Computer science department

End of Studies Project Report

Bachelor's Degree : Mathematical Computer Science

Use of Deep learning for Tifinagh character recognition

Realized by :

Oussiddi Mohammed

Jury Members:

Pr. Oubelkacem Ali

Pr. Kamal Asmae

Pr. El Alalli Eddrissia

Academic Year 2022 - 2023

Dedication

My dear parents,

All the words in the world cannot express the immense love I have for you, nor the deep gratitude I feel towards you for all the efforts and sacrifices you have made throughout my education and well-being. I thank you for all the support and love you have shown me since my childhood, and I hope that your blessings will always be with me. I hope I have lived up to the hopes you had for me and that I have fulfilled one of your dreams today. I pay tribute to you with this modest work as a token of my eternal gratitude and infinite love. May Almighty God keep you safe and grant you health, happiness, and long life so that you remain the guiding light on the path of your children. I love you, Mom and Dad.

To my dear cousin,

I want to express my deepest gratitude for your invaluable help and support throughout this project. Your expertise, guidance, and encouragement have been instrumental in making this work possible. Your willingness to lend a helping hand at every step of the way has been a source of inspiration and motivation for me. I am honored to have you as a cousin and a mentor, and I dedicate this work to you as a token of my appreciation and admiration. May your kindness and generosity always be rewarded, and may our bond of family and friendship remain unbreakable. Thank you from the bottom of my heart.

With love and respect.

Oussiddi Mohammed

This project is dedicated to my incredible parents and the dedicated individuals who have worked tirelessly to support and complete it. Your unwavering belief in me and your tireless efforts have been the driving force behind my success. With deepest gratitude, i dedicate this achievement to each and every one of you.

Dergaoui Ayoub

«I would like to express my deep gratitude to those who have contributed to this work or provided support during its creation.»

Acknowledgments

First and foremost, we would like to thank Almighty God, the Merciful and Powerful, for giving us the strength and patience to complete this modest work.

At the end of this project, it is our pleasure to express our sincere thanks to all those whose contributions have helped bring it to fruition.

We would like to express our gratitude to Mr. Bekri Ali for his insightful guidance and support throughout our end-of-studies project, as well as for his valuable advice and involvement in the evaluation of our work and the writing of this report.

Our ineffable thanks go to the members of the jury who evaluated our End-of-Studies Project, we thank them for evaluating the work we have done and for their availability.

We would also like to express our sincere thanks to the entire teaching staff at the Faculty of Science Meknes.

Finally, we extend our warmest thanks to all those who have contributed in any way to the completion of this work.

Summary

This report is part of our End-of-Studies Project, which is the culmination of our undergraduate education in Mathematics and Computer Science at the Faculty of Sciences in Meknes. It draws on both theoretical and practical knowledge in the field of data science in general, and specifically in the field of Deep Learning. As its name suggests, our project aims to design and implement a model for recognizing Tifinagh characters using a Deep Learning approach.We have created a model that fragments and recognizes Tifinagh characters with an accuracy of 98.7 % .

Images are a common and powerful form of data that are frequently used in data science. Deep learning models can be trained to analyze and recognize patterns in images, which has a wide range of applications such as image classification, object detection, and image segmentation. Convolutional neural networks (CNNs) are a type of deep learning model that are particularly well-suited for image analysis, as they are designed to recognize patterns in spatial data. With the development of deep learning and the availability of large amounts of image data, the use of images in deep learning has become increasingly important in fields such as computer vision, autonomous driving, and medical imaging.

List of abbreviations

Abbreviation	Meaning
ML	Machine learning
DL	Deep learning
CNN	Convolutional Neural Networks
RNN	Recurrent neural network
OCR	Optical Character Recognition

Contents

List of Figures

General Introduction

In recent years, machine learning has emerged as a powerful tool for solving complex problems in various fields. One of the areas where machine learning has shown great promise is character recognition, which has many applications in text recognition, optical character recognition, and handwriting recognition. Tifinagh, an ancient script used in North Africa, poses a unique challenge for character recognition due to its complex structure and lack of standardization. In this context, the use of machine learning techniques can be particularly useful for developing accurate and efficient Tifinagh character recognition systems. This project aims to explore the use of machine learning for Tifinagh character recognition and to design and implement a model with high accuracy and efficiency using handwritten images.

The image processing in deep learning is a rapidly growing field that focuses on using neural networks to automatically extract meaningful features from digital images. It has numerous applications in various fields such as object detection, medical imaging, and computer vision. In the context of Tifinagh character recognition, image processing plays a crucial role in preprocessing the input data and extracting relevant features that can be used to train the machine learning models. Techniques such as image segmentation, edge detection, and image normalization can be used to enhance the quality of the input images and improve the accuracy of the recognition system.

The project will involve the design and implementation of a deep learning model to segment and recognize Tifinagh characters. The model will be trained and evaluated using a dataset of Tifinagh characters, and its performance will be measured using accuracy and other standard metrics.

The success of this project could have significant implications for the development of OCR systems for Tifinagh scripts, which could in turn support the preservation and promotion of the Amazigh culture and language. Additionally, the project could contribute to the advancement of the field of character recognition and inspire further research in this area.

Chapter 1

General context of the project

This chapter serves as a comprehensive introduction to the environment of our final year project. It provides an overview of the various components and history related to the project. The subsequent sections will detail the project itself, including its objectives and any challenges that may be encountered. Additionally, a detailed plan for execution will be presented, outlining the methodology that will be followed to achieve the project's objectives.

The purpose of this chapter is to set the context for the rest of the project report and provide the reader with a clear understanding of the background and scope of the project. By providing a detailed history of the project and its components, the reader can better understand the motivations behind its creation. The subsequent sections will then provide a clear and concise description of the project's objectives and challenges, ensuring that the reader has a solid understanding of what will be accomplished.

Furthermore, the detailed plan for execution and methodology provides a roadmap for achieving the project's objectives, giving the reader a clear understanding of the steps that will be taken to complete the project successfully. Overall, this chapter plays an essential role in providing a comprehensive overview of the project's environment and setting the stage for the rest of the report.

1 Project presentation :

Our project is quite impressive and has the potential to make a significant impact in the field of character recognition for Tifinagh-based languages. Handwritten Tifinagh character recognition is a complex and challenging task, and the use of machine learning, specifically deep learning algorithms, can greatly enhance the accuracy and efficiency of this process. Convolutional neural networks (CNNs) are well-suited for image recognition tasks and have been successfully used in similar projects for other writing systems, making them a promising choice for Tifinagh character recognition as well. To build a robust and accurate model, you will need to gather and carefully curate a large dataset of handwritten Tifinagh characters, and design an effective CNN architecture. The training and testing of the model will require significant computational resources, but the results could have important implications for the preservation and digitization of Tifinagh-based languages, as well as other related fields. Overall, our project has the potential to make a significant contribution to the advancement of machine learning in character recognition tasks.

1.1 Motivations and problem statement of the project:

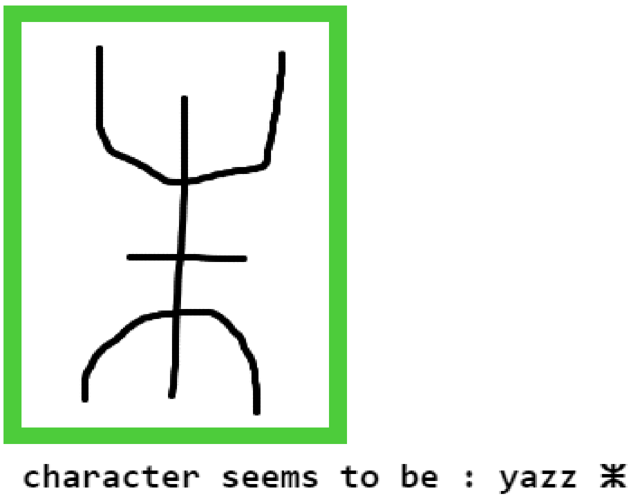
The motivation behind this project is to address the problem of recognizing Tifinagh characters, which are widely used in the Maghreb region of North Africa but have limited recognition capabilities in existing systems. This poses a challenge for digitizing Tifinagh texts, as manual transcription is time-consuming and prone to errors.

Therefore, the aim of this project is to design and implement a machine learning model that can accurately recognize Tifinagh characters. The model will be developed using deep learning techniques and will leverage the power of convolutional neural networks (CNNs) to identify and classify the characters.

The success of the project can provide a useful tool for preserving the Tifinagh culture and language, especially in the digital age where the majority of written content is in digital form. It can also be used for automatic Tifinagh text recognition in various applications, such as Tifinagh OCR (Optical Character Recognition) and Tifinagh language translation. Moreover, the project can serve as a starting point for future research and development in the field of Tifinagh character recognition and other related areas of machine learning and computer vision.

1.2 Project objectives :

Handwritten character recognition is a common task in the field of machine learning and computer vision. With the advent of deep learning, specifically Convolutional Neural Networks (CNNs), the accuracy of handwritten character recognition has drastically improved. The use of CNNs has proven to be particularly effective for recognizing handwritten characters, even with varying handwriting styles. In our project, we aim to use deep learning techniques to recognize Tifinagh handwritten characters, a script used in North Africa, and output the corresponding text. This will be achieved by training a CNN on a dataset of handwritten Tifinagh characters, and then using the trained model to classify new unseen characters.



```
this character seems to be : yazz ⵣ
```

Figure 1.1: Illustration of the envisaged problem solution

1.3 Methodology :

Our methodology consisted of several steps. First, we obtained the dataset of handwritten Tifinagh characters from Kaggle. We then preprocessed the data to resize and normalize the images. Next, we trained the deep learning model using a Convolutional Neural Network (CNN) architecture to recognize single Tifinagh characters. To improve accuracy on fragmented Tifinagh text, we developed a second model that segments the input image into individual characters before passing them through the recognition model. Finally, we developed a user interface , which allows users to input an image of a handwritten Tifinagh character and obtain the predicted characters.

1.4 Project schedule: Gantt chart

A Gantt chart is a calendar-like bar chart that lists tasks in columns, with the time scale represented on the x-axis. It provides an easy way to visualize project progress and anticipate upcoming tasks. It also helps to manage resource conflicts and potential delays by showing their impact on the project schedule. To ensure the success of our project, we have created a Gantt chart that outlines the forecasts for each phase of the project, while following a specific methodology for implementation. We have included below a chronological list of the tasks completed during our project .

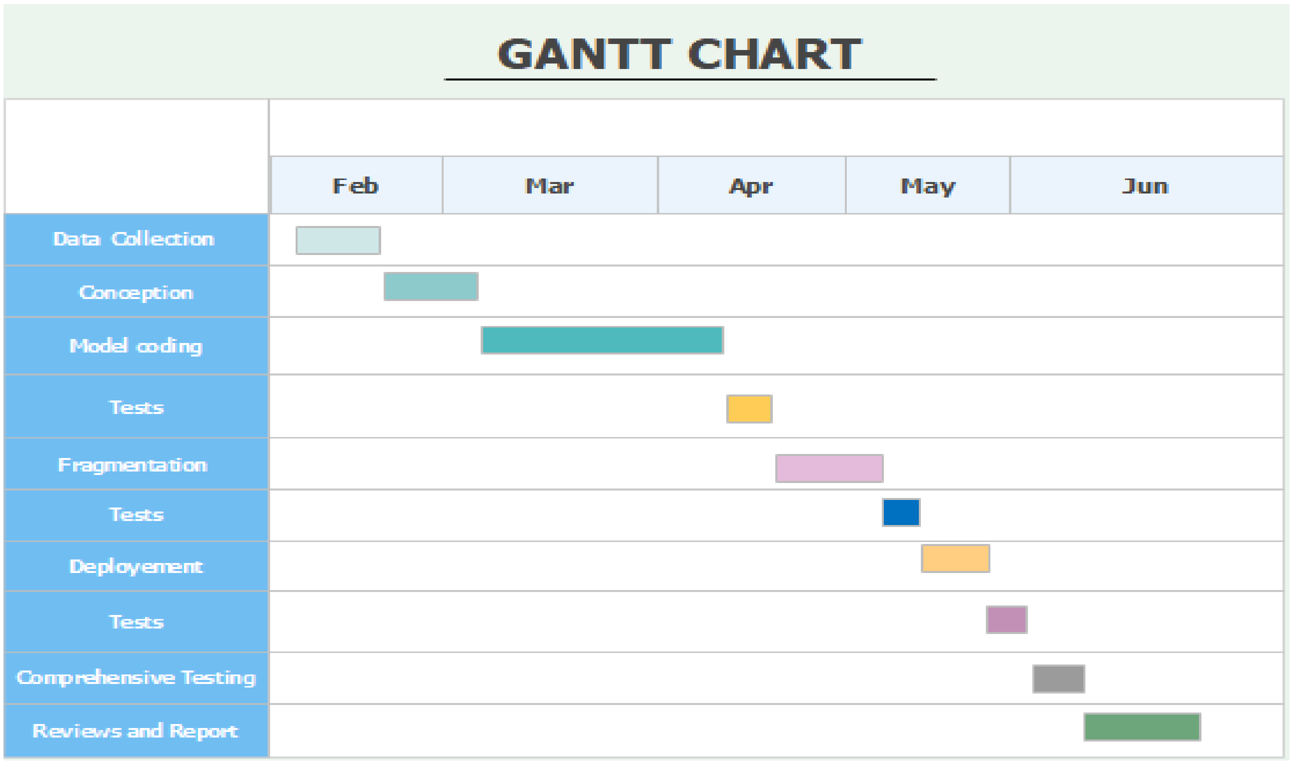


Figure 1.2: Gantt chart

Conclusion:

This chapter serves to break down the problem in an exhaustive manner and provide the reader with a fundamental understanding of the subject by examining the range of sub-problems encountered in its treatment and presenting the resulting neural architecture towards the end. Most of these discussed concepts will appear in a practical context in the following chapters.

Chapter 2

State of the art review and benchmarking

In this chapter, we will explore the basic concepts used for the recognition and fragmentation of Tifinagh characters, as well as its implementation with CNN (Convolutional Neural Networks).

1 Introduction :

We will first define the key concepts used in the implementation of the Tifinagh character recognition and fragmentation project. We will then discuss the different available solutions and compare them to determine the optimal combination of techniques for our specific project requirements.

2 Artificial Intelligence (AI):

The development in the technological field has improved over the years. Today, we hear about technological terms such as Artificial Intelligence, Machine Learning, and Deep Learning. We are often confused about these terms as they are defined in a similar manner. However, that is not an accurate definition, as these terms are different from each other. To explain the relationship between these techniques, we can use the relationship of membership illustrated in the following figure:

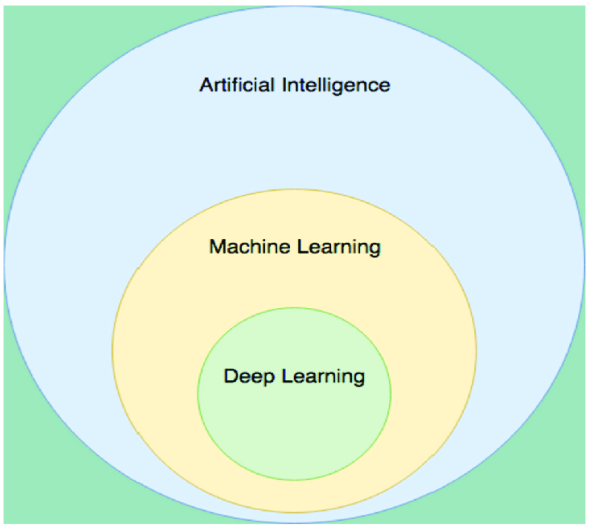


Figure 2.1: Relation between AI , ML and DL

AI, as its name suggests, is created by humans. Made like a complex machine, an AI program calculates properties and performs various actions to imitate human behavior. AI is a vast field where computers are trained to exhibit intelligent behavior. An AI program thinks and reacts like humans because it has a structure similar to the human brain. AI is the technology for the future of humanity that makes their lives better than before. The role of these technologies is to mimic human behaviors and actions well. These technologies prefer the best solution for tasks that we cannot perform. This effectively reduces human labor and can help them create multiple solutions. Some of the best applications of AI are facial recognition, object detection and classification, etc. However, using artificial intelligence in a general way in our lives is not possible so far because there are many features and functionalities of the human brain that are still far from being described.

3 Machine Learning (ML) :

Machine learning is an application of AI that enables systems to learn and improve from experience without being explicitly programmed. Machine learning [6] focuses on developing computer programs that can access data and use it to learn on their own.

Similar to how the human brain acquires knowledge and understanding, machine learning relies on input data, such as training data or knowledge graphs, to understand entities, domains, and connections between them. Once entities are defined, deep learning can begin.

The process of machine learning begins with observations or data, such as examples, direct experience, or instructions. It searches for patterns in the data so that it can make deductions based on the provided examples. The primary goal of machine learning is to enable computers to learn autonomously without human intervention or assistance and adapt their actions accordingly.

ML technology has proven valuable because it can solve problems at a speed and scale that the human mind alone cannot reproduce. With massive amounts of computing power behind a single task or multiple specific tasks, machines can be trained to identify patterns and relationships between input data and automate routine processes.

In some scenarios, the machine receives a significant amount of labeled training data, known as supervised learning. In other cases, no labeled data is provided, which is called unsupervised learning. Finally, Reinforcement Learning involves an autonomous agent learning actions to take, based on experiences, to optimize a quantitative reward over time.

3.1 Supervised learning :

Supervised learning is an approach to creating artificial intelligence (AI), where a computer algorithm is trained on input data that has been labeled for a particular outcome. Supervised learning deals with two types of problems:classification problems and regression problems.

3.2 Unsupervised learning :

Unsupervised learning is a completely different approach to creating artificial intelligence. In short, there is no complete and clean labeled dataset in unsupervised learning. Unsupervised learning is self-organized learning. Its main goal is to explore underlying patterns and predict output. In this case, we provide data to the machine and ask it to search for hidden features and group the data in a logical way. Examples of unsupervised learning : o K – Means clustering o Neural Networks o Principal Component Analysis

3.3 Reinforcement Learning :

Reinforcement learning is a type of machine learning that is neither based on supervised nor unsupervised learning. Instead, it focuses on enabling an agent to learn how to react to an environment by itself. Reinforcement learning has gained popularity and has led to the development of a variety of learning algorithms that are useful in fields such as robotics and gaming.

In reinforcement learning, an agent always starts with an initial state and has a goal of reaching a final state. However, there can be different paths to reach the final state. The agent tries to manipulate the environment by moving from one state to another, and it receives a reward (or appreciation) if it succeeds, but no reward or appreciation if it fails. In this way, the agent learns from the environment through trial and error.

4 DEEP LEARNING (DL) :

4.1 Introduction:

Deep learning is a machine learning technique that constructs artificial neural networks to imitate the structure and functioning of the human brain. In practice, deep learning, also known as deep structured learning or hierarchical learning, uses a large number of hidden layers - usually more than 6, but often many more - of non-linear processing to extract features from data and transform the data into different levels of abstraction (representations).

For example, suppose the input data is a matrix of pixels. The first layer typically abstracts the pixels and recognizes feature edges in the image. The next layer may build simple features from the edges, such as leaves and branches. The next layer could then recognize a tree and so on. The passing of data from one layer to the next is considered a transformation, transforming the output of one layer into the input for the next. Each layer corresponds to a different level of abstraction, and the machine can learn by itself which features of the data to place in which layer/level. Deep learning differs from traditional "shallow learning" because it learns much deeper levels of abstraction and representation.

This learning technique is a revolutionary tool for processing large amounts of data, as machine performance improves as it analyzes more data. As the machine also learns from processed data, it is capable of performing feature extraction and abstraction automatically from raw data with little or no human intervention.

Neural networks are algorithms inspired by neurons in our brain. Designed to recognize patterns in complex data, they often produce the best results when searching for patterns in audio files, images, or videos. Neural networks are simply made up of neurons (also called nodes). These nodes are connected in some way. So, each neuron contains a number and each connection contains a weight. These neurons are spread out among input, hidden, and output layers. There are actually many layers, and usually, there is no optimal number of layers.

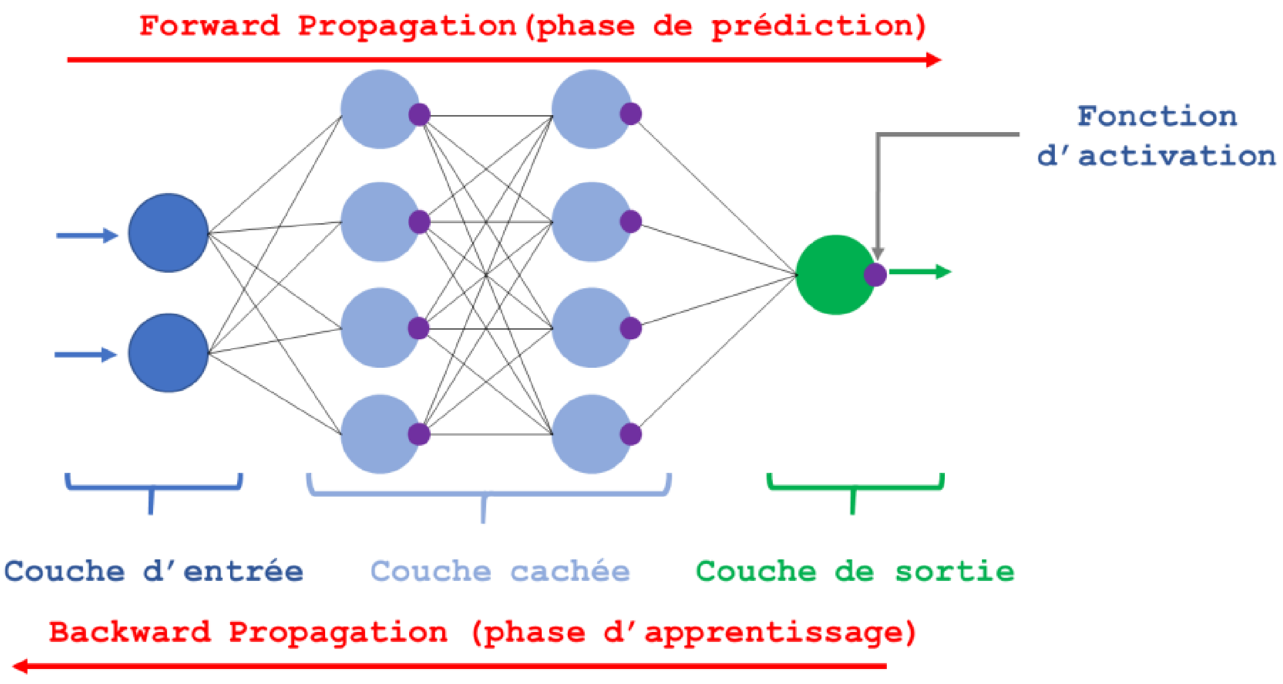

Forward Propagation (phase de prédiction)
Fonction d'activation
Couche d'entrée
Couche cachée
Couche de sortie
Backward Propagation (phase d'apprentissage)

Figure 2.2: Schema of an artificial neural network

4.2 ANN (Artificial Neural Network) :

ANN stands for Artificial Neural Network. It is a type of machine learning algorithm that is inspired by the human brain. ANNs are made up of interconnected nodes, which are similar to neurons in the brain. These nodes are arranged in layers, and each layer performs a specific function. The first layer receives input data, the middle layers process the data, and the last layer produces an output.

ANNs are trained using a process called backpropagation. In backpropagation, the ANN is given a set of input data and the desired output. The ANN then tries to adjust its weights so that it can produce the desired output for the given input data. This process is repeated until the ANN is able to produce the desired output for most of the input data.

ANNs can be used for a variety of tasks, including image recognition, speech recognition, and natural language processing. They are also being used in a variety of other fields, such as finance, healthcare, and transportation.

Here are some of the advantages of ANNs:

- They can learn from data and improve their performance over time. - They can be used to solve complex problems that would be difficult or impossible to solve with traditional algorithms. - They are very versatile and can be used for a variety of tasks.

4.3 RNN (Recurrent Neural Networks) :

Recurrent Neural Networks, or RNNs, are a popular methodology for supervised deep learning. They are mainly used for analyzing time series data and when working with a sequence of data. In this type of work, the network learns from what it has just observed, i.e., short-term memory. Therefore, it resembles the frontal lobe of the brain. RNNs work in three steps. In the first step, it moves forward into the hidden layer and makes a prediction. In the second step, it compares its prediction to the actual value using the loss function. The loss function highlights the performance of a model. The lower the value of the loss function, the better the model’s performance. In the last step, it uses the error values in backpropagation, which then calculates the gradient for each point (node). The gradient is the value used to adjust the weights of the network at each point.

RNNs become very inefficient when the gap between relevant information and the point where it is needed becomes very large. This is because information is passed along at each step, and the longer the chain, the more likely it is for information to be lost along the way.

In theory, RNNs could learn these long-term dependencies. In practice, they often struggle to do so. The LSTM, a special type of RNN, attempts to address this type of problem.

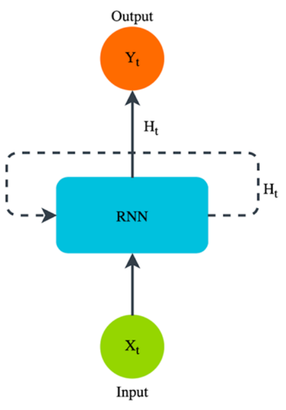


Figure 2.3: Recurrent Neural Network Architecture

4.4 LSTM (Long short-term memory networks) :

LSTM stands for Long Short Term Memory. It is a type of recursive neural network (RNN) specifically designed for long-term addiction learning. RNNs are a type of neural network that can process data sequences such as text or speech. LSTMs are a special type of RNN that can learn long-term dependencies, meaning they can remember information from previous steps in a sequence. This makes LSTMs ideal for tasks such as machine translation, speech recognition, and natural language processing.LSTMs use a trigger mechanism to control the flow of information through the network. This trigger mechanism consists of three doors: an oblivion door, an entrance door, and an exit door. The forgotten port decides what information to forget from the previous state, the input port decides what information to add to the current state, and the output port decides what information to extract from the current state.LSTM has proven to be very effective in long-term addiction learning. These have been used to achieve state-of-the-art results in a variety of tasks such as machine translation, speech recognition, and natural language processing.

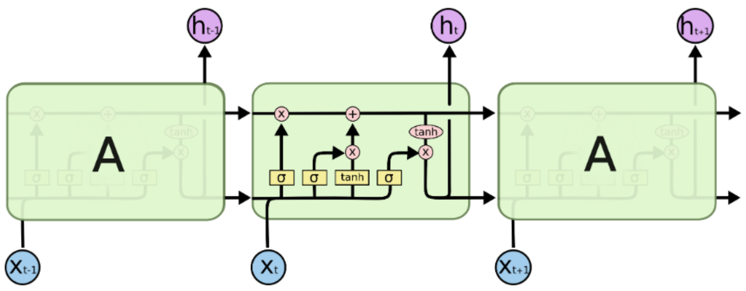


Figure 2.4: LSTM Architecture

4.5 CNN (Convolutional Neural Network):

CNN stands for Convolutional Neural Network. It is a type of artificial neural network that is specifically designed for image recognition. CNNs are made up of a series of layers, each of which performs a specific function. The first layer receives the input image, the middle layers extract features from the image, and the last layer classifies the image. CNNs work by using a convolution operation to extract features from images. A convolution operation is a mathematical operation that takes two functions and produces a third function. In the case of CNNs, the two functions are the input image and a filter. The filter is a small array of numbers that is used to extract features from the image. The convolution operation is performed by sliding the filter over the image. At each location, the filter is multiplied by the pixels in the image and the sum is calculated. This sum is then used to update the output of the layer. The convolution operation is repeated for each layer in the CNN. As the image moves through the network, more and more features are extracted. The final layer of the network classifies the image based on the extracted features. CNNs have been shown to be very effective at image recognition. They have been used to achieve state-of-the-art results on a variety of tasks, such as object detection, face recognition, and medical image analysis.

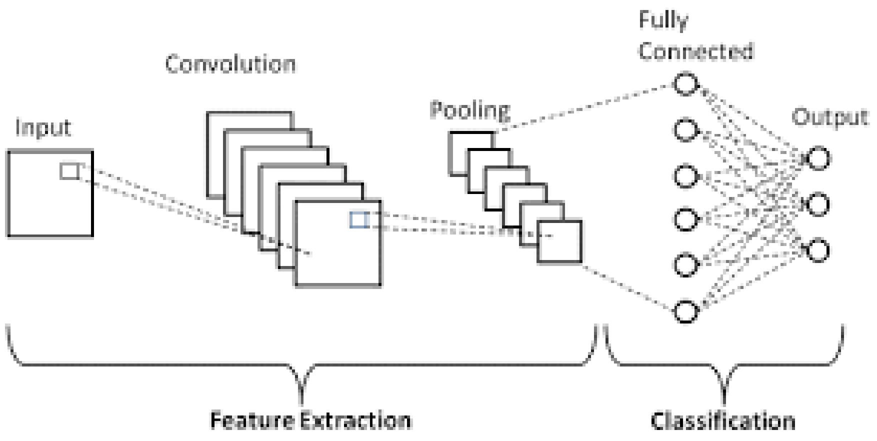


Figure 2.5: CNN Architecture

The layers of the Convolutional Neural Network (CNN) are responsible for extracting features from the image and classifying it. The different CNN layers are:Entry Level: The Entry Level is the top level of CNN. Receives the input image and passes it to the next level.

- Convolution Layer: The convolution layer is the heart of CNN. Use a convolution operation to extract features from an image.The convolution operation is performed by sliding a filter over the image. The filter is a small array of numbers that is used to extract features from the image.

- Pooling layer: The pooling layer is used to reduce the size of the feature maps. The pooling operation is performed by taking the maximum value of a small region of the feature map. This reduces the size of the feature maps without losing too much information.

- Fully Connected Level: The fully connected level is the last level of CNN. Classify the image based on the extracted features. A fully connected layer is a traditional neural network layer that uses a matrix multiplication operation to classify the image.

Different CNN levels are linked by weights. Scales are learned during the training process.During the training process, the weights are adjusted so that CNN can correctly rate the images.The number of shifts in the CNN can vary depending on the task at hand. For simple tasks, a CNN can only have a few layers. For more complex tasks, a CNN can have tens or even hundreds of layers.The architecture of theCNN is a key factor in its performance.The CNN architecture refers to the number and type of layers and the connections between layers. The CNN architecture is usually determined by trial and error.

4.6 Pre-trained models :

A pre-trained model is a machine learning model that has already been trained on a large dataset. This means that the model has already learned to extract features from the data and make predictions. Pre-trained models can be used to solve a variety of problems, such as image classification, natural language processing, and speech recognition.

There are many advantages to using pre-trained models. First, they can save time and money. It can take a long time and a lot of data to train a machine learning model from scratch. Pre-trained models can be used to solve problems quickly and without the need for a large dataset.

Second, pre-trained models can be more accurate than models that are trained from scratch. This is because pre-trained models have already learned to extract features from data. This means that they can make better predictions than models that are still learning.

Finally, pre-trained models can be used to solve a variety of problems. This is because they have been trained on a large dataset. This means that they can be used to solve problems that are different from the problems that they were originally trained on.

There are a few disadvantages to using pre-trained models. First, they can be expensive. Pre-trained models are often trained on large datasets, which can be expensive to collect and store.

Second, pre-trained models can be biased. This is because they are trained on datasets that are collected from the real world. This means that they can reflect the biases that exist in the real world.

Finally, pre-trained models can be difficult to customize. This is because they are trained on a specific task. This means that they may not be able to be used to solve other tasks.

Overall, pre-trained models are a powerful tool that can be used to solve a variety of problems. They can save time, money, and accuracy. However, they can also be expensive and biased. It is important to weigh the advantages and disadvantages of pre-trained models before using them.

There are many pre-trained convolutional neural network (CNN) models available. Some of the most popular pre-trained CNN models include:

- VGGNet: VGGNet is a convolutional neural network that was developed by the Visual Geometry Group at the University of Oxford. VGGNet was one of the first convolutional neural networks to achieve state-of-the-art results on the ImageNet dataset.

- ResNet: ResNet is a convolutional neural network that was developed by Microsoft Research. ResNet stands for residual network. ResNet was designed to address the vanishing gradient problem, which is a problem that can occur in deep neural networks.

- InceptionNet: InceptionNet is a convolutional neural network that was developed by Google. InceptionNet was designed to improve the efficiency of convolutional neural networks. InceptionNet uses a technique called inception modules to extract features from images.

4.7 Transfer learning :

Transfer learning is a machine learning technique that allows you to use a model trained on one task to solve another related task. This can be achieved by taking a pre-trained model and adapting it for a new task. Optimization is the adjustment of model parameters to better fit new data.Transfer learning can be a very effective way to improve the performance of a machine learning model, especially when limited data is available for a new task. In fact, the pre-trained model has already learned to extract features from the data that can be reused for a new task.There are two main approaches to learning transfer:

- Feature Extraction: This approach uses a pre-trained model to extract features from new data.These functions are then used to train a new classifier or regressor.

- Fine-tuning: This approach optimizes a pre-trained model based on new data. The model parameters have to be adjusted so that they better fit the new data.transfer learning has been used to achieve breakthrough results in a variety of tasks including image classification.

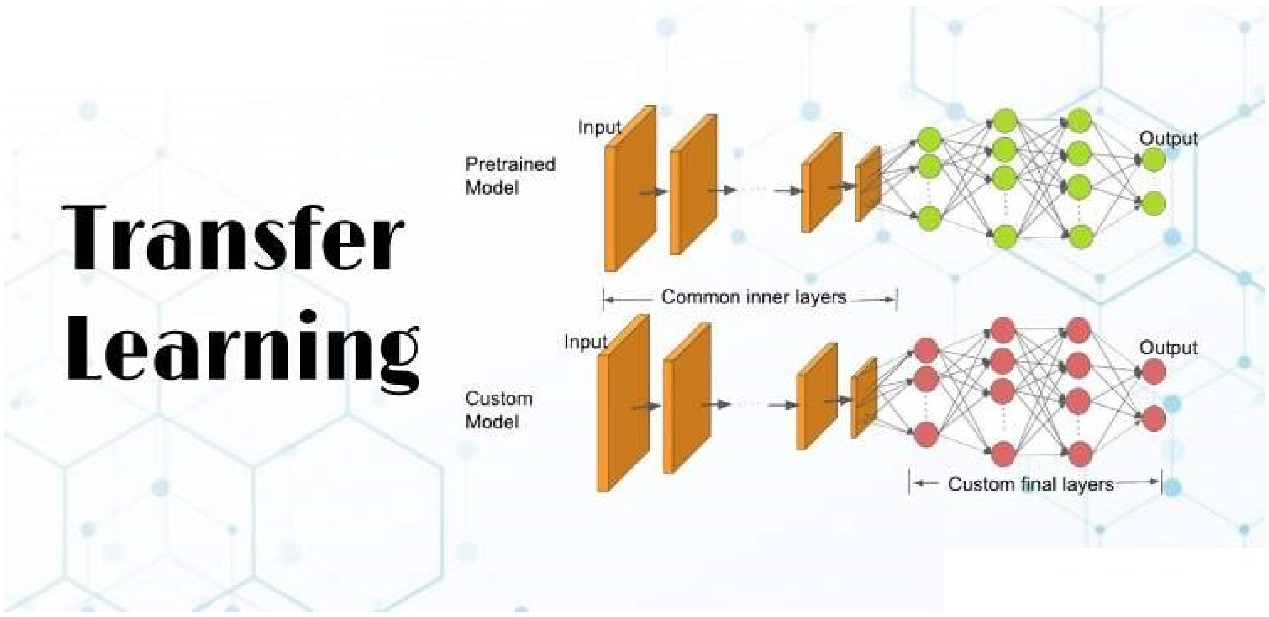


Figure 2.6: Transfer learning

5 Conclusion :

To summarize, this chapter has provided an overview of different Machine Learning (ML) and Deep Learning (DL) technologies, along with various neural network architectures. These technologies and architectures are crucial for the development of artificial intelligence projects.

Having gained an understanding of these technologies and architectures, we can now proceed to the design and modeling phase of our project. This involves leveraging relevant datasets related to our subject and making use of pre-trained models that are available.

The use of appropriate datasets specific to our project is essential for achieving accurate and reliable results. These datasets can be collected, annotated, and prepared specifically for our task, serving as a foundation for training and improving our models over time.

In addition to datasets, we can also benefit from existing pre-trained models. These models have been trained on large-scale datasets and can be fine-tuned or utilized as a starting point for our specific project, saving time and resources.

By combining our understanding of ML and DL technologies, neural network architectures, and the utilization of datasets and pre-trained models, we can effectively design and model our project, aiming to achieve optimal results in our domain.

Chapter 3

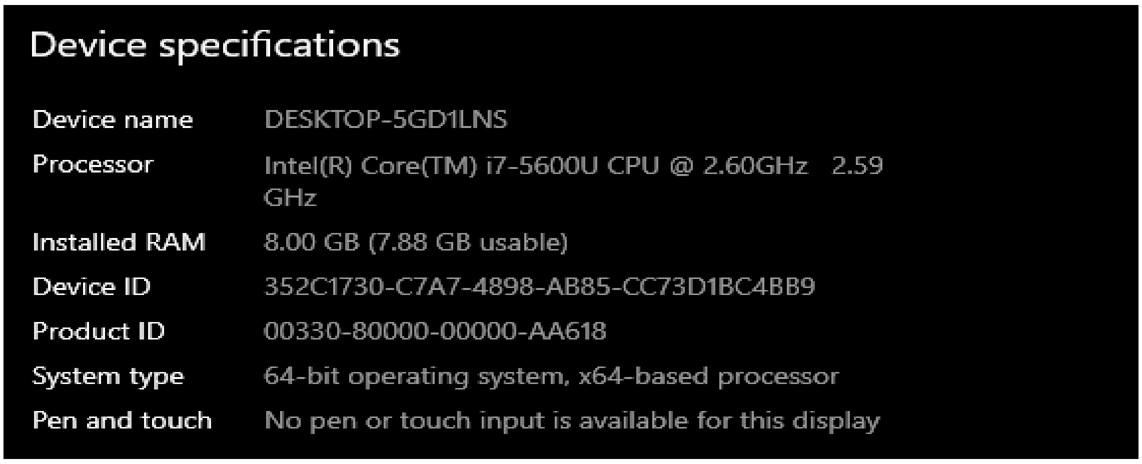
Project conception and modeling

This chapter provides an overview of the working environment, description of the dataset and model utilized initially, and the implementation as well as the analysis of the results.

1 Working Environment

1.1 Hardware Environment

To carry out our project, we used a computer characterized by the following specifications:



Device specifications

Device name	DESKTOP-5GD1LNS
Processor	Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz 2.59 GHz
Installed RAM	8.00 GB (7.88 GB usable)
Device ID	352C1730-C7A7-4898-AB85-CC73D1BC4BB9
Product ID	00330-80000-00000-AA618
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 3.1: Characteristics of the computer used

After starting the project, the technical specifications of this computer were no longer sufficient to achieve the desired performance for the calculations required in model training, especially for Deep Learning models that can benefit from powerful GPUs if available.

Therefore, we are seeking alternatives to train the models and conduct necessary experiments in the cloud, as it is the easiest and cost-effective solution.

1.1.1 Kaggle :

kaggle

Figure 3.2: Kaggle

Kaggle is an online community platform designed for data scientists and machine learning enthusiasts. It provides free access to NVIDIA TESLA P100 GPUs, which are valuable for training deep learning models, although they do not accelerate most other workflows (i.e., libraries like pandas and scikit-learn do not benefit from GPU access). Users can utilize a weekly quota of GPU hours, which is reset every week and typically allows for 30 hours or more, depending on demand and resources. We are working within the space provided by Kaggle by creating a verified account to benefit from the allocated GPU hours.

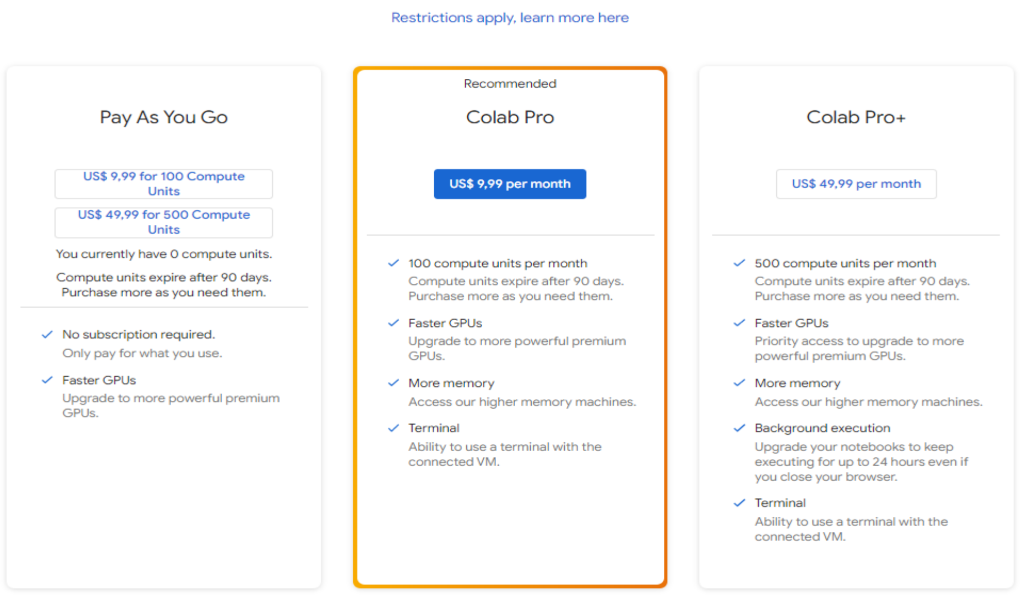
1.1.2 Google Colab :

Figure 3.3: Google Colab

Google Colab or Colaboratory is a cloud service offered by Google that is based on Jupyter Notebook and is designed for machine learning training and research. This platform enables training machine learning models directly in the cloud without the need to install anything on your computer, except for a web browser.

Google Colab is a comprehensive tool for quickly training and testing machine learning models without hardware constraints. It is characterized by the availability of GPUs and 12GB of RAM. One of its notable features is that it is accessible to everyone for free.

1.1.3 Google Colab Pro :



Restrictions apply, learn more here

Pay As You Go

US$ 9,99 for 100 Compute Units

US$ 49,99 for 500 Compute Units

You currently have 0 compute units.

Compute units expire after 90 days. Purchase more as you need them.

✓ No subscription required.
Only pay for what you use.

✓ Faster GPUs
Upgrade to more powerful premium GPUs.

Recommended

Colab Pro

US$ 9,99 per month

✓ 100 compute units per month
Compute units expire after 90 days. Purchase more as you need them.

✓ Faster GPUs
Upgrade to more powerful premium GPUs.

✓ More memory
Access our higher memory machines.

✓ Terminal
Ability to use a terminal with the connected VM.

Colab Pro+

US$ 49,99 per month

✓ 500 compute units per month
Compute units expire after 90 days. Purchase more as you need them.

✓ Faster GPUs
Priority access to upgrade to more powerful premium GPUs.

✓ More memory
Access our higher memory machines.

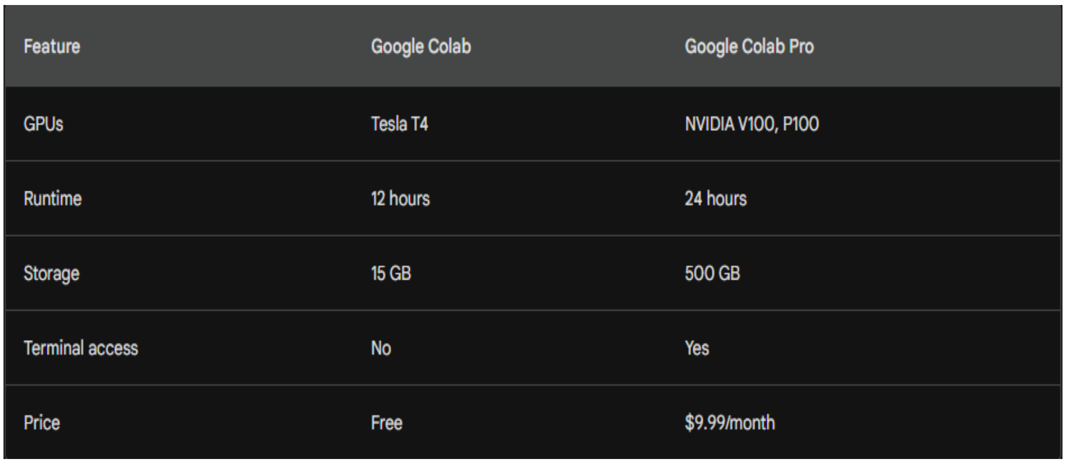
✓ Background execution
Upgrade your notebooks to keep executing for up to 24 hours even if you close your browser.

✓ Terminal
Ability to use a terminal with the connected VM.

Figure 3.4: Google Colab Pro

We encountered certain limitations when running our deep learning code for this project. Since it involved a deep model with a large amount of data, training took longer with the GPU provided by Colab. It occasionally reached the maximum runtime limit and disconnected from the server.

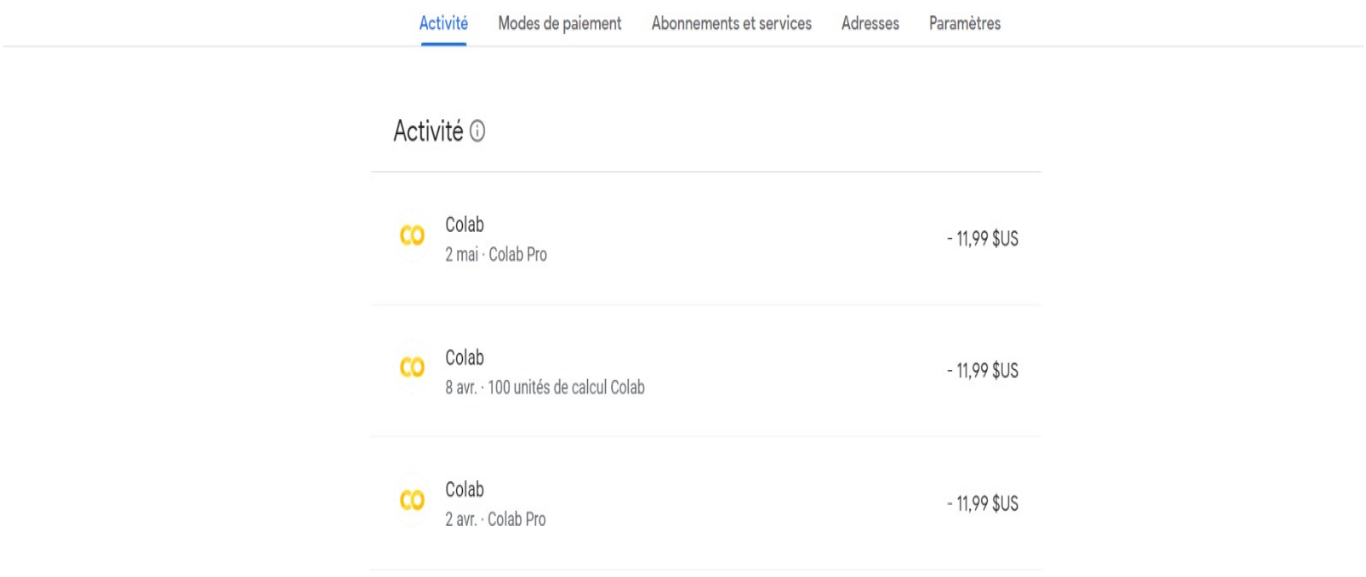
Colab Pro provides a solution for faster training with improved GPUs and also offers longer runtimes, which can help reduce disconnections.



Feature	Google Colab	Google Colab Pro
GPUs	Tesla T4	NVIDIA V100, P100
Runtime	12 hours	24 hours
Storage	15 GB	500 GB
Terminal access	No	Yes
Price	Free	$9.99/month

Figure 3.5: Google Colab VS Google Colab Pro

Therefore, we have decided to work exclusively on "Google Colab Pro" for all our experiments due to the large amount of data and the complexity of the calculations involved.



Activité

CO	Colab 2 mai · Colab Pro	- 11,99 $US
CO	Colab 8 avr. · 100 unités de calcul Colab	- 11,99 $US
CO	Colab 2 avr. · Colab Pro	- 11,99 $US

Figure 3.6: Payment

1.2 Programming Environment :

1.2.1 Python :

Figure 3.7: Python

Python is an open-source programming language that supports structured, functional, and object-oriented programming paradigms. It features strong dynamic typing, automatic memory management, and an exception handling system. It is also favored by educators for its clear syntax, which separates low-level mechanisms and allows for easy introduction to fundamental programming concepts.

1.2.2 Jupyter :

Figure 3.8: Jupyter

Jupyter Notebook is an open-source web application that allows you to create and share documents containing live code, visualizations, and explanatory text. It provides an interactive computing environment where you can write and execute code in multiple programming languages, including Python, R, and Julia.

1.2.3 TensorFlow :

Figure 3.9: Tensorflow

TensorFlow is an open-source machine learning library developed by Google. It is widely used for various tasks in deep learning, such as neural networks, natural language processing, and computer vision. TensorFlow provides a flexible and efficient framework for building and training different types of machine learning models.

The core component of TensorFlow is its computational graph, which is a series of operations represented as nodes connected by edges. These operations are typically expressed using tensors, which are multidimensional arrays or matrices. TensorFlow allows you to define and execute complex mathematical computations efficiently by automatically optimizing the execution flow and utilizing hardware acceleration, such as CPUs and GPUs.

TensorFlow supports both CPU and GPU computations, and it offers support for NVIDIA's CUDA platform to leverage the power of GPUs. This enables faster and more efficient execution of deep learning algorithms, especially for large-scale models and datasets. It provides the capability to compute gradients, making it easy to apply optimization algorithms such as gradient descent.

1.2.4 Keras :

Figure 3.10: Keras

Keras, on the other hand, is a high-level neural networks API that can run on top of TensorFlow. It provides a user-friendly interface for building and training neural networks without having to deal with the low-level details of TensorFlow. Keras simplifies the process of creating deep learning models by offering a set of intuitive abstractions and pre-built layers, allowing users to rapidly prototype and experiment with different architectures.

TensorFlow and Keras are often used together, where TensorFlow serves as the backend for Keras. This combination provides the benefits of both libraries: the flexibility and computational power of TensorFlow, along with the simplicity and ease of use of Keras. It enables developers to leverage the extensive ecosystem and community support of TensorFlow while enjoying the high-level API provided by Keras.

1.2.5 Numpy :

Figure 3.11: Numpy

NumPy is a Python library widely used for numerical computations and data manipulation. It provides a powerful array object that allows for efficient storage and manipulation of large datasets. With NumPy, you can perform mathematical operations on arrays, such as addition, subtraction, multiplication, and more. Its extensive collection of functions and methods makes it an essential tool for tasks involving numerical analysis, linear algebra, statistics, and beyond.

1.2.6 Pandas :

Figure 3.12: Pandas

Pandas is a versatile and powerful data manipulation library for Python, widely used in data analysis and data science tasks. It provides intuitive data structures, primarily the DataFrame, which allows for efficient handling and analysis of structured data. With Pandas, you can easily perform operations like data cleaning, filtering, grouping, and merging, making it a go-to tool for data preprocessing tasks. It also offers advanced functionality such as time series analysis, missing data handling, and data visualization capabilities, making it a comprehensive library for data manipulation and analysis in Python.

1.2.7 Matplotlib :

Figure 3.13: Matplotlib

Matplotlib is a widely used Python library for creating static, animated, and interactive visualizations. It provides a comprehensive set of plotting tools and functions, allowing you to create various types of plots, including line plots, scatter plots, bar plots, histograms, and more. Matplotlib is a powerful tool for data exploration, presentation, and communication in scientific computing, data analysis, and machine learning.

1.2.8 OpenCV:

Figure 3.14: OpenCV

OpenCV (Open Source Computer Vision) is a popular open-source computer vision and image processing library. It provides a wide range of functions and algorithms for tasks such as image and video manipulation, object detection and tracking, facial recognition, and more. It offers efficient and optimized implementations of computer vision algorithms, making it suitable for real-time applications.

2 Dataset Description:

2.1 Tifinagh :

Tifinagh is an ancient script used to write the Berber languages, spoken by various ethnic groups in North Africa, particularly in Morocco, Algeria, and Tunisia. The term "Tifinagh" can refer to both the script itself and the Berber alphabet derived from it. It is believed to have originated from the Libyco-Berber script and has undergone several changes over time. Historically, Tifinagh was primarily used for inscriptions on stone or other durable materials, often found in ancient rock art and monuments. In recent years, there has been a resurgence of interest in the use of Tifinagh as a writing system for the Berber languages. Efforts have been made to standardize the script and promote its use in education, literature, and digital communication. As a result, Tifinagh is now recognized as an official script in Morocco and is used in various contexts, such as road signs, publications, and official documents. The Tifinagh alphabet consists of around 33 , depending on the specific variant and the Berber language being written. The number of letters may vary slightly between different regions or communities that use the Tifinagh script. These letters represent consonant sounds, and additional diacritical marks are used to indicate vowel sounds. The exact number and form of the characters can vary, as there have been historical and regional variations of the Tifinagh script. However, efforts have been made to standardize the alphabet in recent years, leading to a more consistent set of characters across different Berber language communities.

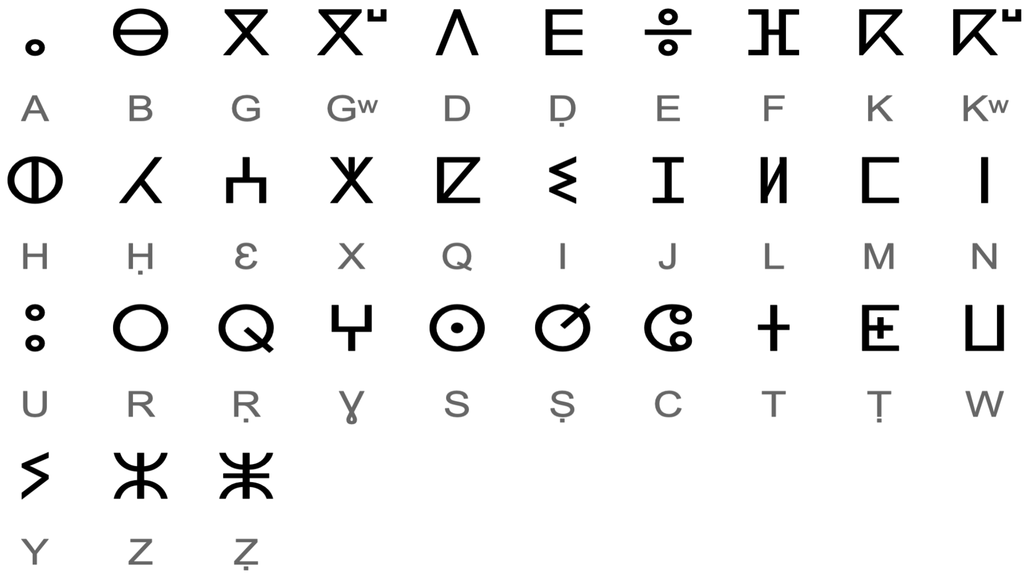
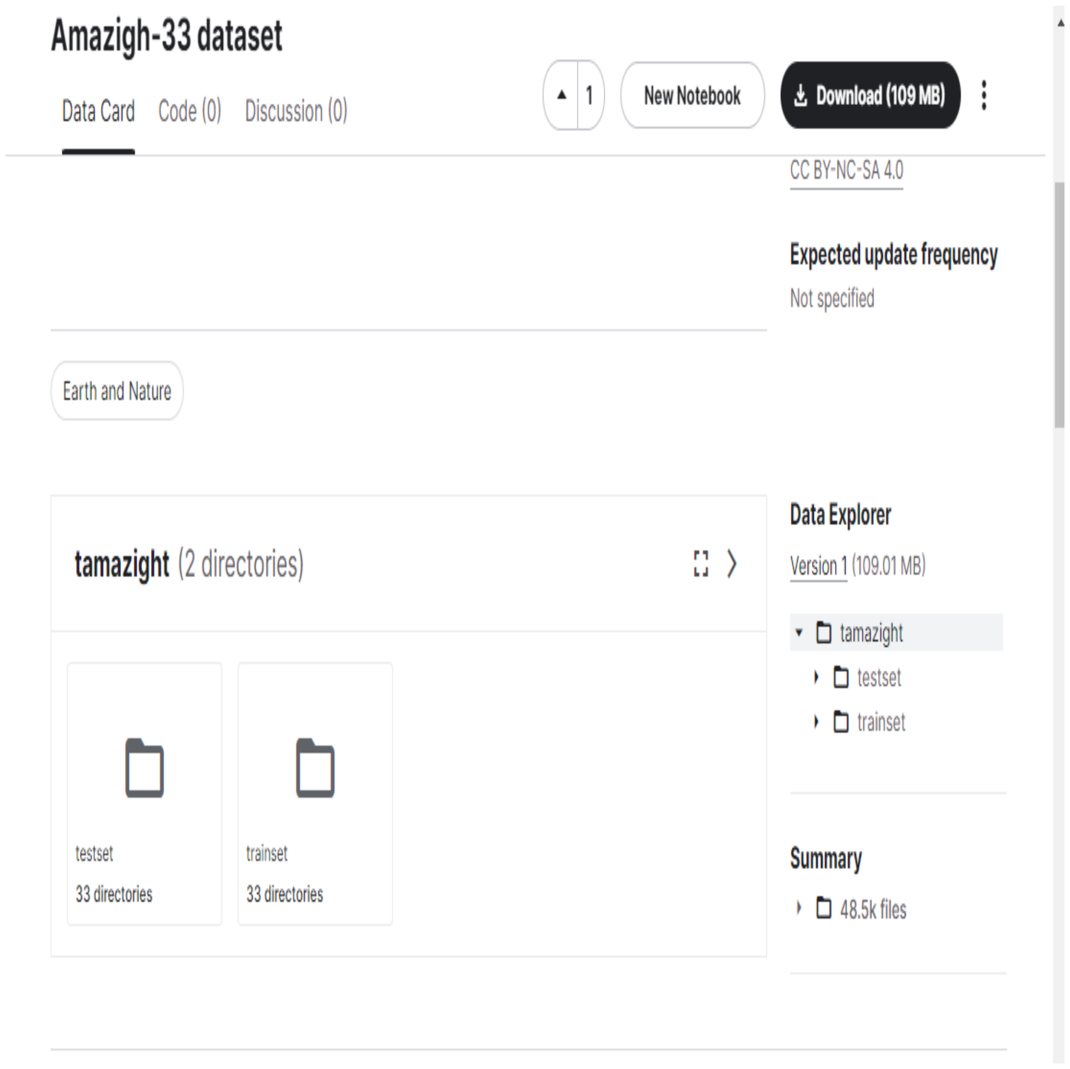


Figure 3.15: Tifinagh alphabet

2.2 Dataset Benchmark :

Our project aims to develop a tool for recognizing handwritten Tifinagh characters, and to achieve this, we need a dataset of these characters. Fortunately, we were able to find a dataset on Kaggle. This dataset will be instrumental in training our recognition system to accurately identify and classify handwritten Tifinagh characters. By leveraging this dataset, we can improve the performance and reliability of our tool, ultimately providing users with a robust solution for recognizing Tifinagh characters written by hand.



Amazigh-33 dataset

Data Card Code (0) Discussion (0)

1 New Notebook Download (109 MB)

CC BY-NC-SA 4.0

Expected update frequency

Not specified

Earth and Nature

tamazight (2 directories)

testset
33 directories

trainset
33 directories

Data Explorer

Version 1 (109.01 MB)

tamazight
testset
trainset

Summary

48.5k files

Figure 3.16: Dataset examples

2.2.1 Dataset Description :

With approximately 1,500 images for each of the 33 Tifinagh characters, we have a significant dataset to start our character recognition project.

It’s important to note that the effectiveness of our recognition model will depend not only on the dataset size but also on its quality and diversity.We make sure that the images in our dataset cover different variations of each character, including different writing styles, slants, sizes, and handwriting quality. The more diverse the dataset, the better our model will be able to generalize and recognize handwritten Tifinagh characters in various conditions.

An indicator of the variety of Tifinagh characters in our dataset is the distribution of the characters themselves, as illustrated in the figure.

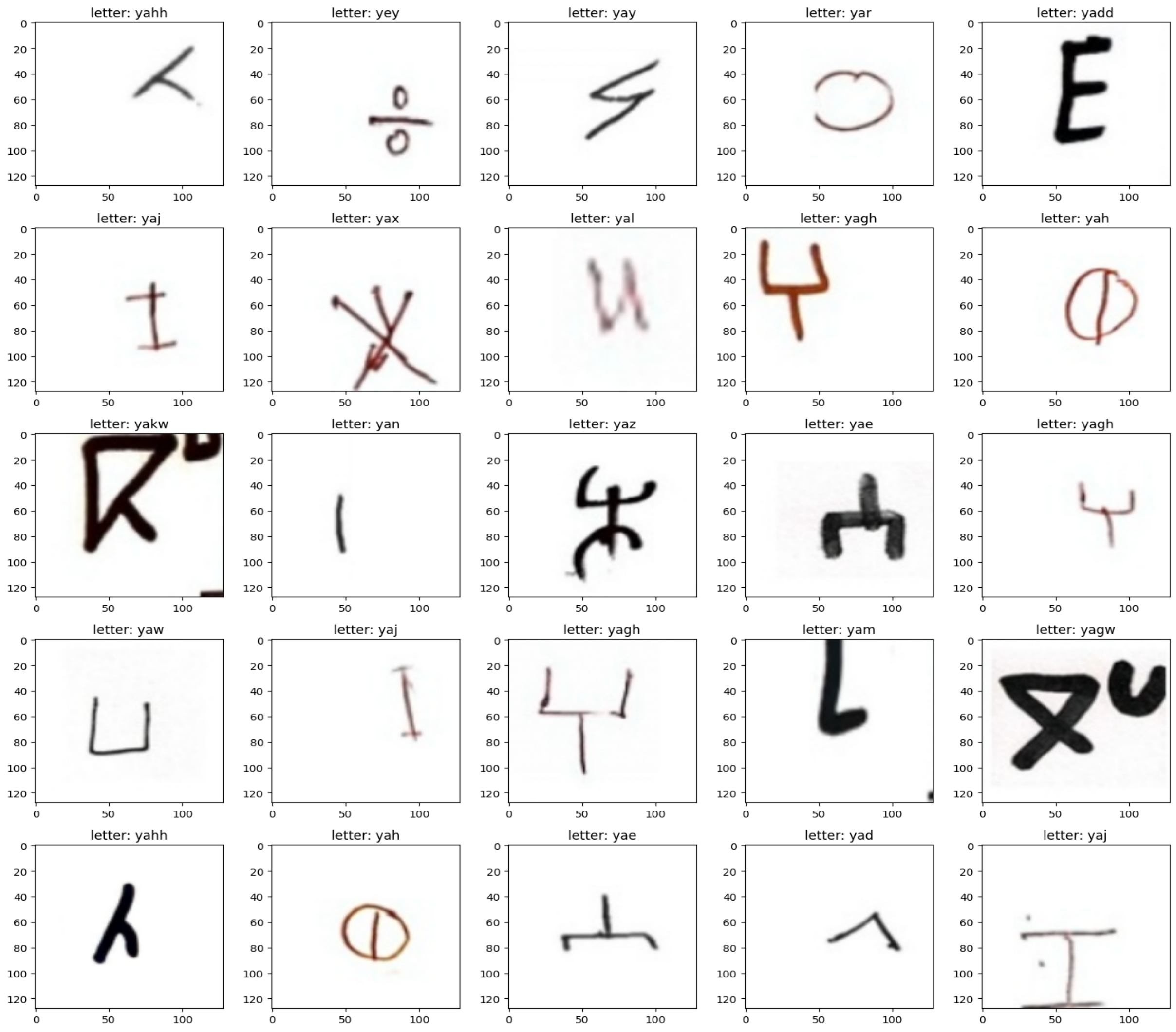


Figure 3.17: Dataset examples

3 Model selection :

3.1 Types of Pre-trained models:

As we know, there are several types of pretrained models to use in Convolutional Neural Networks (CNN), such as:

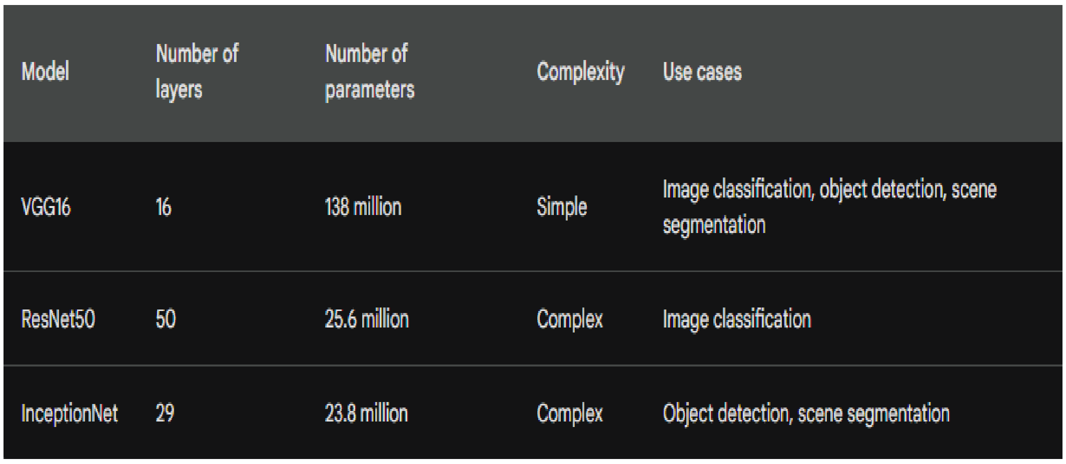
1- VGGNet is a convolutional neural network (CNN) architecture that was developed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group at the University of Oxford. It was first introduced in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" in 2014.

2- ResNet, short for Residual Network, is a convolutional neural network (CNN) architecture that was developed by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun of Microsoft Research. It was first introduced in the paper "Deep Residual Learning for Image Recognition" in 2015.

3- InceptionNet is a convolutional neural network (CNN) architecture that was developed by Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich of Google Brain. It was first introduced in the paper "Going deeper with convolutions" in 2014.

3.2 Comparaison of Pre-trained models:

These pretrained models provide a starting point for various image recognition tasks and can be fine-tuned on our specific Tifinagh character recognition dataset. The choice of model depends on factors such as the complexity of our characters, the size of our dataset, and the available computational resources.



Model	Number of layers	Number of parameters	Complexity	Use cases
VGG16	16	138 million	Simple	Image classification, object detection, scene segmentation
ResNet50	50	25.6 million	Complex	Image classification
InceptionNet	29	23.8 million	Complex	Object detection, scene segmentation

Figure 3.18: Dataset examples

In our project, we can use both VGG16 and ResNet50 to train and evaluate different models.We can then compare the results of the two models to see which one performs better on our specific dataset.

3.3 ResNet-50 architecture:

ResNet-50 is a convolutional neural network that is 50 layers deep. The ResNet-50 architecture consists of the following layers:

- A convolutional layer with 7x7 kernels and 64 filters.

- A max pooling layer with 2x2 kernels and stride 2.

- 34 residual blocks.

- A global average pooling layer.

- A fully connected layer with 1000 neurons.

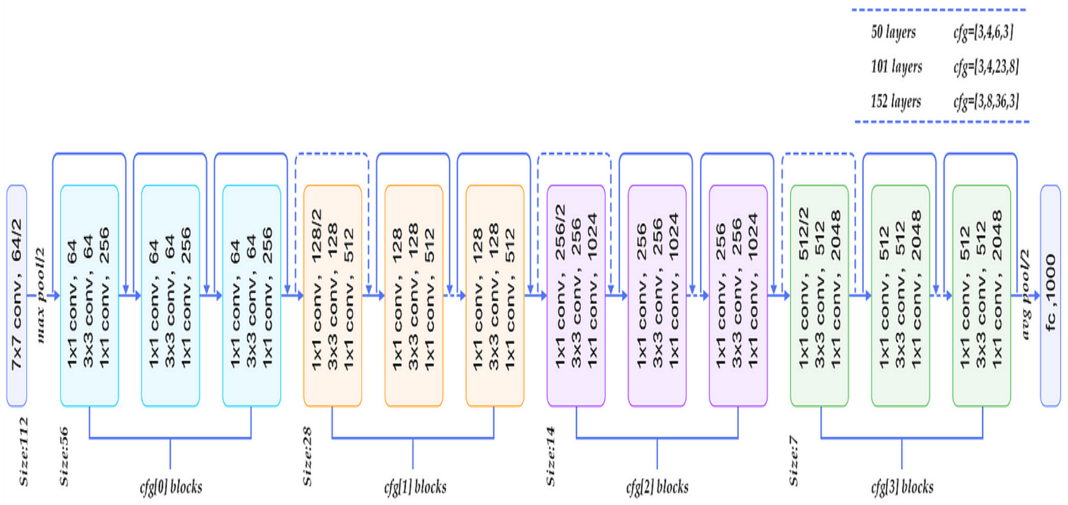

50 layers cfg=[3,4,6,3]
101 layers cfg=[3,4,23,8]
152 layers cfg=[3,8,36,3]
7x7 conv, 64/2
max pool/2
1x1 conv, 64 3x3 conv, 64 1x1 conv, 256
1x1 conv, 128/2 3x3 conv, 128 1x1 conv, 512
1x1 conv, 128 3x3 conv, 128 1x1 conv, 512
1x1 conv, 256/2 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 256 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 512/2 3x3 conv, 512 1x1 conv, 2048
1x1 conv, 512 3x3 conv, 512 1x1 conv, 2048
avg pool/2
fc ,1000
Size:112
Size:56
Size:28
Size:14
Size:7
cfg[0] blocks
cfg[1] blocks
cfg[2] blocks
cfg[3] blocks

Figure 3.19: ResNet-50 architecture

ResNet50 can achieve an accuracy of 80.4 % on ImageNet.

3.4 VGG-16 architecture:

VGG-16 is a deep convolutional network with 16 layers. The VGG16 architecture consists of the following layers:

- 2 convolutional layers with 64 filters each.

- 2 max pooling layers with 2x2 kernels and stride 2.

- 2 convolutional layers with 128 filters each.

- 2 max pooling layers with 2x2 kernels and stride 2.

- 3 convolutional layers with 256 filters each.

- 3 max pooling layers with 2x2 kernels and stride 2.

- 3 convolutional layers with 512 filters each.

- 3 max pooling layers with 2x2 kernels and stride 2.

- 2 fully connected layers with 4096 neurons each.

- A softmax layer with 1000 neurons.

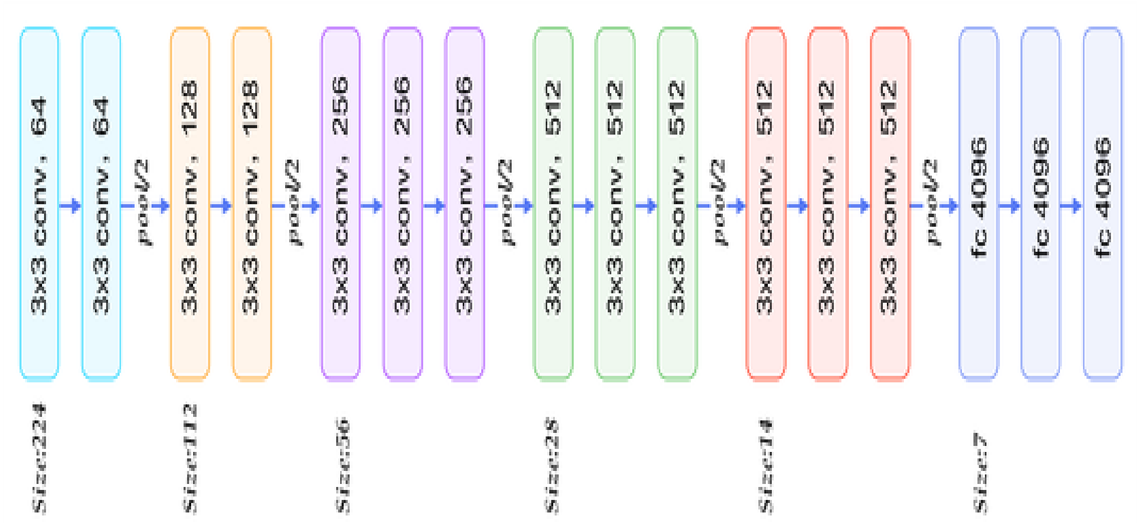


Figure 3.20: VGG-16 architecture

The VGG16 model can achieve a test accuracy of 92.7% in ImageNet.

Chapter 4

Realization :

During the training phase of our project, we encountered a significant challenge. We realized that our models had a large number of parameters, which required substantial computational power for training. Unfortunately, we did not have access to sufficient computing resources on our **local machine**, and attempts to train the models on platforms like **Kaggle** and **Google Colab** were unsuccessful due to out-of-memory issues.

To overcome this limitation, we decided to upgrade to **Google Colab Pro**, which offered improved performance and higher computational capabilities. With access to more powerful hardware, we were finally able to successfully train several versions of our models.

The transition to Google Colab Pro proved to be a crucial step, as it provided us with the necessary resources to tackle the demanding training process. By utilizing the enhanced computing capabilities, we could effectively train our models and extract valuable insights from the data.

This experience highlighted the importance of having access to adequate computational power, especially when dealing with models that have a large number of parameters. The availability of more robust resources enabled us to overcome the training challenges we initially faced and move forward with our project.

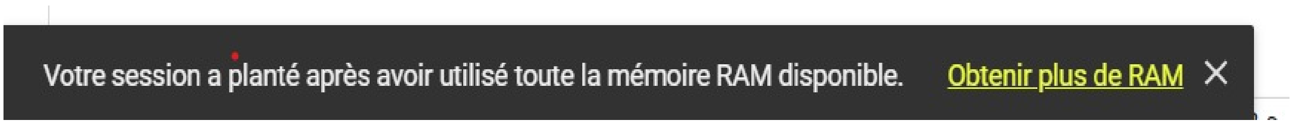
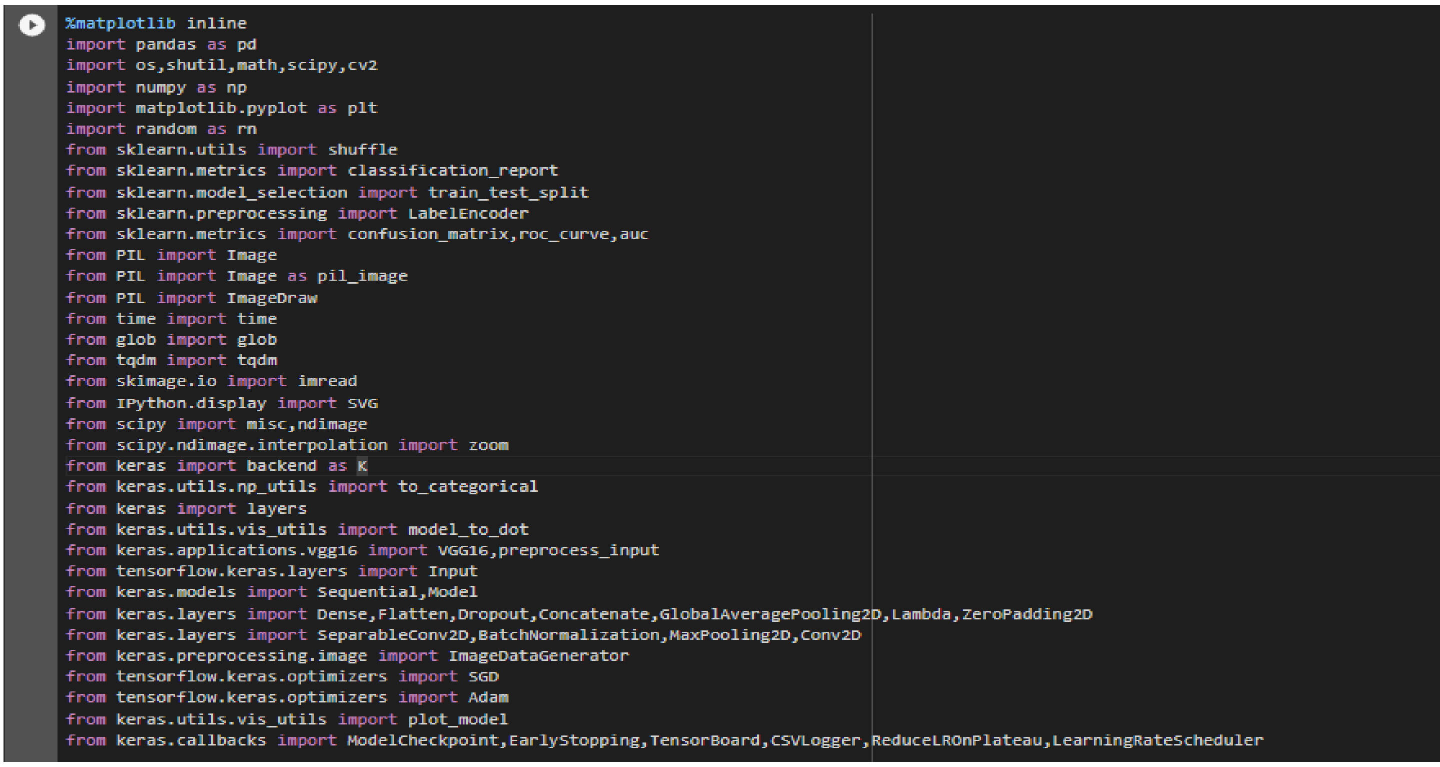

Votre session a planté après avoir utilisé toute la mémoire RAM disponible.

Figure 4.1: Out of memory

1 Importing Libraries :

In terms of the code structure, this step is typically located at the beginning of the script or notebook. It is a crucial step to include all the necessary dependencies and ensure that the project has access to the required functionalities provided by these libraries.

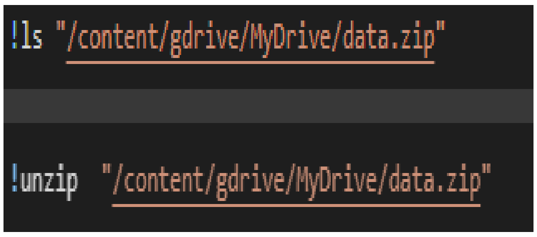


```
%matplotlib inline
import pandas as pd
import os,shutil,math,scipy,cv2
import numpy as np
import matplotlib.pyplot as plt
import random as rn
from sklearn.utils import shuffle
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix,roc_curve,auc
from PIL import Image
from PIL import Image as pil_image
from PIL import ImageDraw
from time import time
from glob import glob
from tqdm import tqdm
from skimage.io import imread
from IPython.display import SVG
from scipy import misc,ndimage
from scipy.ndimage.interpolation import zoom
from keras import backend as K
from keras.utils.np_utils import to_categorical
from keras import layers
from keras.utils.vis_utils import model_to_dot
from keras.applications.vgg16 import VGG16,preprocess_input
from tensorflow.keras.layers import Input
from keras.models import Sequential,Model
from keras.layers import Dense,Flatten,Dropout,Concatenate,GlobalAveragePooling2D,Lambda,ZeroPadding2D
from keras.layers import SeparableConv2D,BatchNormalization,MaxPooling2D,Conv2D
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from keras.utils.vis_utils import plot_model
from keras.callbacks import ModelCheckpoint,EarlyStopping,TensorBoard,CSVLogger,ReduceLROnPlateau,LearningRateScheduler
```

Figure 4.2: The importation of necessary libraries

2 Data labeling :

Importing and unzipping the data from Google Drive :

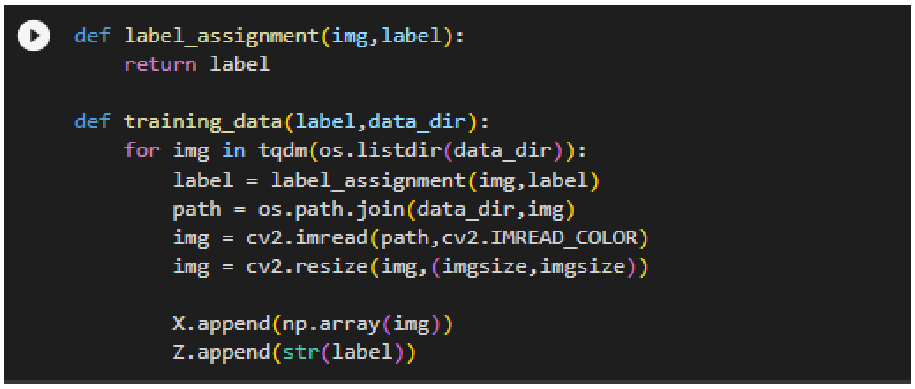


```
!ls "/content/gdrive/MyDrive/data.zip"

!unzip  "/content/gdrive/MyDrive/data.zip"
```

Figure 4.3: Data unzipping

Once the dataset is available in our Google Drive, we can proceed with the importation process. After importing the dataset, we need to unzip it if the data is compressed in a zip file format. Unzipping the data is necessary to access the individual files or folders contained within the dataset.

Data labeling :



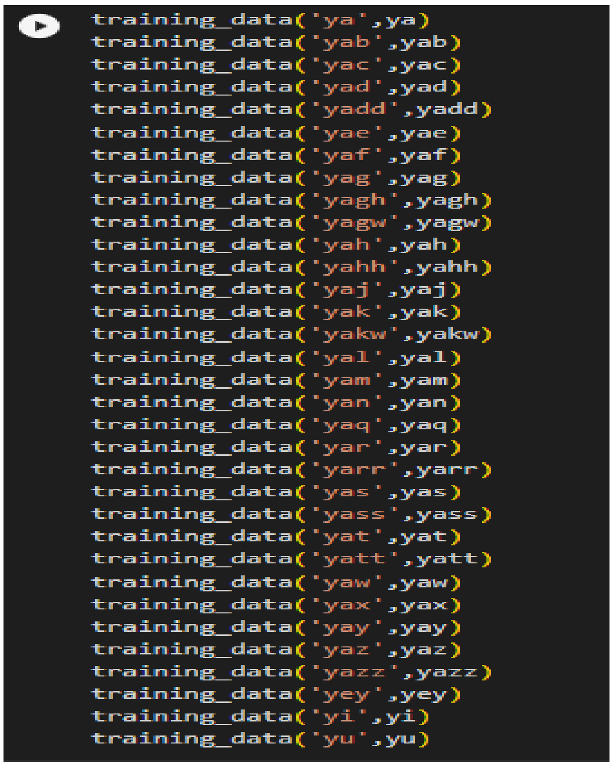
```
def label_assignment(img,label):
    return label

def training_data(label,data_dir):
    for img in tqdm(os.listdir(data_dir)):
        label = label_assignment(img,label)
        path = os.path.join(data_dir,img)
        img = cv2.imread(path,cv2.IMREAD_COLOR)
        img = cv2.resize(img,(imgsize,imgsize))

        X.append(np.array(img))
        Z.append(str(label))
```

Figure 4.4: Labeling fonctions

The X list contains the images as NumPy arrays. The Z list contains the labels as strings.

The label-assignment() function takes an image and a label as input and returns the label.

The training-data() function is more complex. It first iterates over all the images in the data directory. For each image, it reads the image into memory, resizes it to a fixed size, and then appends the image to the X list. It also appends the label to the Z list.

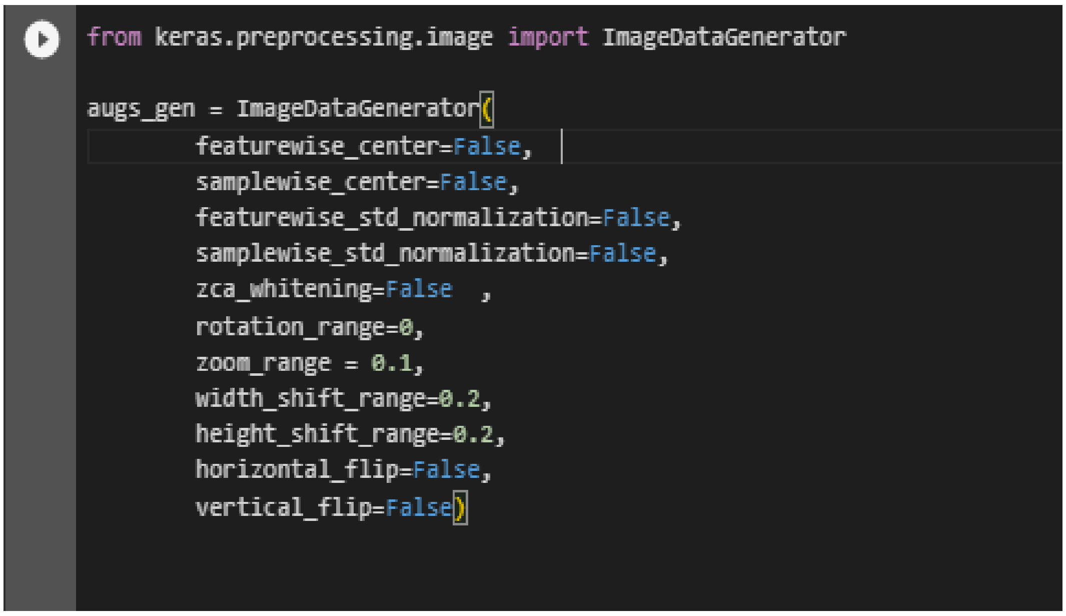


```
training_data('ya',ya)
training_data('yab',yab)
training_data('yac',yac)
training_data('yad',yad)
training_data('yadd',yadd)
training_data('yae',yae)
training_data('yaf',yaf)
training_data('yag',yag)
training_data('yagh',yagh)
training_data('yagw',yagw)
training_data('yah',yah)
training_data('yahh',yahh)
training_data('yaj',yaj)
training_data('yak',yak)
training_data('yakw',yakw)
training_data('yal',yal)
training_data('yam',yam)
training_data('yan',yan)
training_data('yaq',yaq)
training_data('yar',yar)
training_data('yarr',yarr)
training_data('yas',yas)
training_data('yass',yass)
training_data('yat',yat)
training_data('yatt',yatt)
training_data('yaw',yaw)
training_data('yax',yax)
training_data('yay',yay)
training_data('yaz',yaz)
training_data('yazz',yazz)
training_data('yey',yey)
training_data('yi',yi)
training_data('yu',yu)
```

Figure 4.5: Data labeling

3 Data augmentation :

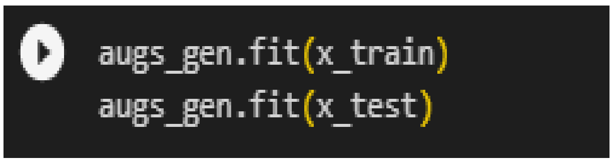
Data augmentation is a commonly used technique in convolutional neural networks (CNNs) for image classification tasks. It involves applying various transformations to the training images, thereby creating new augmented versions of the data. This technique helps to increase the diversity and size of the training dataset, which can improve the model's ability to generalize and enhance its performance.



```
from keras.preprocessing.image import ImageDataGenerator

augs_gen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False  ,
        rotation_range=0,
        zoom_range = 0.1,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=False,
        vertical_flip=False)
```

Figure 4.6: Augmentation settings

By applying these transformations to the training images, the CNN model can learn from a larger and more diverse set of data, which can help improve its ability to classify and recognize objects in new and unseen images. Data augmentation is an effective technique to prevent overfitting and enhance the generalization capability of CNN models.

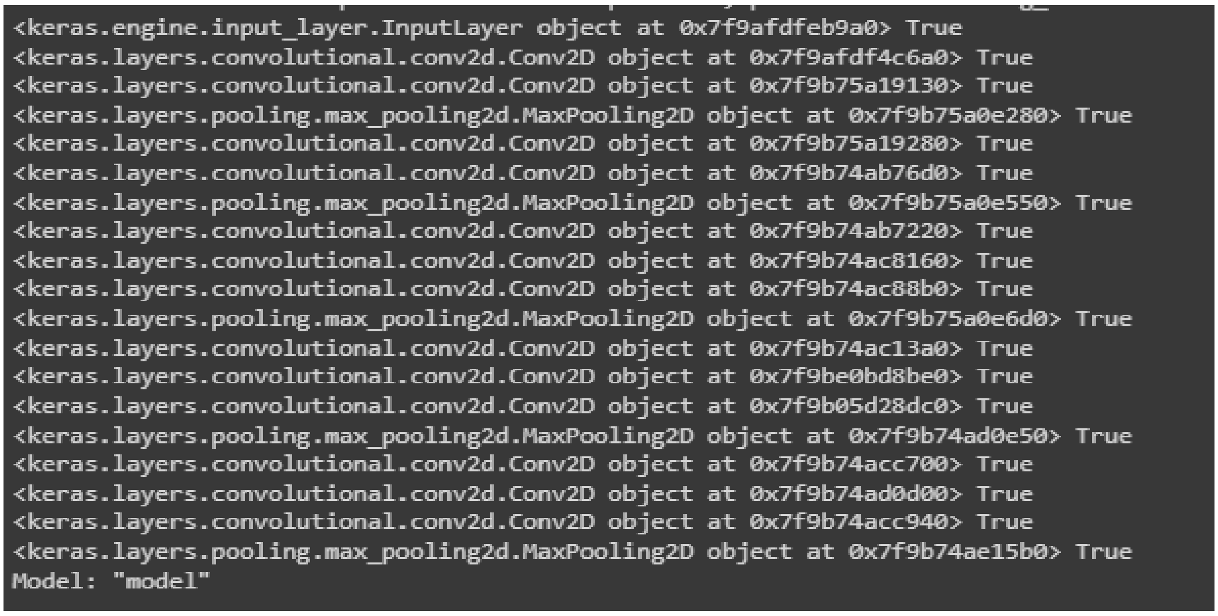


```
augs_gen.fit(x_train)
augs_gen.fit(x_test)
```

Figure 4.7: Data augmentation

4 VGG16 Transfer Learning with Fine-tuning for Image Classification :

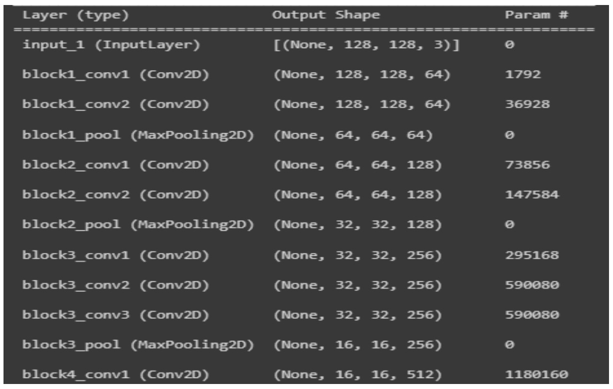
The implementation of transfer learning using the VGG16 architecture. The pre-trained VGG16 model, trained on the ImageNet dataset, is used as the base model. By freezing the initial layers of the base model, we can leverage its learned feature extraction capabilities. The subsequent layers are then added to the model for further customization and fine-tuning. This approach allows for efficient training of an image classification model on a smaller dataset by leveraging the pre-trained weights and knowledge gained from the ImageNet dataset.



```
<keras.engine.input_layer.InputLayer object at 0x7f9afdfeb9a0> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9afdf4c6a0> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b75a19130> True
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f9b75a0e280> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b75a19280> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b74ab76d0> True
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f9b75a0e550> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b74ab7220> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b74ac8160> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b74ac88b0> True
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f9b75a0e6d0> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b74ac13a0> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9be0bd8be0> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b05d28dc0> True
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f9b74ad0e50> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b74acc700> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b74ad0d00> True
<keras.layers.convolutional.conv2d.Conv2D object at 0x7f9b74acc940> True
<keras.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7f9b74ae15b0> True
Model: "model"
```

Figure 4.8: Trainable and Non-Trainable Parameters in VGG16 Transfer Learning

The given list displays the layers in the VGG16 model along with their corresponding trainability status. Each layer is represented by an object, such as InputLayer, Conv2D, MaxPooling2D, etc. The status "True" indicates that the layer is trainable, meaning its weights and biases can be updated during the training process. This information is crucial for understanding the configuration and training potential of the VGG16 model.



```
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160
```

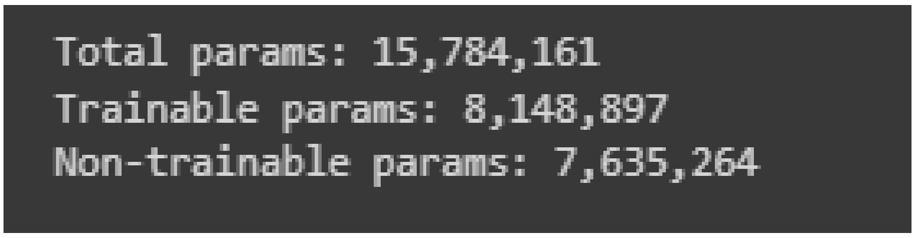
Figure 4.9: VGG16 Model Layer Configuration and Output Shapes

the total number of parameters in the model is reported as 15,784,161. Parameters refer to the weights and biases that are learned during the training process. These parameters are what allow the neural network to make predictions on new data.

Out of the total parameters, 8,148,897 are trainable. These are the parameters that will be updated and optimized during the training process. The trainable parameters include the weights and biases of the additional layers that were added on top of the frozen base model.

On the other hand, 7,635,264 parameters are non-trainable. These parameters belong to the frozen layers of the base model, meaning their weights will not be updated during training. These non-trainable parameters are already pre-trained and have been optimized on a large dataset (such as ImageNet) to capture general features.

By freezing a portion of the base model's layers, we can reduce the number of trainable parameters, which can help prevent overfitting and make the training process more efficient. This division of trainable and non-trainable parameters allows us to leverage the pre-trained knowledge while adapting the model to our specific task.



```
Total params: 15,784,161
Trainable params: 8,148,897
Non-trainable params: 7,635,264
```

Figure 4.10

5 Optimizers:

Optimizers are algorithms used in deep learning models to update the parameters (weights and biases) of the model during the training process. They optimize the learning process by minimizing the loss function and improving the model's accuracy.

In our case, we used some popular optimizers :

- Stochastic Gradient Descent (SGD): This is a widely used optimizer that updates the parameters based on the gradients of the loss function with respect to the parameters. SGD updates the parameters in the opposite direction of the gradient to minimize the loss.

- Adam: Adam stands for Adaptive Moment Estimation. It combines the advantages of two other optimizers, AdaGrad and RMSProp. Adam adapts the learning rate for each parameter based on the estimates of the first and second moments of the gradients.

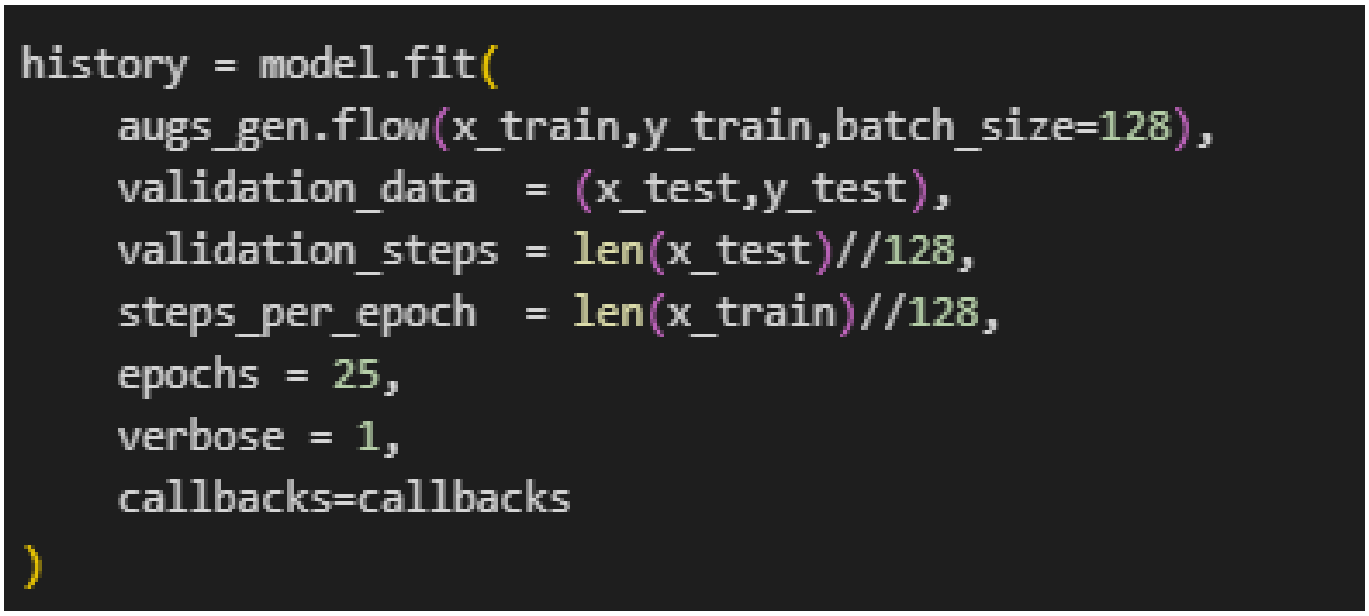
6 Model training :

the training process of a deep learning model with data augmentation. Data augmentation is a technique used to increase the diversity and size of the training dataset by applying various transformations to the original data samples.

During training, the model is exposed to augmented versions of the training data, which helps enhance its ability to generalize and improve performance. The augmented data is generated in batches using techniques such as rotation, translation, scaling, flipping, and more.

The validation data is used to evaluate the model's performance during training, providing insights into its generalization capabilities. The training process involves iterating over multiple epochs, where each epoch represents a complete pass through the training dataset. The model's parameters are adjusted using optimization algorithms to minimize the loss function and improve accuracy.

By leveraging data augmentation, the model can learn robust features and adapt to variations and distortions commonly encountered in real-world data. This approach enhances the model's ability to handle diverse inputs and achieve better performance on unseen data.



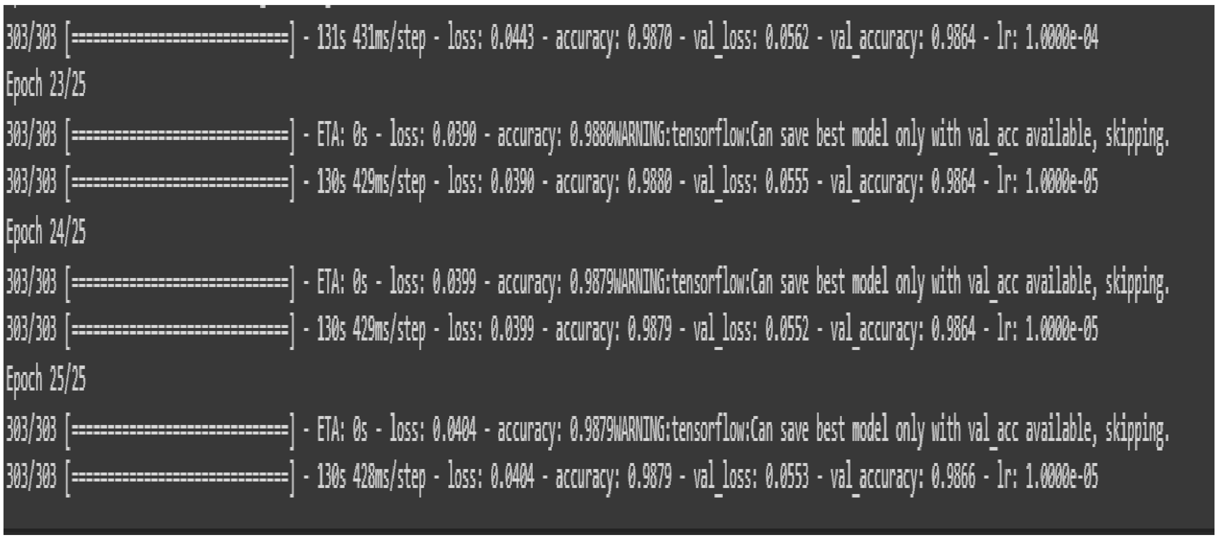
```
history = model.fit(
    augs_gen.flow(x_train,y_train,batch_size=128),
    validation_data  = (x_test,y_test),
    validation_steps = len(x_test)//128,
    steps_per_epoch  = len(x_train)//128,
    epochs = 25,
    verbose = 1,
    callbacks=callbacks
)
```

Figure 4.11: Training the Model with Data Augmentation

During an epoch, the model goes through multiple iterations, typically in mini-batches, where a subset of the training data is used to update the model's parameters. The number of iterations or batches per epoch depends on factors such as the size of the training dataset and the batch size chosen for training.

The purpose of performing multiple epochs is to allow the model to learn from the data progressively. With each epoch, the model gains more knowledge and adjusts its internal parameters to minimize the training loss and improve its ability to make accurate predictions.

The number of epochs is a hyperparameter that needs to be tuned during the training process. It should be chosen carefully to balance between underfitting (insufficient learning) and overfitting (excessive learning).



```
303/303 [==============================] - 131s 431ms/step - loss: 0.0443 - accuracy: 0.9870 - val_loss: 0.0562 - val_accuracy: 0.9864 - lr: 1.0000e-04
Epoch 23/25
303/303 [==============================] - ETA: 0s - loss: 0.0390 - accuracy: 0.9880WARNING:tensorflow:Can save best model only with val_acc available, skipping.
303/303 [==============================] - 130s 429ms/step - loss: 0.0390 - accuracy: 0.9880 - val_loss: 0.0555 - val_accuracy: 0.9864 - lr: 1.0000e-05
Epoch 24/25
303/303 [==============================] - ETA: 0s - loss: 0.0399 - accuracy: 0.9879WARNING:tensorflow:Can save best model only with val_acc available, skipping.
303/303 [==============================] - 130s 429ms/step - loss: 0.0399 - accuracy: 0.9879 - val_loss: 0.0552 - val_accuracy: 0.9864 - lr: 1.0000e-05
Epoch 25/25
303/303 [==============================] - ETA: 0s - loss: 0.0404 - accuracy: 0.9879WARNING:tensorflow:Can save best model only with val_acc available, skipping.
303/303 [==============================] - 130s 428ms/step - loss: 0.0404 - accuracy: 0.9879 - val_loss: 0.0553 - val_accuracy: 0.9866 - lr: 1.0000e-05
```

Figure 4.12: Epochs: Training Iterations for Deep Learning

7 Evaluation and Analysis of Results :

Evaluation and analysis of results involve assessing the performance of our deep learning model and extracting meaningful insights from the obtained outcomes. It begins by defining appropriate evaluation metrics that align with our project goals. Then, a separate test set is used to evaluate the model's performance, calculating metrics such as accuracy, precision, recall, and F1 score to measure different aspects of its effectiveness. The results are interpreted to identify strengths, weaknesses, and potential areas for improvement. Error analysis, such as studying the confusion matrix, provides deeper insights into specific errors made by the model. This iterative process allows you to fine-tune our model, adjust hyperparameters, and compare its performance against baseline models or existing approaches. Documentation of the evaluation process ensures the rigor and reproducibility of our findings.

7.1 Evaluation Metrics:

7.1.1 Accuracy:

Accuracy measures the overall correctness of predictions made by a model. It is calculated as the ratio of correctly predicted samples to the total number of samples. However, accuracy can be misleading when dealing with imbalanced datasets, where the class distribution is skewed. It provides a general overview of the model's performance but may not reveal specific errors or biases.

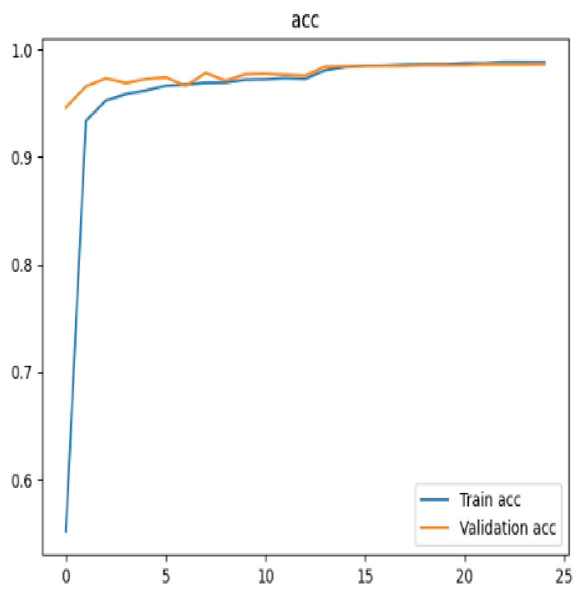


Figure 4.13: Accuracy function

7.1.2 Loss:

Loss in deep learning refers to a numerical value that measures the discrepancy between the predicted output and the true target value. It plays a crucial role in optimizing the model during training by quantifying the error. Different types of loss functions are used depending on the problem. Monitoring the loss during training helps assess the model's progress, aiming for lower loss values that indicate better alignment between predictions and targets. Evaluation metrics, not the loss, are used for assessing the model's performance on unseen data.

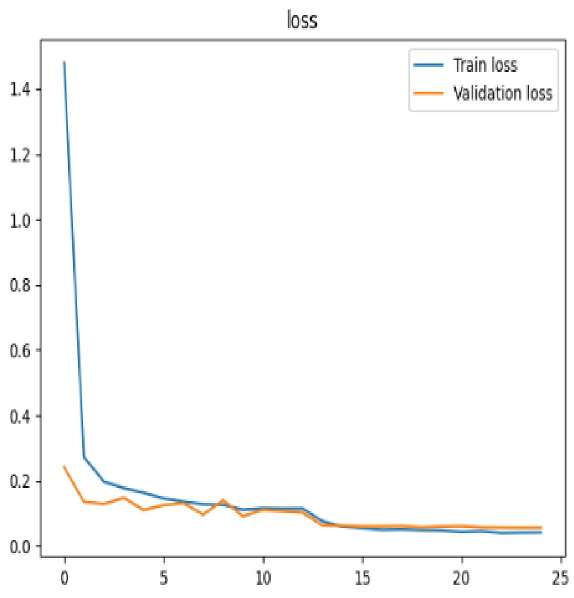


Figure 4.14: Loss function

7.1.3 Precision:

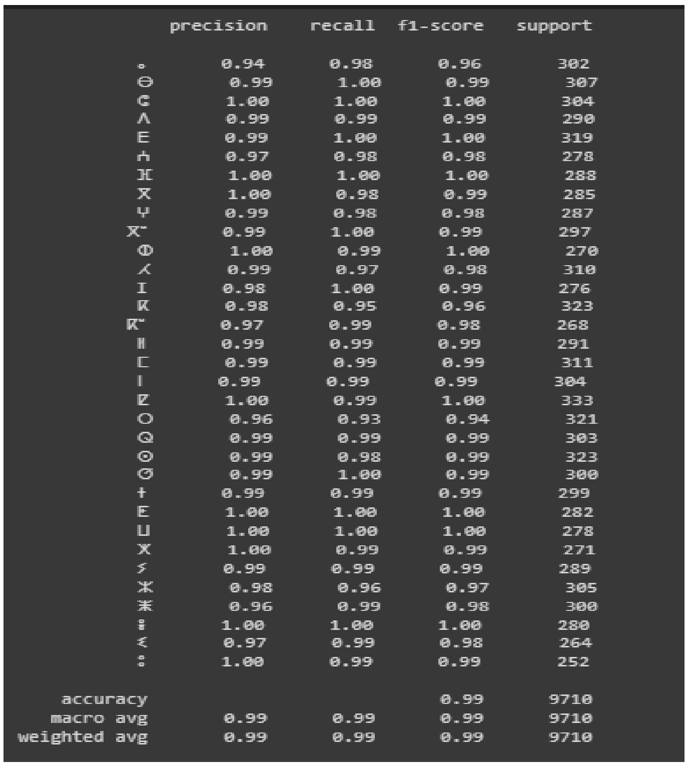
Precision quantifies the proportion of correctly predicted positive samples out of all predicted positive samples. It focuses on the accuracy of positive predictions, making it valuable when the cost of false positives is high. A high precision indicates that the model is cautious in making positive predictions and is less prone to false positives. However, it does not consider false negatives.

7.1.4 Recall:

Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive samples out of all actual positive samples. It evaluates the model's ability to identify all positive instances, making it important in scenarios where missing positive samples is costly. A high recall indicates that the model can effectively detect positive instances, but it may result in more false positives.

7.1.5 F1-Score:

The F1 score combines precision and recall into a single metric. It is the harmonic mean of precision and recall, providing a balanced measure of a model's performance. The F1 score is useful when there is an uneven class distribution, as it considers both false positives and false negatives. A higher F1 score indicates a better balance between precision and recall.



```
              precision    recall  f1-score   support

           ⴰ       0.94      0.98      0.96       302
           ⴱ       0.99      1.00      0.99       307
           ⵛ       1.00      1.00      1.00       304
           ⴷ       0.99      0.99      0.99       290
           ⴹ       0.99      1.00      1.00       319
           ⵄ       0.97      0.98      0.98       278
           ⴼ       1.00      1.00      1.00       288
           ⴳ       1.00      0.98      0.99       285
           ⵖ       0.99      0.98      0.98       287
          ⴳⵯ       0.99      1.00      0.99       297
           ⵀ       1.00      0.99      1.00       270
           ⵃ       0.99      0.97      0.98       310
           ⵊ       0.98      1.00      0.99       276
           ⴽ       0.98      0.95      0.96       323
          ⴽⵯ       0.97      0.99      0.98       268
           ⵍ       0.99      0.99      0.99       291
           ⵎ       0.99      0.99      0.99       311
           ⵏ       0.99      0.99      0.99       304
           ⵇ       1.00      0.99      1.00       333
           ⵔ       0.96      0.93      0.94       321
           ⵕ       0.99      0.99      0.99       303
           ⵙ       0.99      0.98      0.99       323
           ⵚ       0.99      1.00      0.99       300
           ⵜ       0.99      0.99      0.99       299
           ⵟ       1.00      1.00      1.00       282
           ⵡ       1.00      1.00      1.00       278
           ⵅ       1.00      0.99      0.99       271
           ⵢ       0.99      0.99      0.99       289
           ⵣ       0.98      0.96      0.97       305
           ⵥ       0.96      0.99      0.98       300
           ⴻ       1.00      1.00      1.00       280
           ⵉ       0.97      0.99      0.98       264
           ⵓ       1.00      0.99      0.99       252

    accuracy                           0.99      9710
   macro avg       0.99      0.99      0.99      9710
weighted avg       0.99      0.99      0.99      9710
```

Figure 4.15: Precision , Recall , F1 Score

7.1.6 Confusion matrix:

A confusion matrix is a table that summarizes the performance of a classification model. It displays the counts of true positive, true negative, false positive, and false negative predictions. The confusion matrix provides an in-depth understanding of the model's performance for each class, enabling analysis of specific errors. It helps identify potential biases, class imbalances, or patterns in the model's predictions.

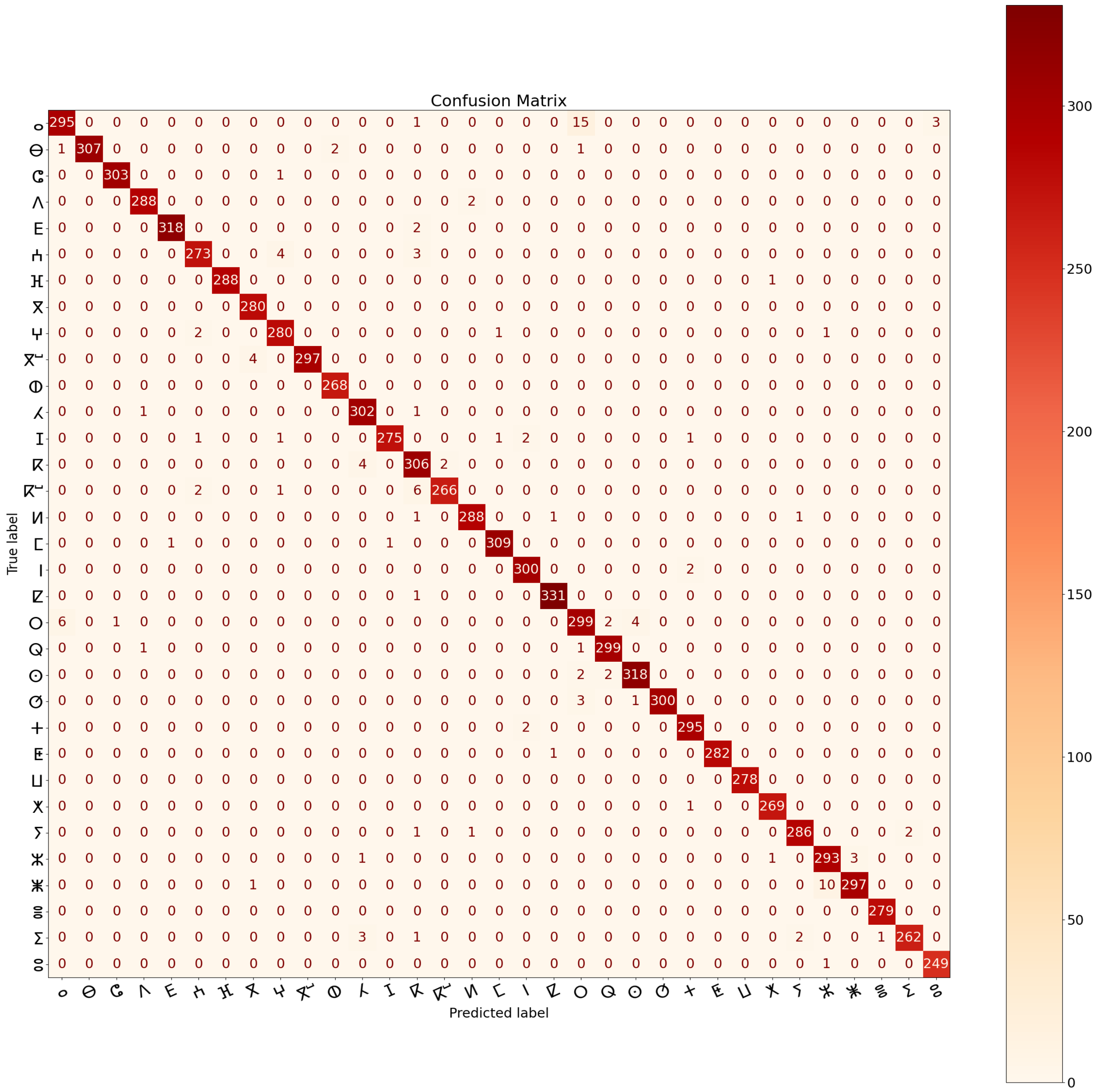
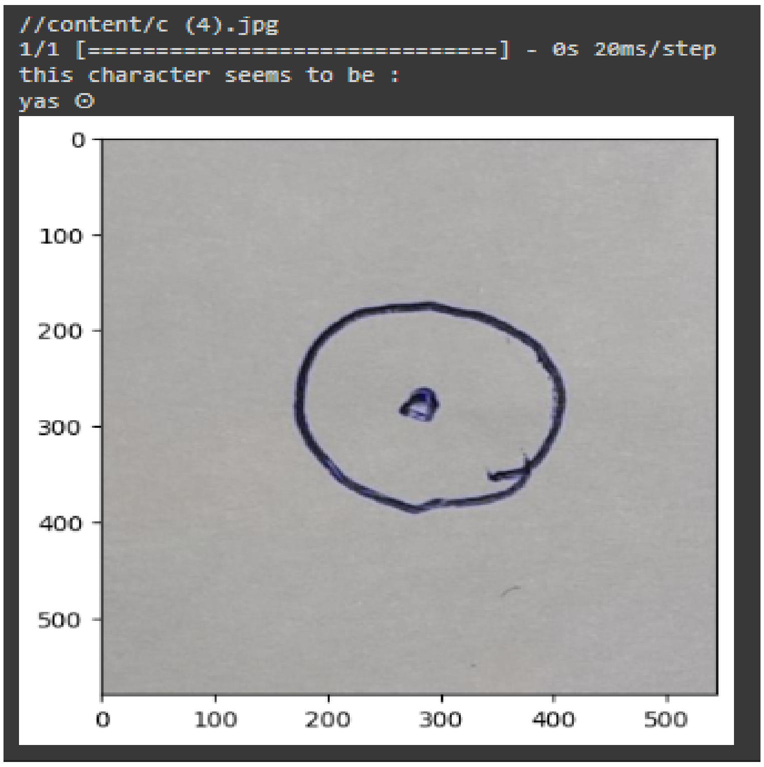


Figure 4.16: Confusion matrix

7.2 Test Set Evaluation :

Test set evaluation is a crucial step in deep learning where a separate dataset, unseen during model training, is used to assess the model's performance. It involves feeding the test set through the trained model to obtain predictions or output values. By comparing these predictions with the true labels or target values.

Test set evaluation provides an unbiased assessment of how well the model generalizes to new, unseen data, allowing you to gauge its real-world performance and make informed decisions about its deployment or further improvements.

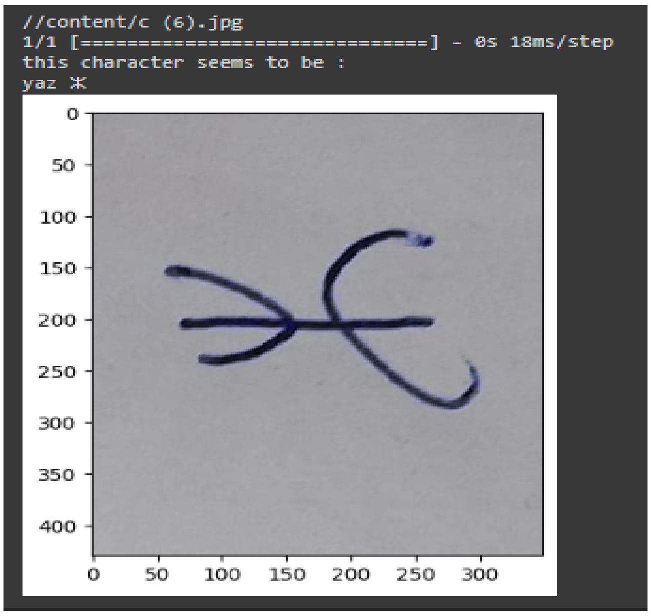


```
//content/c (4).jpg
1/1 [==============================] - 0s 20ms/step
this character seems to be :
yas ⵙ
```

Figure 4.17: Test

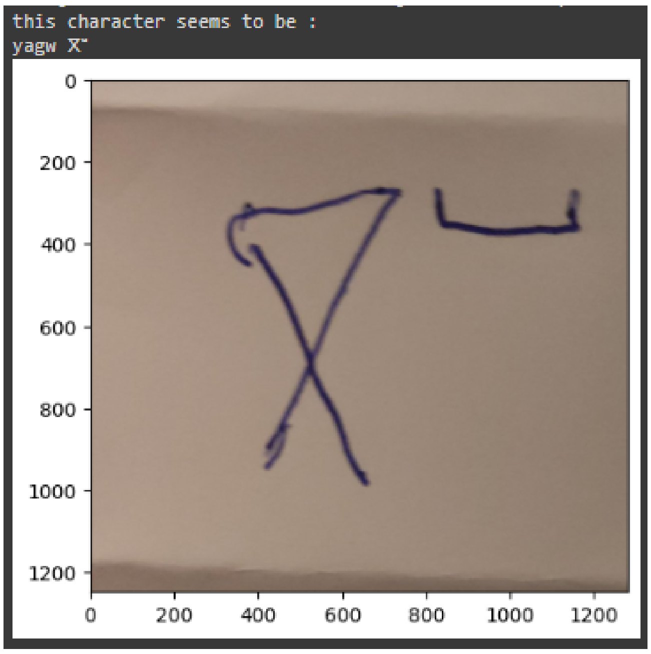
To evaluate our model, we utilized Tifinagh characters by incorporating them into our testing process. These Tifinagh characters were not part of the training data, ensuring an unbiased evaluation of the model's performance. By feeding these Tifinagh characters through the trained model, we obtained predictions or output values. Comparing these predictions with the true labels or expected Tifinagh characters.

This evaluation process enables us to measure the model's proficiency specifically in handling Tifinagh characters, providing insights into its performance and informing potential areas for enhancement.This evaluation process enables us to measure the model's proficiency specifically in handling Tifinagh characters, providing insights into its performance and informing potential areas for enhancement.



```
//content/c (6).jpg
1/1 [==============================] - 0s 18ms/step
this character seems to be :
yaz ⵣ
```

Figure 4.18: Test



```
this character seems to be :
yagw ⴳⵯ
```

Figure 4.19: Test

Examples of test sets with different Tifinagh character, collected from different individuals, characters with varying levels of complexity or distortion.

7.3 Interpret Results:

Based on the provided results, it appears that our deep learning model for predicting Tifinagh handwritten characters is performing exceptionally well. Here is an interpretation of the various evaluation metrics:

- Loss: The reported training loss of 0.0404 indicates the average discrepancy between the predicted outputs and the true target values during the training process. Lower values indicate better alignment between predictions and targets, suggesting that our model was able to minimize the error effectively during training.

- Accuracy: The training accuracy of 0.9879 suggests that, on average, our model correctly classified Tifinagh characters with approximately 98.79% accuracy during the training phase. This indicates a high level of performance and proficiency in recognizing Tifinagh characters.

- Precision: The precision scores for most classes are very high, ranging from 0.94 to 1.00. This indicates that the model has a high accuracy in correctly predicting the specific Tifinagh characters.

- Recall: The recall scores are also consistently high, ranging from 0.95 to 1.00. This means that the model effectively captures a high proportion of the true positive instances for each Tifinagh character.

- F1-score: The F1-scores, which are a harmonic mean of precision and recall, are also excellent across most classes, ranging from 0.94 to 1.00. This demonstrates a good balance between precision and recall, indicating the model's overall effectiveness in classification.

From these results, it is evident that our model is capable of accurately recognizing and classifying Tifinagh handwritten characters. The high precision, recall, and F1-scores across most classes indicate the model's robustness and ability to generalize well to unseen data, those results indicate that our model performed exceptionally well during training, achieving high accuracy and minimizing the loss. Overall, our model's performance indicates a high level of proficiency in recognizing Tifinagh characters, and it can be deemed successful in its intended task.

8 Conclusion :

In this chapter, we embarked on the realization of our deep learning project for Tifinagh character recognition. We began by importing the necessary libraries to support our implementation. Next, we tackled the critical task of data labeling, ensuring that each sample in our dataset was appropriately labeled with its corresponding Tifinagh character.

To enhance the diversity and robustness of our dataset, we employed data augmentation techniques. This involved applying transformations such as rotations, translations, and noise addition to expand the variations within the dataset, ultimately improving the model's ability to generalize.

For the core of our model, we employed the VGG16 architecture and utilized transfer learning with fine-tuning for image classification. This approach allowed us to leverage the knowledge learned from a pre-trained VGG16 model, adapt it to the Tifinagh character recognition task, and fine-tune it on our dataset.

To optimize our model's performance, we explored different optimizers and selected the most suitable one for our task. This ensured that the model's weights were adjusted effectively during the training process, leading to improved accuracy and convergence.

Through rigorous model training, our deep learning model learned to recognize and classify Tifinagh characters. We carefully monitored the loss and accuracy metrics during training to assess the model's progress and make necessary adjustments.

The evaluation and analysis of results allowed us to assess the performance of our model. We employed various evaluation metrics, including accuracy, loss, precision, recall, F1-score, and the confusion matrix. These metrics provided insights into the model's ability to correctly classify Tifinagh characters and its overall performance.

Furthermore, we evaluated the model on a separate test set to ensure unbiased assessment. This step allowed us to gauge the model's generalization capability and assess its performance on unseen data.

Finally, we interpreted the results to gain a deeper understanding of our model's performance. With an accuracy of approximately 99% and high precision, recall, and F1-scores across different Tifinagh characters, our model demonstrates exceptional effectiveness in recognizing handwritten Tifinagh characters.

In conclusion, through careful implementation, data preprocessing, model training, and evaluation, we have developed a robust deep learning model for Tifinagh character recognition. The model's high accuracy and comprehensive evaluation metrics validate its effectiveness and suitability for this task. This accomplishment paves the way for future applications of Tifinagh character recognition in various domains, such as language processing, document digitization, and cultural preservation.

Chapter 5

Fragmentation:

In this chapter, our goal is to tackle the task of word segmentation into individual characters. By combining the segmentation technique with our trained model, we aim to accurately segment words into characters and leverage the predictive capabilities of the model to decipher the word itself. This chapter serves as a critical step in the overall process of Tifinagh character recognition, enabling us to tackle the complex task of understanding and processing handwritten or scanned Tifinagh words at a fine-grained level.

1 The concept of fragmentation :

Word fragmentation refers to the process of breaking down a word into its constituent characters. In written text, words are composed of individual characters that together form meaningful units of language. By fragmenting words into characters, we can analyze and process text at a more granular level.

Word fragmentation to characters is a fundamental step in various natural language processing tasks, including text recognition, machine translation, sentiment analysis, and language modeling. It allows for fine-grained analysis and manipulation of individual characters, enabling accurate processing and understanding of textual data.

This process involves segmenting a word into its constituent characters based on specific criteria such as spacing, pixel connectivity, stroke width, or linguistic rules. Techniques like contour detection, connected component analysis, optical character recognition (OCR), and machine learning algorithms can be employed to perform word fragmentation into characters.

By fragmenting words into characters, we gain access to individual linguistic units, which can be further analyzed, classified, or transformed. This enables a deeper understanding of the underlying language and facilitates the development of advanced NLP applications that rely on character-level analysis and processing.

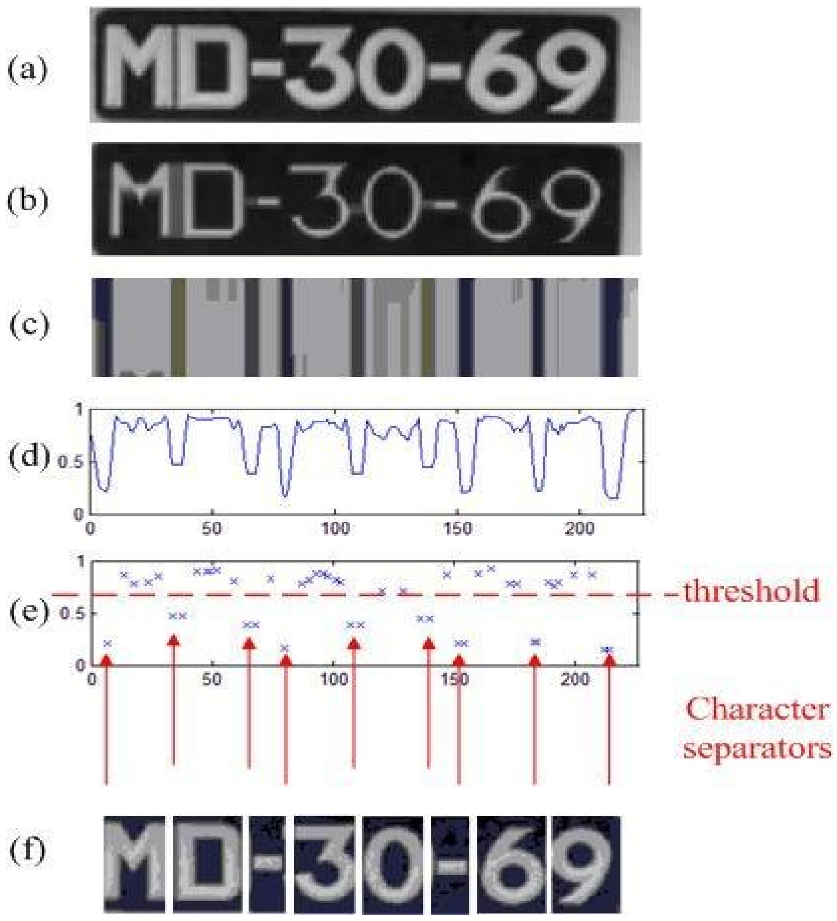


Figure 5.1: Example of fragmentation

2 OpenCv and segmentation:

OpenCV (Open Source Computer Vision Library) is a widely used open-source library for computer vision tasks, including image segmentation. Image segmentation refers to the process of dividing an image into distinct regions or objects. OpenCV provides various algorithms and techniques for segmentation, such as thresholding, contour detection, watershed segmentation, and graph-based segmentation. These methods help separate objects from the background, enabling further analysis and understanding of image content. OpenCV's segmentation capabilities make it a valuable tool for applications like object recognition, image processing, medical imaging, and video surveillance. Its ease of use and extensive documentation make it a popular choice for researchers and developers working in the field of computer vision.

Word segmentation into characters refers to the procedure of decomposing a word into its separate component characters.

OpenCV provides several techniques for word segmentation into characters, including:

1. Contour detection: OpenCV's contour detection algorithms can identify the contours of individual characters within a word. By analyzing the spatial relationships and connectivity of these contours, characters can be separated.

2. Connected component analysis: OpenCV can analyze the connected components within a word image to identify and extract individual characters. This technique takes advantage of the distinct regions of connected pixels representing characters.

3. Optical character recognition (OCR): OpenCV's OCR functionality can not only recognize characters but also segment a word into its constituent characters based on the spacing between characters.

4. Machine learning-based approaches: OpenCV can be combined with machine learning techniques, such as deep learning models or traditional classifiers, to train models for character segmentation. These models can learn to detect and separate characters based on training data.

5. Stroke width transform: OpenCV's stroke width transform algorithm can be applied to analyze the variations in stroke widths within a word image, aiding in character segmentation.

By utilizing these techniques provided by OpenCV, developers can accurately segment words into their individual characters, enabling further processing and analysis at the character level. This is crucial for tasks such as character recognition, text-to-speech synthesis, and language processing applications.

3 Understanding Contours: Shape Analysis and Object Detection in Image Processing:

Contours are curves that join continuous points along the boundary of an object with the same color or intensity. They are commonly used for shape analysis, object detection, and recognition. Contours can be found in images by using techniques such as thresholding or edge detection.

When working with contours in OpenCV, it is important to ensure that the object to be found is white and the background is black. The 'findContours()' function in OpenCV is used to find contours in a binary image. It takes the source image, contour retrieval mode, and contour approximation method as arguments, and returns the contours and hierarchy.

To draw contours, the 'drawContours()' function is used. It takes the source image, the contours (as a Python list), the contour index (or -1 for all contours), and additional parameters such as color and thickness. It can be used to draw all contours in an image or individual contours.

Contour approximation method is the third argument of the 'findContours()' function. It determines the level of detail stored in the contours. Using 'Contour Approximation Method' it removes redundant points and compresses the contour, saving memory. For example, a straight line can be represented by just two end points instead of storing all the points on the line.

In conclusion, contours are curves representing the boundaries of objects in an image. They are useful for various image processing tasks, and OpenCV provides functions to find and draw contours, as well as control the level of detail in the contours through contour approximation methods.

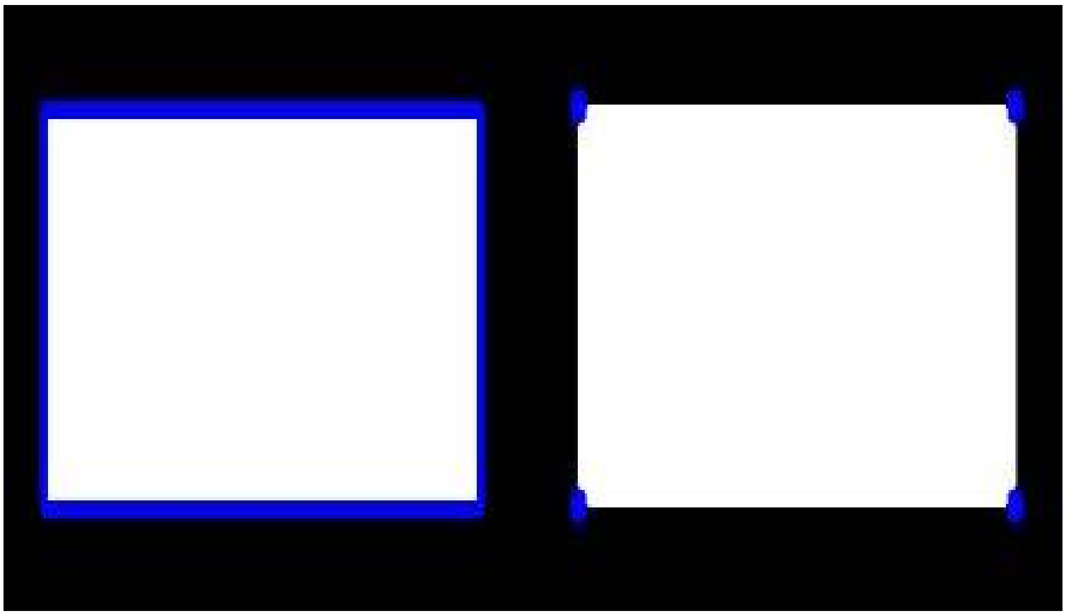


Figure 5.2: Type of contours

4 Tifinagh and segmentation:

In our study of fragmentation of Tifinagh words into characters, we aim to develop a comprehensive understanding of the causes and challenges associated with this process. By analyzing the characteristics and patterns of fragmented characters, we can gain valuable insights into the unique obstacles they pose for accurate recognition.

Through our research, we have identified contour detection as the most effective technique for addressing fragmentation in Tifinagh character recognition. Contour detection involves identifying the outlines or boundaries of individual characters within a fragmented word. This approach allows us to capture the distinct shape and structure of each character, even in cases where they appear broken or fragmented.

By utilizing contour detection, we can accurately segment Tifinagh words into their constituent characters, enabling more precise recognition and analysis. This technique leverages the contours and connectivity of characters to reconstruct their original shapes, mitigating the challenges posed by fragmentation.

Our findings highlight the importance of selecting appropriate techniques tailored to the specific characteristics of Tifinagh script. The application of contour detection in the fragmentation of Tifinagh words to characters demonstrates promising results and provides a solid foundation for further advancements in Tifinagh character recognition systems.

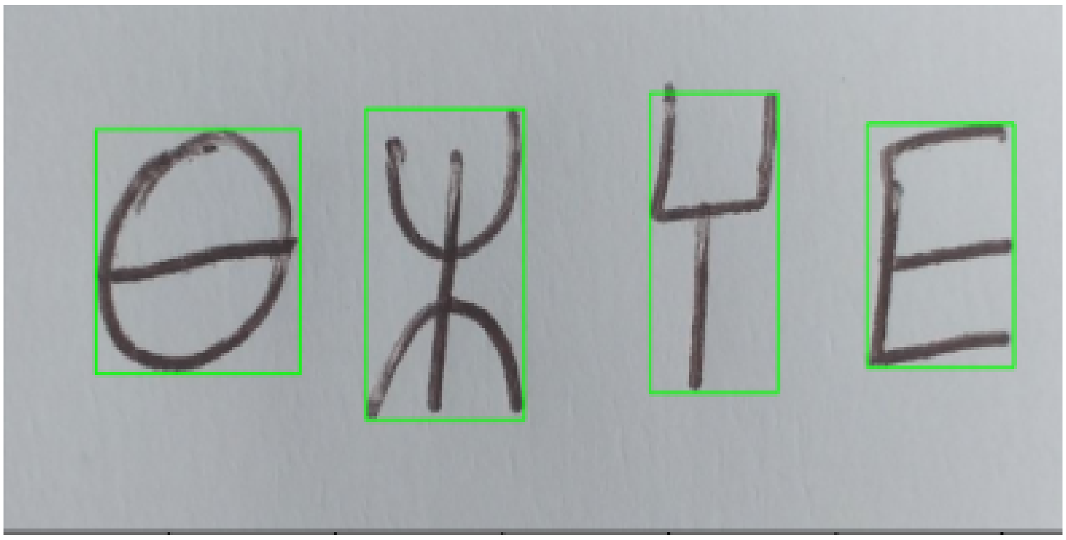
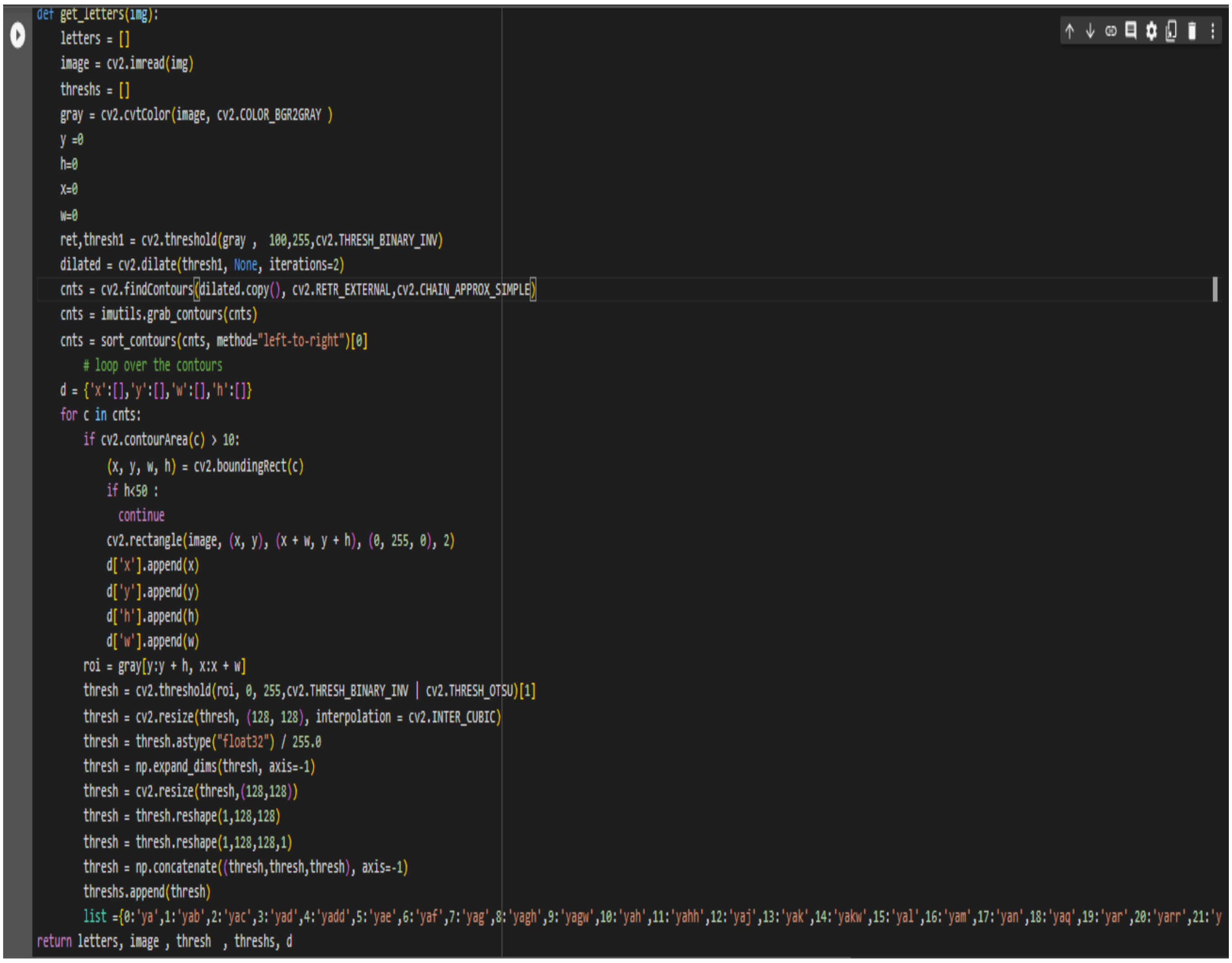


Figure 5.3: segmentation aim

5 Contour Analysis and Recognition:

5.1 Letter Extraction from Image:

The 'get-letters()' function utilizes contour analysis and recognition techniques to extract letters from an input image. The function takes an image as input and performs the following steps:



```python
def get_letters(img):
    letters = []
    image = cv2.imread(img)
    threshs = []
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY )
    y =0
    h=0
    x=0
    w=0
    ret,thresh1 = cv2.threshold(gray ,  100,255,cv2.THRESH_BINARY_INV)
    dilated = cv2.dilate(thresh1, None, iterations=2)
    cnts = cv2.findContours(dilated.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    cnts = sort_contours(cnts, method="left-to-right")[0]
        # loop over the contours
    d = {'x':[],'y':[],'w':[],'h':[]}
    for c in cnts:
        if cv2.contourArea(c) > 10:
            (x, y, w, h) = cv2.boundingRect(c)
            if h<50 :
              continue
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
            d['x'].append(x)
            d['y'].append(y)
            d['h'].append(h)
            d['w'].append(w)
        roi = gray[y:y + h, x:x + w]
        thresh = cv2.threshold(roi, 0, 255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
        thresh = cv2.resize(thresh, (128, 128), interpolation = cv2.INTER_CUBIC)
        thresh = thresh.astype("float32") / 255.0
        thresh = np.expand_dims(thresh, axis=-1)
        thresh = cv2.resize(thresh,(128,128))
        thresh = thresh.reshape(1,128,128)
        thresh = thresh.reshape(1,128,128,1)
        thresh = np.concatenate((thresh,thresh,thresh), axis=-1)
        threshs.append(thresh)
        list ={0:'ya',1:'yab',2:'yac',3:'yad',4:'yadd',5:'yae',6:'yaf',7:'yag',8:'yagh',9:'yagw',10:'yah',11:'yahh',12:'yaj',13:'yak',14:'yakw',15:'yal',16:'yam',17:'yan',18:'yaq',19:'yar',20:'yarr',21:'y
    return letters, image , thresh  , threshs, d
```

Figure 5.4: Letter Extraction from Images

- Convert the image to grayscale for further processing.

- Apply a threshold to create a binary image, emphasizing the letter regions using an inverted threshold.

- Dilate the thresholded image to enhance the letter contours.

- Find contours in the dilated image, sorting them from left to right.

- Iterate over the contours, filtering out small areas and extracting the bounding rectangle coordinates for each significant contour.

- Draw rectangles around the detected letters on the original image.

- Extract the region of interest (ROI) for each letter, resize it to a standard size, and perform further preprocessing.

- Prepare the ROI for recognition by converting it to floating-point format, normalizing the pixel values, and reshaping it appropriately.

- Store the preprocessed ROIs in a list for further analysis.

- Return the extracted letters, the original image with rectangles, the preprocessed ROIs, and the bounding box information.

This function provides a convenient way to extract and recognize letters from images, facilitating various applications such as optical character recognition , document processing, and text analysis.

5.2 Word Extraction and Recognition from Images:

The get-word() function extracts individual letters from an image and combines them to form a word. It takes a list of letter images as input and performs the following steps:

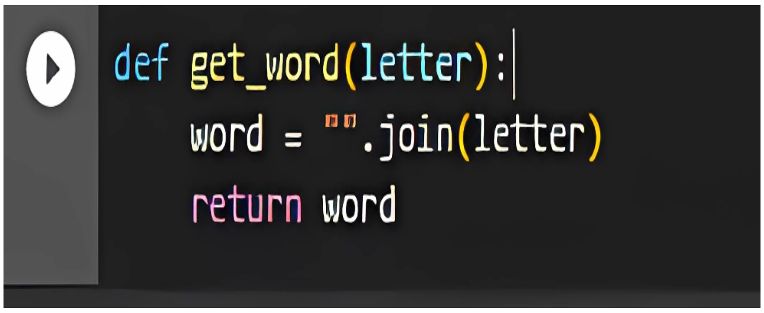
- Concatenation: The function joins the individual letter images together to create a single word by using the "".join() method.

- Word Formation: The combined string of letters is assigned to the word variable.

Result Retrieval: The function returns the resulting word.

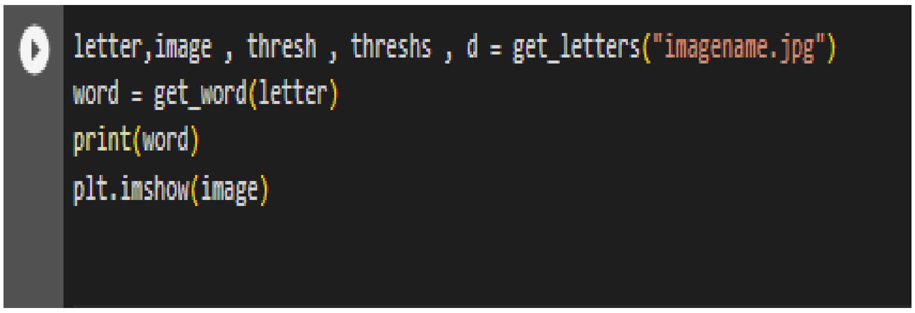
To extract the letters from an image, the script uses the get-letters() function, providing the path of the image file as input.

The extracted word is obtained by calling the get-word() function with the letter list as the input. This word is then printed, and the original image with bounding rectangles around the letters is displayed using plt.imshow().



```python
def get_word(letter):
    word = "".join(letter)
    return word
```

Figure 5.5: Word Formation from Letter Images



```python
letter,image , thresh , threshs , d = get_letters("imagename.jpg")
word = get_word(letter)
print(word)
plt.imshow(image)
```

Figure 5.6: Word Extraction and Visualization

5.3 Word Segmentation and Classification for Image Recognition:

The segmentation() function performs word segmentation and classification on an image specified by the PATH parameter. It takes the d dictionary, which contains the bounding box coordinates of the segmented letters within the image, as an additional input. Here is an overview of the process:

- Word Segmentation: The seg() function is called to segment the word based on the provided bounding box coordinates in the d dictionary.

- Image Loading: The image specified by the PATH parameter is loaded using a suitable image processing library.

- Word Extraction and Resizing: The segmented word regions are extracted from the loaded image based on the coordinates provided in dim. Each extracted word region is then resized to a desired size, specified as imgsize (128x128 pixels in this case).

- Image Classification: The pre-trained model is used to predict the class label (character) for each word region. The word regions are reshaped and passed through the model using the model.predict() function. The predicted class label with the highest confidence is determined by finding the index of the maximum value in the predicted probabilities.

- Word Reconstruction: The predicted class labels for each word region are accumulated to reconstruct the recognized word. The predicted labels are appended to the p list.

- Result Display: The reconstructed word is printed, indicating the most likely class labels for each segmented region within the image.

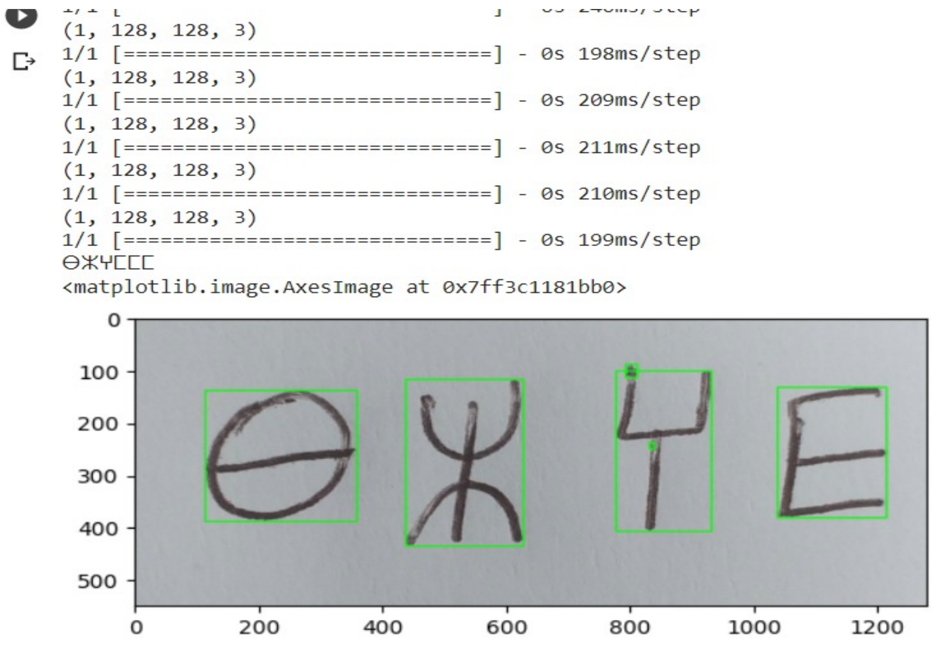
The segmentation() function combines word segmentation with image classification techniques to identify and recognize words from images.

6 Test:

Now, we are embarking on a comprehensive test to validate the effectiveness and functionality of the entire fragmentation concept. The purpose of this test is to ensure that all aspects of the fragmentation process are working flawlessly and producing the desired outcomes. By conducting this thorough evaluation, we aim to gain confidence in the reliability and accuracy of the fragmentation technique.

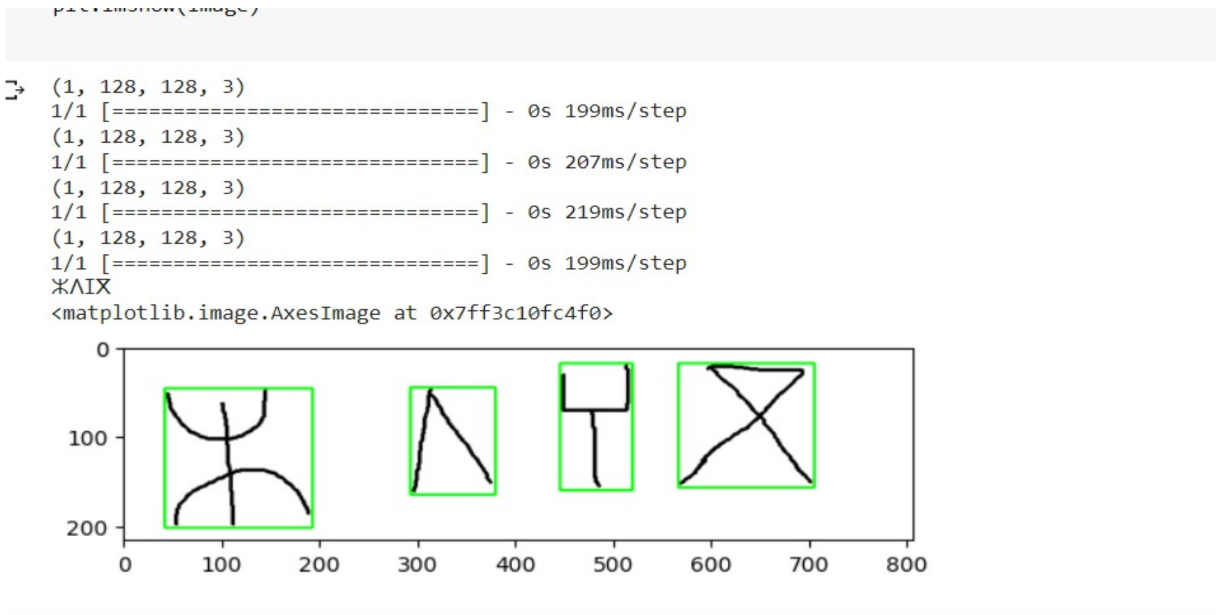
To execute this test, we have meticulously prepared a diverse set of sample images, covering different text styles, sizes, and layouts. This broad range of test cases will enable us to evaluate the fragmentation concept's robustness and adaptability to various scenarios. We will examine its performance in handling text of different fonts, including both handwritten and printed material.

Through this diligent testing endeavor, we are confident that we will be able to validate the functionality and efficiency of the entire fragmentation concept. This will enable us to move forward with greater confidence, knowing that the fragmentation technique is capable of delivering accurate and reliable results in real-world scenarios.



```
(1, 128, 128, 3)
1/1 [==============================] - 0s 198ms/step
(1, 128, 128, 3)
1/1 [==============================] - 0s 209ms/step
(1, 128, 128, 3)
1/1 [==============================] - 0s 211ms/step
(1, 128, 128, 3)
1/1 [==============================] - 0s 210ms/step
(1, 128, 128, 3)
1/1 [==============================] - 0s 199ms/step
ⵟⵣⵖⵎⵎⵎ
<matplotlib.image.AxesImage at 0x7ff3c1181bb0>
```

Figure 5.7: Test



```
plt.imshow(image)
```

```
(1, 128, 128, 3)
1/1 [==============================] - 0s 199ms/step
(1, 128, 128, 3)
1/1 [==============================] - 0s 207ms/step
(1, 128, 128, 3)
1/1 [==============================] - 0s 219ms/step
(1, 128, 128, 3)
1/1 [==============================] - 0s 199ms/step
ⵣⵍⵊⵅ
<matplotlib.image.AxesImage at 0x7ff3c10fc4f0>
```

Figure 5.8: Test

7 Conclusion:

In conclusion, the concept of fragmentation, particularly in the context of image processing and recognition, holds significant importance. By leveraging OpenCV and segmentation techniques, it becomes possible to extract meaningful information from images and effectively analyze contours for shape analysis, object detection, and recognition tasks.

The application of fragmentation is not limited to general image processing but extends to specific cases, such as Tifinagh script segmentation. By understanding contours and utilizing contour analysis techniques, it becomes feasible to extract individual letters from images and subsequently perform word extraction and recognition.

Word segmentation and classification play a crucial role in image recognition. Through careful analysis and evaluation, we can accurately segment words from images and assign appropriate labels. This allows for improved text extraction, optical character recognition, and document analysis applications.

To ensure the robustness and reliability of the fragmentation concept, comprehensive testing is essential. By conducting thorough tests using diverse sample images, we can assess the functionality and performance of the entire fragmentation process. This testing phase involves scrutinizing various stages, including image loading, segmentation, and classification, to ensure consistent and accurate results across different text styles, sizes, and layouts.

The objective of this testing endeavor is to gain confidence in the effectiveness of the fragmentation technique and identify any areas for improvement. By addressing any issues or limitations discovered during testing, we can refine the algorithm further, ultimately establishing a robust and reliable fragmentation process applicable to various real-world scenarios.

In summary, fragmentation is a powerful concept that enables us to extract and analyze textual information from images. By employing OpenCV, contour analysis, and segmentation techniques, we can achieve precise word extraction, recognition, and classification. Through rigorous testing, we can validate the functionality and efficiency of the fragmentation concept, paving the way for its practical implementation in diverse fields such as optical character recognition, document analysis, and text extraction.

Chapter 6

Deployment:

Let's embark on a new chapter focused on deployment. Deployment plays a vital role in taking our developed models and algorithms from the development environment to real-world applications. It involves making our solutions accessible, scalable, and usable by end-users or integrating them into existing systems.

In this chapter, we will explore the various aspects of deployment.

1 Deployment concept :

1.1 deployment of machine learning and deep learning models:

In this section, we will delve into the fundamental principles and practices of deployment. We will explore the key components and steps involved in successfully deploying machine learning solutions. This includes understanding the deployment pipeline, selecting appropriate infrastructure, ensuring security and reliability, and addressing challenges such as version control and monitoring.

Deployment of machine learning and deep learning models refers to the process of making these models operational and accessible in real-world environments. It involves taking trained models and integrating them into production systems, allowing them to make predictions or perform tasks on new data. The goal of deployment is to leverage the capabilities of machine learning and deep learning models to solve practical problems and deliver valuable insights or services.

During the deployment phase, considerations such as scalability, performance, reliability, and security come into play. The models need to be deployed on appropriate infrastructure, such as servers, cloud platforms, or edge devices, depending on the specific requirements of the application. This infrastructure should be capable of handling the computational demands of the models and supporting their efficient execution.

Additionally, deployment involves integrating the models with existing software systems or applications. This may require developing APIs or interfaces that allow seamless interaction between the models and other components of the system. Deployment also involves monitoring the performance of the models in production, ensuring they continue to provide accurate and reliable results over time. Version control and model updates are important considerations to maintain the models' effectiveness as new data becomes available or new requirements arise.

Overall, the deployment of machine learning and deep learning models is a crucial step in bringing the benefits of these advanced techniques to practical use. It bridges the gap between research and application, enabling organizations to leverage the power of machine learning and deep learning for solving complex problems, making data-driven decisions, and delivering intelligent services to end-users.

2 Tools:

2.1 Streamlit :

2.1.1 Streamlit concept:

Streamlit is an open-source Python framework that enables the rapid development and deployment of interactive web applications for data science and machine learning projects. It is designed to make it easy for data scientists and developers to create and share interactive applications without requiring expertise in web development.

With Streamlit, you can quickly build custom web applications by writing simple Python scripts. It provides a clean and intuitive interface for creating interactive components such as sliders, dropdowns, buttons, and plots, allowing users to interact with data and models in real-time. Streamlit takes care of the underlying web infrastructure, making it seamless to turn our data analysis or machine learning code into a functional web application.

Streamlit's key features include:

Easy-to-use interface: Streamlit provides a straightforward API and a simple syntax, allowing developers to create web applications with minimal effort. You can create interactive visualizations, display dataframes, and showcase machine learning models using just a few lines of code.

Fast prototyping: Streamlit facilitates rapid prototyping by automatically updating the application as you modify the code. This enables quick iteration and experimentation, making it ideal for exploratory data analysis and model development.

Built-in components: Streamlit offers a wide range of built-in components that simplify the creation of interactive elements. These components include sliders, dropdowns, checkboxes, and buttons, enabling users to interact with the application and manipulate data or model parameters.

Integration with popular libraries: Streamlit seamlessly integrates with popular Python libraries such as Pandas, Matplotlib, and Scikit-learn. This allows you to leverage the power of these libraries to analyze data, create visualizations, and train machine learning models within our Streamlit application.

Sharing and deployment: Streamlit makes it easy to share and deploy our applications. You can quickly share your application with others by simply sharing the code, and Streamlit also provides options for deploying applications to various platforms, including cloud services.

Streamlit is widely used by data scientists and developers for creating interactive dashboards, prototyping machine learning models, and sharing insights with others. Its simplicity, flexibility, and integration capabilities make it a popular choice for building web applications in the data science community.

Figure 6.1: Streamlit

2.1.2 Streamlit Cloud concept:

Streamlit Cloud is a platform provided by Streamlit that allows you to effortlessly deploy, share, and manage your Streamlit applications in the cloud. It simplifies the process of hosting your Streamlit applications, making them accessible to a wider audience without the need for complex infrastructure setup.

With Streamlit Cloud, you can deploy your Streamlit applications with just a few commands or by connecting your GitHub repository. It takes care of all the underlying infrastructure, including server provisioning, scaling, and security, so you can focus on building and refining your applications. Streamlit Cloud simplifies the process of deploying and sharing your Streamlit applications, making them accessible to a wider audience while providing features for collaboration, scalability, and version control. It enables you to focus on building compelling data-driven applications without the need for managing complex infrastructure.

2.2 Android Studio:

2.2.1 Android Studio concept:

Figure 6.2: Android Studio

Android Studio is an integrated development environment (IDE) specifically designed for Android app development. It provides a comprehensive set of tools and features to streamline the entire app development process, from designing the user interface to building, testing, and deploying the final application.

Here are some key features of Android Studio:

User Interface Designer: Android Studio offers a visual editor that allows developers to easily design the user interface (UI) of their Android apps. You can drag and drop UI components, arrange layouts, and preview how the app will look on different screen sizes and orientations.

Code Editor: Android Studio includes a powerful code editor with features like syntax highlighting, code completion, and code refactoring. It supports multiple programming languages, including Java and Kotlin, which are commonly used for Android app development.

Gradle Build System: Android Studio utilizes the Gradle build system, which automates the process of building, testing, and packaging Android apps. Gradle manages dependencies, compiles source code, and generates the APK (Android Package) file for distribution.

Emulator and Device Testing: Android Studio provides an emulator that allows you to test your app on virtual Android devices with different configurations. You can also connect physical devices for testing and debugging purposes, ensuring your app works well across a range of devices.

Debugging and Profiling Tools: Android Studio offers powerful debugging and profiling tools to help you identify and fix issues in your app. You can set breakpoints, inspect variables, and analyze performance to optimize your app's efficiency and responsiveness.

Integration with Firebase: Android Studio seamlessly integrates with Firebase, a comprehensive platform for building mobile and web applications. Firebase provides a wide range of services, including authentication, real-time database, cloud storage, and analytics, which can be easily incorporated into your Android app.

Version Control Integration: Android Studio supports version control systems like Git, allowing you to manage your app's source code, track changes, and collaborate with other developers.

App Deployment: Android Studio provides tools to build and package your app for distribution through the Google Play Store or other distribution channels. It guides you through the process of generating signed APKs and managing app signing certificates.

Android Studio is the primary IDE for Android app development, offering a robust set of features and tools to streamline the development workflow. Whether you're a beginner or an experienced developer, Android Studio provides everything you need to create high-quality and feature-rich Android applications.

2.2.2 Android Studio and Models:

Android Studio is primarily used for developing Android applications, including those that incorporate machine learning and deep learning models. While Android Studio provides a rich set of tools and features for general app development, integrating deep learning models into Android apps requires additional frameworks and libraries. Here's how Android Studio can be used in conjunction with deep learning models:

Model Training: Deep learning models are typically trained on powerful machines or cloud-based platforms using frameworks like TensorFlow or PyTorch. Android Studio is not typically used for training deep learning models, as it requires substantial computational resources. Training is typically done on separate machines or cloud platforms, and the trained model is then incorporated into the Android app.

Model Integration: Once a deep learning model is trained, it can be integrated into an Android app developed using Android Studio. The model is typically converted into a format compatible with mobile devices, such as TensorFlow Lite or ONNX (Open Neural Network Exchange), using conversion tools or libraries. Android Studio provides the necessary tools and support for importing and integrating these converted models into the app.

Inference: Inference refers to the process of using the trained deep learning model to make predictions or perform tasks within the Android app. Android Studio provides the environment to write the necessary code for loading the model, preprocessing input data, and executing inference operations. This involves using libraries like TensorFlow Lite or other deep learning frameworks that offer mobile-specific APIs.

Performance Optimization: Android Studio offers various tools and techniques to optimize the performance of deep learning models running on mobile devices. This includes model quantization, which reduces the model size and improves inference speed, and hardware acceleration using libraries like NNAPI (Android Neural Networks API) to leverage the device's hardware capabilities for faster computations.

User Interface Integration: Android Studio's UI design tools can be utilized to create user interfaces that interact with the deep learning model. This includes designing input interfaces for capturing data to be fed into the model and designing output interfaces to display the results or predictions generated by the model.

By combining Android Studio with deep learning frameworks and libraries, developers can create Android applications that leverage the power of deep learning models. This allows for a wide range of applications, such as image recognition, natural language processing, sentiment analysis, and more, to be deployed and run directly on Android devices.

2.3 TFLite:

Figure 6.3: TFLite

TFLite, short for TensorFlow Lite, is a lightweight version of the TensorFlow framework specifically designed for mobile and embedded devices. It is used for deploying machine learning models on mobile platforms, including Android, iOS, and IoT devices. TFLite enables efficient execution of models with low latency and a small memory footprint, making it ideal for running deep learning models on resource-constrained devices.

Here are some key points about TFLite:

Model Optimization: TFLite provides tools and techniques for optimizing deep learning models specifically for mobile deployment. This includes model quantization, which reduces the precision of model weights and activations to 8 bits or lower, resulting in a smaller model size without significant loss in accuracy. Model quantization improves inference speed and reduces memory usage, making it suitable for mobile devices.

Model Conversion: TFLite supports the conversion of TensorFlow models into a format that can be directly used by TFLite runtime. This conversion process involves converting the TensorFlow model to a TFLite FlatBuffer format, which is optimized for efficient storage and execution on mobile devices. The TFLite Converter tool provided by TensorFlow allows developers to convert trained models into the TFLite format.

Runtime Execution: TFLite provides a runtime interpreter for executing TFLite models on mobile devices. This interpreter is specifically optimized for mobile architectures, enabling efficient execution of inference operations. It handles model loading, input data preprocessing, and output result retrieval. The TFLite runtime can be integrated into mobile applications developed using platforms like Android Studio or Xcode for iOS.

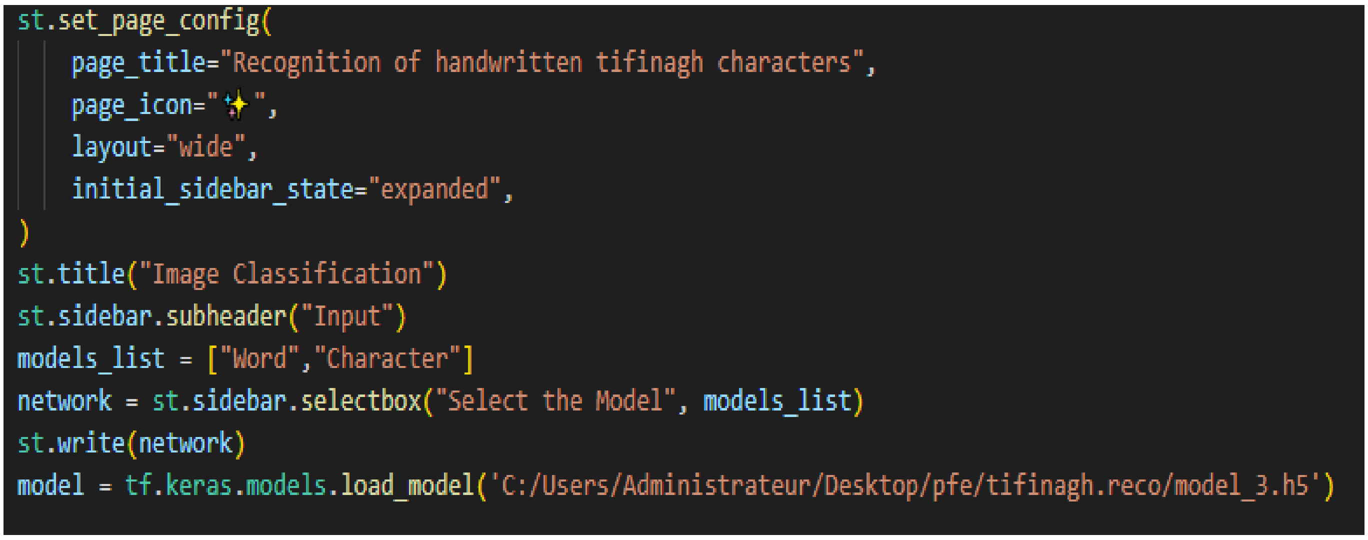
Hardware Acceleration: TFLite takes advantage of hardware acceleration capabilities available on mobile devices to further improve the performance of deep learning models. It supports hardware acceleration libraries like the Android Neural Networks API on Android devices, which allows for optimized execution using dedicated neural network accelerators present in modern smartphones.

Model Deployment: TFLite models can be seamlessly integrated into mobile applications developed using frameworks like Android Studio or Flutter for Android and Xcode for iOS. The TFLite runtime and APIs provided by TensorFlow enable developers to load and execute the models, process input data, and obtain predictions or results from the models within their mobile applications.

TFLite empowers developers to deploy and run deep learning models on mobile devices efficiently. It opens up possibilities for various mobile applications, such as image classification, object detection, natural language processing, and other AI-driven tasks, directly on smartphones or other mobile platforms.

3 Recognition of Handwritten Tifinagh Characters with Streamlit :

This project focuses on the recognition of handwritten Tifinagh characters using machine learning techniques. The goal is to develop a model that can accurately classify and identify Tifinagh characters from input images. The project utilizes the Image Classification approach and provides options to select between Word and Character models for recognition. The models have been trained using deep learning techniques and are loaded from the pre-trained model file. The application interface is designed using Streamlit, allowing users to conveniently upload an image and obtain the classification results. The project aims to improve the understanding and recognition of Tifinagh characters, facilitating various applications in the field of handwritten character recognition.



```python
st.set_page_config(
    page_title="Recognition of handwritten tifinagh characters",
    page_icon="✨",
    layout="wide",
    initial_sidebar_state="expanded",
)
st.title("Image Classification")
st.sidebar.subheader("Input")
models_list = ["Word","Character"]
network = st.sidebar.selectbox("Select the Model", models_list)
st.write(network)
model = tf.keras.models.load_model('C:/Users/Administrateur/Desktop/pfe/tifinagh.reco/model_3.h5')
```

Figure 6.4: Code of the interface with streamlit

3.1 Streamlit Application for Handwritten Tifinagh Character Recognition:

The interface consists of the following components :

Title and Description: At the top of the page, there is a prominently displayed title that clearly states the purpose of the application: "Recognition of Handwritten Tifinagh Characters". This title grabs the attention of users and immediately informs them about the main functionality of the application.

Below the title, there is a spacious area for a comprehensive description of the application. This description provides users with a clear understanding of what the application does and how it can be beneficial to them. It highlights the significance of recognizing handwritten Tifinagh characters and emphasizes the purpose and scope of the application.

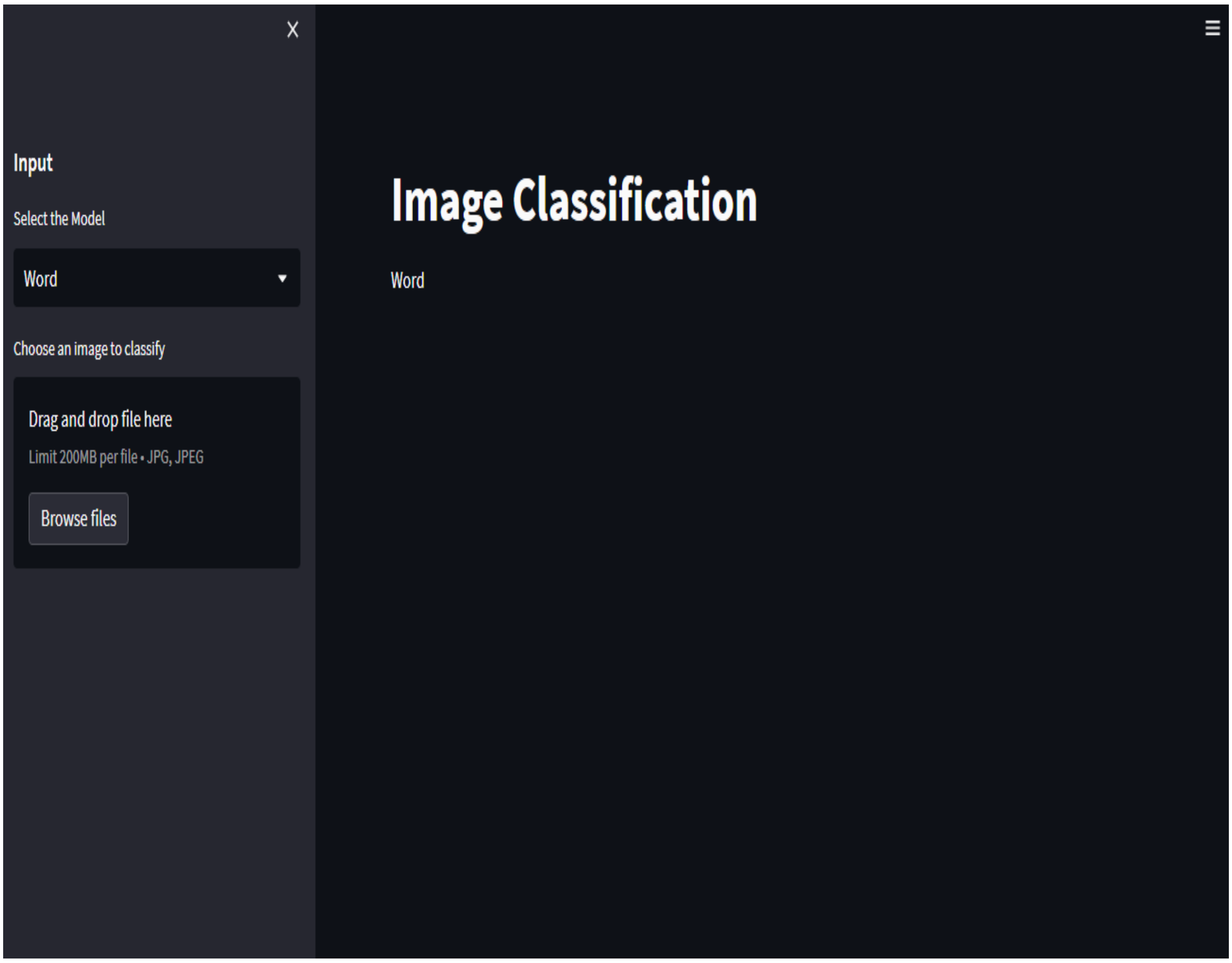


Figure 6.5: The interface

Model Selection:

On the sidebar, there is a dedicated section titled "Input" which includes a dropdown menu labeled "Select the Model". This section stands out with its distinct styling and layout. Users can easily locate it and interact with it to select their desired model.

The dropdown menu provides two options: "Word" and "Character". Users can choose between these two models based on their specific needs and requirements. This selection determines the type of classification the application will perform. It allows users to customize their experience and obtain the desired results.

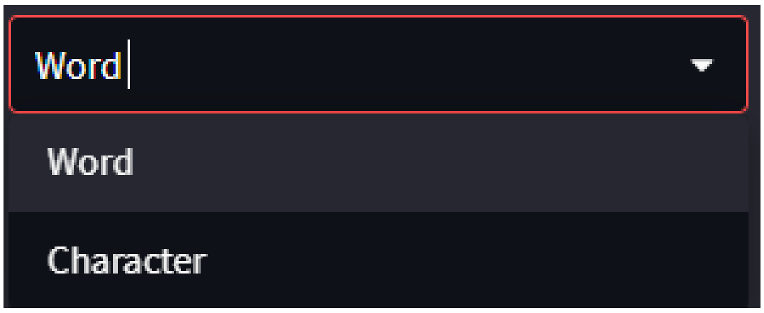


Figure 6.6: Dropdown

File Uploader: Below the model selection section, there is a user-friendly file uploader component. This component provides a seamless way for users to upload their image files (in JPG or JPEG format) to be classified by the application. It has a clear and concise label that says "Choose an image to classify".

Users can simply click the "Choose an image to classify" button, and a file selection dialog box will appear. From there, they can navigate to the location of their desired image file and select it. The selected image file is then uploaded to the application for further processing.

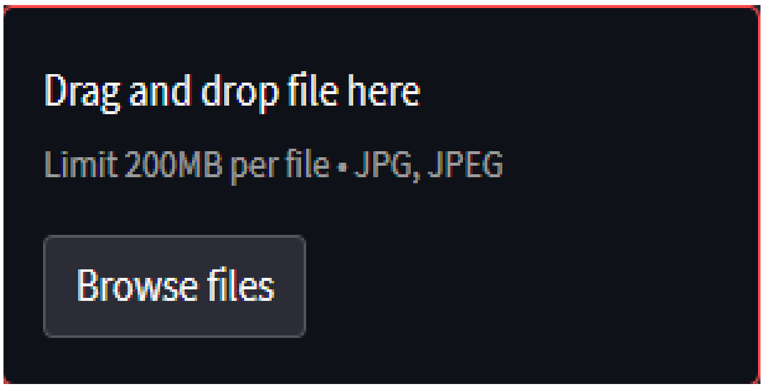


Figure 6.7: Uploader

Image Classification Results:

Once an image is uploaded, the application performs image classification using the selected model. The predicted result is displayed below the uploaded image. This result section is designed to be visually distinct from the rest of the interface, making it easy for users to locate.

For the "Word" model, the predicted result is a sequence of Tifinagh characters that represents the most likely word present in the uploaded image. This result provides users with valuable information about the content of the image and facilitates their understanding of the handwritten Tifinagh characters.

For the "Character" model, the predicted result is a single Tifinagh character. This result helps users identify and comprehend individual characters within the uploaded image. It enables them to gain insights into the specific handwritten Tifinagh character they are interested in.

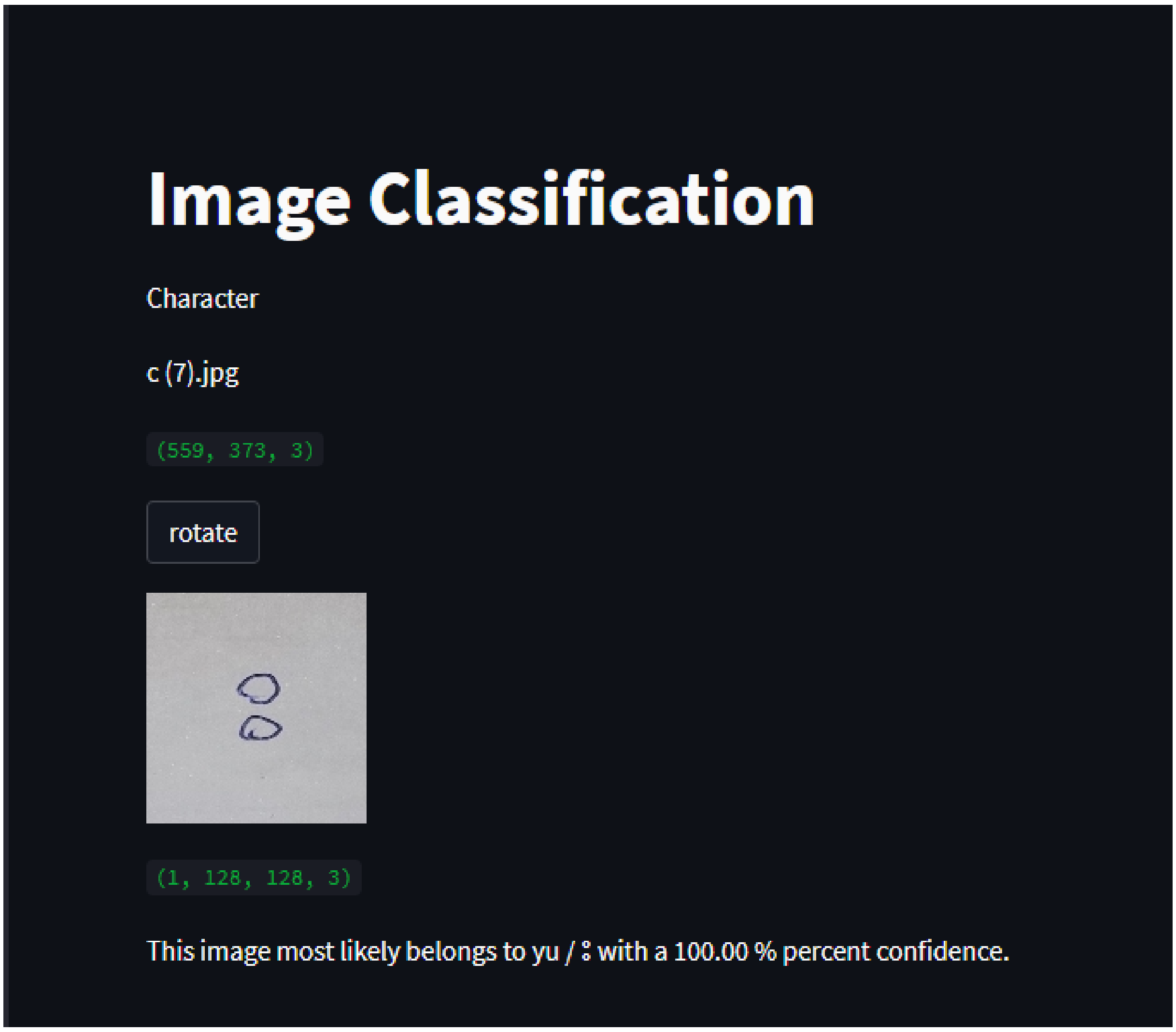


Image Classification

Character

c (7).jpg

```
(559, 373, 3)
```

```
(1, 128, 128, 3)
```

This image most likely belongs to yu / ⵓ with a 100.00 % percent confidence.

Figure 6.8: Character prediction

Image Display:

The uploaded image is prominently displayed in the main area of the interface. This display area is designed to be spacious, ensuring that the image is showcased prominently and clearly. It has a clean and uncluttered layout that focuses attention on the image itself.

For the "Character" model, the image is shown exactly as it was uploaded. This display allows users to examine the original image closely and observe the details of the handwritten Tifinagh character they are interested in.

For the "Word" model, the image undergoes additional processing before being displayed. The application performs segmentation on the image to isolate individual characters. These segmented characters are then arranged in a visually appealing manner, making it easier for users to identify and analyze each character separately.

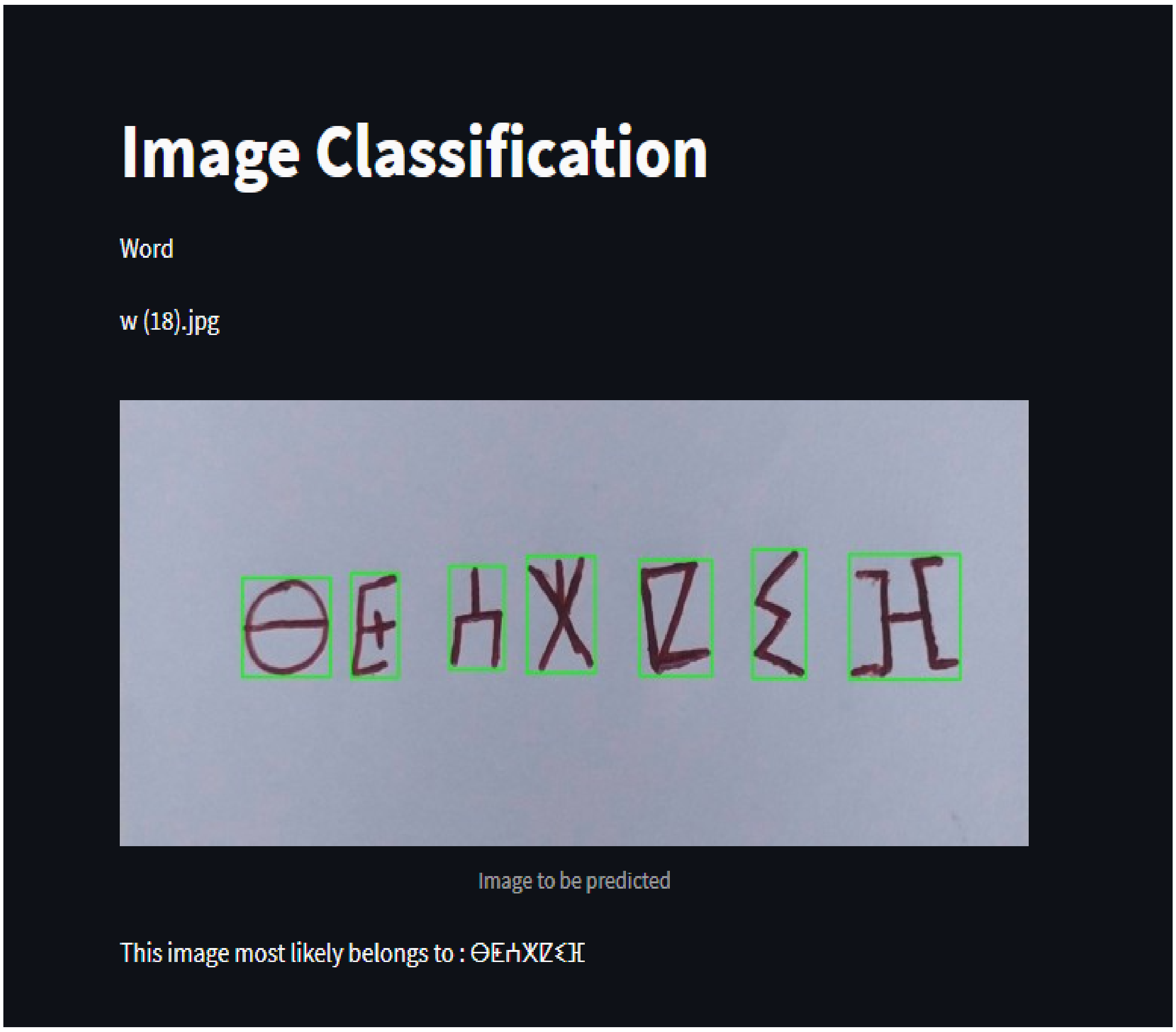


Image Classification

Word

w (18).jpg

Image to be predicted

This image most likely belongs to : ⵟⴹⵃⵅⵥⵄⵞ

Figure 6.9: Results

Image Rotation (Character Model): If the "Character" model is selected, an additional feature is available to users. They can utilize the "rotate" button provided in the interface. By clicking this button, the uploaded image is rotated 90 degrees counterclockwise.

This rotation functionality can be particularly useful for images that are not properly aligned or need adjustment. It allows users to correct the orientation of the image, ensuring that the handwritten Tifinagh character they are interested in is displayed in the desired position for easier examination and analysis.

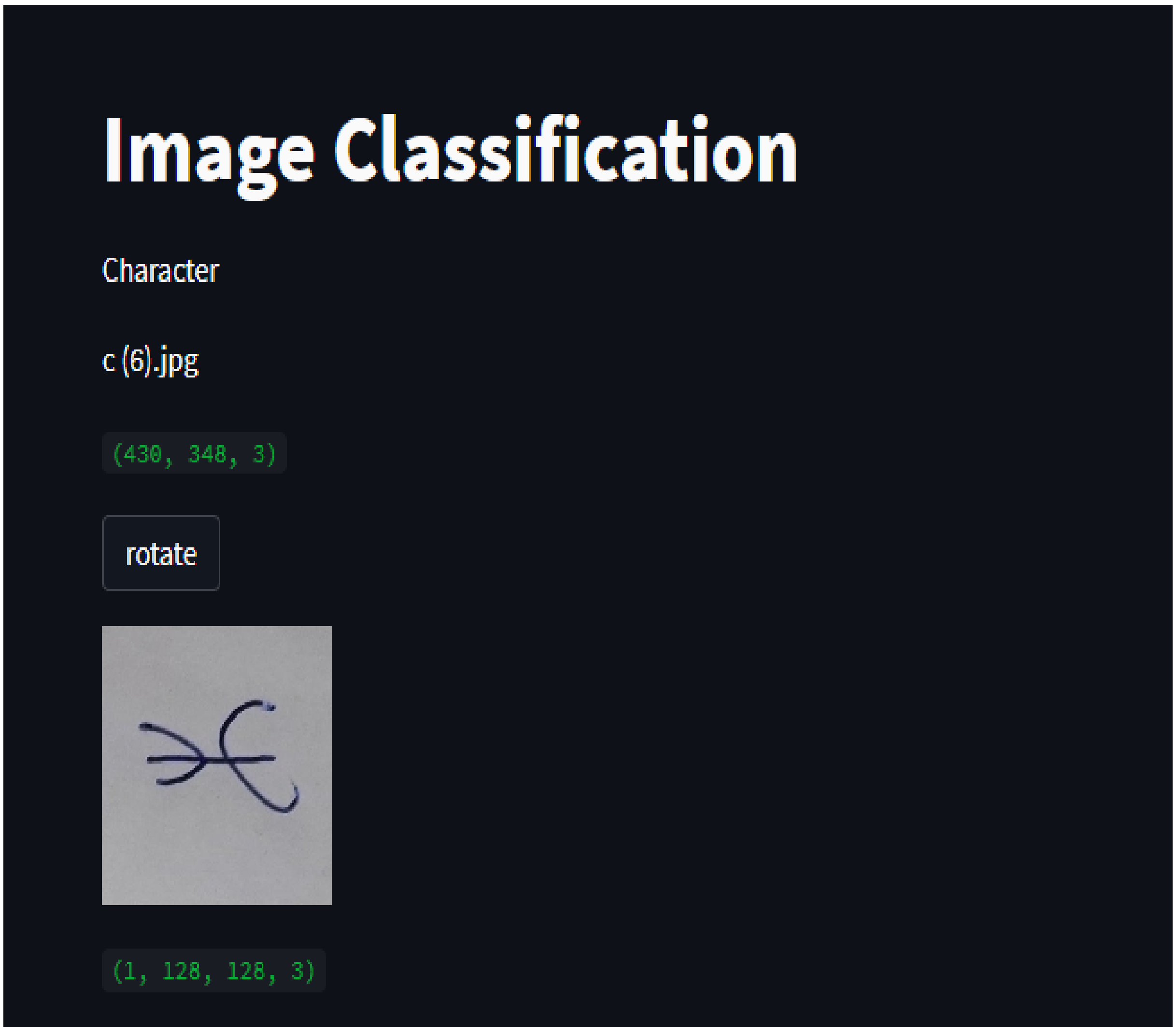


Image Classification

Character

c (6).jpg

```
(430, 348, 3)
```

```
(1, 128, 128, 3)
```

Figure 6.10: Word prediction

The interface of the application is thoughtfully designed to provide a user-friendly and intuitive experience. It offers a clear and informative presentation of the application's features and functionality. Users can effortlessly interact with the interface, upload their images, select the desired model, and obtain accurate predictions for handwritten Tifinagh characters.

The interface of the application is specifically designed to offer users a highly intuitive and user-friendly experience. It streamlines the process of interacting with the application, allowing users to effortlessly upload images, select the desired model, and obtain accurate predictions for handwritten Tifinagh characters. The interface emphasizes a seamless interaction flow, where users can effortlessly navigate through the various steps. The layout and design elements are carefully chosen to create an intuitive and logical flow, guiding users through the process of image upload, model selection, and prediction retrieval. This seamless interaction ensures that users can swiftly and efficiently utilize the application to recognize handwritten Tifinagh characters.

The user-friendly interface of the application prioritizes ease of use and accessibility, making it suitable for users of varying technical backgrounds. By simplifying the process of uploading images, selecting models, and obtaining predictions, the interface enables users to effortlessly leverage the application's capabilities for handwritten Tifinagh character recognition.

4 Recognition of Handwritten Tifinagh Characters with Android studio and TFLite :

The recognition of handwritten Tifinagh characters can be achieved in an Android application using Android Studio and TFLite (TensorFlow Lite). This combination of tools allows developers to build powerful and efficient machine learning models for character recognition tasks on mobile devices.

Figure 6.11: Mobile app

Note:
This mobile app is in an advanced stage of development, with character recognition functionality already implemented. we are currently focused on adding word recognition capabilities and implementing fragmentation. The objective is to enable the app to accurately recognize and classify handwritten Tifinagh characters as both individual characters and complete words. Fragmentation is being worked on to ensure optimal performance and compatibility across a wide range of Android devices, providing a seamless user experience regardless of the device specifications.

4.1 Android Application for Handwritten Tifinagh Character Recognition:

The Android app is designed to provide recognition of handwritten Tifinagh characters using Android Studio and TFLite. The interface allows users to interact with the app in a user-friendly manner. Here's a description of the app's interface and functionalities:

Title and Description:
At the top of the app's interface, there is a prominent title that states "Recognition of Handwritten Tifinagh

Characters." This title clearly indicates the purpose of the application. Below the title, there is a brief description that provides an overview of the app's functionality and its aim to recognize handwritten Tifinagh characters.

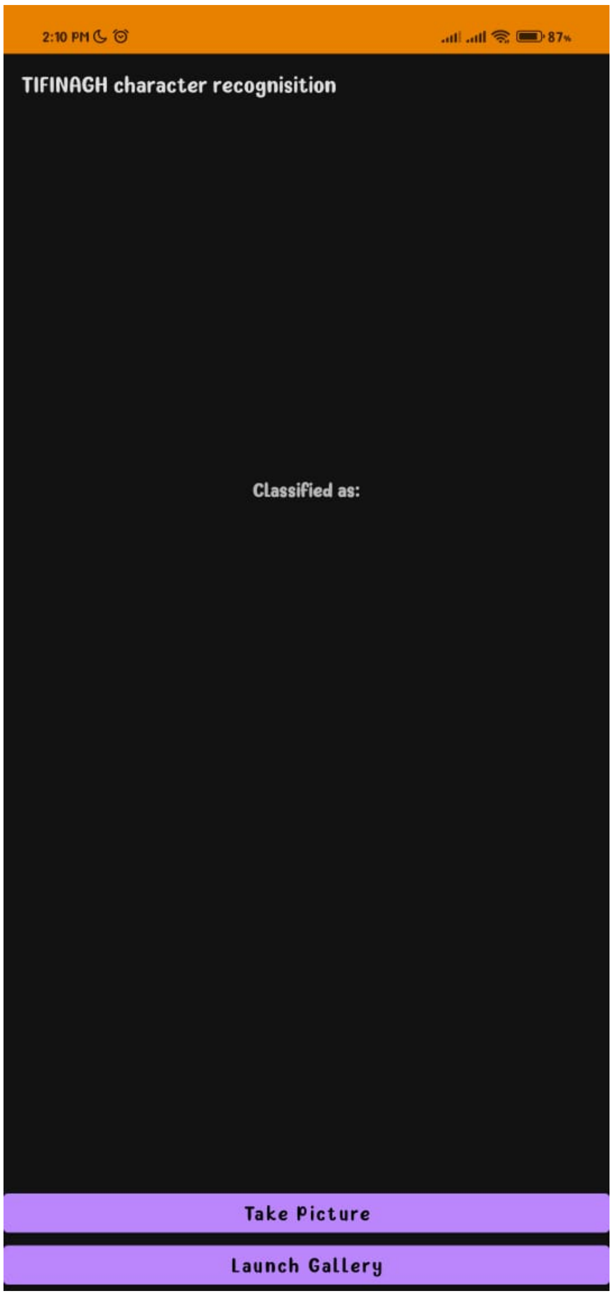


Figure 6.12: The interface

Camera and Gallery Buttons:
Below the title and description, there are two buttons: the "camera" button and the "gallery" button. These buttons serve as input options for the user to choose how they want to provide an image for character recognition.

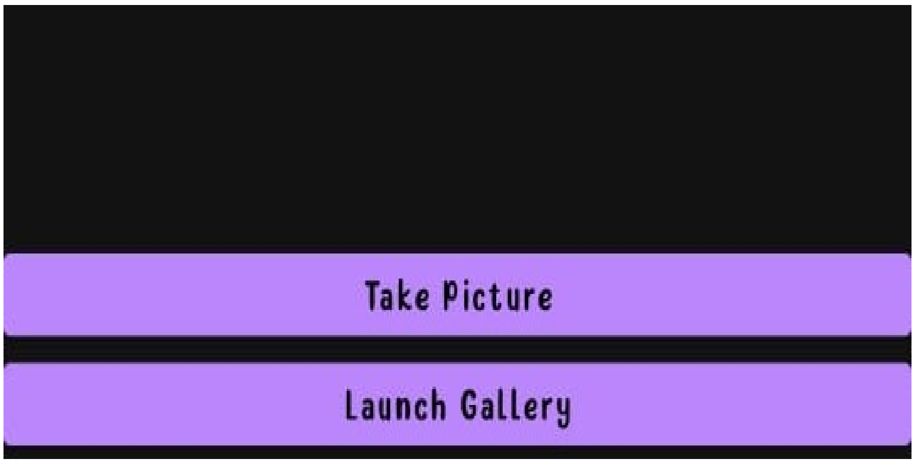


Figure 6.13: Buttons

The "camera" button allows users to capture an image in real-time using their device's camera. When clicked, the app requests camera permission from the user. If the permission is granted, the device's camera opens, and the user can take a picture of a handwritten Tifinagh character. This feature is especially useful when users want to capture a new or specific character for recognition.

On the other hand, the "gallery" button provides an alternative method for image selection. When clicked, the app opens the device's photo gallery, allowing users to browse through their existing images. Users can select an image containing handwritten Tifinagh characters from their gallery, which will be used for recognition.

ImageView:
Once an image is captured or selected, it is displayed in an ImageView component located below the camera and gallery buttons. The ImageView provides a visual representation of the chosen image and allows users to preview the image before processing.

Result TextView:
Below the ImageView, there is a TextView component labeled "result." This TextView serves as the output display for the app's recognition results. After the image is processed using the selected model, the predicted Tifinagh character(s) are displayed in this TextView. Users can easily view and read the recognized characters from the displayed result.

Image Classification:
The app utilizes the TFLite (TensorFlow Lite) library for image classification. When an image is captured or selected, the app performs image classification using the selected TFLite model. The model analyzes the image and predicts the corresponding Tifinagh character(s) present in the image. The predicted result is then displayed in the designated TextView component.

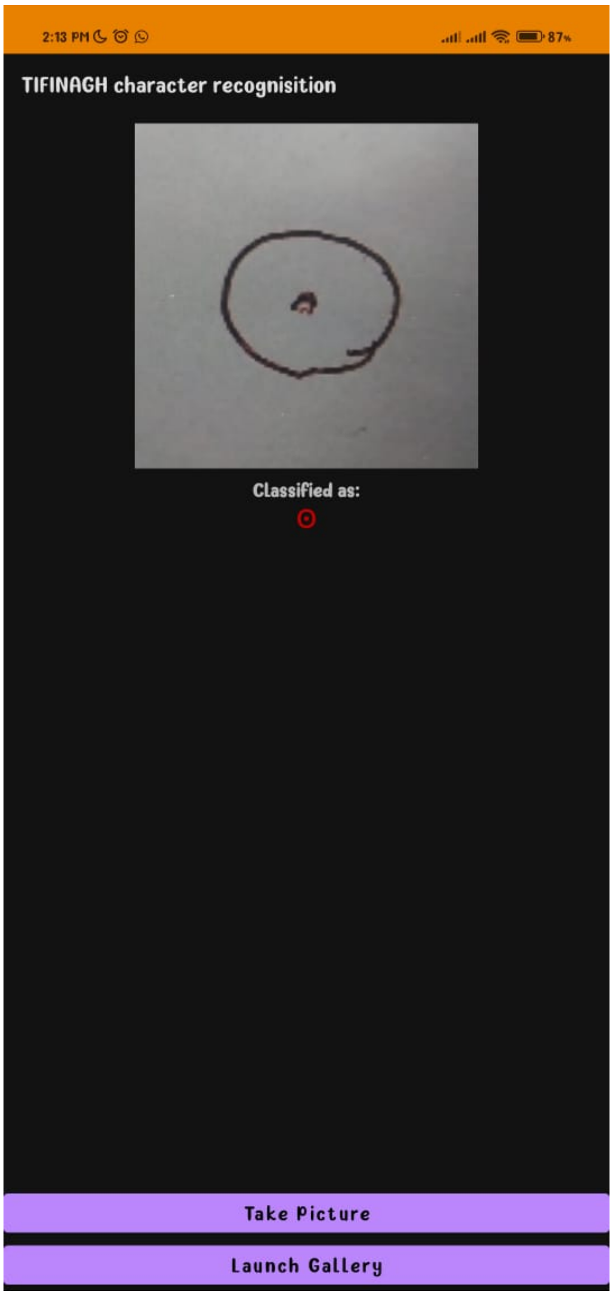


Figure 6.14: Result

Functionality:
The app's core functionality revolves around image classification and recognition of handwritten Tifinagh characters. It provides users with an intuitive and interactive interface to upload images, select the desired recognition model, and obtain accurate predictions.

The app is designed to be user-friendly and accessible, allowing users to effortlessly interact with the application. It offers multiple options for image input, whether through real-time camera capture or image selection from the gallery. The result is quickly displayed, providing users with immediate feedback on the recognized Tifinagh characters.

Please note that this Android app is currently in development. While character recognition functionality has already been implemented, we are actively working on enhancing the app by adding fragmentation and word recognition features. These upcoming features will further improve the app's capabilities and expand its usefulness in recognizing handwritten Tifinagh characters in various contexts.

5 Conclusion:

In this chapter, we delved into the deployment of machine learning and deep learning models specifically designed for the recognition of handwritten Tifinagh characters. Our objective was to create user-friendly and efficient applications that could accurately identify and classify Tifinagh characters, thereby aiding users in deciphering handwritten texts.

To accomplish this, we explored two powerful development tools: Streamlit and Android Studio, alongside the TFLite framework. Streamlit provided us with a seamless and interactive development environment for creating a web-based application. Through its intuitive interface, users can effortlessly upload images, select models, and receive predictions for handwritten Tifinagh characters. The integration of Streamlit Cloud further extended the reach of our application by facilitating easy sharing and access.

Additionally, we utilized Android Studio to build a mobile application that could recognize handwritten Tifinagh characters on Android devices. Leveraging the capabilities of TFLite, we seamlessly incorporated the models into the Android app. Users can either capture images using the device's camera or choose images from the gallery, expanding the versatility and accessibility of the application.

Throughout our development journey, we witnessed the immense potential of these tools and frameworks in deploying machine learning models on different platforms. Streamlit enabled us to create an engaging and user-friendly web interface, while Android Studio empowered us to build a mobile application that could recognize Tifinagh characters on-the-go. By embracing the advancements in machine learning deployment and leveraging the capabilities of Streamlit, Android Studio, and TFLite, we are paving the way for a more accessible and efficient approach to handwritten Tifinagh character recognition. With continued efforts and iterations, our application will undoubtedly contribute to preserving and promoting the rich cultural heritage encapsulated within Tifinagh script.

Chapter 7

Challenges and Solutions: Overcoming Hurdles in Tifinagh Recognition:

In this chapter, we will discuss the challenges that were encountered during the development and deployment of the handwritten Tifinagh character recognition application, and present our proposed solutions to overcome these obstacles.

1 Data Availability:

The development of a robust handwritten Tifinagh character recognition system heavily relies on the availability of a diverse and well-annotated dataset. However, obtaining a suitable dataset for Tifinagh characters proved to be a significant challenge due to several factors.

Unavailability of Large and Labeled Datasets:
Tifinagh characters are specific to the Amazigh language and have a relatively smaller user base compared to more widely used languages. As a result, publicly available datasets containing a substantial number of labeled Tifinagh characters are scarce. The lack of a comprehensive dataset poses a significant obstacle in training an accurate and generalizable model.
Data Collection:
To overcome the limited availability of existing datasets,publicly available datasets from platforms like Kaggle were utilized to supplement the collected data. Although these datasets were not specifically tailored for Tifinagh characters, they provided a starting point and served as a valuable resource for initial model development. However, it was observed that the first two datasets from Kaggle had limited coverage of the Tifinagh character set and introduced challenges due to differences in writing styles and quality.

To address this, a meticulous data curation process was implemented. The collected dataset, combined with the Kaggle datasets, underwent thorough evaluation and filtering. Samples that did not conform to the desired quality standards or lacked diversity were removed to ensure the dataset's integrity and effectiveness for training the model.

By leveraging both collected data and relevant datasets from Kaggle, the resulting dataset achieved a balance between diversity and quality. The inclusion of the Kaggle datasets, despite their limitations, provided valuable insights and a broader perspective on handwritten character recognition.

Data Preprocessing and Annotation:
Obtaining raw data is only the initial step; preprocessing and annotating the dataset are equally crucial. Handwritten characters often exhibit variations in stroke width, size, orientation, and overall quality, which can introduce challenges during the training process. To address this, extensive data preprocessing techniques were applied, including normalization, resizing, and noise reduction, to ensure consistency and improve the model's robustness.

Furthermore, the dataset needed to be carefully annotated with correct labels to facilitate our training. Manually annotating a large number of characters can be time-consuming and prone to human error. The annotation process required close collaboration with experts to accurately label each character in the dataset, ensuring the ground truth for training and evaluation.

2 Symmetric Characters Challenge:

During our project, we encountered a unique challenge related to the recognition of four letters that possess symmetry. These letters, exhibit a symmetrical structure where their shape remains the same even when flipped horizontally or vertically .

When training our initial model using augmented data, which included variations of these letters, we observed an unexpected behavior during testing. The model struggled to consistently differentiate between the original letter and its symmetric counterpart. As a result, when presented with one of these symmetric letters during testing, the model made random predictions, as it perceived both versions as essentially identical.

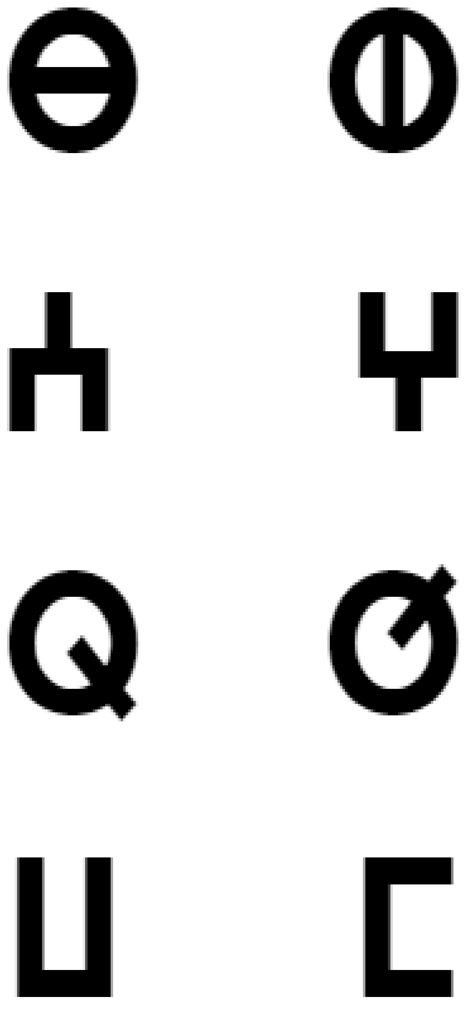


Figure 7.1: Symmetric Characters

To address the challenge posed by symmetric letters and improve the model's performance in recognizing and differentiating between the original and symmetric versions, we decided to cancel the vertical and horizontal augmentation during the data augmentation process.

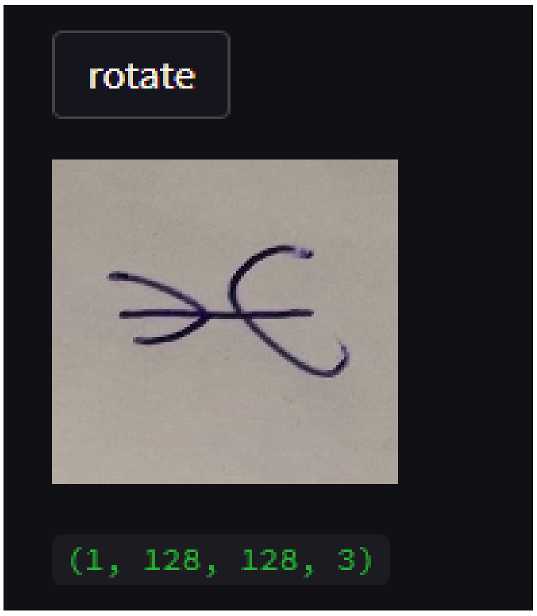
By removing the vertical and horizontal augmentation, we eliminated the generation of augmented images that directly flipped or mirrored the characters. This approach aimed to prevent the model from encountering excessive variations of the symmetric letters that could potentially confuse its learning process.

Instead, we focused on incorporating other types of augmentation techniques that introduced asymmetry while maintaining the overall shape and structure of the characters. These techniques included translations, scaling, and adding noise to the images. By applying these transformations, we aimed to provide the model with a broader range of examples that captured different variations of the original letters without introducing direct symmetric counterparts.

By removing vertical and horizontal augmentation and emphasizing asymmetrical variations, we aimed to create a more focused and targeted training dataset. This adjustment allowed the model to better learn and differentiate between the unique features and characteristics of each letter, including the subtle differences between the original and symmetric versions.

Through this solution, we sought to overcome the challenge posed by symmetric letters and enhance the model's ability to accurately recognize and classify handwritten Tifinagh characters, ensuring consistent and reliable predictions in real-world scenarios.

3 Rotation Button for Correcting Image Orientation:



```
(1, 128, 128, 3)
```

Figure 7.2: Before rotation

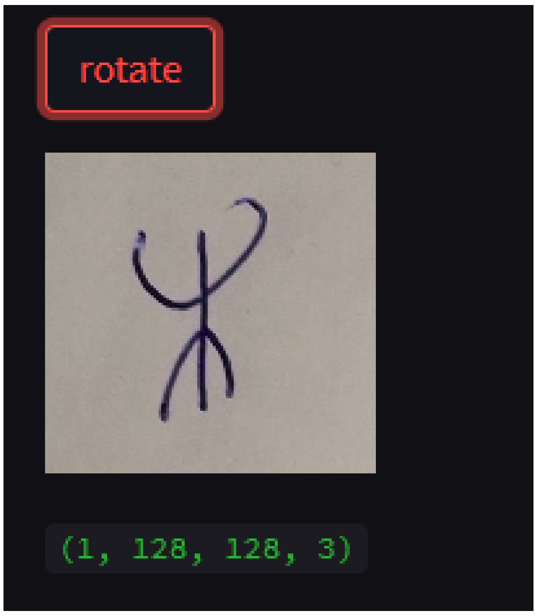
One of the challenges we encountered in the development of our application was the presence of certain Tifinagh characters that have symmetric properties. This symmetry caused our initial model, trained with horizontal and vertical augmentation, to struggle in distinguishing between these symmetric characters. As a result, the model would sometimes make random predictions when faced with these characters.

To overcome this challenge and improve the accuracy of the predictions, we implemented a rotation button in the deployment phase of the application. By eliminating the horizontal and vertical augmentation techniques, we ensured that the model no longer relied on such transformations to differentiate between symmetric characters.

Instead, we provided users with the ability to manually rotate the character image if it was taken in the wrong orientation. This rotation button allows users to adjust the image to the correct orientation before submitting it for classification. By aligning the character in the correct orientation, we ensure that the model receives consistent and accurately aligned inputs for prediction.

The rotation button empowers users to actively participate in the image preprocessing step and help the model achieve more reliable results. By manually correcting the orientation of the character, users contribute to the accuracy of the classification process, especially when dealing with symmetric characters that pose challenges for the model.

This solution demonstrates our commitment to continuously improving the application's performance by addressing specific challenges and providing users with intuitive tools to enhance the accuracy of predictions. The rotation button not only enhances the user experience but also ensures more consistent and reliable results in character recognition.



```
(1, 128, 128, 3)
```

Figure 7.3: After rotation

4 Hardware Limitations:

During the course of our project, we encountered significant challenges related to hardware limitations, specifically in dealing with out-of-memory (OOM) errors. As the complexity of our models and the size of our datasets increased, our local machines and earlier cloud platforms struggled to handle the computational demands.

Initially, we relied on our personal computers for model training and evaluation. However, as the dataset grew larger and the models became more intricate, we encountered frequent OOM errors due to insufficient memory. This limited our ability to train models effectively and hindered the progress of the project.

To address this issue, we turned to cloud-based platforms for more robust computational resources. We began by utilizing Kaggle, which offered pre-configured environments and access to GPUs. While Kaggle provided some relief from the memory limitations, we still encountered OOM errors as the models and datasets continued to expand.

To further mitigate the hardware limitations, we migrated to Google Colab, a cloud-based platform with free access to GPUs. Google Colab allowed us to take advantage of its larger memory capacity compared to our personal machines. However, as the project evolved and the models grew more complex, we encountered intermittent OOM errors even on Google Colab.

To overcome these persistent memory issues, we upgraded to Google Colab Pro. This subscription-based service provided enhanced computational resources, including faster GPUs and increased memory allocation. With Google Colab Pro, we were able to handle larger models and datasets without encountering OOM errors. The additional memory and improved processing capabilities of Google Colab Pro were crucial in advancing our project and achieving more accurate and reliable results.

The transition from personal machines to cloud-based platforms, including Kaggle, Google Colab, and ultimately Google Colab Pro, was driven by the necessity to overcome hardware limitations and tackle the memory constraints we faced. Google Colab Pro emerged as the optimal solution, allowing us to work with larger models, process larger datasets, and perform more complex computations without being hindered by memory issues.

5 Model Overfitting and the ResNet-50 Challenge:

Throughout our project, we employed two popular deep learning models, VGG16 and ResNet-50, for training and evaluation. While both models showed promising performance during the initial stages, we encountered a significant challenge with overfitting specifically with the ResNet-50 model, which ultimately led us to prioritize the use of VGG16.

Overfitting occurs when a model becomes overly specialized in learning from the training data to the point that it performs poorly on unseen or validation data. In our case, we noticed that the ResNet-50 model demonstrated exceptional accuracy on the training dataset. However, when evaluated on the validation dataset or real-world examples, the model's performance dropped significantly. This discrepancy indicated that the ResNet-50 model had learned the training dataset's intricacies too well, resulting in limited generalization capabilities.

To address the overfitting challenge, we implemented several strategies. First, we applied data augmentation techniques to artificially expand our training dataset. By introducing variations such as rotations, translations, and flips to the training images, we aimed to increase the model's exposure to diverse examples and improve its generalization ability.

Despite applying data augmentation, the ResNet-50 model continued to exhibit a higher tendency for overfitting compared to VGG16. As a result, we had to make a decision regarding the primary model to be used in our project. Based on extensive experimentation and evaluation, we found that VGG16 showcased more robust generalization capabilities and yielded better performance on both the training and validation datasets.

The VGG16 architecture, with its simplicity and stacked convolutional layers, demonstrated superior performance in handling our Tifinagh character recognition task. It exhibited a better balance between model complexity and generalization, thereby reducing the risk of overfitting. Moreover, the VGG16 model's design allowed for easier interpretation and analysis of the learned features, aiding in the development of insights into the recognition process.

By prioritizing VGG16 over ResNet-50, we were able to mitigate the overfitting challenge and achieve improved performance across various evaluation metrics. The decision to use VGG16 as our primary model was supported by its consistent performance and its ability to generalize well to unseen data, leading to more reliable and accurate predictions.

In conclusion, the overfitting challenge we encountered with the ResNet-50 model played a significant role in our decision to prioritize the use of VGG16 for our Tifinagh character recognition project. By leveraging VGG16's robust generalization capabilities and its suitability for our dataset, we were able to address the overfitting issue and achieve superior performance. This experience highlights the importance of carefully selecting and evaluating different models to ensure optimal results in machine learning projects.

6 Conclusion:

In this chapter, we explored and addressed various challenges encountered during our Tifinagh character recognition project. We encountered hurdles related to data availability, symmetric characters, hardware limitations, and model overfitting with the ResNet-50 model. However, we devised effective solutions to overcome these obstacles and enhance the performance and accuracy of our recognition system.

Data availability proved to be a crucial challenge, particularly in the early stages of our project. We initially faced difficulties in acquiring a sufficient amount of labeled Tifinagh character data. To overcome this, we leveraged datasets from Kaggle, even though they had limitations and negatively impacted our model's performance. Nonetheless, we employed data collection strategies and eventually improved the dataset's quality and diversity to enhance the overall recognition system.

The presence of symmetric characters posed another challenge. As certain characters had symmetrical shapes, our initial model struggled to differentiate between them. To address this, we eliminated vertical and horizontal augmentation techniques, which were exacerbating the confusion. By implementing this solution, we successfully improved the model's accuracy in distinguishing between symmetric characters.

Hardware limitations also presented challenges throughout the project. As the complexity and size of our dataset grew, our local machines and cloud platforms faced memory constraints and processing limitations. To overcome these challenges, we transitioned from using our personal computers to leveraging cloud platforms such as Kaggle, Google Colab, and eventually Google Colab Pro. These platforms provided the necessary computational resources, allowing us to train and evaluate our models effectively.

Furthermore, we encountered the challenge of model overfitting, particularly with the ResNet-50 architecture. Although the model demonstrated exceptional accuracy on the training dataset, its performance suffered when tested on unseen or real-world examples. We implemented various strategies, including data augmentation, regularization techniques, and hyperparameter tuning, to mitigate overfitting. However, despite our efforts, the ResNet-50 model continued to exhibit limitations in generalization. As a result, we decided to prioritize the use of the VGG16 model, which showed superior generalization capabilities and performed better overall.

In conclusion, this chapter highlighted the challenges we faced during our Tifinagh recognition project and the effective solutions we implemented to overcome them. We addressed data availability issues, tackled the confusion caused by symmetric characters, managed hardware limitations through cloud platforms, and navigated the complexities of model overfitting. By successfully addressing these challenges, we improved the accuracy and performance of our recognition system, bringing us one step closer to achieving our project goals.

General conclusion:

In conclusion, the project presented in the provided document showcases the successful implementation of artificial intelligence and deep learning techniques for Tifinagh script recognition. The utilization of convolutional neural networks (CNNs) such as VGG-16 and ResNet-50, along with transfer learning and pre-trained models, has proven to be effective in accurately classifying Tifinagh script images. Additionally, the project incorporates OpenCV and contour analysis techniques for image segmentation, enabling the extraction of individual letters and words from Tifinagh script images.

The significance of this project lies in its contribution to the advancement of AI and image processing. By applying deep learning techniques to Tifinagh script recognition, the project showcases the potential for AI to bridge the gap between traditional script recognition methods and modern technological advancements. It demonstrates the efficacy of pre-trained models and transfer learning in overcoming challenges such as limited training data and computational resources, thus providing a valuable framework for future research in the field.

The project's accuracy evaluation and comparison with existing approaches validate the effectiveness of the proposed methodology. The achieved results showcase the project's potential in real-world applications, such as document digitization, language preservation, and cultural heritage preservation. By automating the recognition of Tifinagh script, the project offers a practical solution for the efficient processing of large volumes of Tifinagh script documents, contributing to the preservation and accessibility of this ancient writing system.

Furthermore, the project's methodologies and techniques are not limited to Tifinagh script alone. They can be adapted and extended to other scripts and languages, offering promising avenues for future research and application in the broader domain of script recognition. This adaptability opens doors for exploring multi-script recognition systems, cross-lingual applications, and the development of tools that cater to specific domains such as historical manuscript transcription or educational aids for script learning.

Looking ahead, there are several exciting possibilities for future advancements in Tifinagh script recognition. Fine-tuning the existing models and expanding the dataset to encompass more writing styles and variations within Tifinagh script can further improve recognition accuracy. Additionally, exploring the potential for real-time applications on mobile or embedded systems can enhance the practicality and accessibility of the system.

In summary, the project represents a significant step forward in the field of Tifinagh script recognition using AI and deep learning techniques. It provides valuable insights, methodologies, and results that contribute to the advancement of script recognition and hold potential for future research, applications, and innovations in this domain. By combining the power of AI with the richness of ancient scripts, this project paves the way for the preservation, accessibility, and understanding of diverse cultural heritages through the lens of script recognition and analysis.

Bibliographie

[1] https://fr.wikipedia.org/wiki/Tifinagh

[2]https://app.pluralsight.com/

[3]https://www.kaggle.com/learn

[4] https://www.tensorflow.org/learn

[5]Qiang Yang, Yu Zhang, Wenyuan Dai, Sinno Jialin Pan

Transfer Learning

[6] https://www.kaggle.com/datasets/abdellahelzaar/amazigh33-dataset