Pict2Text 2.0. Translating messages with pictograms into text

Final project
Course 2020–2021

Autors
Gasan Mohamad Nazer
and
Veronika Borislavova Yankova

Directors
Virginia Francisco Gilmartín
and
Susana Bautista Blasco

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

Pict2Text 2.0. Translating messages with pictograms into text

Bachelor's Degree Final Project in Software Engineering

Autors
Gasan Mohamad Nazer
and
Veronika Borislavova Yankova

Directors
Virginia Francisco Gilmartín
and
Susana Bautista Blasco

Convocation: *June 2021*

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

March 10, 2021

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Goals . . . 2
- 1.3 Document structure . . . 2

2 State of the Art **3**
- 2.1 Augmentative and Alternative Systems of Communication (AASC) 3
- 2.2 Pictograms . . . 4
  - 2.2.1 Pictogram systems . . . 4
  - 2.2.2 Communication based on pictograms . . . 7
- 2.3 Pict2Text 1.0 . . . 8
  - 2.3.1 Implementation . . . 9
  - 2.3.2 Conclusions . . . 10
- 2.4 Image recognition . . . 11
  - 2.4.1 Machine Learning and Machine Learning model . . . 11
  - 2.4.2 Neural Network . . . 11
  - 2.4.3 Convolutional Neural Network . . . 13
  - 2.4.4 One-shot learning algorithm . . . 15

3 Software development methodology **19**
- 3.1 Kanban . . . 19

4 Pict2Text 2.0 **21**
- 4.1 Loading ARASAAC pictograms . . . 21
- 4.2 Processing ARASAAC pictograms . . . 22
  - 4.2.1 Changing brightness . . . 22
  - 4.2.2 Rotating images . . . 23
  - 4.2.3 Changing colours . . . 23
- 4.3 One-shot learning algorithm . . . 24

List of figures

2.1 Pictogram representing an icecream. 4
2.2 Example of Blissymbolics 5
2.3 Example of Communication System Using Pictograms (CSP) 5
2.4 Construction of symbologic in Minspeak. 6
2.5 Example for gender diferenciation for the word ‘teacher‘ in ARASAAC. 7
2.6 Pictograms in ARASAAC associated with the word ‘teacher‘. 7
2.7 Example of ACC communication book. 8
2.8 Searching for the word "Hombre" in PICT2Text 1.0. 9
2.9 Adding the pictogram "Hombre" to the pictogram sentence panel. 9
2.10 Translating the sentence "El hombre come una pizza." 10
2.11 A simple neural network architecture. 12
2.12 Filtering over the input image and constructing the feature map. 14
2.13 Filtering over an input image, representing the convolution operation in CNN. 15
2.14 A convolutional neural network formed by several features layers followed by classifications. 15
2.15 Architecture of the Siamese Neural Network. 17

4.1 The original image of the pictogram bee(‘abeja‘) from the ARASAAC. 22
4.2 Augmented images of the pictogram bee (‘abeja‘) using the random brightness augmentation script. 23
4.3 Two 90 degree rotations of the pictogram bee (‘abeja‘) generated using the augmentation script. 23
4.4 Four color augmented images of the pictogram bee (‘abeja‘) generated using the color augmentation script. 24

4.5 Image represented in grayscale image and the corresponding matrix. 25
4.6 Colourful RGB image with its RGB three matrices. 25

List of tables

Chapter 1

Introduction

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

1.1 Motivation

Communication is a pillar in interpersonal relationships, a fundamental need in our society. However, for some people communication requires a lot of effort. Their differences create an uncrossable barrier and almost impossible 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.

In order to include pictograms into the communication between people with disabilities and those without, we can use modern technology and create a 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. Pict2Text 1.0 translates pictograms into natural language, but to create the text with pictograms, the pictograms must be selected manually searching for every single one in a search engine by entering the word associated with the pictogram. This is not good enough as people with disabilities who need the pictograms, cannot think about the words that compose the sentence and search for them in the search engine. This problem can be solved, giving those people the option to upload or take a picture of a sentence constructed with pictograms. Our aim is to build a

new version of Pict2Text, to reduce the exclusion of those people with special needs from the society and break the communication barrier, providing the aforementioned functionality and improving the translator coverage.

The beneficiaries of this software are people who don't understand pictograms but want to communicate with people who are using them. This project will help people understand each other better, creating a more equal society, independently of the ability to use the natural language in our communication.

1.2 Goals

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

1. To change the form in which the existing Pict2Text 1.0 application receives the text written with pictograms, allowing the user to upload a picture, as the current application requires the user to write the word in order to select a certain pictogram

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 complicated sentences.

During the implementation, we will follow a Services-Oriented Architecture (SOA), constructing web services that implement each of the functionalities. This will increase the maintainability of the application.

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 state of the first version of Pict2Text, the functionalities we will include to increase its usefulness, and the general idea of the machine learning algorithms and models we will be using. Chapter 3 (Software development methodology) describes the methodology we have chosen to use, it's characteristics, and things specific to our application of it. Chapter 4 (Implementation) provides detailed information on everything we have developed ourselves.

Chapter 2

State of the Art

In this chapter, we have briefly defined 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 present the solution that we have found for the recognition of the pictograms included in an image.

2.1 Augmentative and Alternative Systems of Communication (AASC)

The Augmentative and Alternative Systems of Communication (AASC) 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, depending on the needs, as they are often personalized to match the needs of its user.

AASC includes different systems of symbols: graphic and gestural. The gestural symbols can vary widely from mimicry 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 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 an 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, that we will see in more detail in the following subsections.

2.2.1.1 Blissymbolics

Blissymbolics[1] 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 some Blissymbolics pictograms such as house, person, love, etcetera. Blissymbolics characters do not correspond to the sounds of any spoken language and have their use in the education of people with communication difficulties.

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

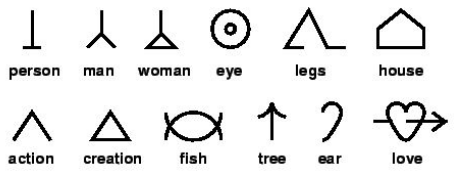

Figure 2.2: Example of Blissymbolics

2.2.1.2 CSUP

The Communication System Using Pictograms (CSP), 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 CSP 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.

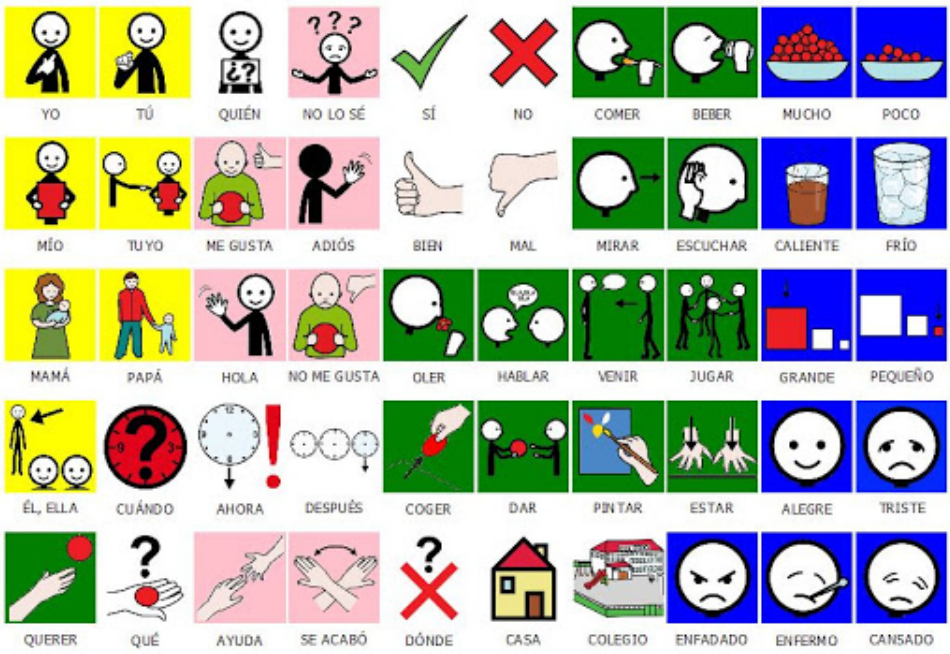

Figure 2.3: Example of Communication System Using Pictograms (CSP)

2.2.1.3 Minspeak

Minspeak 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 for this can be seen in Figure 2.4, where the icons for house and a bed combine to make the word bedroom. In the same figure we can observe how changing the order of the icons can change the meaning - the combination of bed and clothes after that means wardrobe, but if we swap the icons the meaning is pijamas.

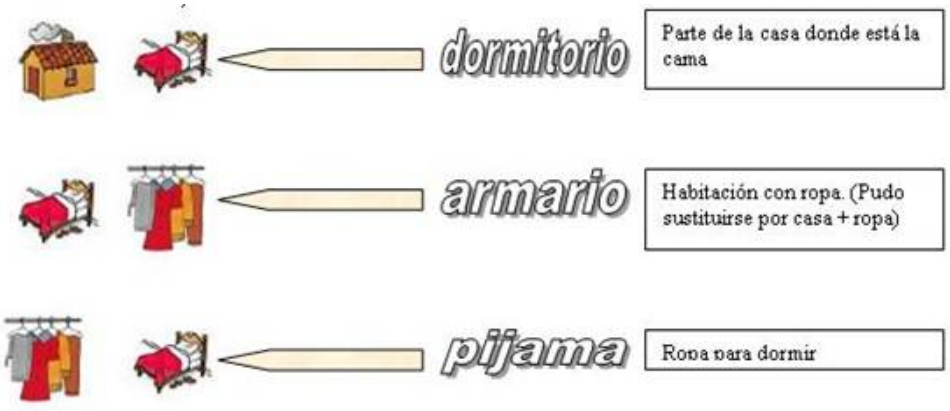

Figure 2.4: Construction of symbologic in Minspeak.

2.2.1.4 ARASAAC

The ARASAAC[2] 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 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 gender differentiation. For example, in Figure 2.5 where you can see the difference between the pictograms representing male and female teachers.

In ARASAAC one word can be represented by various pictograms. In Figure 2.6 we can observe the pictograms for the word 'teacher', which has various representations.

[2]https://arasaac.org/

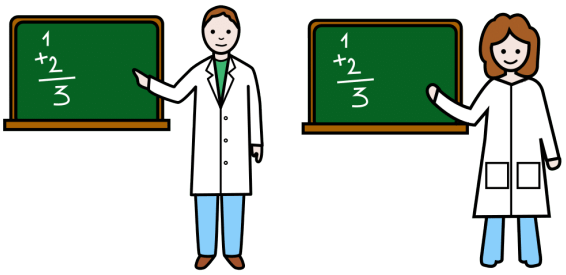

Figure 2.5: Example for gender diferenciation for the word ‘teacher‘ in ARASAAC.

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

Verbs come with a different pictogram for every conjugation, and the tense is determined by pictograms representing yesterday, today, and tomorrow.

ARASAAC is free to use under Creative Commons license, internationally recognized, and has a wide variety of pictograms. ARASAAC also provides a web page from where you can search and/or download the pictograms. For the developers an API is provided that gives functionalities such as: obtaining a pictogram given the id, or the pictograms corresponding to a certain word.

2.2.2 Communication based on pictograms

Communication via pictograms happens with the help of a board or a communication book. Figure 2.7 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.

The ARASAAC 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. In a phrase, past, present, and future are expressed by the pictograms for yesterday, today,

and tomorrow respectively.

Figure 2.7: Example of ACC communication book.

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 sentences written using pictograms to natural language (Spanish).

When entering the website[3] the user can see on the left part, the pictogram sentence panel, with a caption 'Pictograms' above it, and a button '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 into 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 of the right side and click the button 'Buscar'. In Figure 2.8 it can be seen the search for the word 'Hombre'.

After searching the pictogram, the user needs to include it into the left panel with pictograms. This is done by clicking the button "Añadir". In Figure 2.9, 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 2.10 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").

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

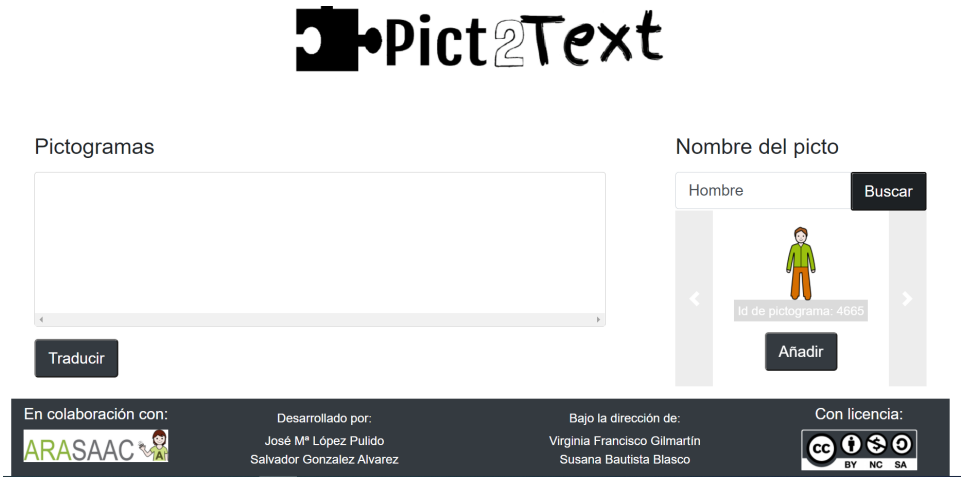

Figure 2.8: Searching for the word "Hombre" in PICT2Text 1.0.

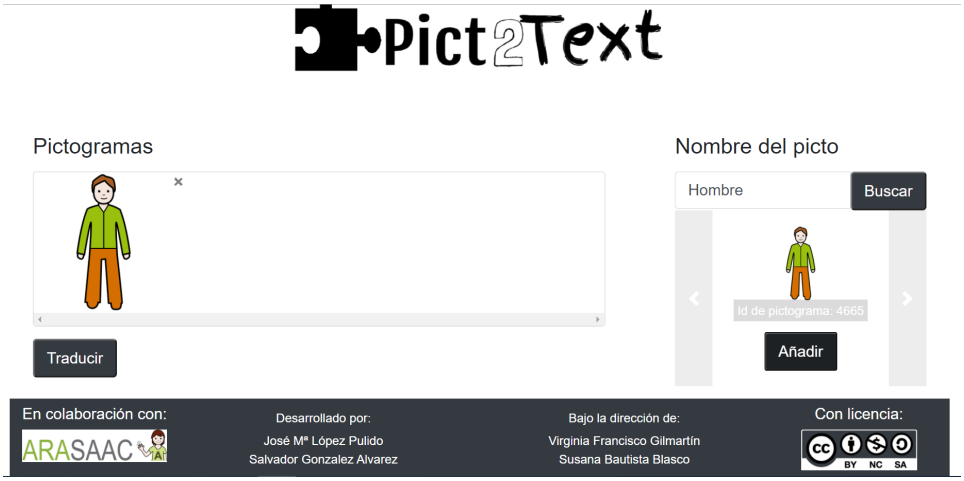

Figure 2.9: Adding the pictogram "Hombre" to the pictogram sentence panel.

2.3.1 Implementation

For the front-end of the project, Angular[4] 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.

The framework Django[5] was used for integration and intercommunication between the implemented web services.

[4] Angular
[5] Django

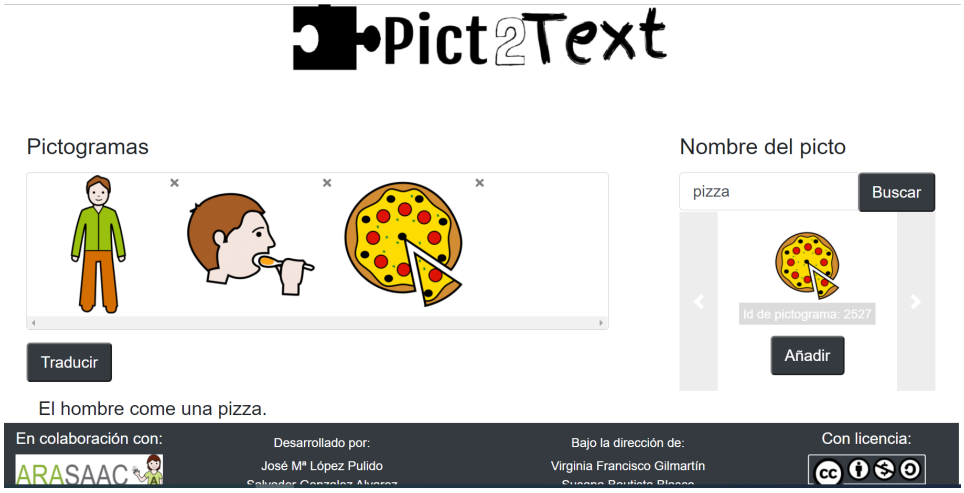

Figure 2.10: Translating the sentence "El hombre come una pizza."

The API of ARASAAC[6] 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 by looking for the subject, verb and proper tense, object, etcetera. To do that Spacy[7], a python library for advanced natural language processing with a high accuracy, was used. The library has an additional support for tokenization - the process of demarcating and possibly classification of words in a given sentence, for more than 65 languages makes it suitable for training on a custom dataset.

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. 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, there are aspects that should be improved in the translation to increase its coverage.

[6] ARASAAC API
[7] Spacy

2.4 Image recognition

To reach the first goal we have established - recognition of pictograms included in a given image, we have to be able to detect the pictograms in the image and its correspondence with ARASAAC pictograms. First we need a machine learning algorithm to identify the pictograms that the image contains. Once the pictograms are identified, we have to determine the exact pictograms shown on the image. Due to a 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. That's why we need a second machine learning model that will be able to tell us which are the pictograms on the picture even with the aforementioned inconsistencies.

2.4.1 Machine Learning and Machine Learning model

Machine learning (ML)[8] 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[9] 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 those data.

2.4.2 Neural Network

Artificial neural networks (ANNs)[10], 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 function of the sum of its inputs. The connections are

[8] Machine learning
[9] What is a machine learning model?
[10] Artificial neural network

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 2.11, 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.

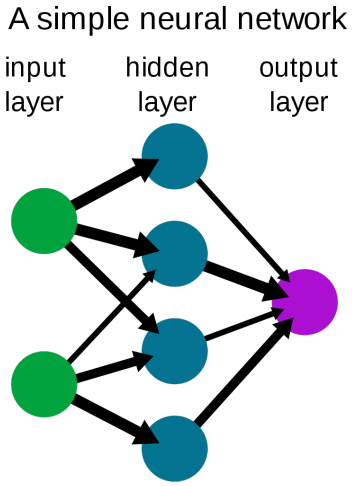

Figure 2.11: A simple neural network architecture.

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 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 is able to 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[11].

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[12] 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

[11] An Introduction to Convolutional Neural Networks
[12] MNIST

dataset, a single neuron in the first hidden layer will contain 784 weights (28x28x1 where 1 bare 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 12288(64x64x3- Height, Width, RGB[13] channels). Also, take 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.

2.4.3 Convolutional Neural Network

Convolutional Neural Networks (CNNs)[14] 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.

The only 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[15], which is explained in the next subsection.

2.4.3.1 Convolution operation

Although 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, this term has a slightly different meaning (the convolution as described in the use of convolutional neural networks, bellow, is a 'cross-correlation'). In the context of a convolutional neural network, convolution is 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. The filter is smaller than the input data, and the type

[13] RGB
[14] An Introduction to Convolutional Neural Networks
[15] Convolution

of multiplication is applied between a filter-sized patch of the input, and the filter is a dot product. A dot product is an 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. 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'[16]- Figure 2.12.

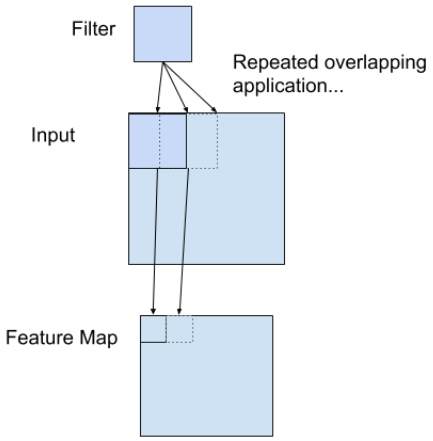

Figure 2.12: Filtering over the input image and constructing the feature map.

As you can see in the Figure 2.13, 3 by 3 filter for vertical edge detection applied to an image with height 6 and width 6 with stride 1 and no padding.

2.4.3.2 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

[16]How Do Convolutional Layers Work in Deep Learning Neural Networks?

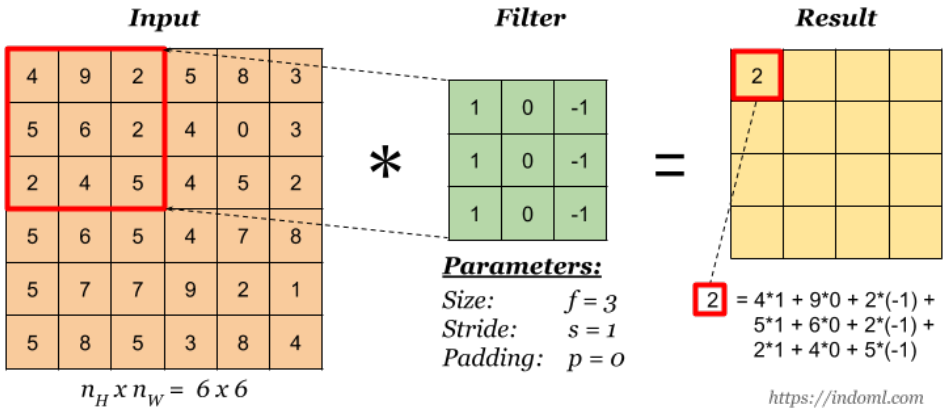

Figure 2.13: Filtering over an input image, representing the convolution operation in CNN.

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 2.14 you can see a CNN architecture can be seen.

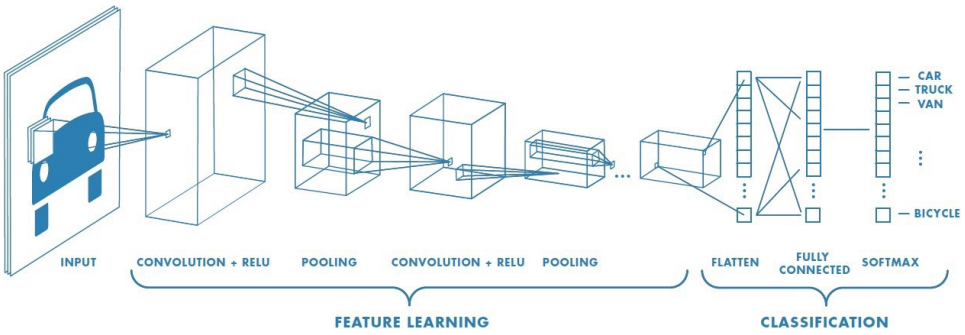

Figure 2.14: A convolutional neural network formed by several features layers followed by classifications.

2.4.4 One-shot learning algorithm

Most computer vision machine learning algorithms are based on object categorization and require a large amount of data. Instead of treating the task as a classification problem, one-shot learning turns it into a difference-evaluation problem. That allows the algorithm to be able to categorize 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 will be higher than the threshold.

One shot learning uses an architecture called Siamese neural network, that is explained in the following subsection. During the training phase some hyperparameters should be tuned - the number of iterations - the number of times the algorithm's parameters are updated, number of pairs to compare during every iteration, etcetera.

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

An example for one-shot learning is the article One Shot Learning with Siamese Networks using Keras, which will also be explained in a following subsection. The problem it solves is letters recognition. The dataset the authors use consists of 1623 characters from 50 alphabets. The images are of handwritten characters in grayscale with 105x105 resolution, with only 20 examples of each of the 1623 classes.

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, patterns that bring classes together, and that enables us to add new classes without training the model again.

In Figure 2.15 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 - 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,

and the opposite - if it is a big number the inputs are not similar at all.

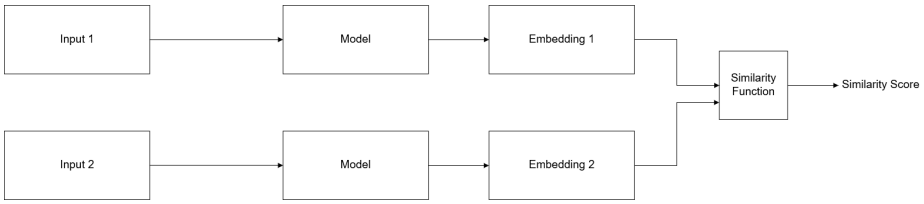

Figure 2.15: Architecture of the Siamese Neural Network.

On the other hand, the cons of the SNN compared to the traditional neural networks are that it needs more training and it takes more time than normal networks and doesnâ€™t output probabilities.

2.4.4.2 Keras

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

TensorFlow 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.

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 neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well.

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 velocity.

The data preprocessing module is one of the essential modules in Keras it provides different utilities for image, timeseries, and text preprocessing. The image preprocessing submodule includes the functions $\text{image}_dataset_from_directory, load_img, img_to_array, fl$

The function $\text{image}_dataset_from_directory provides the functionality to load in memory images from a spec$

The function $\text{load}_img provides the functionality to load a single image in memory. load_img comes with addi$

The function $\text{img}_to_array provides the functionality to convert a PIL Image (Pillow library format) instan$

A.

Chapter 3

Software development methodology

Software development methodologies[1] are used tprovide 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 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 make them more flexible. Some of the most popular agile methodologies are Scrum, Kanban, and Extreme Programming (XP).

In the following sections we will explain the methodology we have chosen, why, and how will we apply it.

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 it as a 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.

[1] Comparison between Agile and Traditional software development methodologies

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

- **To do.** Tasks given to us by the tutors 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 assignments 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. 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, 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'.

Chapter 4

Pict2Text 2.0

In this chapter, we are going to present Pict2Text 2.0. In section 4.1 we describe the pre required scripts we had created to download all the pictograms from ARASAAC and preprocess them. As one of the main goals of this project is to provide the functionality to import a picture of a message written with pictograms and translate it, we had to construct a machine learning model to do that. In section 4.2 we describe the machine learning model implemented to recognize an image of a single pictogram.

4.1 Loading ARASAAC pictograms

First of all, we needed to obtain the ARASAAC pictograms dataset in Spanish, in order to be able to train our model, and later to compare the given image with the whole dataset. We created a Python script that uses a REST client to call the ARASAAC API. The script does a request to the ARASAAC API, which returns the information of all ARASAAC pictograms in Spanish. Later it downloads every one of them calling to the same API. The fetched pictograms are stored locally in separate folders with a maximum size of 4000 images. As the process of downloading the pictograms required more than 3 hours for the complete fetching of over 11000 images, we included concurrency and reduced the execution time to under an hour.

Having the dataset fetched we had to load it in memory in a way that the model we had constructed 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 chapter 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 and resize them to 105 height by 105 width.

4.2 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. As a result of that, during the preprocessing of the dataset, we needed to augment the dataset and generate different representatives of every one of the pictograms. To achieve that, we had created three augmentations scripts. All of them use Keras's preprocessing module but they differ in the way they manipulate the pictograms. Keras's module includes the class ImageDataGenerator which comes with different image manipulation and augmentation capabilities like changing the brightness of an image, rotating it, zooming it, and more.

4.2.1 Changing brightness

The first script changes the brightness of the 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. In Figure **??** is shown the original pictogram of a bee ('abeja').

Figure 4.1: The original image of the pictogram bee('abeja') from the ARASAAC.

In figure **??** are shown augmented images of the pictogram bee ('abeja') using our random brightness augmentation 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.

Figure 4.2: Augmented images of the pictogram bee (‘abeja‘) using the random brightness augmentation script.

4.2.2 Rotating images

The second script rotates the pictogram randomly. Similar to the brightness augmentation script this one is using 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 **??** are shown two augmented images of the bee pictogram using the rotating script.

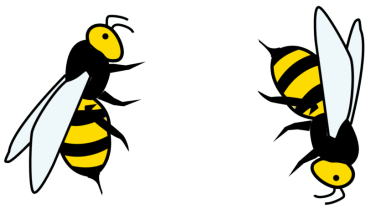

Figure 4.3: Two 90 degree rotations of the pictogram bee (‘abeja‘) generated using the augmentation script.

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

4.2.3 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 **??** are shown four augmentations of the pictogram bee (‘abeja‘) using the color augmentation script.

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

Figure 4.4: Four color augmented images of the pictogram bee ('abeja') generated using the color augmentation script.

4.3 One-shot learning algorithm

For the implementation of the one-shot learning algorithm, we have taken an already implemented one from the *One-Shot Learning with Siamese Networks using Keras* Lamba (2019), recognizing an image of a single letter, as we have explained previously in chapter 2. The implementation shown in the article is suitable for our needs, given that the problem with letter recognition is similar to the one with pictogram recognition.

A difference between the two problems - the one solved in the article and the one we are solving, is the dataset. The original algorithm is implemented to work with a dataset formed by pictures of letters from different alphabets. The aforementioned letters are in grayscale, and our pictograms are in PNG format. As shown in Figure **??**, 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 **??**, the colorful images need three matrices for the channels red, green, and blue for every pixel. In our case, we have a fourth matrix representing the alpha channel (the opacity of the picture), as it is included in the RGBA, png images we use. For that reason, we had to include one additional layer to the input matrices used by the algorithm for the colors and the opacity of our data.

First, we tried the algorithm with fewer data. We have used three data sets - training, validation, and test sets. The training set was used to train the model and achieve the weights we will need to recognize given pictograms, later when we ask the model to predict. The validation set was used to compare the accuracy of the algorithm on different data in order to tune the hyperparameters, like the number of iterations. The last one, the test set, we 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 the test set to compute how similar a given pictogram is to all other pictograms forming the set.

For the training set, we used 22 pictograms that we augmented by using

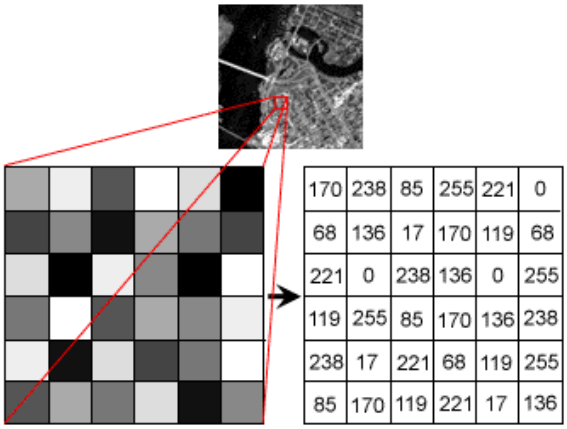

Figure 4.5: Image represented in grayscale image and the corresponding matrix.

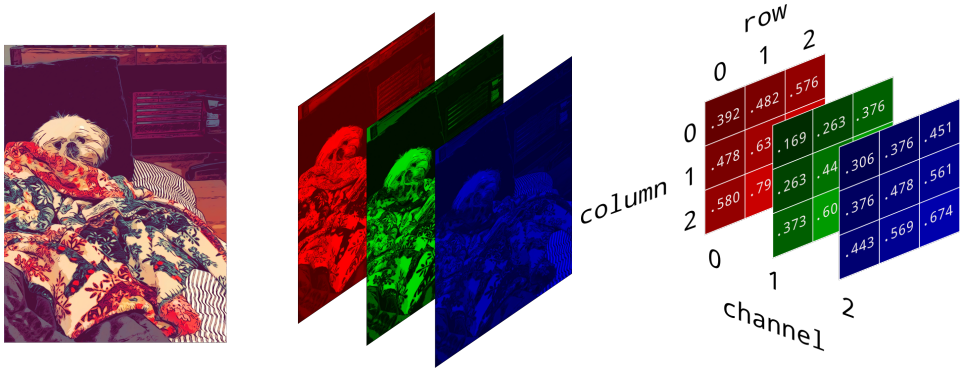

Figure 4.6: Colourful RGB image with its RGB three matrices.

the script described before. After the augmentation, our training set contained 440 pictograms, 20 from each pictogram. For the validation set, we used 10 different pictograms that we augmented, as we did with the training set, making the total size of the validation set to 200. Due to the little amount of data, the best performance was reached at 10 iterations with 100% accuracy on the validation set. After that point, the model started overfitting the training data and the accuracy of the validation set dropped to 60% for 100 iterations.

After we have reached the maximum accuracy, we took 100 different non-augmented pictograms and formed the training set. Using the weights obtained with the 10 iterations we once again obtained 100% accuracy, given that the algorithm recognized correctly all 100 pictograms. For comparison, we ran the algorithm with the same test set and the weights obtained with

100 iterations. From the 100 pictograms, the algorithm correctly recognized only 17 of them, showing that using 100 iterations we overfed the data from the training set.

The problems we faced, which was the main reason why we used only 22 pictograms (440 images with the augmentation) and only 100 pictograms for the test set, were the following. First, we couldn't load all the pictograms we obtained from ARASAAC to try it at once because of a lack of system memory. The second problem was the time the model takes to make a prediction. To correctly compute how similar a given pictogram is to every other pictogram in the test set, the model requires pairs of images. A pair is constructed by the pictogram we want to recognize and one of the test sets. Mapping every pictogram of the test set to the given one creates the list of pairs we give to the algorithm to make predictions on.

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

4.3.1 First Iteration of the One-shot model

In the first iteration(implementation) of the One-shot model we had constructed, we detected several problems.

First, we were able to train it with a small number of pictograms (22 pictograms with augmentation 440) 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 result was bad.

The second problem our algorithm had was the consumption of system memory. This problem was also related to not loading the images in batches.

The third problem our algorithm had was related to the prediction time and the pairing before the predictions step. 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 real 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.

4.3.2 Second Iteration of the One-shot model

In the second iteration of the algorithm we constructed, we aimed to fix the problems from its previous implementation. One of the biggest problems we faced in the first implementation was related to the way we were loading the pictograms because of it we were facing memory capacity problems, we were not able to train the model sufficiently hence it was overfitting and it was giving bad results. As a result of that, the first thing we corrected in the

second iteration was loading the images in batches.

To load the images in batches we used the method $flow_from_directory from ImageDataGenerator class$ described way of creating a DirectoryIterator with the batches of images was used to load the train, validation,

By loading the images in batches we needed to adapt our previous implementation of the algorithm changing the function for creating pairs of half similar images and half different used to create the input and the target for the model to train on. 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.

4.3.2.1 Testing the model

In this iteration of the model, we continue using the previous testing mechanism, taking a random image from the testing set and searching the most similar to it from the same data set. This test was showing 100% accuracy, correctly predicting all pictograms of the test set in the first iteration of the model. Although this result, we detected that this test was not sufficient to verify the effectiveness of our model. The two main reasons for that are:

- 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. In it, the users of the application are going to import an image(a photo) of a message written with pictograms to the system. This photo will never have the quality or the characteristics of the original pictograms used during the training. Hence 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, not randomly taken from the test set, pictogram is to all others.

To resolve the first problem we included photos of pictograms taken by us to the test set.

Our new testing set includes both pictograms from the original dataset and also pictures of pictograms.

Retraining the model with 100 iterations and batches of 2 augmented pictograms (40 images), the model managed to detect all pictograms from the original dataset but from our pictures of the pictograms it managed to correctly predict similarity for 6 out of 10.

Bibliography

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