore we have to separate the individual pictograms in the mage.

To solve this problem, we need to have a machine learning model able to locate where are these pictograms in the image and separate them using a bounding box. In Figure 4.1 is shown an image of a sentence written with pictograms meaning "The boy takes out a toothbrush and a toothpaste". In it, each pictogram is separated using a bounding box. The given image is an example of the expected output of our machine learning model. The model selected for this purpose is YOLO (described in section 2.4.3.1).

To understand how to configure the algorithm and to create the required dataset, we have followed two tutorials [1,2]. In them, the authors describe step

[1] https://youtu.be/hTCmL3S4Obw
[2] https://youtu.be/mmj3nxGT2YQ

Figure 4.1: The sentence "The boy takes out a toothbrush and a toothpaste", written with pictograms. The pictograms are separated using a bounding box.

by step how to use the Google Colaboratory[3] to execute machine learning models in Google Cloud Platform (GCP)[4] using the provided GPUs. They show which are the required configuration files, how to modify them correctly, and how the model should be trained and tested, providing all relevant code. Additionally, in the first tutorial, the author covers how to label the data correctly to the required format for both versions 3 and 4 of YOLO.

YOLO, similarly to other machine learning models, requires its data to be labeled in a specific format (YOLO labeling format[5]). To prepare the dataset in this format, a .txt file with the same name is created for each image file in the same directory. Each .txt file contains the annotations for the corresponding image file, that is object class, object coordinates, width, and height as follows: *<object-class> <x> <y> <width> <height>*:

- <object-class> - integer number of object from 0 to number of classes - 1
- <x> <y> - represent the center of the bounding box.
- <width> <height> - float values relative to width and height of image, in the range (0.0 to 1.0]

In the .txt file corresponding to a certain image an annotation line is created for each object tagged on that image. An example for two objects from two different classes will be as follows:

[3] https://colab.research.google.com/notebooks/intro.ipynb
[4] https://cloud.google.com/
[5] https://towardsdatascience.com/image-data-labelling-and-annotation-everything-you-need-to-know-86ede6c684b1

- 0 0.45 0.55 0.29 0.67
- 1 0.99 0.83 0.28 0.44

In the given example, the first value represents the object class to which the object belongs. Where 0 are the labeled objects from the first class, and 1 those from the second. The second value represents the center of the label bounding box relative to the x-axis, the third, the center of the bounding box relative to the y-axis, and the last two values represent the width and height of the image relative to the total width and height.

Our training set consists of pictures of a single pictogram, with a variation of the scale and viewpoint. To label the data, we have used the graphical image annotation tool "LabelImg"[6]. For the manual preparation of our dataset to the required format, we have followed the steps from the tutorial[7] shown in Figure 4.2. The first steps are to choose the directory where the images we want to label are located and to point to the folder where the labels are going to be saved. Next, we have to pick the YOLO annotation format, fourth to select an image to annotate, fifth to draw a bounding box around the object we want to label. Finally, we have to indicate the name of the object class.

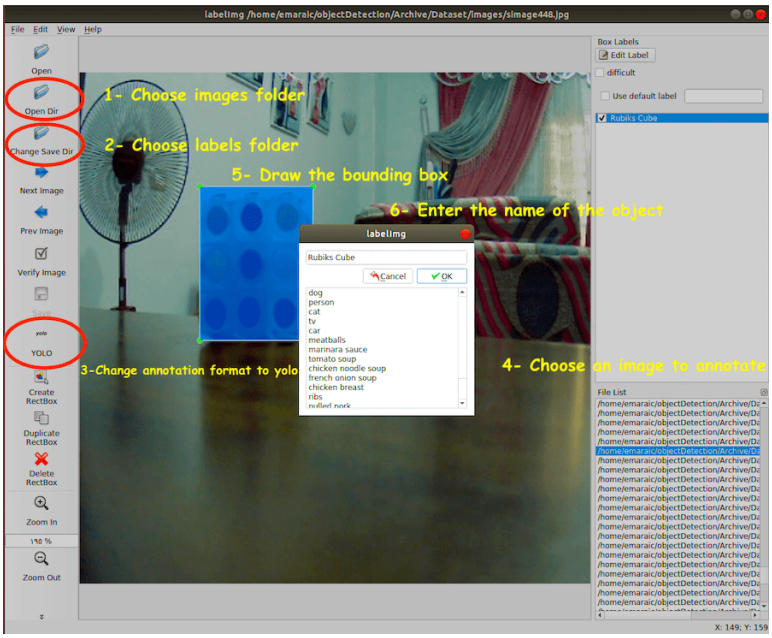

Figure 4.2: Annotation process using the LabelImg tool.

An example for labeling an image from our dataset using the LabelImg

[6] https://github.com/tzutalin/labelImg

[7] https://colab.research.google.com/drive/1_GdoqCJWXsChrOiY8sZMr_zbr_fH-0Fg?usp=sharingscrollTo=8dfPY2h39m-T

tool could be seen in Figure 4.3. After the labeling, the .txt file with the annotations in the YOLO format contains *0 0.499319 0.498152 0.955041 0.977088*, representing the object-class, the coordinates relative to the x-axis and y-axis, and the width and height of the bounding box respectively.

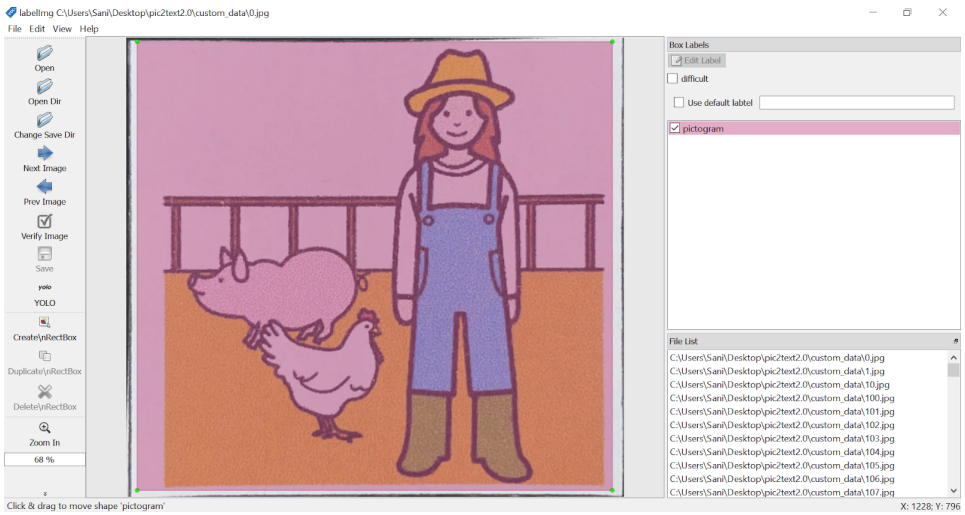

Figure 4.3: Labeling using LabelImg for an image of our dataset.

The training set consists of pictures of pictograms, which require manual tagging, locating, and creating bounding boxes around the pictograms in the image. Our dataset consists of thousands of different classes (the pictograms) and only one example per class. For that reason, we are not going to use the classification part of YOLO, instead, we will detect where are the pictograms in the image but not which pictograms they are. For that reason, after we label our training examples, we will always have 0 as a first value - the object class (for example *0 0.499319 0.498152 0.955041 0.977088*).

In the following subsections, we will explain different versions of YOLO we have made to solve our problem. The way we will evaluate the performance of those versions will be by focusing on the bounding boxes the algorithm predicts given an image with pictograms, as well as the probability percentage (in the upper left corner of the bounding box).

4.1.1 First Version

The main goal of this version was to configure and run the YOLO model with 150 pictures of pictograms labeled in the YOLO format. We cloned the Google Colaboratory Notebook[8] from the tutorial[2], connected it to our

[8] https://colab.research.google.com/drive/1_GdoqCJWXsChrOiY8sZMr_zbr_fH-0Fg?usp=sharingscrollTo=8dfPY2h39m-T

Google Drive, and started modifying it so that we could train the YOLO v4 with our custom data.

4.1.1.1 Configuration

To configure the model, we followed the steps from the notebook, cloning the darknet repository[9] and changing the Makefile of the same repository. We used Google Colaboratory to train our model on GPU instead of CPU, to reduce the time to train the model. After that, we built the darknet so we could use the darknet executable file to run or train object detectors.

Next, we compressed the directory, containing the pictures of pictograms, their labels, a text file with information regarding the name of the class used by YOLO to classify the object as pictograms, and two Python scripts, and we uploaded it to Google Drive. One of the uploaded scripts creates the "train.txt" and "test.txt" files which contain the training and testing examples separated in the proportion 85% for training and 15% for testing. The other one contains the number of classes the model will use, the location of the training and validation sets, the names of the classes taken from the .txt file, and the backup directory where the model will save the weights of every 1.000 iterations.

After decompressing the directory in Google Drive and executing the two scripts, we modified the YOLO layers configuration file to adapt it to our needs. We changed the "batch" size from 64 to 2 (as we do not have thousands of examples to train with), we set "max_batches" to 6.000, and "steps" to "4.800, 5.400". Finally, we changed the filters to 18 in the three convolutional layers and the number of classes to 1 in the three YOLO layers. The parameters were configured, using the following formulas:

max_batches = (# of classes) * 2.000 (but no less than 6000 so if training for 1, 2, or 3 classes it will be 6.000, however detector for five classes would have max_batches=10.000)

steps = (80% of max_batches), (90% of max_batches) (so if max_batches = 10.000, then steps = 8.000, 9.000)

filters = (# of classes + 5) * 3 (so if training for one class then the filters = 18, but if you are training for 4 classes then the filters = 27)

Next, we downloaded pre-trained weights, located in the darknet repository[10], directly to the virtual machine where we have placed the darknet cloned project and all of the configuration we made. Although transfer learning and using pre-trained model weights is not mandatory, it reduces training time, increases accuracy, and makes the model converge faster to a good so-

[9] https://github.com/AlexeyAB/darknet

[10] https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137

lution.

Although the model was using an external GPU (Tesla P100-PCIe-16GB), provided by Google Colaboratory, as this is a deep multi-layers CNN doing a lot of iterations, it required several hours to train. After 2.000 iterations, the model already has achieved a map of 100, and we have stopped its execution to test whether or not it was able to detect the pictograms in a picture of pictograms.

To test the model, we needed to create a new YOLO layers configuration file, get the weights from the backup folder, provide the model with the image we want to detect pictograms from, and specify the bounding box threshold (used to discard all detections under a certain value). For threshold we used 0.2, meaning that only detections higher than 20% will be shown in the output. The results from the tests will be presented in the following subsection.

4.1.1.2 Results

To test the YOLO model we have previously trained, we created three test groups. The examples in them were not used in the training set.

- The first one consisted of 3 examples of a single pictogram in an image. In Figure 4.4 can be seen that the YOLO model correctly detected the pictogram with 90% accuracy.

Figure 4.4: The YOLO model implemented by us detecting a single pictogram from an image of pictogram with an accuracy of 90%.

The following two examples we tested (see Figure 4.5) contained pictures of pictograms zoomed out. For these examples, the model again correctly localized the pictograms in the images, but its probability has dropped to 66%. The dropped accuracy from zoomed-in and zoomed-out images of pictograms is an indicator that maybe the model was not

trained with enough zoom-out examples, so at the moment, it does not know how to detect zoomed-out images, as well as zoomed in.

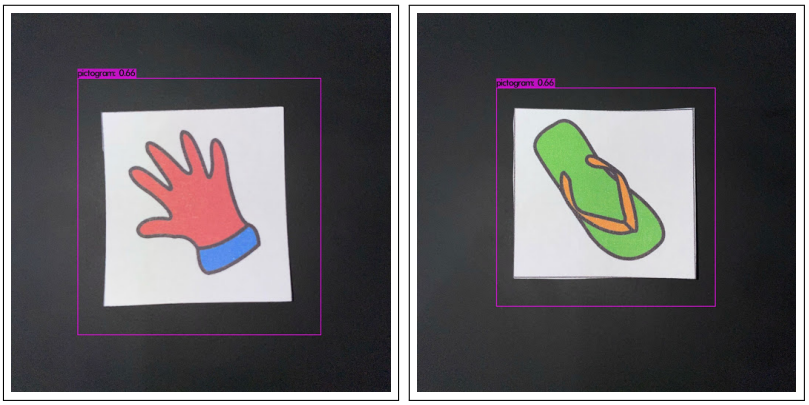

Figure 4.5: Two zoom-out images of pictograms detected with an accuracy of 66% from YOLO.

- The second testing group consists of 2 examples of images with two pictograms on them. In Figure 4.6 is shown the output of the YOLO model tested with two pictograms. YOLO managed to detect the left pictogram (the girl with red hair) as a pictogram with 14% and both pictograms combined as another pictogram with 90%.

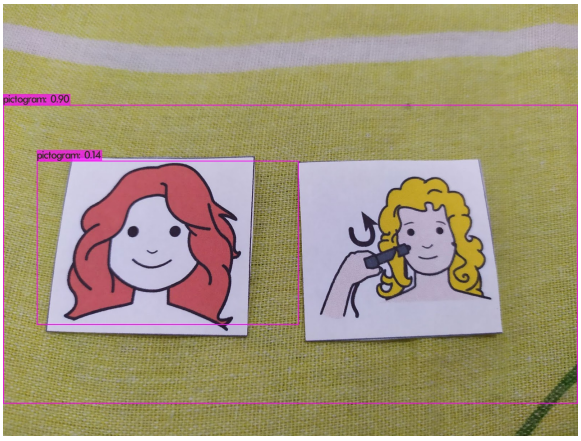

Figure 4.6: Testing YOLO with an image with two pictograms. Correctly detecting one of the pictograms but incorrect result in general.

In this example, the model incorrectly detected both pictograms as a single pictogram, although it managed to localize one of the pictograms separately, even with a small percentage.

With the picture in Figure 4.7 the result given by YOLO is better: it detected the left pictogram (the girl with red hair) as a pictogram with 66% and again both of the pictograms as a single pictogram with 32%.

Figure 4.7: Testing YOLO with an image with two pictograms zoomed in. Correctly detecting one of the pictograms but incorrect result in general.

From the examples of this group, once again, the results indicate that there could be a potential problem with the detection related to viewpoint and scale variation. Also, from these tests, it is clear that the model is not working as expected when there are multiple pictograms in the same image.

- The third test group contains two examples of images with three pictograms on them. In the example of Figure 4.8 the model detected the three pictograms as a single one with 92%, which is incorrect.

Figure 4.8: Testing YOLO with an image with three pictograms. Incorrectly detecting all of them as a single pictogram.

Figure 4.9 shows the result from the YOLO model tested with the zoomed-in version of the image with three pictograms. It managed to detect the left pictogram (the girl with red hair) as a single pictogram

with 94%, but the following detection contains all of them as a single pictogram which is incorrect.

Figure 4.9: Testing YOLO with an image with three pictograms zoomed-in. Incorrectly detecting all of them as a single pictogram but correctly localizing the left pictogram (the girl with red hair).

As can be seen in Table 4.1, from the tested examples, we can conclude that the first version of the YOLO model shows good results with a single pictogram in an image when the image is zoomed in and worse results in zoomed-out. When testing with two and three pictograms, the model incorrectly groups all of the images, and detects them as a single pictogram in all of the cases. Also, in the tests with multiple pictograms in an image, the accuracy problem when the image is zoomed-out exists, too. Better results are obtained when the image is zoomed in.

#pictograms in the picture	Accuracy	Prediction
Single pictogram (zoomed-in)	90%	Correct
Single pictogram (zoomed-out)	66%	Correct
Two pictograms (zoomed-in)	66%\32%	One of the pictograms\Both as one pictogram. Incorrect
Two pictograms (zoomed-out)	14%\90%	One of the pictograms\Both as one pictogram. Incorrect
Three pictograms (zoomed-in)	94%	All as a single pictogram. Incorrect
Three pictograms (zoomed-out)	92%	All as a single pictogram. Incorrect

Table 4.1: Results from version one of our YOLO model.

To solve the above-described problems, in the following versions of the algorithm, it should be trained with a dataset with more augmented pictograms.

4.1.2 Second Version

In our first version of the YOLO model, the following problems were detected:

- The model incorrectly groups multiple pictograms and outputs a bounding box for all the pictograms in the picture instead of one for each pictogram.
- The scale variation of the image affects the model accuracy.

To solve the above-described problems, we augmented our training dataset.

4.1.2.1 Augmentation of the training dataset

For the augmentation of the images, we decided to start with a simple 45-degree rotation to the left, since that doesn't imply taking new pictures of pictograms, as it would if we have tried to directly solve our scale variation problem. For that, we create a python script using the PIL library. In Figure 4.10 is shown a picture of a pictogram rotated 45-degree to the left using our python script. The script goes through the images of the picture of pictograms from a given directory, rotates them, and saves them in the same folder. That way, we double the original set to 300 images. Later we labeled the augmented pictures using LabelImg. Next, we retrained the model with the new dataset.

Figure 4.10: 45-degree rotation to the left of a picture of a pictogram using our python script.

4.1.2.2 Results

Testing the model using the same three testing groups as in the first version of our model, we obtained the following results:

- First group (3 examples of a single pictogram in an image). As can be seen in Figure 4.11, the model correctly detected the pictogram with 99% accuracy. For this image compared against the output of the previous version, the accuracy is increased by 9%.

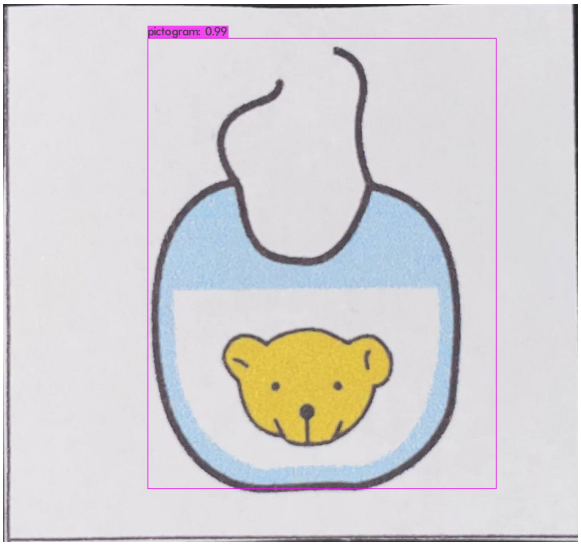

Figure 4.11: The second version of the model detecting a single pictogram from an image of a pictogram with an accuracy of 99%.

For the next two examples, which contained pictures of pictograms zoomed out (see in Figure 4.12), the model correctly localized the pictograms in the images, the accuracy also is high above 90%. There are multiple bounding boxes returned by the model, but the ones with the lower accuracy should not appear. Comparing the output from the previous model and taking into consideration only the bounding box with the highest accuracy from the current output, the accuracy has increased from 66% to 90-97%.

Figure 4.12: Two zoom-out images of pictograms detected with accuracy over 90% from the second version of our model.

- Second testing group (2 examples of images with two pictograms on them). In this case (Figure 4.13), the model incorrectly detected both pictograms as a single pictogram. Although with high accuracy of 99%, the model was not able to separate the pictograms. Comparing the output from the previous version, for the concrete example, the previous version was returning better results. In the previous version, the model was able to detect the pictogram of the girl, and in this version, it is not.

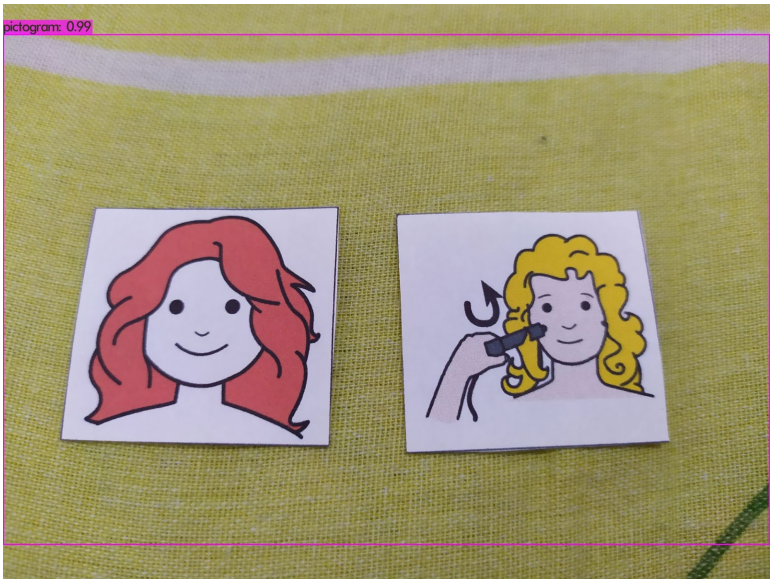

Figure 4.13: Testing the second version of the model with an image with two pictograms. Incorrectly detecting both pictograms as a single one.

Testing the new version of the model with the zoomed-in image of two pictograms, the model correctly detected both of the pictograms. In Figure 4.14 can be seen that the pictogram with the girl with red hair was detected as a pictogram with 97% and the other pictogram with 98%. Those results are significantly better than the results for the previous version of the model.

In the last example of this group for the first time, the model correctly localized multiple pictograms in a given image. This is a good indication that even though the model is making mistakes, the second version of the model increases the model capabilities.

- Third test group (two examples of images with three pictograms on them). In the example of Figure 4.15, the model detected the three pictograms as a single one with 100%, which is incorrect. In comparison with the previous version of the model, the accuracy is increased.

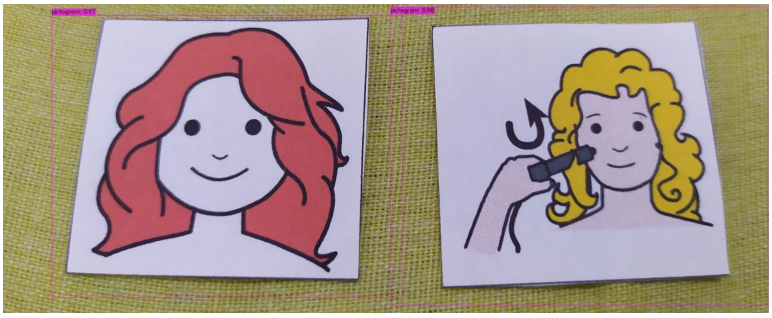

Figure 4.14: Testing the second version of the model with an image with two pictograms zoomed in.

Figure 4.15: Testing the second version of the YOLO model with an image with three pictograms. Incorrectly detecting all of them as a single pictogram.

In Figure 4.16 is shown the result from the new model tested with the zoomed-in version of the image with three pictograms. The model returned four bounding boxes, correctly detecting the girl with the red hair with 59% and the backpack with 10%. The other two bounding boxes group multiple pictograms, and they are incorrect. Comparing the current results with those for the previous version, the localization of a second pictogram from the images is a sign of permanence progression of the model.

Figure 4.16: Testing the second version of the YOLO model with an image with three pictograms zoomed-in. Detecting the two pictograms from the extremums.

4.1.2.3 Conclusions

In the second version of our model, as can be observed in Table 4.2 we managed to increase the model performance by augmenting the dataset. For the examples with multiple pictograms in an image, the model was able to correctly detect more than one pictogram. The problems with the grouping of multiple pictograms and detecting it as a single one, and the other related to the scale variation of the image persists. The decision to increase the dataset was a correct one, even though the model still makes a lot of mistakes.

#pictograms in the picture	Accuracy	Prediction
Single pictogram (zoomed-in)	99%	Correct
Single pictogram (zoomed-out)	90%	Correct
Two pictograms (zoomed-in)	97% and 98%	Detected pictogram one and pictogram two. Correct
Two pictograms (zoomed-out)	99%	All as a single pictogram. Incorrect
Three pictograms (zoomed-in)	59% and 10%	Detected pictogram one and pictogram three. Incorrect
Three pictograms (zoomed-out)	100%	All as a single pictogram. Incorrect

Table 4.2: Results from version two of our YOLO model.

Having the above-described problems, the next versions of the model should be trained with a dataset containing images with scale variation, different rotations, and other augmentations, taken in multiple backgrounds.

4.2 Pictogram identification using One-shot learning algorithm

Once the pictograms from a picture are detected, we have to identify the word associated with each one of those pictograms. Considering the particularity of the dataset, having more than 12.000 classes (the pictograms) and only one example of each, we decided to use the One-shot learning algorithm (explained in section 2.4.4.2).

For the implementation of the algorithm, we have taken an already implemented one from (Lamba, 2019) (presented in section 2.4.4.2). The original

algorithm worked with a dataset consisting of images of letters of different alphabets. Those letters were in grayscale, and our pictograms are in PNG format. As shown in Figure 4.17, in machine learning, a picture in grayscale is represented by a two-dimensional matrix, with dimensions the width and the height of the picture. Each cell corresponds to 1 pixel and contains a number between 0 and 255, representing the shade of grey where 0 is black, and 255 is white. On the other hand, as you can see in Figure 4.18, the colorful images need three matrices for the channels red, green, and blue. In our case, we have a fourth matrix representing the alpha channel (the opacity of the picture), as it is included in the PNG images we use. For that reason, we also have a fourth matrix, which is included in the PNG images to represent the opacity of the pictures (alpha channel).

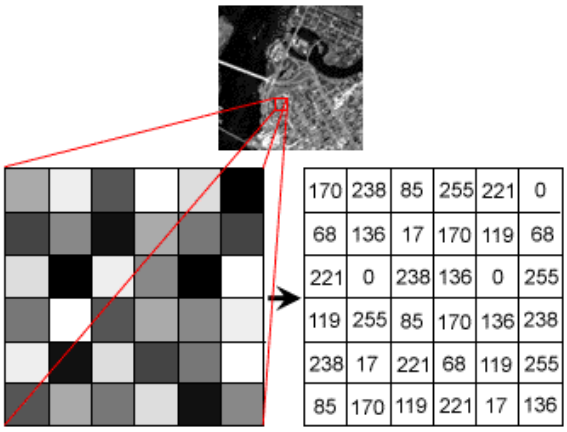

Figure 4.17: Image represented in grayscale image and the corresponding matrix.

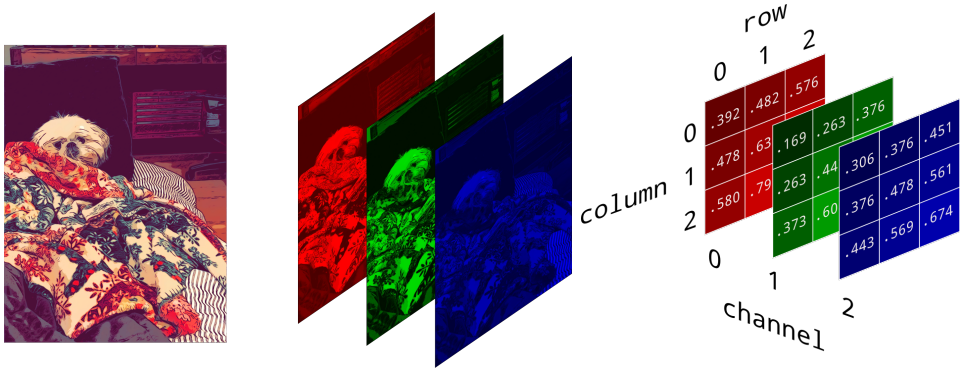

Figure 4.18: Colourful RGB image with its RGB three matrices.

To train and test the model, we have used three data sets:

- **Training set.** Used to train the model and achieve the weights we will need to recognize given pictograms, later when we ask the model to predict.

- **Validation set.** Used to compare the accuracy of the algorithm on different data in order to tune the hyperparameters, like the number of iterations.

- **Test set.** Used to provide an unbiased evaluation of the model, simulating a near-real situation where the model would face pictograms never seen by it before. We use it to compute how similar a given pictogram is to all other pictograms forming the set.

The objective is to have all pictograms of the ARASAAC dataset in the test set so that we could compare a given pictogram against all others.

In the following sections we will explain the preparation of the training set, and the versions in the development and testing we have done.

4.2.1 Preparing the dataset

To prepare the training, validation, and test sets we have used, as well as to fetch the ARASAAC pictograms, we have implemented a few scripts presented in the following subsections.

4.2.1.1 Loading ARASAAC pictograms

First of all, we needed to obtain the ARASAAC pictograms dataset in Spanish to be able to train our model. We created a Python script that uses a REST client to call the ARASAAC API[7]. The script makes a request to the ARASAAC API[11], which returns the information of all ARASAAC pictograms in Spanish. Later it downloads every ARASAAC pictogram calling the ARASAAC service[9] that given a pictogram id returns a JSON with the information regarding it.

The fetched pictograms are stored locally in separate folders numbered from 0 to n, with a maximum size of 4.000 images. We have chosen this number of elements because in a Unix (Linux) based system some commands like rm, cp, move, etcetera place the files as parameters of the functions, and the maximum number of parameters given to a function is limited to 4.096. Restricting the number of images to 4.000 ensures that all common commands will work correctly.

As the process of downloading the pictograms required more than 3 hours for the complete fetching of over 11.000 images, we included concurrency and reduced the execution time to under an hour.

[11] https://api.arasaac.org/api/pictograms/all/es

Having the dataset fetched, we had to load it in memory in a way that the model could use it. We have implemented another Python script to achieve that. In it, we have used the image preprocessing module of Keras (explained in section 2.5.2) to read the pictograms from a directory and store them in memory. All ARASAAC pictograms are in the image format png, and during loading, we had to configure the module of Keras to load them as four-color channels (RGBA[12]) and resize them to 105 height by 105 width.

4.2.1.2 Processing ARASAAC pictograms

When the user uploads a picture of a pictogram, likely, it will not be the same as the original online version. It could be rotated, the colours may not be the same, it could be blurred, etcetera. For this reason, the dataset obtained in the previous step must be processed to augment the dataset generating different representatives of each one of the pictograms. To achieve that, we had created three processing scripts. All of them use Keras preprocessing module, but they differ in the way they manipulate the pictograms. Keras's module includes the class ImageDataGenerator (explained in section 2.5.2), which comes with different image manipulation options and augmentation capabilities like changing the brightness of an image, rotating it, zooming it, and so on. In the following sections, we will explain the manipulation scripts we have developed.

Changing brightness

The first script changes the brightness of each pictogram. We provided a range between 0.2 and 1.0 to the ImageDataGenerator, specifying that the augmented image brightness should take a random value in that range. The two extremes represent a very dark and very bright image. Figure 4.19 shows the original pictogram of a bee ('abeja') and Figure 4.20 shows processed images of the pictogram bee ('abeja') using our brightness changing script.

This type of augmentation will help the model to detect the concept represented in the pictogram independently of the light or brightness conditions from the provided image.

Rotating images

The second script rotates the pictogram randomly. Similar to the brightness augmentation script, it used the ImageDataGenerator class, specifying the rotation_range attribute to 90 degrees. This specification will augment the provided pictogram generating random 90 degrees rotations. In Figure 4.21 are shown two images of the bee pictogram generated by the rotating script.

[12] https://en.wikipedia.org/wiki/RGBA_color_model

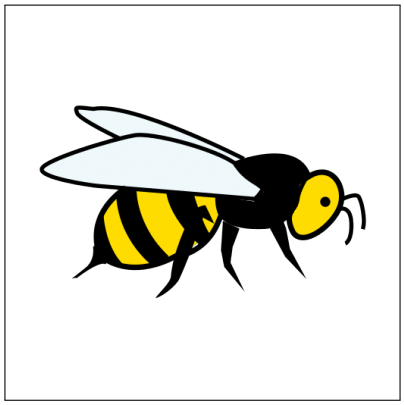

Figure 4.19: The original image of the pictogram bee(‘abeja‘) from the ARASAAC.

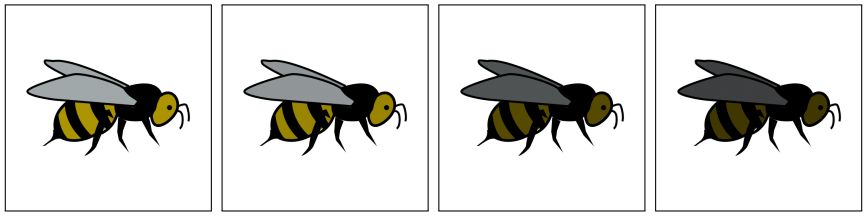

Figure 4.20: Augmented images of the pictogram bee (‘abeja‘) using the brightness changing script.

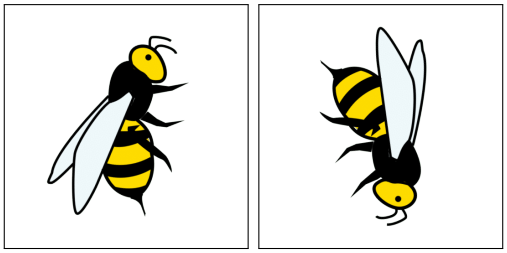

Figure 4.21: Two 90 degree rotations of the pictogram bee (‘abeja‘) generated by the augmentation script.

This type of augmentation will help the model to recognize the pictogram even if the provided image of it is rotated or tilted aside.

Changing colours

The third augmentation script changes the color of the pictogram. The script iterates through the width and height of the given pictogram and

changes randomly the RGB channels of it. The script does not change the alpha channel of the image as we would like to keep the original background. In Figure 4.22 are shown four augmentations of the pictogram bee (‘abeja‘) using the color augmentation script.

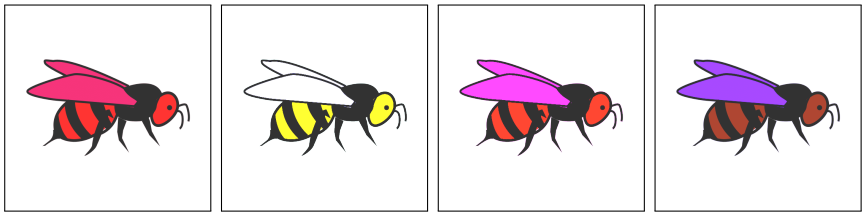

Figure 4.22: Four color augmented images of the pictogram bee (‘abeja‘) generated using the color augmentation script.

This type of augmentation will help the model to find the pictogram even if the provided image has different colors.

4.2.2 First Version

In the first version of the model we used the following datasets:

- **Training set.** We used 22 pictograms that we augmented by using the scripts described in section 4.2.1. After augmentation, our training set contained 440 images, 20 images for each of the original pictograms.
- **Validation set.** We used ten different pictograms that we augmented, as we did with the training set. After the augmentation, the validation set was composed of 200 images (20 images for each of the original pictograms).
- **Test set.** Composed of 100 digital pictograms.

Due to the little amount of data, the best performance was reached at ten iterations with 100% accuracy on the validation and test set. After that point, the model started overfitting the training data, and the accuracy of the validation set for 100 iterations dropped to 60% for the validation set and 17% for the test set.

In the first version of the One-shot model we constructed, we detected several problems:

- We were able to train it only with a small number of pictograms (22 original pictograms, which turned into 440 after the augmentation process) because we faced a problem loading more pictograms as we were not batching them during loading. As the model was trained

with this small number of pictograms it was overfitting. Although the non-realistic tests were showing good results, the realistic testing was affected by the overfitting problem, and its results were bad.

- Consumption of system memory: As we were not batching our dataset, we were loading a lot of images simultaneously, which reflected in high usage of memory, to the extent of which the machine we were using to train the model was not operational.

- Prediction time and pairing before the predictions step: to test the model we take a random image from the testing set and search for the most similar to it from the same data set. Although we tried a small testing set of 100 pictograms, the construction of the pairs was taking a lot of time. Also, as it was not compared against all other pictograms from the whole dataset, it was not a realistic representation of the actual requirement of the algorithm (the objective of the model is to detect a pictogram from a given image of a pictogram). The results obtained during this version you can see in Table 4.3.

	Training set		Validation set			Test set		
#iterations	#examples	Type of data	#examples	Type of data	Accuracy	#examples	Type of data	Accuracy
10	440	Augmented pictograms	200	Augmented pictograms	100%	100	Original pictograms	100%
100	440	Augmented pictograms	200	Augmented pictograms	60%	100	Original pictograms	17%

Table 4.3: Results from version one of the pictogram identification algorithm.

4.2.3 Second Version

In the second version of the algorithm, we aimed to fix the problems from its previous implementation. One of the biggest issues we faced in the first version was related to the way we were loading the pictograms which cause memory capacity problems. We were not able to train the model enough so it was overfitting and it returned bad results. As a result that, the first thing we did in the second version was loading the images in batches.

To load the images in batches we used the method flow_from_directory from the ImageDataGenerator class of Keras. The method takes as an input the directory name (where the augmented pictograms are situated), the size of the pictograms (we specified size of 105 widths by 105 height), the color channels (in our case RGBA), the batch size (we selected to have batches of 3 pictograms which augmented turns on 60 images) and returns a DirectoryIterator which contains the loaded images separated in batches and other relevant information. The above-described way of creating a DirectoryIterator with the batches of images was used to load the train, validation, and testing sets.

By loading the images in batches we needed to adapt our previous implementation of the algorithm. We changed the function that creates positive (the two examples in the pair are from the same class) and negative pairs (the examples belong to two different classes), which is used in the training of the algorithm. Also, we needed to modify the function for mapping a pictogram to all other pictograms used in the prediction of the model.

Additionally, we added a new function to display all pictograms of a currently loaded batch to visually verify the loaded images. In the training of the model, the only change we made in the new version is calling next of the DirectoryIterator, on every training iteration, getting the next batch of pictograms of the training set.

In this version of the model, we continue using the previous testing mechanism. This test was showing 100% accuracy, correctly predicting all pictograms of the test set in the first version of the model. The summary of this version you can find in Table 4.4.

	Training set		Test set		
#iterations	#elements	Type of data	#elements	Type of data	Accuracy
100	440	Augmented pictograms	100	Original pictograms	100%

Table 4.4: Results from the second version of the pictogram identification algorithm.

This test is not enough to verify the accuracy of our model, mainly for the following reasons:

- The test set consisted of a sample of the original pictograms from ARASAAC. These pictograms had the same high quality, format, and characteristics as the training set used to train the model (the pictograms in the test and train sets were different but their domain distribution was the same) and they were not representing the real use case. The users of the application are going to import an image or a photo of a message written with pictograms to the system. If the user uploads a photo, it will never have the quality or the characteristics of the original pictograms used during the training. Therefore, the model will never predict correctly.

- The testing function we were using was randomly choosing a pictogram from the test set and it was comparing it against all other pictograms from the same set. In the real use case, the model should detect the ARASAAC pictogram from a photo of it. The model should be able to predict how similar a specific pictogram is to all others, not randomly taken from the test set.

4.2.4 Third Version

This version of the algorithm was centered around testing the algorithm with pictures of pictograms, not only with the original digital ones since the functionality we aim to give to the user is to upload a picture of pictograms to be translated.

Initially, we took ten pictures of pictograms and added them to the test set. Then, we retrained the model with 100 iterations and batches of 2 augmented pictograms (40 images), and the model managed to detect all pictograms from the original dataset, but from our pictures of the pictograms, it managed to correctly predict similarity of 10%. After examination of the wrongly predicted examples, we concluded that the similarity between the picture and the predicted pictogram was minimal. For that reason, next, we focused on increasing the accuracy of the second group. To do so, we decided to extend the number of pictures in the training set. We took 54 new pictures of randomly chosen pictograms (44 pictures of pictograms for the training set and 10 for the test set). Thus, in our new training set, we had 44 classes, each containing the original digital pictogram, 19 augmented digital pictograms, and the picture of it (924 in total).

First, we tried the algorithm with 15 pairs, and we obtained the best results with 30 iterations - the percentage of correctly predicted pictures of pictograms got up to 30%.

After that, we increased the number of pairs per iteration to 44, where we reached 40% accuracy for the images of pictograms with the same test set as before, both with 30 and 50 iterations. When we examined the results we noticed that when the algorithm wasn't predicting the right pictogram id, it was predicting pictograms either with a similar shape or a similar colour. For example, in Figure 4.23 can be observed the picture of a pictogram we are trying to predict on the left and the predicted one on the right. As you can see, the shape of the two pictograms is the same, since both of them mean "robar". In our case, this prediction is considered right, since our goal is to predict the word corresponding to a picture of a pictogram.

With the obtained results (Table 4.5), we concluded that the model is overfitting the training set.

		Training set		Test set		
#iterations	Batch size	#examples	Type of data	#examples	Type of data	Accuracy
30	15	924	Augmented pictures and pictograms	100	Original pictograms\Pictures of pictograms	100\30%
50	15	924	Augmented pictures and pictograms	100	Original pictograms\Pictures of pictograms	100\30%
30	44	924	Augmented pictures and pictograms	100	Original pictograms\Pictures of pictograms	100\40%
50	44	924	Augmented pictures and pictograms	100	Original pictograms\Pictures of pictograms	100\40%

Table 4.5: Results from the third version of the pictogram identification algorithm.

Figure 4.23: A picture given to the algorithm of a pictogram with id 8210, and the pictogram predicted by it (with id 8209).

4.2.5 Fourth Vesion

This version of the algorithm aimed to increase the accuracy obtained with pictures of pictograms. Our solution was to take more pictures of pictograms, in order to expand the number of pictures of pictograms in all three sets we have.

When we finished taking, cropping, and changing the names of the pictures into the format "<id>-<meaning>.png" (a format we use to know which pictograms have the same id or name for the training and accuracy calculation respectively), we decided to change the three datasets. We increased the number of different pictograms in the training set from 44 to 111. We augmented the pictures of the pictograms (rotate them, and change their colour and brightness). In the end, for each pictogram in the training set, we had 21 examples: the original pictogram, the picture of the pictogram, and 19 examples of the picture augmented. In total, the number of training examples was 2.331.

As we had 151 images of pictograms in total, and we left 111 for the training set, for the validation and test sets, we had 20 pictures of pictograms in each. To check the accuracy of the validation and test set we compare each of the 20 images with a hundred digital pictograms, including the 20 we want to recognize.

We removed the digital pictograms and the augmented digital pictograms from all sets because we wanted to focus on receiving good results on the pictures of pictograms (since this is the bigger challenge regarding this new functionality we want to add to Pict2Text 1.0).

Each time we trained the algorithm we mainly concentrated on three things: loss (a function showing if predictions derivate too much from the actual results), the accuracy achieved from the validation set, and the accu-

racy on the test set. For each of the following model training, we increment the number of iterations to 2.000. As we were training the algorithm, we were validating and saving the weights for every 50 iterations.

First, we trained the algorithm with the new training set, without changing any parameters, except the number of iterations, as mentioned before. Even though we had more data, the algorithm performed similarly as before: we achieved a maximum of 30% accuracy on the validation set and 35% on the test set with different weights. We got the best performance from 600 iterations, where we had 30% accuracy both on the validation and the test sets. The loss on that iteration was 1.124. In Table 4.6 you can observe the data we obtained from the training for the weights with which we got more than 20% accuracy on the validation set.

#iterations	Learning rate	Batch size	Accuracy validation set	Accuracy test set	Loss
50	0.00006	44	25%	20%	3.625
150	0.00006	44	20%	35%	2.625
400	0.00006	44	25%	20%	1.459
450	0.00006	44	25%	30%	1.417
600	0.00006	44	30%	30%	1.124
850	0.00006	44	20%	20%	0.850
900	0.00006	44	30%	25%	0.911
950	0.00006	44	25%	25%	0.862
1.000	0.00006	44	25%	20%	0.774
1.050	0.00006	44	20%	35%	0.651
1.100	0.00006	44	30%	15%	0.681
1.350	0.00006	44	20%	15%	0.596
1.400	0.00006	44	25%	25%	0.593
1.750	0.00006	44	30%	25%	0.5699
1.800	0.00006	44	25%	20%	0.545
1.900	0.00006	44	20%	20%	0.40037
1.950	0.00006	44	25%	25%	0.479

Table 4.6: Results from the first training in version four of the pictogram identification algorithm.

The second time we trained the algorithm, we changed the learning rate from 0.00006 to 0.0006. The performance of the algorithm decreased drastically. The highest accuracy it hit was 5% on the validation set. For that reason, we decided to not even try it on the test set. Table 4.7 represents the data obtained where the validation set accuracy was higher than 0%. We didn't try the obtained weights on the test set because we expected similar results.

For the next training, we went back to a 0.00006 learning rate, and we increased the number of examples it takes for each iteration (batch size) of the training from 40 to 80. With this change, we achieved a minimum of 40% on each iteration over the 950th iteration. The validation set accuracy increased up to 60% various times with 850, 1.100, 1.200, and 1.750. The

#iterations	Learning rate	Batch size	Accuracy validation set	Loss
700	0.0006	44	5%	3.625
750	0.0006	44	5%	2.625
1.500	0.0006	44	5%	1.459
1.550	0.0006	44	5%	1.417
1.900	0.0006	44	5%	1.124
1.950	0.0006	44	5%	0.850

Table 4.7: Results from the second training in version four of the pictogram identification algorithm.

best performance we got with 1.750 iterations where the test set accuracy was also 60%, and the loss dropped to 0.4892. At this training, the algorithm performed better. In Table 4.8 we show the data for the weights performing more than 40% accuracy on the validation set.

#iterations	Learning rate	Batch size	Accuracy validation set	Accuracy test set	Loss
650	0.00006	80	45%	35%	1.124
750	0.00006	80	40%	30%	0.850
850	0.00006	80	60%	50%	0.850
950	0.00006	80	55%	45%	0.862
1.000	0.00006	80	55%	35%	0.774
1.050	0.00006	80	40%	25%	0.651
1.100	0.00006	80	60%	45%	0.681
1.150	0.00006	80	55%	50%	0.651
1.200	0.00006	80	60%	45%	0.681
1.250	0.00006	80	50%	45%	0.596
1.300	0.00006	80	45%	50%	0.681
1.350	0.00006	80	55%	55%	0.596
1.400	0.00006	80	50%	40%	0.593
1.450	0.00006	80	40%	40%	0.651
1.500	0.00006	80	50%	40%	0.681
1.550	0.00006	80	45%	50%	0.596
1.600	0.00006	80	50%	55%	0.681
1.650	0.00006	80	45%	50%	0.596
1.700	0.00006	80	50%	55%	0.681
1.750	0.00006	80	45%	50%	0.5699
1.800	0.00006	80	45%	45%	0.545
1.850	0.00006	80	60%	60%	0.479
1.900	0.00006	80	55%	60%	0.40037
1.950	0.00006	80	55%	50%	0.479
2.000	0.00006	80	55%	60%	0.40037

Table 4.8: Results from the third training in version four of the pictogram identification algorithm.

For the fourth and last test, we increased the number of examples for each iteration from 80 to 111 (all the classes we have in the training set).

As expected, based on the previous training, the accuracy increased up to 75% on the validation set. To check the accuracy of the test set, we took the weights that achieved over 40% on the validation set, as can be seen in Table 4.9. The best performance until now we achieved from 1.850 iterations, where we got 65% accuracy on the validation set and 75% on the test set.

#iterations	Learning rate	Batch size	Accuracy validation set	Accuracy test set	Loss
150	0.00006	111	60%	40%	2.4545
500	0.00006	111	40%	35%	1.2385
600	0.00006	111	60%	35%	0.9836
650	0.00006	111	50%	40%	1.0229
700	0.00006	111	45%	60%	0.9769
750	0.00006	111	40%	50%	0.8192
800	0.00006	111	50%	40%	0.8441
850	0.00006	111	65%	45%	0.7795
900	0.00006	111	55%	45%	0.7832
950	0.00006	111	75%	60%	0.7516
1.000	0.00006	111	70%	55%	0.6713
1.050	0.00006	111	65%	60%	0.70989
1.100	0.00006	111	70%	50%	0.6544
1.150	0.00006	111	70%	50%	0.6544
1.200	0.00006	111	75%	55%	0.5825
1.250	0.00006	111	50%	55%	0.6150
1.300	0.00006	111	70%	55%	0.5847
1.350	0.00006	111	55%	55%	0.5436
1.400	0.00006	111	55%	50%	0.5351
1.450	0.00006	111	45%	60%	0.5195
1.500	0.00006	111	50%	50%	0.517888
1.550	0.00006	111	65%	65%	0.5171
1.600	0.00006	111	70%	55%	0.4691
1.650	0.00006	111	60%	55%	0.4758
1.700	0.00006	111	55%	60%	0.4864
1.750	0.00006	111	65%	55%	0.5699
1.800	0.00006	111	55%	50%	0.4557
1.850	0.00006	111	65%	75%	0.4557
1.900	0.00006	111	70%	50%	0.42737
1.950	0.00006	111	65%	60%	0.40606
2.000	0.00006	111	65%	50%	0.40231

Table 4.9: Results from the fourth training in version four of the pictogram identification algorithm.

From the results we have obtained (Table 4.10), it's clear that the algorithm performs better with a bigger training set, more examples per iteration, and more iterations. We haven't train it with more iterations, because due to the lack of computational power and the number of training examples, each training took us around 10 hours. What we tested was to run the same testing set against 1.000 pictograms instead of 100. The weights we used for this test are the ones we obtained from 1.850 iterations during the fourth training of this version, and the accuracy we achieved was 55%. Having more examples to test against, the algorithm has more probability to make mis-

takes, therefore the drop in the accuracy was expected. As mentioned before, this can be avoided by training the algorithm with a bigger training set.

		Training set		Validation set			Test set		
#iterations	Batch size	exampes	Type of data	exampes	Type of data	Accuracy	exampes	Type of data	Accuracy
600	44	2.331	Picture and augmented picture	20	Pictures of pictograms	30%	20	Pictures of pictograms	30%
1850	80	2.331	Picture and augmented picture	20	Pictures of pictograms	60%	20	Pictures of pictograms	60%
1850	111	2.331	Picture and augmented picture	20	Pictures of pictograms	65%	20	Pictures of pictograms	75%

Table 4.10: Best results obtained during the first, third and forth training in version four of the pictogram identification algorithm respectively.

4.3 API

As we have two machine learning models, to integrate and use them in Pict2Text 1.0 or any other application, we needed to create an API, providing separate services to execute each of the two algorithms. We aimed to provide services that developers could easily use, connect to, and integrate into other applications.

We have the following endpoints:

- /detect_pictograms: Provided a picture of a sentence written with pictograms, this service executes the algorithm implemented to detect pictograms included in an image (see section 4.1) and returns the output image of it. It contains the coordinates of the detected pictograms, the bounding boxes around them as well as the probability of each one of them.

- /classify_pictograms: This service receives a picture of a single pictogram. If it is in .jpg format, the service converts it to png, and then it executes our algorithm that identifies the word associated with a pictogram in the image (see section 4.2). The result is the id of the predicted pictogram and the similarity score.

To connect the first and the second services, we implemented an additional internal functionality that crops the image of a sentence written with pictograms according to the bounding boxes provided by our YOLO algorithm and sends them to the second service.

4.4 Web Application

To present our models and give our users the option to test our models, we implemented a web application on Flask. The application supports the following URLs:

- uploadImage: As shown in Figute 4.24 this endpoint provides two options to the user. The first one (shown on the top) is to upload an image of a sentence written with pictograms, which should be in .jpg formal. The other option (the bottom one) is to choose one of our testing examples - a picture of one, two, or three pictograms. Once selected an option and a picture, both algorithms are executed and the user is redirected to the next endpoint.

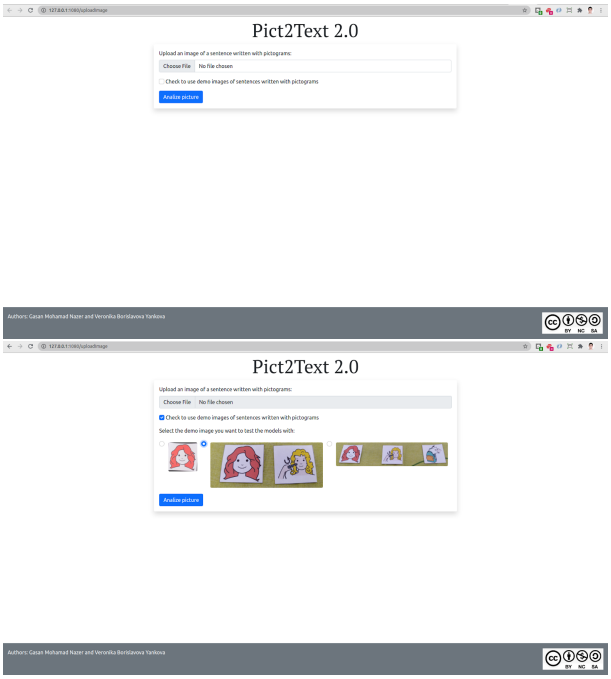

Figure 4.24: The options to upload a jpg picture (top), and to select a demo picture of pictograms (bottom), provided by our application.

- show_results: An endpoint which shows the results obtained by our models. In Figure 4.25 can be seen the result returned from the YOLO algorithm, consisting of the bounding boxes where the algorithm detected pictograms. Then we have a view showing the cropped images (Figure 4.26), which were crop according to the bounding boxes. Last, as can be seen in Figure 4.27, we present the result of executing our image recognition model. Under each of the cropped images appears the id, and the similarity score returned by the model. As the model predicts an id of a pictogram, and we need the corresponding word for that id, we call the ARASAAC API[13] to obtain each word. In the end, we have a link that takes the user back to the previous endpoint where

[13] https://api.arasaac.org/api/pictograms/id_pictogram/languages/es

they can try with a different image.

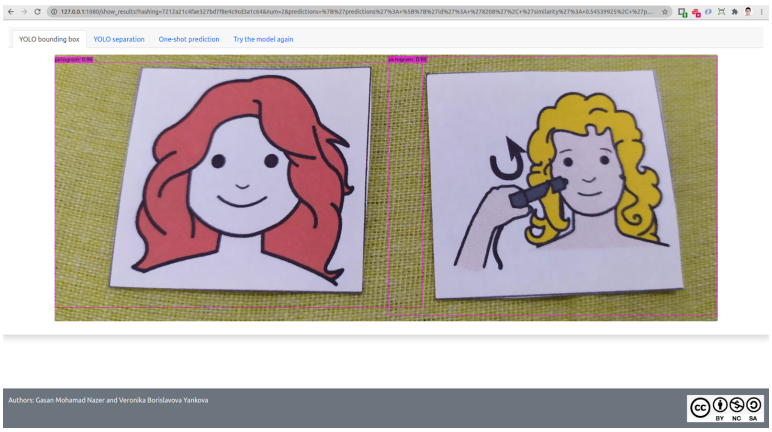

Figure 4.25: The bounding boxes predicted by the YOLO algorithm for a given image.

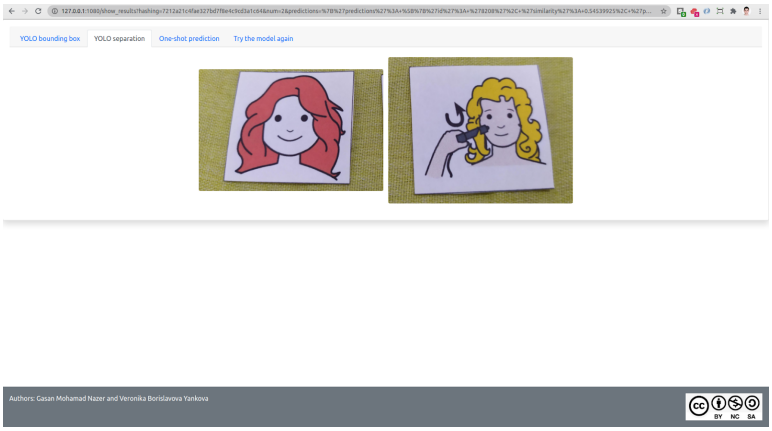

Figure 4.26: The cropped images according to the predicted bounding boxes.

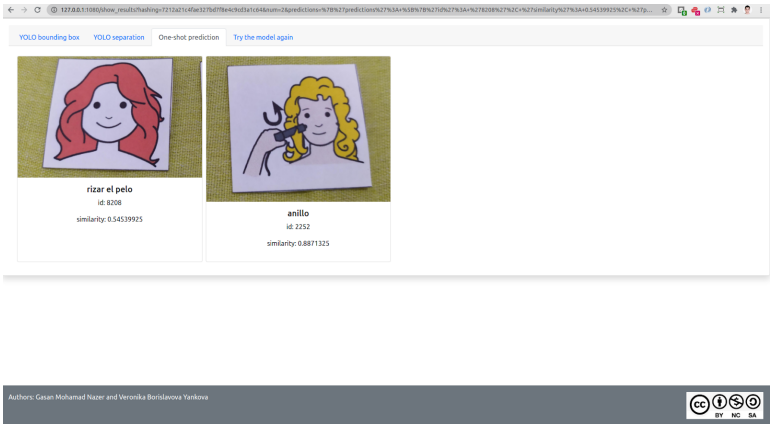

Figure 4.27: The predictions made by our image identification algorithm for each of the cropped images, including information about the corresponding word, id, and similarity score.

Chapter 5

Individual work

In this chapter we will explain the individual work each one of us has done.

5.1 Gasan Mohamad Nazer

At first, we started to write the first chapter of the documentation. From this chapter, I wrote the introduction part, as well as the goals we have established for this project.

After that, I read the documentation of Pict2Text 1.0, and tested the provided application, to see its functionalities, what has been done, and what hasn't. Once I was familiar with the tool and its functionalities and flaws, I wrote the documentation section corresponding to Pict2Text 1.0 (section 2.3).

My next task was to research a pictogram identification algorithm. Although I had previous knowledge related to Machine Learning it was not enough to determine the correct algorithm required to solve our problem. l started by revising the basic machine learning concepts, Neural Network architectures, and computer vision in general. Later I have continued learning about Convolutional Neural Network, classification models based on that, and tools and frameworks to build machine learning models with. With that and after analyzing the ARASAAC dataset I was able to select the algorithm that best suited our needs and wrote the sections corresponding to the Machine Learning model (section 2.4.1), including the subsections Neural Network and Convolutional Neural Network inside, and Keras (section 2.5.2).

When the software development methodology was chosen, I decided what type of tests we will do and wrote that part of the documentation. According to what we have decided, I also reviewed everything my partner did, as we have explained in section 3.1.

Once we have decided which machine learning model to use for image identification, I have developed the scripts to load the pictograms. I also did the script for image rotation and the ones for brightness and colour changing. The corresponding sections in the documentation were also written by me (sections 4.2.1.1 and 4.2.1.2).

As we have both tested the classification algorithm, I wrote sections 4.2.2 and 4.2.3, explaining the first two versions of the algorithm implementation, the results obtained by them, and the issues we have faced. At the beginning of version two, I also included one part describing which problems from the first version we were intending to solve and how.

After the second version of the pictogram identification model, we separated our work with Veronika focusing on correcting the model, while I was researching the second model for object detection. After the initial research, I have written the section 2.4.2 explaining the difference between image classification, image localization, and object detection. In the same section, I explained why we need a second model to detect and separate the pictograms from a sentence written with them. In section 2.4.3 I have written more about object detection. In section 2.4.3.1 I have explained the YOLO model which we had later used to localize pictograms from a picture of them.

Next, I have started researching more about the different YOLO versions and how they are implemented and configured, making the first version of our detection model and its documentation (sections 2.4.3.1, 4.1 and 4.1.1). For the first version of our model, I have labeled a custom dataset with a pictogram in the required format for the algorithm, writing the sections 2.5.3 and 4.1. The second version of our YOLO model was also implemented and written by me (section 4.1.2).

To integrate the two models together, I implemented an API that connects the two models so their performance can be tested. Also, the API provides endpoints for developers, who may want to use our services. After that, I wrote the corresponding documentation (section 4.3). Later I configured the Continuous Deployment for our API described in section 3.3.

I was participating in the integration of the web application to the API creating the cropping functionality, and the way the images are saved to the file system, and the general flow of execution from provided image to returned result passing by the detection and classification models.

All of the work done with the container provided by the university was also done by me. Initially started with the idea of using Docker to dockerize our API and the web application, and also the two models we implemented, so that they could be easily deployed. This idea was discarded after several unsuccessful intentions and faced unresolvable problems. Later I have created the service of the API used in the Continuous Deployment and the unsuccessful integration of the two models.

5.2 Veronika Borislavova Yankova

The first thing I did when we started the project was to select the software development methodology. To do that, I had to research the different methodologies, evaluate the positive and negative sides of each, to finally select Kanban as the most suitable one for us. Once it was chosen, I created the Kanban board to keep everyone updated on the progress of the project, and to organize the tasks in the best way possible as Kanban requires. I added the tutors and my project partner and created the initial tasks we had established at the first meeting. After that, I described everything in the documentation (section 3), so it will be easier for everyone to follow the methodology.

As we were going to work with pictograms, both my partner and I had to familiarize ourselves with them, what they are, what they are used for, and so on. To do so, one of us had to investigate and introduce the other to the new concepts. I started by researching what the Augmentative and Alternative Systems of Communication are, why they are necessary, and what types exist, and then dove into specific research on pictograms, in particular those provided by ARASAAC. In the end, I added that information to the documentation (sections 2.1 and 2.2).

After my partner chose the model we used for the pictogram identification, we discussed and chose which implementation to use among various he had previously found. Having the implementation, I started reading the article and the code, in order to be able to modify the paramethers according to our dataset (the ARASAAC pictograms), and run the model with that data.

As we expected, the algorithm couldn't work with big training and testing sets, because we couldn't load that many pictures at a time. To solve that, for the second version of the algorithm, I changed it in a way it worked with mini-batches, so it didn't compute everything at once. I tested the algorithm to make sure it was working properly, and then tested it with a different number of pairs for each version in order to see which will work better for our algorithm.

In the third version of One-shot learning, I centered my effords into making the algorithm work with images of pictograms, instead of only with digital ones. For that purpouse, I took a hundred pictures of pictograms to train and test the model with pictures. As the pictograms are in png and the pictures were in jpg, I also had to make a script to convert the jpg images into png ones, so we could use them to train and test the model.

For the fourth version of the pictogram identification algorithm, our goal was to inceese the ability of the model to recognize pictures of pictograms. To have more accurate representation of which training perform better with images, I made changes in the three datasets (training, validation and test

sets). The training set was augmented and the data inside was changed from augmented digital pictograms to augmethed pictures of pictograms plus the original digital pictogram. For the validation and test sets I left only pictures so we could have more realistic results based on our goal (to recognize images of pictograms). After that, I trained and tested the model several times changing different parameters, and wrote the result in section 4.2.5.

As I was the one changing the one-shot learning algorithm, I wrote the parts corresponding to the second two versions of the model in the documentation. To make following the results obtained easier, I created tables in all four versions which represent the accuracy, difference in sets, and so on, in a more visual way, which also facilitated the comparison between the different versions.

To present the results we obtained from the two models, I implemented a web application, where the person can upload a picture or use some demo ones, and see the results obtained from our algorithms (the bounding boxes predicted by YOLO, and word, id and similarity score for each pictogram YOLO detected). As we have used Flask for the API, I prepared the templates that resemble HTML files in the static parts. For the styles I mainly used Bootstrap, but I also included some custom CSS styles. Last but not least, for the dynamic behavior of each of the two pages (one to upload a picture, and the other with the results), JavaScript was used. After the front end was ready, my partner and I integrated it with the API.

Then I started researching the technologies we could use for the continuous deployment, as we wanted to implement it for our project so the changes we make to the API could easily be propagated to the virtual container we have. This research is presented in sections 2.5.4, 2.5.5, and 3.3.

At the first meeting, and also later with the methodology, it was determined that every few weeks we would have a meeting with our tutors. During those, we discussed the work done for the time between the two meetings, which goals established at the previous one were achieved, and what would be the goals for the next one. We also submitted a version of the documentation before the meeting, which we discussed with the tutors and had to correct and add the new things we have done for the next time. From the very first review of the documentation, I have been in charge of correcting everything the tutors found as mistakes. I also kept section 1.3, as well as the introduction of each chapter, updated as we added new things into the documentation.

Chapter 6

Conclusions and Future Work

In this chapter, we will present the conclusions of this project (section 6.1) and the future work that could be done to continue it (section 6.2).

6.1 Conclusions

Communication plays a vital role in our lives. It allows us to share information with others and build connections with them. That's why it's essential to make communication easier for people who, due to disability, cannot use the traditional ways of communication. In this final project, we have focused our efforts on helping people whose interactions with others consist mainly or exclusively of pictograms.

The main objective we had was to improve Pict2Text 1.0, a tool that translates pictograms into written text. One of the main things we needed to improve was the way users introduce the message with pictograms that they wanted to translate into natural language. In version 1.0 of the tool, users had to create the message manually by searching each of the pictograms that compose the sentence in the pictogram search engine, integrated into the tool. To improve that, we have established as our first goal to provide users with the option to upload a picture with the input sentence, written with pictograms. To achieve this goal, we have implemented two Machine Learning models: one to detect all the pictograms that appear in the uploaded image, and another one to determine which ARASAAC pictogram corresponds to each detected pictogram.

We have implemented two versions for the detection model (based on the YOLO algorithm). In the first version, explained in section 4.1.1, we managed to configure the model, train it, and test it with our custom dataset. Although the model managed to localize pictograms when there was a single one in an image, it was not able to do so when there were multiple ones. This first

version also showed a drop in the accuracy when the images had different scale variations. Those problems were not solved by the second version of our model, but in it, we managed to augment the training dataset and increase its performance. In this second version (section 4.1.2), the model successfully localized multiple pictograms for some examples. Although the performance of the model is far from perfect, it is performing better just by providing more training examples. Thus being said, we concluded that increasing the dataset increases the performance as well. This conclusion is essential, given the fact that we were able to prove that the algorithm is working, and its incorrect predictions are related to the small dataset we have trained it with and not due to a mistake in the implementation.

To identify the word associated with a pictogram, we have created a model based on the One-shot learning algorithm. We have implemented four versions of it, each solving problems the previous one has detected. In the first version, which we dedicated to making the model work with our data, we noticed that the algorithm was taking a significantly large amount of time not only to train the model but also to test it. In the second version, we solved that problem by loading the data set in batches. As the data we will obtain will be pictures of pictograms, the third and fourth versions were dedicated to increasing the accuracy of the algorithm, when given an image of a pictogram. To do that, we took pictures of various pictograms and trained and tested the model with them. The best result we obtained was 65% on the validation set and 75% on the training set, comparing 20 pictures of pictograms against 100 digital ones for each dataset, and 55% testing against 1.000 pictograms. Those results can be improved by expanding the training set with more examples of pictures of pictograms, a conclusion based on the results of the two final versions, as it is clear that the algorithm performs better and better with the expansion of the dataset.

Given those results, the first goal we have established, to change the form uses introduce a message written with pictograms by giving them the option to upload a picture, was partially met. We didn't extend the functionality of Pict2Text 1.0, since both algorithms need training with bigger training datasets, but we managed to locate more than one pictogram on an image and to predict which word corresponds to those pictograms.

We implemented an API, in order to integrate the models with Pict2Text 1.0 or any other application that would need it. We also implemented a web application, to validate the execution flow between the models. Unfortunately, we couldn't execute that application in the container given to us, due to various problems related to the integration of our models in the given environment. Nevertheless, the application works correctly executed locally.

The full source code of the models, the API, and the application, as well as descriptions of how to execute them and the documentation, can be found

in our GitHub repository[1].

The second goal, improving the translation provided by Pict2Text 1.0, was not met, because the difficulty of the first goal surpassed our initial expectations.

One of the goals we had for this project was to use the knowledge obtained at the university, as well as to expand it and obtain new. Some of the subjects we have studied that helped us were

- **Fundamentos de Programación (Fundamentals of Programming) and Tecnología de la Programación (Programming Technology).** Those subjects gave us basic programming knowledge. Even though we didn't use the programming languages seen in those subjects (C++ and Java), thanks to what we have learned in those subjects, we can adapt faster to using new programming languages (for example Python and JavaScript, used in this project).

- **Ingeniería del Software (Software Engineering), Modelado de Software (Software Modelling), and Gestión de Proyectos Software y MetodologÃas de Desarrollo (Software Project Management and Development Methodologies).** Those subjects taught us about traditional and agile software development methodologies and the importance of using them in a project. In them, we have also learned how to build software products with good and solid architecture. Those subjects also gave us the initial knowledge to implement our continuous deployment.

- **Aprendizaje automático y Big Data (Machine Learning and Big Data) and Ingeniería del Conocimiento (Knowledge Engineering).** From those two subjects, we have obtained a basic idea of what Machine Learning is, as well as some initial idea of what Deep Learning is. We have also learned how to use the programming language Python, the main one used in this project, to construct machine learning models.

- **Aplicaciones Web (Web Aplications).** In this subject, we have learned HTML, CSS, and JavaScript, all used for the front-end of the application created to try our algorithms.

To implement this project, we also needed to acquire additional knowledge (reading papers, and other sources), which, as mentioned, was also an objective we had completed. Some of that knowledge includes the use of Latex, continuous deployment, the creation of APIs, and computer vision and the two models used.

[1] https://github.com/NILGroup/TFG-2021-Pict2Text2.0

6.2 Future Work

Even though we have advanced significantly in the improvement of Pict2Text 1.0, to reach the maximum potential of the tool, some things have to be improved. As future work for this project, we can establish the following:

- **Training the models with more data.** To increase the accuracy and achieve better results, the models need to be trained with more data. For this, more pictures of pictograms have to be taken and labeled. Then they should be given to the models to train on.
- **Extending the set of digital pictograms used for prediction.** At this current state, after the pictograms are localized, they are compared against 100 digital pictograms. That limits the pictograms that can be predicted. In a future version, the detected and cropped image of a pictogram should be compared against all ARASAAC pictograms.
- **Adding this new functionality to Pict2Text 1.0.** To complete the first goal we have established, the functionality of uploading a picture in order to translate it to a written text has to be added to the previous version of the Pict2Text tool. A field to upload pictured should be added to it, and the tool should be connected to our services.
- **Improving the translation of pictograms to text.** Improve the NLP model in order to translate more complicated sentences, as it currently translates simple phrases containing only one subject, one object, and one verb. Some of the things that have to be included in the translation are reflective sentences, sentences containing more than one verb, with several nouns in the subject, crossed-out pictograms indicating a negative statement, placing prepositions in the right place, extend the coverage of the sentence tense detection, pictograms representing expressions.

Appendix A

Continuous deployment configuration

```yaml
name: Deploy the API
on:
  # Triggers the workflow on push event on master
  push:
    branches: [ master ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:

      # Connect to the UCM container using SHH, push the changes of the repository and restart the API service
      - name: API deployment
        uses: appleboy/ssh-action@master #garygrossgarten/github-action-ssh@release
        with:
          command: ls -a
          host: ${{ secrets.HOST }}
          port: ${{ secrets.PORT }}
          username: ${{ secrets.SSH_USER }}
          password: ${{ secrets.SSH_PASSWORD }}
          script: |
            ls
            cd TFG/TFG-2021-Pict2Text2.0
            git pull
            cd API
            export HISTIGNORE="*sudo -S*"
            echo ${{ secrets.SSH_PASSWORD }} | sudo -S -k systemctl restart pict2text2.service
            systemctl status pict2text2.service
            echo "Pipeline finished"
```

Bibliography

Lamba, H. One shot learning with siamese networks using keras. *towards data science*, 2019.

González Álvarez, S. and López Pulido, J. M. Traductor de Pictogramas a Texto. *UCM*, 2019.

O'Shea, K. and Nash, R. An Introduction to Convolutional Neural Networks. *arxiv*, 2015.

Brownlee, J. How Do Convolutional Layers Work in Deep Learning Neural Networks? *Deep Learning for Computer Vision*, 2019.

Javanmard, M. and Alian, M. Comparison between Agile and Traditional software development methodologies. *ResearchGate*, 2015.

Kingma, D. P. and Lei Ba, J. Adam: A method for stochastic optimization. 2015.

Beukelman, D. R. and Light, J. C. Augmentative Alternative Communication Supporting Children and Adults with Complex Communication Needs 2013.

Brownlee, J. A Gentle Introduction to Object Recognition With Deep Learning 2019.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection 2016.

Redmon, J. and Farhadi, A. YOLO9000: Better, Faster, Stronger 2016.

Redmon, J. and Farhadi, A. YOLOv3: An Incremental Improvement 2018.

Bochkovskiy, A. and Wang, C. and Liao, H. M. YOLOv4: Optimal Speed and Accuracy of Object Detection 2020.

BOOCH, G. Object Oriented Design with Applications 1991.

FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures 2000.

RICHARDSON, L. AND RUBY, S. RESTful Web Services 2007.

GIRSHICK, R. AND DONAHUE, J. AND DARRELL, T. AND MALIK, J AND BERKELEY, UC Rich feature hierarchies for accurate object detection and semantic segmentation 2014.

GIRSHICK, R. Fast R-CNN 2015.

REN, S. AND HE, K. AND GIRSHICK, R AND SUN, J. Faster R-CNN: Towards Real-Time ObjectDetection with Region Proposal Networks 2016.

CAI, Z. AND VASCONCELOS, N. Cascade R-CNN: Delving into High Quality Object Detection 2017.

LIU, W AND ANGUELOV, D. AND ERHAN, D. AND SZEGEDY C. AND REED, S. AND FU, C. AND BERG, A. C. SSD: Single Shot MultiBox Detector 2016.

ZHANG, S. AND WEN, L. AND BIAN, X. AND LEI, Z AND LI, S. Z. Single-Shot Refinement Neural Network for Object Detection 2018

DAI, J. AND QI, H. AND XIONG, Y. AND LI, Y. AND ZHANG, G. AND HU, H. AND WEI, Y. Deformable Convolutional Networks 2017

VIOLA, P. AND JONES, M. Robust Real-time Object Detection 2001

VIOLA, P. AND JONES, M. Rapid object detection using a boosted cascade of simple features 2001

LOWE, D. G. Object Recognition from Local Scale-Invariant Features 1999

CORTES, C. AND VAPNIK, V. Support-Vector Networks 1995

MITCHELL, T. Machine Learning 1997

BHADESHIA, H. K. D. H. Neural Networks in Materials Science 1999