AVATAR CUSTOMIZATION USING DEEP LEARNING'S STYLE TRANSFER TECHNOLOGY

Roberto Montagud Cenalmor
Final Degree Project
Bachelor's Degree in Video Game Design and Development
Universitat Jaume I

July 2023

Tutor: Rafael Fernández Beltrán

Acknowledgments

First of all, I would like to thank all the teachers and classmates who have accompanied and helped me in my journey and stage at the UJI, without whom I would not have been able to reach this decisive moment in my training.

I would also like to mention the figure of my tutor Rafael Fernández, who has guided, supervised and given me light at specific moments of this last course, being decisive and comforting for me his presence and help.

Abstract

An important aspect in the world of video games is the player's identity, In this regard player´s avatar plays a significant role in this. However, we often find ourselves limited to a few predefined options, which can restrict our individual expression. This document presents a project report on an application aiming to address this issue by providing more customization options to players. This application, developed in tkinter, utilizes a Style Transfer Model [1] using deep learning techniques (Specifically Convolutional Neural Networks) to transform user's self images into a specific artistic style.

Also, it offers a user-friendly interface where users can upload their avatar images or take a photo, and choose from a predefined list of artistic styles. Once the desired style is selected, the application applies the style transfer model to generate the transformed image.

Keywords

Digital Avatar Customization
Artificial Intelligence
Deep learning.
Neural Style Transfer.
Convolutional Neural Network.

Contents

1. INTRODUCTION

Contents

This document presents a comprehensive overview of the project, including its inception, development and the results obtained throughout this effort. Within this introductory section, we explore the driving forces that led to the initiation of this project, outlining its objectives and delineating the initial conditions. It also encompasses a thorough analysis of the decisions made, encompassing both externally imposed and self-imposed considerations, before embarking on this work.

1.1 Work Motivation

The project was initially proposed by the tutor based on an open idea for the generation of these custom avatars, from there, personal motivations come into play.

The creation of an application for the personalization of avatars through style transfer has multiple motivations. One of them is the desire to explore and learn about the field of deep learning and artificial intelligence. The implementation of style transfer techniques involves the understanding and application of advanced algorithms and models, which provides a valuable opportunity to acquire practical knowledge in this area. It is important to highlight that artificial intelligence is gaining increasing relevance in everyday life and in more ambitious applications, encompassing virtually all areas of society. As a result, there is a growing number of people interested in artificial intelligence, as we can observe today due to its increasing popularity. As more people become familiar with the concept of artificial intelligence and its applications, there is a growing demand for products and services that incorporate these capabilities. This trend appears to be on the rise in the coming years.

In addition, the personalization of avatars has become a growing trend in the digital world. Avatars are used on social networks, gaming platforms, and online communities, and many people seek to express their identity and creativity through them. The possibility of using an artificial intelligence-based

application to create unique and personalized avatars, by transferring artistic styles from reference images, is attractive to both casual users and digital artists.

There have been several studies that have addressed the importance of using avatars in virtual environments, such as:

1. "Virtual Embodiment: Presence in Virtual Worlds Can Alter Perceptual Experience" (Mel Slater, Maria V. Sanchez-Vives, and Mavi Sanchez-Martin, 2010): This study examined how the representation of a user through an avatar can affect their perceptual experience in virtual environments. The findings demonstrated that users can experience changes in their own bodily perception and sensorimotor abilities when identifying with an avatar.

2. "The Impact of Avatar Personalization and Immersion on Virtual Body Ownership, Presence, and Emotional Response" (Panagiotis Kourtesis, Stavroula-Evita Fotinea, and Dimitrios Tzovaras, 2019): The study explored how avatar personalization and immersion influence virtual body ownership, the sense of presence, and users' emotional response. The results showed that avatar personalization and immersion can enhance the sense of virtual body ownership and presence in virtual environments.

1.2 Objectives

- To learn the functioning of the Neural Style Transfer technology for digital images processing.
- To implement a deep-learning based model on to make the style transfer between two images.
- To develop a program that allows the users to generate their own avatars with customized styles.

1.3. Environment and Initial State

As previously mentioned, the project originated as a proposal from the supervisor, with the article "A Neural Algorithm of Artistic Style" by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge [2] serving as the main point of support for the deep learning model.
“Here we introduce an artificial system based on a Deep Neural Network that creates artistic images of high perceptual quality. The system uses neural representations to separate and recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images. Moreover, in light of the striking similarities between performance-optimised artificial neural

networks and biological vision, our work offers a path forward to an algorithmic understanding of how humans create and perceive artistic imagery. “

The style transfer approach is a technique that involves using pre-trained Convolutional Neural Networks (CNNs) [3] to extract high-level features from target images. These features are then combined iteratively to generate a new image that combines the style of one image with the content of another.CNNs approach

In addition to CNNs, there are other alternatives in style transfer, most of them based on CNNs, such as Generative Adversarial Networks (GANs)[4]. GANs consist of a generator network that generates new images and a discriminator network that tries to distinguish between the generated images and real images. Through an adversarial training process, the generator network learns to produce images that closely resemble the style of the input image while maintaining the content of the target image.

Instance normalization is another technique used in style transfer to enhance control over the style and content of the generated image. It normalizes the features at each spatial location independently, which helps to preserve the global style information while allowing for local variations.

Efficiency and speed improvements have also been made in style transfer algorithms to handle high-resolution images more efficiently. Various optimization techniques and network architectures have been proposed to reduce the computational complexity and memory requirements, enabling faster processing of large images.

Furthermore, style transfer has been extended to other modalities beyond images. For example, there have been attempts to transfer artistic styles to videos, where each frame is processed independently or through temporal coherence constraints. Style transfer has also been applied to other domains such as music and text, enabling the generation of music with a specific style or transforming the style of written text.

Overall, style transfer techniques continue to evolve, incorporating advancements in neural network architectures, optimization algorithms, and extensions to different modalities, opening up possibilities for creative applications in various fields.

2. PLANNING AND RESOURCES EVALUATION

Contents

At this stage, we will present and outline the project objectives, as well as assess their feasibility. We will organize the tasks and provide a time estimation for their completion. Additionally, we will offer an estimation of the economic costs involved in the project, broken down into hardware, software, working hours, and indirect costs.

2.1. Planning

Below are the accomplishments achieved during the project's development, demonstrating the various tasks completed. It is important to note that the tasks were not strictly executed in a linear fashion, with one task being completed before moving on to the next. Rather, there was a dynamic approach that involved interweaving and alternating between different tasks. To provide a visual representation of the project's progress, a Gantt chart has been included in this section (refer to Figure 1).

- **Studying and learning about Neural Style Transfer technology (30 hours):** Search for information on various elements related to artificial intelligence, but especially on Convolutional Neural Networks and Style Transfer Techniques, by reading articles [5] and watching videos from people with extensive knowledge on the subject.

- **Alternative's analysis and choice of the deep learning model (30 hours):** Analyze different alternatives and select the most suitable deep learning model for us. Various options are evaluated and a comparative analysis is carried out to determine which model best fits the requirements and objectives of the project. The goal is to choose the option that has the best performance and ability to fulfill the specific tasks of avatar personalization. Furthermore, the feasibility of the proposed tools and their training should also be considered in this phase.

- **Implementation of the chosen model (80 hours):** The concepts and algorithms of the model are translated into programming code, following the established guidelines and specifications. The necessary components are developed, the appropriate parameters are configured, and the model is integrated into the avatar personalization

application. This stage is crucial to ensure that the selected model works correctly and meets the requirements of the project.

- **Development of a data set for the avatar's generation (10 hours):** Various data sources, such as images, illustrations, and relevant artistic styles for avatar personalization, are collected and prepared.

- **Model training and analysis for the digital avatar personalization (30 hours):** The previously collected and prepared data is used to evaluate the artificial intelligence model. During the training process, the model learns to generate personalized avatars using deep learning and style transfer techniques. Once the training is completed, a thorough analysis of the model's performance is carried out. Metrics such as the quality of personalization, visual fidelity, and diversity of generated avatars are evaluated. In addition, adjustments and improvements are made to the model if necessary, with the aim of optimizing its ability to create avatars that meet the needs and preferences of users.

- **User interface development (60 hours):** The visual elements are designed and created to create an intuitive and attractive interface.

- **Documentation and presentation (60 hours):** Preparation of documents such as the Final Degree Work report, the presentation, technical proposal or other necessary documents.

The objective "To learn the functioning of the Neural Style Transfer technology for digital images processing" is primarily accomplished in stage 1: "Studying and learning about Neural Style Transfer technology (30 hours)", and also to a significant extent in stage 2: "Alternative's analysis and choice of the deep learning model (30 hours)". However, this objective continues to be achieved throughout all stages of the project's development.
The objective "To implement a deep-learning based model to perform style transfer between two images" is accomplished in stage 3: "Implementation of the chosen model (80 hours)".

The objective "To develop a program that allows users to generate their own avatars with customized styles" is accomplished in stage : "Development of a dataset for avatar generation (10 hours)", stage 5: "Model training and analysis for digital avatar personalization (30 hours)" and Point 6: "User interface development (60 hours)". In these points, the development of a program that allows users to generate personalized avatars with customized styles is carried out, covering everything from data preparation to the user interface for user interaction.

When it comes to completing tasks, a self-planned approach has been followed, breaking them down into subtasks. Notes have been taken along the way, facilitating small-scale progress and a gradual transition towards larger-scale advancements. Several follow-up meetings were held periodically with the project supervisor, seeking recommendations on the preferred direction or potential implementation of changes.

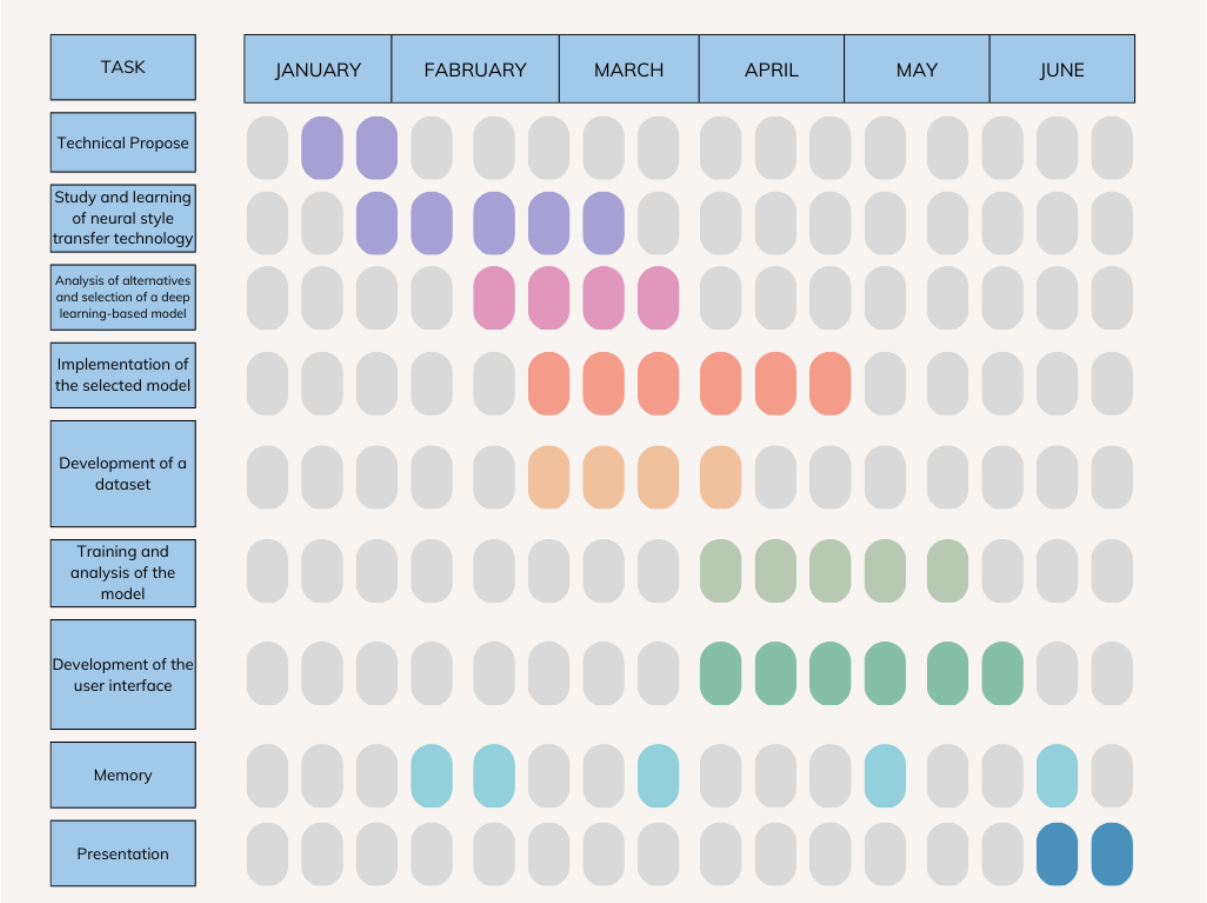

Figure 1: Gantt Diagram of the memory.

2.2. Resources Evaluation

Next, an assessment is made of the project resources, divided into hardware, software, and human resources.

2.2.1 Hardware Resources

The plan was to carry out the entire development using a PC capable of running the Neural Network (NN) model (for its integration with the application). Additionally, it is necessary for the device to have a webcam for taking photos.
These are the minimum requirements to run the application:

- Processor: Intel Core i5 or equivalent
- Graphics Card: NVIDIA GTX 1050 or a similar graphics card
- RAM: 8GB or higher

- Storage: Solid State Drive (SSD) with at least 256GB capacity
- Operating System: Windows 10 or superior
- Webcam: Built-in or external webcam for photo capture
- Internet Connectivity: Wired or wireless network adapter for internet access
- Software: Compatible software tools and libraries required for implementing the NN model and application development.

In the development of this type of projects its important to have a good Graphics Processing Unit (GPU). GPUs are essential for computing neural networks due to their capability for massive parallelism, specialized architecture, fast memory, and the availability of optimized libraries and frameworks. Their use enables accelerated training time and inference of neural networks, driving the advancement and application of machine learning techniques and deep neural networks in various fields.

2.2.2 Software Resources

In terms of software, we will utilize Google Colab [6] to compose and execute Neural network model proposal implementation (being Python most relevant programming language in this regard), enabling us not only to develop and train neural network model proposal, but also to organize several performance and utility test over it. This software has a free version, with limited resources as computation time, use or GPU etc.. Also there are some purchase plans that remove these limitations.

For project purposes free is considered to provide enough resources to NN model developing and testing

Additionally, auxiliary libraries are needed to support our deep learning and matrix handling objectives, including:

- Keras [7] is a popular open-source deep learning framework designed to enable fast experimentation with deep neural networks. It offers a user-friendly interface for building and training deep neural networks using pre-built layers and an easy-to-use API. Keras can run on top of various backends, including TensorFlow, Theano, and CNTK, and provides a range of built-in loss functions, optimizers, and metrics to optimize the network during training.

- NumPy is an open-source Python library used for scientific computing and data analysis. It offers tools for working with multi-dimensional arrays and matrices, as well as a wide range of mathematical functions to operate on these arrays.

- Matplotlib is an open-source Python library for creating static, animated, and interactive visualizations. It provides a broad range of tools for creating plots and charts, such as line plots, scatter plots, bar plots, histograms, and more. Matplotlib allows users to customize every aspect of their visualizations, including colors, labels, fonts, and other properties.

Additionally, various image processing libraries have been used, such as OpenCV (cv) and Pillow:

- OpenCV is a widely-used library in computer vision that offers a wide range of functions and algorithms for image processing and analysis.

- Pillow is a Python image processing library that provides functions for opening, manipulating, and saving images in different formats. These libraries have been instrumental in performing various image-related tasks in the project.

In terms of the graphical user interface, the selected technology will be Tkinter, which is the default GUI library in Python, because of its ease of use in creating simple applications. Additionally, an Anaconda [8] environment has been used to make use of all the necessary libraries, and Spyder has been utilized to implement the model code in the application.

Finally, online communication with the supervisor has been conducted via Gmail and Google Meet for tracking purposes. These platforms have provided a convenient and efficient means of exchanging updates, discussing project progress, and seeking guidance. Gmail has facilitated regular email communication, allowing for detailed discussions and sharing of documents, while Google Meet has been utilized for virtual meetings and video conferences, enabling real-time discussions and screen sharing. This online communication has ensured a smooth and effective collaboration with the supervisor, allowing for timely feedback, addressing any concerns, and ensuring the project stays on track.

2.2.3 Human Resources

Determining the cost per hour of hiring a person to perform each of the above tasks can vary depending on multiple factors, such as geographic location, the level of experience of the professional, the type of contract, among others. First, we will estimate the cost of developing and implementing the style transfer model based on the cost of hiring a Deep Learning professional [9], and then also add the estimated cost for a Python application developer [10], as follows in the Chart 1:

Service	Duration	Price/Hour	Total Cost
Development and implementation of the model	150 hours	40 €	6000 €
User interface development	60 hours	20 €	1200 €

Chart 1: Costs of the human resources.

3. SYSTEM ANALYSIS AND DESIGN

Contents

This proposal outlines the development of an application that utilizes artificial neural networks for neural style transfer. It includes a discussion on the theoretical framework of both artificial neural networks and Convolutional Neural Network, along with a proposed neural model for style transfer as essential foundations to understand the implemented model.

The proposal also includes functional and nonfunctional requirements, as well as system design. In this section, a functional definition and UML diagrams have been included to provide a clear overview of the system. Additionally, the system architecture will be described, along with the minimum system requirements necessary to execute the application flawlessly on a PC.

3.1 Theoretical Framework

Due to the complexity of the project, it needs a prior theoretical context to understand its development, so before starting with the design and architecture of the project, a theoretical framework must be established. It includes a discussion of both, artificial neural networks and convolutional neural networks, along with a proposed neural model for style transfer.

Then, the focus of the analysis will be on describing and examining the functional and non-functional requirements that underpin the application.

3.1.1 Artificial Neural Networks

An Artificial Neural Network is a computational model inspired by the structure and function of biological neurons operate inside the brain [11]. It is made up of interconnected nodes or neurons, which process and transmit information through the network.

An artificial neuron consists of three main components: inputs, weights, and an activation function. The inputs represent the signals or input values that the neuron receives from other neurons or the environment. Each input is associated with a weight, which determines the importance or relative impact of that input on the neuron's output.
The activation function is a mathematical function that takes the weighted sum of the inputs multiplied by their respective weights and transforms it into an output. This function introduces non-linearities and defines how the neuron responds and generates an output based on the received inputs.
The output of an artificial neuron can be used as input for other neurons in the network, thus forming a network of connections that allows for parallel information processing and the execution of complex tasks such as pattern recognition, data classification, or decision-making.

These artificial neurons are organized into layers inside the network, with each layer responsible for a specific type of computation. The input layer receives data from the outside world, while the output layer produces the final result. In between the input and output layers, there can be multiple hidden layers that perform complex computations on the input data.

The process by which an artificial neural network learns from a dataset is called training. During the training process, the network adjusts its weights and biases based on the input data and desired output, allowing it to learn and improve its performance. Once the network has been trained, it can be used for a variety of applications, such as image recognition, natural language processing, and prediction.

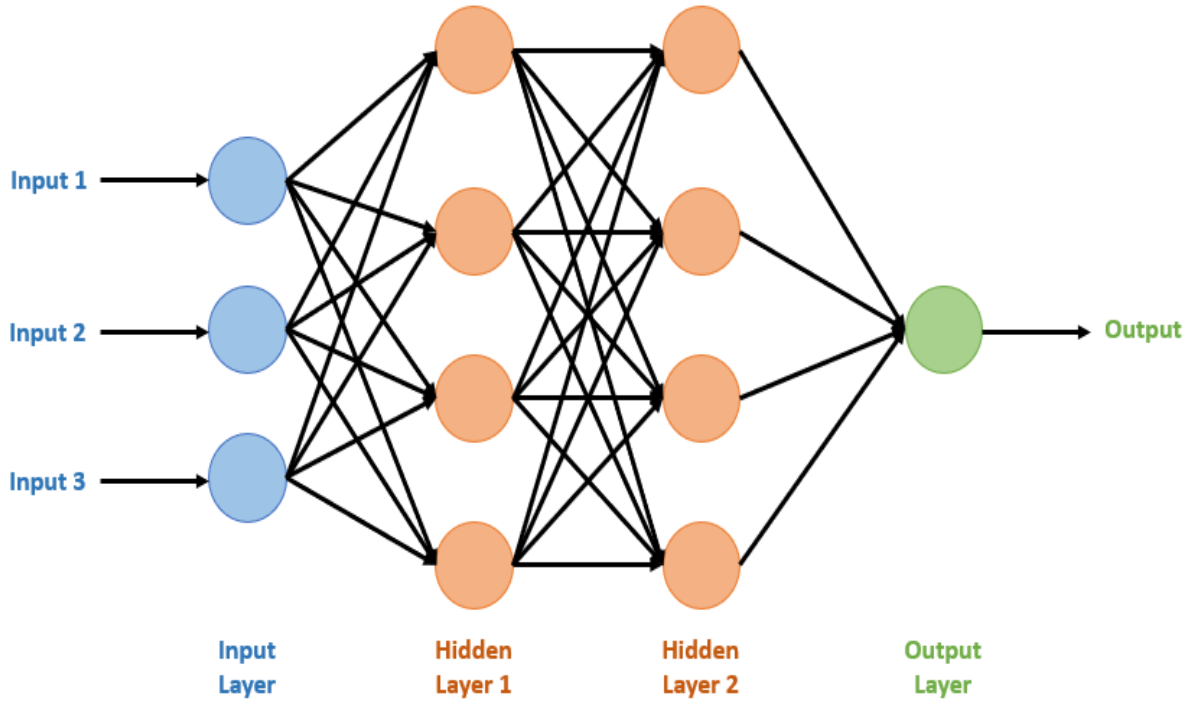

Figure 2: An Artificial Neural Network schema.

The input layer receives the input data, which is then processed by one or more hidden layers. Each hidden layer consists of multiple neurons, each of which performs a computation on the input data. The output of one hidden layer is then passed on to the next hidden layer until the final hidden layer is reached.

The output layer produces the final output of the network. The number of neurons in the output layer depends on the type of problem being solved. For example, for a binary classification problem, the output layer would consist of a single neuron that outputs a value between 0 and 1 indicating the probability of a positive outcome.

During training, the weights and biases of each neuron in the network are adjusted based on the input data and desired output, allowing the network to learn and improve its performance over time. Once the network has been trained, it can be used to make predictions on new data.

3.1.2. Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of neural network that is designed specifically for image and video recognition. It uses a process called convolution [12], where the network learns filters to detect specific features in the input data.

Convolution is a mathematical operation that involves applying a filter or kernel to the input data in order to extract specific features. The filter is a small matrix of weights that slides or convolves across the input, performing element-wise multiplication and summation at each position. This process allows the network to learn and detect patterns or features in the input data, such as edges, textures, or shapes. By performing convolutions at multiple layers of the network, CNNs can progressively learn more complex and abstract features, enabling them to recognize patterns and make predictions in image and video data.

A basic schematic of a CNN architecture would typically include the following layers:

- Input layer: The layer that receives the raw image data.
- Convolutional layer: Applies a set of filters to the input image, producing a feature map.
- Pooling layer: Reduces the spatial dimensions of the feature map, while retaining the most important information.
- Activation layer: Introduces non-linearity to the network, allowing it to learn complex relationships between the inputs and outputs.
- Fully-connected layer: Connects every neuron in one layer to every neuron in the next layer, producing the final output of the network.
- Output layer: The final layer that provides the network's prediction or decision.

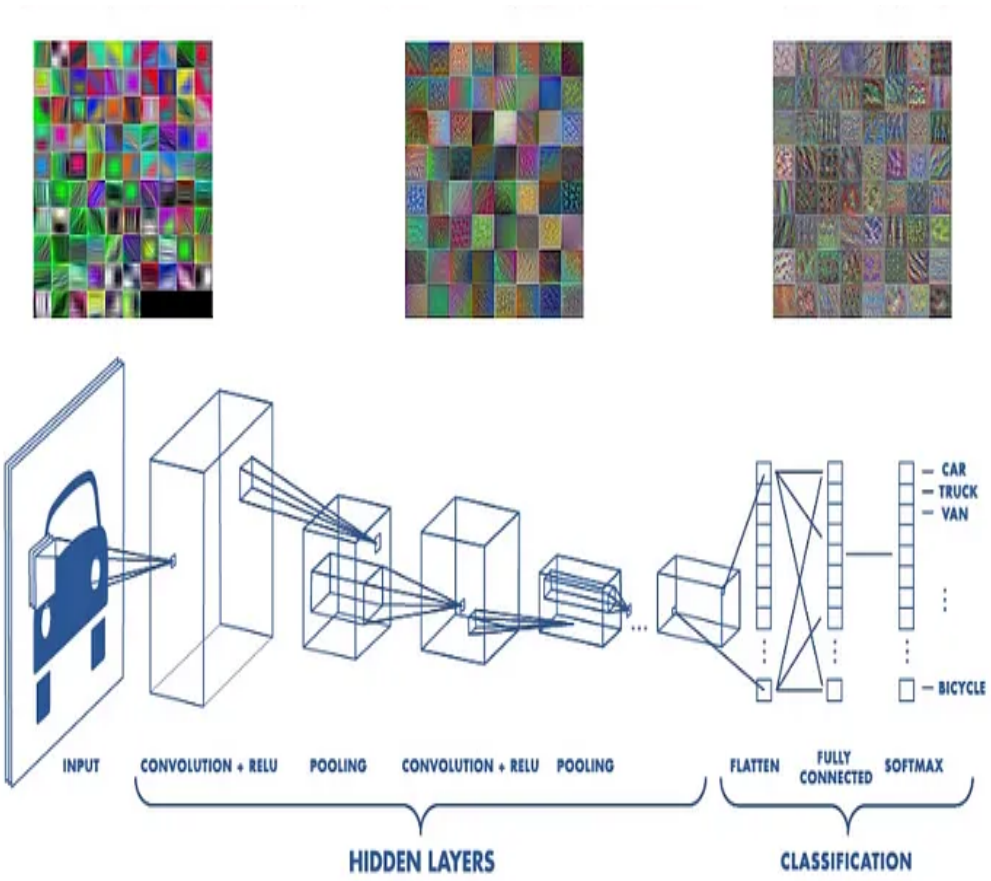

Figure 3: A Convolutional Neural Network schema.

Note that this is just a basic schematic, and actual CNN architectures can be much more complex and can include multiple convolutional, pooling, and fully-connected layers, as well as different types of activation functions and regularization techniques.

3.1.3 ReLu

"ReLu" [13] is an abbreviation for "Rectified Linear Unit," which refers to an activation function utilized in neural networks. The ReLu function can be defined by the mathematical expression:

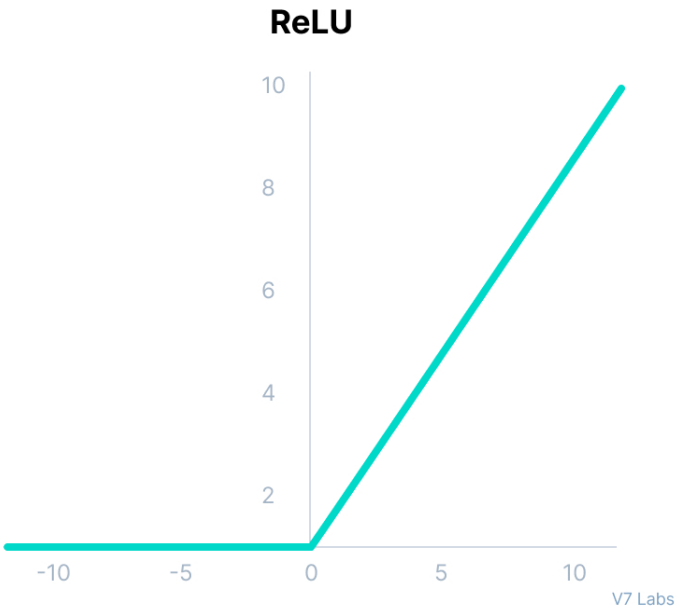

Figure 4: ReLu activation function.

$$ReLU$$

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

Figure 5: ReLu mathematical representation.

In simple terms, the output of the function equals the input value if it is greater than or equal to zero, while it is zero otherwise.

The ReLu activation function has gained popularity in neural network architectures due to its simplicity, computational efficiency, and non-linear characteristics. It permits the network to capture non-linear relationships between inputs and outputs, making it a useful tool for recognizing patterns in complex data.

In a neural network, the ReLu activation function is applied during the forward pass to the output of either a fully connected or a convolutional layer. This action allows the network to introduce non-linearity, enabling it to learn and identify more intricate patterns in the data.

3.1.4 Neural Model Proposal

Neural Style Transfer is a technique that uses a neural network to apply the style of one image to the content of another image. The general structure of a Neural Style Transfer network involves the following steps:

- Input images: The network takes as input two images: a content image and a style image. These images need to be preprocessed to meet the requirements of the CNN.

- Feature extraction: Both the content image and the style image are processed by a pre-trained Convolutional Neural Network, and the activations of certain layers are extracted. These activations capture high-level information about the content and style of the images.

- Content reconstruction: The activations of the content image are used to reconstruct the original image. This is done by minimizing the difference between the activations of the content image and the activations of a generated image.

- Style reconstruction: The activations of the style image are used to define a style representation, which is used to generate a new image that has the same style as the style image. This is done by minimizing the difference between the activations of the style representation and the activations of the generated image.

- Output image: The output image is generated by combining the content of the content image with the style of the style image. This is done by applying the activations of the content reconstruction to the style reconstruction. Finally, the generated image undergoes post-processing to restore it to its original format.

The result is a new image that contains the content of the content image, but with the style of the style image applied. This process can be repeated with different content and style images to produce a variety of artistic effects.

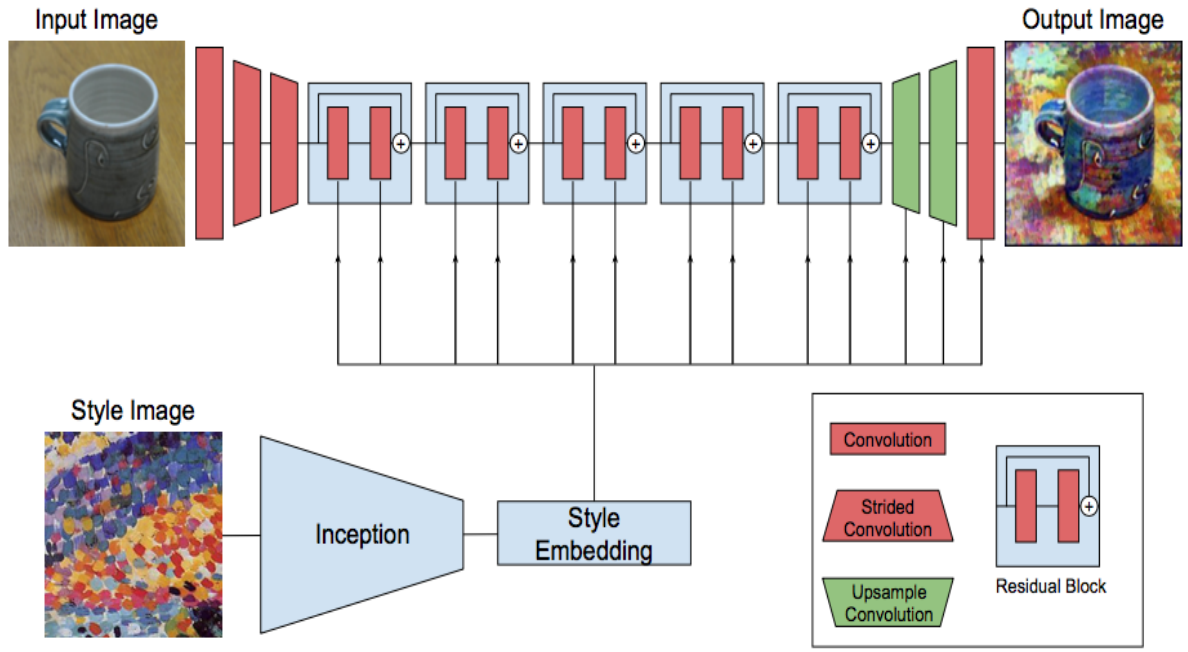

Figure 6: Neural Style Transfer Schema and example.

Finally, the optimization process is performed iteratively, requiring a number of iterations to achieve realistic style transfer.

Regarding the pre-trained CNN, a popular choice is the VGGNet [14], which is important for style transfer due to its ability to learn visual features at different levels of abstraction, allowing it to capture both content and style of images. Its deep and uniform structure, combined with its training on large datasets, is effective for feature extraction in style transfer. Next, we will see two versions of the VGGNet considered for implementation.

VGG16 [15] is a convolutional neural network architecture that was developed by the Visual Geometry Group at Oxford University. It is commonly used for image classification and other computer vision tasks.
VGG19 [16] is a deeper version of VGG16, with 19 layers instead of 16. It has additional convolutional layers that make it more powerful, but also more computationally expensive to train and use.

Finally, VGG19 is selected as it is the most complete one.
VGG19 is also developed by the Visual Geometry Group (VGG) at the University of Oxford. It is a variant of the VGGNet architecture and is characterized by its use of small, stacked convolutional layers.

The architecture of VGG19 consists of 19 layers, including 16 convolutional layers and 3 fully connected layers. The convolutional layers are responsible for learning hierarchical representations of the input image, while the fully connected layers perform the final classification.
VGG19 uses 3x3 convolutional filters and max pooling layers to reduce the spatial dimensions of the feature maps, and 2x2 pooling layers are used to further reduce the size.

The architecture also includes Rectified Linear Unit (ReLU) activation functions, which introduce non-linearities into the network and improve the network's ability to model complex relationships between inputs and outputs.

VGG19 has been trained on the ImageNet dataset [17], a large collection of labeled images, and has achieved state-of-the-art results on various image classification benchmarks. It is often used as a pre-trained model for transfer learning, where its pre-trained weights on large image classification datasets are used as the starting point for training on a new, smaller dataset.

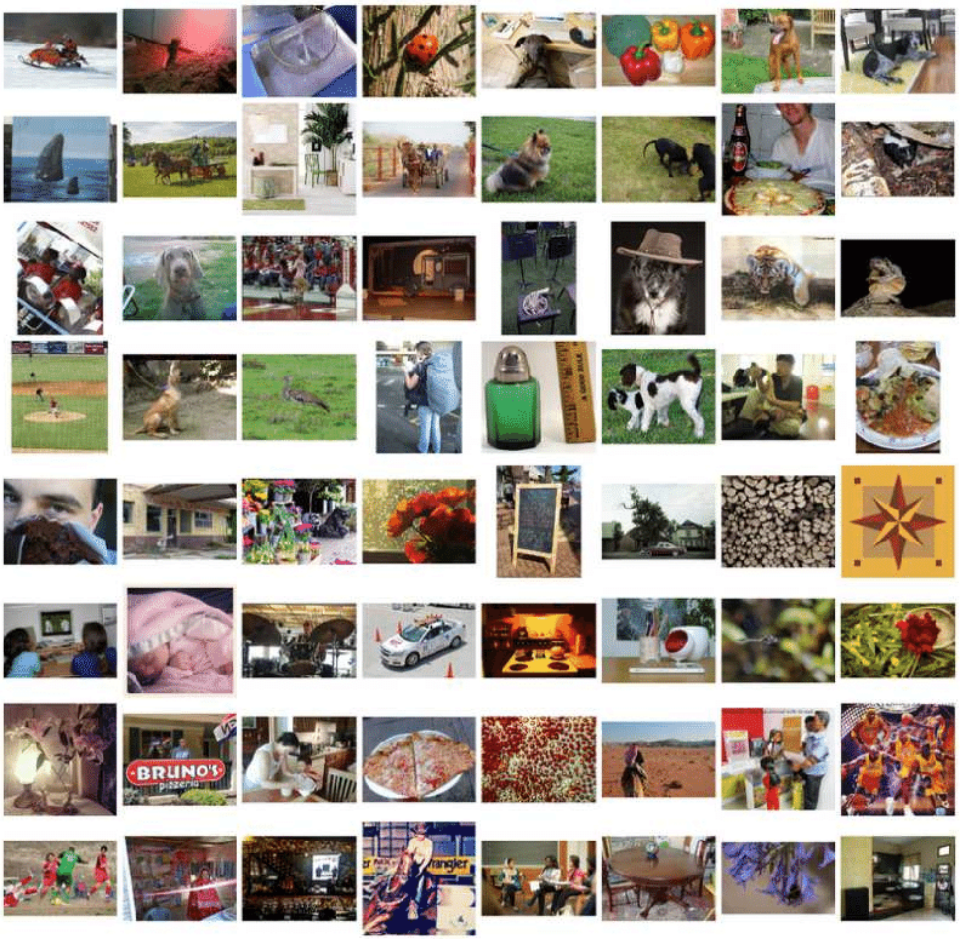

Figure 7: ImageNet Dataset Examples

3.2 Requirements Analysis

In this section, the functional and non-functional requirements of the application are listed, which are taken into account in the next section to establish its design .

3.2.1. Functional Requirements

• **R1:** the user can start the application.

• **R2:** the user can select an artistic style.

• **R3:** the user can upload an image.

• **R4:** the user can take a picture.

• **R5:** the user can quit the application.

• **R6:** the user can crop the image.

• **R7:** the user can choose the quality of the output image.

• **R8:** the system will be able to generate a stylized image.

• **R9:** the player can download the output image.

3.2.2 Non-Functional Requirements

Non-functional requirements impose conditions on the design or implementation. In this
project, the non-functional requirements are:

• **R10:** the system will be able to access the user's device files to obtain an image.

• **R11:** the system will be able to access the user's device files to save an image.

• **R12:** the system will take around one minute to make the style transfer.

• **R13:** the application will be simple and intuitive.

3.3 System Design

Next, in this section, the system design is presented based on the functional requirements, followed by a use case diagram (Figure 8), a sequential diagram (Figure 9) and a UML Classes Diagram (Figure 10).

Requirements	R1
Actor	User
Description	The user starts the application by opening it
Preconditions	
Normal sequence	1. The user open the application 2. The system loads the main screen
Alternative sequence	None

Chart (2.1) Case of Use

Requirements	R2
Actor	User
Description	The user chooses between the 3 available artistic styles
Preconditions	1. The user must be in the main menu
Normal sequence	1. The user press the model selector button 2. The system shows the available models to choose from
Alternative sequence	None

Chart (2.2) Case of Use

Requirements	R3
Actor	User
Description	The user can choose an image from their device to upload
Preconditions	1. The user must be in the main menu
Normal sequence	1. The user press the button *Upload* 2. The system displays the images from the user's archive
Alternative sequence	None

Chart (2.3) Case of Use

Requirements	R4
Actor	User
Description	The user can take a photo with their device
Preconditions	1. The user must be in the main menu
Normal sequence	1. The user press the button *Take Picture* 2. The system open the camera of the device
Alternative sequence	2.1 The system does not open the camera of the device because it is not available

Chart (2.4) Case of Use

Requirements	R5
Actor	User
Description	The user can quit the application
Preconditions	1. The user must be in the main menu
Normal sequence	1. The user press the button *Exit* 2. The system quit the application
Alternative sequence	None

Chart (2.5) Case of Use

Requirements	R6
Actor	User
Description	The user can crop the uploaded or taken image
Preconditions	1. The user must be in the main menu 2. The user must have uploaded or taken an image
Normal sequence	1. The user press the button *Crop* 2. The system displays the current image in a pop-up window 3. The user chooses the area to be cropped. 4. The user confirm the crop
Alternative sequence	None

Chart (2.6) Case of Use

Requirements	R7
Actor	User
Description	The user can choose the quality of the output image using a slider
Preconditions	1. The user must be in the main menu
Normal sequence	1. The user slides the slider. 2. The system change the quality of the output image
Alternative sequence	None

Chart (2.7) Case of Use

Requirements	R8
Actor	User
Description	The system perform the stylization of the selected image
Preconditions	1. The user must have uploaded or taken an image
Normal sequence	1. The user press the button *Confirm* 2. The system performs the transformation of the image

Alternative sequence	None

Chart (2.8) Case of Use

Requirements	R9
Actor	User
Description	The user can download the stylized image to their device
Preconditions	1. The system must have performed the stylization
Normal sequence	1. The user press the button *Download* 2. The system save the stylized image in the users device.
Alternative sequence	None

Chart (2.9) Case of Use

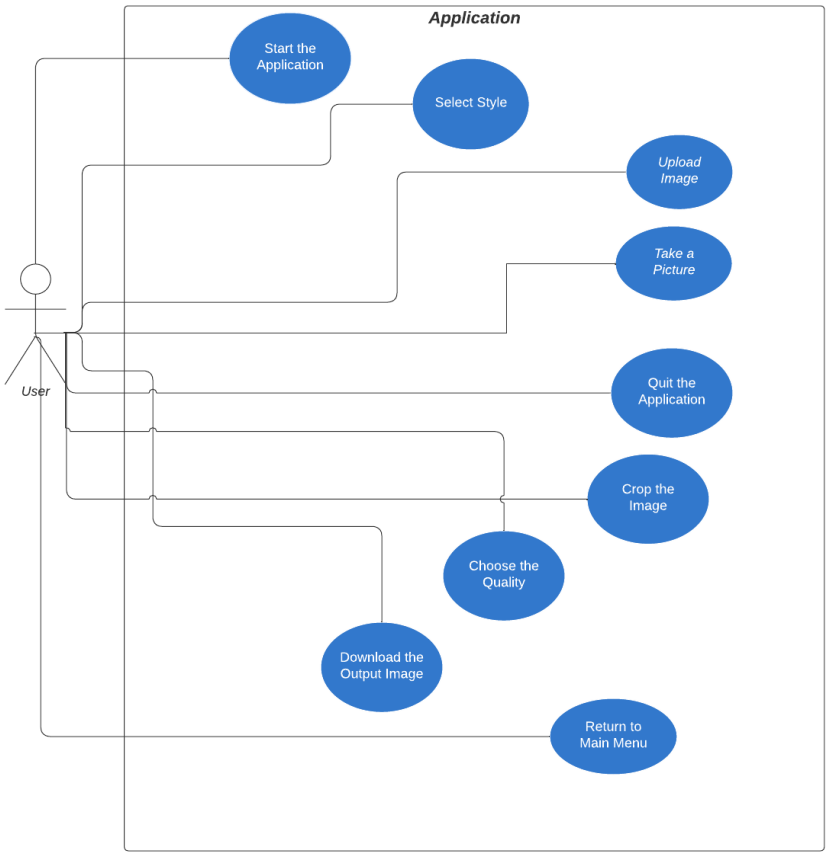

Figure 8: Use Case Diagram

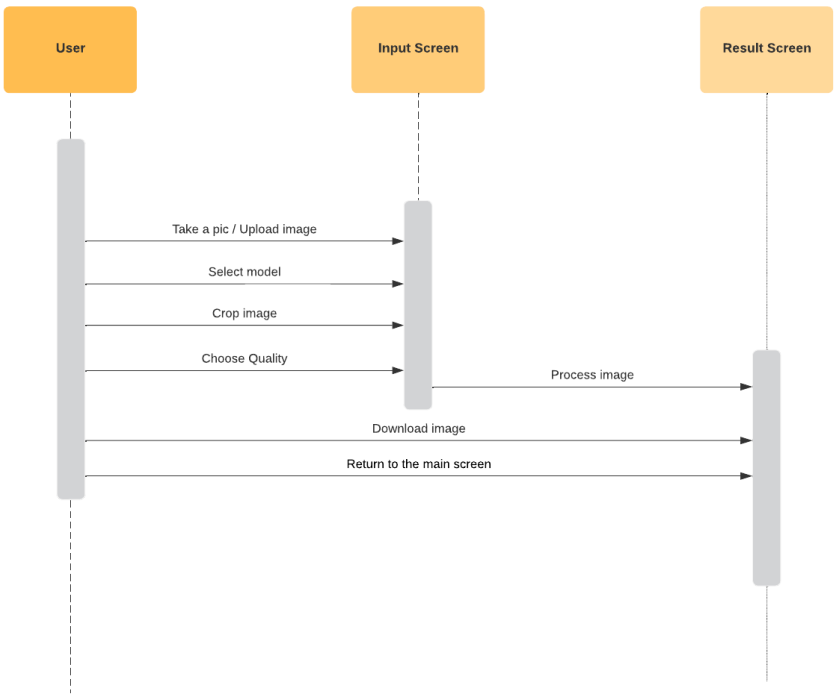

Figure 9: Sequence Diagram

Class Diagram

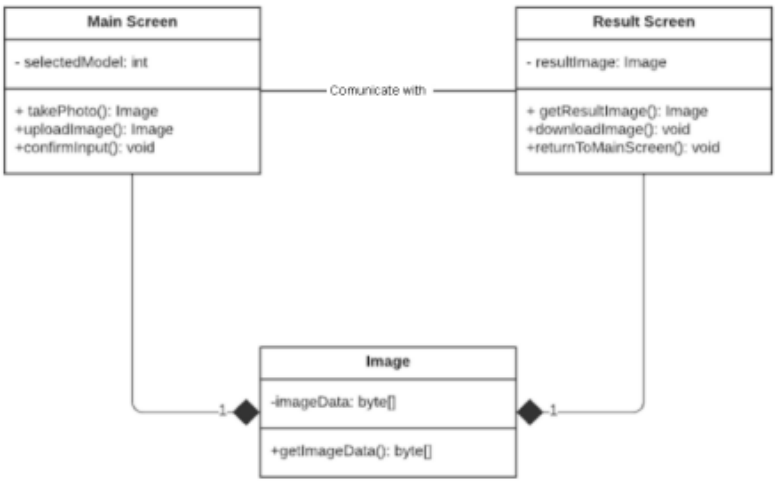

Figure 10: UML Clases Diagram

3.4 System Architecture

The requirements to play the build of this project in a PC are:

- OS: Windows 10 or Higher (home and professional editions)
- CPU: Intel Core i3/ AMD Ryzen 3
- RAM: 4 GB
- GPU: CUDA 3.5 compatible or Higher [18]
- STO: At least 100 MB of available hard disk space.

3.5 Interface Design

This description details the implementation of a style transfer application that allows users to generate artistic images from an input image or by capturing a real-time photograph. This application is based on a neural network model that combines the style of a user-selected reference image with the content of an input image to create a new output image with a unique and personalized aesthetic.

The user interface of the application has been designed considering the principles of usability and visual aesthetics. An intuitive and user-friendly approach has been employed to ensure an optimal user experience. The following are the mock-ups of the interface:

Figure 11: Tkinter Application.

The implementation process of the project was carried out in an organized and comprehensive manner, following the planned stages. The following describes each of these stages:

- Data Acquisition: Reference images of various artistic styles were collected to train the style transfer model. These images were properly cited and included in the appendices.

- Data Preprocessing: The reference images were processed to extract styles and content. Techniques such as image processing and feature extraction were used to obtain suitable representations for the neural network model.

- Model Training: The style transfer model was implemented using a convolutional neural network architecture. The model parameters were adjusted, and an iterative training process was conducted until satisfactory results were achieved.

- Integration with the User Interface: The trained model was integrated with the application's user interface. The necessary functionalities were developed to allow users to select a reference image, upload an input image, or capture a real-time photograph, and generate the output image with the transferred style.

During the implementation process, some problems arose that required additional solutions. These problems included optimizing the model's performance on mobile devices, managing memory to avoid overloading, and improving the visual quality of the generated images. Adjustments were made in the implementation and optimization techniques were applied to address these issues.

Throughout the project, the following milestones were achieved:

- Successful implementation of the style transfer model.
- Effective integration of the model with the user interface.
- Generation of output images with accurately transferred styles and visually appealing results.
- Optimization of the application's performance on different platforms and devices.

The implementation of the style transfer application has been successful, allowing users to generate personalized artistic images from an input image or real-time photograph. The proposed objectives have been achieved, and the challenges encountered have been overcome.

4. WORK DEVELOPMENT AND RESULTS

Contents

In the field of project development, it is essential to understand the progress and outcomes achieved. In this part of the work, the objectives accomplished at each stage of the development will be examined, and the potential adjustments and modifications made throughout the project will be highlighted. This analysis will provide a comprehensive view of how the project has evolved from its initial conception to the final results obtained.

4.1. Work Development

This section of the project will be a description of how the main objectives have been achieved, the problems encountered, the way they have been resolved, and the modifications that the work has had to undergo compared to the initial planning in order to conclude it.

Before starting with the development of the application, other preliminary tasks had to be carried out since the development of this application requires prior learning, both about neural networks and the functioning of Google Colab.

After reading and studying various texts and videos related to the topic, an in-depth exploration was conducted into the functioning of Google Colab. After this, it is decided to depart from Gatys' article, which became evident during testing that it has a limitation in providing fast results in few iterations, which means a longer time. Therefore, an existing implementation extracted from Keras was used as a template. Its worth to consider that these models are truly complex, many of them being continuously that these models are truly complex, many of them being continuously revised by researchers.

After evaluating all possible options, pre-trained CNN was the option that better suit to reach specific requirements and desired constraints according project planning specifically vgg19 model, which stands as the most comprehensive Convolutional Neural Network model currently available. In the implementation of Neural Style Transfer used as template [19], multiple layers are used to perform the necessary operations. These layers are designed to extract features from a reference style image and apply those features to a target content image.

Without going into too much technical detail, the general process involves the following stages:

Load the CNN model: A pre-trained model, such as VGG16 or VGG19, which has been trained on a large amount of visual recognition data, is used. These models have a deep architecture with many convolutional layers and serve as a starting point.

Build the Style Transfer Model: Specific layers from the base model are taken and a new model is constructed that retains those layers. These selected layers act as feature extractors.

Define Loss functions: Loss functions are defined to capture the difference between the reference style image and the target content image. These loss functions include a content loss and a style loss, which represent the similarity in content and style respectively.

Optimize the target image: An optimization algorithm is used to adjust the target content image and minimize the previously defined Loss functions. This involves iterating the image update process multiple times to obtain a better approximation of the desired style. It is important to note that while the Keras implementation provides a solid foundation for the Neural Style Transfer process, it can also be customized and improved according to specific project needs.

Next step was to choose an artistic style that would work well with our model, and this is where we encountered the first problem. The main idea was to create an application that, in addition to stylizing images with classic artistic styles, could also do the same with artistic styles from video games.

Tests were carried out using images sourced from diverse video games- However, they lacked the desired distinctiveness. Consequently, games were selected based on their notable visual aesthetics rather than their realism, such as "Gris" [20] and "Journey" [21], alongside another game recognized for its unique style, namely "Hotline Miami" [22].

Here are some examples of the outputs with different styles:

Figure 12:Examples of close images with “Hotline Miami” and “Journey” style.

Figure 13: Example of landscape with Hotline Miami style.

As is evident from the given information, there are cases where the behavior with wide images, such as the landscape of Paris, doesn't have any negative issues with the styles. However, when it comes to closer images like a person or a pet, the output is too poor to be used. the project's main objective, since the purpose of the application is to use it for avatars, these outputs have to be discarded since most avatar images are of people, characters, or animals.

Considering this, tests were conducted with more famous and distinctive artistic styles, and good results were obtained. Some of these styles were chosen as the final ones, such as "The Scream" by Edvard Munch (1893), "Les Demoiselles d'Avignon" by Pablo Picasso (1907) and "Harlequin's Carnival" by Joan Miró (1924).

Next, the modification and testing stage of the models with the chosen styles begins in order to obtain the best possible outputs. Different changes are included, for example, by using different layers of the VGG19 model to achieve different results depending on the artistic style. For instance:

To perform style transfer using "The Scream" style, various convolutional layers can be utilized to capture different levels of features and styles, such as the layers:

- Convolutional Layer: block1_conv1
- Convolutional Layer: block2_conv1
- Convolutional Layer: block3_conv1
- Convolutional Layer: block4_conv1
- Convolutional Layer: block5_conv1

These layers represent different levels of abstraction in the network and capture more detailed features as they progress to higher layers. By combining the style representations extracted from these layers with the content features of an input image, style transfer can be achieved, reflecting "The Scream's" distinctive style in the target image.

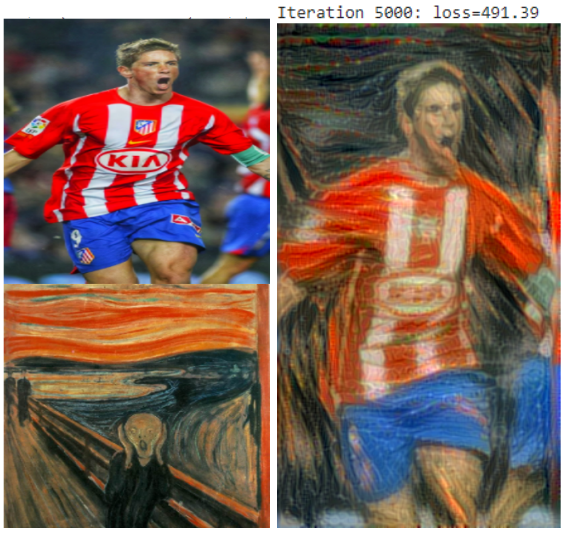

Figure 14: Example of The Scream transfer style.

Another example is:

- Convolutional Layer: block1_conv1
- Convolutional Layer: block2_conv2
- Convolutional Layer: block3_conv3
- Convolutional Layer: block4_conv4
- Convolutional Layer: block5_conv4

These layers also represent different levels of abstraction in the network and capture more detailed features as you go deeper into the higher layers. By combining the style representations extracted from these layers with the content features of an input image, style transfer can be achieved, reflecting the distinctive artistic style of "Demoiselles d'Avignon" in the target image.

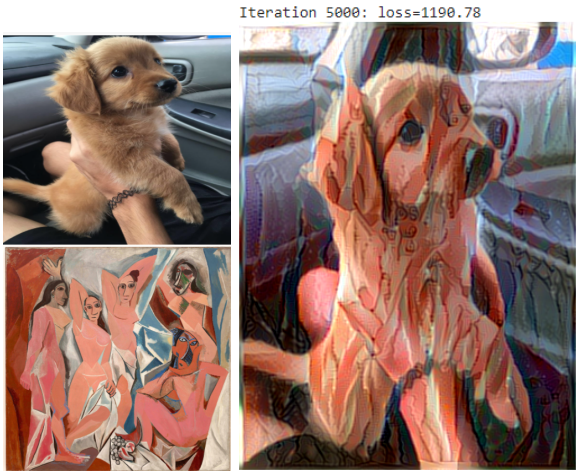

Figure 15: Example of Demoiselles d'Avignon transfer style.

Once tuning over model was completed, next step was developing this model as application together with user interface, as it have to execute in a personal computer,installing Deep learning libraries along with python was necessary, migrate to Anaconda environment along with Spyder, as they were

necessary due to the large number of libraries needed for the application development. Similarly, the main challenge encountered in the app development, which has taken the most time, has been the multiple errors when installing all the libraries since specific versions of each library need to be used to ensure compatibility among them, especially with Python, TensorFlow and cuDNN [23].

Some of these errors were easily solved, especially thanks to internet searches and solutions provided by users on Stack Overflow. Other errors occurred due to compatibility issues between certain libraries that were initially compatible but had problems with a specific library in the end.

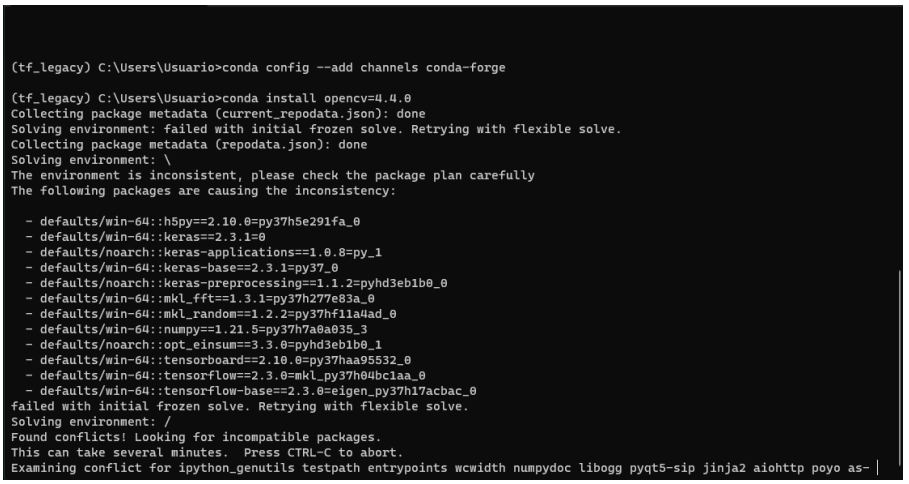

Figure 16: OpenCV incompatibility error.

And finally, some errors occurred where, after installing a library, when trying to use it, an error occurred indicating that it wasn't installed, even though it appeared as installed in the environment. With the help of the tutor and dedicating a significant amount of time, we managed to find solutions to continue with the project.

Once we have started the development of the model and the application, we can now discuss the relationship between both of them. In this case, there is an obvious and significant problem: there is a very high waiting time when transforming one image into another through style transfer. To find a solution to this problem, a significant reduction in image resolution is applied, allowing the app itself to perform preprocessing of the input images. This simple modification significantly reduces the waiting time during the stylization of the images, turning a process that used to take nearly 5 minutes into just around a minute.

To further reduce the image generation time, the use of the AdaIN technique for Style Transfer [24] was considered. AdaIN (Adaptive Instance Normalization) is a technique used in Style Transfer in the field of machine learning. Unlike a model based on VGG19, which focuses on capturing content and style features in different convolutional layers, AdaIN focuses on Adaptive Instance Normalization. In the style transfer process, AdaIN takes a content image and a style image and adjusts the instance normalization of the content image to statistically resemble the style image. This is achieved by adjusting the mean and standard deviation of the features in the convolutional layers, allowing the content image to acquire stylistic characteristics similar to the style image.

Thanks to this, when generating a stylized image, AdaIN is faster than VGG19, which could help further reduce the waiting time. However, the problem is that using AdaIN for style transfer requires training the model with a dataset, which takes a significant amount of time. Generally, the convergence time can vary from several days to even several weeks, which is not feasible according to project planning and time spent. Therefore, we ultimately discarded this option and continued with the previously chosen model, VGG19.

4.2. Results

As a result, after the project development, all the initial objectives proposed have been achieved, although there have been some modifications regarding the initial idea of the app. Initially, it was supposed to be a simple application where the user would choose an image or take a photo and perform style transfer with the chosen style. To these functions, the option for the user to crop images, choose from multiple models, and select higher or lower quality of the resulting images has been added.

The following is a table showing the comparison between the expected hours for each task and the actual time needed:

TASK	Expected Time	Actual Time
Studying and learning about Neural Style Transfer technology	30 hours	30 hours
Alternative's analysis and choice of the deep learning model	30 hours	30 hours
Implementation of the chosen model	80 hours	90 hours
Development of a data set for the avatar's generation	10 hours	10 hours
Model training and analysis for the digital avatar personalization	30 hours	20 hours
User interface development	60 hours	70 hours
Documentation and presentation	60 hours	50 hours

Chart 3: Tracking Table for Planned Hours and Actual Hours Used.

As observed in the table, there has been a variation in the hours used compared to the ones initially planned. This variation is attributed to the issues encountered during the development process, as previously mentioned. As a result, it was necessary to allocate more hours than expected for the application development, both in terms of implementing the model and developing the interface and app functions.

The final result of this project was obtaining transformed images through style transfer using convolutional networks, so to conclude this section, we are going to analyze the final images obtained.

To perform this analysis, we will take into account the perceptual loss.

Perceptual loss is a metric that quantifies the difference between the generated image and the reference image in terms of perceptual features. It is calculated using pre-trained neural network models, such as VGGNet, by measuring the differences in activations of intermediate layers for the generated image and the reference image. A lower perceptual loss indicates a better match in perceptual features between the images.

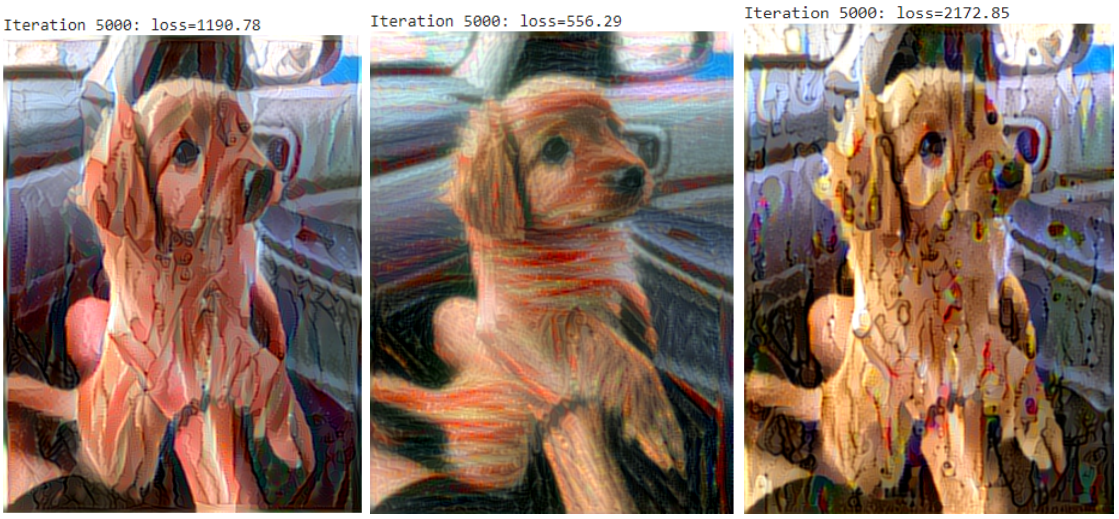

Figure 17: Resulting images with their loss.

In the loss functions of the resulting images (see Figure 16), different losses are obtained depending on the model. Therefore, subjective evaluation should also be taken into account when assessing the results, as it is important to note that no metric is perfect, and there may be discrepancies between objective metrics and human perception. Therefore, it is recommended to use a combination of objective metrics and subjective evaluation to obtain a more comprehensive and accurate assessment of style transfer.

5. CONCLUSIONS AND FUTURE WORK

Contents

5.1. Conclusions

Throughout my degree, we have developed various video games or applications, but we have never had to undertake a project of this nature. In this Bachelor's Thesis, an exhaustive analysis and development of Style Transfer techniques based on Convolutional Neural Networks have been carried out. As a result of this, I have been able to learn a lot about a field that is in growing development and is increasingly gaining importance, such as the field of artificial intelligence and deep learning. Although this field is briefly touched upon in the degree program, having this prior knowledge has undoubtedly been helpful.

Finally, it is crucial to highlight that recent advances in artificial intelligence are exciting and promising. Artificial Intelligence has enormous potential to positively transform our society, from medicine and education to transportation and industry.

However, we must remember that we are responsible for its proper development and application. Only by addressing these advancements with ethical awareness and a focus on the collective well-being can we fully harness the benefits of artificial intelligence and build a fairer and more equitable future.

5.2. Future Work

In the short term, one of the works that will be done to improve the application and make it more complete is to expand the number of art styles available. This would provide users with a greater variety of options to choose from. In addition, another service that is planned to be implemented in the future is integration with cloud services. For example, users could be allowed to easily store and share their stylised images online.

These enhancements, along with others that could be implemented, aim to create an application that is as complete as possible so that it can be sold, and subsequently incorporated by the relevant team into popular platforms for both PC and mobile devices. From Steam and Xbox to Android and web browsers, the aim is to broaden the reach of the application and reach a wider audience. In this way, users of these platforms themselves will be able to conveniently enjoy the service of our application, allowing them to edit their avatars and give them a distinctive touch in a simple way.

Furthermore, the utilization of AdaIn should be thoroughly examined and further investigated in scenarios where the initial application of style transfer did not yield the desired results.

6. CITED WORKS

[1] Dey, S. (2022, 30 marzo). Neural Style Transfer on Real Time Video (With Full implementable code). Medium. https://towardsdatascience.com/neural-style-transfer-on-real-time-video-with-full-implementable-code-ac2dbc0e9822

[2] "A Neural Algorithm of Artistic Style" by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge **[1508.06576] A Neural Algorithm of Artistic Style (arxiv.org)**

[3] Tharsanee, R. M., Soundariya, R. S., Kumar, A. S., Karthiga, M., & Sountharrajan, S. (2021). Deep Convolutional Neural Network–based image classification for COVID-19 diagnosis. En Data Science for COVID-19 (pp. 117–145). Elsevier. https://www.sciencedirect.com/topics/engineering/convolutional-neural-network

[4] Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F., & Zheng, Y. (2019). Recent Progress on Generative Adversarial Networks (GANs): A Survey. IEEE Access, 7, 36322-36333. https://doi.org/10.1109/access.2019.2905015
Recent Progress on Generative Adversarial Networks (GANs): A Survey | IEEE Journals & Magazine | IEEE Xplore

[6] EdXD. (2022, 6 noviembre). What is Google Colab: A Beginner's Guide - ByteXD. https://bytexd.com/what-is-google-colab-a-beginner-guide/?utm_content=cmp-true

[7] Team, K. (s. f.). Keras: Deep Learning for humans. https://keras.io/

[8] Osterbuhr, T. (2023). What is Anaconda and how does it relate to Python? Venture Lessons. https://www.venturelessons.com/what-is-anaconda/

[9] UpWork Machine Learning Expert artificial intelligence - Upwork

[10] UpWork App Developer app developer - Upwork

[11] Puri, M., Solanki, A., Padawer, T., Tipparaju, S. M., Moreno, W. A., & Pathak, Y. (2016). Introduction to artificial neural network (ANN) as a predictive tool for drug design, discovery, delivery, and disposition. En Artificial Neural Network for Drug Design, Delivery and Disposition (pp. 3–13). Elsevier.

[12] Mishra, M. (2020, agosto 26). Convolutional neural networks, explained. Towards Data Science.
https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[13] A Gentle Introduction to the Rectified Linear Unit (ReLU) - MachineLearningMastery.com. (s.f.). MachineLearningMastery.com. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[14] Chan, K., Im, S., & Ke, W. (2020). VGGreNet: A Light-Weight VGGNet with Reused Convolutional Set. https://doi.org/10.1109/ucc48980.2020.00068

[15] Hassan, M. U. (2023). VGG16 – Convolutional Network for Classification and Detection. Neurohive. https://neurohive.io/en/popular-networks/vgg16/#:~:text=VGG16%20%E2%80%93%20Convolutional%20Network%20for%20Classification%20and%20Detection,%E2%80%9CVery%20Deep%20Convolutional%20Networks%20for%20Large-Scale%20Image%20Recognition%E2%80%9D.

[16] Mascarenhas, S., & Agarwal, M. (2021). A comparison between VGG16, VGG19 and ResNet50 architecture frameworks for Image Classification. https://doi.org/10.1109/centcon52345.2021.9687944

[17] Papers with Code - ImageNet Dataset. (s. f.). https://paperswithcode.com/dataset/imagenet

[18] Minimum requirements https://www.tensorflow.org/install/gpu?hl=es-419

[19] Team, K. (s. f.). Keras documentation: Neural style transfer. https://keras.io/examples/generative/neural_style_transfer/

[20] Home - Nomada Studio. (s. f.). Nomada Studio. https://nomada.studio/

[21] Tgc. (2020, 14 abril). Journey - thatgamecompany. thatgamecompany. https://thatgamecompany.com/journey/

[22] HOTLINE MIAMI. (s. f.). https://hotlinemiami.com/

[23] NVIDIA CUDA Deep Neural Network (cuDNN). (2023, 27 junio). NVIDIA Developer. https://developer.nvidia.com/cudnn

[24] Huang, X., & Belongie, S. (2017). Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization. https://doi.org/10.1109/iccv.2017.167

Appendix

A. Source Code

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import vgg19

base_image_path = keras.utils.get_file("perro.jpg",
"https://i.imgur.com/pwsKT2G.jpeg")
style_reference_image_path = keras.utils.get_file(
    "grito.jpg", "https://i.imgur.com/QxSAVyJ.jpeg"
)
result_prefix = "perro_generated"

# Weights of the different loss components
total_variation_weight = 1e-6
style_weight = 1e-6
content_weight = 2.5e-8

# Dimensions of the generated picture.
width, height =
keras.preprocessing.image.load_img(base_image_path).size
img_nrows = 256
#img_ncols = 256

img_ncols = int(width * img_nrows / height)

def preprocess_image(image_path):
        # Util function to open, resize and format pictures into
appropriate tensors
    img = keras.preprocessing.image.load_img(
        image_path, target_size=(img_nrows, img_ncols)
    )
```

```
    img = keras.preprocessing.image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = vgg19.preprocess_input(img)
    return tf.convert_to_tensor(img)


def deprocess_image(x):
    # Util function to convert a tensor into a valid image
    x = x.reshape((img_nrows, img_ncols, 3))
    # Remove zero-center by mean pixel
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    # 'BGR'->'RGB'
    x = x[:, :, ::-1]
    x = np.clip(x, 0, 255).astype("uint8")
    return x

from IPython.display import Image, display

display(Image(filename=base_image_path, width=500, height=500))
display(Image(filename=style_reference_image_path,            width=500,
height=500))


# The gram matrix of an image tensor (feature-wise outer product)


def gram_matrix(x):
    x = tf.transpose(x, (2, 0, 1))
    features = tf.reshape(x, (tf.shape(x)[0], -1))
    gram = tf.matmul(features, tf.transpose(features))
    return gram


# The "style loss" is designed to maintain
# the style of the reference image in the generated image.
# It is based on the gram matrices (which capture style) of
# feature maps from the style reference image
# and from the generated image


def style_loss(style, combination):
```

```
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows * img_ncols
      return tf.reduce_sum(tf.square(S - C)) / (4.0 * (channels**2) *
(size**2))


# An auxiliary loss function
# designed to maintain the "content" of the
# base image in the generated image


def content_loss(base, combination):
    return tf.reduce_sum(tf.square(combination - base))


# The 3rd loss function, total variation loss,
# designed to keep the generated image locally coherent


def total_variation_loss(x):
    a = tf.square(
            x[:, : img_nrows - 1, : img_ncols - 1, :] - x[:, 1:, :
img_ncols - 1, :]
    )
    b = tf.square(
         x[:, : img_nrows - 1, : img_ncols - 1, :] - x[:, : img_nrows -
1, 1:, :]
    )
    return tf.reduce_sum(tf.pow(a + b, 1.25))

# Build a VGG19 model loaded with pre-trained ImageNet weights
model = vgg19.VGG19(weights="imagenet", include_top=False)

# Get the symbolic outputs of each "key" layer (we gave them unique
names).
outputs_dict = dict([(layer.name, layer.output) for layer in
model.layers])

# Set up a model that returns the activation values for every layer in
# VGG19 (as a dict).
```

```
feature_extractor = keras.Model(inputs=model.inputs,
outputs=outputs_dict)


# List of layers to use for the style loss.
style_layer_names = [
    "block1_conv1",
    "block2_conv1",
    "block3_conv1",
    "block4_conv1",
    "block5_conv1",
]
# The layer to use for the content loss.
content_layer_name = "block5_conv2"


def compute_loss(combination_image, base_image, style_reference_image):
    input_tensor = tf.concat(
        [base_image, style_reference_image, combination_image], axis=0
    )
    features = feature_extractor(input_tensor)

    # Initialize the loss
    loss = tf.zeros(shape=())

    # Add content loss
    layer_features = features[content_layer_name]
    base_image_features = layer_features[0, :, :, :]
    combination_features = layer_features[2, :, :, :]
    loss = loss + content_weight * content_loss(
        base_image_features, combination_features
    )
    # Add style loss
    for layer_name in style_layer_names:
        layer_features = features[layer_name]
        style_reference_features = layer_features[1, :, :, :]
        combination_features = layer_features[2, :, :, :]
        sl = style_loss(style_reference_features, combination_features)
        loss += (style_weight / len(style_layer_names)) * sl

    # Add total variation loss
    loss += total_variation_weight *
total_variation_loss(combination_image)
```

```
    return loss

@tf.function
def        compute_loss_and_grads(combination_image,        base_image,
style_reference_image):
    with tf.GradientTape() as tape:
                loss   =   compute_loss(combination_image,   base_image,
style_reference_image)
    grads = tape.gradient(loss, combination_image)
    return loss, grads


optimizer = keras.optimizers.SGD(
    keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate=100.0, decay_steps=100, decay_rate=0.96
    )
)

base_image = preprocess_image(base_image_path)
style_reference_image = preprocess_image(style_reference_image_path)
combination_image = tf.Variable(preprocess_image(base_image_path))

iterations = 1000
for i in range(1, iterations + 1):
    loss, grads = compute_loss_and_grads(
        combination_image, base_image, style_reference_image
    )
    optimizer.apply_gradients([(grads, combination_image)])
    if i % 100 == 0:
        print("Iteration %d: loss=%.2f" % (i, loss))
        img = deprocess_image(combination_image.numpy())
        fname = result_prefix + "_at_iteration_%d.png" % i
                        keras.preprocessing.image.save_img(fname,   img,
target_size=(img_nrows, img_ncols))
        display(Image(result_prefix + "_at_iteration_%d.png" % i))
        print(combination_image.shape)
```

import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
from tkinter import Canvas
from tkinter import NW

```
import cv2

from tensorflow import keras
import numpy as np
from tensorflow.keras.applications import vgg19
import tensorflow as tf
from tkinter import ttk




img_nrows = 256
img_ncols = 256


class MainScreen(tk.Frame):

    def update_quality(self, value):
            # Actualizar la calidad de la imagen aquí
            img_nrows=value
            img_ncols=value
            self.value_label.config(text=f"Quality: {value}")

    def __init__(self, master):
        super().__init__(master)
        self.master = master
        self.image = None
        self.pack()
        self.create_widgets()
        self.camera = cv2.VideoCapture(0) # inicializar la cámara

    def create_widgets(self):
        # Definir la fuente personalizada
        my_font = ("Garamond", 12)

        # Establecer la fuente predeterminada para todos los widgets de la aplicación
        self.master.option_add("*Font", my_font)



        self.create_title_label()

        # variable de control para el modelo seleccionado
        self.model_var = tk.StringVar(self)
```

```
        self.model_var.set("MODEL 1")

        # lista de imágenes para cada modelo, reescaladas a la mitad
        self.model_images = {
            "MODEL 1": tk.PhotoImage(file="model1.png").subsample(5),
            "MODEL 2": tk.PhotoImage(file="model2.png").subsample(7),
            "MODEL 3": tk.PhotoImage(file="model3.png").subsample(5)
        }

        # widget Label para mostrar la imagen
                                            self.model_image_label = tk.Label(self,
image=self.model_images[self.model_var.get()], bg="white", font="Garamond" )
        self.model_image_label.pack()


        # función para actualizar la imagen cada vez que cambie el modelo seleccionado
        def update_image(*args):
                self.model_image_label.config(image=self.model_images[self.model_var.get()],
bg="white")

        # actualizar la imagen al inicio
        update_image()

        # asociar la variable de control con el widget OptionMenu
            self.model_option_menu = tk.OptionMenu(self, self.model_var, "MODEL 1",
"MODEL 2", "MODEL 3", command=update_image)
        self.model_option_menu.pack()

        self.image_label = tk.Label(self)
        self.image_label.pack()

        # Crear un Frame para los botones "Take Photo" y "Upload Image"
        button_frame = tk.Frame(self)
        button_frame.config(bg="white")
        button_frame.pack()

        # Crear el botón "Take Photo"
                self.photo_button = tk.Button(button_frame, text="TAKE PICTURE",
font=("Garamond"), command=self.take_photo)
        image = tk.PhotoImage(file="camera.png")
        image = image.subsample(15)
                            self.photo_button.config(image=image, width=160, height=40,
compound=tk.LEFT)
        self.photo_button.image = image
```

```
        self.photo_button.pack(side=tk.LEFT, padx=5)


        # Crear el botón "Upload Image"
            self.upload_button = tk.Button(button_frame, command=self.upload_image, bg="white", fg="black", font="Garamond")
             upload_image = tk.PhotoImage(file="uploadImage.png")  # Reemplaza "ruta_de_la_imagen.png" con la ruta de tu imagen
        upload_image = upload_image.subsample(15)
            self.upload_button.config(image=upload_image, text="UPLOAD IMAGE", width=160, height=40, compound=tk.LEFT)  # Ajusta los valores de width y height según tus necesidades
        self.upload_button.image = upload_image
        self.upload_button.pack(side=tk.LEFT, padx=5)

        # Crear un Frame para los botones "Crop" y "Confirm"
        crop_confirm_frame = tk.Frame(self)
        crop_confirm_frame.config(bg="white")
        crop_confirm_frame.pack()

        # Crear un botón para iniciar el recorte
                self.crop_button = tk.Button(crop_confirm_frame,command=self.crop, bg="white", fg="black", font="Garamond")
        crop_image = tk.PhotoImage(file="crop.png")
        crop_image = crop_image.subsample(15)
         self.crop_button.config(image=crop_image, text="CROP", width=100, height=40, compound=tk.LEFT)  # Ajusta los valores de width y height según tus necesidades
        self.crop_button.image = crop_image
        self.crop_button.pack(side=tk.LEFT, padx=5, pady=5)

        # Crear un botón para confirmar
            self.confirm_button = tk.Button(crop_confirm_frame, text="CONFIRM", command=self.confirm, bg="white", fg="black", font="Garamond")
        confirm_image = tk.PhotoImage(file="confirm.png")
        confirm_image = confirm_image.subsample(25)
         self.confirm_button.config(image=confirm_image, text="CONFIRM", width=120, height=40, compound=tk.LEFT)  # Ajusta los valores de width y height según tus necesidades
        self.confirm_button.image = confirm_image
        self.confirm_button.pack(side=tk.LEFT, padx=5)



        # Crear el slider de calidad
```

```
        self.quality_scale_value = tk.IntVar()
        self.quality_scale_value.set(128)  # valor inicial del slider
        self.quality_scale = tk.Scale(self, from_=32, to=512, length=200,
                                      variable=self.quality_scale_value, command=self.update_quality,
orient=tk.HORIZONTAL)
        self.quality_scale.set(128)  # valor inicial del slider
        self.quality_scale.pack()

        # Crear la etiqueta para mostrar el valor del slider
        self.value_label = tk.Label(self, text="Quality: 128")
        self.value_label.pack()

        # Lista de valores fijos permitidos para el control deslizante
        fixed_values = [32, 64, 128, 192, 256, 512]

        def update_quality(self, value):
            # Redondear el valor del control deslizante al valor fijo más cercano
            closest_value = min(fixed_values, key=lambda x: abs(x - float(value)))
            self.quality_scale.set(closest_value)
            self.value_label.config(text=f"Quality: {closest_value}")

        # Inicializar la variable de instancia para la imagen recortada
        self.cropped_image = None

    def create_title_label(self):
            self.title_label = tk.Label(self, text="AVATAR STYLER", font=("Perpetua", 24),
bg="white", fg="black")
        self.title_label.pack()


    def crop(self):

        if self.image is None:
            return

        # Crear una nueva ventana para el recorte
        self.crop_window = tk.Toplevel(self.master)

         # Mostrar la imagen en un Canvas para permitir que el usuario seleccione la zona
de recorte
                        self.canvas = Canvas(self.crop_window, width=self.image.width,
height=self.image.height)
        self.canvas.pack()
        self.tk_image = ImageTk.PhotoImage(self.image)
```

```
        self.canvas.create_image(0, 0, image=self.tk_image, anchor=NW)

        # Permitir que el usuario seleccione la zona de recorte
        self.crop_area = self.canvas.create_rectangle(0, 0, 0, 0, outline="red")
        self.canvas.bind("<ButtonPress-1>", self.start_crop)
        self.canvas.bind("<B1-Motion>", self.crop_drag)
        self.canvas.bind("<ButtonRelease-1>", self.end_crop)

        # Crear un botón para confirmar el recorte
                self.confirm_button = tk.Button(self.crop_window, text="Confirmar",
command=self.confirm_crop)
        self.confirm_button.pack()

    def start_crop(self, event):
        self.start_x = event.x
        self.start_y = event.y

    def crop_drag(self, event):
        self.canvas.coords(self.crop_area, self.start_x, self.start_y, event.x, event.y)

    def end_crop(self, event):
        self.end_x = event.x
        self.end_y = event.y

    def confirm_crop(self):
        # Obtener las coordenadas de la zona de recorte
        x1 = min(self.start_x, self.end_x)
        y1 = min(self.start_y, self.end_y)
        x2 = max(self.start_x, self.end_x)
        y2 = max(self.start_y, self.end_y)

        # Recortar la imagen y almacenarla en la variable de instancia
        self.cropped_image = self.image.crop((x1, y1, x2, y2))

        # Actualizar la imagen en el label original
        self.photo = ImageTk.PhotoImage(self.cropped_image)
        self.image_label.configure(image=self.photo) #AQUIIIIIIIIII

        # Cerrar la ventana de recorte
        self.crop_window.destroy()


    def take_photo(self):
        ret, frame = self.camera.read()  # Capturar un frame de la cámara
```

```
        if ret:
            # Convertir el frame a una imagen PIL y mostrarla en el widget Label
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            self.image = Image.fromarray(image)
            self.photo = ImageTk.PhotoImage(self.image)
            self.image_label.config(image=self.photo)


    def upload_image(self):
        # Abrir un cuadro de diálogo para seleccionar un archivo de imagen
        file_path = filedialog.askopenfilename()
        if file_path:
            # Cargar la imagen seleccionada y redimensionarla
            self.image = Image.open(file_path)
              resized_image = self.image.resize((256, 256))  # Redimensionar a un tamaño fijo (256x256)

            # Mostrar la imagen redimensionada en el widget Label
            self.photo = ImageTk.PhotoImage(resized_image)
            self.image_label.config(image=self.photo)


        # # Abrir un cuadro de diálogo para seleccionar un archivo de imagen
        # file_path = filedialog.askopenfilename()
        # if file_path:
        #     # Cargar la imagen seleccionada y mostrarla en el widget Label
        #     self.image = Image.open(file_path)
        #     self.photo = ImageTk.PhotoImage(self.image)
        #     self.image_label.config(image=self.photo)




    def confirm(self):
        loading_screen = LoadingScreen(self)
        if self.cropped_image:
            image_path = "cropped_image.png"  # Ruta de archivo temporal para la imagen recortada
                self.cropped_image.save(image_path)  # Guardar la imagen recortada en un archivo temporal
        else:
                image_path = "uploaded_image.png"  # Ruta de archivo temporal para la imagen cargada
```

```
            self.image.save(image_path)  # Guardar la imagen cargada en un archivo temporal
        loading_screen.start_process(image_path)
        self.destroy()




class LoadingScreen(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.master = master
        self.pack()
        self.create_widgets()

    def create_widgets(self):
        self.loading_label = tk.Label(self, text="Loading...")
        self.loading_label.pack()

            self.progress_bar = ttk.Progressbar(self, orient='horizontal', length=200, mode='determinate')

        self.progress_bar.pack()

        self.start_process()


    def start_process(self):
        # Obtén la imagen seleccionada por el usuario
            input_image = self.master.cropped_image if self.master.cropped_image else self.master.image

        # Carga el modelo de style transfer
        model = keras.models.load_model("modeloGritoFinal.h5")

        # Obtén la ruta del archivo de imagen
                image_path = "cropped_image.png" if self.master.cropped_image else "uploaded_image.png"
        input_image.save(image_path)  # Guardar la imagen en un archivo temporal

          # Preprocesa la imagen de entrada para adaptarla al formato requerido por el modelo
        preprocessed_image = self.preprocess_image(image_path)
```

```
        # Aplica el estilo artístico a la imagen mediante la transferencia de estilo
        # stylized_image = model.predict(preprocessed_image)


        style_transfer = StyleTransfer()
        content_image_path = preprocessed_image
        style_image_path = "model2.png"
        output_image = style_transfer.aplica_estilo(image_path, style_image_path, 500)
        output_image.save("result2.png")


        #deprocessed_image = Image.open('ruta_de_tu_imagen_deprocesada.jpg')
        # deprocessed_image = self.deprocess_image(output_image)
            # output_image = np.squeeze(output_image, axis=0)  # Elimina la dimensión adicional del lote

        # print(deprocessed_image.shape)

        # output_image = Image.fromarray(deprocessed_image)
        # output_path = "imagen_estilizada.png"




        # # Guarda la imagen estilizada en un archivo
            # stylized_image = np.squeeze(stylized_image, axis=0)  # Elimina la dimensión adicional del lote

        # print(stylized_image.shape)

        # stylized_image = self.deprocess_image(stylized_image)  # Desprocesa la imagen
            # output_image = Image.fromarray(stylized_image)  # Crea una instancia de PIL.Image
        # output_path = "ruta/a/donde/guardar/imagen_estilizada.png"
        # output_image.save(output_path)  # Guarda la imagen estilizada en un archivo

        # Muestra la pantalla de resultados
        self.result_screen = ResultScreen(self.master)
        self.destroy()

    @staticmethod
    def preprocess_image(image_path):
        img = keras.preprocessing.image.load_img(
            image_path, target_size=(img_nrows, img_ncols)
```

```
        )
        img = keras.preprocessing.image.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = vgg19.preprocess_input(img)
        return img

    @staticmethod
    def deprocess_image(x):

        x = x.reshape((img_nrows, img_ncols, 3)).astype("float64")
        # Reajusta la imagen a su rango original
        x[:, :, 0] += 103.939
        x[:, :, 1] += 116.779
        x[:, :, 2] += 123.68
        # Convierte de BGR a RGB
        x = x[:, :, ::-1]
        # Asegúrate de que los valores estén en el rango de 0 a 255
        x = np.clip(x, 0, 255).astype("uint8")
        return x



class StyleTransfer:
    def __init__(self, img_nrows=256, img_ncols=256):
        self.img_nrows = img_nrows
        self.img_ncols = img_ncols
        self.total_variation_weight = 1e-6
        self.style_weight = 1e-6
        self.content_weight = 2.5e-8
        self.style_layer_names = [
            "block1_conv1",
            "block2_conv1",
            "block3_conv1",
            "block4_conv1",
            "block5_conv1",
        ]
        self.content_layer_name = "block5_conv2"
        self.model = None
        self.feature_extractor = None
        self.optimizer = None

    def preprocess_image(self, image_path):
        img = keras.preprocessing.image.load_img(
            image_path, target_size=(self.img_nrows, self.img_ncols)
```

```
        )
        img = keras.preprocessing.image.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = vgg19.preprocess_input(img)
        return tf.convert_to_tensor(img)

    def deprocess_image(self, x):
        x = x.reshape((self.img_nrows, self.img_ncols, 3))
        x[:, :, 0] += 103.939
        x[:, :, 1] += 116.779
        x[:, :, 2] += 123.68
        x = x[:, :, ::-1]
        x = np.clip(x, 0, 255).astype("uint8")
        return x

    def gram_matrix(self, x):
        x = tf.transpose(x, (2, 0, 1))
        features = tf.reshape(x, (tf.shape(x)[0], -1))
        gram = tf.matmul(features, tf.transpose(features))
        return gram

    def style_loss(self, style, combination):
        S = self.gram_matrix(style)
        C = self.gram_matrix(combination)
        channels = 3
        size = self.img_nrows * self.img_ncols
        return tf.reduce_sum(tf.square(S - C)) / (4.0 * (channels ** 2) * (size ** 2))

    def content_loss(self, base, combination):
        return tf.reduce_sum(tf.square(combination - base))

    def total_variation_loss(self, x):
            a = tf.square(x[:, : self.img_nrows - 1, : self.img_ncols - 1, :] - x[:, 1:, : self.img_ncols - 1, :])
        b = tf.square(x[:, : self.img_nrows - 1, : self.img_ncols - 1, :] - x[:, : self.img_nrows - 1, 1:, :])
        return tf.reduce_sum(tf.pow(a + b, 1.25))

    def compute_loss(self, combination_image, base_image, style_reference_image):
        input_tensor = tf.concat([base_image, style_reference_image, combination_image], axis=0)
        features = self.feature_extractor(input_tensor)

        loss = tf.zeros(shape=())
```

```
        layer_features = features[self.content_layer_name]
        base_image_features = layer_features[0, :, :, :]
        combination_features = layer_features[2, :, :, :]

                    loss += self.content_weight * self.content_loss(base_image_features,
combination_features)

        for layer_name in self.style_layer_names:
            layer_features = features[layer_name]
            style_reference_features = layer_features[1, :, :, :]
            combination_features = layer_features[2, :, :, :]
            sl = self.style_loss(style_reference_features, combination_features)
            loss += (self.style_weight / len(self.style_layer_names)) * sl

        loss += self.total_variation_weight * self.total_variation_loss(combination_image)
        return loss

    def aplica_estilo(self, content_image_path, style_image_path, iterations):
        content_image = self.preprocess_image(content_image_path)
        style_image = self.preprocess_image(style_image_path)

        base_image = tf.Variable(content_image, dtype=tf.float32)
        style_reference_image = tf.Variable(style_image, dtype=tf.float32)
        combination_image = tf.Variable(content_image, dtype=tf.float32)

        self.feature_extractor = vgg19.VGG19(
                    include_top=False, weights="imagenet", input_shape=(self.img_nrows,
self.img_ncols, 3)
        )

                        outputs_dict = dict([(layer.name, layer.output) for layer in
self.feature_extractor.layers])
                self.feature_extractor = keras.Model(inputs=self.feature_extractor.inputs,
outputs=outputs_dict)
        self.feature_extractor.trainable = False

        self.optimizer = tf.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)

        @tf.function()
        def train_step(combination_image):
            with tf.GradientTape() as tape:
                                loss = self.compute_loss(combination_image, base_image,
style_reference_image)
```

```
            grads = tape.gradient(loss, combination_image)
            self.optimizer.apply_gradients([(grads, combination_image)])
                                combination_image.assign(tf.clip_by_value(combination_image,
clip_value_min=0.0, clip_value_max=255.0))

        for i in range(iterations):
            train_step(combination_image)

        final_image = self.deprocess_image(combination_image.numpy())
        final_image = Image.fromarray(final_image)
        return final_image


class ResultScreen(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.master = master
        self.pack()
        self.create_widgets()

    def create_widgets(self):
        self.result_label = tk.Label(self, text="Result:")
        self.result_label.pack()

        self.result_image = tk.PhotoImage(file="result2.png")
                    self.result_canvas = tk.Canvas(self, width=self.result_image.width(),
height=self.result_image.height())
        self.result_canvas.create_image(0, 0, anchor=tk.NW, image=self.result_image)
        self.result_canvas.pack()

                    self.download_button = tk.Button(self, text="Download Image",
command=self.download)
        self.download_button.pack()

                self.return_button = tk.Button(self, text="Return to Main Screen",
command=self.return_to_main)
        self.return_button.pack()

    def download(self):
        # TODO: Implement image downloading functionality
        pass

    def return_to_main(self):
        main_screen = MainScreen(self.master)
```

```
        self.destroy()

if __name__ == "__main__":
    root = tk.Tk()
    root.geometry("400x400")
    main_screen = MainScreen(root)
    root.mainloop()
```

B. Diagram and Figure List

2. PLANNING AND RESOURCES EVALUATION

3. SYSTEM ANALYSIS AND DESIGN

4. WORK DEVELOPMENT AND RESULTS