Classification of Paintings Using PyTorch CNN

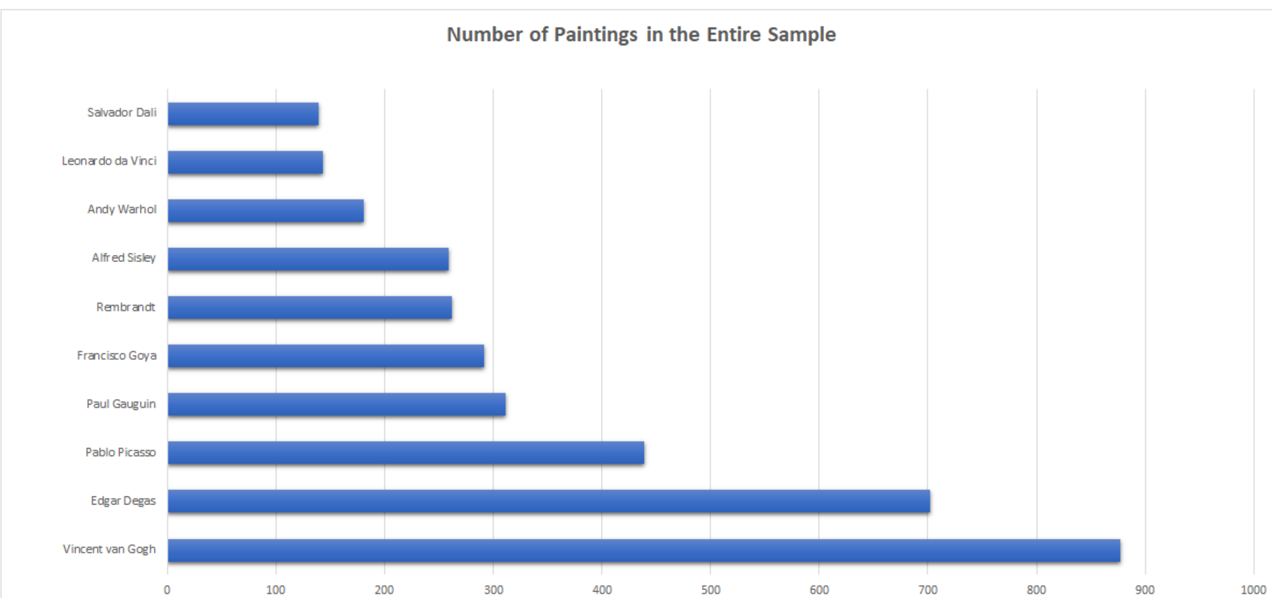
Spring 2019 Machine Learning II
Group 2: Naixin Zhu

Introduction

Classifying artistic works using convolution neural network has been under research by many scholars. Human, after a short time of training, could distinguish between genres of artistic work. Some may attribute to the art taste. However, there are object similarities that distinguish one artist's work from another. For example, Van Gough uses bold colors and distorted shapes, while Gauguin's color range is warmer. Can the machine learning algorithm tell the subtleties? We will find that out by apply convolution neural network to paintings from ten famous artists of all times.

Data Set

The original dataset is 8,355 paintings belonging to 50 artists collected by Icaro and made public on Kaggle[1]. I hand picked 10 artists that interests me and cut down the data to 3,604 paintings. I assigned 80% of that to training and 20% to testing. The ratio is kept constant within each class. The unfortunate part is that the dataset is unbalanced, figure below shows the number of paintings belonging to each category:



Van Gogh and Degas are the top two classes in this dataset. The may influence the accuracy of classification. The painters who are over-represented in the dataset tend to get higher accuracy score, which is confirmed by the results.

I use pytorch ImageFolder to import data into dataset class, then use data loader to load the train and test sets. In order to use ImageFolder, I preprocessed files into two parent folders: train and test. Under each

[1] https://www.kaggle.com/ikarus777/best-artworks-of-all-time

parent folder, there are ten classes. I kept the train test ratio constant within each class, so I calculated the split number based on the total number of painting in each class. I imported OS package and wrote the code in 0_load_and_preprocess.py.
I resized all images to 800 pixels and center cropped at 800. I normalized all three channels with mean 0.5 and standard deviation 0.5.

CNN Algorithm

Convolution network is a powerful algorithm that abstract the features from complicated paintings, and it uses element-wise matrix multiplication to see whether a picture contains that feature. A feature map convolves from left to right and from top to bottom across a graph. Every stop on the graph it calculates the element-wise product. If the product is a large number, it denotes that the particular area of the picture contains shape/color similar to the feature map; if the product is a small number, it denotes the opposite.

One can set the stride, which is by how many pixels a feature map moves to the right every time. Setting a large stride will prevent the overlap of feature maps during convolving. As we move deeper through the network, without padding the image size would decrease fast. If we want to preserve the original image size, or prevent it from decreasing to fast, we can add padding. Padding adds zeros to the outer border of image matrix so it maintains its size.

Experiments

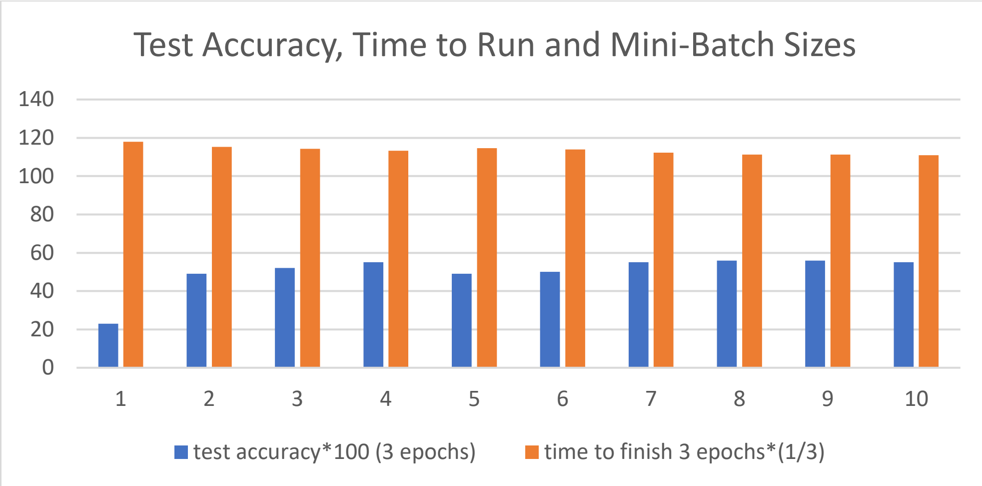
Model 1 (Baseline)

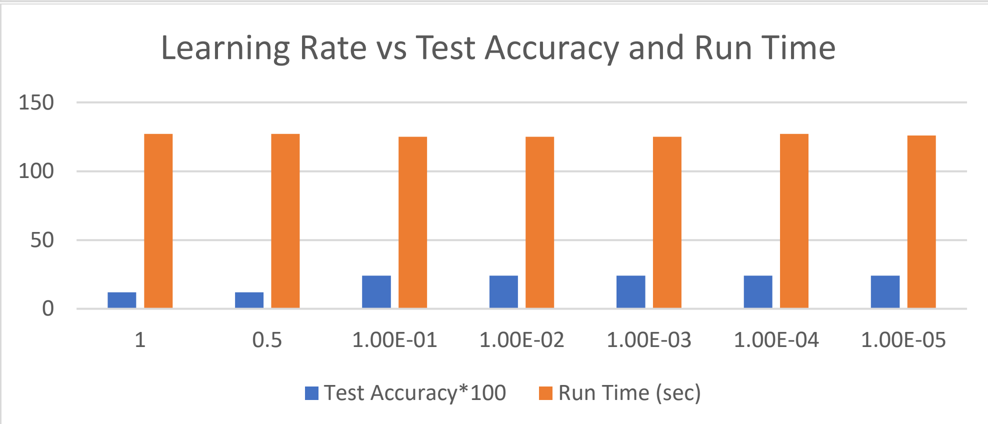
I will use a baseline model of 2 layers of CNN, and the network design is as follows:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 50, kernel_size=10, stride=5,  padding=2),
            nn.BatchNorm2d(50),
            nn.ReLU(),
            nn.MaxPool2d(4))
        self.layer2 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size=10, stride=5,  padding=2),
            nn.BatchNorm2d(100),
            nn.ReLU(),
            nn.MaxPool2d(4))
        self.fc = nn.Linear(100, 10)
```

All models' criteria are set to CrossEntropyLoss. The baseline model has Adam as optimizer. Except for models 2.1 and 2.2, which uses different optimizers, all other models use the same Adam optimizer.

In order to find the optimal mini batch size and learning rate, I looped over 1 through 10 for mini batch size and 1 through 1e-5 for learning rate on the baseline model. I found that batch size 4 and learning rate 1e-3 are the most optimal in terms of test accuracy and running time. The figures below show the test accuracy and running time for both variables, and file 1_cnn_model_minibatch.py and 1_cnn_model_learningrate.py are the code for running those loops. Thus, the batch size 4 and learning 1e-3 is fixed for all models in the research.





All the models are run with epoch number fixed to 20.

Model 2.1 (SGD with learning rate schedular StepLR)

The only difference between Model 2.1 and the baseline is optimizer and learning rate schedule:
optimizer = torch.optim.SGD(cnn.parameters(), lr=learning_rate)

StepLR means after every 2 epochs, the learning rate reduce by a multiple of gamma. This aims to reduce learning rate after we move closer to the minimum. We do not want large step size to miss the minimum of the loss function.
scheduler = StepLR(optimizer, step_size=2, gamma=0.95)

Model 2.2 (Rprop with learning rate schedular StepLR)

Rprop stands for resilient back propagation algorithm. According to Wikipedia, Rprop does the following: "for each weight, if there was a sign change of the partial derivative of the total error function compared to the last iteration, the update value for that weight is multiplied by a factor η^-, where $\eta^- < 1$. If the last iteration produced the same sign, the update value is multiplied by a factor of η^+, where $\eta^+ > 1$."[2]

The only difference between Model 2.2 and the base is optimizer and learning rate schedule:

[2] https://en.wikipedia.org/wiki/Rprop

optimizer = torch.optim.Rprop(cnn.parameters(), lr=learning_rate)
scheduler = StepLR(optimizer, step_size=2, gamma=0.95)

Model 3 (Increasing number of layers)

The reasons to add more layers is because as many CNN suggest, the deeper you get through
The design for model 3 is:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 50, kernel_size=10, stride=1, padding=0),
            nn.BatchNorm2d(50),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size=10, stride=1, padding=2),
            nn.BatchNorm2d(100),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(100, 200, kernel_size=10, stride=1, padding=0),
            nn.BatchNorm2d(200),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer4 = nn.Sequential(
            nn.Conv2d(200, 400, kernel_size=10, stride=1, padding=0),
            nn.BatchNorm2d(400),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer5 = nn.Sequential(
            nn.Conv2d(400, 150, kernel_size=10, stride=1, padding=0),
            nn.BatchNorm2d(150),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer6 = nn.Sequential(
            nn.Conv2d(150, 50, kernel_size=10, stride=1, padding=0),
            nn.BatchNorm2d(50),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.fc = nn.Linear(3802500, 10)
```

Model 4 (Increasing number of kernels)

The design for model 4 is:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 500, kernel_size=10, stride=5, padding=2),
            nn.BatchNorm2d(500),
            nn.ReLU(),
            nn.MaxPool2d(4))
        self.layer2 = nn.Sequential(
            nn.Conv2d(500, 1000, kernel_size=10, stride=5, padding=2),
            nn.BatchNorm2d(1000),
            nn.ReLU(),
            nn.MaxPool2d(4))
        self.fc = nn.Linear(1000, 10)
```

Model 5 (Add dropout layer)

The design for model 5 is:

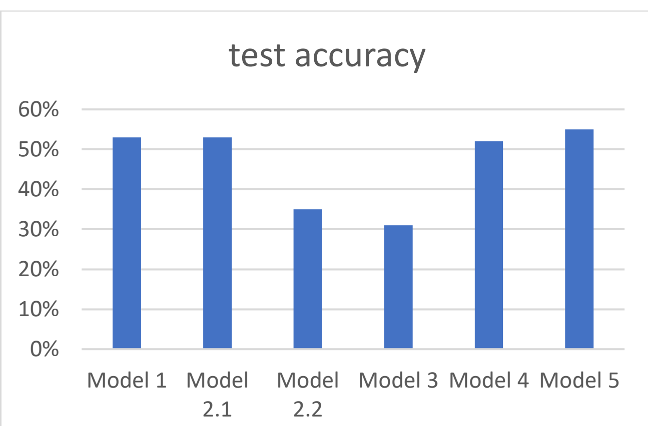
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 50, kernel_size=10, stride=5,  padding=2),
            nn.BatchNorm2d(50),
            nn.ReLU(),
            nn.MaxPool2d(4))
        self.dropout = nn.Dropout2d(p=0.5)
        self.layer2 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size=10, stride=5,  padding=2),
            nn.BatchNorm2d(100),
            nn.ReLU(),
            nn.MaxPool2d(4))
        self.fc = nn.Linear(100, 10)
```

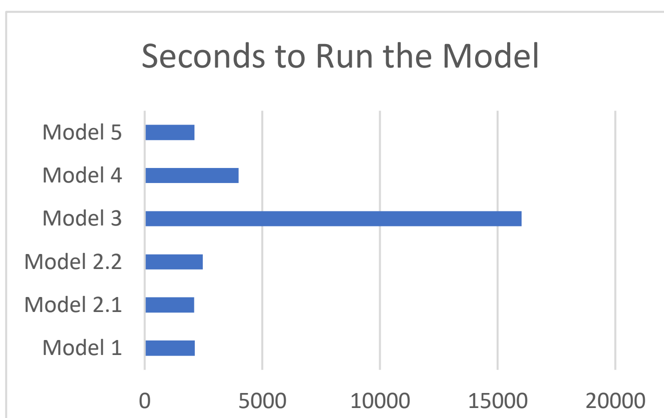
I will evaluate the effective of those models by using confusion matrix.

Results

Overall Comparison Across Models

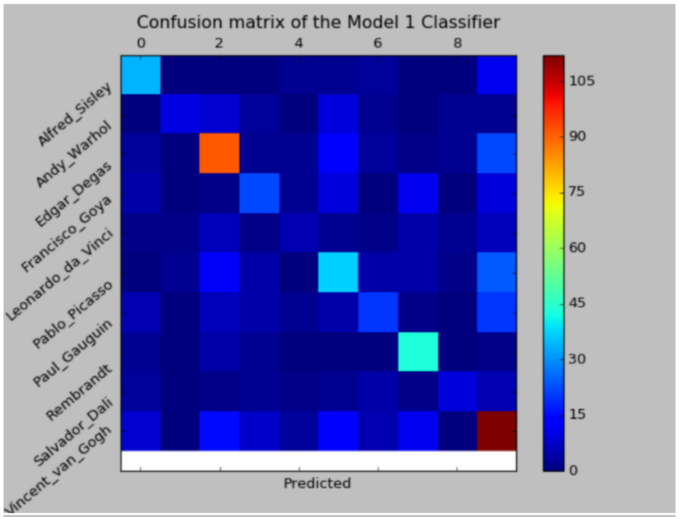
	Test Accuracy	Run Time (sec)	Top 1 correctly classified class	Number of correctly classified in Top 1	Top 2 correctly classified class	Number of correctly classified in Top 2
Model 1	53%	2127	Van Gogh	112	Degas	91
Model 2.1	53%	2107	Van Gogh	149	Degas	84
Model 2.2	35%	2464	Van Gogh	78	Degas	72
Model 3	31%	16014	Van Gogh	84	Degas	61
Model 4	52%	3997	Van Gogh	116	Degas	85
Model 5	55%	2114	Van Gogh	103	Degas	114

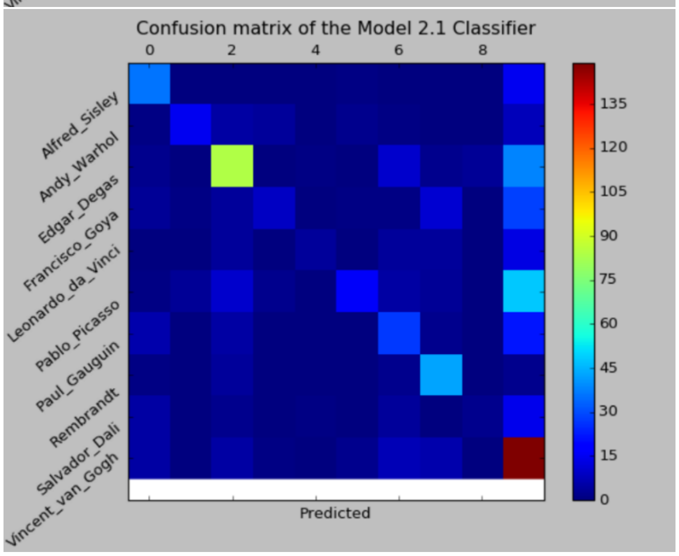


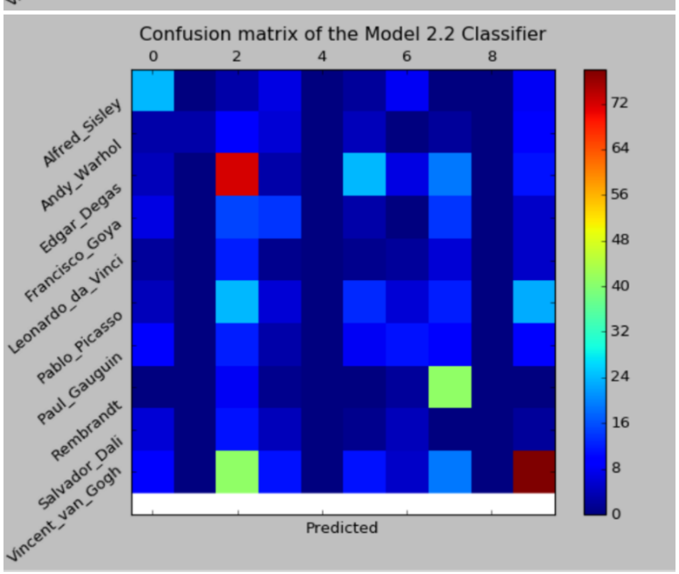


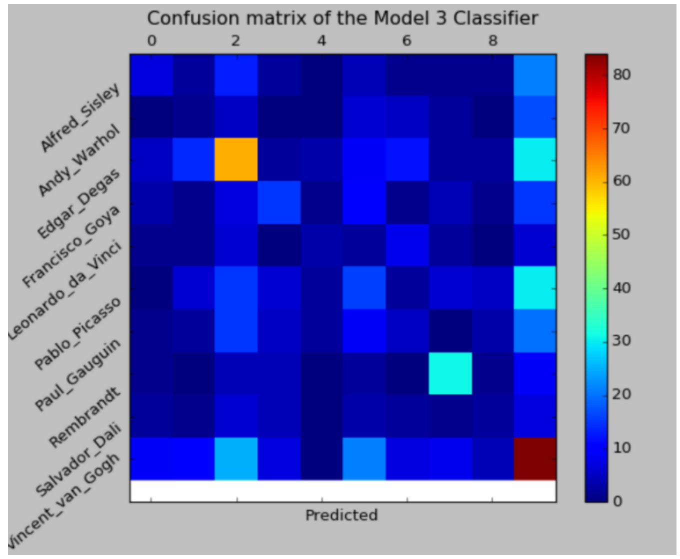
The diagonal of the confusion matrix measures the number of correctly classified samples. The area to the lower left of the diagonal is false negatives, where it belongs to the class while been classified as not, and the area to the upper right of the diagonal is false positives, where it does not belong to a class and been wrongly classified so. The brighter the color in a confusion matrix, the higher value is in that cell. Across all five models, Van Gogh and Degas generally get higher accuracy in test data prediction. This maybe that Van Gogh and Degas have proportionately larger training and testing samples than other paintings in the dataset, and thus, the algorithm could exploit the features of their paintings more than they could on others'.

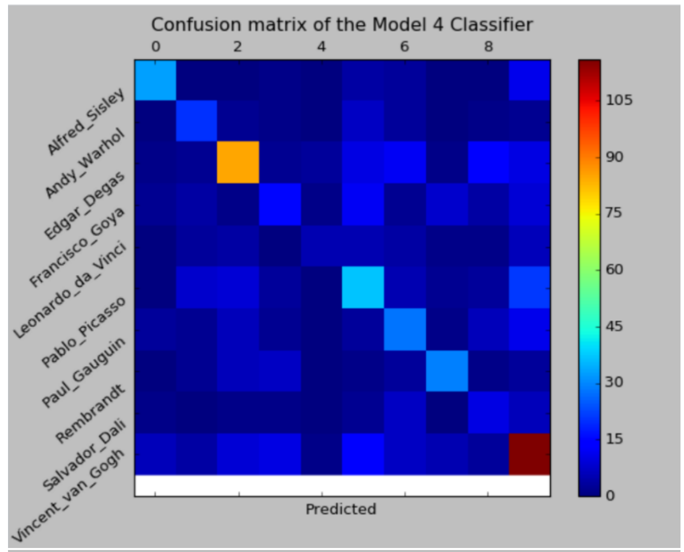
I will align the confusion matrix of all five models together, the matrix with percentages will follow the color blocks on the next page. The last page will show the training loss over number of iterations.

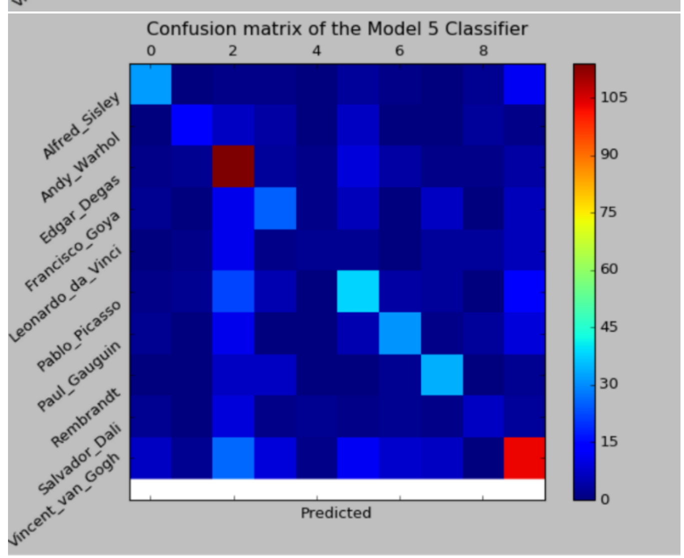

Confusion matrix of the Model 1 Classifier
0 2 4 6 8
Alfred_Sisley
Andy_Warhol
Edgar_Degas
Francisco_Goya
Leonardo_da_Vinci
Pablo_Picasso
Paul_Gauguin
Rembrandt
Salvador_Dali
Vincent_van_Gogh
Predicted
105 90 75 60 45 30 15 0


Confusion matrix of the Model 2.1 Classifier
0 2 4 6 8
Alfred_Sisley
Andy_Warhol
Edgar_Degas
Francisco_Goya
Leonardo_da_Vinci
Pablo_Picasso
Paul_Gauguin
Rembrandt
Salvador_Dali
Vincent_van_Gogh
Predicted
135 120 105 90 75 60 45 30 15 0


Confusion matrix of the Model 2.2 Classifier
0 2 4 6 8
Alfred_Sisley
Andy_Warhol
Edgar_Degas
Francisco_Goya
Leonardo_da_Vinci
Pablo_Picasso
Paul_Gauguin
Rembrandt
Salvador_Dali
Vincent_van_Gogh
Predicted
72 64 56 48 40 32 24 16 8 0


Confusion matrix of the Model 3 Classifier
0 2 4 6 8
Alfred_Sisley
Andy_Warhol
Edgar_Degas
Francisco_Goya
Leonardo_da_Vinci
Pablo_Picasso
Paul_Gauguin
Rembrandt
Salvador_Dali
Vincent_van_Gogh
Predicted
80 70 60 50 40 30 20 10 0


Confusion matrix of the Model 4 Classifier
0 2 4 6 8
Alfred_Sisley
Andy_Warhol
Edgar_Degas
Francisco_Goya
Leonardo_da_Vinci
Pablo_Picasso
Paul_Gauguin
Rembrandt
Salvador_Dali
Vincent_van_Gogh
Predicted
105 90 75 60 45 30 15 0


Confusion matrix of the Model 5 Classifier
0 2 4 6 8
Alfred_Sisley
Andy_Warhol
Edgar_Degas
Francisco_Goya
Leonardo_da_Vinci
Pablo_Picasso
Paul_Gauguin
Rembrandt
Salvador_Dali
Vincent_van_Gogh
Predicted
105 90 75 60 45 30 15 0

MODEL 1	Alfred_Sisley	Andy_Warhol	Edgar_Degas	Francisco_Goya	Leonardo_da_Vinci	Pablo_Picasso	Paul_Gauguin	Rembrandt	Salvador_Dali	Vincent_van_Gogh
Alfred_Sisley	65%	0%	0%	0%	4%	4%	6%	0%	0%	21%
Andy_Warhol	0%	28%	22%	8%	0%	25%	6%	0%	6%	6%
Edgar_Degas	2%	0%	65%	1%	1%	10%	2%	1%	1%	16%
Francisco_Goya	7%	0%	2%	38%	3%	16%	0%	19%	0%	16%
Leonardo_da_Vinci	3%	3%	21%	3%	17%	7%	3%	14%	7%	21%
Pablo_Picasso	0%	2%	14%	5%	0%	42%	5%	5%	1%	27%
Paul_Gauguin	8%	0%	10%	6%	3%	6%	32%	2%	0%	32%
Rembrandt	4%	0%	8%	4%	0%	0%	0%	83%	0%	2%
Salvador_Dali	11%	0%	4%	7%	4%	7%	14%	4%	32%	18%
Vincent_van_Gogh	5%	0%	9%	4%	2%	8%	3%	6%	0%	64%

MODEL 2.1	Alfred_Sisley	Andy_Warhol	Edgar_Degas	Francisco_Goya	Leonardo_da_Vinci	Pablo_Picasso	Paul_Gauguin	Rembrandt	Salvador_Dali	Vincent_van_Gogh
Alfred_Sisley	69%	0%	0%	0%	0%	2%	0%	0%	0%	29%
Andy_Warhol	3%	42%	14%	11%	0%	6%	3%	0%	0%	22%
Edgar_Degas	1%	0%	60%	0%	1%	0%	7%	1%	2%	27%
Francisco_Goya	5%	2%	7%	16%	0%	2%	2%	19%	0%	48%
Leonardo_da_Vinci	0%	0%	14%	0%	14%	0%	14%	14%	0%	45%
Pablo_Picasso	1%	3%	11%	2%	0%	18%	6%	3%	0%	55%
Paul_Gauguin	10%	0%	8%	0%	0%	0%	44%	3%	0%	35%
Rembrandt	2%	0%	8%	0%	0%	0%	4%	83%	0%	4%
Salvador_Dali	18%	0%	7%	0%	4%	0%	14%	0%	7%	50%
Vincent_van_Gogh	3%	0%	3%	1%	0%	1%	4%	3%	0%	85%

MODEL 2.2	Alfred_Sisley	Andy_Warhol	Edgar_Degas	Francisco_Goya	Leonardo_da_Vinci	Pablo_Picasso	Paul_Gauguin	Rembrandt	Salvador_Dali	Vincent_van_Gogh
Alfred_Sisley	46%	0%	6%	13%	0%	4%	15%	0%	0%	15%
Andy_Warhol	8%	8%	25%	17%	0%	11%	0%	6%	0%	25%
Edgar_Degas	3%	0%	51%	2%	0%	17%	5%	14%	0%	8%
Francisco_Goya	12%	0%	26%	24%	0%	5%	0%	24%	0%	9%
Leonardo_da_Vinci	7%	0%	41%	3%	0%	3%	7%	21%	0%	17%
Pablo_Picasso	5%	0%	27%	7%	0%	15%	7%	14%	0%	26%
Paul_Gauguin	15%	0%	19%	5%	0%	13%	18%	16%	0%	15%
Rembrandt	0%	0%	15%	2%	0%	0%	4%	79%	0%	0%
Salvador_Dali	21%	0%	39%	14%	0%	4%	14%	0%	0%	7%
Vincent_van_Gogh	6%	0%	23%	6%	0%	6%	3%	11%	0%	45%

MODEL 3	Alfred_Sisley	Andy_Warhol	Edgar_Degas	Francisco_Goya	Leonardo_da_Vinci	Pablo_Picasso	Paul_Gauguin	Rembrandt	Salvador_Dali	Vincent_van_Gogh
Alfred_Sisley	13%	4%	25%	4%	0%	8%	2%	2%	2%	40%
Andy_Warhol	0%	3%	14%	0%	0%	17%	14%	6%	0%	47%
Edgar_Degas	4%	10%	44%	1%	2%	6%	9%	1%	1%	21%
Francisco_Goya	5%	2%	12%	26%	2%	17%	2%	7%	2%	26%
Leonardo_da_Vinci	3%	3%	21%	0%	10%	7%	28%	7%	0%	21%
Pablo_Picasso	0%	7%	17%	7%	2%	18%	2%	7%	6%	34%
Paul_Gauguin	2%	3%	24%	8%	3%	15%	8%	0%	5%	32%
Rembrandt	2%	0%	8%	8%	0%	4%	0%	60%	2%	17%
Salvador_Dali	7%	4%	21%	14%	0%	11%	7%	4%	7%	25%
Vincent_van_Gogh	5%	6%	14%	4%	0%	12%	4%	5%	2%	48%

MODEL 4	Alfred_Sisley	Andy_Warhol	Edgar_Degas	Francisco_Goya	Leonardo_da_Vinci	Pablo_Picasso	Paul_Gauguin	Rembrandt	Salvador_Dali	Vincent_van_Gogh
Alfred_Sisley	63%	0%	0%	2%	0%	8%	6%	0%	0%	21%
Andy_Warhol	0%	56%	6%	3%	0%	19%	8%	0%	3%	6%
Edgar_Degas	1%	1%	61%	1%	2%	7%	9%	1%	10%	7%
Francisco_Goya	3%	7%	2%	26%	2%	21%	3%	14%	7%	16%
Leonardo_da_Vinci	0%	10%	14%	0%	17%	17%	14%	3%	3%	21%
Pablo_Picasso	0%	9%	10%	3%	0%	42%	6%	2%	3%	24%
Paul_Gauguin	5%	3%	10%	3%	0%	5%	45%	2%	10%	18%
Rembrandt	0%	4%	12%	13%	0%	2%	6%	56%	2%	6%
Salvador_Dali	4%	0%	4%	4%	0%	7%	25%	0%	36%	21%
Vincent_van_Gogh	3%	2%	5%	6%	1%	8%	4%	3%	2%	66%

MODEL 5	Alfred_Sisley	Andy_Warhol	Edgar_Degas	Francisco_Goya	Leonardo_da_Vinci	Pablo_Picasso	Paul_Gauguin	Rembrandt	Salvador_Dali	Vincent_van_Gogh
Alfred_Sisley	62%	0%	2%	2%	0%	6%	2%	0%	4%	23%
Andy_Warhol	0%	39%	19%	11%	0%	19%	0%	0%	8%	3%
Edgar_Degas	1%	1%	81%	2%	1%	6%	3%	1%	1%	3%
Francisco_Goya	3%	0%	19%	43%	2%	10%	0%	12%	0%	10%
Leonardo_da_Vinci	0%	3%	38%	3%	7%	7%	0%	10%	10%	21%
Pablo_Picasso	1%	2%	25%	6%	0%	43%	5%	3%	0%	15%
Paul_Gauguin	3%	0%	18%	0%	0%	8%	50%	2%	5%	15%
Rembrandt	0%	0%	13%	13%	0%	0%	4%	65%	0%	4%
Salvador_Dali	7%	0%	32%	4%	7%	4%	7%	4%	25%	11%
Vincent_van_Gogh	4%	1%	15%	5%	1%	7%	5%	4%	0%	59%

Figure 1. The Train Loss for Model 1

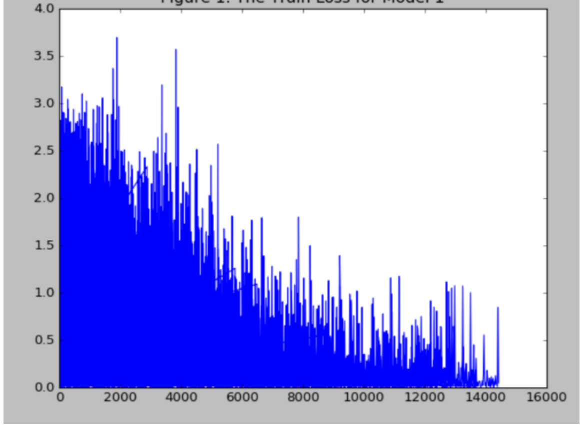


Figure 2.1. The Train Loss for Model 2.1

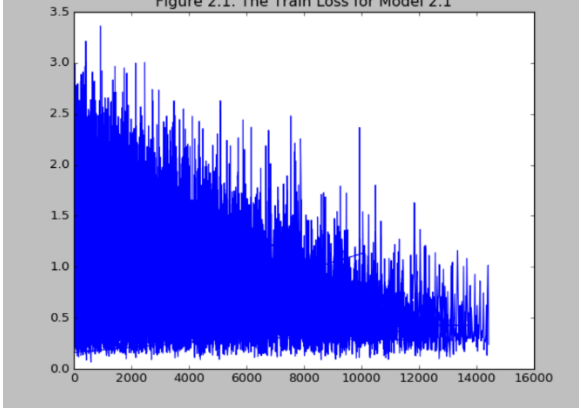


Figure 2.2. The Train Loss for Model 2.2

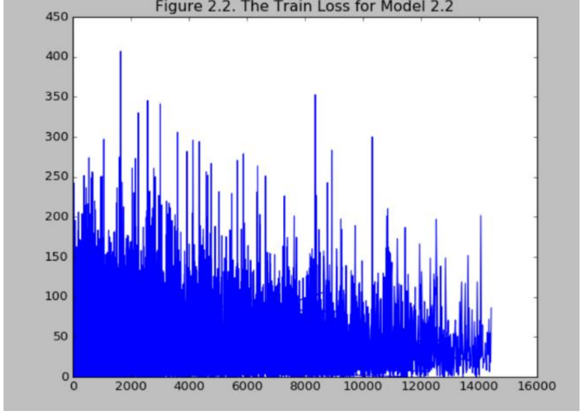


Figure 3. The Train Loss for Model 3

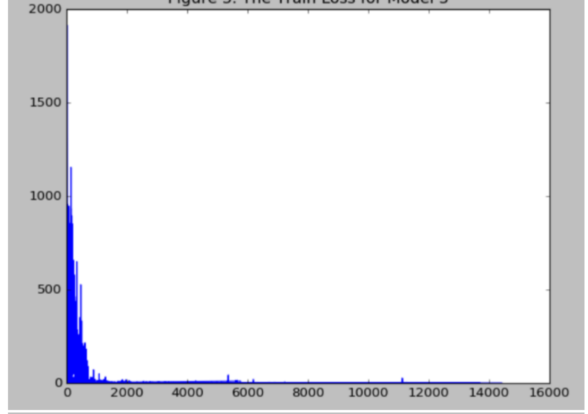


Figure 4. The Train Loss for Model 4

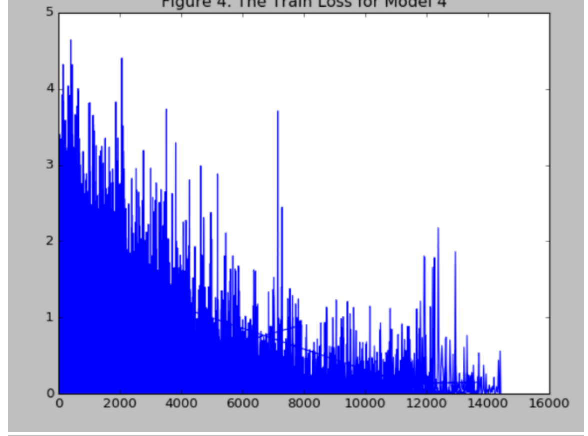
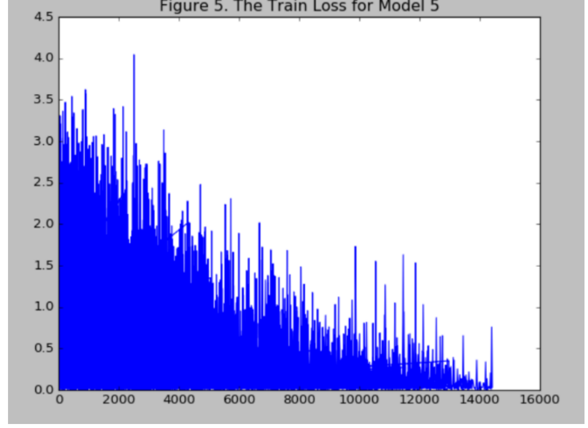


Figure 5. The Train Loss for Model 5



Conclusion

One obvious shortcoming of this project is data preprocessing, the future research could incorporate the following (some thoughts are from Prof. Jafari on his comments on presentation):

1. Use rotation or flip on the under represented classes and achieve a more balanced dataset.
2. Center crop as it may miss some important information contained in the periphery of the painting.
3. Add some dropout layers in model 3 because it may be overfitting. Even though model 3's train loss is near zero, its classification accuracy is no better compares to other models.
4. Pre-process the black and white paintings separately from colorful paintings.
5. Design a mechanism to choose the best mean and standard deviation for normalizing the images.

References

Wikipedia Page on Rprop: https://en.wikipedia.org/wiki/Rprop

Amir Jafari's Github on Pytorch CNN: https://github.com/amir-jafari/Deep-Learning/tree/master/Pytorch_/6-Conv_Mnist

Adit Deshpande, Beginner Guide on CNN: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Appendix

Please see the code py files.