Style Based Art Classification

Rana Shahzaib Ur Rehman
rrehman.bee21seecs@seecs.edu.pk
SEECS, NUST, Islamabad

15 February 2022

Abstract

Recognizing art style can be used for arranging and collecting art data effectively and efficiently. This paper proposes a way of executing this task along with the data analysis and experimental results. I chose to use ResNet50 and MobileNetV2 for final training and testing. Wiki-paintings was used as the source of data for this task. I was able to achieve the validation accuracy of 60.00% after setting the chosen hyperparameters empirically. I also concluded why the accuracy for this dataset and perhaps for art classification, in general, is very low.

1 Introduction

For this task, I used state-of-the-art Residual Neural Networks. It has been noted that traditional CNNs are not that effective in such tasks and Residual Neural Networks (ResNets) outperform them in almost every case. ResNets were introduced back in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their paper in Pattern Recognition at Cornell University. In the very paper they said:

> Residual networks are easier to optimize, and can gain accuracy from considerably increased depth.

ResNets have architectural improvements from traditional convolutional neural networks. Their implementation includes skip connections via addition which allows the output of one layer to be fed to some other deeper layer skipping the layers in between.
We need some potential information from art images that can be used to differentiate between two classes. Considering the diversity of art in a specific art style, the task is not trivial. One approach may be to extract gradient features using different standard methods like Dense SIFT and HOG as proposed by Alexander Blessing and Kai Wen in this paper. To start with, we let the neural network figure out the necessary details on its own.

2 Approach

Images are merely 2x2 grids of 3 Channel(RGB) pixels and intensity values of each channel can be used to manipulate images and extract vital information for classification. The very first step is to get the datasets containing enough images for training.

2.1 Image Dataset

For the image dataset, I used the data available at wikiart.org. The complete dataset includes images for around 300 different art styles but the number of images per class is not constant. For the purpose of demonstration, I used 25 top classes with the most images making a total of 82,065 images. Art styles are organized in separate folders having names corresponding to their styles so that labeling could be done easily. Furthermore, this dataset is divided into three subclasses namely, training, validation, and testing datasets containing around 66,528, 7,383, and 8,154 images respectively. Model is trained on the training dataset while validation dataset is used for getting validation accuracies during training and finally, the testing dataset is used to get predicted results. The model never trains on the validation and test datasets so their accuracies can be considered credible.

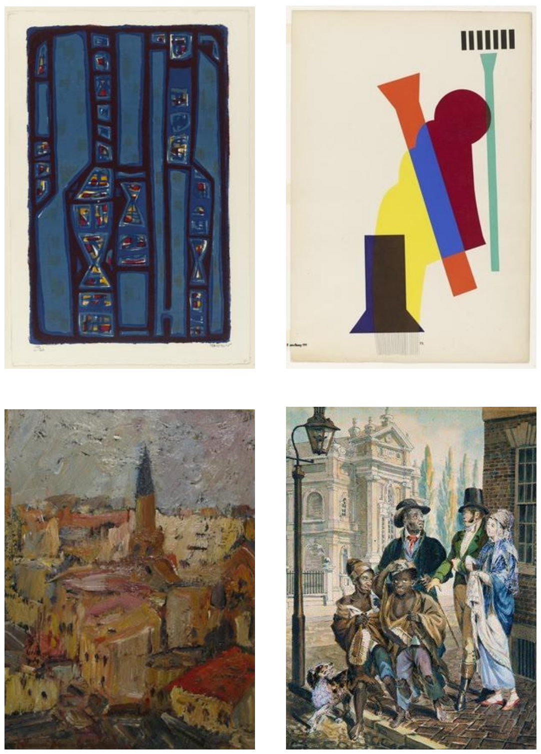


Figure 1: Sample images from dataset. Art_Informal(top left). Abstract Art(top right). Impressionism(bottom left). Realism(bottom right)

The data is not distributed evenly over all classes rather some classes have more images than others. A histogram of a number of images with respect to each class is given below:

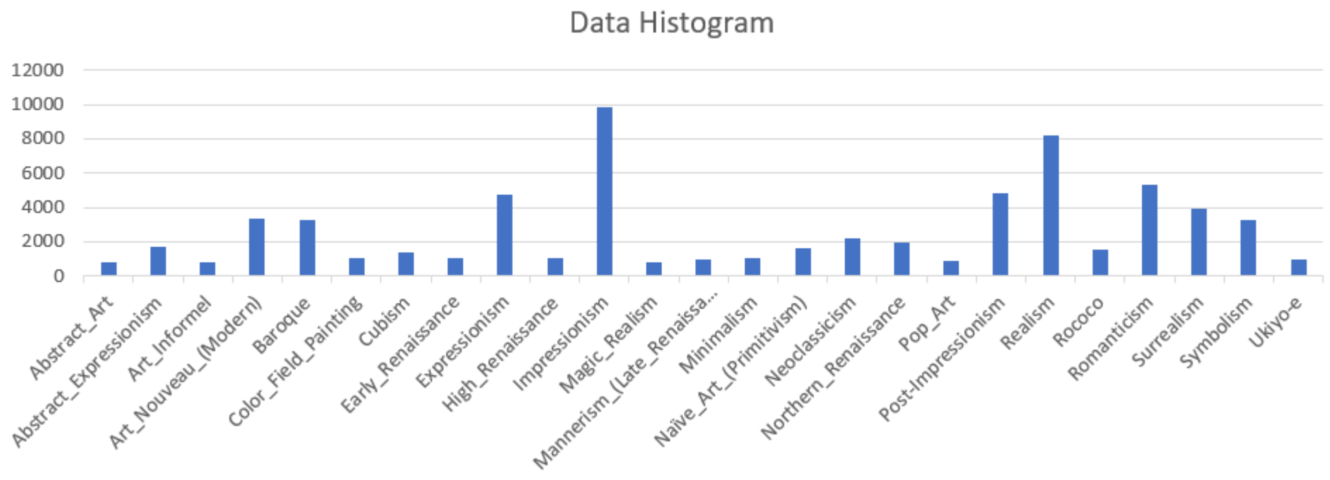


Figure 2: Distribution of images in different classes

2.2 Neural Networks

I used two different models with almost the same architectures namely MobileNetV2 and ResNet50 for training. While ImageNet is faster, ResNet is considered deeper and learns better. Both are available in Keras applications with pre-trained weights from ImageNet.

2.2.1 ResNet50

It takes input image arrays in shape 224 × 224 × 3 and comprises multiple identity and convolutional blocks with shortcut or skip connections implemented after some layers. The size of the filter increases from 64 to 2048 as we go deeper. The activation function for all layers other than the fully connected top is Rectified Linear Unit (ReLU). For the top dense layer, softmax activation is used which classifies the output of the model into classes. Elaborated architecture can be found in the code.

2.2.2 MobileNetV2

For comparison, I also used MobileNetV2, a convolutional neural network architecture with pre-trained imagenet weights. The architecture is quite similar, only it is faster to compute but potentially lags in accuracies. The input dimensions are $160 \times 160 \times 3$. Also, the images need to be rescaled between 1 to 128 as a part of preprocessing. Similarly, the activation function for all layers except the top layer is ReLU for which softmax activation is used.

3 Implementation

3.1 Keras

Keras is a python based open-source library which works as an interface for machine learning tasks. It comes with many useful APIs which are very handy and efficient. For this task, I have mainly used the standard library code rather than implementing my own code for every function. Keras supports multiple backends but I chose to prefer TensorFlow.

3.2 Pre-Processing

All data needs to be preprocessed when using ImageNet weights as they were trained in BGR. So we have to convert every image from RGB format to BGR. To avoid repetition and make the user oblivious of this fact, it was made a part of the model pipeline. Moreover, the data needs to be in the shape $224 \times 224 \times 3$ so every image was reshaped into these dimensions and this too was made a part of the model.

3.3 Data Augmentation

A very crucial characteristic of any machine learning model is that it should be able to identify images in different orientations. For example, flipping an image about any axis should not change its art style. To counter this problem, we introduce randomness in data in terms of orientation. The images are first flipped and rotated randomly only then they are fed to the model. This increases the accuracy and genericness of our model.
It can also be used when the dataset is small and we do not have enough images of each class to generalize the scenario. The model might overfit when data is very limited. Data augmentation comes in handy in such cases and can help increasing the dataset without repeating the data hence potentially preventing overfitting.

3.4 Pre Training

Initially, the model was trained from scratch by initializing random weights. Following the literature, I also tried training on models pre-trained on ImageNet datasets which were initially able to classify images into 1000 classes. I used the weights from the pre-trained model i.e., base-model, and replaced the last classification layer with my own dense layer to classify into 25 art styles. The base model is not trained and said to be 'freezed' and only the last layer is trained to classify. In the data analysis section, I have compared the results from both methods.

3.5 Hyperparameters

We have established different techniques to increase our validation accuracy but the extent to which these techniques should be applied is yet to be determined. Turns out that there is no way to figure out the 'sweet spot' values beforehand. Only after hit and trial, do we get to know the right choice. We define such features as hyperparameters of our learning. In my case, I chose data augmentation, pretraining, and fine-tuning as hyperparameters.

3.5.1 Data Augmentation

We can nudge the degree of data augmentation to get the best results out of our model. For example, we can change the range of rotation of images to see how it affects the training. We can also remove the data augmentation completely to see for ourselves if it actually makes a difference.

3.5.2 Fine-tuning

After the top of a pre-trained model has been trained to classify our images, we can benefit from the deeper layers of the model by 'unfreezing' them and re-training them with our data. But the number of layers to train is to be found out by the hit and trial method. Hence it can also be set as a hyperparameter.

3.5.3 Learning rates

While updating parameters, the gradient descent is multiplied by a constant factor known as the learning rate. It scales the change in parameter values per iteration. Changing the learning rate can be fruitful for better accuracies. In our case, I set the learning rate to be 0.001 in initial training and 0.0001 in fine-tuning. Finding an optimum value by hit and trial might have increased the accuracies somewhat.

3.6 Callbacks

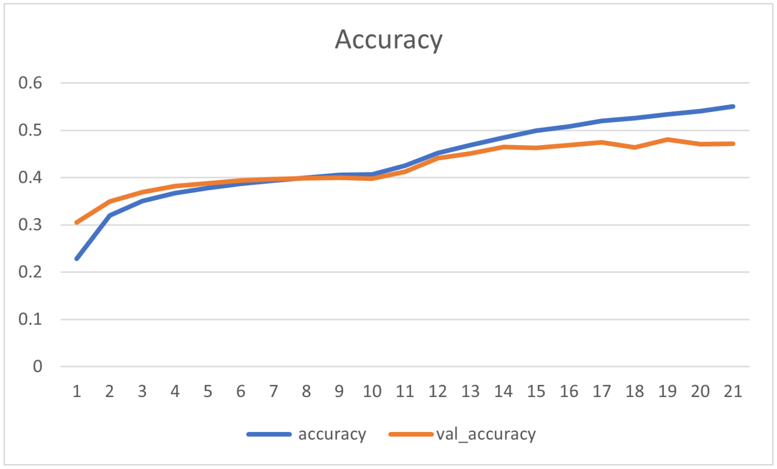
When the dataset is not big enough to handle the versatility of data, deep models try too hard to find some patterns in the data and they end up adapted to the 'noise' in the training data after some iterations. This can be easily identified when the training accuracies are way higher than the validation accuracies. To stop this from happening, we implement a mechanism called early stopping which stops training when the model starts to overfit. It waits for 5 epochs and monitors the validation loss. If it does not get any lower in 5 epochs, the training is terminated.
Some other callbacks implemented include CSVLogger and Checkpoint.

4 Experiments

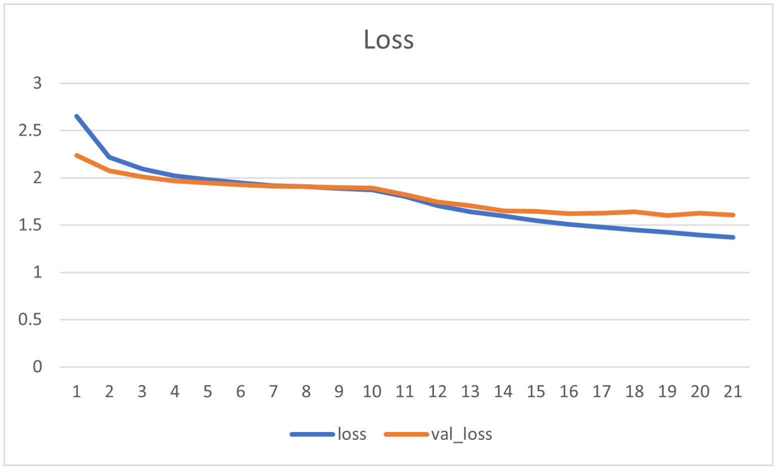
All models are trained using the image datasets introduced in section 2.1 with the implementation defined in section 3. The result and analyses for each experiment are given below.

4.1 MobileNetV2 with ImageNet weights

Initially, the model was set to train the top layer for 10 epochs and it yielded an accuracy of about 39.8%. Afterward, the top 100 layers were fine-tuned in 10 epochs on the same data. Finally, it yielded validation accuracy of 47.2% and testing accuracy of 47.3% on the test dataset. Following are the learning and loss curves:



(a) Validation and Training Accuracies



(b) Validation and Training Losses

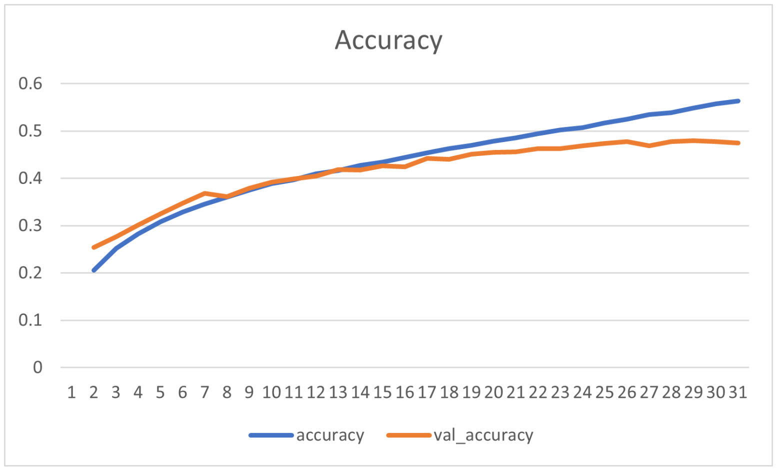
Figure 3: MobileNetV2 PreTrained

4.1.1 Analysis

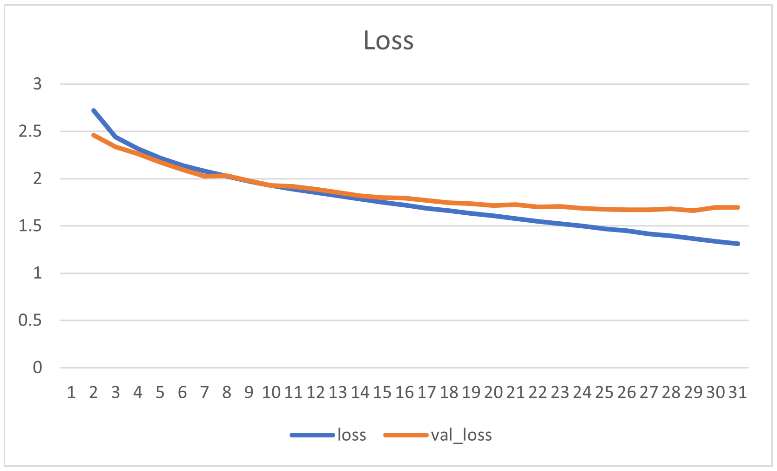
We can clearly see that fine-tuning top layers was quite fruitful since it increased the accuracy by about 8%. After the first 10 epochs, accuracy jumps from 39% to 47%. The model is not overfitting but the gradient has flattened out so running more epochs won't do any good.

4.2 ResNet50 trained from scratch

The model is built from scratch and initialized with random weights. The whole model is trained for 30 epochs on *train_dataset* to get 47.5% validation accuracy. The data plots are given below.



(a) Validation and Training Accuracies



(b) Validation and Training Losses

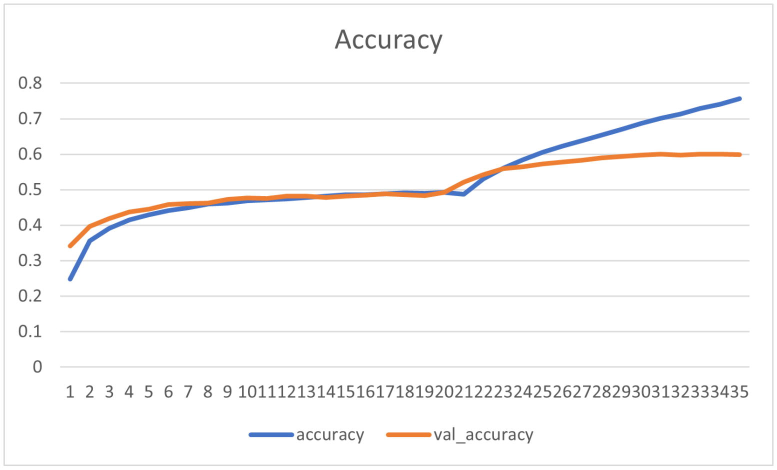
Figure 4: ResNet50 Trained from scratch

4.2.1 Analysis

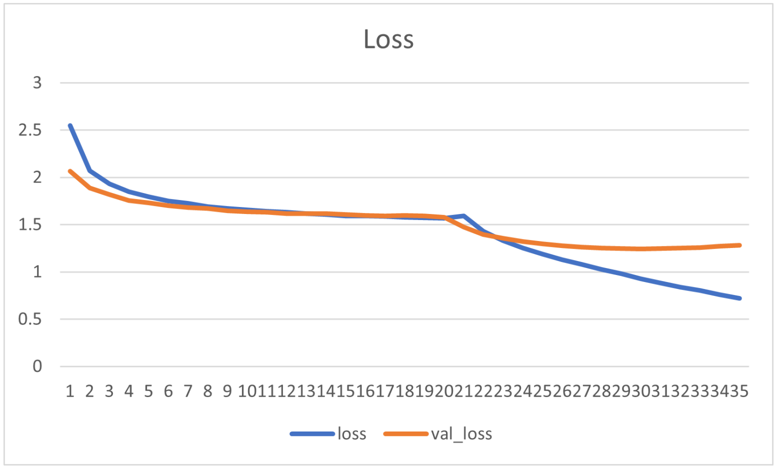
The graphs in Figure 4 show that validation accuracies have become more or less constant and the curve has flattened out. Looking at the training accuracy curve (in blue), it looks like the model has started to overfit at around 25 epochs. Even if more epochs are run, validation accuracy would not increase significantly.

4.3 ResNet50 with ImageNet weights

For the final test, I used a pre-trained ResNet50 trained on the ImageNet dataset to classify 1000 types of images and objects. The whole model is freezed and only top layer is trained. 20 epochs are run initially and it yielded an accuracy of 49.1%. After that, the top 30 layers are unfreezed and trained on the same data for another 20 epochs. After 13 more epochs, the training stopped with a peak validation accuracy of 60.00%. Training curves are plotted below.



(a) Validation and Training Accuracies



(b) Validation and Training Losses

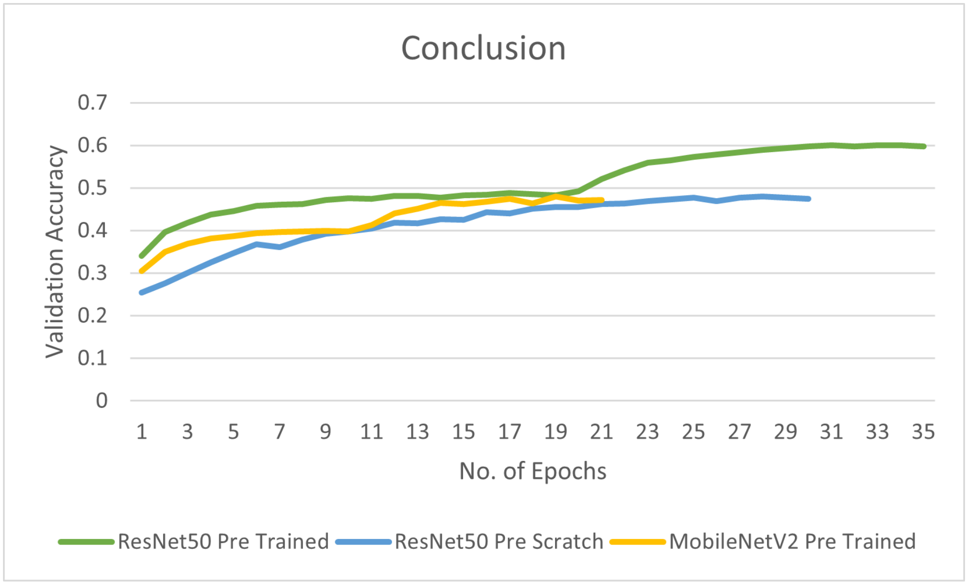
Figure 5: ResNet50 with ImageNet weights

4.3.1 Analysis

Figure 5 (a) shows that fine-tuning top 30 layers improved the accuracy by 10%. The data also shows that ResNet50 has outperformed MobileNetV2 by a margin. After 10 epochs, the validation accuracy for PreTrained ResNet50 was 47.6% while for PreTrained MobileNetV2, it was only 39.8%.
The curves also reveal that model quickly started to overfit in finetuning after about 25 epochs. That is why the model was trained for only 33 epochs while it should have run for 40 epochs. The callbacks implemented stopped training. This means that the dataset is the limiting factor here and ResNet can actually do a lot better if a larger dataset is used. The neural network is too deep and it has started to recognize the 'noise' in training data. That is why training accuracy is approaching 80% but validation accuracy is only 60%.

5 Results

Compiling the results from all three experiments, we come to know that PreTrained ResNet50 gives the best performance with a peak accuracy of 60%. It is also noticed that the performance of ResNet50 trained from scratch was equivalent to that of PreTrained MobileNetV2. This directly means that ResNet is better than MobilNet. Evolving accuracies of all three models are illustrated below.



A summary of validation accuracies of respective models are given here:

Models	Accuracies	Epochs
PreTrained MobileNetV2	47.2%	20
ResNet50 (scratch)	47.5%	30
PreTrained ResNet50	60%	34

Table 1: Accuracies of all models

6 Conclusion

Analyzing the accuracy values and dataset used, it is concluded that insufficient data is the limiting factor here and accuracies can be improved by collecting more data to truly represent the diversity of art in the data. The main problem is 'cross-class similarity' and 'intra-class diversity'. For Example, minimalistic art and abstract art have quite similar images but at the same time, Realism has different kinds of images that do not look similar.

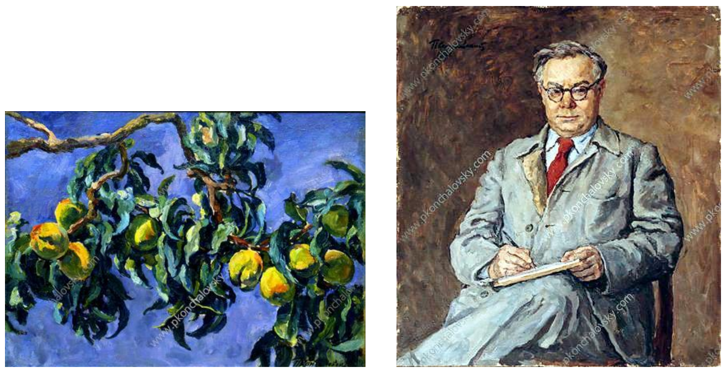


Figure 6: Two Images from 'Realism' art style

Following are some images taken from three different art styles namely 'Abstract_Art', 'Abstract_Expressionism', and 'Minimalism'. It is noticeable that all images are very similar in nature. To extract such minimal differences, we need a larger dataset.

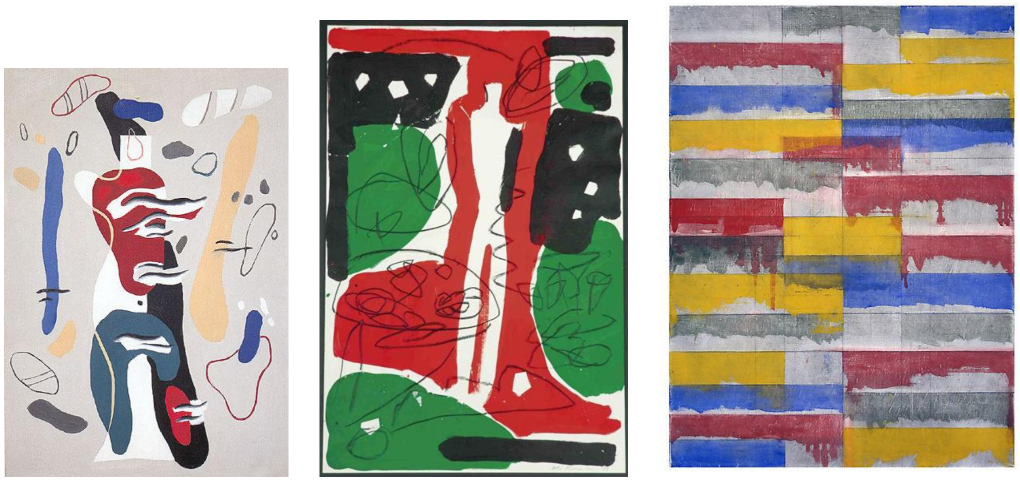


Figure 7: Images from three different art styles

Also, Figure 2 shows that the data distribution for all classes is not constant and some classes have very few training samples. Dropping some classes with fewer samples can increase overall accuracy but that scenario would not be practical.
This implementation has ultimately yielded an accuracy of 60% for the classification of 25 art styles based on the WikiArt dataset mentioned in section 2.1. I plan on investigating this idea further by getting more training data and using different neural networks with some improved techniques.