# Art Classification Using Machine Learning

Andrew Levandowski, Alexander Lukito, and Jesus Aispuro
San Diego State University
5500 Campanile Dr.
San Diego, CA 92182
Emails: {andrewnlevandowski, alexanderlukito1}@gmail.com, jesus10_an@hotmail.com

***Abstract:* This report will explore the differences between different Convolutional Neural Networks and baseline models through the use of an art classification image recognition dataset. The CNN models include a custom variety, LeNet and AlexNet while our baseline models are the Support Vector Machine model and a Logistic Regression model. The dataset used features the five art types of drawings, engravings, iconography, paintings and sculptures. The performances of five different models will be compared and analyzed in order to make assessments of the parameters, methods and model specifics, like layer amount and type in the case of the CNNs, and the chosen dataset.**

## I. Introduction

### A. Task Description

In this project there are five datasets that each contain a different type of art: Drawings, Engravings, Iconography, Paintings, and Sculptures. Images in the datasets are of various sizes and quality.
The main task is predicting the art type for an input image. The performances of different machine learning models would be used to analyze the data and the models' effectiveness.

### B. Major Challenges and Solutions

Once we decided that our focus of this project would be about supervised image recognition, the decision to use a Convolutional Neural Network was immediate. The first challenge became choosing what other models we would use to make meaningful comparisons to the main neural network. We wanted to find other models and algorithms that would reveal their strengths and weaknesses when put to the test of our image recognition dataset. Because of this, we selected a more basic, probabilistic model, Logistic Regression, and a non-probabilistic model that's stronger with feature recognition, Support Vector Machines. On the other hand, we chose additional Convolutional Neural Network models that had been successful in standardized image recognition challenges that would fair a better chance and likely outperform our own version of a Convolutional Neural Network. From those highly developed, and often well funded, CNNs we chose

LeNet and AlexNet.

The next challenge was to decide on a specific dataset that added a new challenge to image recognition that we had not necessarily confronted in our course. With that in mind, we found a dataset that had more than 2 classes, specifically 5, and that had more complex features. Some of these features were more concrete, like the texture or material used to create the artwork, while others were more nuanced like the subject or style of the artwork.

This decision also lead to further challenges as we proceeded with color images as opposed to grayscale images. This effectively triples the amount of data and changes the samples to be made of three vectors for each pixel, one for each color channel, instead of one. This meant that our models had to be modified to handle larger and more complex data. Ultimately, we were able to handle the larger amount of data, but the large variety of features would prove quite difficulty for our models to identify, even after some tweaking.

One final challenge was again about the amount of data. Technological limitations of the computers available to us meant that the amount of samples used would need to be decreased. Due to performance, refraining from using all the data available to us at one time allowed us to test training results more often and in some cases allowed the training process to even complete before encountering memory or processor issues. This meant we would have more time to change the parameters of the model being tested and perform more analysis.

## II. Methods and Results

### A. Data and Models

*Raw Data*

The original dataset is downloaded from Kaggle as 2 folders, a training set and a validation set. Both of those folders contain 5 subfolders, one for each category of art. The training set folder has a total of 7,721 images (1107 drawings, 757 engravings, 2,077 iconography, 2,042 paintings, and 1,738 sculptures). The validation set has a total of 856 images (122 drawings, 84 engravings, 231 iconography, 228 paintings, and 191 sculptures). In total there are 8,577 RGB images of various sizes and quality in the dataset after removing corrupted images.

*Data Preprocessing*

Considering the large amount of images and our limited computing power, we decided to limit our input training data to 1000 images (200 from each category) and our input test data to 200 images (40 per category).

As all of the images are already sorted into appropriate subfolders (i.e. all images of drawings are contained in a subfolder called drawings), we decided to make our code iterate through the first 200 images (30 if in the validation folder) of every subfolder. Each of those images were converted into a numpy arrays, reshaped into dimensions of 64 by 64 pixels, and assigned a label based on the current subfolder. We utilized the openCV library to shrink the images with area interpolation to preserve details.

The shapes of our X_train.npy, Y_train.npy, X_test.npy, and Y_test.npy files respectively are: (1000, 64, 64, 3), (1000, 5), (200, 64, 64, 3), and (200, 5)

*Custom CNN*

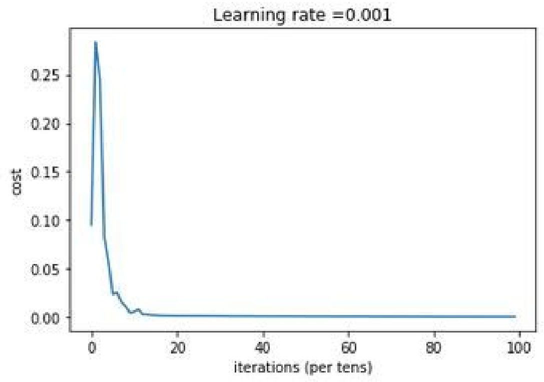
For our Custom CNN, we experimented with various hyperparameters and parameters and ended up with the following model: conv2d -> relu -> conv2d -> relu -> conv2d -> maxpool -> conv2d -> relu -> maxpool -> dropout -> fully-connected -> fully-connected.

The conv2d layer followed by a relu layer/maxpool layer is a pattern that occurs often among well-known models such as LeNet and AlexNet which are discussed below. In an attempt to decrease overfitting, we added a dropout layer before 2 fully-connected layers which was inspired by AlexNet.

Specifically, our 4 convolutional layers used the following weights respectively: 4 by 4 filter and 8 kernels, 3 by 3 filter and 12 kernels, 5 by 5 filter and 36 kernels, and 3 by 3 filter and 48 kernels. After our convolutional layers we applied a dropout rate 0.5 then had two fully-connected layers, outputting 20 and 5 respectively.

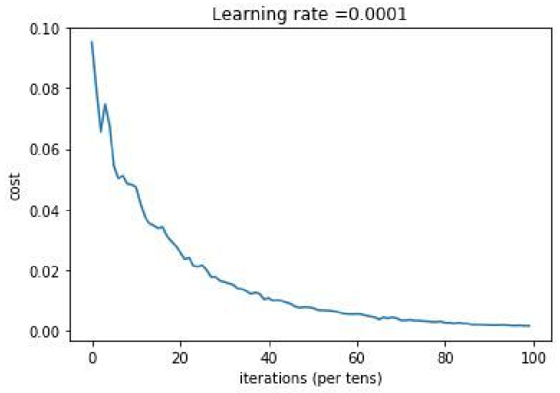
Images below are the results for the learning rates 0.001 and 0.0001.

```
Cost after epoch 80: 0.000578
Cost after epoch 85: 0.000554
Cost after epoch 90: 0.000500
Cost after epoch 95: 0.000455
```



```
Tensor("Mean_1:0", shape=(), dtype=float32)
Train Accuracy: 0.822
Test Accuracy: 0.565
Total run time: 1639.0989000581765
```

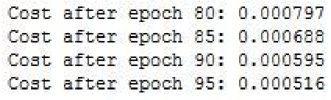
```
Cost after epoch 80: 0.002664
Cost after epoch 85: 0.002438
Cost after epoch 90: 0.001957
Cost after epoch 95: 0.001773
```



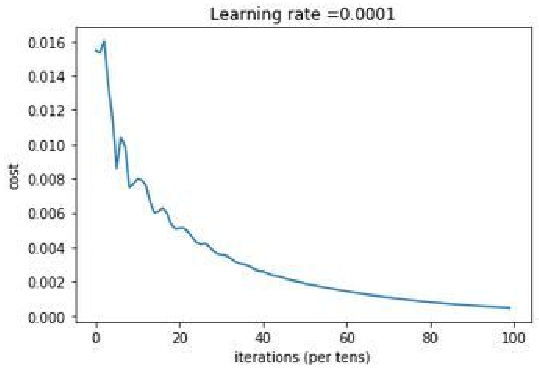
```
Tensor("Mean_1:0", shape=(), dtype=float32)
Train Accuracy: 0.706
Test Accuracy: 0.615
Total run time: 1791.9976860026957
```

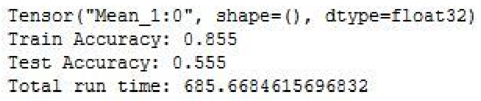
*LeNet-5*

LeNet-5 is an architecture containing 2 convolution layers, 2 average pool layers, and 1 fully-connected layer. The original model was applied to the MNIST dataset which utilizes images of size 32 by 32 by 1. As such, we adapted the model to fit our dataset containing 64 by 64 by 3 images. The first convolutional layer contains 6 filters of size 5 and stride of 1. The next layer is an average pooling layer of filter size 2 and stride of 2. Third layer is another convolutional layer with 16 filters of size 5 and stride of 1. Fourth layer is another average pooling layer of filter size 2 and a stride of 2. Fifth layer is a fully connected convolutional layer with 120 units. The final layer is another fully connected convolutional layer with 5 units instead of 10 (as MNIST had 10 categories where we have 5). One aspect that we changed was utilizing relu as our activation as opposed to tanh as relu provided better results.



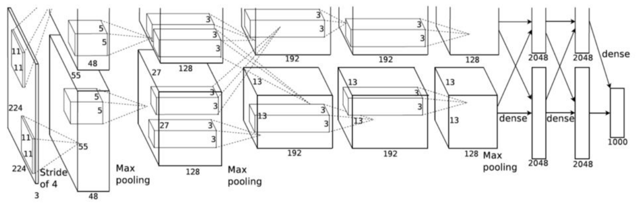
```
Cost after epoch 80: 0.000797
Cost after epoch 85: 0.000688
Cost after epoch 90: 0.000595
Cost after epoch 95: 0.000516
```



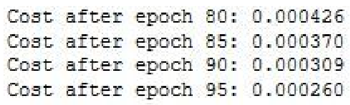


```
Tensor("Mean_1:0", shape=(), dtype=float32)
Train Accuracy: 0.855
Test Accuracy: 0.555
Total run time: 685.6684615696832
```

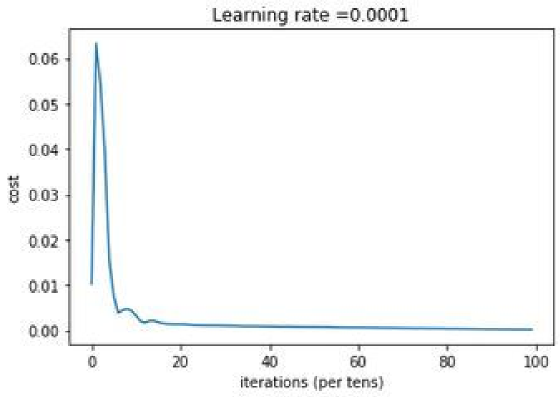
*AlexNet*

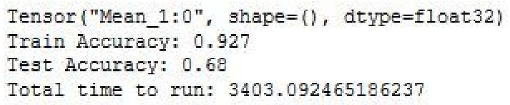


Similar to LeNet-5, we modified AlexNet to take images of 64 by 64 by 3 as the original dimensions as specified by the paper (227 by 227) would simply take too long to train for the amount of images in our dataset. AlexNet consists of 5 convolutional layers and 3 fully connected layers. All of the other hyperparameters/parameters should be the same as the original specifications on the paper.



```
Cost after epoch 80: 0.000426
Cost after epoch 85: 0.000370
Cost after epoch 90: 0.000309
Cost after epoch 95: 0.000260
```





```
Tensor("Mean_1:0", shape=(), dtype=float32)
Train Accuracy: 0.927
Test Accuracy: 0.68
Total time to run: 3403.092465186237
```

*SVM*

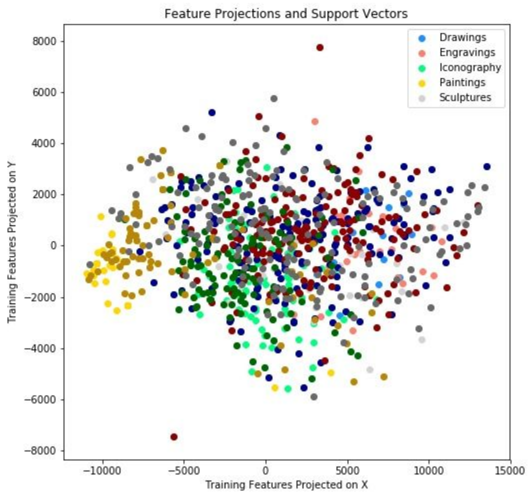
The main task of a Support Vector Machine model is to classify its data by identifying a decision boundary. A decision boundary can be thought of a line that is drawn between data points which separates them into different classes based on their features.

An optimal decision boundary is found by maximizing the margin from the data. This means you compare the all the distances between the proposed optimal decision boundaries and their nearest data point. The decision boundary is furthest from the nearest data point is the optimal one.

Many datasets have complex features which do not allow the data to be separated linearly. In this case, a kernel method can be applied which transforms the data into another dimension. In this dimension, data that was once not linearly separable can now be separated, though the separation is no longer linear.

Once the transformation occurs, the decision boundary is no longer a line in the new dimension. Instead, the data is separated by a plane if the transform is three dimensional and separated by a hyperplane if the transformation is a above three dimensional.

Train Accuracy of 1.0
Test Accuracy of 0.605
We found that a polynomial kernel gave us the best results.



*Support vectors are the darker points.

*Logistic Regression*

As opposed to the similar Linear Regression model, the probabilities that the Logistic Regression model uses for training are based off of a logarithmic cost function. This means it is often better suited for binary classification because data of two classes doesn't always fit a linear cost function.

Like Linear Regression, Logistic Regression attempts to model the relationship between a dependent and one or more independent variables. In order to eventually predict the relationship when new data is encountered, gradient descent is used to maximize the Log Likelihood (or Probability).

Train Accuracy of 1.0
Test Accuracy of 0.515
The newton-cg solver ,which uses Newton's Method to minimize our cost function, and the use of up to 1000 iterations provided the best results.

B. Evaluation Metrics

The table below is for the 3 CNN models that we implemented/adapted for our data at a learning rate of 0.0001.

| Learning Rate = 0.0001 | Custom CNN | LeNet-5 | AlexNet |
|---|---|---|---|
| Training Accuracy | 0.706 | 0.855 | 0.927 |
| Test Accuracy | 0.615 | 0.555 | 0.68 |

In terms of performance, AlexNet clearly did the best as the model had the highest training and test accuracy. However, it would seem that both AlexNet and LeNet experience greater overfitting compared to the Custom CNN. Overall, the CNN models did not perform great which could be due to various reasons. For example, the original images were of various sizes (such as 1024 by 768) and shrank to 64 by 64. Due to the rescaling, many details of the image, possibly distinguishing features, were lost which would explain why accuracy on test data was so poor. Another reason could be implementation errors or a poor pick of hyperparameters.

Comparing the CNN models to the SVM and Logistic Regression models, the CNNs are moderately more successful on average. Due to the baseline models' higher efficiency in processing of binary data, they were not able to handle the feature rich five-class dataset. Predicting the probability or using a decision border to recognize where a given data point would fall in one of five classes is was not the core of these models' purpose when they were designed.

The gap between the two levels of models was smaller than expected likely due to the less optimized state of the CNNs. Since the baseline models were implemented using fully featured packages, they had more parameters to change when creating the model and had more room for optimization.

C. Analysis and Discussion

Our Custom CNN does not work as well as LeNet-5 or AlexNet. It nearly mimics AlexNet in structure - ours has 4 convolutional layers, 4 RELU layers, 2 maxpool, and 2 fully-connected layers compared to AlexNet's 5 convolutional layers, 5 RELU, 3 maxpool and 3 fully-connected layers. However, besides having more layers, AlexNet also has far larger kernels in the convolutional layers. AlexNet at one point has 384 kernels compared to our Custom CNN that at maximum had 48 kernels.

Although LeNet-5 has less layers, it does have a fully-connected layer that outputs 120 units which again is far more than our Custom CNN's fully-connected layer which outputs 20.

Our Custom CNN works but it does neither conv2d or fully-connected layers well. We don't believe there is much room for improvement for our baseline models as they are more limited in terms of the scope of their training ability. The approaches of both models were different but the results were similarly low. Compared to the CNNs, the disparity was less than expected due to the CNNs' greater need for fine tuning and their large variety of variables like their layer amounts, types and multiple parameters.

## III. Conclusion and Future Works

To conclude, the CNN's did not perform that much better on the test data in comparison to the SVM and Logistic Regression. AlexNet performed the best out the CNN's while our Custom CNN performed the worst. Of the baseline models, the SVM performed best.

In the future, we would like to implement our models with mini-batch for better generalization of data. It would also be great if we could run models on higher quality images as opposed to 64 by 64. With many features of the dataset being more subtle, being able to train the models more with larger datasets would improve the results of the CNNs. Also, within the same limitations of this project, we could attempt using a different dataset that is equally as interesting and practical with less features and classes.

## References

https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/

https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637