Measuring Aristic Similarity of Paintings

Jay Whang
Stanford SCPD
jaywhang@stanford.edu

Buhuang Liu
Stanford SCPD
buhuang@stanford.edu

Yancheng Xiao
Stanford SCPD
ycxiao@stanford.edu

Abstract

In this project, we survey various approaches to the task of artist similarity detection, where a machine is asked to determine if two paintings are by the same artist. Specifically, we compare the performance of manually-engineered features to that of a neural network in both online and offline settings. We also present two different ways to treat training data and show which one achieves superior accuracy.

1 Introduction

Recent advances in deep learning have culminated in several breakthroughs in computer vision tasks, some even outperforming humans [7]. In particular, Gatys, et al. [4] showed that it is possible to capture and mimic artistic styles of different painters. More recently, Dumoulin et al. [3] trained a single network that is able to combine multiple styles and apply them on any image in real time.

Based on the success of above projects, a Kaggle [1] challenge was started where participants were asked to train a model that can determine whether two given paintings were drawn by the same painter or not.

In the past, some attempts have been made to detect artist styles of paintings. The most widely-used approach involves hand-engineered features based on Computer Vision techniques and using a clustering algorithm such as K-nearest neighbors to classify paintings [6]. In another paper, Lee [9] extracted features by applying statistical methods on hue, brightness and saturation. He also segmented the image using k-means clustering and computed the regions in the images that are characteristic of the painting's style.

2 Data and Features

The data is obtained from the official Kaggle competition *Painter by numbers* [1]. The training data contains 79433 paintings coming from 1584 artists in JPEG format, with 23817 paintings in the test set. While the dataset contains other metadata such as title, genre and style (e.g. Cubism), only the actual image data was used for training and testing.

2.1 Pairwise Examples

As described in the Methods section below, some of our models were trained on pairs of images. Given nearly 80,000 paintings, however, there are over 6 billion distinct pairs of images. This was prohibitively large for most of the offline training algorithms we used. We tried training models on tiny subsets of all pairs (e.g. 600,000 pairs, which is only 0.01% of 6 billion), but virtually all models failed to generalize to unseen image pairs.

Thus, we decided to restrict ourselves to the top 100 most prolific painters (by number of paintings) and only used 100 paintings from each of them. We then enumerated all possible same-artist

(positive) and different-artist (negative) pairs from the chosen 10,000 paintings, and sampled equal number of positive and negative examples to form the training set.

Because we wanted to investigate the effect of training data on model performance, we constructed four training sets: `mini`, `small`, `medium`, and `large`, with 4,000, 10,000, 40,000 and 80,000 pairs, respectively.

We also constructed two test sets. `test1` is built from previously unseen paintings by the *same* 100 painters in the training set. `test2` is built from paintings from new painters that are not included in the training set. Both `test1` and `test2` have equal number of positive and negative examples as well, meaning **random guess would achieve around 50% accuracy**. This served as a good baseline metric to measure model performance.

2.2 Individual Examples

We also used an *embedding* approach, where a model is trained to generate a low-dimensional embedding of a given image. Then we can use some distance (e.g. cosine similarity) of two embedding vectors for a pair of images to predict if they are by the same artist.

To make a fair comparison, we again restricted the ourselves to the same 100 painters as above. After excluding all paintings that appear in `test1` and `test2`, we obtained a training set with 25348 individual images.

2.3 Features

We wanted to explore how well hand-engineered features could perform compared to a neural network-based approach. To capture stylistic information, we chose features that represent lighting, lines, texture and color of the painting.

- arithmetic mean and the standard deviation of the pixel intensity of the image
- percentage of dark pixels (intensity below 65) in the image [6]
- number of local and global maxima which can be used to estimate the use of light in the image [6]
- number of local maxima in the image by finding the peaks in a luminance histogram [6]
- mean, variance, skewness, and kurtosis of the 256-bin image intensity histogram
- mean, variance, skewness, kurtosis of the hue histogram [9]
- color depth, defined as the size of the file divided by the total number of pixels squared
- aspect ratio of the image
- total number of edge pixels divided by the total number of pixels in the painting, using Canny, Prewitt and Sobel edge detectors [10] to detect the edges. This effectively represents "edge density."
- number of lines that span more than 7 pixels, calculated using Hough transform. The idea is that certain styles like Cubism have many long running lines while Impressionist paintings do not.
- length of the longest straight line in the image
- number of corners detected by Harris corner detector
- entropy, a measure of randomness of the pixel values of the image, to capture texture [11].

3 Methods

3.1 Combining Feature Vectors

Because each image was featurized into a vector as described above, each example pair produced two feature vectors v_1 and v_2. We experimented with various ways to combine them into a single input into a binary classifier. Specifically, we considered the following three options:

- Concatenation: $v_{input} = [v_1 v_2]$
- Difference: $v_{input} = v_1 - v_2$
- Positive Difference: $v_{input} = |v_1 - v_2|$

3.2 Traditional Techniques

Using combined feature vectors, we trained several binary classifiers using the following algorithms:

- *Logistic Regression* estimates the probabilities of y given feature x using a sigmoid function(g), with the hypothesis defined as

  $$\mathtt{h}_\theta(x) = g(\theta^T x) = \frac{1}{1+\epsilon^{-\theta^T x}}$$

  For a binary classification problem, the likelihood of training data $\{(x^{(i)}, y^{(i)}\}$ becomes

  $$\mathtt{L}(\theta) = \prod_{i=1}^{m} (\mathtt{h}_\theta(x^{(i)}))^{y^{(i)}} (1 - \mathtt{h}_\theta(x^{(i)}))^{1-y^{(i)}}.$$

  We employed 5-fold cross validation to find optimal θ.

- *Support Vector Machine* (SVM) is a margin-based classifier that minimizes the following hinge-loss with regularization penalty:

  $$\mathtt{J}_\lambda(\alpha) = \frac{1}{m} \sum_{i=1}^{m} \left[1 - y^{(i)} K^{(i)T} \alpha\right]_+ + \frac{\lambda}{2} \alpha^T K \alpha,$$

  where labels $y \in \{-1, 1\}$, $K \in R^{m \times m}$ is the kernel, and $\lambda > 0$.
  We trained SVMs with linear, polynomial and Gaussian kernels. For example, the Gaussian kernel is defined as

  $$K(x, z) = \exp(-\frac{1}{2\tau^2} \|x - z\|_2^2),$$

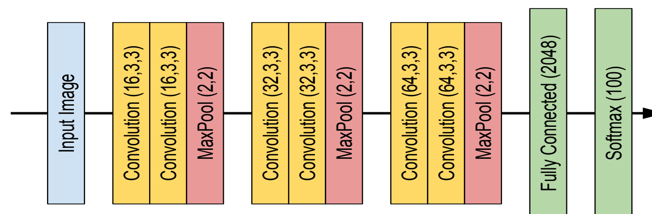
  where $\tau > 0$ is the bandwidth parameter for the kernel.

- *Random Forest* is an ensemble learning method where multiple decision stumps are combined to produce a model. It is known to generalize better (i.e. less overfitting) compared to decision tree learning. For our models, we limited the tree depth to 10 and used 200 estimators.

3.3 Image Embedding by Convolutional Neural Network

Instead of dealing with such large number of pairs from all training images, we can *learn* an embedding for each image. To train such embedding, we chose to use a convolutional neural network (CNN) based on its recent success in vision tasks. Specifically, we trained a CNN to predict the painter of a given image. The final layer of the network – the predicted class distribution (*softmax*) over all painters – was then used as the embedding. Finally for a given pair of paintings, we used the cosine similarity of the two embedding vectors to determine if they are by the same painter.

Below is the specific architecture of the CNN we used:



- `Convolution(n_filters, width, height)`: A convolutional layer with `n_filters` filters of size `width`-by-`height`
- `MaxPool(w, h)`: A max-pooling layer that reduces each `w`-by-`h` block in the input image into a single pixel
- `FullyConnected(dim)`: A hidden layer of size `dim` that is fully connected to the previous layer

Table 1: Performance of different algorithms on test set 1

Method—Test set 1	train_mini	train_small	train_medium	train_large
logistic regression	0.5350	0.5418	0.5388	0.5393
SVM with linear kernel	0.4344	0.4317	0.5628	took too long
SVM with polynomial kernel	0.5281	0.4722	took too long	took too long
SVM with rbf kernel	0.500	0.500	took too long	took too long
random forest	0.6495	0.6531	0.6604	0.6653

Table 2: Performance of different algorithms on test set 2

Method—Test set 2	train_mini	train_small	train_medium	train_large
logistic regression	0.5114	0.5166	0.516	0.5172
SVM with linear kernel	0.4567	0.4497	0.5492	took too long
SVM with polynomial kernel	0.5357	0.5123	took too long	took too long
SVM with rbf kernel	0.5013	0.501	took too long	took too long
random forest	0.5834	0.5828	0.5890	0.5923

- `Softmax(dim)`: A softmax layer of size `dim` that output the predicted probability distribution over all classes (i.e. painters)

To provide a fixed-size input to the network, we used 128-by-128 crop from the center of the image.

4 Results and Analysis

4.1 Feature Combination

We discovered that among the three ways to combine feature vectors mentioned above, only *positive difference* allowed for models to make any sort of progress, which matched our intuition. Take logistic regression for example. Since it produces a linear decision boundary, there is no way to correlate the corresponding components from two feature vectors when they are concatenated. And for difference, note that the sign plays an important role. If the model was shown two pairs (v_1, v_2) and (v_2, v_1), the combined feature vector would be exact opposites ($(v_1 - v_2)$ vs. $(v_2 - v_1)$), even though their labels are the same. Thus, these two example pairs would negate each other, and the net result of training would be effectively null.

Theoretically, a SVM with non-linear kernel on concatenated feature vectors could pick up the correlation across two feature vectors. In practice, this was not the case. Non-linear SVMs overfitted and failed to generalize to test sets for concatenated feature vectors.

4.2 Performance of Different Methods

Out of all the methods that we used, Random Forest gave the best performance while SVM with Gaussian kernel gives the worst performance. For each method, we have trained on all four training sets and tested on both `test1` `test2`. We used **classification accuracy** as the metric. Table 1 and Table 2 summarize our findings.

Besides Random Forest, it seems like other traditional techniques either failed to learn or barely learned anything. We think there are several reasons. Most importantly, our features are not representative enough to really differentiate artistic styles from different painters. Looking at the images in training data, we noticed that even the same painter often exhibits multiple styles. In the train

Table 3: Performance of neural network

test set—training size	1000	2000	5000	10000	25000
test set 1	0.55359	0.56105	0.57335	0.59535	0.62721
test set 2	0.52276	0.52044	0.53386	0.54360	0.55511

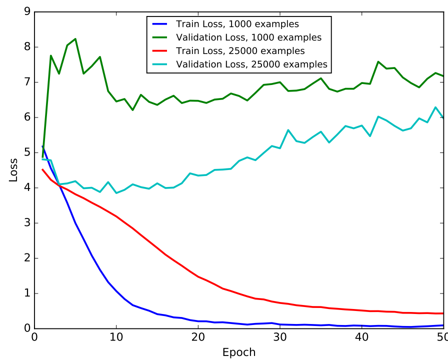


Figure 1: Train vs. Validation Loss for CNN

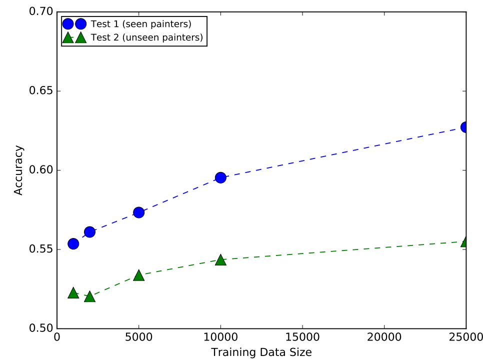


Figure 2: Accuracy vs. Size of Training Data

and test sets we constructed, a painter could have up to ten distinct styles, ranging from cubism to impressionism. In fact, we think it is hard to manually construct these representative features. Also because features are not representative, models with high capacity do not really work, since they easily overfit to the train set (e.g. the Gaussian kernel).

Convolutional neural network, while still behind Random Forest, was able to surpass 60% accuracy on `test1`. This result was surprising because the accuracy on validation set during training never went beyond 20%. This means that even though the model fails to correctly classify painters, the final output of the network still captured some of the stylistic information.

4.3 Generalization to unseen data

By comparing the performance on `test1` and `test2`, we can see how our models generalize to previously unseen painters. While random forest on `test1` (same painters as train set with new paintings, Table 1) achieves 66.5% accuracy, it manages to get 60% on `test2` (completely new painters, Table 2).

The situation is similar to CNNs. As shown in Figure 1, the validation loss actually increases during training, even though the training loss decreases. We also observed that this behavior was consistent for different values for regularization. This suggests that our CNN model does not have the sufficient capacity required to generalize to unseen paintings.

4.4 Effect of Training Data Size

We also observed that as training set increased, performance on both test sets also increased (refer to Table 1 and 2), suggesting that the scalability of the model (which restricts the size of the training data) is a major bottleneck. In fact, CNN benefited the most from larger training data, as shown in Figure 2, on both `test1` and `test2`.

5 Conclusion and Future work

Increasing performance from larger training data suggests that training a bigger model, especially for the CNN, could push the accuracy even further. While we believe that manual feature engineering cannot outperform CNN on this problem when full data set is used, carefully engineered features with random forest can likely do much better than 66% on both test sets. For CNN, we believe that instead of using cosine similarity, training a model – such as Siamese Network – to measure similarity on top of the embedding vectors could push the performance even further.

Interestingly, the Kaggle challenge ended as we were working on this project. The winner of the challenge was able to achieve over 90% accuracy with a CNN, trained on a NVidia Titan X GPU for 300 epochs over several days [5]. This validated our analysis that with a bigger dataset, CNN can outperform other methods.

References

[1] Painter by numbers — kaggle. `https://www.kaggle.com/c/painter-by-numbers`. Accessed: 2016-11-20.

[2] F. Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[3] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. *CoRR*, abs/1610.07629, 2016.

[4] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[5] N. Ilenic. Winning solution for the painter by numbers competition on kaggle. `https://github.com/inejc/painters`, 2016.

[6] T. E. Lombardi. The classification of style in fine-art painting. *ETD Collection for Pace University*.

[7] C. Lu and X. Tang. Surpassing human-level face verification performance on LFW with gaussianface. *CoRR*, abs/1404.3840, 2014.

[8] B. Saleh, K. Abe, R. S. Arora, and A. M. Elgammal. Toward automated discovery of artistic influence. *Multimedia Tools Appl.*, 75(7):3565–3591, 2016.

[9] E.-Y. C. Sang-Geol Lee. Style classification and visualization of art paintings genre using self-organizing maps. *Human-centric Computing and Information Sciences*.

[10] I. The MathWorks. Edge detection methods for finding object boundaries in images, 2016. Accessed: 2016-12-12.

[11] I. The MathWorks. Entropy of grayscale image, 2016. Accessed: 2016-12-12.

[12] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation, 2001. Accessed: 2016-12-12.