# Distinguishing Professional and Amateur Abstract Art Using Convolutional Neural Networks

Sharman Tan
sharmant@stanford.edu

Category: Computer Vision

December 8, 2019

## Introduction

In 2007, Freddie Linksky fooled the world into paying large numbers for his abstract paintings – people didn't realize they were a two-year-old's ketchup art [9]. Distinguishing between professional and amateur artwork is vital when evaluating its commercial value. In 2010, psychologists Hawley-Dolan and Winner found that even untrained adults can distinguish between professionals' and children's abstract works [4]. Can a machine learn to do the same?

CNNs generally perform well on many image classification tasks that require distinguishing between the objects represented by images. The task of classifying professional and amateur abstract artwork may be challenging for humans, and research thus far has called into question the balance of content-based and style-based information necessary for models to perform well on such tasks [6][5][7]. In this paper, I demonstrate that by applying deep learning techniques to address issues of high variance, even standard CNN architectures are able to successfully distinguish between most professional and amateur artwork.

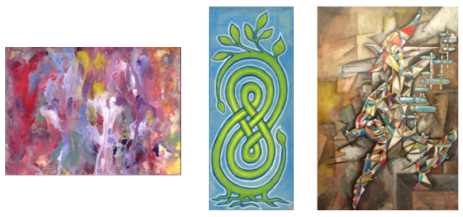

Figure 1: deviantArt images

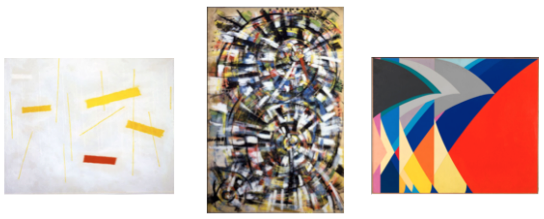

Figure 2: MART images

## Data

### deviantArt

DeviantArt, the world's largest online community for amateur and professional artists and art enthusiasts, features millions of artworks and has over 44 million registered members [2]. [8] selected 500 amateur artworks by 406 different authors from deviantArt by filtering to abstract art, manually removing paintings with recognizable objects, and selecting a range of most to least favored paintings. I manually downloaded each of these paintings from their links in [1] to get a dataset of 440 artworks (60 unavailable). I labeled these images 0, corresponding to "amateur."

Table 1: Model Architectures

| **Baseline Model** | | **Shallow Model** | |
|---|---|---|---|
| Layer | Output | Layer | Output |
| Input | 3, 512, 512 | Input | 3, 512, 512 |
| Conv1 (f=3, p=1, s=1) * 8 | 8, 512, 512 | Conv1 (f=3, p=1, s=1) * 8 | 8, 512, 512 |
| ReLU | 8, 512, 512 | ReLU | 8, 512, 512 |
| MaxPool2D(2,2) | 8, 256, 256 | MaxPool2D(2,2) | 8, 256, 256 |
| Conv2 (f=3, p=1, s=1) * 16 | 16, 256, 256 | Conv2 (f=3, p=1, s=1) * 8 | 8, 256, 256 |
| ReLU | 16, 256, 256 | ReLU | 8, 256, 256 |
| MaxPool2D(2,2) | 16, 128, 128 | MaxPool2D(2,2) | 8, 128, 128 |
| Conv3 (f=3, p=1, s=1) * 32 | 32, 128, 128 | Conv3 (f=3, p=1, s=1) * 16 | 16, 128, 128 |
| ReLU | 32, 128, 128 | ReLU | 16, 128, 128 |
| MaxPool2D(2,2) | 32, 64, 64 | MaxPool2D(2,2) | 16, 64, 64 |
| Conv4 (f=3, p=1, s=1) * 32 | 64, 64, 64 | Conv4 (f=3, p=1, s=1) * 16 | 16, 64, 64 |
| ReLU | 64, 64, 64 | ReLU | 16, 64, 64 |
| MaxPool2D(2,2) | 64, 32, 32 | MaxPool2D(2,2) | 16, 32, 32 |
| FC1 | 256 | FC1 | 256 |
| ReLU | 256 | ReLU | 256 |
| FC2 | 84 | FC2 | 84 |
| ReLU | 84 | ReLU | 84 |
| Dropout | 84 | Dropout | 84 |
| Softmax | 2 | Softmax | 2 |

### MART

The MART collection contains over 20,000 artworks from 1913 to 2008 by Italian, European, and American artists [3]. With an art historian, Sartori et al. selected 500 images [1] of abstract paintings from this collection. I labeled these images 1, corresponding to "professional."

### Preprocessing & Augmentation

First, I resized the images to be the same dimensions (512 × 512 pixels) using bicubic interpolation. Then, I mean-centered and normalized each of the images between 0 and 1 inclusive. I then store each of the images as a 512 × 512 × 3 dimensional matrix and finally randomly split the 940 images into train/dev/test sets using a 60/20/20 split.

In half of my experiments, I use this data to train and evaluate my models. In the latter half of my experiments, I instead train my models with more data by performing vertical flips on the preprocessed images. Specifically, I first vertically flip all the preprocessed images to get a total of 1880 images, double of what I originally had. Then, I randomly split these images into train/dev/test sets using a 60/20/20 split. When training, I train on this new set of training data (containing 1128 images). However, I keep my dev and test sets the same as before (containing 188 images non-augmented images each) for evaluation.

## Method

**Note**: All code for my models is available at my Github repo *sharmant/abstract-art-classification*. All experimental results run on Google Colab are displayed in the Python3 notebook at
*https://colab.research.google.com/drive/1B4rgRkUycxYdySqgZUfcNjnq6afr0FeA*.

The architectures of models are outlined in Table 1. My **baseline VanillaCNN** consists of 4 CONV+ReLU+MaxPool layers followed by dropout and 2 fully connected layers, and finally a softmax layer that outputs probabilities of each class. I divide the non-data augmented training data into 8 equally sized batches and the data-augmented training data into 16 equally sized batches when training. I use an Adam optimizer with cross entropy loss to train the model.

The **ShallowCNN** model is a "shallow" version of the VanillaCNN in that the CONV layers have fewer filters, in an attempt to simplify the model overall. When I train these two models on the larger data augmented (DA) training set, I call these models **VanillaCNN-DA** and **ShallowCNN-DA**.

Table 2: VanillaCNN Training and Dev Accuracy (Tuning Dropout Probability)

| Dropout Prob | Learning Rate | Epochs | Train Acc (%) | Dev Acc (%) |
|---|---|---|---|---|
| 0.1 | 1e-4 | 50 | 97.16 | 73.40 |
| 0.2 | 1e-4 | 50 | 92.91 | 73.94 |
| 0.3 | 1e-4 | 50 | 99.29 | 69.68 |
| 0.4 | 1e-4 | 50 | 97.34 | 73.94 |
| 0.5 | 1e-4 | 50 | 98.94 | 69.68 |
| 0.6 | 1e-4 | 50 | 93.97 | 74.47 |
| **0.7** | **1e-4** | **50** | **99.11** | **75.00** |
| 0.8 | 1e-4 | 50 | 99.11 | 70.74 |

Table 3: ShallowCNN Model Training and Dev Accuracy (Tuning Dropout Probability)

| Dropout Prob | Learning Rate | Epochs | Train Acc (%) | Dev Acc (%) |
|---|---|---|---|---|
| 0.1 | 1e-4 | 50 | 89.72 | 73.94 |
| **0.2** | **1e-4** | **50** | **84.57** | **76.60** |
| 0.3 | 1e-4 | 50 | 87.77 | 70.74 |
| 0.4 | 1e-4 | 50 | 84.57 | 72.87 |
| 0.5 | 1e-4 | 50 | 82.62 | 71.80 |

Table 4: VanillaCNN-DA Model Training and Dev Accuracy (Tuning Dropout Probability)

| Dropout Prob | Learning Rate | Epochs | Train Acc (%) | Dev Acc (%) |
|---|---|---|---|---|
| **0.1** | **1e-4** | **100** | **99.56** | **93.09** |
| 0.2 | 1e-4 | 100 | 99.56 | 87.77 |
| 0.3 | 1e-4 | 100 | 99.56 | 90.96 |
| 0.4 | 1e-4 | 100 | 99.29 | 90.96 |
| 0.5 | 1e-4 | 100 | 99.02 | 87.77 |

Table 5: ShallowCNN-DA Model Training and Dev Accuracy (Tuning Dropout Probability)

| Dropout Prob | Learning Rate | Epochs | Train Acc (%) | Dev Acc (%) |
|---|---|---|---|---|
| 0.1 | 1e-4 | 100 | 96.28 | 88.30 |
| 0.2 | 1e-4 | 100 | 99.65 | 90.96 |
| 0.3 | 1e-4 | 100 | 98.76 | 91.49 |
| **0.4** | **1e-4** | **100** | **99.65** | **92.02** |
| 0.5 | 1e-4 | 100 | 99.02 | 89.36 |

Table 6: Final Tuned Model Train and Test Accuracies

| Model | Learning Rate | Dropout | Epochs | Train Acc (%) | Test Acc (%) |
|---|---|---|---|---|---|
| VanillaCNN (Baseline) | 1e-4 | 0.7 | 50 | 99.11 | 77.13 |
| ShallowCNN | 1e-4 | 0.2 | 50 | 84.57 | 78.19 |
| **VanillaCNN-DA** | **1e-4** | **0.1** | **100** | **99.56** | **93.09** |
| ShallowCNN-DA | 1e-4 | 0.4 | 100 | 99.65 | 92.55 |

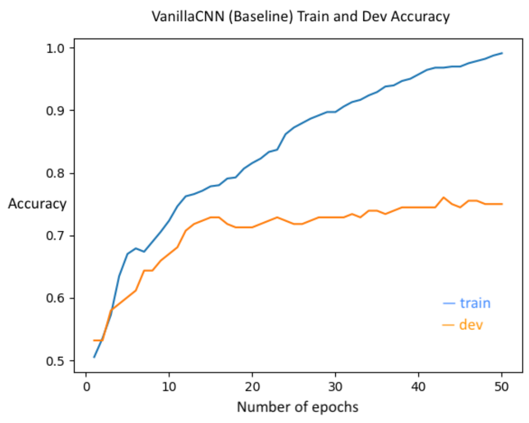

Figure 3: VanillaCNN (Baseline)

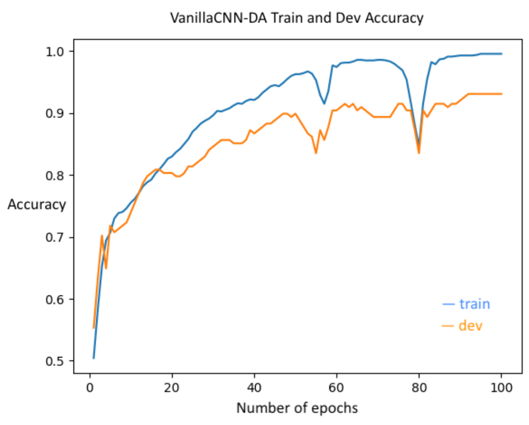

Figure 4: VanillaCNN-DA

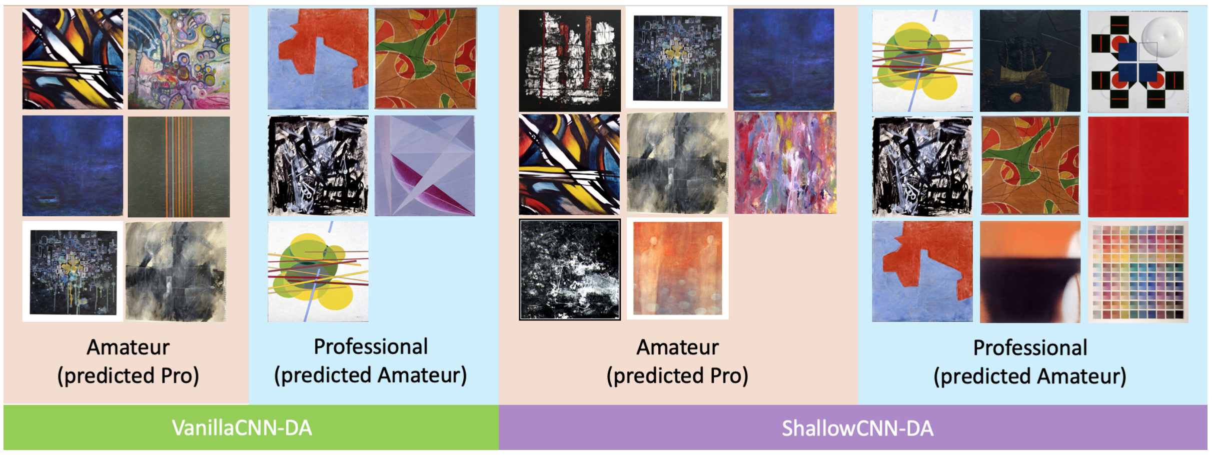

Figure 5: All Incorrectly Classified Images for VanillaCNN-DA and ShallowCNN-DA

## Experiments and Results

### More Epochs & Tuned Learning Rate

I ran each of my experiments on Google Colab using 1 NVIDIA K80 GPU. First, I increased the number of training epochs when training VanillaCNN to 50 (from 20 in the milestone) and tuned the learning rate (trying 1e-5, 3e-5, 1e-4, 3e-5, 1e-3) to find the optimal value of 1e-4. This drastically improved my results, achieving train and dev accuracies of 92.91% and **73.94%**, respectively. Although this is a major improvement from my results from the milestone (which was comparable to a random classifier), the VanillaCNN still suffered from high variance. This is evident from Figure 3, as the gap between the train and dev accuracy is significant.

### Tuned Dropout Probability & ShallowCNN

To address the issue of high variance, I tuned the dropout probability from its value of 0.2 and achieved train and dev accuracies of 99.11% and 75.00% respectively with a dropout probability of 0.7, as shown in Table 2. Then, I simplified the VanillaCNN model by decreasing the number of filters at most of the convolutional layers, resulting in the ShallowCNN model. After tuning the dropout probability of the ShallowCNN (see Table 3), I achieved train and dev accuracies of 84.57% and **76.60%**, respectively.

### Data Augmentation

Although tuning the dropout probability and simplifying the VanillaCNN increased my dev accuracy, the model still seemed to suffer from high variance. Therefore, I performed data augmentation by vertically flipping the images as detailed in the Preprocessing & Augmentation section. This doubled my training data while maintaining my original dev and test datasets.

I then trained the VanillaCNN and ShallowCNN on this training dataset and tuned the dropout probability for both models (Tables 4 and 5). VanillaCNN-DA achieved train and dev accuracies of 99.56% and **93.09%**, respectively, while ShallowCNN-DA achieved train and dev accuracies of 99.65% and **92.02%**, respectively. As the train and dev accuracy curves of VanillaCNN-DA demonstrate (Figure 4), augmenting the image data largely resolved the issue of high variance previously seen in the original VanillaCNN.

Table 6 displays the final train and test accuracies achieved by each of the 4 models. **VanillaCNN-DA** performs best, achieving a test accuracy of **93.09%** (identical to its dev accuracy), followed by **ShallowCNN-DA**, which achieves a test accuracy of **92.55%**.

## Discussion and Analysis

Each of the experiments and their corresponding train, dev, and test performances demonstrate that high variance was the main issue my baseline suffered from. Although tuning hyperparameters such as the dropout probability and simplifying the model architecture alleviated this issue, augmenting the data produced the most prominent improvement in accuracy. Vertically flipping abstract artwork may not seem to make much of a difference to the art, especially when compared to genres such as realism, but doing so was key to achieving impressive performance on this classification task.

To better understand how the models perform, I qualitatively evaluate their misclassified examples. Figure 5 displays all images misclassified by VanillaCNN-DA and ShallowCNN-DA. Notably, the misclassified images for both models are evenly split between the two classes, so it seems that the models perform almost equally well when classifying amateur and professional abstract artworks.

The misclassified amateur artworks seem to consist mostly of *highly complex and colorful* art, along with some simpler *monochrome* pieces. Professional abstract art may often consist of pieces in either extreme, and these may be difficult to distinguish based on skill. On the other hand, the misclassified professional artworks seem to be more *geometric* and often consist of *2 main colors*. Geometric shapes (such as the the solidly colored squares in the artwork in the bottom right corner) and larger bodies of color (such as the red/lavender and the olive green/orange pieces) may be simpler for amateur artists to create compared to more complex artworks. Therefore, the models may have learned to classify such pieces as amateur ones.

## Conclusion & Next Steps

Originally, because the problem of classifying amateur and professional abstract art may not simply be a content-based task, my goal in implementing a vanilla CNN for my baseline model was to understand how standard CNNs perform on this unique problem. However, after identifying high variance as the main issue and performing deep learning techniques to alleviate this issue, I was able to achieve an impressive test accuracy of 93.09% with VanillaCNN-DA. Therefore, CNNs are actually effective ways to tackle this task that humans have demonstrated difficulty with. Moving forward, I would like to explore more advanced models that may overcome the limitations my current models have when classifying specific types of abstract art.

## Acknowledgements

I would like to sincerely thank Huizi Mao for his valuable advice throughout this project and the rest of the CS230 staff for all that I have learned throughout the course. I would also like to thank Google for making free GPUs available through Google Colab.

## References

[1] Andreza sartori. http://disi.unitn.it/~sartori/. Accessed: 2019-10-10.

[2] Deviantart. https://www.deviantart.com. Accessed: 2019-10-10.

[3] Mart. http://www.mart.trento.it. Accessed: 2019-10-10.

[4] Angelina Hawley-Dolan and Ellen Winner. Seeing the mind behind the art: People can distinguish abstract expressionist paintings from highly similar paintings by children, chimps, monkeys, and elephants. *Association for Psychological Science*, 2011.

[5] Wei-Lin Hsiao and Kristen Grauman. Learning the latent "look": Unsupervised discovery of a style-coherent embedding from fashion images. *CoRR*, abs/1707.03376, 2017.

[6] Sergey Karayev, Matthew Trentacoste, Helen Han, Aseem Agarwala, Trevor Darrell, Aaron Hertzmann, and Holger Winnemoeller. Recognizing image style. *arXiv preprint arXiv:1311.3715*, 2013.

[7] Shin Matsuo and Keiji Yanai. Cnn-based style vector for style image retrieval. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, pages 309–312. ACM, 2016.

[8] Andreza Sartori, Victoria Yanulevskaya, Alkim Almila Akdag Salah, and Jasper R. R. Uijlings. Affective analysis of professional and amateur abstract paintings using statistical analysis and art theory. *The ACM Transactions on Interactive Intelligent Systems*, 2015.

[9] Ellen Winner. Could your child really paint that? *The Wall Street Journal*, 2018.