LAB University of Applied Sciences

Kanji Recognition with AI

LAB University of Applied Sciences
Bachelor Degree in Information and communication technology (ICT)
2024
Rafael José Rodríguez Badas

Abstract

Author(s)	Publication type	Completion year
Rodríguez Badas, Rafael José	Thesis, UAS	2024
	Number of pages 62	

Title of the thesis

Kanji Recognition with AI

Degree, Field of Study

Bachelor's degree in information and communication technology (ICT)

Name, title and organisation of the client

Ismo Jakonen, Senior Lecturer, Media Technology

Abstract

This thesis explores the development of a mobile application for recognizing Kanji characters using artificial intelligence (AI) techniques. Beginning with an introduction to the Japanese language and an overview of Kanji, the thesis continues with the fundamentals of AI, including its types and applications in various domains. It discusses machine learning principles, focusing on supervised and unsupervised learning, along with error metrics such as overfitting and underfitting. The thesis then delves into deep learning concepts, including neural networks, activation functions, and training methodologies. Special attention is given to convolutional neural networks (CNNs) due to their efficacy in image recognition tasks. Moreover, it examines standard tools used in AI applications, particularly Python programming language and associated libraries. The thesis culminates in designing and implementing a Japanese kanji Recognition mobile app, detailing its frontend and backend components.

Keywords: Kanji recognition, Mobile application, Artificial intelligence (AI), Japanese language, Machine learning, Deep learning, Neural networks, Convolutional neural networks (CNNs), Language learning tools, Image recognition

Contents

List of abbreviations/Concepts/Terms

AGI: Artificial General Intelligence

AI: artificial intelligence

ANN: Artificial Neural Network

API: application programming interface

BP: Back Propagation

CNN: Convolutional Neural Network.

FCN: Fully Connected Network

Framework: a tool that provides ready-made components or customised solutions to

GAN: Generative adversarial network.

GUI: graphical user interface

LLM: Large Language Model

NLP: Natural Language Processing.

OCR: Optical Character Recognition

RNN: Recurrent Neural Network.

ReLU: Rectified Linear Unit

SAI: Super Artificial Intelligence

SDK: Software Development Kit

SGD: Stochastic Gradient Descent

SVM: Support Vector Machine

Script: programs or sequences of instructions that are interpreted and used to automatise tasks

TF: TensorFlow

TFLite: TensorFlow Lite

UI: user interface

WYSIWYG: What You See Is What You Get

1 Introduction

Undoubtedly, technology has been one of the driving forces in the last centuries, redefining and shaping how we live, work, and interact with the world. It has dramatically improved the quality of life we enjoy today, has significantly enhanced our quality of life, and has led to remarkable breakthroughs in almost every field of human knowledge. Time and time again humanity has defied the odds with incredible discoveries.

AI has taken the headlines of the world in the last few years. It is no surprise that AI has taken the center stage, with new technologies capable of doing seemingly impossible things being developed every few weeks. The advancements in this field have been so rapid that professionals often struggle to keep up.

Learning a new language can be challenging, especially if it involves non-Latin scripts, like the Japanese language. One of the most challenging aspects of learning Japanese is memorizing the kanji characters, ideographic characters used in the Japanese writing system. There are over 2,000 regular-use kanji characters, each with its own meaning and pronunciation. Learning to recognize and read them can be overwhelming for learners.

This thesis aims to explore how the use of AI can help in recognizing such characters, in order to provide an efficient way to look up kanji characters without drawing or searching for radicals, which are the components that made up a kanji character. This way, the thesis hopes to facilitate the learning process and help learners improve their reading and writing skills in Japanese.

2 The Japanese language

2.1 About the Japanese language

Japanese is the mother tongue for almost all Japanese citizens, numbering around 128 million in 2011, making it the ninth largest native-speaking population globally. Additionally, as of November 2011, approximately 128,000 individuals in Japan were non-native speakers studying Japanese as a foreign language. Beyond Japan, around 3.65 million people across 133 countries were learning Japanese in 2009. (Hasegawa, Y. 2014.)

Japanese employ two scripts known as kana, namely hiragana and katakana, which depict the same set of phonetic sounds in different forms. Hiragana and katakana encompass nearly 50 characters, derived from simplified Chinese characters adopted for phonetic representation. Additionally, Japanese employs kanji or Chinese characters extensively in writing, with over 40,000 characters existing, though a learner must master around 2,000 "joyo" (official everyday use) kanji to be functionally literate in the language, as these kanjis constitute most of the written text. Kanji is crucial for distinguishing between words in the absence of spaces and for disambiguating homophones, a common occurrence due to the language's limited distinct sounds. The use of these three scripts can be seen in Figure 1. (Kim, T. 2012.)

Hiragana primarily serves grammatical functions, representing challenging kanji, colloquialisms, and onomatopoeias. It is also commonly used by beginner Japanese learners and children in place of unfamiliar kanji. Conversely, while representing the same sounds as hiragana, katakana is primarily employed for loanwords from Western languages, lacking associated kanji. (Kim, T. 2012.)

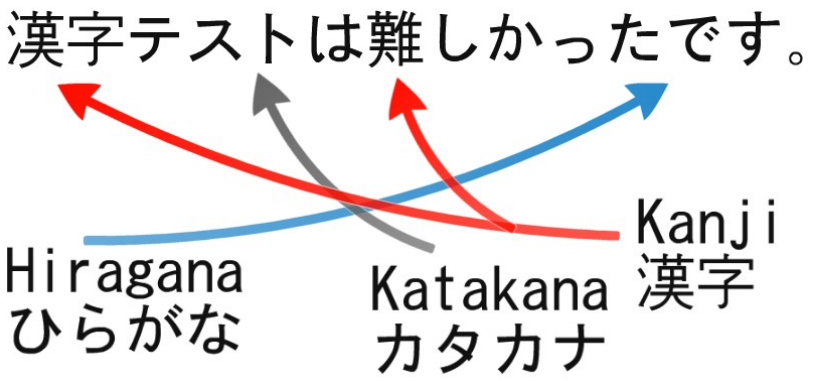


Figure 1. Hiragana, katakana and kanji used together in a sentence (NihongoShark)

2.2 Introduction to Kanji

During the 5th century, Japanese Buddhist monks introduced Chinese texts written in Chinese to Japan. As Japanese lacked a written form at that time, they adopted Chinese characters, known as kanji or “漢字”. Initially, these texts would have been read following the Chinese language. Even documents written by Japanese scholars were essentially imitations of Chinese texts in terms of grammar, morphology, and syntax, despite the vast linguistic differences between Chinese and Japanese. (Beermann, R. E. 2006.)

In Japanese, nouns, adjective stems, and verbs are predominantly written in kanji, necessitating knowledge of Chinese characters for comprehending most words. However, not all words use kanji; for instance, verbs like "to do" are consistently written in hiragana. (Kim, T. 2012.)

One of the challenges about learning kanji, is that each kanji character typically possesses two readings: on'yomi, taken from Chinese, and kun'yomi, the native Japanese reading. Compound words generally use on'yomi readings, while single kanji characters often utilize kun'yomi. Some characters, especially prevalent ones, may have multiple readings. Moreover, certain compound words may have unique readings requiring individual memorization. (Kim, T. 2012.)

Kun'yomi is also prevalent in adjectives and verbs, often accompanied by kana strings known as okurigana, to maintain pronunciation consistency during conjugation. Furthermore, kanji readings may undergo slight alterations in compound words for ease of pronunciation, such as changes from /h/ to /b/ or /p/ sounds. (Kim, T. 2012.)

Another intriguing and challenging aspect of kanji is the existence of synonyms with similar readings but nuanced differences in meaning, exemplified by pairs like “聞く” (“kiku”) and “聴く” (“kiku”), where the latter implies a deeper level of attention. In most cases, when it comes to listening to music, the verb “聴く” is more commonly used than “聞く”. Similarly, slight variations in kanji can alter the meaning, as seen in “書く” (“kaku”) meaning "to write," contrasting with "描く" (“kaku”) meaning "to draw," but assuming the reading “egaku” when describing abstract imagery, such as a scene on a book. Also, some kanji have multiple readings, like “今日” (“kyō”), (“konjitsu”), or (“kon ni chi”), with preferred readings varying by context. (Kim, T. 2012.)

2.3 The prevalence of kanji

One might question why the Japanese did not switch from Chinese characters to romaji (Roman alphabet) to simplify the learning process. After all, Korea successfully streamlined its written language by adopting its own alphabet. However, when converting typed hiragana into kanji, one is often faced with multiple choices (homophones), sometimes up to ten, due to the limited number of distinct sounds in Japanese. This stands in contrast to the Korean alphabet, which comprises 14 consonants and 10 vowels, allowing for a much wider array of sounds. Additionally, Korean allows for the attachment of a third or even fourth consonant to create a single letter, resulting in a theoretically vast number of possible sounds. (Kim, T. 2012.)

As reading speed typically outpaces speaking pace, visual cues are crucial for swiftly identifying words. English achieves this through the varied shapes of words, even when misspelled. Korean employs a similar strategy, thanks to its ample characters capable of creating words with distinct shapes. However, spaces must be added to remove ambiguities, presenting challenges regarding their placement. Kanji resolves many of these issues by eliminating the need for spaces and largely mitigating problems with homophones. Without kanji, even with added spaces, the lack of visual cues and resulting ambiguities would significantly hinder the readability of Japanese text. (Kim, T. 2012.)

3 Fundamentals of AI

3.1 Introduction to AI

Before talking about artificial intelligence, it is convenient to first talk about intelligence. Intelligence, while a common word in the dictionaries of many languages, does not have an agreed-upon definition. Philosophers, psychologists, scientists, and engineers all have different answers about the definition of this word and how intelligence came to be. In their 2007 work on AI research, Legg and Hutter consolidated various definitions from existing literature into one single definition: *Intelligence measures an agent's ability to achieve goals in a wide range of environments.* (Chollet, F 2019.)

In general, autonomy and adaptiveness are signs of intelligence; autonomy is the lack of need for constant instructions, and adaptiveness is the ability to change behaviour depending on the environment or problem space. The problem is that, in any scientific and technological field, it is mandatory to have formal measuring methods. Therefore, AI will not mature until a formal and global method for measuring intelligence is developed. (Linares 2021a.)

Looking at this definition, one can reason that associating intelligence only with human beings seems reasonable. But as shown in Figure 2, this anthropocentric vision does not consider that human intelligence is actually a subset of natural intelligence, and AI overlaps with both human and natural intelligence. (rodrivers 2019.)

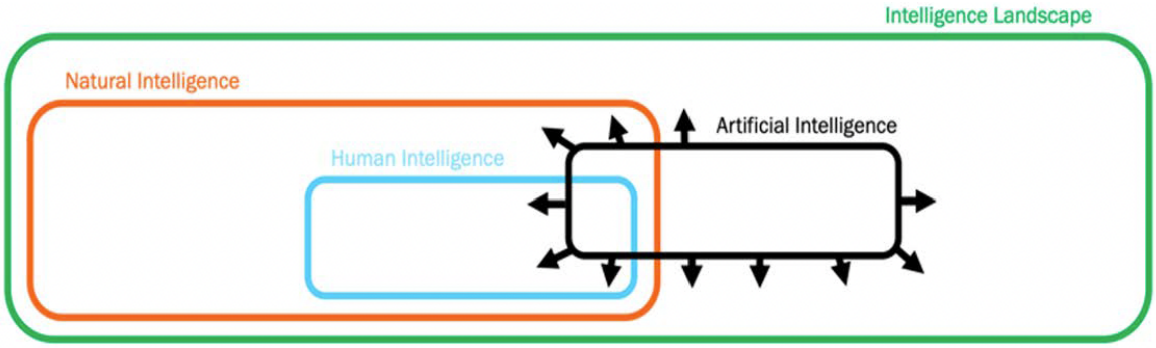


Figure 2. Intelligence Landscape (rodrivers 2019)

Like intelligence, AI also has multiple definitions to choose from. Elaine Rich and Kevin Knight defined it in their book Artificial Intelligence as *the study of how to make computers do things at which, at the moment, people are better.* (Rich and Knight 1991.)

The concept of Artificial Intelligence has been a subject of fascination and curiosity for many years, placing itself as a crucial research area in academia and industry, which is currently undergoing a significant bottleneck due to its exponential growth in the last few years. AI

has evolved into a vast discipline in recent decades, drawing upon computer science, mathematics, linguistics, and many others. (Shao Z et al. 2022.)

Even though AI has become the new playing field for scientific and technological innovation as well as industrial transformation, to the point that it established itself as the so-called fourth industrial revolution, its development has been far from smooth. AI's history witnessed several ups and downs throughout its lifetime. (Shao Z et al. 2022.)

3.2 Overview of AI

Isaac Asimov's science fiction novel Runaround published in 1942 is considered to be the origin of AI. During the 1940s and 1950s, researchers from diverse disciplines such as mathematics, psychology, and engineering began to explore the concept of an artificial brain. This idea was influenced by neurological research indicating that the brain functioned as a network of neurons emitting pulses. In 1943, neurologist Warren McCulloch and mathematician Walter Pitts collaborated on a book that merged mathematics with algorithms. This work paved the way for the development of neural networks and the mathematical modelling of artificial neural networks. (Shao, Z et al. 2022.)

In his publication Computing Machinery and Intelligence, Turing detailed the process of assessing a machine's intelligence. The Turing Test, a recognized benchmark for determining the intelligence of a machine, was introduced in the same publication, and is still widely utilized today. Figure 3 illustrates the Turing test, which involves a human evaluator assessing conversations between a human and a machine programmed to imitate human responses. All participants in the test are isolated from each other. The evaluator knows that one of the conversational partners is a machine. The machine is considered to have passed the test if the evaluator cannot consistently tell it apart from a human. The outcome of the test does not depend on the correctness of the machine's responses, but on how indistinguishable they are from responses a human might provide. (Shao, Z et al. 2022; Wikipedia contributors 2024d.)

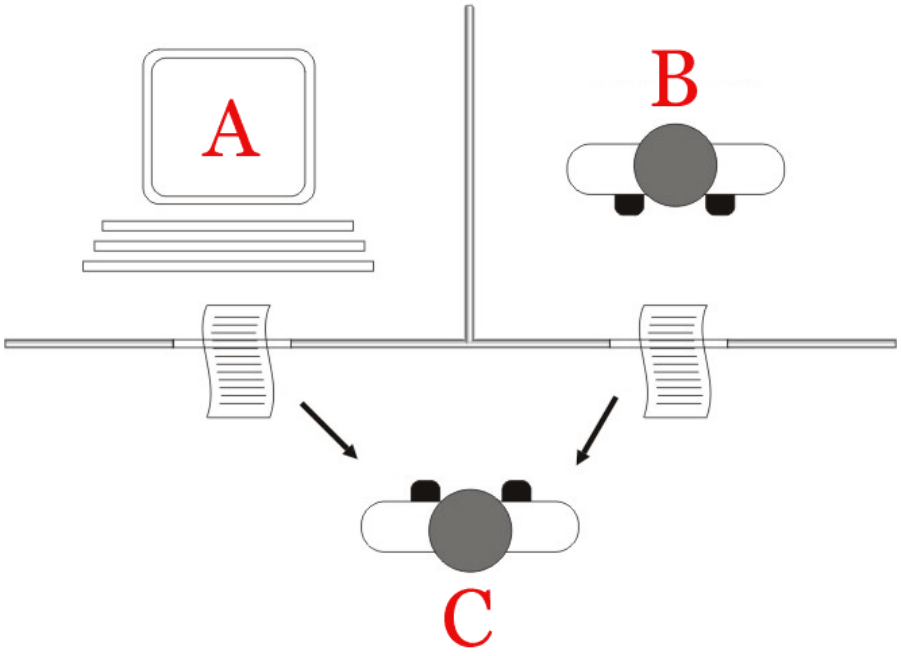


Figure 3. Turing test (Wikipedia contributors 2024d)

The formal proposal of artificial intelligence as a concept was made by John McCarthy in 1956 during a seminar at Dartmouth. This event is considered the official birth of AI. The seminar focused extensively on the possibility of using machines to mimic human intelligence. Many of the attendees of this conference subsequently became significant contributors to the development of AI over the next several decades. (Shao, Z et al. 2022.)

During the 1960s, the field of AI had its first significant breakthrough, thanks to the development of symbolic logic, which helped to solve various joint problems. One of the significant events in the history of AI was the development of a system called STUDENT by Daniel Bonrow in 1964. The system was written in Lisp and could understand natural language input and solve algebraic word problems. Around this period, it seemed that AI development would only accelerate. Some even made bold predictions, such as *in twenty years, machines will be able to do all the work that humans do*. These high expectations for AI development soon proved unrealistic, as researchers underestimated the complexity of their field. Also, in the 1970s, AI faced difficulties that could not be overcome at the time. The limited memory and processing power of computers hindered the progress of practical AI problems. This made the research progress come to a halt, marking the start of the first "AI winter". (Shao, Z et al. 2022.)

It was not until ten years later that AI entered its second climax, thanks to the adoption of expert systems by worldwide companies. These systems imitate the decision-making process of humans and give suggestions to non-experts. The first expert system, DENDRAL

was proposed by Edward Feigenbaum. Nevertheless, the maintenance and upgrading of these systems was complex. This fact led to what is referred to as the second "AI winter" in the history of artificial intelligence. Until then, AI had relied on models of reasoning that attempted to replicate human intelligence, such as planning, reasoning, and decision-making, using knowledge and experience. A prominent example of such an AI system was IBM's Deep Blue, the chess program that famously beat the world chess champion in 1997. (Shao, Z et al. 2022.)

The field of AI, now more than half a century old, saw its third wave of success with the appearance of deep learning. Without a doubt, 2012 is a year that rings the bell in the mind of any AI professional, given that this marks the beginning of the deep modern learning era through an enormous breakthrough in solving the challenges of ImageNet. (Shao, Z et al. 2022.)

The prosperity AI enjoys today can be summarized in three significant facts:

1. New meaningful learning algorithms start emerging, such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Transfer Learning, etc.
2. The advancements in computer hardware and software allow AI to overcome problems that could not be solved, especially in computer vision and Neuro-linguistic Programming (NLP).
3. AI has been integrated into people's daily life, such as self-driving cars, virtual assistants, recommendation engines, spam filters, etc.
4. The development of several optimization methods for neural networks, like Batch Normalization, Layer Normalization, Dropout, Gradient Descent method, etc.

(Shao, Z et al. 2022.)

A whole summary of the history of AI can be seen in Figure 4.

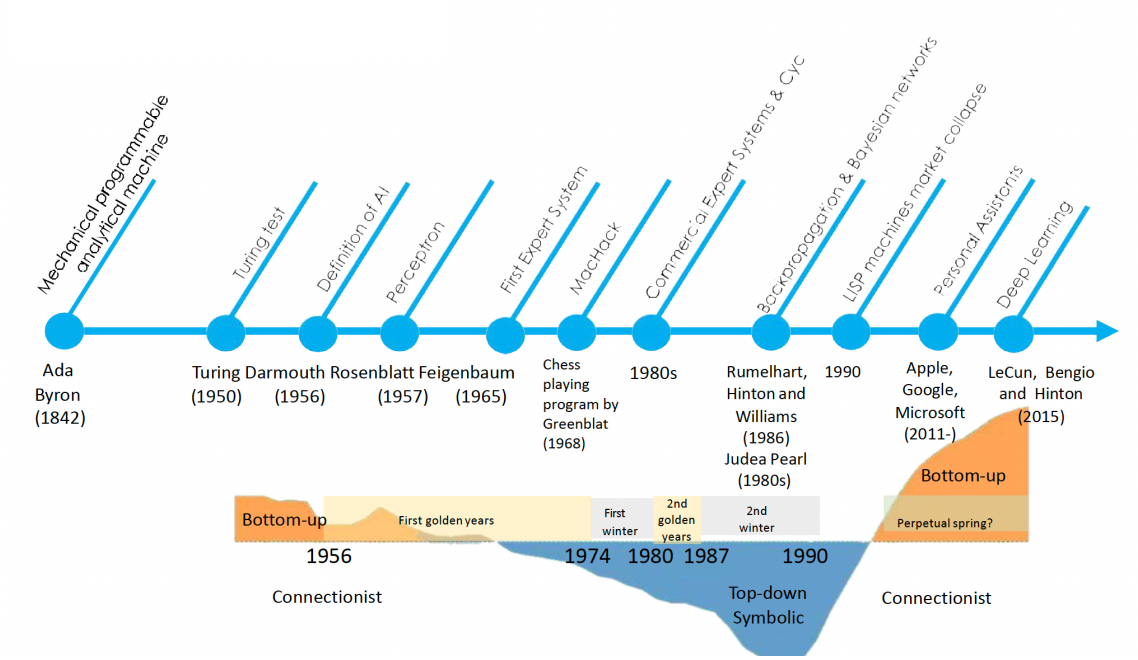


Figure 4. Chronology of AI (Linares 2021a)

Bo Zhang suggested that the evolution of AI could be segmented into three distinct phases:

1. Symbolic AI, often referred to as the knowledge-driven approach.
2. The data-driven approach, which relies on deep learning.
3. The Third Generation AI, which is a theory that integrates both previous approaches.

(Shao, Z et al. 2022.)

3.3 Types of AI

The most common way to classify the types of AI is by weak, general, or super AI. Weak or narrow AI attempts to model the human mind, like how weather conditions, climate change, or other natural phenomena are modelled, to focus on a specific problem domain, like cars, face recognition, spam filters, etc. On the other hand, general or strong AI actually seeks to reproduce the human mind, in the sense that it exhibits intelligence in a broad range of tasks. A major example of strong AI is Artificial General Intelligence (AGI). Lastly, super AI (SAI) is defined by Nick Bostrom as *an intellect that is much smarter than the human mind in every field*. Looking at how humanity still has not been able to achieve true AGI, although Microsoft researchers claim GPT-4 to be the first "sparks," SAI will be contained to science fiction for the foreseeable future. (Flowers, J.C. 2019; Linares 2021a; Bubeck, S et al. 2023.)

3.4 AI in fiction media

The growing popularity of artificial intelligence has led to mixed feelings of both excitement and fear in popular discourse. AI is widely regarded as a significant technological field rapidly advancing and shaping the future. Thus, futuristic concepts involving AI are becoming more common in popular media. (Goode, L. 2018.)

Since the early 20th century, scenarios in which AI surpasses human intelligence have frequently appeared in science fiction. These narratives often unfold in dystopian settings marked by machine rebellions. In 1921, Karel Capek's play "RUR" (Rossum's Universal Robots) became the first science fiction work to depict a revolt by humanoid robots against their human oppressor, introducing the term "robot" to both the literary and scientific communities. The concept of threatening machine intelligence has persisted in science fiction, reflecting societal fears as advancements in digital technology accelerate. However, there are notable exceptions that feature compassionate AI, such as Wall-E (2008), Data from "Star Trek: The Next Generation" (1987-1994), Robby from "Lost in Space" (1965-1968), and TARS from "Interstellar" (2014). Isaac Asimov's mid-20th-century robot stories, particularly "I, Robot" which was adapted into a film, significantly shaped public views on AI or machine intelligence. Asimov introduced what he called the "Frankenstein complex", which is the notion that despite implanting AI with safety protocols to prevent danger or rebellion, the underlying fear that it could ultimately turn against us still remains. However, in most cases, Japanese fiction is known for representing AI as a friend or tool. Mighty Atom, known as Astroboy in English, was the friendly protagonist of a manga series that ran from 1952 to 1968. Doraemon was also a manga series first serialized in 1969, in which a podgy, friendly, blue robot cat from the future aids a boy named Nobita Nobi. (Cave, S et al. 2018; Goode, L. 2018*;* Wikipedia contributors 2024b; Wikipedia contributors 2024c; Wikipedia contributors 2024d.)

Another significant entry in Japanese fiction is "Sword Art Online: Alicization" (2018-2020), a part of the larger "Sword Art Online" series that explores the theme of sentient artificial intelligence. In this storyline, the Soul Translator is introduced as an advanced full-dive interface created by the private institute Rath. Unlike traditional methods that send signals directly to the brain, the Soul Translator interacts with the user's Fluctlight, which is the technological counterpart to the human soul. This device creates a virtual realm known as the Underworld. The series eventually reveals that the primary goal of this technology is to develop a new, more sophisticated form of AI. In this virtual world, a human civilization with human emotions is presented. Later in the show, Alice, one of the inhabitants of the Underworld and the main protagonist, is introduced as the first autonomous Artificial Intelligence

(Figure 5) by implanting her Fluctlight into an artificial body. In the Underworld, the lines between reality and virtual existence become blurred, offering a profound exploration of the ethical, moral, and existential questions surrounding sentient AI. (Myanimelist.net; Wikipedia contributors 2023a).

Figure 5. Alice being presented as the first autonomous Artificial Intelligence (SAO: Alicization 2018)

3.5 The two schools of AI

Since its birth, AI was divided into two schools of thought: The symbolic, or top-down approach, which relies on explicit rules and representations; and the connectionist, or bottom-up approach, which makes use of neural networks and focuses on learning from data and patterns. (Linares 2021a.)

3.5.1 Symbolic approach

The symbolic, or top-down approach, was the first school of AI that originated, and in the early days of AI, most of the efforts were focused on this approach for many years. This classical approach relies on encoding a model of the problem by programming a set of predefined rules and logical principles and asking the system to process the input data under this model so that a solution can be provided. This set of rules and logic represents knowledge provided by experts, allowing for the creation of expert and decision-support systems. Systems in this category employ deductive reasoning, logical inference, and some

type of search algorithm that finds a solution within the constraints of the problem. (Bajada, J., 2019; Linares 2021a.)

One of the advantages of this approach include its high interpretability, as one can easily trace the reasoning process back to the logical rules that were applied and the ease of updating the system's rules as new information becomes available. (Bajada, J. 2019.)

3.5.2 Connectionist approach

The connectionist, or bottom-up approach, was also one of the first approaches to AI, but it failed to maintain relevancy due to hardware, data, and theoretical limitations at the time. The names come from the network topology that characterizes the algorithms in this class. Researchers akin to this school thought that AI should be inspired by biology (the brain), in the fact that it learns from observation and experience. Convolutional Neural Networks (CNN), the main focus of this thesis, fall into this category. What differentiates this approach from the previous one is that the rules and logic of the domain being modelled do not need to be specified. The network learns these rules by itself from training data. Neural Networks do not require a model of the world; instead, they rely on substantial training data from which a model of the world can be statistically inferred. While this is the strongest point going for this approach, it is also a double-edged sword. If the training data is biased, has data with little to do with the problem trying to be solved, or is insufficient, connectionist algorithms will perform poorly. In fact, the lack of data was a massive constraint at the time and prompted the connectionism approach to fall out of relevance compared to the symbolic approach. (Bajada, J. 2019; Linares 2021a.)

4 Fundamentals of Machine Learning

4.1 Introduction to Machine Learning

Machine Learning, a subset of artificial intelligence and computer science, focuses on how computers can learn from experience without explicit programming. It resides at the nexus of computer science and statistics, forming the foundation of artificial intelligence and data science. Advancements in machine learning have been propelled by new algorithms and theories, the thriving availability of data online, and affordable computing resources. Currently, machine learning techniques, which are data-intensive, are increasingly utilized across diverse sectors including healthcare, manufacturing, education, financial services, law enforcement, and marketing, facilitating decisions based on data. (Jordan, M.I., & Mitchell, T. 2015.)

In the past twenty years, machine learning has transitioned from a theoretical idea to a fundamental technology with widespread practical and commercial uses. It has emerged as the preferred method in AI for addressing challenges in fields such as computer vision, speech recognition, and natural language processing. AI developers have discovered that training systems with examples of desired outcomes is often simpler than programming them to anticipate responses for every conceivable input. This shift has significantly influenced industries that handle complex systems such as diagnostics or logistics. Additionally, machine learning's efficacy in analysing large volumes of experimental data has made it valuable in empirical sciences like biology, cosmology, and even social science. (Jordan, M.I., & Mitchell, T. 2015.)

As with intelligence, there is no common agreement on what learning is. Yet, regarding humans, learning can be defined as *changes in behaviour that result from experience or mechanistically as changes in the organism that result from experience.* Since computers are mathematical machines, learning in computers involves programmatic changes. Figure 6 contrasts traditional programming, where the program dictates outputs from given inputs, with machine learning, which involves determining a suitable model from a set of inputs and outputs. This model, once trained, can then generate new outputs for new inputs. (Duarte, D., & Ståhl, N. 2018.)

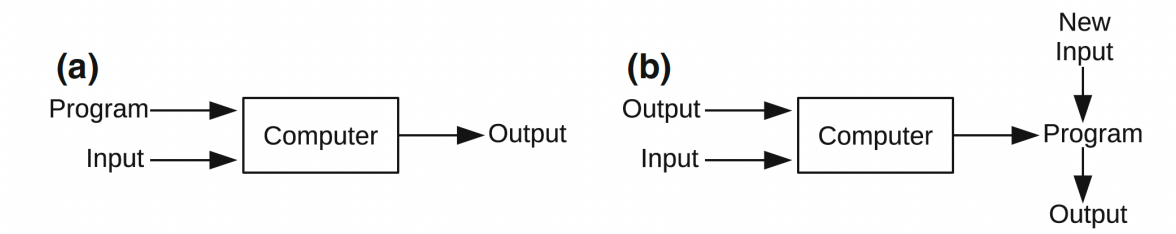


Figure 6. Traditional programming (a) vs Machine Learning (b) (Duarte, D., & Ståhl, N. 2018)

Tom M. Mitchell provides a formal definition of machine learning, stating that a computer program learns from experience (E) concerning a specific task (T) and performance measure (P) if its performance at T, as evaluated by P, improves with experience E. For example, to develop a model that classifies emails as spam or not spam, one might start with a set of emails (Se) divided into spam (Sse) and not spam ($Snse$). This dataset serves as the experience (E) for the model, which then classifies new emails based on this training. The effectiveness of the model, or performance (P), is contingent upon the quality of the input dataset (Se), with better data potentially leading to more accurate classifications. (Duarte, D., & Ståhl, N. 2018.)

4.2 Growth of Machine Learning

The so-called Big Data is a phenomenon that describes how, in the past decade, computers and networked systems have significantly improved their ability to gather and transport huge amounts of data. Machine learning has emerged as a powerful tool for scientists and engineers, enabling them to derive meaningful insights and predictions from vast amounts of data. The significant rise of machine learning can be attributed to advancements in mobile and embedded systems, which now have the capability to collect extensive data about individuals. This data collection facilitates the development of personalized services aimed at individual needs. Across diverse sectors, including commerce, science, and government, the accumulation of large datasets is being used to improve services and productivity. In the medical field, for instance, historical patient records are analysed to identify the most suitable treatments for individual patients, traffic control and congestion reduction are being improved with the help of historical traffic data, historical crime data is being utilized to assign local police to particular locations at specific times. (Jordan, M.I., & Mitchell, T. 2015.)

Even so, it must be considered that machine learning is not a magic formula capable of solving every problem. Machine learning works best when

- a task is too hard to be programmed
- the problem involves a large search space

- adaptivity is essential
- a large enough data set is available.

(Linares 2021c.)

4.3 Types of Machine Learning

When solving a machine learning problem, the first step to address is which machine learning algorithm to use. This is quite challenging as there are already many options to choose from, and each year, another hundred are proposed. The hypothesis space is the collection of potential learning algorithms that can be used for a specific machine learning problem. To decrease this hypothesis space, the learning components of a problem can be categorized into three distinct groups:

1. Representation: By understanding the type of learning to be accomplished, the hypothesis space can be reduced since the task needs to be represented by a particular algorithm.
2. Evaluation: It's necessary to evaluate the effectiveness of the chosen algorithm by assessing its predicted outputs.
3. Optimization: Based on the evaluation results, subsequent optimization is required. The learning algorithm should aim to enhance a specific performance metric.

(Duarte, D., & Ståhl, N. 2018.)

Machine learning can be broadly classified as supervised, and unsupervised learning. (Linares 2021c.)

4.3.1 Supervised learning

Supervised learning is applied when the dataset includes labels for each example, referred to as the ground truth. Consequently, the dataset is split into two parts: the features of the examples, represented as X, and the labels, denoted as y. Supervised learning tries to create a model that receives a vector of features and generates an output. Broadly speaking, a supervised learning model aims to generalize outputs in front of non-observed input data, using the knowledge acquired by previous examples. Mathematically, supervised learning systems typically generate their predictions by means of a learned mapping function $f(x)$, which provides an output y for each input x (or a probability distribution over y given x). There are numerous mapping methods for f, such as decision trees, logistic regression, support vector machines, neural networks, and Bayesian classifiers. Additionally,

general techniques like boosting and multiple kernel learning exist to combine the outputs from several learning algorithms. (Jordan, M.I., & Mitchell, T. 2015; Duarte, D., & Ståhl, N. 2018; Linares 2021c).

The model's output can be discrete (classes) or continuous (numeric values). If the output is a class or a label, the model is said to be a classifier, whereas if the output is a continuous value, the model is a regressor. (Duarte, D., & Ståhl, N. 2018; Linares 2021c).

Regression

Regression is a supervised learning technique with the goal of predicting a numeric value from new input elements. Estimating the future price of a house, predicting currency exchange rates, or weather prediction are all examples of regression. The most common type of regression is linear regression, although there are other types beyond this paper's scope. (Duarte, D., & Ståhl, N. 2018; Linares 2021c).

Linear regression can be defined mathematically as in Equation 1:

$$h_\Theta(X) = \theta_0 + \theta_1 \times x_1^{(i)} + \cdots + \theta_m \times x_m^{(i)} \quad (1)$$

where θ_j are weights (θ_0 is the bias, and θ_k is the weight for the kth feature of $x^{(i)}$, $1 \leq k \leq m$), and $x_j^{(i)}$ is the jth feature of the ith example in X (dataset). The weights indicate how important each feature is for the output of the model. (Duarte, D., & Ståhl, N. 2018.)

Classification

On the other hand, classification tries to infer a label from the input vector by choosing among several categories or classes. Taking this into account, a regression problem can be transformed into a classification problem by discretizing the continuous values, that is, grouping them into classes. Table 1 is an example of a model that classifies a given sample of wind and temperature into a pace. (Duarte, D., & Ståhl, N. 2018; Linares 2021c):

Table 1. Data set with two features (wind speed and temperature) with their predicted class (fast or normal) (Duarte, D., & Ståhl, N. 2018)

Wind speed (km/h)	**Temperature (ºC)**	**Pace**	**# Class**
10.5	12.3	Fast	0
8.9	15.4	Fast	0
20.2	13.7	Normal	1
5.10	3.1	Normal	1

Classes may be binary like Table 1 or multiclass. Classifying emails between spam and not spam would be an example of binary classification, while a medical diagnosis from a vector of features (sex, blood pressure, cholesterol, etc.) would be an example of multiclass classification. As with regression, a multiclass problem can be translated into a binary classification problem using *one* versus *all* classifications. (Duarte, D., & Ståhl, N. 2018; Linares 2021c.)

Supervised learning can be divided into two phases (Figure 7):

- Training process: A selection of pertinent features is identified from the input data. These features are extracted using methods like Exploratory Data Analysis (EDA), which enables data scientists to examine and explore data sets in order to summarize their key attributes, frequently making use of data visualization techniques. However, data scientists must be careful to avoid “the curse of dimensionality”, which means worse results as the number of features increases.
- Prediction process: A set of the same features from new data are fed into the trained model to make a prediction.

(Linares 2021c.)

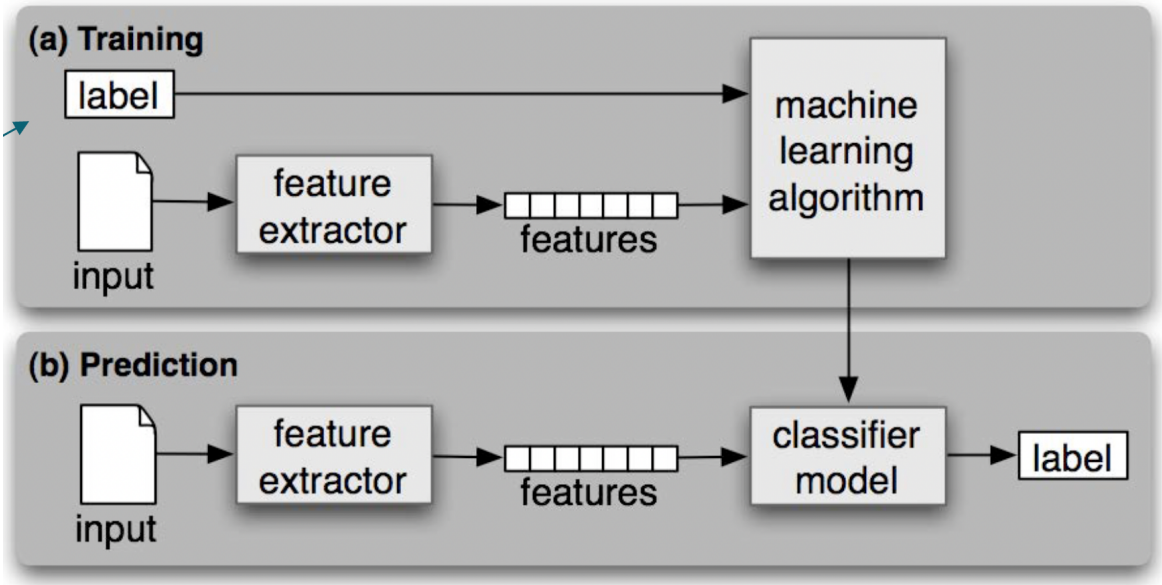


Figure 7. Supervised learning pipeline (Linares 2021c)

4.3.2 Unsupervised learning

Unsupervised learning is involved when the dataset has no labels (examples are not pre-classified). The goal here is to deduce classes without the ground truth. Since this approach is less objective than supervised learning, understanding the dataset's domain is crucial for building effective models; otherwise, the results may not be comprehensible. Even though at first glance this technique might seem less relevant than supervised learning, its importance prevails in the fact that there are more unlabelled than labelled datasets. Moreover, many mainstream problems like recommendation systems, classification of user's behaviour on a website, or market segmentation are closely related to unsupervised learning. By far, the most popular technique for unsupervised learning is clustering. (Duarte, D., & Ståhl, N. 2018.)

Clustering refers to the process of identifying related groups within datasets that lack any labelling. This technique has proven useful in detecting anomalies, assessing the similarity between organisms, and identifying meaningful features, among other applications. K-means is one of the most popular clustering algorithms, and one of the easiest to comprehend. The fundamental concept revolves around defining k centroids utilized to form the clusters. For each example present in the dataset, there is a corresponding association with one of the k centroids. The dataset is considered a collection of points on a plane, and the algorithm seeks to group these points into clusters by measuring the distance between a given data point and a centroid. (Duarte, D., & Ståhl, N. 2018.)

There are other techniques such as supervised learning, or techniques that combine supervised and unsupervised learning, like semi-supervised or self-supervised learning, but these go beyond the scope of the paper. (Linares 2021c.)

4.4 Error metrics

When working with machine learning models, measuring how well a given model is behaving is critical. For that reason, many error metrics have been developed in order to gauge the performance of machine learning models. There are different metrics for supervised and unsupervised learning. In particular, for supervised learning, a training set taken from the dataset is used to see how well the model is learning. It is not a good idea, however, to rely only on the metrics the model produces over the training set, as this does not guarantee the model will accurately predict results for previously unseen inputs. That is why it is common practice to use a technique called *hold-out*, where a test set is used for evaluating the model in data it has not seen in the training process. The overall goal of a machine learning model is to be able to generalize, i.e., producing good results in front of unseen data. Hence,

introducing two concepts referring to the generalization of a model is necessary: overfitting and underfitting. An underfitting model performs poorly on both the training and testing datasets, whereas an overfitting model performs exceptionally well in the training dataset but poorly on the test data set. (Linares 2021c.)

4.4.1 Overfitting

Overfitting happens when a model has become too specialized in the training set. This can happen either because the model was overtrained, or because that it has learned the noise and details of the dataset. This negatively affects the model's generalization ability, as noise is specific to each training sample and does not apply as a general pattern to recognize new data. There are several techniques to prevent overfitting, such as:

- Cross-validation: It is a systematic repetition of the hold-out technique that gives more statistically accurate metrics.
- Regularization: It involves penalizing the learning process to prevent the model from learning the training set too well.
- More data: Having a more extensive dataset will help prevent overfitting, although the data quality is more important than the quantity.
- Ensembling: It is a set of techniques that combine predictions from multiple algorithms together. The most famous examples are Bagging and Boosting.

(Narayan, S., & Tagliarini, G.A. 2005; Linares 2021c.)

4.4.2 Underfitting

Underfitting occurs when a model cannot learn the training data nor generalize it to new data. This issue is often disregarded because it is easy to detect and can be resolved by switching to different machine learning algorithms or enhancing the quantity or quality of the data. Contrarily, overfitting is a more complex problem and is more common in the field. (Narayan, S., & Tagliarini, G.A. 2005; Linares 2021c.)

In Figure 8, an example of both underfitting an overfitting can be seen.

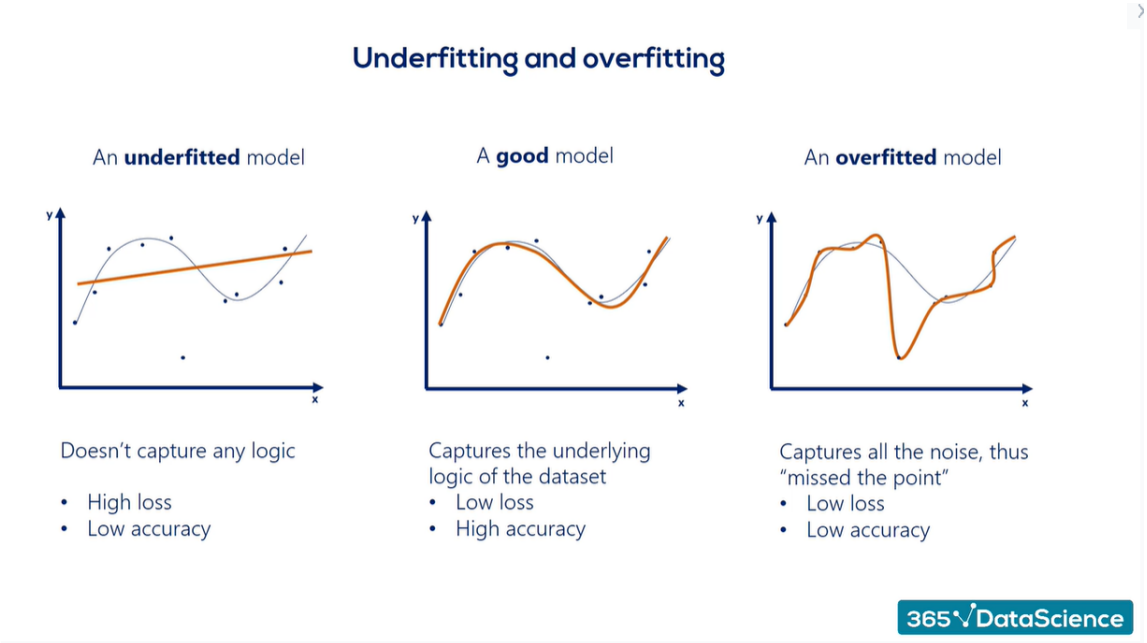


Figure 8. Overfitting and underfitting (The 365 team 2023)

Despite being commercially successful, machine learning is still a young field with numerous research opportunities yet to be explored. Unfortunately, there are still challenges to overcome, such as the fact that much of the data required for these opportunities is privately held and owned. Despite the challenges, the potential and ongoing developments indicate that machine learning will probably be one of the most transformative technologies of the 21st century. (Jordan, M.I., & Mitchell, T. 2015.)

5 Neural Networks and Deep Learning

5.1 Introduction to Deep Learning

Deep learning is a branch of machine learning algorithms where machines autonomously develop internal representations from raw data to perform tasks such as regression or classification. Deep learning models are made in a layer-wise structure, with each layer learning hidden representations often too obscure for human observers to comprehend. These representations are nonlinear compositions of the previous layer's representations, enabling the model to progress from simple to increasingly complex and abstract features in each layer. For example, deep learning models in image processing often begin by identifying fundamental features like edges and strokes. These basic features are then integrated to form basic objects, and as the model progresses through subsequent layers, these basic objects are further combined to create more intricate and complex structures. These algorithms can be used in supervised or unsupervised training for applications in pattern analysis (unsupervised) and classification (supervised). The rapid growth of deep learning has been driven by three key factors: extensive data availability, powerful computational capabilities, and innovative algorithms. (Duarte, D., & Ståhl, N. 2018; Xia, Z. 2019.)

Deep learning, while a recent advancement in technology, has a substantial history that dates back 50 years. It originated in the 1940s with the development of artificial neural networks (ANNs). The initial models of these ANNs were simple linear models that linked input x to output y. In the mid-1980s, the introduction of the backpropagation (BP) algorithm marked a significant leap in learning the parameters of artificial networks, sparking a resurgence in statistical model-based machine learning. With the BP algorithm, these artificial networks are capable of learning statistical rules from extensive datasets and predicting outcomes for new cases. For this reason, neural networks are excellent for tackling complex learning problems. In 2012, with the breakthrough in image recognition during ImageNet, the world realized the true potential of neural networks. Here, AlexNet managed to improve performance by 20%. (Xia, Z. 2019.)

At the moment, deep learning is considered the most advanced machine learning technique. However, it may not always be the best option, especially for structured data. More traditional machine learning algorithms can deliver excellent results in such cases. Sometimes, even better results can be obtained from small datasets using options like XGBT (Extreme Gradient Boosting), which the winners of Kaggle contests often prefer. However, for unstructured data, such as images, videos, text, and graphs, deep learning undoubtedly provides the best results, provided that the dataset is large enough. (Linares 2021c.)

5.2 Representation learning

A key factor that has contributed to the growth of deep learning is representation learning. In the past, researchers have relied on their expertise to determine which attributes in the data were most important for a particular task. These attributes, also known as features, are validated through the performance of learning tasks like clustering or classification. However, feature engineering can be challenging, particularly when the raw data is unstructured and high-dimensional. In representation learning, as Figure 9 shows, the model does not just learn; it also decides which are the best set of features and patterns to pay attention. This automatic learning of the data representation, i.e., the ability to find this embedding automatically, makes deep learning much more powerful than other techniques. (Moyano, L.G. 2017.)

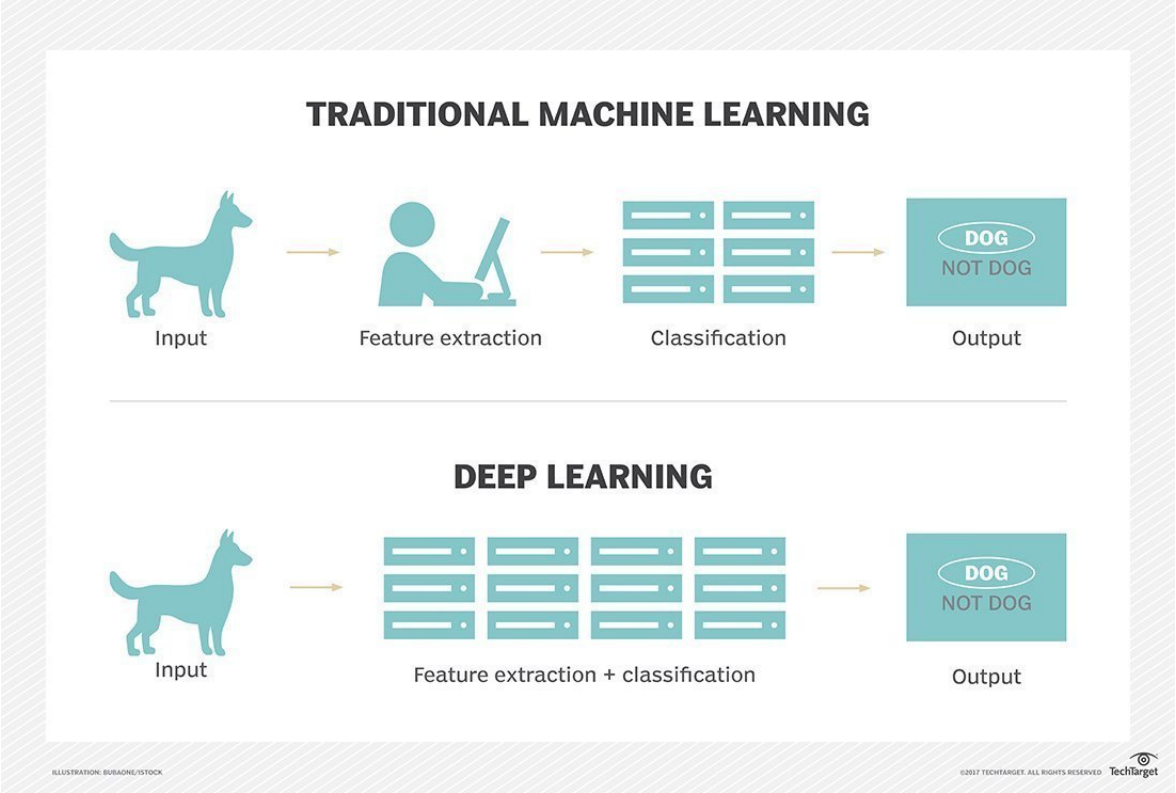


Figure 9. In representation learning, the model performs the feature extraction (Robinson, S. 2020)

5.3 Basics of Neural Networks

Informally, a function is a system of outputs and inputs, as shown in Equation 2:

$$x \rightarrow f(x) \rightarrow y \qquad (2)$$

A function takes an input, makes some kind of processing, and gives an output as a result. All the possible values for x and y can be drawn in a graph to obtain a line. Thus, if the function is known, the correct output (y) can always be calculated for any given input (x). But what happens when a function is not known but only some values for x and y? If there was some way to reverse engineer such function, obtaining the y value from a given x value that was not originally in the data set could be possible. Even with some amount of noise in the data set, by capturing the general pattern of the data, values for y can be produced. These values for y would not be perfect, but close enough to the real ones to be useful. Therefore, what is needed is a function approximator, as Equation 3 describes. (Emergent Gardena 2022.)

$$f(x) \approx T(x) \qquad (3)$$

An artificial neuron itself is just a function that can take any number of inputs and produces an output. Each input is multiplied by a weight and added up, plus a bias, as shown in Equation 4. Weights are real numbers that represent how important is a given input to the output. These values (weights and biases) are the parameters of the neuron, which can change while the neuron learns. One of the first type of artificial neurons that originated was the perceptron. It was developed in 1958 by psychologist Frank Rosenblatt, inspired by the works of McCulloch and Pitts. Even though today it is more common to use other types of artificial neurons, it serves as the basis for understanding neural networks. A perceptron takes several inputs x_1, x_2, ..., and produces a single output, as depicted in Figure 10. (Nielsen, M. 2019; Emergent Garden 2022)

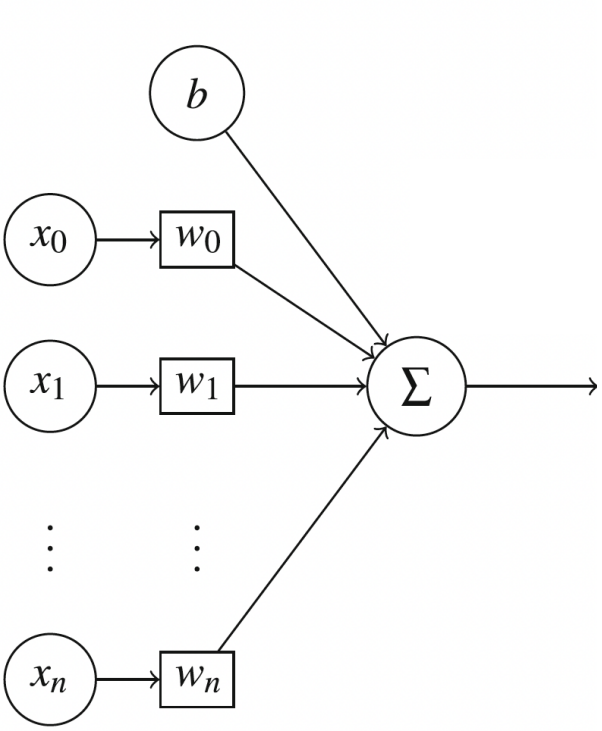


Figure 10. Graphical representation of a perceptron (Duarte, D., & Ståhl, N. 2018)

$$y = (\sum_{i=1}^{m} x_i * w_i) + b \qquad (4)$$

Where x_i is a given input, w_i is the weight of that respective input, m is the number of inputs to the perceptron, and b is a bias. Conceptually, a perceptron is a device that makes decisions weighting up evidence. (Duarte, D., & Ståhl, N. 2018; Nielsen, M. 2019.)

By this logic, it seems reasonable that more complex problems can be modelled by combining multiple perceptrons. This is where neural networks come into play. (Emergent Garden 2022.)

5.4 Deep Neural Networks

A neural network is just a collection of neurons connected through various layers. Generally, when talking about neural networks, what is being referred to is one of their main types: a fully connected feedforward neural network. A feedforward neural network is a type of artificial neural network that allows information to flow in only one direction. This means that feedforward networks do not contain loops and are, therefore, acyclical. Figure 11 depicts the typical layout of a feedforward network. A standard multilayer architecture for feedforward networks involves interconnecting multiple neurons in layers. In every layer, apart from the final output layer, each neuron is directly connected to all neurons in the next layer. A

network containing more than one hidden layer is known as a deep neural network or a multilayer perceptron. Similar to a perceptron, the output of each neuron that is passed to the following layer is the weighted sum of all inputs plus a bias. The fundamental concept of deep learning is depicted through this propagation: building increasingly complex representations using simpler, foundational concepts. (Duarte, D., & Ståhl, N. 2018.)

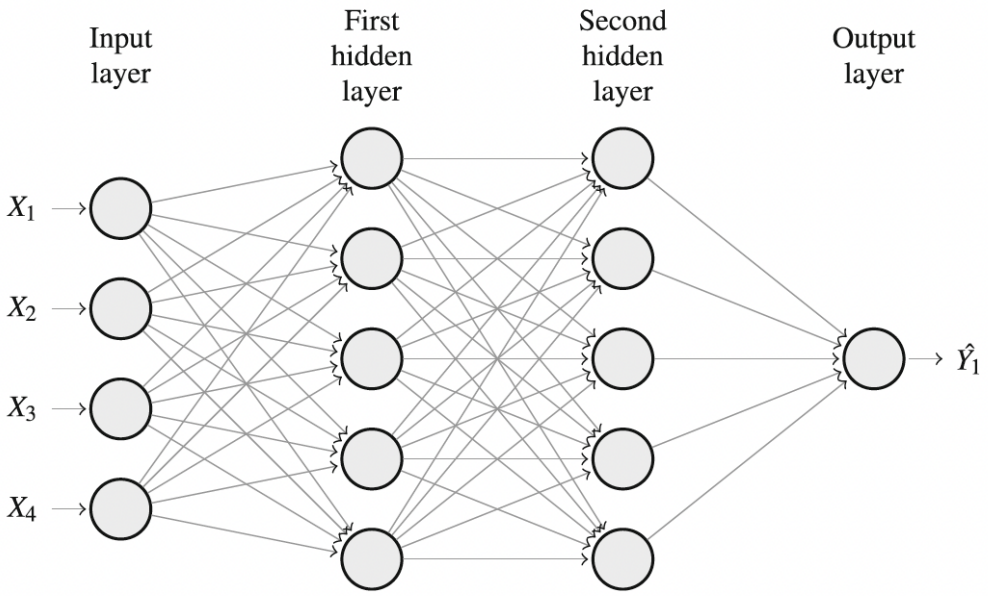


Figure 11. Layout of a deep neural network with two hidden layers, 5 neurons each (Duarte, D., & Ståhl, N. 2018)

5.5 Activation Functions

As stated in section 4.3, it seems logical that combining multiple neurons would allow to model more complex problems. However, as seen in Equation 4, a neuron is just a linear function. Combining several linear functions only results in another linear function. Thus, as seen in Figure 12, if the problem is nonlinear, like in most cases, a neural network will still perform as badly. That is why activation functions were developed. Activation functions provide the network with the necessary non-linearity for learning complex representations. (Emergent Garden 2022.)

Therefore, the mathematical definition of a neuron needs to be updated as Equation 5 shows:

$$y = f((\sum_{i=1}^{m} x_i * w_i) + b) \qquad (5)$$

Where f is the activation function.

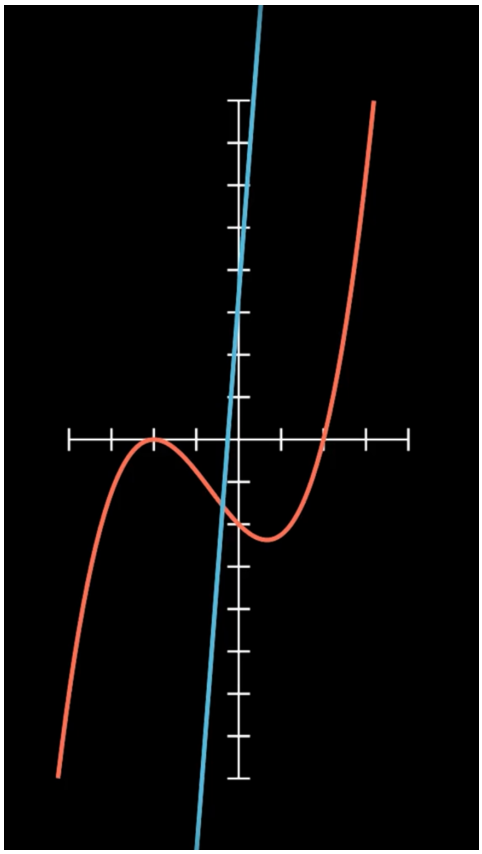


Figure 12. In red the function to be approximated, and in blue the output of the network (Emergent Garden 2022)

5.5.1 Sigmoid activation function

Initially, the Sigmoid function was the preferred activation function in neural networks. However, researchers soon discovered that its small derivative could cause the vanishing gradient problem. This led to the adoption of the Rectified Linear Unit (ReLU) as a more effective activation function. Consequently, ReLU has become the most widely used activation function in neural networks today. Despite considerable efforts to find an activation function that outperforms ReLU, none have matched its popularity, largely because of ReLU's simplicity. (Rasamoelina, A.D. et al 2020.)

The Sigmoid activation function is a nonlinear function that transforms inputs ranging from (-∞, +∞) to an output range of [0, 1]. This squashing of the output can lead to a vanishing gradient issue, particularly in deep networks. The gradient, a crucial mathematical tool used by neural networks to adjust and learn, becomes increasingly small, which complicates the optimization process. Consequently, learning becomes extremely challenging, and at times, nearly unfeasible as the network depth increases. The cause is the saturation of a sigmoid unit. As seen in Figure 13, when the function approaches 0 or 1, changes on the input

variable does not change the output, causing the saturation of the sigmoid unit. Another disadvantage is that the computation of e^{-z} can be expensive. Also, since the output is not cantered in 0, the function always outputs positive values, which causes inefficiencies on the training. (Rasamoelina, A.D. et al. 2020; Linares 2021d.)

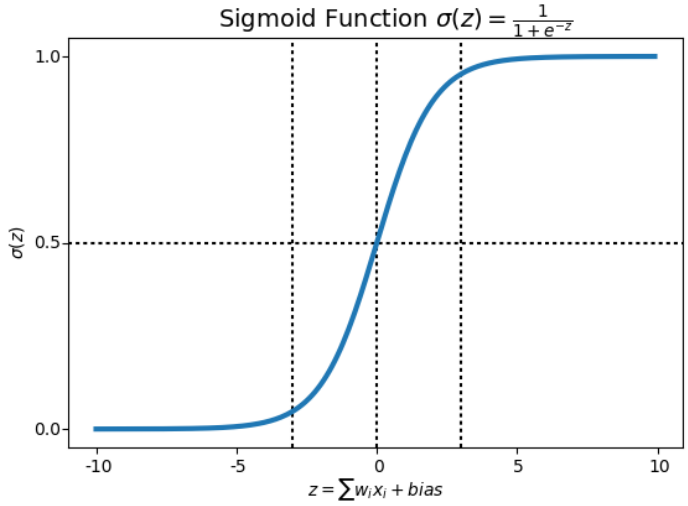


Figure 13. Output landscape of the sigmoid activation function (Pant, A. 2021)

The Sigmoid or Logistic Activation function is defined in Equation 6:

$$\sigma(z) = \frac{1}{1+ e^{-z}} \quad (6)$$

5.5.2 Rectified Linear Unit (ReLU)

ReLU (Rectified Linear Unit) is the activation function preferred by deep learning researchers. Its popularity stems from its ability to outperform other activation functions in terms of training efficiency. (Rasamoelina, A.D. et al. 2020.)

The ReLU function is defined as Equation 7:

$$ReLU(x) = max(0, x) \quad (7)$$

Figure 14 depicts the plotting of the ReLU equation into a graph.

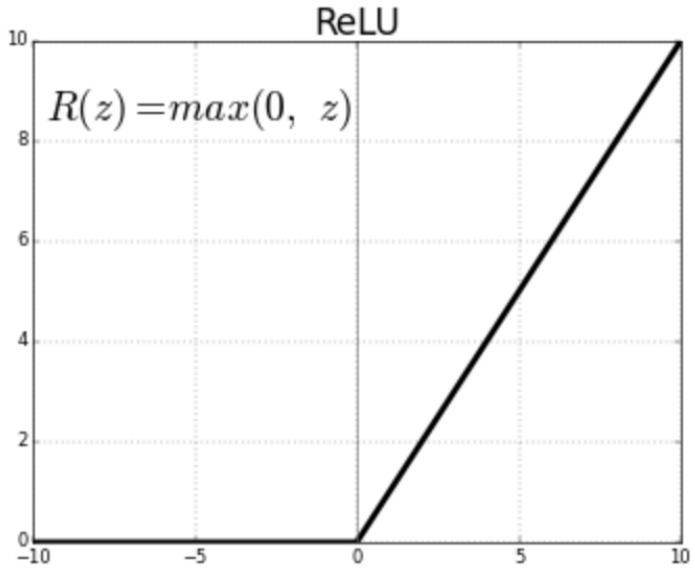


Figure 14. Output landscape of the ReLU activation function (Kathuria, A. 2018)

As shown in Equation 7 and Figure 14, ReLU is linear for positive values and outputs zero for negative values. Thus, the output values range from zero to infinity. This activation function is very efficient and do not saturate with positive values like the sigmoid function. Even so, it is still centered in 0, and sometimes can kill neurons during training. This issue arises because negative values are mapped to zero, resulting in every negative unit outputting zero. Once a neuron becomes negative, it is unlikely to recover. This however can be fixed by using a small learning rate, or using some variations of the ReLU function that were developed to address this problem. (Rasamoelina, A.D. et al. 2020; Linares 2021d.)

5.5.3 Softmax activation function

While the sigmoid function is suitable for binary classification, it is not appropriate for multi-class classification. The reason being that sigmoid outputs isolated probabilities. This works for binary classification, but in multi-class classification problems, this results in an output vector where the elements do not add up to 1, as seen on Figure 15. (Furnieles, G 2022.)

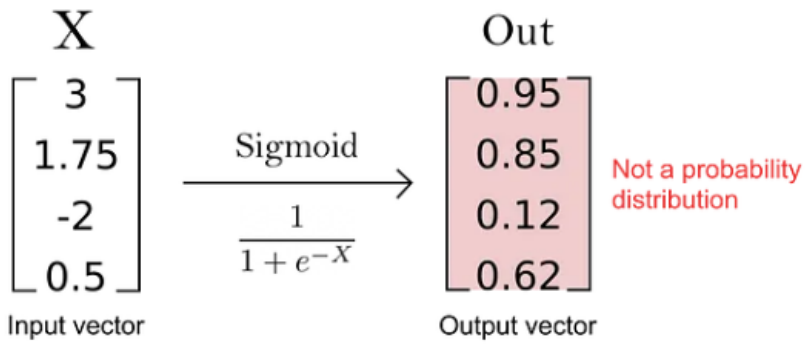


Figure 15. Sigmoid does not outputs a probability distribution because the components do not add up to 1 (Furnieles, G. 2022)

Instead, a probability distribution across all predicted classes is necessary. The softmax function serves this purpose effectively. It accepts a vector of real numbers as input and outputs another vector with the same dimensions, where each value lies between 0 and 1. Each element of this output vector represents the probability that the input corresponds to a particular class. (Furnieles, G. 2022.)

The softmax function is defined as Equation 8 indicates:

$$P(y = i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \qquad (8)$$

$P(y = i)$ represents the probability that the input falls into a specific class, with z_i being the raw score associated with that class. The denominator, which is the sum of exponentiated logits for all classes, ensures that the output forms a valid probability distribution. (Nik. 2023.)

5.6 Universal approximation theorem

George Cybenko (1989) proved that neural networks can approximate everything that can be represented as a continuous function. So, in theory, as long as the data can be codified in numbers, and the processing to be done over this data can be represented as a function, a neural network can approximate it. As such, neural networks are considered the most powerful machine learning technique, although depending on the context, other techniques may be applied. (Emergent Garden 2022.)

Of course, many limitations can prevent neural networks from learning in specific problem domains:

- The number of neurons cannot be infinite. This limits the network's size and what it can model.
- The search for the best hyperparameters of the network also puts limitations on what can be learned.
- If there is insufficient data, it does not matter how complex the model is; the approximation will be poor.
- Other limitations such as overfitting.

(Emergent Garden, 2022.)

Even so, neural networks have proven helpful in problems that would be hard to solve with conventional algorithms, as these problems may require some degree of intuition and confusing logic. (Emergent Garden, 2022.)

5.7 Training Neural Networks

As mentioned in section 4.3.1, supervised learning first entails a training phase, where the model parameters are tuned so it can learn the given dataset. In this process, it is said that the model "fits" to the dataset. In the case of neural networks, these parameters are the weights and biases, as seen in Equation 4 for a single neuron.

5.7.1 The cost function

The main goal of the training is to seek a set of values for the weights and biases that better approximates the function $f(x)$ for all the values x, as said in section 5.3. Therefore, measuring how well the neural network approximated $f(x)$ is essential. To achieve this, a cost function can be defined as in Equation 9, where a represents the output of the network, n signifies the total number of samples in our training dataset, and w and b encompass all the parameters of the neural network, with the summation of x representing the sum of all training samples. While various formulas can be utilized to compute the cost function, the most common one is Mean Squared Error (MSE), which is the one used in this case. (Linares 2021e.)

$$E(w,b) = \frac{1}{2n}\sum_{x}\|f(x) - a\|^2 \qquad (9)$$

By defining this cost function, the objective of the training is to determine the set of values for w and b that minimize $E(w,b)$. Geometrically, the cost function can be visualized as an error surface over the weight space, as depicted in Figure 16. The problem of minimizing

the cost function thus involves seeking the minimum of this error surface. An absolute minimum of the cost function, called a global minimum, corresponds to the weight vector w_a. However, other local minima may exist, such as those corresponding to the weight vector w_b . (Bishop, C. M. 1994; Linares 2021e.)

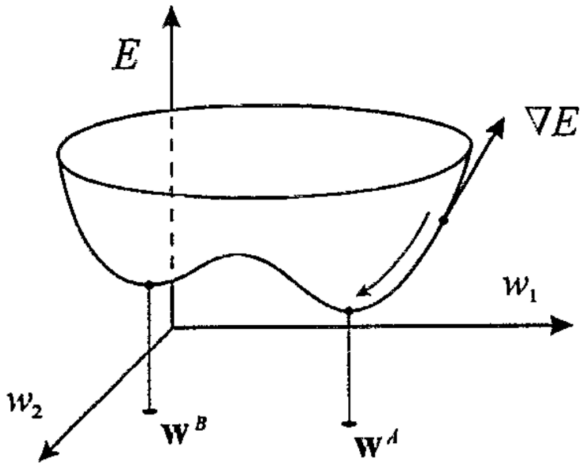


Figure 16. The cost function E(w) seen as a surface over weight space (the space spanned by the values of the weight and bias parameters $w = \{w\_1, \ldots, w\_l\}$) (Bishop, C. M. 1994).

For single-layer networks with linear activation functions, the cost function lacks local minima, allowing for a straightforward finding of its global minimum through solving a set of linear equations by analytically using the derivative: $C'(v) = 0$. In Figure 16 it is easy to locate these minimums because it is a three-dimensional function, where the network only has two parameters: w_1 and w_2. But for multilayer networks, the error function is highly nonlinear and highly dimensional regarding the weights, so calculating the derivative in this way is infeasible. Consequently, the search for the minimum typically proceeds iteratively, starting from a randomly chosen point in weight space. This random initialization is not trivial since if all weights are set to the same value, like zero or one, every hidden neuron will receive identical signals. This means that each unit in the hidden layer will be the same, regardless of the input. This hinders the learning process because most basic training algorithms are greedy, meaning they do not search for the global optimum but the "nearest" local solution. As a result, any fixed initialization will bias the solution towards a particular set of weights. To avoid the learning process getting stuck in some strange part of the error surface, the weights are randomly initialized. In machine learning terminology, this is referred to as "breaking the symmetry". (Bishop, C. M. 1994; Shayan RC 2013; Linares 2021e.)

While some algorithms converge to the nearest local minimum, others can escape local minima, potentially reaching a global minimum. Generally, the cost function surface is exceedingly complex, and achieving a good local minimum may suffice for many practical applications. (Bishop, C. M. 1994.)

5.7.2 Gradient Descent Algorithm

As stated before, traditional calculus methods are not applicable to multilayer perceptrons. Fortunately, there is an elegant analogy that proposes a rather effective algorithm. If one glance at the Figure 16, it is possible to picture the cost function surface as a valley. Now, if a ball was rolling down the slope of this valley, common experience suggests that the ball will eventually settle at the bottom. Could this concept be used to locate a minimum for the function? One might start by selecting a random starting point for an imaginary ball and then simulate its motion as it rolls down to the valley's bottom. This simulation could be accomplished by computing derivatives (and possibly some second derivatives) of the cost function. These derivatives would provide insight into the local "shape" of the valley and guide the ball's motion. So, what laws could be imposed to ensure it always rolled to the valley's bottom? To refine that question, it is necessary to define what the gradient is. The gradient, symbolized as ∇, is a vector whose components are the function's first-order partial derivatives with respect to its variables. The gradient vector provides directional guidance toward the most rapid increase in a function's value. Essentially, it acts as a compass, indicating the direction of maximum growth. This gradient vector is marked as ∇E in Figure 16. (Nielsen, M. 2019; The Math Sorcerer, 2020.)

Therefore, the gradient for the cost function in Figure 16 would be defined as Equation 10 shows:

$$\nabla E \equiv \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}\right) \quad (10)$$

But because the aim is to minimize E, what is interesting is the negative of the gradient, which would indicate the direction of minimum growth. By defining Δw to be the vector of changes in w, $\Delta w \equiv (\Delta w_1, \Delta w_2)$, and transposing both this vector and Equation 10, it is possible get a new Δw that further minimizes E using Equation 11:

$$\Delta w = -\eta \nabla E \quad (11)$$

Where η is a small, positive number known as the learning rate. This Equation 11 can then be used to update the parameters of the network as shown in Equation 12:

$$w \rightarrow w' = w - \eta \nabla E \qquad (12)$$

If this update rule is used iteratively, E will keep decreasing until, hopefully, reaching a local or global minimum. This is the gradient descent algorithm. To put it briefly, the gradient descent algorithm operates by calculating the gradient ∇E multiple times and then going in the opposite direction. This results in "falling down" the slope of the valley. (Nielsen, M. 2019.)

To ensure gradient descent to function properly, a small enough learning rate must be selected. At the same time, η should not be too small, as this would cause the changes Δw to be insignificant, slowing down the gradient descent algorithm. In practice, the value of η may be varied while the network is learning to achieve a good approximation without causing the algorithm to be too slow. (Nielsen, M. 2019.)

Several challenges appear when applying the gradient descent rule. To understand one of these issues, revisiting the quadratic cost outlined in Equation 9 is crucial. This cost represents an average over costs for individual training examples. In practical terms, computing the gradient ∇E requires computing the gradients ∇E_x separately for each training input, x, and then averaging them. Unfortunately, when dealing with a large number of training inputs, this process can be computationally expensive, leading to slow learning. To speed up learning, an approach known as stochastic gradient descent can be employed. This method estimates the gradient by computing ∇E_x for a small sample of randomly selected training inputs. A reliable estimate of the true gradient ∇E_x can be obtained by averaging over this sample, facilitating a faster gradient descent, and learning. (Nielsen, M. 2019.)

To elaborate further, stochastic gradient descent operates by randomly selecting a small number m of training inputs, forming what is called a mini-batch. Training proceeds by selecting and utilizing a randomly chosen mini-batch of training inputs, and repeating this process until all training inputs have been exhausted, completing an epoch of training. Subsequently, a new training epoch starts. (Nielsen, M. 2019.)

5.7.3 Backpropagation Algorithm

However, how the actual gradient of the cost function is calculated is still to be explained, and this is not a trivial matter. The algorithm that does such a calculation is called backpropagation. While the backpropagation algorithm was initially introduced in the 1970s, its significance was not fully recognized until the publication of a renowned paper in 1986 by David Rumelhart, Geoffrey Hinton, and Ronald Williams. This paper presents various neural

networks where backpropagation significantly outperforms earlier learning approaches, enabling the solution of previously unsolvable problems using neural networks. Currently, the backpropagation algorithm is the cornerstone of learning in neural networks. At the core of backpropagation lies an expression for the partial derivative ∂E/∂w of the cost function E concerning any weight w (or bias b) in the network. This expression gives information about the rate at which the cost changes when modifying the weights and biases. The algorithm applies a concept in calculous called the chain rule, propagating the gradient recursively and backwards through the network, giving the algorithm its name. (Nielsen, M. 2019; Linares 2021e.)

5.8 Convolutional Neural Networks (CNNs)

CNNs or Convolutional Neural Networks are similar to traditional ANNs as they consist of self-optimizing neurons that learn through experience. However, CNNs are primarily used to recognize patterns within images. One significant difference between the two is that the layers within the CNN are made up of neurons organized into three dimensions, including height, width, and depth. There are three types of layers within a CNN: convolutional, pooling, and fully connected. (O'Shea, Keiron & Nash, Ryan 2015.)

5.8.1 Convolutional layer

The convolutional layer plays a vital role in the functioning of Convolutional Neural Networks (CNNs), as indicated by the network's name. During a convolution process, a filter moves over the image's pixels to identify and capture specific features. (O'Shea, Keiron & Nash, Ryan 2015; Bisong, E. 2019.)

To understand the convolution layer, it is important to define what convolution means. Convolution is a technique for extracting specific information from a matrix by applying a function to it. The function operates as a sliding window moving across the matrix and is often referred to as either a convolutional filter or a kernel, with both terms being used interchangeably in research. In Figure 17, the process is illustrated with a filter sliding through the matrix to extract information. (Bisong, E. 2019.)

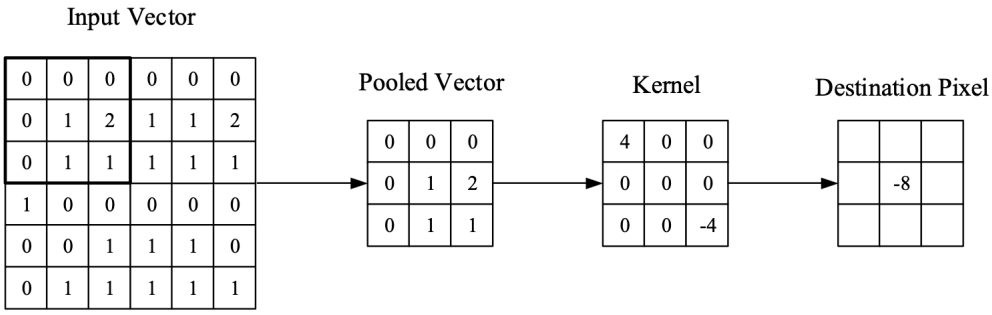


Figure 17. An illustration of a convolutional layer. The input vector is covered by the kernel's central element, and a weighted sum of the nearby pixels and itself is calculated and replaced (O'Shea, Keiron & Nash, Ryan 2015)

Filters in a convolutional layer function as neurons and are allocated specific weights. As previously mentioned, they operate through a sliding window mechanism. The result of applying a filter is known as a feature map. These filters also serve as neurons and include a non-linear activation function. The input to a filter might either be the pixel matrix of an image at the input layer or the feature maps from an earlier convolutional layer when the filter is employed in deeper layers of the network. (Bisong, E. 2019.)

Filters are designed with a predefined square input size, often a 3 x 3 size, which corresponds to their local receptive field. Usually, researches use a larger number of filters in the deeper layers of the network while limiting their number at the input layer. Each cell within the filter has a specific weight, and these weights are used to multiply the corresponding pixel intensities of the input. The products of these multiplications are then summed to populate the appropriate cell in the convolutional output. The configuration of these weights dictates the operation of the filter and, as a result, the type of features that are extracted. Various filters specialize in different functions, such as detecting edges and lines. (Bisong, E. 2019.)

Key considerations to make when designing a convolutional layer are

- the filter size
- the stride of the filter
- the padding for the layer input.

(Bisong, E. 2019.)

The filter's stride specifies the number of pixels the filter moves from one image activation to the next. Using a stride of 1 is common, but it can be increased for large images. When the filter size and stride are chosen, they may not evenly divide the input's size. To prevent

the loss of pixel information, zero padding is used to add a layer of zeros to the image pixels' borders. This includes the zeros in the convolution and enables the filter to move evenly through all pixels in the image. (Bisong, E. 2019.)

Feature maps represent the outputs produced by filters in a convolutional layer, highlighting specific patterns within the input image, such as horizontal lines, vertical lines, and edges. These feature maps, when combined from various neurons, constitute a convolutional neural layer. This configuration allows the layer to recognize and learn complex patterns and features within an image. (Bisong, E. 2019.)

5.8.2 Pooling layer

Pooling layers are typically placed after one or more convolutional layers. Their primary role is to reduce or downsample the feature map produced by the convolutional layer. This downsampling is a form of summarization of the image features captured in earlier layers of the network, which contributes to preventing overfitting. Moreover, by reducing the input size, pooling layers help to lower processing and memory demands during network training. Essentially, a pooling layer acts as an aggregation function that compiles and extract key features from previous layers. In contrast to convolutional layers, pooling layers do not perform any multiplicative transformations on the input feature maps. Common aggregation functions used in pooling include max, sum, and average, with max pooling being the most frequently utilized in practice. (Bisong, E. 2019.)

The aggregation functions in the pooling layer function similarly to filters, like those in convolutional layers. They operate within a receptive field, which is generally smaller than that of the convolutional layer, and use a defined stride width. However, unlike convolutional filters, the filters in the pooling layer, also viewed as neurons, do not possess any weights or biases. (Bisong, E. 2019.)

5.8.3 Fully connected layer

The layer performs the same tasks as those found in standard NNs, trying to generate class scores based on the activations, which can be used for classification purposes. (O'Shea, Keiron & Nash, Ryan 2015.)

To input an image into the fully connected network (FCN), the image matrix needs to be flattened. For instance, an image matrix sized at 28 x 28 x 3 would be converted into 2352 input weights, with an additional bias value of 1, before being fed into the FCN. (Bisong, E. 2019.)

5.8.4 Advantages of CNNs

When using a standard neural network, there must be as many input neurons as the image's pixels, flattening the image into a one-dimensional array of pixels. Therefore, if the input images were 300 by 300 pixels, with three channels (RGB), the neural network would need 270000 input neurons. That means that each neuron in the second layer, would have 270001 parameters that must be trained. (Linares 2021b.)

However, in CNNs, the number of parameters does not scale with the input's spatial dimensions (x and y). This occurs because the parameters of the convolutional layers, which are the kernels or filters, typically maintain fixed dimensions and can be applied to inputs with varying spatial dimensions by using appropriate padding. It is important to note that padding may enlarge feature maps, but these maps are not the parameters of the neural network; rather, they are the outputs of the convolutional layers. This might be the source of confusion when observing a CNN diagram, as larger feature maps may suggest an increase in parameter count. (nbro, 2020.)

6 Optical Character Recognition

6.1 Introduction to OCR

Whenever encountering the term Handwritten Character Recognition, the first concept that typically comes to mind is OCR, short for Optical Character Recognition. OCR is a technology that decodes characters and translates them into a format machines can understand. (Beohar, D., & Rasool, A., 2021.)

Recognizing isolated handwritten digits has been a longstanding area of research, yet it remains a focal point both academically and in commercial settings. This interest stems from its practical applications such as automated form processing and handwritten postal code identification. Recognition systems' effectiveness heavily depends on the classification methods employed, given that writings vary across languages and scripts. Therefore, developing these systems was particularly challenging before modern AI technologies. Previously, designing a classifier required manual feature extraction, and machines made decisions based on the programmed features. However, technological advancements have led to the emergence of powerful deep learning algorithms such as Artificial Neural Networks, Convolutional Neural Networks, and Recurrent Neural Networks, which now deliver exceptional results. (Daniel Keysers, 2007; Beohar, D., & Rasool, A., 2021.)

6.2 Stages of OCR

OCR involves a five-stage process: Image Acquisition, Pre-processing, Segmentation, Feature Extraction, and Classification, with each stage playing a critical role:

- Image Acquisition: This initial stage focuses on collecting, filtering, and cleaning images before further processing.
- Pre-Processing: This crucial step involves cleaning images to minimize noise and eliminate unwanted data. It optimizes images by filling gaps and straightening lines and includes algorithms for correcting skew. The outcome of this stage is a binary image achieved through binarization and texture filtering.
- Segmentation: This involves breaking down an image into smaller segments. Segmentation can be categorized into line, word, and character types. Line segmentation splits an image into individual lines, word segmentation breaks it into words, and character segmentation divides words into characters.

- Feature Extraction: An essential part of dimensionality reduction, this stage simplifies data for easier processing. For large datasets like MNIST, it enhances the efficiency of the recognition process by removing redundant data while preserving essential features. As said in section 5.2, this process is carried out automatically by deep learning models.
- Classification: The final decision-making stage in the recognition process, where the output from feature extraction is classified.

(Beohar, D., & Rasool, A., 2021.)

6.3 The MNIST dataset

The MNIST dataset is an excellent instance of the problem of handwritten digit classification. Created by the National Institute of Standards and Technology (NIST), this dataset includes 60,000 training images and 10,000 test patterns, each 28x28 pixels with 256 gray levels. The challenge of recognizing these digits is generally not considered 'difficult' because the human error rate is low, around 0.2%, and the large volume of training data helps machine learning models to generalize effectively. Some examples from the MNIST dataset can be seen on Figure 18. (Daniel Keysers, 2007; Beohar, D., & Rasool, A., 2021.)

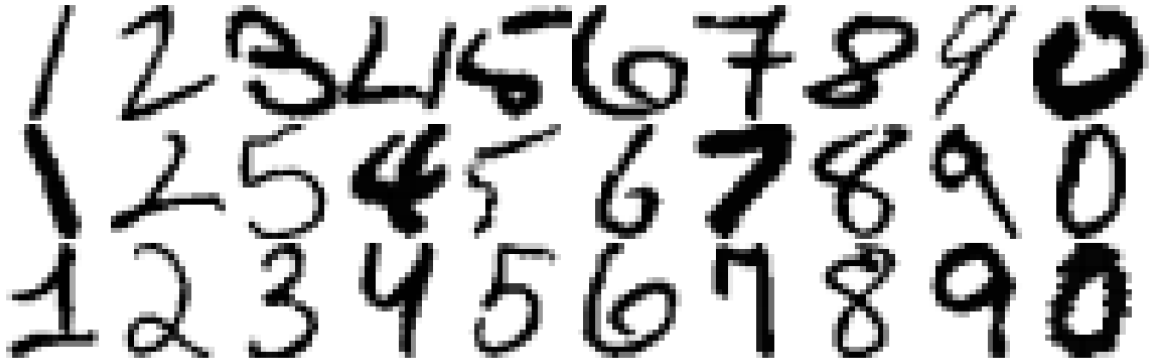


Figure 18. Samples from the MNIST dataset (Daniel Keysers, 2007)

7 Common tools for AI applications

7.1 Programming languages used in AI

Python is a top choice for data science in various industries mainly due to its straightforward syntax and abundant machine learning/deep learning libraries written for this language. These libraries facilitate the development of data science solutions without the need to learn the intricacies of specific algorithms or techniques. With a vast community of developers, these packages are continually enhanced and maintained through collaborative efforts. (Bisong, E 2019.)

Java is a powerful programming language extensively used in various software development contexts, especially notable in the mobile app sector. It is preferred by AI developers for several reasons, including its ease of debugging, user-friendliness, and maintainability. Its compatibility across different platforms ensures that AI-driven projects can be implemented on multiple devices. A key example of an AI project using Java is Deeplearning4j (DL4J), a leading open-source deep learning library. (Anoriega 2022.)

C++, on the other hand, is a programming language celebrated for its speed and efficiency, qualities that make it ideal for machine learning and neural network projects where performance is crucial. Although it may be more complex to program in than some other languages, C++'s execution speed makes it well-suited for performance-intensive applications. It is often used alongside other languages to create AI-focused software. A prominent example of a tool that integrates C++ for AI applications is OpenCV. This library is among the most complete collections of machine learning and computer vision algorithms available, enabling capabilities such as object identification, face recognition, 3D scanning of objects, and more through its advanced computer vision algorithms. (Anoriega 2022.)

JavaScript is the most popular programming language globally, according to GitHub rankings. This status is hardly surprising given its significant role in shaping the modern web, facilitating much of the interactivity prevalent in everyday websites. JavaScript is a good choice for artificial intelligence due to its array of high-level tools and libraries for machine learning tasks. Notably, TensorFlow.js is an exemplary tool capable of running directly within the browser. This capability opens up many possibilities for web developers, enabling the creation of browser-based AI applications. (Anoriega 2022.)

Numerous programming applications demand the integration of two or more programming languages, typically pairing one with high performance, like C++, and another that simplifies programming complexities, such as Python. While this approach proves functional, it often creates a conflict between performance and ease of use. Conceived at MIT in 2009, Julia

sought to address some of these challenges. Julia stands out in AI programming due to its built-in package manager and strong support for parallel and distributed computing. It is especially valuable for scientific computing and data analysis. Its parallelism capabilities, which enable multiple processes to run concurrently, are highly relevant for machine learning and AI applications. As a result, Julia's significance in the field is expected to increase. (Anoriega 2022.)

7.2 Python libraries for AI applications

NumPy is an open-source Python library created in 2005 tuned for numerical computations. It closely resembles MATLAB in its functionality and power, especially when paired with complementary packages like SciPy for scientific operations, Matplotlib for visualizations, and Pandas for data analysis. Abbreviated as "numerical python," NumPy excels in its capacity to handle and manipulate n-dimensional arrays, a crucial feature for developing machine learning and deep learning models. Since data is commonly structured as a matrix-like grid with rows representing observations and columns indicating variables or features, NumPy's 2-D array perfectly accommodates the storage and manipulation of datasets. (NumPy - About us; Bisong, E 2019.)

Pandas is a specialized Python library for data analysis, particularly excelling with large datasets. It was created in 2008 at AQR Capital Management. It offers user-friendly features for handling tasks such as data reading and writing, managing missing data, reshaping datasets, and manipulating data through slicing, indexing, inserting, and deleting variables and records. In summary, Pandas is the essential tool for data cleaning and exploration tasks. (pandas - Python Data Analysis Library; Bisong, E 2019.)

Before applying a machine learning algorithm or any other analytical technique, it is crucial to make observations of the variables within a dataset. Data visualization is a fundamental tool for comprehending the dataset and extracting insights into its underlying structure. These insights help scientists to determine suitable statistical analyses or choose the most appropriate learning algorithms for the dataset. Additionally, visualization offers valuable insights into potential transformations that could enhance the dataset. Matplotlib is a fundamental graphics package for data visualization within Python, playing a central role in the Python data science ecosystem and seamlessly integrating with NumPy and Pandas. Its pyplot module closely mirrors MATLAB's plotting commands, facilitating a smooth transition to Python-based plotting for MATLAB users. (Bisong, E 2019.)

In contrast, Seaborn extends the capabilities of Matplotlib, offering a simpler set of methods for generating visually appealing graphics in Python. Seaborn is particularly well-suited for working with Pandas DataFrames, providing enhanced integration. (Bisong, E 2019.)

7.3 Machine Learning libraries for Python

TensorFlow (TF) is an open-source library created by Google in 2015, specialized numerical computation intended for machine learning applications. It has garnered widespread adoption among machine learning researchers and industry professionals for its efficacy in developing deep learning models and architectures. It is the preferred tool for deploying trained models into production servers and software products. TensorFlow functions as an interface for designing machine learning algorithms and as an implementation tool for executing them efficiently. It provides exceptional versatility, enabling computations defined in TensorFlow to run on a wide range of systems, from mobile devices like phones and tablets to large-scale distributed systems consisting of hundreds of machines and thousands of computing devices. TensorFlow has proven invaluable for both research and production deployment of machine learning systems across many fields, such as speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and computational drug discovery. TensorFlow essentially lets users create mathematical functions on tensors (hence the name), which are multidimensional arrays of numbers akin to matrices or vectors but extending to any number of dimensions. It does this using computational graphs, allowing users to define and compute gradients of these functions. Regarding capabilities, TensorFlow offers what NumPy does, but with the added advantage of GPU acceleration and automatic differentiation. TensorFlow Lite (TFLite) is a library that allows developers to implement on-device machine learning across mobile, embedded, and IoT devices through a comprehensive set of tools. Optimized for on-device machine learning, its key features prioritize latency, privacy, connectivity, and power consumption. The framework supports various platforms, including Android and iOS devices, embedded Linux, and microcontrollers. Additionally, TensorFlow Lite offers built-in support for multiple programming languages such as Java, Swift, Objective-C, C++, and Python. (Martín Abadi et al. 2015; Bisong, E, 2019; Ketkar, N.S. 2021; Seeed Studio Wiki 2023.)

Keras is a deep learning API initially operated independently from TensorFlow, serving as an interface for model creation with TensorFlow as one of its backend frameworks. However, with the release of TensorFlow 2.0, Keras became a built-in part of the TensorFlow codebase, now the favored high-level API for deep learning tasks. (Bisong, E 2019.)

PyTorch is a relatively new addition in the deep learning framework landscape. It provides a Python interface to the Torch engine, originally based on Lua, to define mathematical functions and compute their gradients. Unlike frameworks like TensorFlow that follow a define-compile-run model (where users write mathematical expressions in a computational graph format that gets compiled for execution), PyTorch adopts a define-by-run approach. This dynamic nature eliminates the need for compilation, allowing users to define expressions and compute gradients directly. Pytorch code often appears more intuitive and closely resembles the mathematical descriptions of the network compared to TensorFlow. Debugging is also more straightforward with PyTorch due to its dynamic structure. In contrast, debugging in TensorFlow requires navigating two layers of abstraction: the Python code for building the computational graph and the compiled graph itself. However, it's worth noting that TensorFlow's define-compile-run paradigm enables greater optimization of underlying computations. (Ketkar, N.S. 2021).

7.4 Frontend

In web development, the frontend comprises all technologies that run on the client side. However, it does not imply ignorance of Backend workings (server-side), as understanding the backend is necessary for consuming data and structuring the UI layouts effectively for user comfort. Frontend is responsible for styling the page so that it can present information in a user-friendly manner. The frontend developer must be acquainted with user experience techniques to enhance interaction between the user and the visited page and also possess knowledge of interaction design to position elements for quick and comfortable user navigation. Numerous technologies must be known to the frontend developer. For instance, JavaScript has frameworks like Angular and React Native. For other languages, Flutter or Pynecone are some notable examples. (Pérez Ibarra, S. G et al. 2021.)

React Native

Developed by Facebook, React Native originated from an internal hackathon aimed at streamlining iOS and Android development processes. Initially introduced in 2015, React Native has evolved into an open-source framework, with contributions not only from Facebook but also from individual developers and notable companies like Samsung and Microsoft. Categorized as an interpreted cross-platform framework, React Native employs the standard native rendering API of the target platform to render UI components. Utilizing JavaScript interfaces, React Native applications can access platform-specific features such as the phone's microphone or camera. Built upon Facebook's React JavaScript library for building user interfaces, React Native applications, like React applications, are written using a combination of JavaScript and JSX. The primary distinction between React Native and

React is their target platforms, with React Native focusing on mobile platforms rather than browsers. (Hjort, E. 2020.)

Flutter

Flutter is a cross-platform framework designed for the development of mobile applications. Google publicly introduced Flutter in 2016, and it stands out as Google's chosen application-level framework for its Android OS. What sets Flutter apart is its reliance on the device's widgets instead of utilizing web views. Utilizing its own high-performance rendering engine, Flutter renders each view component independently, offering the potential to create applications with performance levels comparable to native ones. In Flutter, every application is developed using Dart, a programming language developed and maintained by Google. Dart is extensively utilized within Google and has demonstrated its capability in building large-scale web applications like AdWords. Originally intended to replace JavaScript, Dart incorporates many key features of JavaScript, including the "async" and "await" keywords. However, Dart adopts a syntax reminiscent of Java to appeal to developers unfamiliar with JavaScript. In terms of architecture, during compilation, Dart code is transformed into native code. Flutter's hot reload feature, known as stateful hot reload, significantly enhances the development cycle by allowing developers to make changes and instantly see them reflected without altering the application's structure. This is achieved by sending updated source code to the running Dart Virtual Machine (Dart VM), ensuring that the application's transitions and actions remain intact after reloading. Flutter applications refresh the view tree with each new frame, a process distinct from many other systems that utilize reactive views. While this approach ensures consistency, it presents a drawback: numerous objects, even those needed for a single frame, are created. Even though, with Dart's modern design, Flutter optimizes memory management through "Generational Garbage Collection" to handle such scenarios efficiently at the memory level. (Tashildar, A., et al. 2020.)

7.5 Back end

The backend refers to the data access layer of a software inaccessible to the end-user, housing the application logic for handling data. The backend developer operates on the server side and must be skilled in web application or cross-platform application development. They must understand interactions with different databases, discerning the differences and qualities of commonly used ones. This does not mean that a backend developer should completely disregard frontend work but rather possess the necessary knowledge for effective teamwork, as both roles complement each other. The backend developer needs expertise, depending on their workplace, in server-side languages like Java, C#, PHP,

Node.JS, among others, as well as those interacting with databases such as MySQL, PostgreSQL, SQLServer, MongoDB, and others. Some backend technologies include Next.JS for Javascript, or Flask and FastAPI for Python. (Pérez Ibarra, S. G et al. 2021.)

Flask

Flask is a WYSIWYG (what you see is what you get) web application framework renowned for its simplicity and ease of use while still capable of handling complex applications. Initially conceived as a modest wrapper around Werkzeug and Jinja, it has evolved into one of the most sought-after Python web application frameworks. One of Flask's notable characteristics is its non-intrusive approach. While it offers recommendations, it refrains from imposing specific dependencies or project structures on developers. Instead, it empowers developers to handpick the tools and libraries that best suit their needs. Additionally, the Flask community provides many extensions, simplifying the process of incorporating new functionalities into applications. (Pallets.)

FastAPI

FastAPI is as a web framework aimed at crafting APIs using Python 3.8 and above. Noteworthy among its attributes is its remarkable speed, a trait comparable to industry benchmarks like NodeJS and Go, facilitated by the integration of Starlette and Pydantic. Ease of use remains a fundamental aspect of FastAPI's philosophy, highlighted by its user-friendly design and minimal learning curve. By prioritizing simplicity, developers can focus more on actual implementation than navigating extensive documentation. Furthermore, its concise syntax mitigates code duplication, maximizing efficiency and minimizing the likelihood of bugs. (Tiangolo.)

8 Practical case: Kanji recognition mobile app

8.1 Introduction

Now that I have established some of the foundational concepts, I will describe the development of a practical case study that focuses on a mobile application that recognizes kanji characters through the integration of AI techniques.

The idea of the practical case came to me when I first started learning Japanese. Searching information for specific characters when I did not have the option to copy and paste them into the search bar was difficult. For example, in Jishoo.org, a common Japanese dictionary, you either have to search by radicals or by drawing. If I was walking around Japan, for example, and I saw a character I did not know, it would be convenient to have a way to search for information about that character quickly. Thus, the main objective of the practical case is having a quick way to get relevant information about these characters without going through all the troubles of searching for them in conventional Japanese dictionaries.

The use of AI for this case is justified because recognizing these characters in images is infeasible through conventional programming. As I said in section 2.1, there are over 2000 daily-use characters. How can someone, using regular logic and programming constructs, develop an algorithm to differentiate between all of them? It is not viable. This is where AI comes into play. As said in the theoretical case, AI, particularly deep learning, works very well for tasks that are hard to program and with unstructured data such as images.

8.2 The model

My first step in developing the practical case was to search for a model trained to recognize kanji. Nowadays, there is almost an AI model for everything, so I preferred to get a working app as fast as possible by looking for a pre-trained model instead of reinventing the wheel and training one by myself. My search concluded with two possible candidates:

- manga-ocr by kha-white.
- DaKanji-Single-Kanji-Recognition by CaptainDario.

Manga-ocr is an optical character recognition for Japanese text, with the main focus being Japanese manga. The first version of the backend was actually developed with this model, and it worked well. As the repo says, the model focuses on recognizing text from manga, but it performed well in most cases with pictures of street signs, which is the primary cause I thought for my application. However, I also wanted my application to recognize handwritten text, and the repo itself states that *it probably won't be able to handle handwritten text*

though. This was one of the reasons why I discarded this model, but the other reason was that this model uses a Transformer. Transformers is one of the most powerful AI techniques nowadays, and it is actually what the famous ChatGPT uses under the hood (Toews, R. 2024). The problem was that because Transformers are complex models, I would have had a hard time understanding them and explaining them in the theoretical case. Therefore, I decided to use the second model. (Kha-White.)

DaKanji-Single-Kanji-Recognition by CaptainDario is a model for recognizing single kanji characters. It is less powerful than the other model in that sense, as manga-orc could recognize entire lines of text at once. Regardless, the use case of my app was to provide information about a single kanji Character, so DaKanji still fitted my needs. The biggest advantage was that DaKanji is a CNN, so I could understand how it worked. In fact, because the repo provided the code used to train the model, one can notice that it is using a network architecture called EfficientNetLite, as seen on Figure 19. (CaptainDario.)

```
from efficientnet_lite import EfficientNetLiteB0

eff_net_lite = EfficientNetLiteB0(
    include_top=True,
    weights=None,
    input_shape=(64, 64, 1),
    classes=len(ls),
    pooling="avg",
    classifier_activation="softmax",
)

f16_model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(None, None, 1)),
    img_augmentation,
    eff_net_lite,
])
f16_model.summary()
```

Figure 19. DaKanji uses the EfficientNet architecture (CaptainDario)

EfficientNet is a convolutional neural network architecture designed specifically for image recognition and is currently regarded as one of the leading frameworks in this field. Developed by Mingxing Tan and Quoc V, this architecture is the product of several years of comprehensive research and incorporates multiple innovative techniques. At its core, EfficientNet utilizes inverted residual blocks from the MobileNetV2 architecture along with the MnasNet search strategy. Although these smaller blocks were not present when MnasNet was initially designed, their incorporation has significantly enhanced the performance of the network models derived from this research. (Tan, M., & Le, Q. 2019; Koonce, B., & Koonce, B. 2021.)

Testing the model

Before doing any serious work, I set up this simple Google Collab notebook for testing the model and seeing how it performs with some images of my own. As said in section 4.3.1, when doing predictions, is necessary to feed the model the same features it was trained on. Looking at the Python notebook the author used to train the model, it looks like he used 64 by 64 grayscale images of the characters over a black background (CaptainDario). So, before feeding my own images into the model, I had to write code in order to apply the same effects to my images, as seen on Figure 20. In this piece of code, I first resize the image to be 64 by 64 and then I normalize the pixel values to be between 0 and 1. Normalizing the input values fed into the network is important because when a feature within a dataset significantly outweighs others in scale, it can become dominant and influence the predictions made by a neural network, leading to inaccuracies. Also, as explained in section 5.3, during forward propagation, neural networks compute outputs by taking the dot product of weights with input features. When input values are excessively high, calculating the output requires considerable computation time and memory resources. By normalizing inputs, neural networks can operate more efficiently, leading to faster convergence and improved prediction accuracy. The rest of the lines just reshape the image to a suiting shape for feeding into the model. (user11530462 2020.)

```
def preprocess_image(image, display=False):
    image = cv2.resize(image, (64, 64), interpolation=cv2.INTER_LINEAR)

    # Invert the image
    image = 255 - image

    # Convert to numpy array
    image_array = np.array(image, dtype=np.float32)

    if display:
      plt.imshow(image_array, cmap='gray')
      plt.axis('off')
      plt.show()

    image_array = np.expand_dims(image_array, axis=0)
    image_array = np.expand_dims(image_array, axis=-1)

    return image_array
```

Figure 20. Python function for pre-processing images before feeding them into the model

However, this function was not enough to achieve good results. With this version, the model did not work at all. I had to apply three fixes to get a reasonable accuracy. The first fix, seen in Figure 21, was introducing code to decide whether the image should be inverted or not, so I did not have to do this manually. This fix was more of a quality-of-life improvement than something to improve accuracy. This function takes an image as input along with optional parameters for border size and a threshold value. It decides whether the image should be

inverted or not based on the average pixel intensity of its borders. The second fix was the most impactful of all. I noticed that after inverting the images, some grey pixels were still left on the background of the images. I figured this could be confusing the model, so I coded a function for cleaning the image and left the background of the images completely black. The function applies a threshold to the input image, converting it into a binary mask where pixels with intensity values greater than or equal to 128 are set to 255 (white), and pixels with intensity values less than 128 are set to 0 (black). Next, it applies the binary mask to the inverted input image using bitwise AND operation, effectively removing the parts of the image where the binary mask is black. Finally, I applied a sharpen filter over the image to help the neural network. With all these fixes, I managed to have a relatively good working model, as seen in Figure 22.

```python
def should_invert(image, border_size=5, threshold=127.5):
    # Extract pixels from the top, bottom, left, and right borders
    top_border = image[:border_size, :]
    bottom_border = image[-border_size:, :]
    left_border = image[:, :border_size]
    right_border = image[:, -border_size:]

    # Compute the average pixel intensity for each border
    avg_top = np.mean(top_border)
    avg_bottom = np.mean(bottom_border)
    avg_left = np.mean(left_border)
    avg_right = np.mean(right_border)

    # Compute the average intensity across all borders
    avg_intensity = (avg_top + avg_bottom + avg_left + avg_right) / 4.0

    # If average intensity is closer to 0, no inversion needed
    if avg_intensity < threshold:
        return False
    else:
        return True

def clean(image):
    # Apply threshold to create binary mask
    _, binary_mask = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)
    # Apply the binary mask to the inverted image
    cleaned_image = cv2.bitwise_and(image, binary_mask)
    return cleaned_image

def sharpen(image):
    sharpen_kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    image = cv2.filter2D(image, -1, sharpen_kernel)
    return image
```

Figure 21. Python functions used in the pre-processing of the images

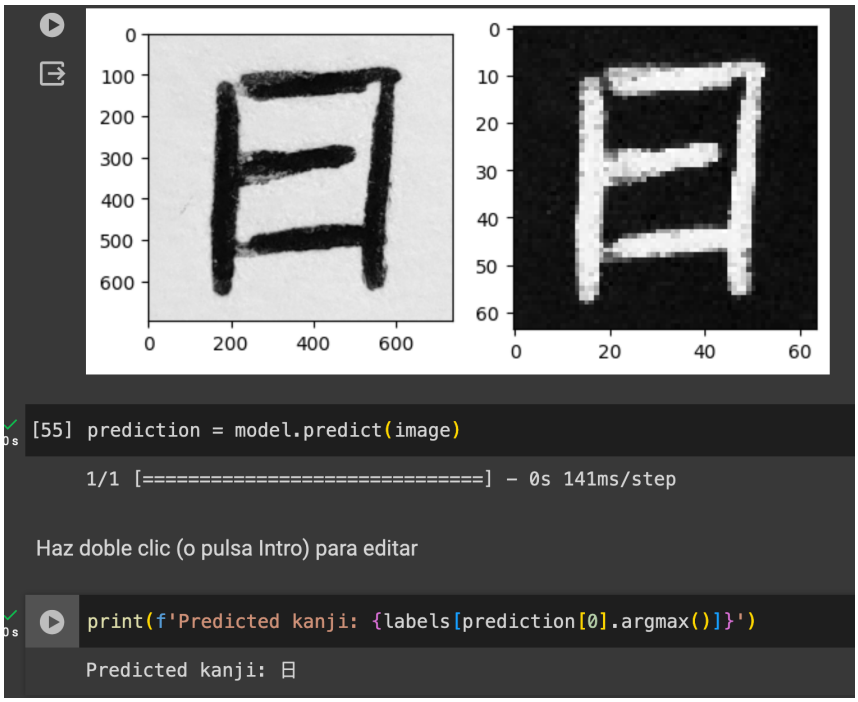


Figure 22. The model can be seen to be working correctly

8.3 Making a test application

Now that I know the model works, it is time to integrate it into a mobile application. Of course, I needed some kind of framework for developing mobile applications. In this case, I decided to use Flutter as I had already developed Flutter applications before, and it allowed me to develop mobile applications with little difficulty. In all my projects, I always start with something small that I can build upon after things start to work, so I started developing a very basic application. For this first version, I placed a button in the middle of the screen to select a picture to send to the backend so a prediction can be made. After the app gets back the result, it displays it in a text below the button, as seen in Figure 23. For the backend, I chose Python as the server programming language. It was the easiest option for loading and using the model in an API. Furthermore, using FastAPI, getting an API working to make predictions only took a couple of hours. The functionality is straightforward and can also be seen on Figure 23 as a flowchart. The server receives an image over the network as a request, pre-processes the image, makes a prediction, and returns the resulting character.

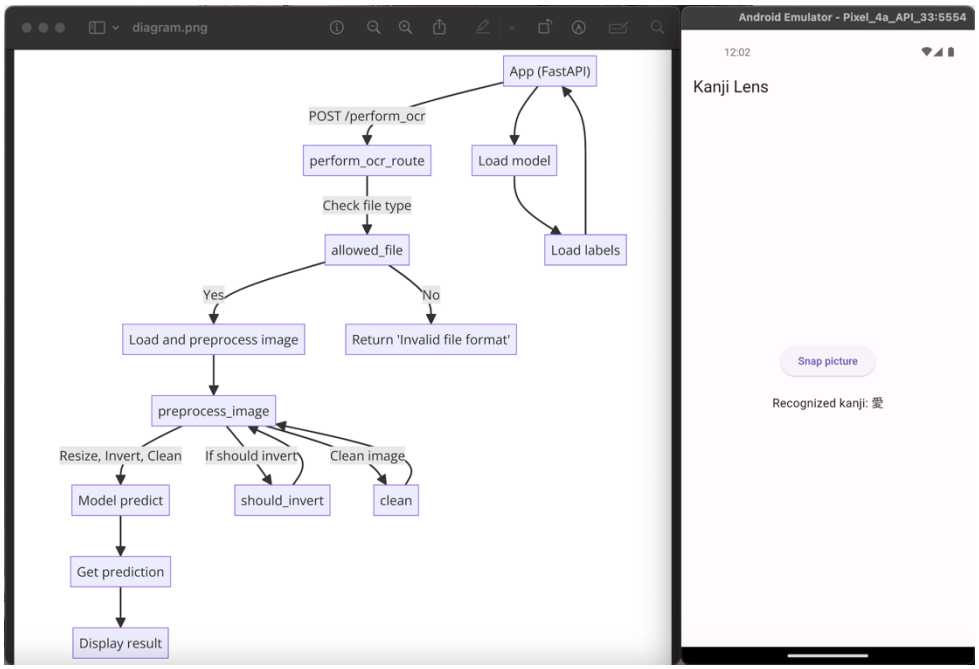


Figure 23. First version of the app alongside the backend code flowchart. In the app, the user can select a picture and see the predicted character as a text

Now, because this backend was running on my laptop, my phone needed to be on the same network as my MacBook for my phone and my laptop to communicate the requests and responses back to each other. However, the requests on my phone did not seem to reach my laptop for some reason. The first culprit that came to my mind was the MacBook's firewall blocking the HTTP request because pings did work, but even after disabling the firewall on the settings page, requests still did not work. I discarded my phone as the problem, as I could see with Wireshark, a program to sniff network traffic, that the requests were being sent. After much searching, I discovered a question on StackExchange.com with a user experiencing a similar problem. It turns out there was another application firewall running called socketfilterfw. After turning it off by issuing the command seen in Figure 24, everything worked as it should.

```
sudo /usr/libexec/ApplicationFirewall/socketfilterfw --setglobalstate off
Password:
Firewall is disabled. (State = 0)
```

Figure 24. Command to disable macOS's application firewall blocking the requests

8.4 Using TensorFlow Lite

Before enhancing the application, I wanted to avoid depending on my laptop to have the backend running. This added freedom would make testing my application by other users easier, and it would also greatly enhance the app's further development. My first idea was

to run the machine learning model on the device itself instead of an external server. Because portable devices cannot really run standard TensorFlow models, Google developed TensorFlow Lite as mentioned in section 6.3. Luckily, CaptainDario offered a TensorFlow Lite version of the model, so it was a matter of looking at how to integrate it with Flutter. This is where problems started to arise because out of the bad, there were three different libraries for TensorFlow Lite in pub.dev (official package repository for Dart and Flutter apps): tflite, tflite_v2, and tflite_flutter. This last one is the official package developed by the TensorFlow team, so this was the reasonable package to choose. However, because I did not understand the instructions entirely, I decided to go with tflite_v2 since it seemed easier to use. After following the instructions to install it, I could not build the application, as it would always throw an error. After some searching, I found no successful answer, so I decided to use tfile_flutter and try to get it working. After following the install instructions, I got another error, shown in Figure 25. In this error output, Flutter suggested that I change the minimum SDK (software development kit) version of the project to 26, and so I did. With this change, the app finally built correctly.

```
/Users/shyraffy/Documents/tflite/android/app/src/debug/AndroidManifest.xml Error:
    uses-sdk:minSdkVersion 19 cannot be smaller than version 26 declared in library [:tflite_flutter]
     /Users/shyraffy/Documents/tflite/build/tflite_flutter/intermediates/merged_manifest/debug/AndroidManifest.xml as the
      library might be using APIs not available in 19
    Suggestion: use a compatible library with a minSdk of at most 19,
        or increase this project's minSdk version to at least 26,
        or use tools:overrideLibrary="org.tensorflow.tflite_flutter" to force usage (may lead to runtime failures)

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':app:processDebugMainManifest'.
> Manifest merger failed with multiple errors, see logs

* Try:
> Run with --stacktrace option to get the stack trace.
> Run with --info or --debug option to get more log output.
> Run with --scan to get full insights.

* Get more help at https://help.gradle.org
```

Figure 25. Error thrown by my code editor after installing tflite_flutter in the Flutter project

Despite this, I still had to tackle another problem: the pre-processing of the images. Going serverless meant I also needed to move the pre-processing logic to the device. This imposed a problem because, although there is a package of OpenCV for Flutter called opencv_dart in pub.dev, there is no equivalent of NumPy for Dart, which is an issue as my pre-processing code uses some of NumPy functions. After much trial and error, I could not get this logic to work and even if I kept trying and managed to make it work, the code would have ended up too convoluted and difficult to read.

Since, at this point, I knew going serverless would not be worth the trouble, I thought that finding some sort of hosting server that allowed me to run the prediction script of Figure 23 on the cloud would be good enough. It is not serverless, but I would not have to run the server on my laptop, meaning I could use the application everywhere and anytime as long as I had internet access, and the hosting service was up. I considered several alternatives, like AWS, Google Cloud, Heroku, PythonAnywhere, and Render. Google Cloud and Heroku were discarded quickly as none offered a free tier. AWS did offer a free tier, but it had trouble with the prediction script's dependencies.

On the other hand, PythonAnywhere was also free but did not work correctly, and the functionality and interface were strange. In the end I went with Render. Render allows the connection of a GitLab repository and automatically deploys the application after a commit is made to the repository. This requires some configuring, and I had trouble with the dependencies, especially with OpenCV. After fixing that, the application would have deployed correctly if something had not slipped my mind. I forgot that the free tier of Render only allowed the use of a total of 512 MB of RAM. The standard TensorFlow Model that I was using consumed more than this, causing the deployments to fail. I was about to give up the idea and just continue using my laptop as the server when I realized I could just use the TFLite model on the server, as that would consume much less RAM. Indeed, the deployment was successful after modifying the code to use the TFLite model instead. As seen in the Figure 26, the process if a bit different when using a TFLite model instead of a regular TF model. I finally had my application backend running on the cloud at https://kanji-lens-backend-mtsr.onrender.com/. Note that accessing this URL will not return anything as there is no route defined on the API for that. An image must be sent to https://kanji-lens-backend-mtsr.onrender.com/perform_orc for the prediction to work.

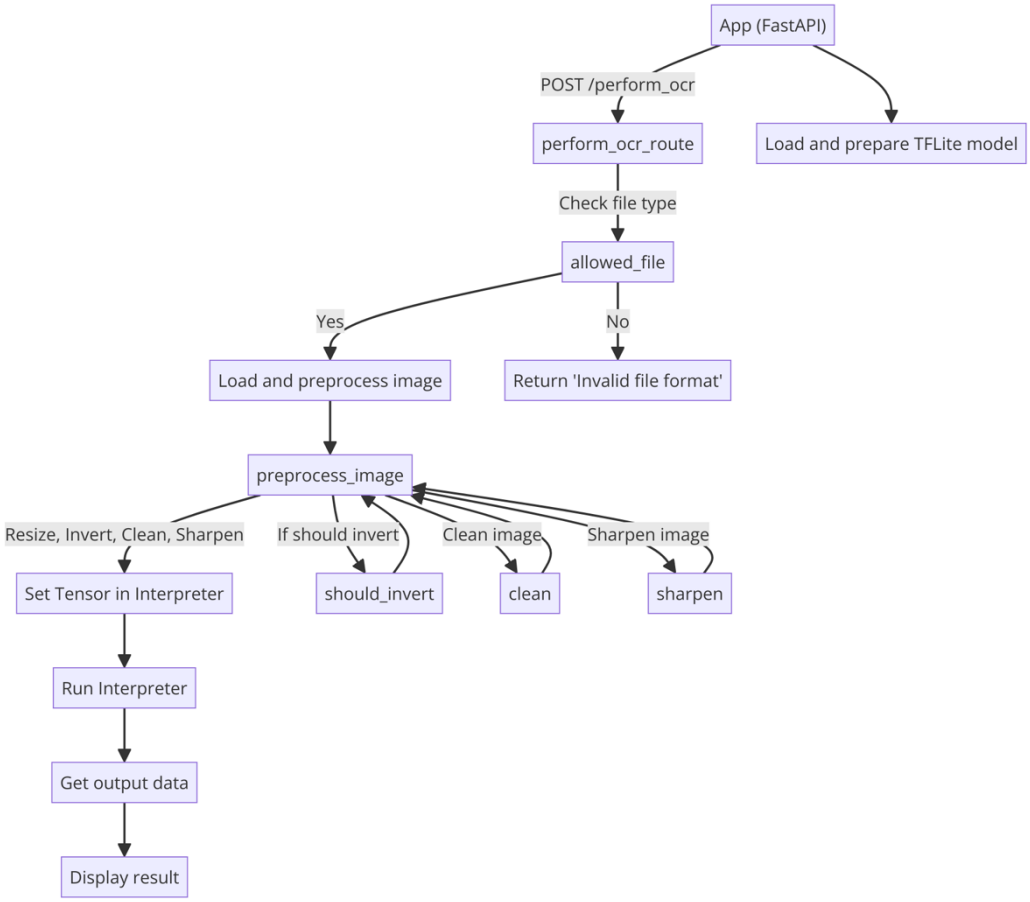


Figure 26. Modified backend flowchart that uses TFLite on the Render cloud

8.5 Final application

Once I had my backend running in the cloud, I continued developing the rest of the application. First, I tried to improve the interface by adding two big buttons, one for taking a picture with the camera and another for selecting a picture from the phone's camera. Additionally, since the whole point of the application was to display information about a specific kanji, I added a page to display the following information (Figure 27):

- Diagram of the kanji: Kanji have a specific stroke order in which they must be drawn in order for them to look balanced. In code, this diagram is a package called "kanji_drawing_animation" found in pub.dev. It is helpful as it allows one to see an animation showing how to draw the kanji following its stroke order.
- JLPT Level: Since 1984, the Japanese-Language Proficiency Test (JLPT) has been offered by the Japan Foundation and Japan Educational Exchanges and Services as a method for assessing and accrediting the Japanese language skills of non-native speakers. The JLPT has five levels: N1, N2, N3, N4 and N5. N5 is the most

basic level, whereas N1 is the most difficult. This field indicates at which level the kanji would appear in a JLPT test. (JLPT Japanese-Language Proficiency Test.)

- Frequency: This field indicates the frequency with which this particular kanji appears in Japanese newspapers. This serves as a reference of how much the user would expect to see this kanji while reading Japanese texts or strolling through Japan.
- Number of strokes: The number of individual strokes it takes to draw the kanji on paper.
- Meaning: different meanings of the kanji.
- On'yomi: On'yomi reading of the kanji, indicated by “音”.
- Kun'yomi: Kun'yomi reading of the kanji, indicated by “訓”.
- Example phrases: This field is self-explanatory. It helps the user see how the kanji is used in a phrase and the different readings it can take.

The data on these fields are fetched from Jishoo.org, through an API also available in pub.dev called “unofficial_jisho_api”.

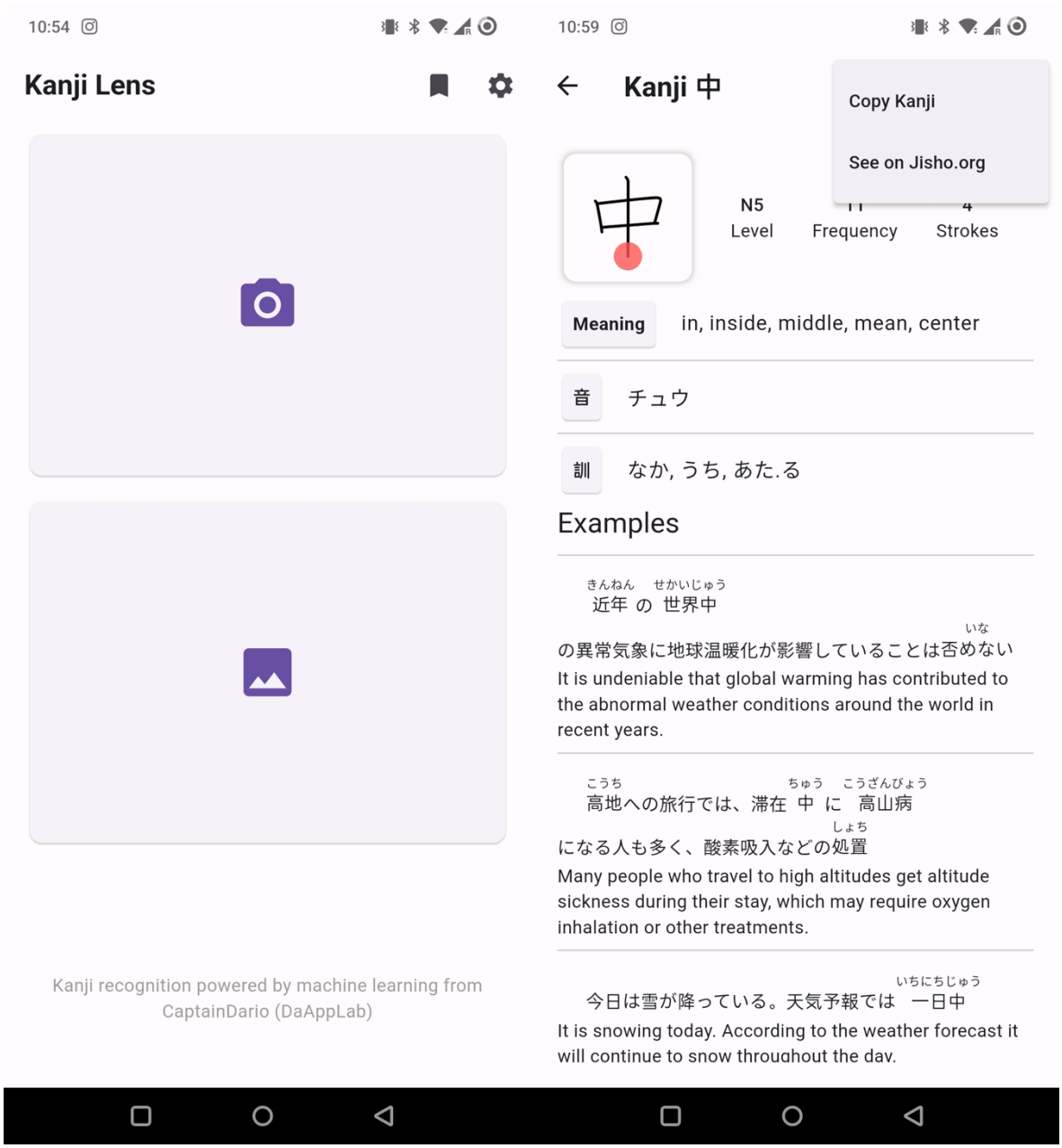
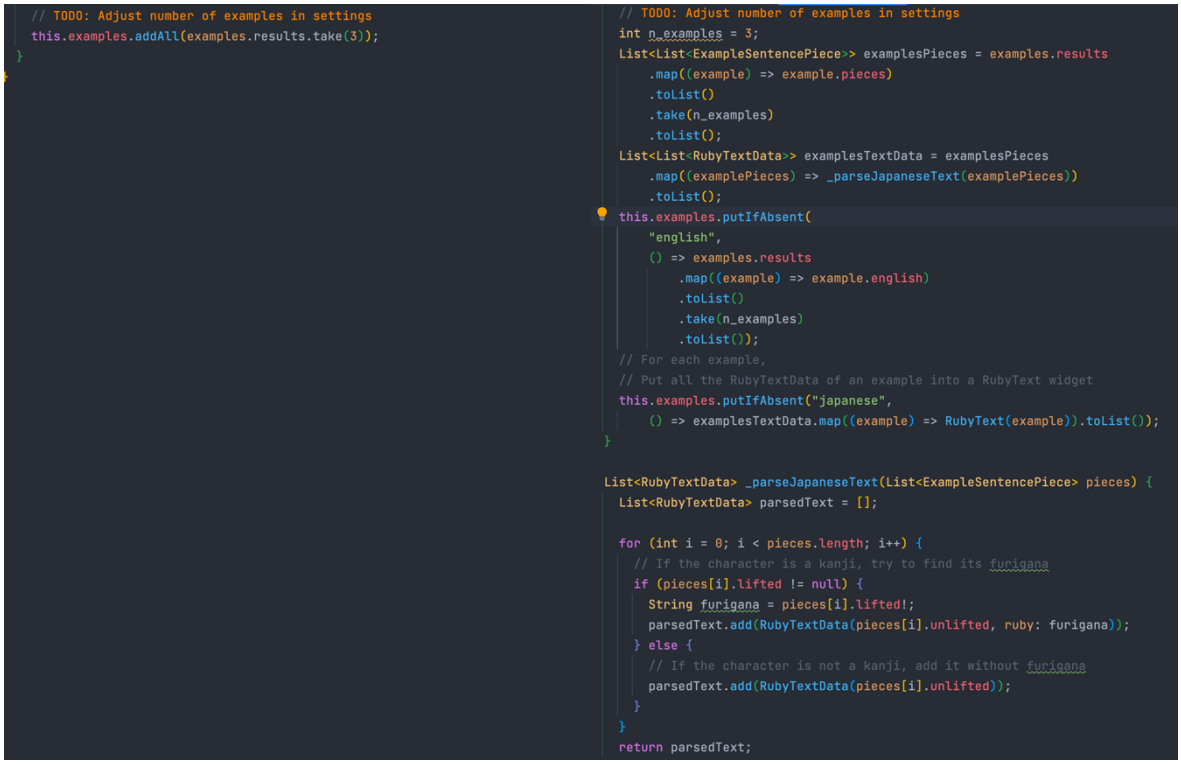


Figure 27. Final look of the application

If one takes a closer look at the Japanese text in the examples, one can see tiny characters on top of the kanji characters. These are furigana, small kana placed adjacent to kanji as a reading aid on how the kanji is pronounced in that particular context (Bullock, B). My first version of the app did not have this, and implementing it was a bit of a challenge. For implementing furigana, I used another package available in pub.dev called “ruby text”. The difficulty was that I had to modify a lot of the logic that fetches the examples and makes it available to the frontend, because instead of plain string of text, I now had to have a list of RubyText widgets, which itself needs a list of RubyTextData that contains the actual text string, as seen on Figure 27. I am not going to get into more detail, but in Figure 27 the

difference in the implementation between the version of the app without furigana on the left, and the app with furigana on the right, can be seen.



```
// TODO: Adjust number of examples in settings
this.examples.addAll(examples.results.take(3));
}
}
```

```
// TODO: Adjust number of examples in settings
int n_examples = 3;
List<List<ExampleSentencePiece>> examplesPieces = examples.results
    .map((example) => example.pieces)
    .toList()
    .take(n_examples)
    .toList();
List<List<RubyTextData>> examplesTextData = examplesPieces
    .map((examplePieces) => _parseJapaneseText(examplePieces))
    .toList();
this.examples.putIfAbsent(
    "english",
    () => examples.results
        .map((example) => example.english)
        .toList()
        .take(n_examples)
        .toList());
// For each example,
// Put all the RubyTextData of an example into a RubyText widget
this.examples.putIfAbsent("japanese",
    () => examplesTextData.map((example) => RubyText(example)).toList());
}

List<RubyTextData> _parseJapaneseText(List<ExampleSentencePiece> pieces) {
  List<RubyTextData> parsedText = [];

  for (int i = 0; i < pieces.length; i++) {
    // If the character is a kanji, try to find its furigana
    if (pieces[i].lifted != null) {
      String furigana = pieces[i].lifted!;
      parsedText.add(RubyTextData(pieces[i].unlifted, ruby: furigana));
    } else {
      // If the character is not a kanji, add it without furigana
      parsedText.add(RubyTextData(pieces[i].unlifted));
    }
  }
  return parsedText;
```

Figure 28. Comparison of the application code that prepares the examples for the frontend, before implementing furigana (left) and after implementing furigana (right)

As seen on Figure 29, I also implemented a functionality where the user can bookmark a kanji after recognizing it, so it can be checked later through a menu, indicating the kanji, the image taken, and when it was recognized. When pressing these items, a menu with the same page as Figure 27 pops up from the bottom. Implementing this also took a lot of refactoring since I wanted to avoid copying and pasting the same code from the UI in Figure 27. The problem was that the code from Figure 26 contained the button and code to bookmark the kanji but having that on the page in Figure 28 did not make sense. So, after some fighting with Dart and moving things around, I managed to get this working with somewhat good-quality code. The rest of the development involved refactoring code and making it more readable and portable. I especially struggled when trying to separate the UI code from the prediction logic, mainly because of problems with Dart's asynchronous programming, which is mandatory when fetching data from the internet. Nevertheless, I stopped my code editor from complaining after playing around a little bit.

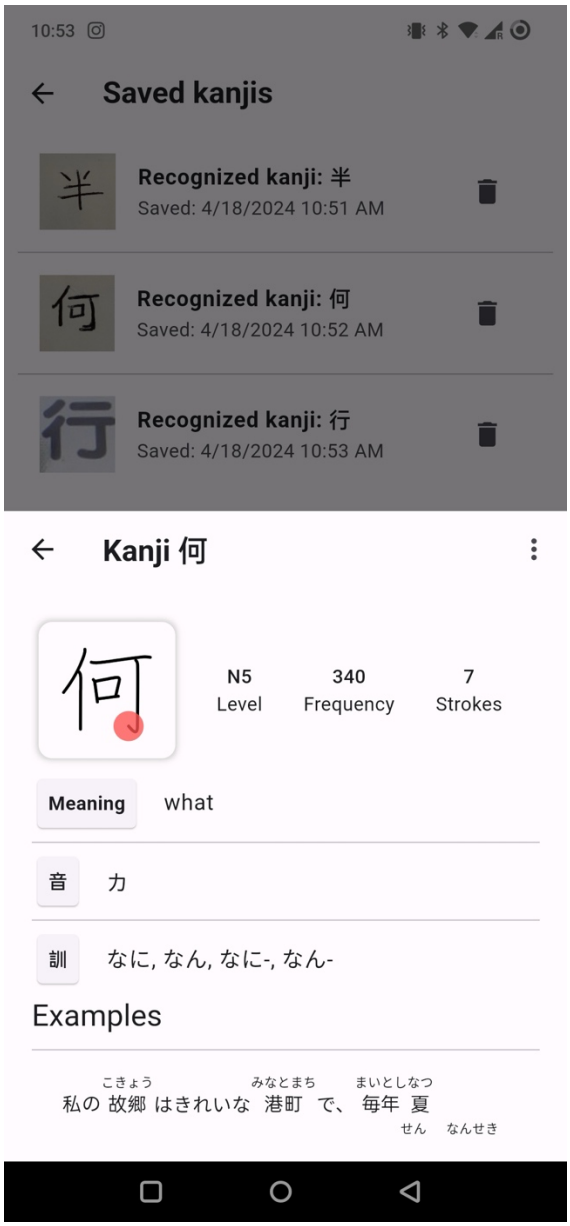


Figure 29. The user can bookmark kanjis and look at them later

8.6 Conclusion

In the end, I got a working application that can recognize kanji correctly most of the time, as seen on Figure 30. So, I can say the objective of the case has been met successfully. Nevertheless, there is a lot of room for improvement. This research had several significant limitations. Mainly the dataset restrictions and the intrinsic complexities introduced by kanji. As I said in my introduction to Japanese, there are around 2000 daily-use kanjis. Training a model that can differentiate between such a number of kanjis is not trivial at all, as this exacerbates any issue the dataset might have, such as class imbalance. Because of this, the model is not as accurate as I would like. Also, it looks like the CNN was trained with a dataset that only contained hand-written kanjis, so it might have problems generalizing to

kanjis in street signs, for example. I tried other approaches, like training an open-source LLM like LLaVA with the same data the CNN was trained on or using another model called Kindai-OCR that I found on GitHub, but neither gave any results. Extracting the data CaptainDario used for training the CNN was not possible in Collab due to the free plan limitations, and even if I managed to re-train LLaVA and the model worked better, which is already an optimistic outcome, each call to LLaVA would have cost me some money. This last thing was a big deal for me as I wanted the application to be open-source while being uploaded to several app stores, and I was not willing to run the app at a loss. As for Kindai-OCR, it looks like the code provided in the actual version does not work, and although I tried to fix it by myself, I concluded that the effort required to fix all the code was not worth it, considering my time constraints. (DeepAps91; Liu, H., et al. 2023.)

Figure 30. Results of the model in a small test dataset

Furthermore, because the model expects a single kanji, the user needs to crop the picture to center the desired character. This is rather inconvenient for the user, but I could not think of a better approach. I tried segmenting the kanjis using a conventional vision algorithm with OpenCV, but I did not manage to get this working successfully. Getting around this problem and providing the user with a smoother and more direct experience in this aspect would significantly improve the application.

Another improvement I would like to pursue is integrating the model into the user's device. As I said in section 8.3, this approach seemed complicated, but I think achieving this would be possible and worth it with more time and effort. I did not mention in the previous section that the free tier of Render makes the server go down after a short period of inactivity. This results in a considerable delay in the first request done to the server after a while. This is such a huge inconvenience that I cannot afford to publish the app on any app store, making debugging and development difficult.

9 Summary

This thesis has looked into the development and implications of an AI mobile application for kanji recognition. What is important in this work is that it can help when learning kanji, which is an inherent part of the Japanese language and famous for its complexity. This paper proposes a practical solution in kanji learning by benefiting from AI techniques, specifically convolutional neural networks.

I started by doing an introduction to the Japanese language and kanji, trying to highlight why it can be hard to learn. Afterward, I did a general introduction to AI, emphasizing in machine learning and deep learning, especially in neural networks and convolutional neural networks, which are pillars of these disciplines. Finally, I summarized some tools used in the field of AI and finished by describing the development process of my practical case.

In my practical case, I developed an application for recognizing kanjis in images using a convolutional neural network and providing information about such kanjis. During the development, I faced many programming challenges. Some of them I solved, but others made me change my approach.

Overall, despite the problems, I think I was able to develop a helpful app that can be handy for Japanese learners, and I also exercised my problem-solving skills and became more confident in my ability to face problems and find solutions.

All the code for the application, as well as releases, is available at my GitLab as open source for everyone to see, modify and improve.

References

Anoriega. (2022). Top 6 AI programming Languages to learn in 2023 | Berkeley Boot Camps. Berkeley Boot Camps. Retrieved March 18, 2024. Available at https://bootcamp.berkeley.edu/blog/ai-programming-languages/

Bajada, J. (2019). Symbolic vs Connectionist A.I. Medium. Retrieved January 25, 2024 from https://towardsdatascience.com/symbolic-vs-connectionist-a-i-8cf6b656927

Beermann, R. E. (2006). Introduction to the Japanese Writing System. Retrieved March 9, 2024. Available at https://www.uni-due.de/imperia/md/content/japan/introduction_to_kanji.pdf

Beohar, D., & Rasool, A. (2021). Handwritten Digit Recognition of MNIST dataset using Deep Learning state-of-the-art Artificial Neural Network (ANN) and Convolutional Neural Network (CNN). Retrieved May 1, 2024. Limited availability at https://ieeexplore.ieee.org/document/9396870

Bishop, C. M. (1994). Neural networks and their applications. Review of scientific instruments, 65(6), 1803-1832. Retrieved March 11, 2024. Available at http://www.stat.purdue.edu/~zdaye/Readings/Neural_Networks_and_Their_Applications.pdf

Bisong, E. (2019). Building Machine Learning and Deep Learning Models on Google Cloud Platform. Retrieved March 11, 2024. Available at https://link.springer.com/content/pdf/10.1007/978-1-4842-4470-8.pdf

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., & Zhang, Y. (2023). Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv preprint arXiv:2303.12712. Retrieved January 26, 2024. Available at https://arxiv.org/abs/2303.12712

Bullock, B. (n.d.). What is furigana? Retrieved April 18, 2024. Available at https://www.sljfaq.org/afaq/furigana.html

CaptainDario. CaptainDario/DaKanji-Single-Kanji-Recognition: A machine learning model to recognize Japanese characters (Kanji, Katakana, Hiragana). GitHub. Retrieved March 27, 2024. Available at https://github.com/CaptainDario/DaKanji-Single-Kanji-Recognition

Cave, S., Craig, C., Dihal, K., Dillon, S., Montgomery, J.A., Singler, B., & Taylor, L.C. (2018). Portrayals and perceptions of AI and why they matter. Retrieved January 29, 2024. Available at https://royalsociety.org/~/media/policy/projects/ai-narratives/AI-narratives-workshop-findings.pdf

Chollet, F. (2019). On the Measure of Intelligence. arXiv preprint arXiv:1911.01547. Retrieved January 24, 2024. Available at https://arxiv.org/abs/1911.01547

Cybenko, G.V. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 2, 303-314. Retrieved February 22, 2024. Available at https://hal.science/hal-03753170/file/Cybenko1989.pdf

Daniel Keysers. (2007). Comparison and Combination of State-of-the-art Techniques for Handwritten Character Recognition: Topping the MNIST Benchmark. Retrieved May 1, 2024. Available at https://arxiv.org/pdf/0710.2231

DeepAps91. DeepApps91/Kindai-OCR: OCR system for recognizing modern Japanese magazines. GitHub. Retrieved April 18, 2024. Available at https://github.com/DeepApps91/Kindai-OCR

Duarte, D., & Ståhl, N. (2018). Machine Learning: A Concise Overview. Data Science in Practice. Retrieved January 27, 2024. Limited availability at https://link.springer.com/chapter/10.1007/978-3-319-97556-6_3

Emergent Garden. (2022). Youtube Video. Retrieved February 1, 2024. Available at https://www.youtube.com/watch?v=0QczhVg5HaI&t=394s

Flowers, J.C. (2019). Strong and Weak AI: Deweyan Considerations. AAAI Spring Symposium: Towards Conscious AI Systems. Retrieved January 26, 2024. Available at https://ceur-ws.org/Vol-2287/paper34.pdf

Francis from ResponseBase (2019). Figuring out what is blocking HTTP request on macOS Mojave? Online forum post. Stack Exchange. Retrieved March 29, 2024. Available at https://apple.stackexchange.com/questions/362416/figuring-out-what-is-blocking-http-request-on-macos-mojave

Furnieles, G. (2022). Sigmoid and SoftMax Functions in 5 minutes - Towards Data Science. Medium. Retrieved February 29, 2024. Available at https://towardsdatascience.com/sigmoid-and-softmax-functions-in-5-minutes-f516c80ea1f9

Getting started with TensorFlow Lite | Seeed Studio Wiki. (2023). Retrieved March 22, 2024. Available at https://wiki.seeedstudio.com/reTerminal_ML_TFLite/

Goode, L. (2018). Life, but not as we know it: A.I. and the popular imagination. Culture Unbound, 10(2), 185–207. Retrieved January 29, 2024. Available at https://doi.org/10.3384/cu.2000.1525.2018102185

Hasegawa, Y. (2014). Japanese: A linguistic introduction. Cambridge University Press. Retrieved March 9, 2024. Available at

https://books.google.fi/books?id=6S_CBQAAQBAJ&lpg=PR15&ots=PhZi4c5R-i&lr&hl=es&pg=PA4#v=onepage&q&f=false

Hjort, E. (2020). Evaluation of React Native and Flutter for cross-platform mobile application development. Retrieved March 18, 2024. Available at https://www.doria.fi/bitstream/handle/10024/180002/hjort_elin.pdf?sequence=2

Jordan, M.I., & Mitchell, T. (2015). Machine learning: Trends, perspectives, and prospects. Science, 349, 255 - 260. Retrieved January 27, 2024. Available at https://www.cs.cmu.edu/~tom/pubs/Science-ML-2015.pdf

Kathuria, A. (2018). How to chose an activation function for your network. Paperspace Blog. Referenced on February 22, 2024. Available at https://blog.paperspace.com/vanishing-gradients-activation-function/

Ketkar, N.S. (2021). Deep Learning with Python. Retrieved March 18, 2024. Available at https://link.springer.com/content/pdf/10.1007/978-1-4842-2766-4.pdf

Kha-White. kha-white/manga-ocr: Optical character recognition for Japanese text, with the main focus being Japanese manga. GitHub. Retrieved March 27, 2024. Available at https://github.com/kha-white/manga-ocr

Kim, T. (2012). Japanese Grammar Guide. Retrieved March 9, 2024. Available at https://www.guidetojapanese.org/grammar_guide.pdf

Koonce, B., & Koonce, B. (2021). EfficientNet. Convolutional neural networks with swift for Tensorflow: image recognition and dataset categorization. Retrieved March 27, 2024. Available at https://link.springer.com/content/pdf/10.1007/978-1-4842-6168-2.pdf

Liu, H., Li, C., Li, Y., & Lee, Y. J. (2023). Improved baselines with visual instruction tuning. arXiv preprint arXiv:2310.03744. Retrieved April 18, 2024. Available at https://arxiv.org/abs/2310.03744

Linares Pellicer, J. (2021)a. Introduction AI. In Universitat Politècnica de València. Retrieved, January 24, 2024. Limited availability at https://poliformat.upv.es/access/content/group/GRA_14411_2020/Basic%20slides/10%20-%20Introduction%20to%20AI.pdf

Linares Pellicer, J. (2021)b. Introduction to CNNs. In Universitat Politècnica de València. Retrieved April 18, 2024. Limited availability https://poliformat.upv.es/access/content/group/GRA_14411_2020/Basic%20slides/10%20-%20Introduction%20to%20CNNs.pdf

Linares Pellicer, J. (2021)c. Introduction to Machine Learning. In Universitat Politècnica de València. Retrieved, January 27, 2024. Limited availability https://poliformat.upv.es/access/content/group/GRA_14411_2020/Basic%20slides/10%20-%20Introduction%20to%20Machine%20Learning.pdf

Linares Pellicer, J. (2021)d. Regularization and optimization in deep learning. In Universitat Politècnica de València. Retrieved, January 27, 2024. Limited availability https://poliformat.upv.es/access/content/group/GRA_14411_2020/Basic%20slides/10%20-%20Regularization%20and%20optimization%20in%20deep%20learning.pdf

Linares Pellicer, J. (2021)e. Training deep neural networks. In Universitat Politècnica de València. Retrieved March 11, 2024. Limited availability https://poliformat.upv.es/access/content/group/GRA_14411_2020/Basic%20slides/10%20-%20Training%20deep%20neural%20networks.pdf

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Mon-ga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Retrieved March 18, 2024. Available at https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf

Moyano, L.G. (2017). Learning network representations. The European Physical Journal Special Topics, 226, 499-518. Retrieved February 20, 2024. Available at https://link.springer.com/content/pdf/10.1140/epjst/e2016-60266-2.pdf

Narayan, S., & Tagliarini, G.A. (2005). An analysis of underfitting in MLP networks. Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., 2, 984-988 vol. 2. Retrieved February 20, 2024. Limited availability at https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1555986

Nielsen, M. (2019). Neural Networks and Deep Learning. Retrieved, February 1, 2024. Available at http://neuralnetworksanddeeplearning.com/

NihongoShark. The Japanese Writing System. Referenced on March 9, 2024. Available at https://www.nihongoshark.com/post/the-japanese-writing-system

Nik. (2023). Softmax Activation Function for Deep Learning: A complete guide. Datagy. Retrieved February 29, 2024. Available at https://datagy.io/softmax-activation-function/

NumPy - About us. Retrieved March 17, 2024. Available at https://NumPy.org/about/

O'Shea, Keiron & Nash, Ryan. (2015). An Introduction to Convolutional Neural Networks. ArXiv e-prints. Retrieved March 11, 2024. Available at https://arxiv.org/pdf/1511.08458.pdf

Pallets. pallets/flask: The Python micro framework for building web applications. GitHub. Retrieved March 22, 2024. Available https://github.com/pallets/flask

Pant, A. (2021). Introduction to Logistic Regression - towards data science. Medium. Referenced on February 20, 2024. Available at https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

Pérez Ibarra, S. G., Quispe, J. R., Mullicundo, F. F., & Lamas, D. A. (2021). Herramientas y tecnologías para el desarrollo web desde el FrontEnd al BackEnd. In XXIII Workshop de Investigadores en Ciencias de la Computación (WICC 2021, Chilecito, La Rioja). Retrieved March 18, 2024. Available at https://sedici.unlp.edu.ar/bitstream/handle/10915/120476/Ponencia.pdf-PDFA.pdf?sequence=1

Rasamoelina, A.D., Adjailia, F., & Sinčák, P.J. (2020). A Review of Activation Function for Artificial Neural Network. 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI), 281-286. Retrieved February 20, 2024. Limited availability at https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9108717

Rich, E. and Knight, K. (1991) Artificial Intelligence. McGraw-Hill, New York. Retrieved, January 24, 2024.

Robinson, S. (2020). Understanding how deep learning black box training creates bias. Enterprise AI. Referenced on February 20, 2024. Available at https://www.techtarget.com/searchenterpriseai/feature/Understanding-how-deep-learning-black-box-training-creates-bias

Shao, Z., Zhao, R., Yuan, S., Ding, M., & Wang, Y. (2022). Tracing the evolution of AI in the past decade and forecasting the emerging trends. Expert Systems with Applications, 118221. Retrieved, January 24, 2024. Available at https://www.sciencedirect.com/science/article/am/pii/S0957417422013732

Shayan RC (2013). Why should weights of neural networks be initialized to random numbers? Online forum post. Stack Overflow. Retrieved March 11, 2024. Available at https://stackoverflow.com/questions/20027598/why-should-weights-of-neural-networks-be-initialized-to-random-numbers

Sword Art Online: Alicization. Myanimelist.net. Retrieved January 29, 2024. Available at https://myanimelist.net/anime/36474/Sword_Art_Online__Alicization

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning (pp. 6105-6114). PMLR. Retrieved March 27, 2024. Available at https://proceedings.mlr.press/v97/tan19a/tan19a.pdf

Tashildar, A., Shah, N., Gala, R., Giri, T., & Chavhan, P. (2020). Application development using flutter. International Research Journal of Modernization in Engineering Technology and Science, 2(8), 1262-1266. Retrieved March 18, 2024. Available at https://www.irjmets.com/uploadedfiles/paper/volume2/issue_8_august_2020/3180/1628083124.pdf

The 365 team (2023). Overfitting vs. Underfitting: What Is the Difference? 365 Data Science. Referenced on February 20, 2024. Available at https://365datascience.com/tutorials/machine-learning-tutorials/overfitting-underfitting/

The Math Sorcerer. (2020). Youtube Video. Retrieved March 12, 2024. Available at https://www.youtube.com/watch?v=H1DJjyDFalw

Tiangolo. tiangolo/fastapi: FastAPI framework, high performance, easy to learn, fast to code, ready for production. GitHub. Retrieved March 22, 2024. Available at https://github.com/tiangolo/fastapi

Toews, R. (2024). Transformers revolutionized AI. What will replace them? Forbes. Retrieved March 27, 2024. Available at https://www.forbes.com/sites/robtoews/2023/09/03/transformers-revolutionized-ai-what-will-replace-them/?sh=58de18799c1f

What is the Japanese-Language Proficiency Test? JLPT Japanese-Language Proficiency Test. Retrieved April 8, 2024. Available at https://www.jlpt.jp/e/about/index.html

Wikipedia contributors. (2023)a. Sword Art Online: Alicization. In Wikipedia, The Free Encyclopedia. Retrieved January 29, 2024. Available at https://en.wikipedia.org/w/index.php?title=Sword_Art_Online:_Alicization&oldid=1188577949

Wikipedia contributors. (2024)b. Doraemon. In Wikipedia, The Free Encyclopedia. Retrieved January 29, 2024. Available at https://en.wikipedia.org/w/index.php?title=Doraemon&oldid=1200039528

Wikipedia contributors. (2024)c. Interstellar (film). In Wikipedia, The Free Encyclopedia. Retrieved January 29, 2024. Available at https://en.wikipedia.org/w/index.php?title=Interstellar_(film)&oldid=1200075060

Wikipedia contributors. (2024)d. Turing test. In Wikipedia, The Free Encyclopedia. Retrieved, January 24, 2024. Available at https://en.wikipedia.org/w/index.php?title=Turing_test&oldid=1194139533

Xia, Z. (2019). An Overview of Deep Learning. Deep Learning in Object Detection and Recognition. Retrieved, February 1, 2024. Available at https://link.springer.com/content/pdf/10.1007/978-981-10-5152-4_1.pdf?pdf=inline%20link

nbro (2020). Does the number of parameters in a convolutional neuronal network increase if the input dimension increases? Online forum post. Stack Exchange. Retrieved April 18, 2024. Available at https://ai.stackexchange.com/questions/22075/does-the-number-of-parameters-in-a-convolutional-neuronal-network-increase-if-th

pandas - Python Data Analysis Library. (n.d.). Retrieved March 17, 2024, 2024. Available at https://pandas.pydata.org/about/

rodrivers. (2019). - AI: Measures, Maps and Taxonomies. AIethics.AI - Artificial Intelligence and Robot Ethics. Retrieved, January 24, 2024. Available at https://robotethics.co.uk/ai-measures-maps-and-taxonomies/

user11530462 (2020). Why do we have to normalize the input for an artificial neural network? Online forum post. Stack Overflow. Retrieved March 28, 2024. Available at https://stackoverflow.com/questions/4674623/why-do-we-have-to-normalize-the-input-for-an-artificial-neural-network