Università degli Studi di Milano

Data Science for Economics

Algorithms for Massive Data

Project 4: Picture Recognizer

Ketrin Hristova
Matriculation Number: 13209A

Declaration

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work, and including any code produced using generative AI systems.
I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying.
This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

1 Introduction

The aim of this project is to implement Convolutional Neural Network (CNN) models for the task of image classification. Specifically, the goal is to classify images of artworks based on their respective authors, using the dataset provided by Kaggle. The dataset, `prado.csv`, contains information on various artworks and their respective authors, and it will be used to train the CNN models to recognize and categorize these artworks into four specific authors.

1.1 Dataset Description

The dataset used in this project is **prado.csv** from the Prado Museum Pictures collection, which was accessed via the Kaggle API. It contains 13,487 records across 30 columns, where each record represents a unique artwork. Important columns include details such as the image URL and the author's name.

Additionally, a new column, **work_id**, was created by extracting the unique identifier from each artwork's image URL. This enables easy linking of the images to their respective metadata for further processing in the project.

1.2 Data Preprocessing

The original dataset consists of artworks from a range of artists. However, for the classification task, a focused subset of four artists was selected: **Goya**, **Bayeu**, **Haes**, and **Pizarro**. This selection served to reduce the complexity of the dataset, making it more manageable and well-suited for effective model training.
To ensure consistent input to the model, all images were resized to a uniform dimension of `224x224` pixels, in line with the model's input requirements. Additionally, the color mode was set to `rgb`, retaining full color information necessary for recognizing the visual details of artworks. The dataset was then split into **80% training** and **20% validation** subsets, ensuring that proper model evaluation could be conducted. Labels for each image were automatically derived from the directory structure, while the images were processed in batches of 32, optimizing training efficiency.

Class Imbalance Mitigation: To address potential issues of class imbalance, where certain classes have significantly more samples than others, **class weighting** was employed. Class imbalance can cause the model to become biased towards the majority class, resulting in poor performance on the underrepresented classes. By assigning *higher weights* to the underrepresented classes, the model was encouraged to focus more on these categories, thus promoting a more balanced and effective learning process. This approach helped improve classification performance across all classes.

1.3 Data Augmentation

To enhance the model's generalization ability and prevent overfitting, **data augmentation** was applied during the training process. Data augmentation artificially expands the training dataset by applying random transformations to the images, allowing the model to learn more robust and invariant features. The following augmentation techniques were implemented:

- **Random Horizontal Flip:** The image is flipped horizontally at random, helping the model become invariant to left-right orientations.
- **Random Rotation:** The image is rotated by a random angle, introducing slight variations in its orientation.
- **Random Zoom:** The image is zoomed in or out, simulating scale changes in the artwork objects.
- **Random Contrast:** The image contrast is randomly adjusted, making the model more resilient to variations in lighting and exposure.

In addition to augmentation, the images were **normalized** by scaling pixel values to the range $[0, 1]$, achieved by dividing each pixel by 255. This normalization step is a standard practice to ensure that the pixel values are appropriately scaled for model training.

For improved performance, the dataset was **cached**, **shuffled**, and **prefetched**, optimizing the input pipeline for efficient training.

2 CNN Models Implementation

This section outlines the implementation of various CNN models for classifying artwork images based on their respective authors. The process began with a base model that did not utilize data augmentation techniques. This model served as a baseline, allowing for the evaluation of more complex architectures and the effects of data augmentation on the model's generalization ability.

To enhance model performance and reduce the risk of overfitting, data augmentation was introduced in the subsequent models. Different architectural variations were explored by increasing the number of convolutional layers and modifying the dense layers. The goal was to identify the optimal model structure before proceeding with hyperparameter tuning to finalize the model.

2.1 Execution Specifications

For the CNN model implementations, a **GPU** was used for training to accelerate computation, which is crucial given the large size of the dataset and the complexity of the neural network models. Using a GPU allowed for faster training, enabling the experimentation with different model architectures and hyperparameters within a reasonable timeframe.

The models were trained for **40 epochs** in each configuration. This number of epochs was chosen to strike a balance between allowing the model enough time to *converge* and learn from the data, while also minimizing the risk of overfitting. Training for too few epochs might result in underfitting, while training for too many could lead to overfitting.

For the **loss function**, *Sparse Categorical Cross-Entropy* was utilized. This loss function is particularly effective for multi-class classification tasks where the labels are integers rather than one-hot encoded vectors. In this context, it helps compute the difference between predicted probabilities and the true class labels, ensuring that the model learns to classify the artwork correctly based on the respective authors.

The **Adam optimizer** was chosen for training. This optimizer adapts the learning rate for each parameter individually, which enhances convergence speed and often results in improved performance compared to traditional gradient descent. By adjusting the learning rate during training, Adam ensures that the model parameters are optimized efficiently, leading to more stable and faster convergence.

2.2 Base Model (Without Data Augmentation)

The first model implemented is a base Convolutional Neural Network (CNN) used to classify artwork images without data augmentation. This model serves as a benchmark to assess the impact of data augmentation on the model's performance and ability to generalize.

The model begins by **rescaling the input images** to a range of 0 to 1 using a `Rescaling` layer, normalizing pixel values for more efficient learning. The model then applies three **convolutional layers** with 32, 64, and 128 filters, respectively. These convolutional layers are each followed by `max-pooling` layers.

After the convolutional layers, the feature maps are **flattened into a one-dimensional vector** and passed through a `dense layer` with 128 neurons. A `dropout layer` with a rate of 0.5 is applied to prevent overfitting by randomly setting half of the neurons to zero during training, promoting better generalization.

The model uses **Sparse Categorical Cross-Entropy** as the loss function, suitable for multi-class classification with integer labels. The **Adam optimizer**, with a learning rate of 0.0001, is employed for faster convergence and stable training. The model is trained for **20 epochs** to allow sufficient learning.

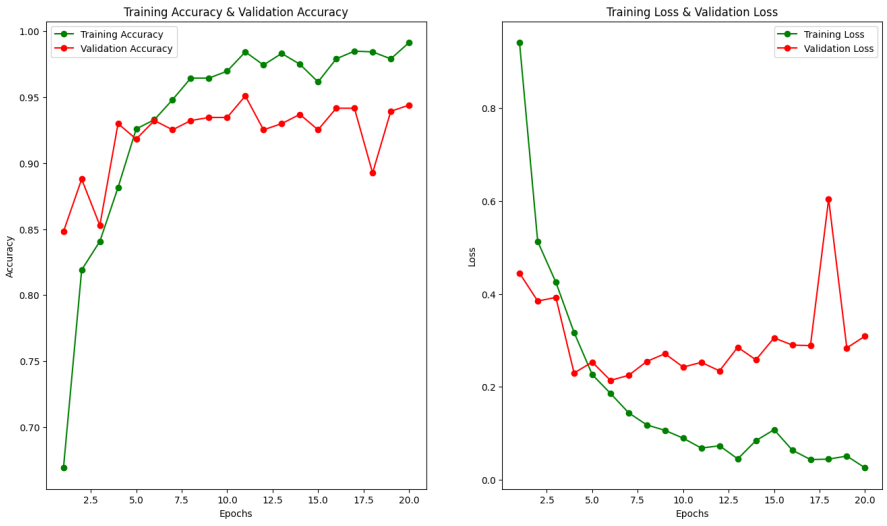

Figure 1: Base Model Architecture

The model achieved a **training accuracy of 99.11%**, indicating that it successfully learned the patterns in the training data. The low **training loss of 0.0279** further supports the model's effective learning.

However, the **validation accuracy of 94.39%** and **validation loss of 0.3092** show that while the model performs well on unseen data, there is still room for improvement. This suggests that, although the model generalizes effectively, there may be slight overfitting to the training data.

These results provide a solid foundation for further experimentation with more complex models and **data augmentation techniques** to improve generalization.

2.3 Experimentation with CNN Architectures (With Data Augmentation)

In this section, the results of the three models implemented with data augmentation are presented to determine the optimal model structure for further hyperparameter tuning. Each model was trained for 40 epochs, and performance was evaluated based on **accuracy** and **loss** on both the training and validation datasets.

The first model follows a conventional CNN architecture, with three convolutional layers incorporating **32, 64, and 128 filters**, each followed by max-pooling layers. After flattening, the model applies a fully connected dense layer with **128 neurons** and a **dropout rate of 0.5** to mitigate overfitting, concluding with a final output layer.

This model achieved a **training accuracy of 87.47%** and a **validation accuracy of 86.21%**, with a training loss of **0.3328** and validation loss of **0.3521**. The minimal difference between training and validation accuracy indicates relatively low overfitting, making this model a strong candidate for further development.

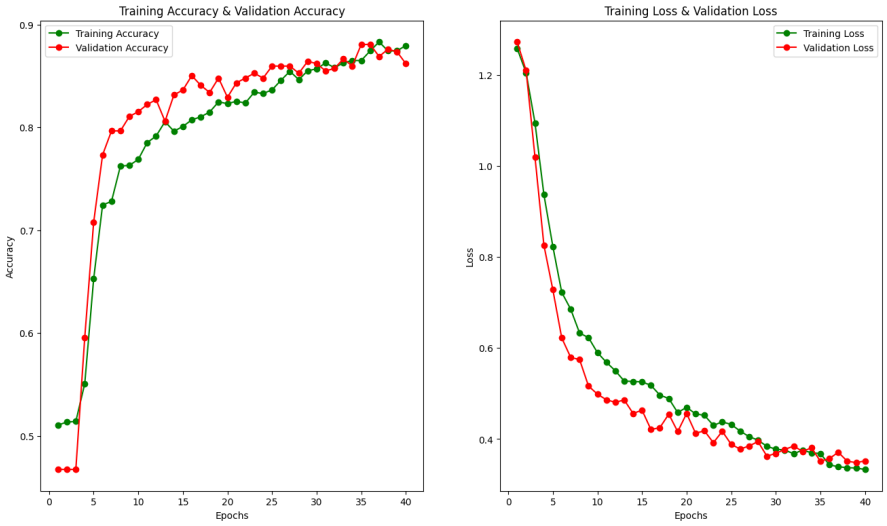

Figure 2: 3 conv 1 dense Architecture

The second model extends the architecture by adding an additional convolutional layer with **256 filters**, followed by max-pooling. This modification resulted in a **significant improvement in training accuracy**, reaching **94.63%**, and a reduction in training loss to **0.1344**. However, the **validation accuracy remained at 86.21%**, with an increased validation loss of **0.6027**. The high training accuracy and the large gap between training and validation accuracy **suggest potential overfitting**, indicating that this model may not generalize as effectively as the first one.

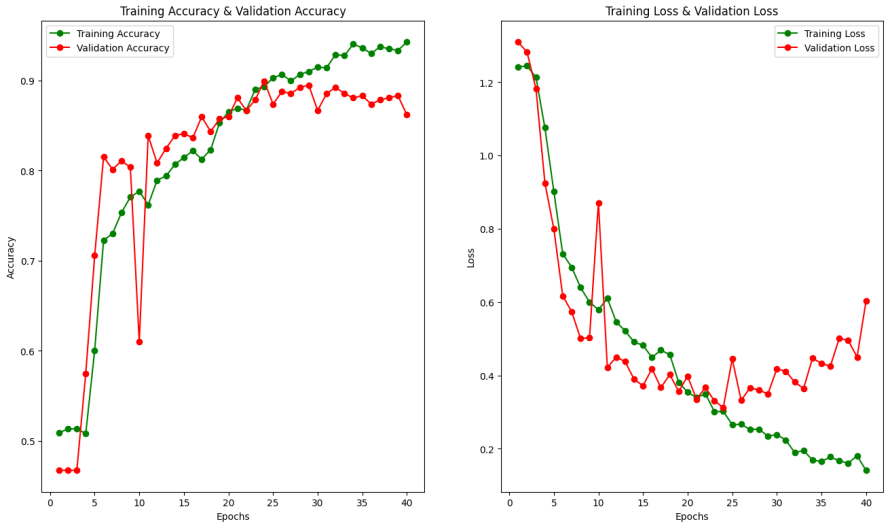

Figure 3: 4 conv 1 dense Architecture

The third model is similar to the first but incorporates an **additional dense layer with 64 neurons** and utilizes a **softmax activation function** for the output layer. This model achieved a **training accuracy of 85.17%** and a **validation accuracy of 84.58%**, with a training loss of **0.3998** and validation loss of **0.3938**. Despite the increased architectural complexity, this model **underperformed compared to the first model**, showing **higher overfitting** and lower overall accuracy.

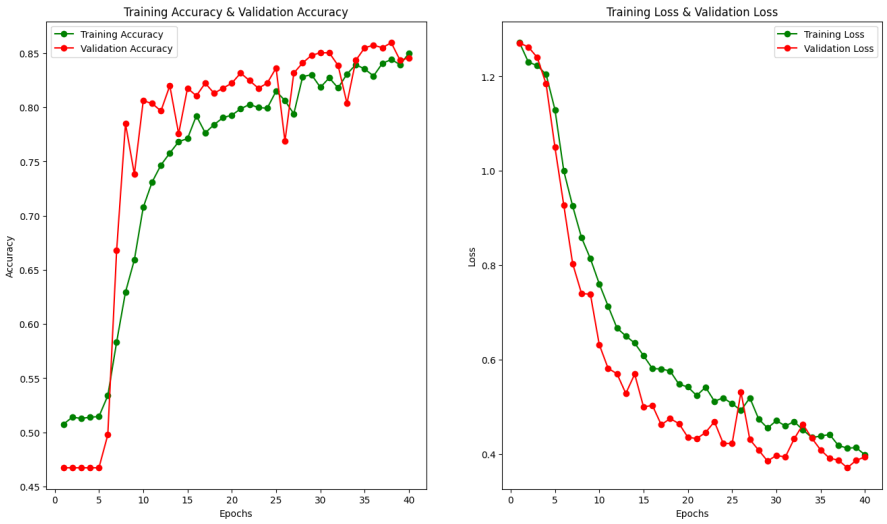

Figure 4: three conv 2 dense Architecture

Based on these results, I selected the first model for further development. It had the least overfitting and performed well on both the training and validation sets, making it the best choice for proceeding with hyperparameter tuning.

2.4 Hyperparameter Tuning and FInal Model

Hyperparameter tuning was conducted to optimize the model's performance. Using the `Keras Tuner` library, several important hyperparameters were systematically explored to improve validation accuracy. The key hyperparameters tuned included: Number of convolutional filters, Number of dense units, Dropout rate, Learning rate.

The tuning process began with determining the optimal number of filters for the convolutional layers. The best configuration was:

- **32 filters** for the first convolutional layer
- **64 filters** for the second layer
- **128 filters** for the third layer

The number of dense units was optimized next, with the best configuration being **512 dense units**.

A **dropout rate of 0.3** provided an effective balance between reducing overfitting and maintaining performance on training data.

Finally, the learning rate was tuned to find the optimal value of **0.0001**, ensuring stable and efficient training.

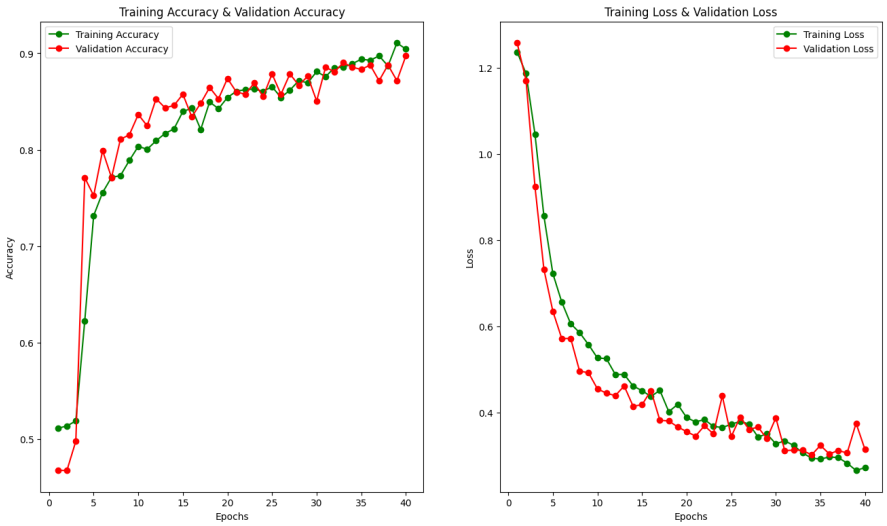

Figure 5: Final Model Architecture

The model was trained for 40 epochs, with the following results:

- **Training Accuracy**: 90.72%
- **Validation Accuracy**: 89.72%
- **Training Loss**: 0.2798
- **Validation Loss**: 0.3145

These results indicate that the model performed well on both the training and validation datasets, achieving high accuracy with a minimal difference between training and validation performance. This suggests that the model is not overfitting and is generalizing well to unseen data.

3 Conclusion

In this project, high accuracy was achieved in the classification task. The models performed exceptionally well, with the final optimized model reaching **90.72% accuracy** and demonstrating **low overfitting**. The results highlight the effectiveness of the chosen architecture and hyperparameter tuning strategies, showcasing the models' strong generalization capabilities.

Despite these promising results, there remains potential for further improvement. Experimenting with more advanced techniques and expanding the dataset could help refine classification performance even further.