Classification of Paintings from Different Perspectives

Melisa Ankut [1] Mehmet Ertürk Gürses [2]

Abstract

Every artist is inspired by different artists and is curious about the artists s/he likes. Here we will try to predict which trends and styles of famous artists resemble the drawing to be analyzed. We want to help artists to motivate themselves, learn more about their styles, etc. We do experiment on a dataset with 20000 images and 10 features. Our goal is to make style predictions on images, based on their specifications. We use CNN to train our model. We will cover the technical details in the following sections. However, we must say that many features should be considered in this classification. This greatly affects the accuracy rate. We will tell you what we have done on this subject and what those who want to work on this subject can do.

1. Introduction

Throughout history, people have created many visual artworks, starting with murals and continuing today as electronic paintings. During this period, many movements and artists emerged. While these movements sometimes have very different styles, sometimes they can be difficult to distinguish. Especially people far from the art history may have difficulties in this regard. A new painter may not know exactly which art movement or artist he is close to. If there is no one to guide them at this point, they may experience some difficulties. For example, this information is very important in terms of knowing other artists that he can take as an example. In this regard, we wanted to develop a machine learning model in which one can learn to what trend his or her picture is close to. Our goal is for the classifier to be able to predict about 10 different art movements. Therefore, we are planning to use CNN.We'll talk about the technical details later, but of course we do not claim that this method is the best way. With progress in this area, much more complex structures can be built and a much higher accuracy can be achieved. We believe that much more work needs to be done and will be done in this regard.

Figure 1. Example Pictures

2. Related Work

With the digitalization of everything in today's world, a branch of art that we call AI art has emerged. There were also many projects related to the development of this branch. But most of these projects were about drawing and associating the artist. When we look at the basics, we have benefited from these projects in our roadmap because the current and painter classification are overlapping.

One of the most popular techniques for creating models is linear kernel with a SVM [1]. In order to better understand which features were most useful in the project, they initially used a Naive Bayes classifier to assign predicted tags to test data, then switched to SVM.Initially they were working on only two labels and later they increased the number of labels. They employed a one-vs-all SVM.

In another project [2] , they extract SURF properties from images and generate a Linear SVM. To develop this foundation, they create two CNN models using transfer learning. But they do this because the datasets are very small, we did not choose to use the ResNet-18 and VGG-16 models.

In another important project[7], they created their own datasets from WikiArt.org. They have compared the performance of deep neural networks with traditional machine learning techniques that would require hand-crafted feature extraction. And according to the article, they got the highest accuracy rate with CNN.

Feature extraction method	Supervised learning method	Accuracy
Image Processing	Nearest Neighbor	65.71%
Image Processing	SVM	68.33%
Image Processing	Genetic Algorithm	78.33%
RBM	Nearest Neighbor	64.41%
RBM	SVM	77.50%
RBM	Genetic Algorithm	73.92%
Image Processing + RBM	Nearest Neighbor	68.71%
Image Processing + RBM	SVM	71.66%
Image Processing + RBM	Genetic Algorithm	90.44%
Convolutional Autoencoder	**CNN**	**96.52%**

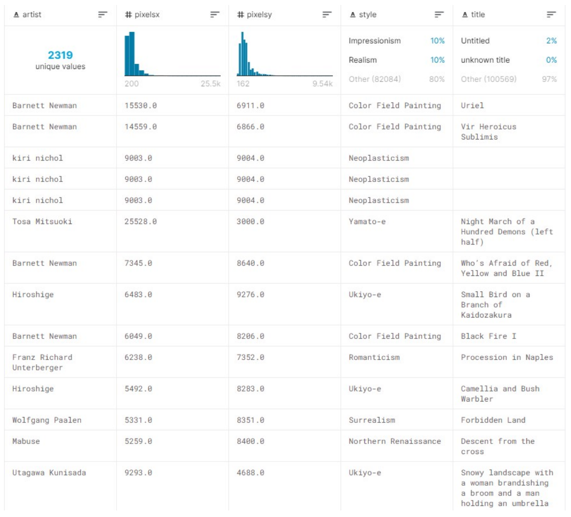
Figure 2. The Accuracy of Some Method Pairs

If we look at the project [3] we take as a basis, They used Convolutional Autoencoder as a feature extraction method, and CNN as supervised learning method. We saw that the highest accuracy results were achieved with CNN based model.

3. The Approach

3.1. Dataset

In order to train our model, we'll need pictures of many paintings. We found a large dataset compiled from WikiArt for a Kaggle competition. It has more than 100,000 paintings belonging to 2,300 artists, covering a lot of styles and eras. Distribution is balanced by us to improve accuracy. We won't use some pictures to do this balancing. Each style has the same number of pictures. Some styles are impressionism, realism, romanticism, etc. While we resized the images as 32x32 in our previous works, we now resize them to 256x256. Because we used to want to save time, but now we aimed to increase our accuracy rate instead of saving time.



artist	pixelsx	pixelsy	style	title
2319 unique values	200 – 25.5k	162 – 9.54k	Impressionism 10% Realism 10% Other (82084) 80%	Untitled 2% unknown title 0% Other (100569) 97%
Barnett Newman	15530.0	6911.0	Color Field Painting	Uriel
Barnett Newman	14559.0	6866.0	Color Field Painting	Vir Heroicus Sublimis
kiri nichol	9003.0	9004.0	Neoplasticism	
kiri nichol	9003.0	9004.0	Neoplasticism	
kiri nichol	9003.0	9004.0	Neoplasticism	
Tosa Mitsuoki	25528.0	3000.0	Yamato-e	Night March of a Hundred Demons (left half)
Barnett Newman	7345.0	8640.0	Color Field Painting	Who's Afraid of Red, Yellow and Blue II
Hiroshige	6483.0	9276.0	Ukiyo-e	Small Bird on a Branch of Kaidozakura
Barnett Newman	6049.0	8206.0	Color Field Painting	Black Fire I
Franz Richard Unterberger	6238.0	7352.0	Romanticism	Procession in Naples
Hiroshige	5492.0	8283.0	Ukiyo-e	Camellia and Bush Warbler
Wolfgang Paalen	5331.0	8351.0	Surrealism	Forbidden Land
Mabuse	5259.0	8400.0	Northern Renaissance	Descent from the cross
Utagawa Kunisada	9293.0	4688.0	Ukiyo-e	Snowy landscape with a woman brandishing a broom and a man holding an umbrella

Figure 3. Example Pictures

We also preprocessed our dataset for some reason. Some of these reasons are: corrupted images, unnecessary fields, missing or incorrect labels, images that are too small for the sizes we are working with, etc. We also made corrections on images that we could detect and thought were misclassified. Apart from these, since our dataset was very large, visual reading processes took a very long time. Therefore, before the training, we saved our dataset as some special file formats (HDF5) using some libraries (hickle).

As you can see in Figure 4, the art movements we chose were shaped like this. In summary, we randomly selected 2000 photographs of the top 10 art movements with the most photographs in the data we have.

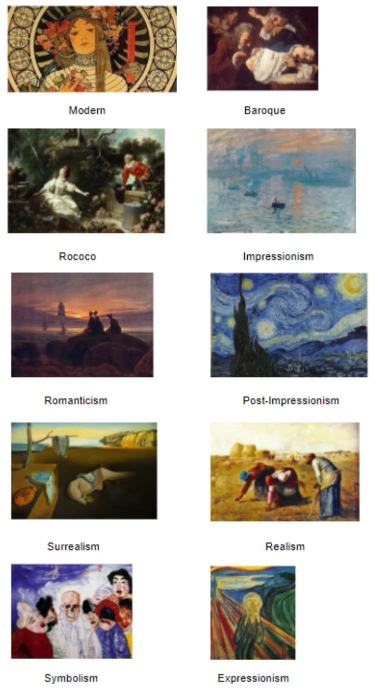


Figure 4. Example Pictures

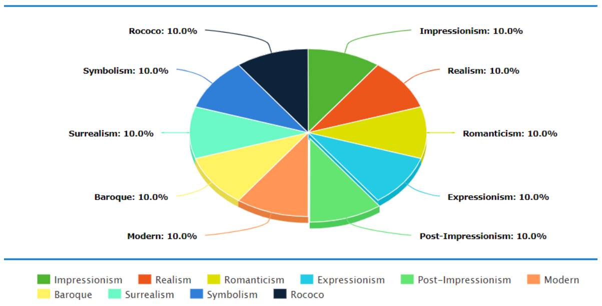
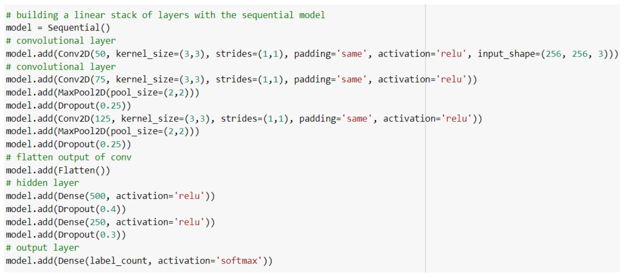


Figure 5. Frequencies of Pictures

3.2. Convolutional Neural Networks

In CNN, we use several components which are stacked on top of each other. Some layers are the max-pooling layer, convolutional layer, etc. These layers get the subsamples of the picture (e.g., for each 2 × 2 regions select only the maximum value, thus resulting in four times reduction in size) and process them. Guidelines CNNs are normally utilized in a managed system, where a huge preparing dataset (normally including, at any rate, a large number of pictures per class) is accessible. Accordingly, utilizing CNNs for start to finish painter grouping is risky, because of fewer preparing tests accessible per painter.



```
# building a linear stack of layers with the sequential model
model = Sequential()
# convolutional layer
model.add(Conv2D(50, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu', input_shape=(256, 256, 3)))
# convolutional layer
model.add(Conv2D(75, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(125, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
# flatten output of conv
model.add(Flatten())
# hidden layer
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(250, activation='relu'))
model.add(Dropout(0.3))
# output layer
model.add(Dense(label_count, activation='softmax'))
```

Figure 6. Our CNN structure

3.3. Layers

The full CNN contains the following layers:
1- Input layer
2- Max-pooling layer
3- Fully connected layer
4- Convolutional layer
5- Output softmax layer
Now let's see what these layers are.

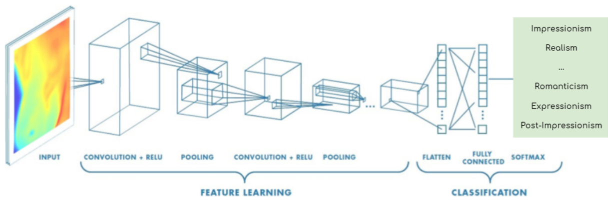


Figure 7. Layers representation

3.3.1. Input Layer

The input layer consists of the raw image (resized to 256 × 256 pixels) in three channels.

3.3.2. Max-pooling Layer

Like the convolutional layer, the pooling layer also aims to reduce the dimensionality. In this way, both the required processing power is reduced and the captured unnecessary features are ignored and more important features are focused on. In the pooling layer, which has a filter like the convolutional layer, this filter again hovers over the image. However, instead of convolutional operation, he applies the determined pooling technique this time. For max pooling, this is to get the largest value in the area covered by the filter. In this way, the size is reduced and the important features remain in our hands.

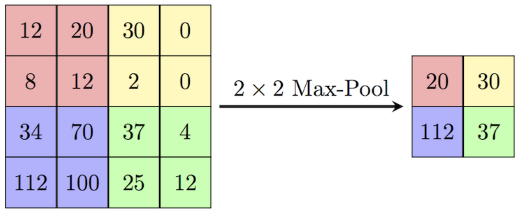


Figure 8. Max-pooling representation

3.3.3. UpSampling2D Layer

The simplest way to upsample an input an input is to double each row and column, which is what the upsampling layer does.The Keras deep learning library provides this capability in a layer called UpSampling2D.

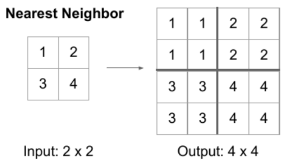


Figure 9. UpSampling2D representation

3.3.4. Fully connected Layer

In the fully connected layer, our visual that passes through the convolutional layer and the pooling layer several times and is in matrix form is turned into a flat vector.

3.3.5. Convolutional Layer

The convolution layer is the first layer that handles the image and the main building block of CNN. As it is known, visuals are actually matrices consisting of pixels with certain values. In the convolution layer, a filter smaller than the original image size hovers over the image and tries to capture certain features from these images.

3.3.6. Output Softmax Layer

The output softmax layer is the layer that implements the softmax function by using the score values from the fully connected layer. Softmax function generates a probability-based loss value using the score values generated by the artificial neural network. As a result of softmax, the probability value is generated for the similarity of the test input for each class. The score values that the artificial neural network model outputs are non-normalized values. Softmax normalizes these values and converts them into probability values. Converting to probability value is done with the maximum likelihood function, which is frequently used in statistics. Since the values are very large values, likelihood preserves the structure of the distribution by taking the log of the function.

3.4. Convolutional Autoencoders

Convolutional Autoencoder is a variant of Convolutional Neural Networks that are used as the tools for unsupervised learning of convolution filters. They are generally applied in the task of image reconstruction to minimize reconstruction errors by learning the optimal filters.

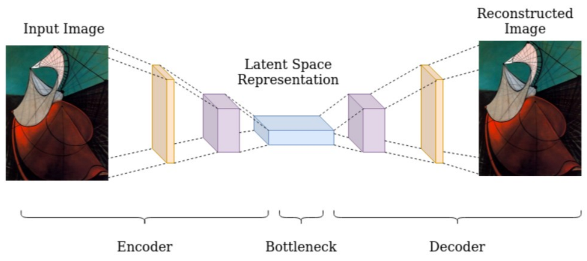
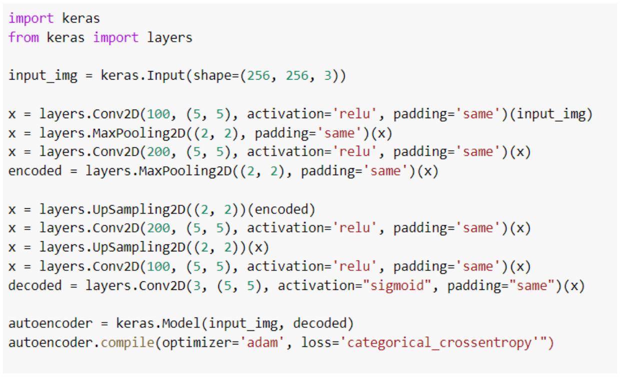


Figure 10. CAE representation

Our convolutional autoencoder contains the following layers:
1- The input layer consists of the raw image (resampled to 256×256 pixels) in three channels (R, G, and B)
2- convolutional layer of size $100 \times 256 \times 256$
3- max-pooling layer of size 2×2
4- convolutional layer of size $200 \times 128 \times 128$
5- max-pooling layer of size 2×2
6- unpooling layer of size 2×2
7- deconvolutional layer of size $200 \times 128 \times 128$
8- unpooling layer of size 2×2
9- deconvolutional layer of size $100 \times 256 \times 256$

The convolution filter sizes are always of size 5×5.



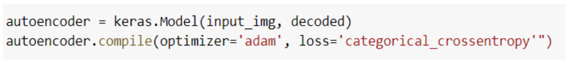
```
import keras
from keras import layers

input_img = keras.Input(shape=(256, 256, 3))

x = layers.Conv2D(100, (5, 5), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(200, (5, 5), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

x = layers.UpSampling2D((2, 2))(encoded)
x = layers.Conv2D(200, (5, 5), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(100, (5, 5), activation='relu', padding='same')(x)
decoded = layers.Conv2D(3, (5, 5), activation="sigmoid", padding="same")(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='categorical_crossentropy'")
```

Figure 11. CAE implementation in our code

3.5. Multi-Class Cross-Entropy Loss

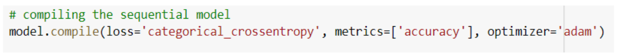
Cross-entropy is the default loss function to be used in multiclass classification problems. For this situation, it is planned to be utilized with multiclass classification where the objective qualities are in the set (0,1,2...,n) and each class is allocated an unique number worth. Cross-entropy will figure a score summing up the normal distinction between the real and anticipated likelihood dispersions for all classes in the issue. The score is limited and an ideal cross-entropy esteem is 0.

Cross-entropy can be specified as a loss function in Keras when compiling the model by specifying it as 'categorical_crossentropy'.



```
autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='categorical_crossentropy'")
```

Figure 12. Using multiclass cross-entropy in our code's CAE part



```
# compiling the sequential model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```

Figure 13. Using multiclass cross-entropy in our code's CNN part

3.6. Optimization Algorithm

Optimization methods are used to find the optimum value in the solution of nonlinear problems. Optimization algorithms such as stochastic gradient descent, adagrad, adadelta, adam, adamax are widely used in deep learning applications.

$$g = \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$$

$$m = \beta_1 m + (1-\beta_1)g$$

$$s = \beta_2 s + (1-\beta_2)g^T g$$

$$\theta = \theta - \epsilon_k \times m/\sqrt{s + eps}$$

Figure 14. Training loss results with MNIST dataset

There are differences between these algorithms in terms of performance and speed. We chose 'adam' as the optimization algorithm. As a result of the examinations we made in various studies[4,5], we observed that it performed well in terms of accuracy and speed.

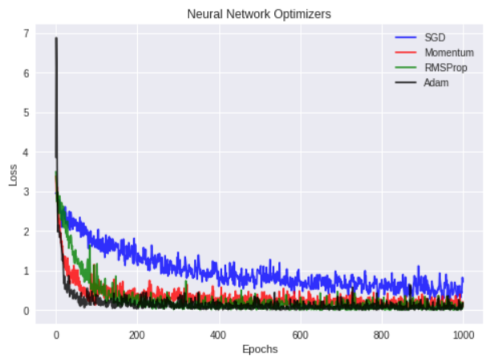
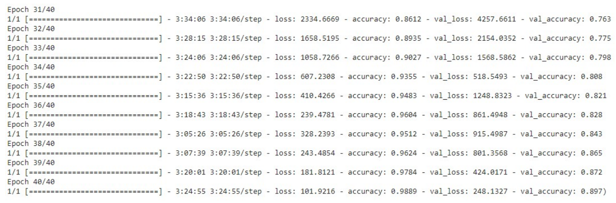


Figure 15. Training loss results with MNIST dataset

4. Experimental Evaluation

We splitted our dataset into Train as %80 and Test as %20. Our compiling time was quite long. The size of our data, which we reshaped to 256x256 in order to increase the accuracy rate, and the fact that we could not increase our batch size due to memory shortage caused us to lose a lot of time.

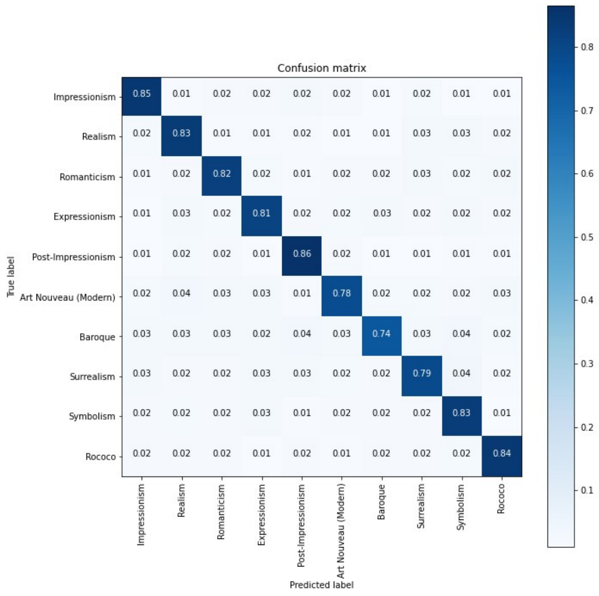
In order to make up for the lost time, we changed our learning rate, which is defined as 0.01 by default, to 0.05 and aimed to achieve the same accuracy rate with a decrease in the number of epochs.



```
Epoch 31/40
1/1 [==============================] - 3:34:06 3:34:06/step - loss: 2334.6669 - accuracy: 0.8612 - val_loss: 4257.6611 - val_accuracy: 0.763
Epoch 32/40
1/1 [==============================] - 3:28:15 3:28:15/step - loss: 1658.5195 - accuracy: 0.8935 - val_loss: 2154.0352 - val_accuracy: 0.775
Epoch 33/40
1/1 [==============================] - 3:24:06 3:24:06/step - loss: 1058.7266 - accuracy: 0.9027 - val_loss: 1568.5862 - val_accuracy: 0.798
Epoch 34/40
1/1 [==============================] - 3:22:50 3:22:50/step - loss: 607.2308 - accuracy: 0.9355 - val_loss: 518.5493 - val_accuracy: 0.808
Epoch 35/40
1/1 [==============================] - 3:15:36 3:15:36/step - loss: 410.4266 - accuracy: 0.9483 - val_loss: 1248.8323 - val_accuracy: 0.821
Epoch 36/40
1/1 [==============================] - 3:18:43 3:18:43/step - loss: 239.4781 - accuracy: 0.9604 - val_loss: 861.4948 - val_accuracy: 0.828
Epoch 37/40
1/1 [==============================] - 3:05:26 3:05:26/step - loss: 328.2393 - accuracy: 0.9512 - val_loss: 915.4987 - val_accuracy: 0.843
Epoch 38/40
1/1 [==============================] - 3:07:39 3:07:39/step - loss: 243.4854 - accuracy: 0.9624 - val_loss: 801.3568 - val_accuracy: 0.865
Epoch 39/40
1/1 [==============================] - 3:20:01 3:20:01/step - loss: 181.8121 - accuracy: 0.9784 - val_loss: 424.0171 - val_accuracy: 0.872
Epoch 40/40
1/1 [==============================] - 3:24:55 3:24:55/step - loss: 101.9216 - accuracy: 0.9889 - val_loss: 248.1327 - val_accuracy: 0.897)
```

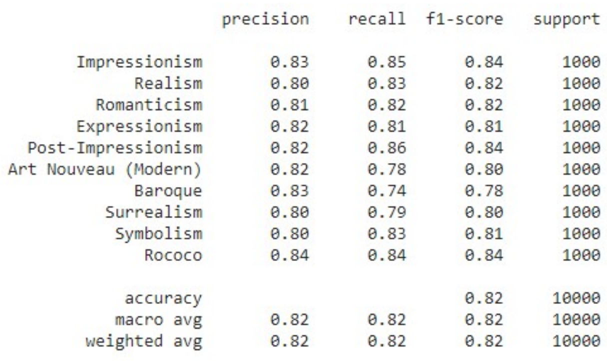
Figure 16. Epoch Results

To show the difference of the convolutional autoencoder, we trained our model twice and tested the results. While we used CAE as feature extraction in one, we did not use it in the other. Using CAE caused a 6% increase in our accuracy rate. Also when we used the autoencoder, we achieved an increase in accuracy. You can see this from the confusion matrices below.



	Impressionism	Realism	Romanticism	Expressionism	Post-Impressionism	Art Nouveau (Modern)	Baroque	Surrealism	Symbolism	Rococo
Impressionism	0.85	0.01	0.02	0.02	0.02	0.02	0.01	0.02	0.01	0.01
Realism	0.02	0.83	0.01	0.01	0.02	0.01	0.01	0.03	0.03	0.02
Romanticism	0.01	0.02	0.82	0.02	0.01	0.02	0.02	0.03	0.02	0.02
Expressionism	0.01	0.03	0.02	0.81	0.02	0.02	0.03	0.02	0.02	0.02
Post-Impressionism	0.01	0.02	0.02	0.01	0.86	0.02	0.01	0.01	0.01	0.01
Art Nouveau (Modern)	0.02	0.04	0.03	0.03	0.01	0.78	0.02	0.02	0.02	0.03
Baroque	0.03	0.03	0.03	0.02	0.04	0.03	0.74	0.03	0.04	0.02
Surrealism	0.03	0.02	0.02	0.03	0.03	0.02	0.02	0.79	0.04	0.02
Symbolism	0.02	0.02	0.02	0.03	0.01	0.02	0.02	0.02	0.83	0.01
Rococo	0.02	0.02	0.02	0.01	0.02	0.01	0.02	0.02	0.02	0.84

Figure 17. Confusion Matrix without Autoencoder



```
                      precision    recall  f1-score   support

       Impressionism       0.83      0.85      0.84      1000
             Realism       0.80      0.83      0.82      1000
         Romanticism       0.81      0.82      0.82      1000
       Expressionism       0.82      0.81      0.81      1000
  Post-Impressionism       0.82      0.86      0.84      1000
Art Nouveau (Modern)       0.82      0.78      0.80      1000
             Baroque       0.83      0.74      0.78      1000
          Surrealism       0.80      0.79      0.80      1000
           Symbolism       0.80      0.83      0.81      1000
              Rococo       0.84      0.84      0.84      1000

            accuracy                           0.82     10000
           macro avg       0.82      0.82      0.82     10000
        weighted avg       0.82      0.82      0.82     10000
```

Figure 18. Accuracy Results without Autoencoder

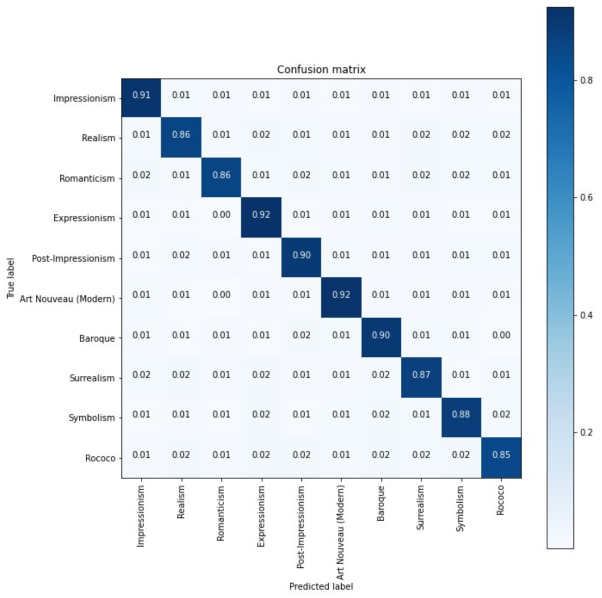
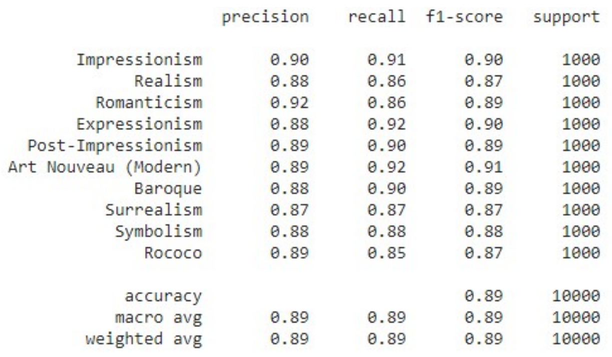


Figure 19. Confusion Matrix with Autoencoder



```
                      precision    recall  f1-score   support

       Impressionism       0.90      0.91      0.90      1000
             Realism       0.88      0.86      0.87      1000
         Romanticism       0.92      0.86      0.89      1000
       Expressionism       0.88      0.92      0.90      1000
  Post-Impressionism       0.89      0.90      0.89      1000
Art Nouveau (Modern)       0.89      0.92      0.91      1000
             Baroque       0.88      0.90      0.89      1000
          Surrealism       0.87      0.87      0.87      1000
           Symbolism       0.88      0.88      0.88      1000
              Rococo       0.89      0.85      0.87      1000

            accuracy                           0.89     10000
           macro avg       0.89      0.89      0.89     10000
        weighted avg       0.89      0.89      0.89     10000
```

Figure 20. Accuracy Results with Autoencoder

5. Conclusion

Since the style classifier has many outputs, it needs a lot of training data. As we can see from the studies we have referenced, our layers are at a sufficient level.Our dataset, which we created with 2000 photos for each style, achieved the accuracy rate we wanted by using convolutional autoencoder for feature extraction and CNN as a supervised learning method. However, since we had serious problems with the size of the data in our first trials, we could not attach importance to the data distribution and obtained low results. From here, we understood how important sub-sampling techniques are, no matter how good our model is.

If we look at the weaknesses of our project; Since we prefer an autoencoder during the feature extraction process, our image processing process takes a long time. Since we have very serious problems in the dimension of the dataset, we worked on the first 10 art movements with the most data. While selecting the images, we had to do a random elimination. This may have caused a decrease in our accuracy rate.

If we look at the strengths of our project; we caught a serious decrease in the error rate with the feature extraction we made using the autoencoder.We think that it is an open-ended project that can be developed much more if there are no memory and time constraints.

6. Future Works

The model can be trained for twenty classes instead of ten classes. The data set we use is suitable for this, but because we are limited in time and we want to achieve high results in a much narrower area, we limited it to ten classes.

We think that the dataset we used and our training approach are also suitable for artist classification. In addition to the style, artist classification can also be added to the classification.

We are currently processing on the CPU. This causes the training period to be very long. We will try to run it on TPU, which we think is suitable for CNN models. In this way, we think that the time will be shortened by about 10 times. We had some small trials, but we couldn't complete it because it needed to be optimized.

7. References

1. Using Machine Learning for Identification of Art Paintings

2. Artist Identification with Convolutional Neural Networks

3. DeepPainter: Painter Classification Using DeepConvolutional Autoencoders

4. Determining the Best Model with Deep Neural Networks: Keras Application on Mushroom Data

5. Most Commonly Used Hyper-parameters in Deep Learning Applications

6. Kaggle dataset

7. Comparison of Machine Learning techniques for painting classification

8. Recognizing Art Style Automatically in painting with deep learning

9. Neural Network Optimization Algorithms

10. Genre Classification of Paintings

11. Style classification and visualization of art painting's genre using self-organizing maps

12. Genre and Style Based Painting Classification