Classifying Paintings with an Artificial Neural Network

Isaiah King
May 2019
Accuracy: 82%

---

1. PROBLEM DESCRIPTION

The *Greatest Artworks of All Time* dataset contains over 8,000 paintings from artists of all genres, time-periods, and even mediums. There are over 40 different styles of artwork from over 60 painters, each tagged via filename and categorized by painter. My initial plan was to use clustering on the data to find out if artistic movements would be close together in higher-dimensional space. I quickly discovered this task was all but impossible. Images, though our brains are quite good at clustering them, don't lend themselves to being well-clustered by computers–at least not with KNN or DBSCAN. This is because a mere difference in color is enough to put paintings worlds apart. A dark, dim Rembrandt would have no chance of being placed near a bright, pastel Caravaggio no matter how similar their styles of painting are; the computer only sees them as long vectors with little to no similarity.

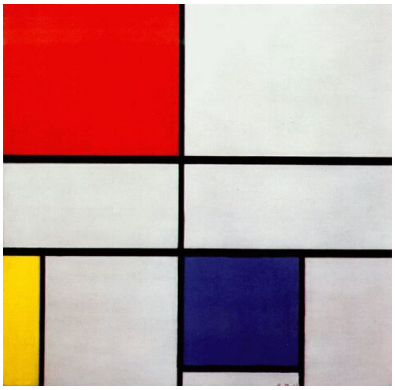
However, I found that though my initial goal was next-to-impossible, I may be able to extract something useful from the images using an artificial neural network (ANN). ANNs view image data in a much more contextual way through pooling and convolutional layers (more on this in the next section). This approach led me to learn a great deal about the construction of ANNs, how image classification works, and how to build more powerful models in the future. This paper will detail how I built a binary classifier to determine if a painting's style is closer to neoplasticism or renaissance with 82% accuracy.

2. METHODS

2.1 Data Preparation

In the form I downloaded it, the paintings in the dataset were tagged by the artist who painted them; I aimed to classify them by artistic movement. Artists, and artistic styles may be somewhat self-similar in form, but in general take on a tremendous array of aesthetic qualities. The differences between renaissance and baroque, for example, are difficult to distinguish even for humans. To make matters worse, artists–and especially great artists–often paint in varying styles. Picasso for example, painted in 3 very unique ways: his rose and blue periods, which were more or less realistic, if a little cartoonish, and his cubist style, which is radically abstract.

All of this is to say categorizing art by movement proved to be a very subjective task–a task made no easier when in its original form, the works were categorized by painter. Because of this ambiguity, while tagging the paintings I wanted to use the most objective standard I could: extremely distinctive art styles, and painters who used that style consistently. I finally settled on neoplasticism and renaissance.



Composition C., Piet Mondrain – Neoplasticism

Sistine Chapel, Michelangelo – Renaissance

Despite the careful selection of these tags there was still a great deal of noise and inconsistency in the data. There were sketches, and photos of sculptures, and duplicates of paintings in every folder. The data, I later learned, was scraped from various art gallery websites and put together rather haphazardly. Because of this, I had to painstakingly go through each image, one by one, and remove noisy documents. By the end of this process, there were just over 100 samples of each style. This lack of data later became problematic, so to compensate, I manually sorted through the WikiArt website[1] to download more public domain samples. In the end, however, I was only able to find about 50 more paintings in each class, making the dataset as a whole only 400 samples large. The exiguous size of the dataset necessitated a robust way of generating synthetic, or augmented samples (more on this in section 2.2).

The python library I used for this project has an excellent function to read data straight from folders, and infer their labels based on directory names. Because of this, all that was required to tag the data was to move images of the same class into the same directory. After that, I simply made a random partition in the data to quarantine 10% of it in a different directory forming a test set to make validation possible.

2.2 Loading the Images

The small and noisy dataset meant overfitting would be the most likely source of error. Without the proper precautions, the model would grow too fixated on small details in the paintings, and neglect to look at, literally, the bigger picture. I took several steps to compensate for this. The first, and least successful step to counter this was to use a dropout layer. This causes nodes in the ANN to randomly disconnect from the next layer. In theory, this serves to make the model less overfit, as it causes the details the model wants to fixate upon to suddenly become unimportant, and pushes it to look at the whole rather than its parts. These layers were not particularly helpful, however. During training, the model never converged. It instead swung randomly between 40-60% accuracy. Instead, I went with a preprocessing approach.

When working with very small datasets, it's often helpful to use a technique called data augmentation during preprocessing. This is a process that randomly flips, rotates, and changes the colors or brightness of input images as the model trains on them. In this way, a set of 100 images can be turned into a set of 100,000 just through random augmentation. Hernández-García and König have even argued that the practice of using dropout layers is wasteful, and if

one uses extremely liberal augmentation, such layers are not required.[2] With this in mind, I set the preprocessing function in the model to rotate the paintings up to 180°, flip them horizontally and vertically, zoom in on them by up to 20%, and shift their colors by up to 10%. Initially this value was 50%, but I later discovered by being less liberal with allowed color shifting, accuracy increased by an additional 5%. My reasoning was that by doing this, the model would become less focused on the relative details within the images, and more focused on the larger details shared between samples. This change was met with quite a bit of improvement; the model's accuracy increased by about 15%.

2.3 ANN Architecture

The python package Keras makes it very easy to build an ANN for classifying images. That is, easy for those who know what all the various layers actually do. Going into this project I did not, but from the Keras documentation, and several excellent papers on the subject of convolutional layers, I learned enough to have a basic grip on it.

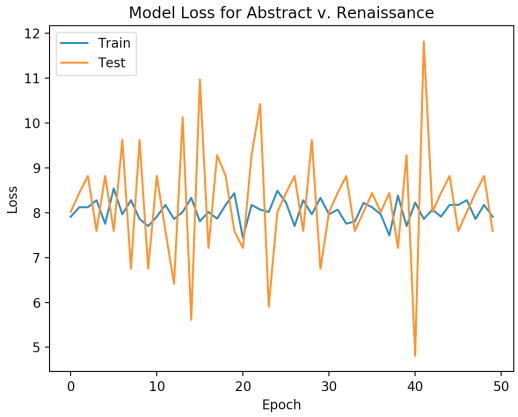
When attempting to classify images, very often what is called a convolutional layer is used. Convolutional layers look at images in a more holistic way than something like clustering does. Simard et al. state "spatial topology is well captured by convolutional neural networks.",[3] which is to say, models that employ convolutional layers do a better job of looking at big parts of images relative to each other. This is because the way convolutional layers work is by grouping large swaths of pixels into single nodes called kernels, rather than considering each pixel of an image individually. The way these pixels are combined is up to the pooling layer. My theory was that art pieces are similar to each other because each chunk of a given piece is most similar to chunks of other pieces in the same style. Because of this, to combine the vectors that made up chunks the model uses average pooling. This causes the pixels making up each kernel to be averaged together into one tensor. Each kernel is then processed as if it was one large vector and looked at as a whole in later fully connected layers.

In the best implementation, the model uses four convolutional layers. Each layer looks at successively smaller subsections of the image. In its initial form, it was loosely based on the structure used by James Le which employed the same hierarchical structure of convolutional layers.[4] However, I made some pretty radical modifications: I removed the dropout layers, added more convolutional layers with different kernel sizes, and added more dense layers to the end.

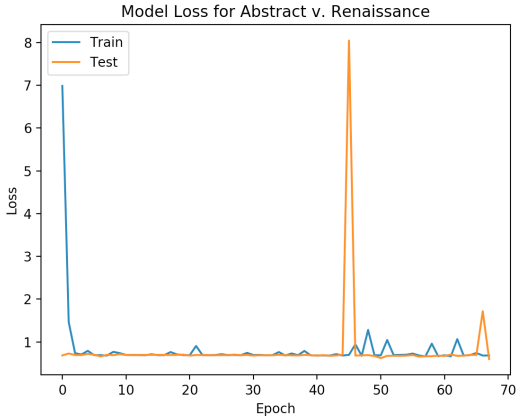
In full, the model consists of the following: four convolutional layers with kernel sizes 10x10, 5x5, 4x4, and 3x3 and 3 fully connected layers with input sizes of 500, 300, and 150, leading into two output nodes. Each layer uses an ReLU activation to avoid the sigmoid saturation problem save for the final layer, which uses softmax activation so the model can more easily push their values to one or zero. Several other architectures were tested, but this one consistently yielded the best accuracy. In earlier iterations, the model also had two dropout layers, one between the convolutional and fully connected layers, and one between the last fully connected layer and the output node, however I found augmenting the data to be more effective. I also found the addition of the final two fully connected layers improved the model's rate of convergence compared to the original, unedited structure used by Li.

The description of the ANN would be incomplete without discussing which optimiser was used. For my purposes, I found the default SGD optimiser to be lackluster. The loss during

training was far too irregular, bouncing randomly without convergence. To counter this, I selected the AdaDelta optimiser. This optimiser adjusts the model's learning rate in real time, and is best for use on small datasets. This is because it is especially optimal for noticing important, large-scale details.[5] This change was so significant in the result of the overall model, it may have been the only reason it worked at all. This becomes evident when looking at the change in model loss over time during training.



Loss over time with SGD



Loss over time with AdaDelta

As is clear from the graphs, before this optimiser was used the model had a very hard time converging. However, by changing to the AdaDelta optimiser the model was able to dynamically change the learning rate and fixate on the details that actually mattered. This was the key reason the model was able to work and was the last major addition. All other changes resulted in either negative, or no change in accuracy.

3. RESULTS

Before the change in optimiser, I was sure this task was impossible. However, with that simple change the classifier was able to predict if a painting was renaissance or neoplasticist with 82% accuracy. In the future, I would like to see how it would handle more, equally distinct classes. However, because the two I selected were so distinct, and because that style was so much of the focus of the painters who created them, I felt that for an introductory problem, this was a promising place to start.

4. CONCLUSIONS

4.1 Problems

This level of accuracy was only attainable through a long period of trial and error riddled with problems. The first problem I encountered was the data itself. It was noisy and poorly organized, yes, but the bigger problem was the lack of it. To build a really powerful ANN image classifier, more datapoints are necessary. A model with a chance of breaking 90% accuracy for a problem of this complexity would require a dataset on the order of 1,000 images for each class, even with data augmentation.[6] Though techniques were used to compensate for the lack of data,

there were some images in the test set that were just too tricky for the classifier to guess. More data could have helped to build an even more accurate model.

There are techniques that exist to aid in problems like this. One such example is transfer learning, the practice of taking a model somebody else already trained to do a similar task, and retraining it on new data.[7] This may have been a more successful route, but it feels near Faustian to just repurpose someone else's model. I wanted to create my own from scratch, even if it was imperfect.

4.2 What Did I Mine

Before the model could find patterns in the data, it had to clean and augment it. In attempts to find hidden patterns in the data, I was forced to think about what those patterns might be, and how the model might exploit them. To do this, I used convolutional layers to look at large parts of image data, which were irregularly shaped, sized and colored. All of this was in an attempt to discover what patterns still exist when features that are only relative (size, color, brightness, etc.) are stripped away. Unfortunately, it seemed my hypothesis that color was irrelevant was largely flawed; removing the color shifting aspect of the data augmentation process improved the accuracy of the resulting model. However, this means there is still some hidden pattern and maybe it's simpler than we think. Maybe classical art simply doesn't use some range of colors that modern art does. I hope the solution wasn't that simple, but there's no way to know.

The black-box nature of ANNs makes it difficult to determine what exact patterns it found in the data, so we can only speculate. From the preprocessing we can infer that that patterns cannot exist in the sizes and angles of items in the images alone, but without attempting to analyse the ANN (an impossible task) that's all we as outsiders can know.

4.3 What I learned

Before attempting this project, I had only a vague inkling of how image classification with ANNs worked. Now that I've completed it, I feel I have the knowledge required to build another from scratch. Both from a theoretical and a practical standpoint, this project forced me to dive head-first into the strange world of ANN architecture, theory and construction, and I feel I am better for it. It felt good to work on a problem I was really excited about and I hope I can extend upon the classifier I created for this assignment to make something even more complex. A neoplasticism vs. renaissance classifier is interesting, but I want to make something with a broader scope and higher accuracy.

I now understand how CNNs work, the practical aspects of building an image classifier, and what to look out for in future projects of this nature. I learned how easy it is to build an ANN and how hard it is to build a good one. Most important of all, I learned to appreciate how complex seemingly simple tasks are. I knew at the beginning that identifying artistic genres is a tricky problem, but I severely underestimated its moiling complexity. I still think a model could be made with even higher accuracy, but that would require more data, more research, and more knowledge on the part of its creator. I hope to someday revisit this project. Maybe after a few more years of school I'll be able to finally conquer it, but for now, I'm satisfied in the knowledge that I tried and 82% is the best I can get. Now that I have seen how difficult this task was for a computer, I can fully appreciate the complex tasks our brains manage to do without breaking a sweat.

REFERENCES

1. "Artworks by style: Neoplasticism." [Online]. Available: https://www.wikiart.org/en/paintings-by-style/neoplasticism#!#filterName:all-works,viewType:masonry
2. A. Hernández-García and P. König, "Do deep nets really need weight decay and dropout?" *CoRR*, vol. abs/1802.07042, 2018. [Online]. Available: http://arxiv.org/abs/1802.07042
3. P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis," *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, 2003. [Online]. Available: http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2016/pdfs/Simard.pdf
4. J. Le, "How to easily build a dog breed image classification model," Mar 2019. [Online]. Available: https://medium.com/nanonets/how-to-easily-build-a-dog-breed-image-classification-model-2fd214419cde
5. W. Cheng, "Quick notes on how to choose optimizer in keras," Mar 2018. [Online]. Available: https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/
6. J. Brownlee, "Impact of dataset size on deep learning model skill and performance estimates," Dec 2018. [Online]. Available: https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/
7. "You can probably use deep learning even if your data isn't that big." [Online]. Available: https://beamandrew.github.io/deeplearning/2017/06/04/deep_learning_works.html