# Classifying Artwork Style with Machine Learning

**Hugh Han** **Benjamin Hoertnagl-Pereira**

**Cheng-Hao Cho** **David Li**

## Abstract

This paper examines whether machine learning and image analysis tools can be used to assist art experts in the classification of unknown or disputed paintings. Machine learning techniques have been suggested for many tasks, and are especially suited for those related to classification. We investigate the suitability of applying machine learning techniques to the problem of classifying art by style, time period, and other labels. Our approach is concentrated on using methods such as multi-class SVMs, boosted decision trees, and convolutional neural networks. Our results show that machine learning has potential to classify artwork by style and is powerful on a variety of data sets.

## 1 Introduction

Determining a paintings authenticity or classifying a previously unknown piece of artwork can be extremely challenging. Typically, art experts thoroughly consider many different forms of evidence before making any conclusions. Data such as correspondence from the artists lifetime and documents tracing the paintings history of ownership provide clues but not definitive answers. Other methods, such as technical analysis of the materials used to make the piece and the method of their preparation, study of the creative process as documented in underlayers of the painting (observed through X-ray and infrared imaging), and visual appearance and style of the work are compared against those of the potential artists other works. Even with experts in the field and extremely thorough analysis, the result may be inconclusive, leaving art experts bamboozled. Recently, it has been demonstrated that computational tools from image analysis and machine learning can provide an additional source of analysis of questioned paintings [2, 3]. This assumes that a painting may have previously unconsidered features, such as an artists brushwork, which might be found by machine learning methods and used as an additional piece of evidence to classify and authenticate. It is clear that there has been great focus on paintings in the field of machine learning; however, not much can be found on determining the genre or style of artwork.

## 2 Related Work

Research on applications of machine learning to artwork is not too common and has only surfaced within the last ten years. In 2008, Johnson et. al [2] presented an algorithm for verifying the authenticity of a particular painting, by considering very high-resolution images of Van Gogh paintings. In 2009, Polatkan et. al [3] examined if machine learning and image analysis could authenticate or identify contested or unknown paintings. However, the main technique used were hidden Markov trees (HMTs). This inspired students in Stanford University's Machine Learning class in 2010 who attempted to classify 1400 pieces of art by seven different artists, using multi-class SVMs.

Figure 1: From top left to bottom right : post-impressionist, realism, surrealism, expressionism, impressionism, pop art, romanticism, cubism, art deco

## 3 Data Collection

### 3.1 Dataset Creation

Originally, we intended to take our project through an entire machine learning development cycle, so we challenged ourselves to define and create our own dataset, instead of using publicly available material. We scraped uploaded image previews of artwork from the Museum of Modern Art (MoMA) and the art collection in the J. Paul Getty Museum. The images are publicly available on the web and available for fair use for research purposes. We scraped over 30,000 images over the two datasets, a significant increase over the work by Blessing and Wen, who used a dataset consisting of 1400 images. While the artists of our paintings is significantly more diversified, since we are attempting to classify by style and time period, the significantly increased training dataset size should help with our ability to classify artwork correctly. To make training easier, we also collected labels on the data, including the date of completion and artist nationality among others.

A significant amount of time of our project was spent on collecting and processing data – especially because we did not have access to powerful resources to do efficient image scraping and analysis. Also, inconsistencies across the MoMA and Getty datasets, such as invalid or filler images and missing labels among others, created issues that made the creation and formatting of our dataset non-trivial. Eventually, we decided to use a publically available Kaggle dataset that was discovered later into the project.

In hindsight, it may have been a more efficient (or smarter) use of our time to look for datasets that were standardized to begin with instead of scraping our own. Because the MoMA and Getty datasets performed suboptimally due to scraping issues, their results were not reported. Instead, the results using the Kaggle dataset available at `kaggle.com/c/painter-by-numbers/data` are reported in section 5. This dataset contained approximately 30,000 images as well, and consisted of a total of 25 labels which represented the style of each piece of artwork (e.g. expressionism, romanticism, etc.).

After our decision to use the Kaggle dataset instead of polishing up our scraper and processing utilities, we were able to focus more on the actual machine learning techniques we were interested to test for our project, rather than needing to focus more on the software engineering oriented aspects of our project, such as parsing labels, formatting class buckets, and optimizing the scraper itself.

### 3.2 Label Creation

Originally, attributes of each piece of artwork were scraped from the MoMA and Getty datasets using our scrapers `moma.py` and `getty.py`. To create the labels for the style of art, we chose intervals that corresponded with particular (vague) movements dependent on the time period. For example, all pieces of artwork after 1946 were labeled as "contemporary", all pieces of artwork between 1891 and 1945 were labeled as "modern", and so on.

With the Kaggle dataset, label creation was much easier in that the style of each piece of art was explicitly defined in the dataset itself. Because of the large variety of artwork styles, we decided to consider only the 5 most frequently occurring labels, and their corresponding pieces of artwork. The reason for this "pruning" was mainly because of the uneven distribution of pieces of artwork within each label, and its potential effect on overfitting the data to more frequently occurring styles. The most frequently occurring styles in order were the following: "impressionism", "realism", "romanticism", "expressionism", "post-impressionism".

### 3.3 Image Processing

Once we had obtained the required images and their respective labels, we needed to represent the image as a row vector in our design matrix. The most widely used approach to image encoding is unraveling the 2-dimensional array of pixel values into a single vector of values. Of course, to create our design matrix, we would need each image to have the same number of pixels, which lead us to three separate approaches.

#### 3.3.1 Aspect Ratio Preserve

One of our options is to preserve aspect ratio, but shoot for a uniform number of pixels in each image. Doing this would mean the vectors that represent the 2-dimensional array of pixel values are equal in length. Although this does preserve some of the original features of the painting (width/height), we lose resolution because of downscaling. It ended up being too impractical to scale down every single image in our dataset to be the same dimensions while preserving aspect ratios. So, we considered two options – enforcing a uniform shape by cropping or distortion.

#### 3.3.2 Center Crop

This first option involves taking a square of uniform size from the center of each image. While this means all the arrays of pixel values are identical in dimension, we lose a lot of features by simply cropping. Simply taking a square of uniform size is also difficult because all of the artwork is different sized, and so it's not clear how big of a square we should take. It's possible to choose the biggest centered square possible and scale down – this is like square scaling (discussed next).

#### 3.3.3 Square Scaling

The last option, square scaling, was the one we ultimately used to pre-process the scraped images. It involved scaling the image to a centered square and then scaling this square to a uniform size. In this way the images are processed and represented as vectors with the same number of features. This method doesn't lose too much of the original artwork in terms of color/relative location of colors, but it distorts shapes (both shapes of the artwork and within the artwork). This approach worked the best for us in terms of trade-off between preserving features and ease of processing.

# 4 Approach

For each machine learning model (e.g. support vector machine, neural network, etc.) that was used, $k$-fold validation was first used to experiment with small partitions of the training data, to avoid the time cost of processing the entire dataset into memory. Values of $k = 3, 4, 5$ were used, and training using the entirety of the dataset was not used utilized until it was confirmed that our code worked to its intended purpose.

## 4.1 SVM

Linear, polynomial, and RBF kernel support vector machines (SVMs) were used for this project. For each SVM, principle component analysis (PCA) was used to greatly reduce the number of features in our design matrices, such that our experiments would be computationally feasible in a reasonable amount of time.

### 4.1.1 Linear

As we saw in lecture, SVMs generate linear decision boundaries by considering support vectors – those training examples closest to the separating hyperplane. Recall from lecture that the form of the trained linear SVM classifier in the linearly separable case is

$$\widehat{y} = \text{sign}\left\{\widehat{w}_0 + \sum_{\alpha_i > 0} \alpha_i y_i x_i \cdot x\right\}.$$

If our data are not linearly separable, then we can use the kernel trick to work in the feature space $\Phi$, where they are linearly separable. Specifically,

$$\widehat{y} = \text{sign}\left\{\widehat{w}_0 + \sum_{\alpha_i > 0} \alpha_i y_i K(x_i, x)\right\},$$

where $K(x, z) = \phi(x) \cdot \phi(z)$ and $\phi$ is the mapping into the feature space $\Phi$.

The original linear SVM we use uses $K(x, z) = x \cdot z$. We first used this kernel to verify that our code was, in fact, learning. The performance of this kernel will be discussed later in 5

### 4.1.2 Polynomial Kernel

Polynomial kernels are defined as

$$K(x, z; b, p) = (b + x \cdot z)^p,$$

where $b$ and $p$ are hyperparameters – $b$ is a constant bias, and $p$ is the degree of the polynomial.

We ran tests with polynomial kernels of degree 2,3,4 with varying terms for $b$.

### 4.1.3 Radial Basis Function

We also tried using a radial basis function kernel. This is defined as follows:

$$K(x, z; \sigma) = \exp\left(-\frac{1}{\sigma^2}\|x - z\|^2\right).$$

This kernel can be intuitively thought as a measure of similarity between two examples. Unlike the linear kernel or polynomial kernels, the feature space of the RBF kernel is infinite-dimensional – the best representation for this function as a power series requires infinitely many terms! Furthermore, note that we have a hyperparameter $\sigma$. As $|\sigma|$ decreases, the function learns finer features – and is more prone to overfitting. Typically, $\sigma$ is defined in terms of d, the dimensionality of each sample - a default is normally keeping $\sigma$ on the same order of magnitude of d.

### 4.2 Boosting

In addition to investigating SVM's, we looked into two boosting techniques: decision stumps and decision trees. Adaboost ("adaptive-boosting") learns by fitting multiple decision stump estimators in a sequential manner, determining the importance of classification for specific samples by how well they were classified by previous estimators. Gradient Tree Boosting using a similar principle to learn, however it uses slightly more complex decision trees as its base classifier, instead of the decision stumps of Adaboost. For both of these learners, the learning rate $\alpha$ can be used as a regularization parameter - for decision trees the default value of $\alpha = 1$; however, for Adaboost we found tuning this parameter improved results, as will be discussed later. Also, n is defined as the number of learners, which is a shared parameter. The parameter depth only pertains to decision trees, and represents the maximum depth of each individual regression estimator.

### 4.3 Neural Networks

Similarly to the SVM approach, training a neural network is computationally intensive, and was difficult with our large sample size. To solve this, parallelization methods were implemented using TensorFlow such that training of the network was able to be run on the GPU. This led to a great increase in speedup, and was a large factor in the feasibility of our project.

As discussed in lecture, neural networks are especially useful for pattern recognition. Because the task of classifying pieces of artwork greatly involves finding patterns within images, we hypothesized that neural networks would perform the best, compared with our other approaches in doing this task. Furthermore, previous research has suggested that convolutional neural networks (CNNs) are particularly performant for image feature extraction and pattern recognition. Hence, the majority of the time was spent building and training a convolutional neural network, and comparing its performance to an ordinary neural network, along with SVM methods.

We trained a convolutional neural network with many different combinations of layers (see Table 3). We tried flattening layers, fully connected layers, and convolutional layers with ReLU (regularized linear unit) as our activation function. In an effort to regularize, we tried both dropout and early stopping – our results are listed in the table.

## 5 Results

We used 12,000 images which was was partitioned into training and validation sets. The results on the validation set after training on the training set for each model is produced below. In addition to the accuracy metric, we examine the F1 score in order to ensure that our model is not overfitting to a single class.

| Classifier | Hyperparameter(s) | Regularization $C$ | Validation Accuracy | F1 score |
|---|---|---|---|---|
| Linear SVM | None | 10 | 0.248 | 0.246 |
| Polynomial Kernel SVM | $b = 0, p = 2$ | 10 | 0.303 | 0.292 |
| Polynomial Kernel SVM | $b = 0, p = 3$ | 10 | 0.298 | 0.287 |
| Polynomial Kernel SVM | $b = 0, p = 4$ | 10 | 0.297 | 0.282 |
| RBF Kernel SVM | $\sigma = d/0.01$ | 1 | 0.276 | 0.142 |
| RBF Kernel SVM | $\sigma = d/0.003$ | 1 | 0.359 | 0.352 |

Table 1: SVM results.

Overall, the relative performance of our SVM models performed as expected, with the linear variant performing worst and the RBF performing best. Our initial results of $> 24\%$ accuracy on the linear SVM shows that our model is learning, given that random guessing with 5 classes would yield expected accuracies $\leq 20\%$. At first, while training the RBF kernel SVM with the default hyperparameters for $\sigma = d, C = 1$,but it massively overfit by classifying every image to a single

class. After tuning the regularization parameter $C$ without improvement, we realized that our $\sigma$ was still too small, and increased it by a few orders of magnitude in relation to the number of features. Intuitively, this makes sense given that there is high variability between images.

| Classifier | Hyperparameter(s) | Validation Accuracy | F1 Score |
|---|---|---|---|
| Gradient Tree Boost | $n = 5$, depth 1 | 0.306 | 0.302 |
| Gradient Tree Boost | $n = 100$, depth 1 | 0.351 | 0.340 |
| Adaboost Decision Stump | $n = 1000, \alpha = 0.03$ | 0.372 | 0.331 |

Table 2: Boosting results.

We were very impressed with the performance of boosting, considering the simplicity of the decision stumps as a base classifier. Even with only 5 base estimators, we were able to train orders of magnitude faster that the linear and polynomial SVM's while outperforming these models. After increasing the estimators to 100, our boosted decision stump model was able to compete with our RBF kernel SVM.

| Classifier | Layers | Validation Accuracy |
|---|---|---|
| Fully Connected NN | 1 Flattening, 2 Fully Connected | 0.453 |
| Convolutional NN | 1 Convolutional, 1 Flattening, 1 Fully Connected | 0.281 |
| Convolutional NN | 1 Convolutional, 1 Flattening, 2 Fully Connected | 0.461 |
| Convolutional NN | 3 Convolutional, 1 Flattening, 2 Fully Connected | 0.430 |
| Convolutional NN | 1 Convolutional, 1 Dropout (50%), 1 Flattening, 2 Fully Connected | 0.375 |
| Convolutional NN | 1 Convolutional, 1 Dropout (25%), 1 Flattening, 2 Fully Connected | 0.316 |

Table 3: Neural Network Results

As expected, we had most success with neural networks. We noticed that our validation accuracy typically leveled off around $40\%$ while the testing accuracy continued to increase to $> 90\%$, a clear sign of overfitting. To combat this, we included dropout layers between the fully connected portions of our model, which prevented testing accuracy to greatly surpass validation accuracy; however, while using dropout did prevent overfitting, it prevented our model from achieving the validation accuracy peaks found in other models. Note that for our neural network testing, we did not work with the F1 score, instead simply using the confusion matrix to determine the nature of errors and ensure no one class was being overfit to.

In general, it was somewhat difficult to tune the neural network in terms of the number of layers and combination of layers. One issue was that the data took a long time (roughly 15 minutes per run) to load into memory, and training/testing took proportionally as long. Furthermore, we basically did our parameter tuning by trial and error – there's no systematic method to

## 6 Conclusion

From our results, we saw that neural networks, on average, greatly outperformed the SVMs and decision stump methods that we tried. However, we should also note that the RBF kernel SVM with well-tuned parameters along with the adaboost decision stump models were able to perform quite decently–outperforming some of the neural networks with a suboptimal configuration of layers. Ideally, we could have used the full +30,000 image dataset instead of our smaller 12,000 image dataset for training, since this would probably have reduced the overfitting by allowing for more varied examples for each class; however, we simply did not have the proper resources to run these simulations in a feasible amount of time.

Overall, while maximum accuracies in the range of 46% seem quite low for most applications, these metrics are quite good considering the high variability in artwork even within a single style, and the sometimes ill-defined classification of these styles. Upon inspection of the confusion matrices for classification, we found that our models consistently had most difficulty separating Impressionism and Expressionism. In theory, Impressionism focuses on how light interacts with objects, with emphasis on a broader perspective instead of details (eg "Grainstack", by Monet). On the other hand, Expressionism focuses on eliciting emotion and specificity instead of a scene (eg "A Green Thought in a Green Shade", by Helen Frankenthaler). It is clear that the qualities defining style is flexible and unclear at times.

## References


[1] Alexander, J.A. & Mozer, M.C. Template-based algorithms for connectionist rule extraction. In G. Tesauro, D. S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609-616. Cambridge, MA: MIT Press. 1995.

[2] C. R. Johnson, E. Hendriks, I. Berezhnoy, E. Brevdo, S. M. Hughes, I. Daubechies, J. Li, E. Postma, and J. Z. Wang. *Image processing for artist identification: Computerized analysis of Vincent van Goghs brushstrokes*. in IEEE Signal Processing Magazine, 2008.

[3] G. Polatkan, S. Jafarpour, A. Brasoveanu, S. Hughes, and I. Daubechies. *Detection of Forgery in Paintings Using Supervised Learning*. In IEEE International Conference on Image Processing, 2009.

[4] Blessing, A., Wen, K. *Using Machine Learning for Identification of Art Paintings*. For Stanford CS229 Machine Learning. 2010.

[5] J. Zhu, H. Zou, S. Rosset, T. Hastie. *Multi-class AdaBoost*. Statistics and Its Interface Volume 2, pp. 349360. 2009.