**PAINTER RECOGNITION**

**Prepared By:**

Sildo Xhallari

Natalia Charlviani

**Introduction**

In this project, our goal was to develop and train a model capable of analyzing paintings and accurately identifying their artists through data preprocessing, model training, and model evaluation. This task utilizes advanced techniques in computer vision and machine learning to correctly detect the painter.

The dataset used for the project has been downloaded from [Kaggle](https://www.kaggle.com/ikarus777/best-artworks-of-all-time). It contains the paintings of 50 international painters, for a total of 8446 paintings. The paintings differ greatly from each other in size, but also in color and style.

**METHOD**

As we already mentioned, the used dataset contains images of paintings from various artists and to correctly train the dataset we took the following key steps: Data Preparation, Model Architecture designing, Model Training, Model Evaluation.

##### **Data Preparation**

The dataset comprises paintings stored in the directory './paintings\_dataset/images', where each folder within this directory represents a different painter and contains the paintings by that artist. Each image is resized to 224x224 pixels and converted to a tensor using PyTorch's ImageFolder and Transforms functionalities. This prepares the data for training and testing by loading images and applying necessary transformations such as resizing and conversion to tensors. After loading the dataset, we split it into training and testing subsets with an 80/20 ratio to ensure that we could effectively evaluate the model's performance on data.

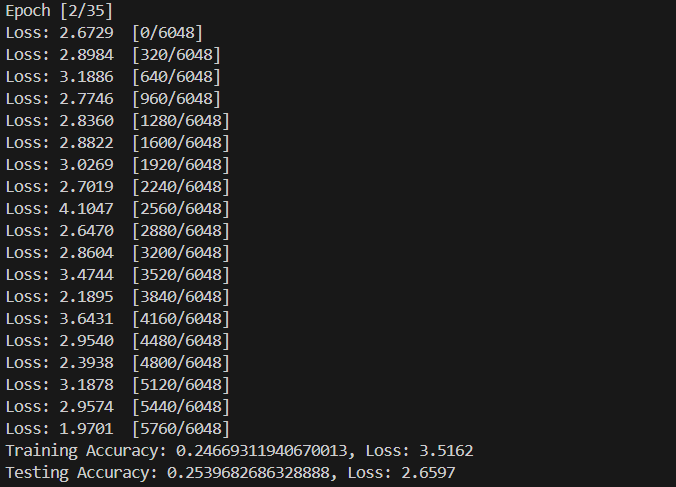
##### **Model Architecture Design**

Next, we designed our CNN model to capture the intricate details and patterns within the paintings. Our model architecture included three sets of convolutional layers, each followed by max-pooling layers to reduce the dimensions while retaining important features. The convolutional layers were essential for extracting hierarchical features from the images. These features were then flattened and passed through fully connected layers to classify the images into different artist categories. We incorporated ReLU activation functions for non-linearity and dropout regularization to prevent overfitting.

#### **Model Training**

With the data prepared and the model designed, we proceeded to the training phase. We used PyTorch's DataLoader to load the training and testing data in batches of 16. The model was configured to use a GPU if available for faster computation, otherwise defaulting to CPU. For optimization, we chose the AdamW optimizer with a learning rate of 0.001, and Cross-Entropy Loss was used as our loss function due to its suitability for multi-class classification tasks. The training process spanned 35 epochs. During each epoch, the model's parameters were updated to minimize the loss, and we tracked both training and testing accuracy to monitor performance.

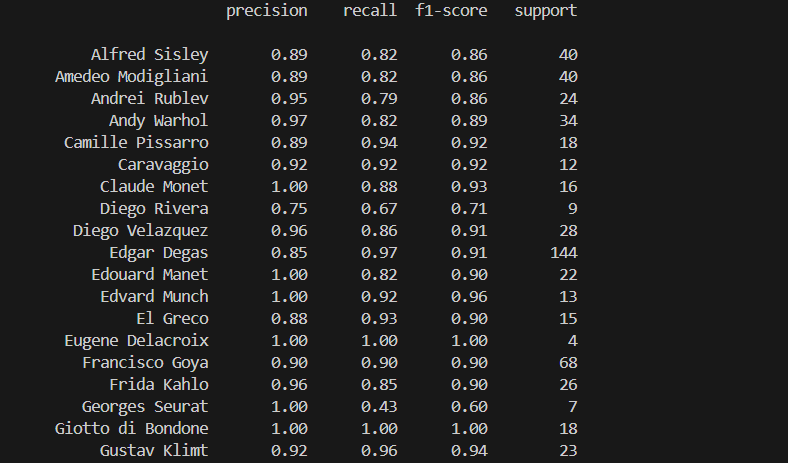
As a result, we got 97.7% training accuracy and 0.1 loss % , 39% validation accuracy and 2.5 loss.

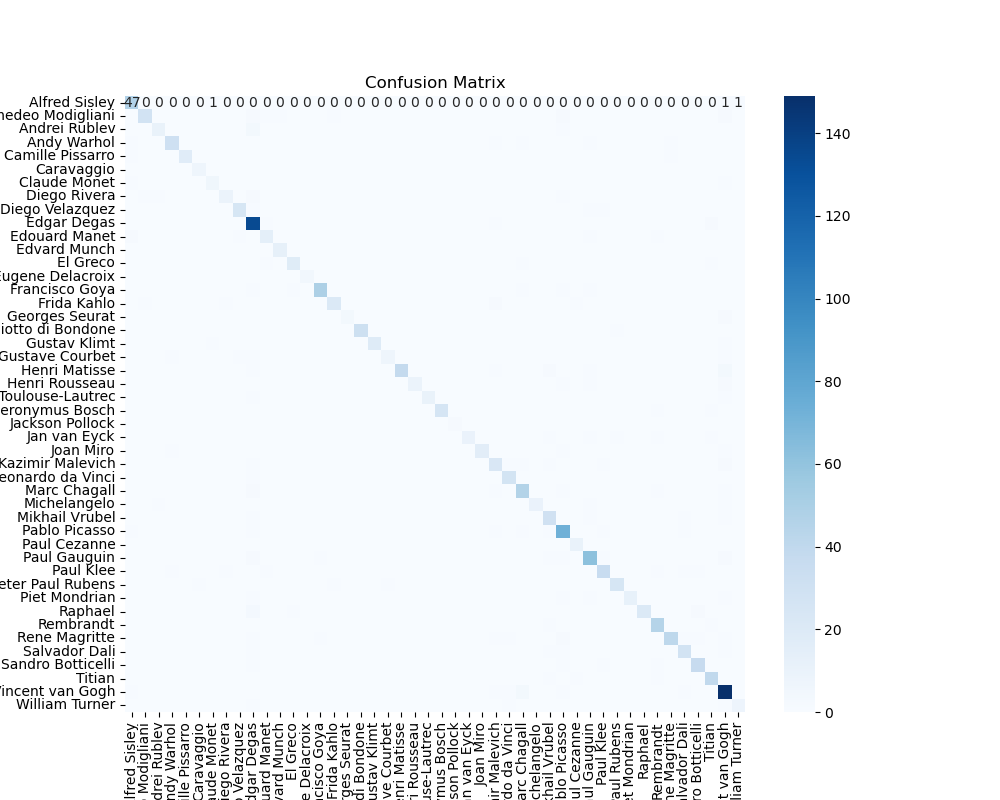


This is the training process.

##### **Model Evaluation**

After completing the training, we evaluated the model's performance using the testing dataset. We loaded the trained model's parameters from a saved state and used sklearn.metrics.classification\_report to compute various performance metrics, including precision, recall, and F1-score. We also generated a confusion matrix to visualize how well the model performed across different artist categories. These evaluations provided a comprehensive understanding of the model's strengths and areas for improvement.





##### **Example Prediction**

To demonstrate the practical application of our model, we performed an example prediction. We defined the predict\_painter function to predict the artist of a painting image ('./kle.jpg') etc. This function preprocessed the image, passed it through the trained model, and outputted the predicted artist. This example highlighted the model's capability to make real-world predictions and its potential utility in art analysis.

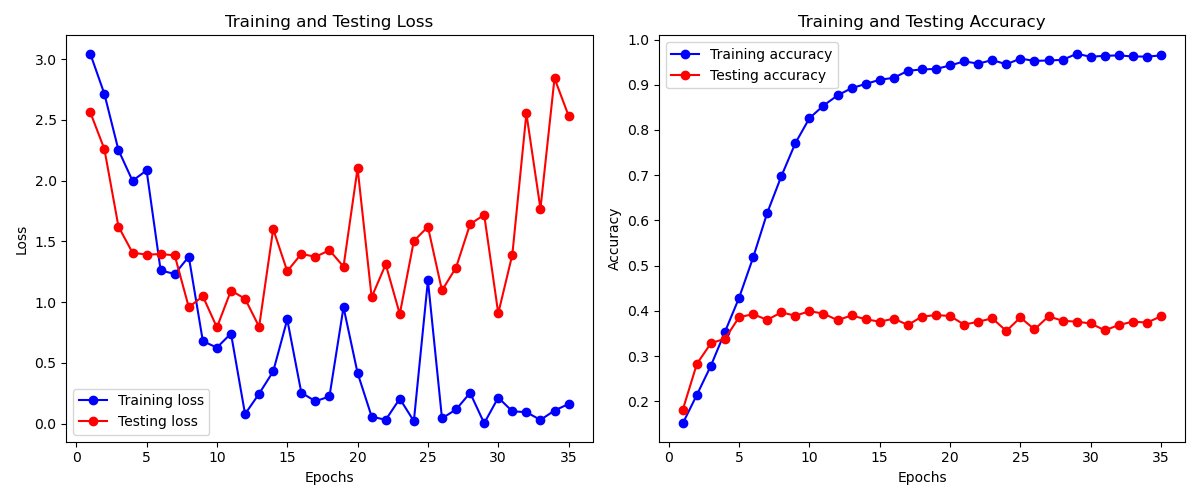
RESULTS

We didn’t expect for the model to not correctly predict the painter but upon further research we discovered that the cause lies within the balance between transformations and Convolutional layers.

By transforming the image too much it lost its distinctive properties, whereas not having enough convolutional layers or not giving them enough filters we couldn’t save many details.

In our case we ended up not applying any transformations to the images besides resizing them, finally resulting in the model not being accurate enough. As we started tuning the convolutional layers and adding the transformations however, we had a few drawbacks (unrelated to the project) that resulted in us not being able to get everything done correctly on time for the project deadline.

As the images above show, with each epoch passing the results got slightly worse. We also thought that another factor to take into account was overfitting.

As time went by testing accuracy got really close to 100% and training loss really close to 0%. This clear difference between training and testing made us think of what other variables might have caused these results. 

Not being able to train the model over again and the large gap between training and testing loss and accuracy tests got us searching answers on the internet. What could have been done to improve our code? Is there a way to reduce the gap between training and testing? During our research we came across a project on github that is very similar to what we did (<https://github.com/enigarv/Painter_recognition/tree/main>).

Here we had the opportunity to look into more detail the things we couldn’t work out on our own and got some more insight on the overall project.

The authors of the linked github project discovered that the data augmentation negatively affects the results just as we thought. It is likely that the excessive modification of the painting does not allow the correct painter to be associated with it.

This came as a surprise because we thought that it could have made the model more accurate. It would have been really interesting to double check this, however we couldn’t have trained and tested the model in time.

We also discovered that having that many painters with such few paintings in the dataset could be one of the reasons for our not so great results. Having a dataset that heavily leans towards certain painters makes it so that there’s a lower chance of recognising painters with few paintings.

The authors of the github project we found solved this problem by eliminating from the dataset all the painters that had less than 200 paintings and by adding weights to each artist (the artists that have more paintings have a lower value and vice-versa).

With the information provided by the aforementioned project we also got to understand how using different models for fine tuning has an impact on overall performance. The authors considered three different known architectures such as VGG-16, MobileNet-v2 and ResNet-50 each with a different amount of layers and size (making them more suitable for certain situations over others).

In conclusion, the developed CNN model effectively learns to classify paintings through data preprocessing, model training, and model evaluation based on the artist using a dataset of painting images. Even though we were not fully satisfied with the results, by the end of the project we got a better understanding of the topic and we hope that with enough time we would have been able to produce a more accurate model.