PAINTING'S STYLE RECOGNITION

Group 3

Ana Carrelha (20200631) Inês Melo (20200624) Inês Roque (20200644)

Introduction

Nowadays the art of painting is a wide and well-known area in the world. Years pass and more painting styles appear, making difficult the task of identifying with certainty which is the style of a painting just by looking at it. Our purpose is to develop a model that helps with this problem. With this project, we want to build a Convolutional Neural Network model that can distinguish painting styles.

Task Definition

Identifying a painting style is a difficult task even to the human eye, given that we have a large number of existing painting styles and that some of them are similar. For that reason, our idea could help anyone classifying the style of a painting in seconds while without it, even for the experts, this could be a harder task. Application examples of this project could be art schools, museums, galleries, or even someone with a smartphone that wants to know the style of a painting seen somewhere.

Due to computational costs associated with this task and the number of existing painting styles, we have decided to identify the painting category between four options: Impressionism, Romanticism, Realism, and Expressionism. These styles were chosen because they were the most common styles in our data.

Our Data

For this project, we used a dataset with a large number of painting images that we found in the scope of Kaggle's challenge whose objective was to identify if two paintings were from the same artist. Even though this task was different from ours, we took advantage of the dataset for our purpose. To filter the data and save only the images that we were interested in, we relied on a csv file with some properties about those paintings, including their style and the name of each picture.

From the initial train dataset, we made our train and validation datasets, with 24698 and 4000 images, respectively, so that our validation data could have 1000 samples for each category. From the test, we ended up with 8766 images. The cleaned dataset that we have used is available on one drive.

https://liveeduisegiunl-my.sharepoint.com/:u:/g/personal/m20200624_novaims_unl_pt/EbfJTviqnLBPuGoiAwxYsFUBi-cY2tLDMVRTbCNHeiEZ_w?e=DerfaW

	Impressionism	*Romanticism*	*Realism*	*Expressionism*
Train	7220	6041	7112	4325
Validation	1000	1000	1000	1000
Test	2423	2244	2411	1688

Table 1- Distribution of images between train, validation, and test datasets

Evaluation measures

In the scope of our problem, it is not important in which class the model is failing (this is, for us it does not make any difference if the model wrongly predicts a Realistic or an Impressionistic painting, for instance) and for that reason, we opted to evaluate the performance of our models using the accuracy measure.

Besides the accuracy, we also based our decisions on the running time and computational costs.

Approach

As we are dealing with a multiclass image classification problem, we used the *categorical crossentropy* loss function and the *softmax* activation function to train our models.

Between all our attempts of constructing better models, we decided to explain in more detail the four that we found more relevant to our final solution, leaving the others in the notebook.

- **Approach 1**

At the beginning of the process of building and training our models, we started by creating a more general structured model that we have done in classes.

Training this model, the optimizer chosen was the RMSprop with a learning rate of 0.0001. In terms of data pre-processing, we resized the images to 150x150 and chose a batch size of 246 due to the fact we have a very large number of images, and also because it is more computationally practical.

We first tried to fit the model with the batch generator with 60 epochs, but we understood that the running time would not be efficient, so we changed the number of epochs to 30.

We obtain the following results:

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_4 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_5 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_5 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_6 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_6 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_7 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_7 (MaxPooling2	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_2 (Dense)	(None, 512)	3211776
dense_3 (Dense)	(None, 4)	2052

Total params: 3,454,660
Trainable params: 3,454,660
Non-trainable params: 0

Figure 1- Summary of network's layers

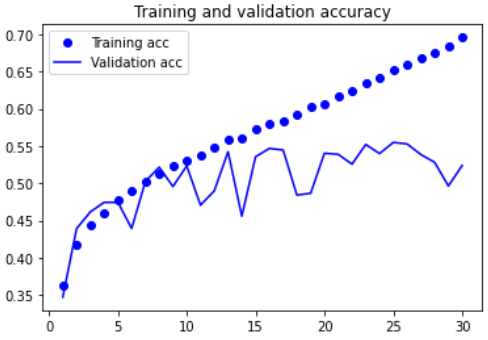


Figure 2- Training and validation accuracy, approach1

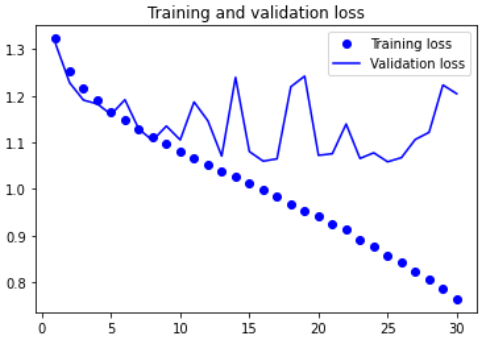


Figure 3- Training and validation loss, approach1

Observing the results, we were able to conclude that we have made a good decision about the number of epochs because we can clearly see that the evolution of values would not improve. Also, when the model presented good values for accuracy, showed a lot of overfitting.

- **Approach 2**

To improve the overfitting of the previous network, we added a Dropout layer after the Flatten layer. To avoid the decrease in the accuracy, we also trained the Convolutional Neural Network using data-augmentation generators so our model could use more perspectives of the same image. Expecting better results, we opted to increase the number of epochs to 40 so we could understand if that increase would be worth the running time and computational cost.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_4 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_5 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_5 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_6 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_6 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_7 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_7 (MaxPooling2	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dropout_1 (Dropout)	(None, 6272)	0
dense_2 (Dense)	(None, 512)	3211776
dense_3 (Dense)	(None, 4)	2052

Total params: 3,454,660
Trainable params: 3,454,660
Non-trainable params: 0

Figure 4 - Summary of network's layers

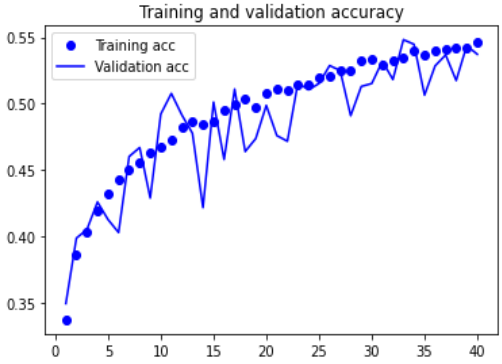


Figure 6 - Training and validation accuracy, approach2

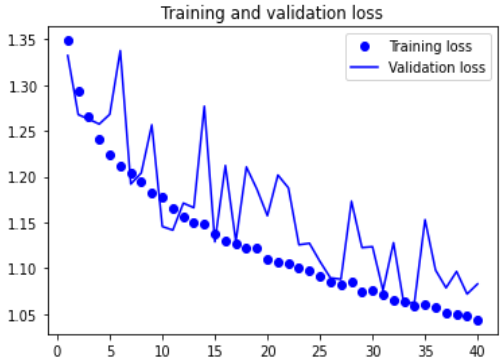


Figure 5 - Training and validation loss, approach2

As we expected, we saw a decrease in overfitting, but with this new approach, the accuracy also decreased. Observing the curve's evolution throughout the epochs, we deduced that increasing the number of epochs would improve the accuracy, but the value of that increase would not worth the difference in time and computational costs associated.

- **Approach 3**

Trying to increase the accuracy of our model, being computationally efficient, we used the VGG16 pre-trained network, so it could extract interesting features from a pre-trained model with the weights loaded and without the fully connected layers and then train our small fully connected network on those extracted “bottleneck features”.

Layer (type)	Output Shape	Param #
flatten_17 (Flatten)	(None, 25088)	0
dense_38 (Dense)	(None, 512)	12845568
dropout_20 (Dropout)	(None, 512)	0
dense_39 (Dense)	(None, 50)	25650
dropout_21 (Dropout)	(None, 50)	0
dense_40 (Dense)	(None, 4)	204

Total params: 12,871,422
Trainable params: 12,871,422
Non-trainable params: 0

Figure 7 - Summary of network's fully connected layers

Creating these “bottleneck features”, we used a target size of 224 x 224 (VGG16 requirement) and a batch size of 50 (given that it is a pre-trained model, it is not necessary to have a large batch size).

Exploring this pre-trained network, we tried to develop models using VGG16 with several combinations of fully connected layers, never achieving better results than in the previous approach (comparing both the overfitting and the accuracy).

We leave below two examples of our attempts, one with the RMSprop optimizer and the other with the Adam optimizer.

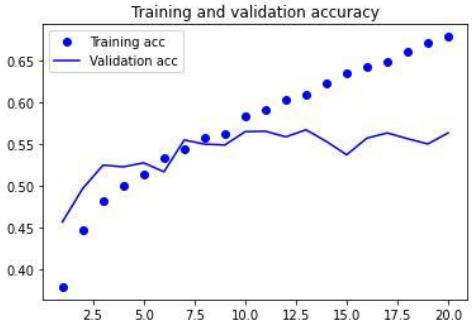


Figure 9 -Training and validation accuracy, approach3 with RMSprop optimizer

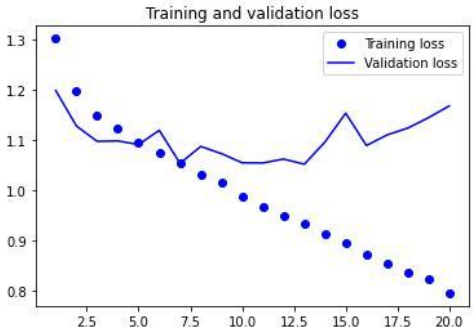


Figure 8 - Training and validation loss, approach3 with RMSprop optimizer

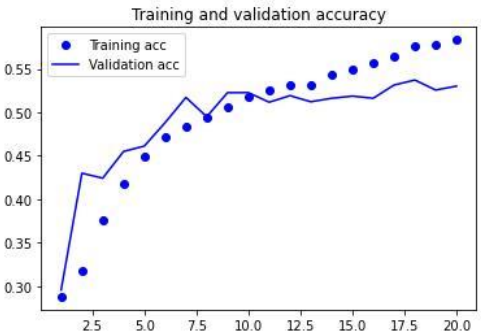


Figure 11 - Training and validation accuracy, approach3 with Adam optimizer

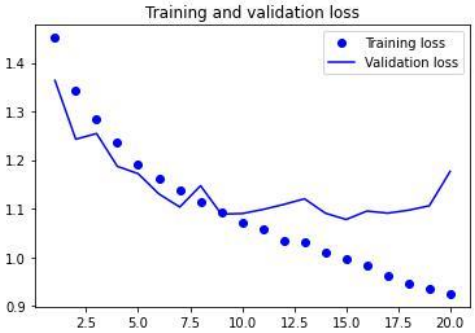


Figure 10 - Training and validation loss, approach3 with Adam optimizer

- **Approach 4**

Since the bottleneck features did not give us better accuracy results than in the second approach, we decided to return to that approach to improve it.

We did that by changing the optimizer from RMSprop to Adam and increasing the number of epochs, obtaining the following results:

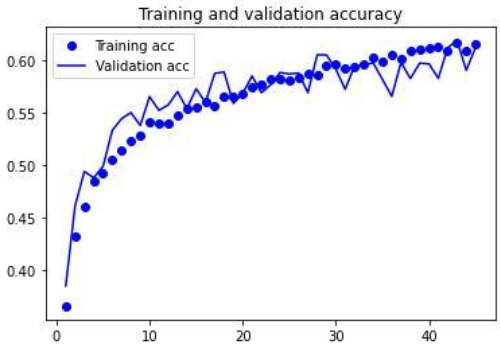


Figure 13 - Training and validation accuracy, approach4

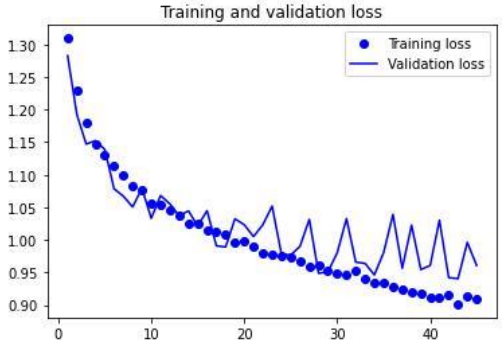


Figure 12 - Training and validation loss, approach4

These values for accuracy, overfitting, and loss were the best that we achieved and for that reason, this is the model that we chose as the final one.

Error Analysis

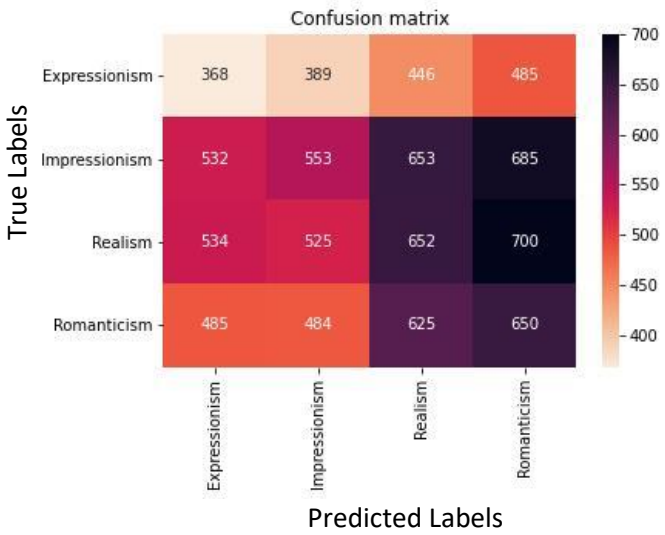
Using the model explained in the fourth approach, we performed the predictions on our test dataset, obtaining an accuracy of 58.44%.

```
test_loss = modeltop.evaluate(test_generator)
```

```
36/36 [==============================] - 199s 6s/step - loss: 0.9965 - accuracy: 0.5844
```

Figure 14 - Model evaluation (accuracy and loss results)

To better understand how our model behaved on the test data and to realize which were the classes that our network predicted better, we designed a confusion matrix to show the relation between true and predicted labels.



Observing the matrix, we can understand that all the classes were more mistaken for Romanticism. In general, we believe this makes sense because Romanticism seems to be a very wide style, including different colors, representations, and genres.

These characteristics may be the reason for so many misclassifications as Romanticism.

Figure 15 - Confusion Matrix with true and predicted labels

Trying to understand some of these misclassifications, we leave here 4 images classified by our model as Romanticism, but only the last one is really a Romanticism painting. Looking at these pictures we can see some aspects in common, so we can realize the difficulty of this task.

Figure 17- Impressionist Painting

Figure 16 - Expressionism Painting

Figure 18 - Realism Painting

Figure 19 - Romanticism Painting

Conclusions

We believe art can be a wide subject in the sense that art sometimes is what that piece makes you feel and not exactly what you see in it. Recognizing a style sometimes requires the knowledge of art history or the artist's style itself. Even inside a style, we can find many different genres (like abstract, landscape, etc.) and so there are a lot of representations for just one style.

After doing this project, we realized that maybe recognizing an art style can be a hard and complex task for a computer. Maybe if we have chosen more unique styles with less common characteristics between them, our model could have had a better performance. The model performance was also negatively influenced by computational resources.

Despite all of this, this project helped us realize the practical usefulness of Deep Learning, specifically Convolutional Neural Networks models, in our world now, and even more in the future, and sparked our interest in this subject.

References

- Persson, Siri (March 6th, 2018) - ***Application of the German Traffic Sign Recognition Benchmark on the VGG16 network using transfer learning and bottleneck features in Keras***, https://www.diva-portal.org/smash/get/diva2:1188243/FULLTEXT02

- Browniee, Jason (September 21st, 2016) - ***How To Improve Deep Learning Performance***, https://machinelearningmastery.com/improve-deep-learning-performance/

- Browniee, Jason (May 15th, 2019) - ***Transfer Learning in Keras with Computer Vision Models,*** https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/

- https://gist.github.com/echen/2b6ffa9b1f461698cce537e694e451f8

Initial dataset source:

- https://www.kaggle.com/c/painter-by-numbers/data