Università degli Studi di Milano

Data Science for Economics

Machine Learning Project

Image Classification with Convolutional Neural Networks

by
Giordano Vitale
Matriculation Number: 14310A

submitted to
Prof. Dr. Dario Malchiodi

June 5, 2024

Contents

Declaration

I declare that this material, which I now submit for assessment, is entirely my 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 work, and including any code produced using generative AI systems. I 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.

Any guides, tutorials, or other material I have taken inspiration from can be found in Chapter 6.

1 Introduction

1.1 Task

This project consists of deploying convolutional neural networks to classify whether an art piece belongs to one of the following art realization techniques: oil paintings, sculptures, mintings, and pencil-drawn paintings. Those are the target classes.

The project is implemented in Python using the TensorFlow framework 2.15.0 in the Google Colab environment. The entire code can be found in this GitHub repository.

The choice of network architecture, as well as its advantages and disadvantages, is scrutinized thoroughly in the subsequent chapters.

1.2 Dataset

The dataset considered for this project consists of a table containing several features regarding artworks belonging to The Prado Museum, among which we can find the URL of the image referring to the art piece. In order to retrieve the images from their corresponding URLs, and subsequently assign them to each respective class, a parallel downloading procedure is applied.

The original, complete dataset collects 13,487 artworks, realized with 841 different techniques. To obtain a consistent final dataset, the 4 most present classes only are kept.

The final dataset structure is captured in the table below:

	Oil	**Minting**	**Pencils**	**Sculpture**	**Total**
Train	3,150	916	360	322	4,748
Test[1]	792	202	115	78	1,187
Total	3,942	1,118	475	400	**5,935**

Table 1.1: Dataset Representation

Once the data is obtained, the images are converted into tensors of shape (224,224,3). The first two dimensions (224,224) define the pixel size of the image, whereas the third dimension refers to the RGB color channels. The color's brightness in each channel is a value ranging between 0 and 255. To foster model comparability, the same input tensors' shape, as well as the same batch size of 32, are adopted.

[1] Also called validation set, especially in the code

1.3 Data Preprocessing

Preprocessing data is an essential step, as it adds variety to the training data, aiding the model in generalizing more effectively to new examples and thereby decreasing the risk of overfitting.

All the adopted preprocessing layers, hereby described, have been selected and studied in the TensorFlow guide to data augmentation.

Random Flipping & Random Rotation

Flipping has the role of helping the model to detect certain patterns, that exist independently from the image orientation. In the proposed model, horizontal and vertical random flipping is applied.

Similarly, Random Rotation rotates the image by a given factor. In all the proposed models, a factor of 0.2 is applied. This means that the image may be rotated, either clockwise or counterclockwise, by a random amount in the range $[-20\% \cdot 2\pi, +20\% \cdot 2\pi]$.

Both these techniques allow the model to be more robust to different orientations in the input data.

Random Contrast

This technique affects the original image by randomly modifying its contrast, according to a specified factor. In the proposed architectures, the factor is set to 0.2. The main idea behind this method is to ensure that the model does not rely excessively on contrast variations, instead of focusing on other main features, during the convolutional phase.

Random Zoom

This layer randomly zooms in or out, according to the passed factor value. A positive factor value results in out-zooming, whereas a negative value produces the opposite. In this project, a factor of 0.3 is chosen: the zoom will occur in the range $[-30\%, +30\%]$.

The core aspect of zooming is to equip the model with the ability to capture patterns regardless of the scale of the input images.

Per-Image Standardization

In this layer, the image gets standardized such that its pixel values across all three color channels will have a mean value of 0 and a variance of 1.

The standardization is performed on a per-image basis, meaning that the resulting image is obtained by exploiting the statistics of the original image itself. This process can be described formally using the following formula:

$$image_{std} = \frac{image - \mu}{sd_{adj}}$$

where μ denotes the mean of all the pixel values in the image and $sd_{adj} = \max(sd, 1/\sqrt{N})$[2].

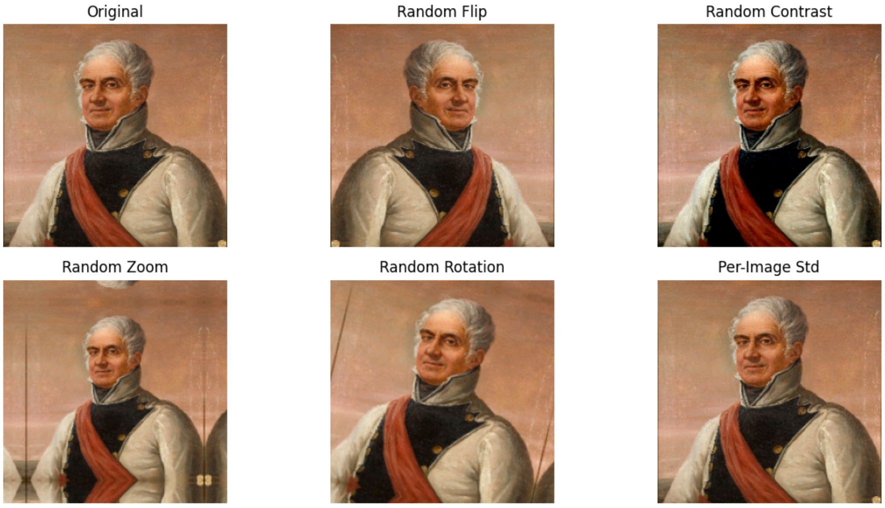


Figure 1.1: Examples of data augmentation results

1.4 Class Imbalance

By inspecting Table 1.1 a severe class imbalance is observed. In fact, the class *Oil* represents 66%[3] of the training set. As a consequence, the model will be particularly biased towards the most prevalent class. This learning behavior would lead the model to perform well on the most prevalent class while failing in predicting the classes with fewer instances.

Upweighting is the solution proposed to handle this problem. It consists of assigning a higher weight to classes having fewer observations, whereas the most common class will receive a lower weight, to compensate for the disproportion.

	Oil	**Minting**	**Pencils**	**Sculpture**
Original Proportion (%)	0.66	0.18	0.09	0.08
Upweighted	1.52	5.64	11.45	13.24

Table 1.2: Class Imbalance: Upweighting less frequent classes

[2] sd is the standard deviation of all the pixel values in the image and N is the total number of pixels.
[3] $3,150/4,748 \approx 0.66$

1.5 Execution Specifications

The loss function considered during the training phase is *Sparse Categorical Crossentropy.* It not only considers whether an image is properly classified, but also assesses the distance between the real label and the provided prediction.

The Adam optimizer, introduced by Kingma and Ba (2014), is chosen as the algorithm for the training optimization.

Every proposed model is run for 50 epochs, enabling their comparison, as this number represents a good trade-off between the model's performance assessment and computational time.

2 ZFNet

The first model architecture has been inspired by following Zeiler and Fergus (2013)'s pioneering work on neural networks. The abovementioned paper was published with the goal of improving the results achieved by Krizhevsky et al. (2012) on the ImageNet competition. Due to limits in computational resources, the proposed architecture has been slightly modified, making it simpler yet coherent with the authors' implementation. All the differences will be properly mentioned.

Although the original paper doesn't mention any data augmentation techniques, in this project's implementation the model can rely on all the pre-processing layers mentioned in section 1.3. This certainly empowers the model to generalize better.

2.1 Architecture

The model architecture consists of 3 convolutional blocks. They all have the same *GlorotUniform* weights initializer and have the goal of capturing the patterns of the input data. The first block is formed by one convolutional layer with 48 filters[1], kernel size (7 × 7). It is followed by a ReLU activation function, which allows the model to detect non-linearity, and its last component is a (3×3) MaxPooling layer which performs dimensionality reduction and helps reducing the risk of overfitting.

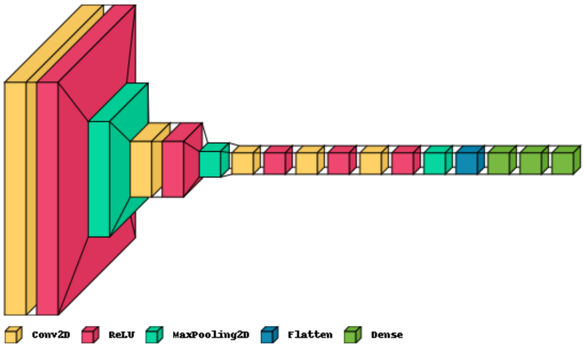


Figure 2.1: ZFNet Architecture

The second block replicates the previous one, except for the number of filters, which is set to 128[2]. This choice reflects the idea of capturing an increasing number of features of the input data.

[1] The original number of filters was 96

[2] The original number of filters was 256

The third block represents a novelty, as it is formed by multiple convolutional layers - and their corresponding activation functions - stacked one after another. This block comprises a total of three convolutional layers having 192, 192, and 128 filters, respectively[3]. Like the previous convolutions, they are followed by ReLU activation functions. At last, a (3×3) MaxPooling layer concludes the third block.

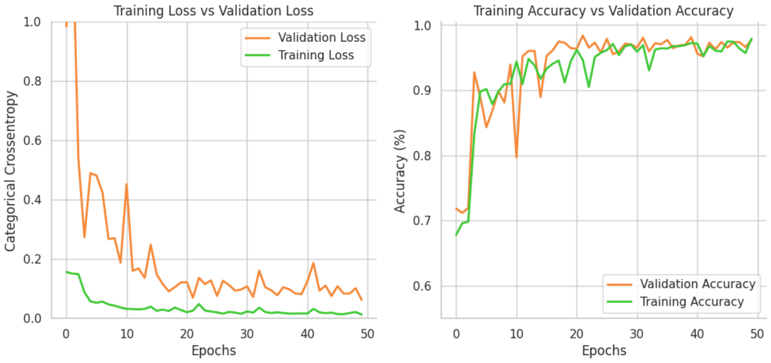
After the main body of the network is defined, a Flatten layer converts the output tensors of the previous block from 3D to 1D.

The last stage of the model consists of a series of 3 fully-connected layers, with 256, 256, and 4 neurons representing the number of classes, respectively[4]. While the first two fully-connected layers are followed by the usual ReLU function, the last one is followed by the Softmax activation function, which ensures that the output associated with each neuron represents the probability of belonging to that corresponding class.

2.2 Results

We can observe that the model offers an encouraging behavior, both in terms of the loss trajectory and when inspecting the overall accuracy. In fact, on the left-hand side of image 2.2, the validation loss quickly converges towards zero. After some fluctuations in the first 10-15 epochs, the trajectory follows a consistent downward trend.

Figure 2.2: ZFNet Results



As regards the accuracy, visible on the right-hand side, the performance on unseen data mimics the trajectory observed for the training data, and it converges towards 1.

It is possible to conclude that the model avoided overfitting, hence it effectively grasped the most important features in discerning the image's class.

Both these results suggest an overall positive capability in generalizing well on new data.

[3] The original number of filters were 384, 384, and 256

[4] The original number of neurons were 4,096, 4,096, and 1,000, respectively

3 GioNet

The next model's architecture is introduced to improve the already-pleasing results scored by ZFNet. To achieve this pursuit, the new architecture will be affected by diverse experimentations, that are deployed to investigate whether (and which) newer elements could potentially enhance the overall performances. All the novelties introduced, as well as the maintained specifications, are thus discussed in the following sections.

Moreover, MaxPooling is replaced by AveragePooling as pooling technique to avoid overfitting and achieve dimensionality reduction. Another important difference is given by the kernel size specified throughout the architecture: they follow an increasing sequence, which reflects the idea of capturing the micro patterns first - hence smaller square matrices - and then the macro patterns in the subsequent convolutional blocks - hence bigger square matrices rolling over the images.

Specifications for padding, strides, activation function, and weight initializers in each convolutional layer are the same as discussed in chapter Chapter 2.

3.1 Architecture

Similar to ZFNet, all the pre-processing steps regarding data augmentation described in section 1.3 are retained.

The first novelty is the network's dimension: this second model is more complex and is made up of more convolutional layers. The first convolutional block is made up of a convolutional layer having (3×3) kernel size, then it is followed by a ReLU activation function, and the third component is an AveragePooling layer of shape (3×3). The second block replicates the first one, with the exception of halving the number of filters.

The third block aims at capturing broader patterns, and this criterion is captured by the bigger (5×5) matrix defining the kernel size. The number of filters is further halved to 56.

The fourth and last convolutional block's objective is to identify even broader patterns. With this in mind, the kernel size is defined by a (7×7) matrix, while the number of filters is set to 28.

Differently from the ZFNet implementation, the Flatten layer is followed by two fully-connected layers only. The first one comprises 64 neurons and is subject to the ReLU function, whereas the second one represents the output layer, hence it is made of 4 neurons representing the probability for the image belonging to that given class.

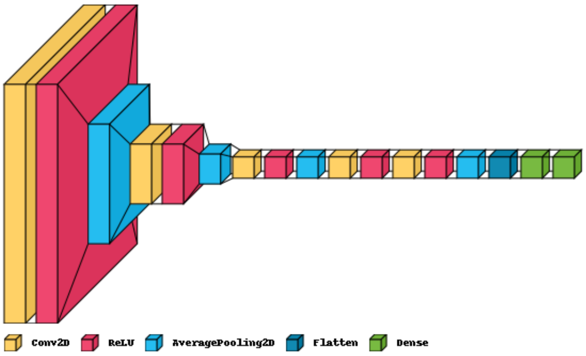


Figure 3.1: GioNet Architecture

3.2 Results

This model offers a disappointing overall performance when looking at the validation loss and accuracy scores.

Although there is no noticeable divergence between the training and validation loss trajectories, the validation loss suffers from severe fluctuations. Overall, it seems to have reached its best validation loss at the 15th epoch, scoring below 0.18. After this epoch, the model is capable of occasionally going back to that score, but with significant instability. This is a clear setback when compared to the ZFNet, since it offered a steady, solid validation loss trajectory converging to the training loss close to 0.

Likewise, when focusing on the accuracy on the right panel of image 3.2, significant unstableness is evident. While the overall accuracy repeatedly exceeds 0.95, thus showing some good potential, it also falls below 0.90 in a couple of epochs. Although this is not a terrible result in general, it is not a desirable behavior as it represents a suboptimal solution.

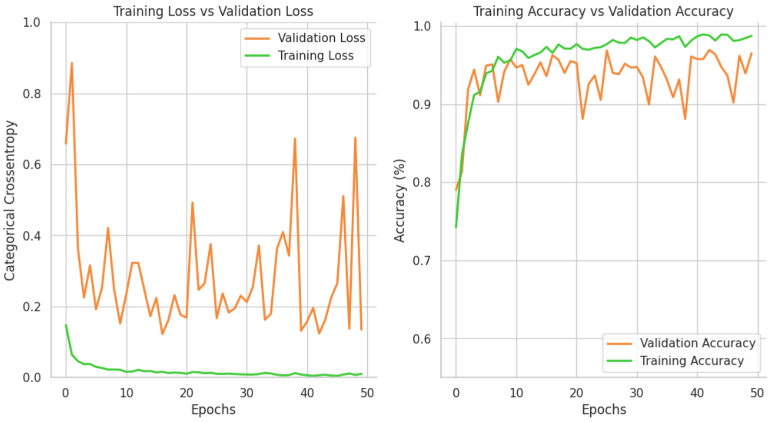


Figure 3.2: GioNet Results

4 GioNetV2

The model explained in Chapter 3 left several opportunities for improvement. Potential causes of its disappointing results can include inadequately specified regularization techniques, inefficient initialization of weights, and the absence of a learning rate schedule, among other factors.

With this in mind, the third model discussed in this project took GioNet and tried to improve it, to make it more solid and potentially overcome ZFNet's performance.

4.1 Architecture

This model differs in a variety of key aspects. The first novelty lies in the preprocessing steps: while the previous two models had all the techniques mentioned in Section 1.3, this architecture only relies on RandomFlipping, RandomContrast, and PerImageStandardization. The reasoning behind this choice is to simplify the input data, thus avoiding potential over-complication of the input, which can potentially result in excessive introduced noise.

One significant innovation in this architecture is represented by BatchNormalization, which replaces AveragePooling in playing the role of regularization term pursuing an overfitting-free model.

Moreover, a different weight initializer is adopted. More precisely, the *He Normal* initializer, proposed by He et al. (2015), had proven to be more precise and reliable when building neural network set-ups with ReLU activation functions. Simultaneously, they showed that the *Glorot* initializer is more properly suited for neural network models that make use of the *tanh* activation functions.

One additional noteworthy variation is represented by the introduction of a learning rate scheduler, whose aim is to stabilize the model's performance and contribute to avoiding overfitting. It is defined as follows:

$$\text{Learning Rate} = \begin{cases} 0.001 & \text{if epoch<10,} \\ 0.001 \cdot e^{-0.005} & \text{if epoch=10,} \\ lr \cdot e^{-0.005} & \text{if epoch>10} \end{cases}$$

The model is formed by three main parts. The first is simpler with respect to its subsequent counterparts, and it comprises a convolutional layer with a more limited number of filters to begin with, namely 32, and a large (7×7) kernel size, whose objective is to capture broad patterns of the images.

The second part exhibits more complexity. In fact, it consists of two convolutional blocks[1], each formed by a convolutional layer with an increased number of filters to 64, and a smaller kernel size of 5 × 5, followed by BatchNormalization and then the ReLU activation function. Its role is to identify less broad patterns of the image.

The third and last element replicates the two-block sequence of its previous companion, but with a further increase in the number of filters, namely 128, and a smaller kernel size of (3 × 3).

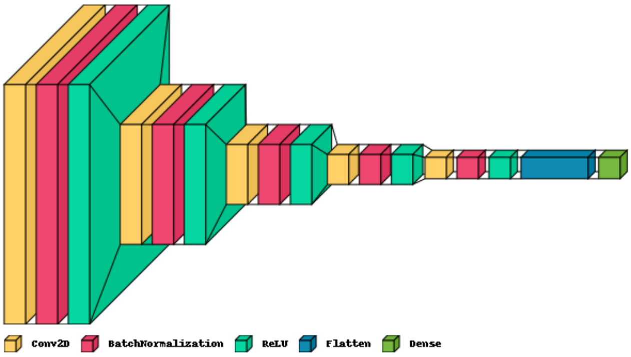


Figure 4.1: GioNetV2 Architecture

After the usual Flatten layer that prepares the data for the final stage, there is only one dense layer, which represents the final output consisting of 4 neurons, as many as the classes.

4.2 Results

This model represents the best-performing one under both the validation loss and accuracy aspects. In fact, it yields remarkable results, establishing itself as the state-of-the-art among those proposed.

While the validation loss starts with high peaks reaching 1.6, it subsequently stables and steadily converges towards the training loss. After the 20th epoch, the validation loss consistently stays in the vicinity of 0.05.

Moving to the right panel of image 4.2, it is observable that the validation accuracy starts from very high values, differently from the previous models. Moreover, it doesn't suffer from volatility, and this represents a great achievement, because this was the primary goal after what has been observed in GioNet. From the 15th epoch onwards, the validation accuracy stabilizes around 99%.

[1] The same block gets repeated twice. This holds for the third part of the model as well.

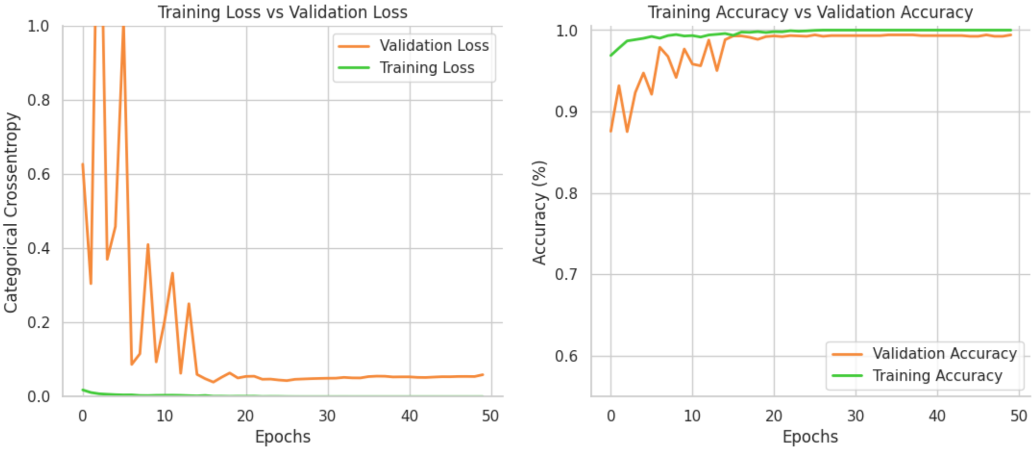


Figure 4.2: GioNetV2 Results

5 Conclusion

5.1 Overall Comparison

Two simple plots are reported in image 5.1. On the left-hand side, it is possible to compare the models' results regarding the validation loss they respectively recorded. Again, GioNetV2 outperforms its counterparts by reaching the absolute minimum around 0.05. On the right-hand side, instead, the validation accuracies are compared. While ZFNet achieved desirable results, GioNet registered poor results, consisting of a worse validation accuracy and a higher, unstable validation loss. At last, the updated model GioNetV2 demonstrated superior results, closely approaching the 100% accuracy on unseen data.

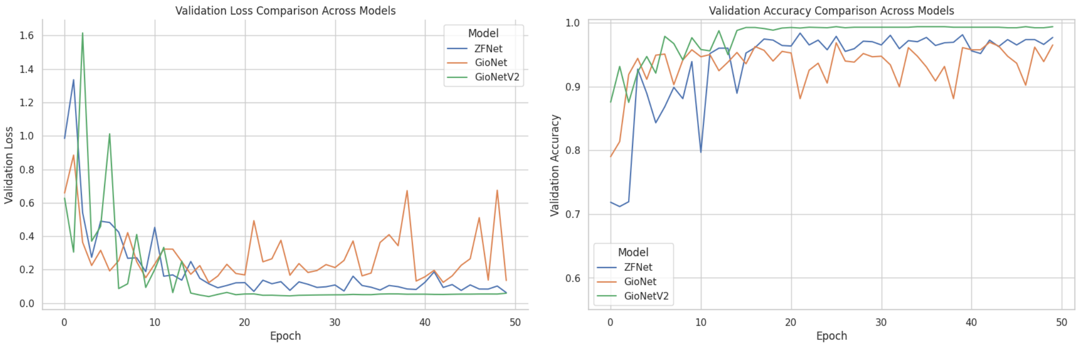


Figure 5.1: Models Comparison, Loss and Accuracy

5.2 General Evaluations

The main insights that can be highlighted from the proposed experiments are:

- Multiple subsequent convolutional layers can improve results through the additional non-linearity of the activation function.
- BatchNormalisation proved to be an extremely accurate regularisation method, which significantly reduced the risk of overfitting.
- The He initializer is preferable to the Glorot initializer, as it plausibly contributed to achieve the phenomenal results of GioNetV2.
- The learning rate scheduler is an effective and useful tool when dealing with volatility in the model results.

- A shallower, yet accurate data augmentation preprocessing phase may be associated with more robust results, as shown by GioNetV2.
- Including several fully-connected layer might be redundant in terms of performances, and costly in terms of model complexity.

5.3 Potential Improvements

A significant area for improvement is hyperparameter tuning. Due to constraints on computational resources, hyperparameter tuning was not conducted. It is acknowledged that a more in-depth examination of multiple hyperparameters could yield additional insights and more robust hyperparameter estimations, potentially leading to further enhancements in model performances.

6 Resources

The following list comprises all the significant resources that have been followed during the implementation of this project.

- Data Augmentation Tutorial by TensorFlow
- Convolutional Neural Network Tutorial by TensorFlow
- ZFNet model explanation from Medium
- Learning Rate Scheduler by Keras
- Handling Class Imbalance by TensorFlow Layer Wight Initializers by Keras

Bibliography

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901.