

Bonus A

In the model I added one more linear layer, which reduces the output size from 300 to 150. This layer can help the model learn more complex features by providing additional non-linearity. I also added a second dropout layer after the linear layer, which helps prevent overfitting by randomly zeroing some of the activations during training. It also includes a second ReLU activation function following the linear_add layer, improving learning and feature representation. I also changed the batch size from 32 to 64.


After making some changes and some experiments with the model's architecture, I found out that there was an issue with the classes, because every time I was getting different accuracy. I think that the `set()` does not guarantee the order of the classes leading to mismatch between the indices used during training and testing. So, in order to fix that I made some changes in the `wikiart` dataset (`wikiart.py`). Each time I load the data I will sort the classes instead of using `set()` and `list()` after I will save the sorted class names to a file ensuring that training and testing process will always use the same index order. The (`class_list_file`) file contains the ordered list of classes. If it exists, it loads the classes from this file. Otherwise, it generates the class list, sorts it, and saves it to `class_list_file` for consistency.

And below it can be seen the accuracy of the model (which is more than 5% than the accuracy of the initial model) after all the changes.

Running...

Gathering files for `wikiart/test`

...finished

100% 
 630/630 [00:02<00:00, 211.57it/s]

Accuracy: 0.16031746566295624

Part 1

For this part I checked the amount of data for each class and I saw that a lot of classes have large number of data while some other classes have less.

Original class distribution: {'Abstract_Expressionism': 449, 'Action_painting': 18, 'Analytical_Cubism': 15, 'Art_Nouveau_Modern': 688, 'Baroque': 721, 'Color_Field_Painting': 268, 'Contemporary_Realism': 91, 'Cubism': 390, 'Early_Renaissance': 246, 'Expressionism': 1127, 'Fauvism': 163, 'High_Renaissance': 198, 'Impressionism': 2269, 'Mannerism_Late_Renaissance': 246, 'Minimalism': 212, 'Naive_Art_Primitivism': 373, 'New_Realism': 41, 'Northern_Renaissance': 413, 'Pointillism': 80, 'Pop_Art': 244, 'Post_Impressionism': 946, 'Realism': 1712, 'Rococo': 369, 'Romanticism': 1157, 'Symbolism': 679, 'Synthetic_Cubism': 38, 'Ukiyo_e': 197}

So, I decided to limit the data for each class to 300 and the rest of the classes that have less than 300, I added duplicate entries until these classes reach the required amount of data. This is how the data looks like after the balancing:

Class distribution after balancing: {'Abstract_Expressionism': 300, 'Action_painting': 300, 'Analytical_Cubism': 300, 'Art_Nouveau_Modern': 300, 'Baroque': 300, 'Color_Field_Painting': 300, 'Contemporary_Realism': 300, 'Cubism': 300, 'Early_Renaissance': 300, 'Expressionism': 300, 'Fauvism': 300, 'High_Renaissance': 300, 'Impressionism': 300, 'Mannerism_Late_Renaissance': 300, 'Minimalism': 300, 'Naive_Art_Primitivism': 300, 'New_Realism': 300, 'Northern_Renaissance': 300, 'Pointillism': 300, 'Pop_Art': 300, 'Post_Impressionism': 300, 'Realism': 300, 'Rococo': 300, 'Romanticism': 300, 'Symbolism': 300, 'Synthetic_Cubism': 300, 'Ukiyo_e': 300}

1. The Autoencoder model is written in the wikiart.py script (see class WikiAutoencoder). It is a convolutional model. The encoder compresses the input images into low-dimensional representations, while the decoder reconstructs the original images. The train2.py is identical to the original train.py. For the training process I used 70 epochs, and I was checking the progress of the model through the loss function. The training started with a loss of 19985

Starting epoch 1/70

```
100%|███████████████████████████████████████████████████████████████████████|  
██████████████████████████████████████████████████████████████████████████| 254/254 [00:32<00:00, 7.89it/s]  
Epoch [1/70], Loss: 19985.0009
```

and finished with a loss of 5178.

Starting epoch 70/70

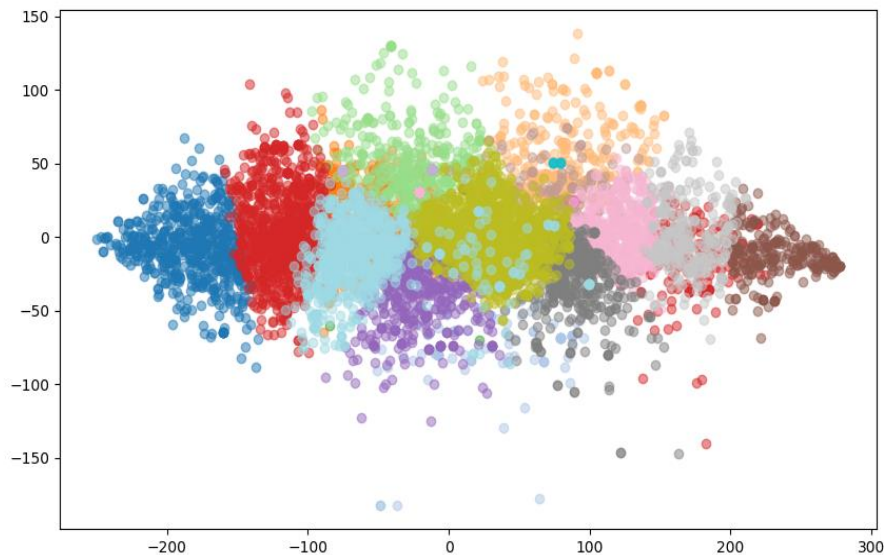
```
100%|██████████████████████████████████████████████████████████████████████████|
██████████████████████████████████████████████████████████████████████████████| 254/254 [00:15<00:00, 16.49it/s]
Epoch [70/70], Loss: 5178.2060
```

Initially I started the training with 100 epochs, but after epoch 70 the loss was stable without going down. So, I decided to retrain the model with 70 epochs.

2. For clustering I have created a different script called `cluster.py`. To run it use: `python3 cluster.py --config config.json`

This code extracts compressed representations of each image by passing them through the model's encoder. Using k-means clustering, the code groups these representations into clusters based on the number of art style classes in the dataset. The clustered data is then reduced to two dimensions via PCA, enabling

visualization of the clusters on a 2D scatter plot. Each cluster is color-coded (but I didn't put the name of classes). I think that my model is not clustering that well. Below there is also the plot.



Part 3

In the WikiAutoencoder2 model, I integrated style embedding through a dedicated linear layer that transforms a given style vector (of dimension 27) into a spatial representation of size 16×16 . This transformation allows us to condition the encoded image features with the style information, which is resized to match the dimensions of the encoded output.

During the forward pass, this style embedding is concatenated with the representation of the input image, enabling the decoder to generate an output image that reflects both the original content and the specified style. The model is trained using an autoencoder approach (from part 2). The model learns to produce stylized images that preserve structural information of the input while adapting to the characteristics dictated by the style embeddings, effectively facilitating a form of style transfer.

To train the model I wrote another script called `train3.py`, almost identical to `train.py`.

Then I created another script called `generate.py`, which generates the original and generated picture.

To run the `generate.py` use: `python3 generate.py --config config.json`

Though the training didn't go that well, the loss was very high, and I needed more 100 epochs to lower it (and a lot of time). Thus, the resulting image is not good.

The left is the original, the right is the generated.

