Paint by GAN

Vanessa Grass

University of New Haven / GalvanizeU

July 22, 2017

Introduction

Generative Adversarial Networks make use of both generative and discriminative models. Generative models describe how data is generated, in terms of a probabilistic model. More concretely, a generative model tries to generate the distribution of $P(x|y)$, where x is an observation in the dataset and y is some class label. This is in contrast to discriminative models, which do not concern themselves with how the data is generated, they simply categorize a given datapoint. In terms of probability, they aim to find the distribution of $P(y|x)$, that is the probability of a class label y given some data x. Discriminative models essentially model the decision boundary between classes. Generative models, on the other hand, are very interesting in that they can be used for more than generating class predictions. Such models are well suited to impute missing data or even generate new data.

In a generative adversarial network (GAN), the generator makes use of a generative model, taking random input values and transforming these inputs into images via a deconvolutional neural network. Although the generation component of a GAN is very exciting, the discriminator, which employs the use of a discriminative model, is equally crucial. The discriminator is essentially a binary classifier that evaluates whether a given generator created image looks like a real image (from the original dataset) or like an artificially created fake image. The discriminator takes the form of a standard convolutional neural network (CNN).

Over the GAN's training process, the weights and bias in both the discriminator and generator are updated through backpropagation. And in this way, the discriminator learns to distinguish the real images from the fake images created by the generator. The generator simultaneously uses feedback from the discriminator to learn to produce increasingly more convincing fake images that ultimately the discriminator can no longer differentiate from the original real images.

Though a large part of my work here was in understanding (or attempting to) the deeper workings of GAN architectures to ultimately build a GAN that can generate novel art from images of historical paintings, I first build a standard CNN that solves a binary classification problem, classifying paintings created by artist Paul Cezanne and Vincent van Gogh. The basic premise behind the concept of a CNN is that it takes as input an image and then through a series of several layers recognizing increasingly complex features, the CNN outputs a single value — in the case of this project, whether the image is a Cezanne or van Gogh painting. This is a valid starting off point as such a model has a architecture very similar to that of the discriminator in a GAN, in fact, the discriminator in a GAN is also solving a binary classification problem, classifying real from fake images. My initial steps in creating this

binary classification CNN was first exploratory data analysis, then establishing a non-neural network baseline model, moving on to prototyping a simple CNN, and ultimately building a deeper CNN that differentiates Cezanne paintings from those of van Gogh.

Data Overview and EDA

The painting dataset used in this project comes from a Kaggle competition "Painter by Numbers" with the majority of images coming from wikiart.org. The dataset is comprised of 103,250 total images — 79,433 labeled as training and 23,817 labeled as testing. Within the dataset there are 42 unique genres represented — for example, portrait, landscape, and abstract genres. In terms of painting styles, there are 135 unique painting styles, ranging from styles like Impressionism and Expressionism to Realism, and many more in between. A total of 2,074 artists are represented in the dataset. For the purposes of my project, I chose to focus on only the artists Cezanne and van Gogh, the reason being both are artists whose works share similarity in color and content but not necessarily exact style (though both are considered to be Post-Impressionistic artists).

Data Preprocessing

The images in the Kaggle painting dataset vary greatly in pixel size and aspect ratio, ranging anywhere from 30,000 by 29,605 pixels on the larger end to 283 by 558 pixels on the smaller end. To make this image data machine learning friendly, I created a script that resizes all images to square 72 by 72 pixel dimension. As this kind of resizing causes distortions in non-square images, there may be better solutions such as some kind of cropping technique, however simple resizing seems to be fairly standard in similar computer vision problems.

For the purposes of binary classification, I subset the original 103,250 images into only Cezanne and van Gogh. Respecting Kaggle's training and testing split, this subset resulted in 804 training images (comprised of 412 Cezanne paintings and 392 van Gogh paintings) and 189 testing images (comprised of 87 Cezanne paintings and 102 van Gogh paintings). As the number of painting images per artist are not the same, there is the issue of class imbalance, minimal in the training set but more pronounced in the testing set. I address this issue in more detail in the modeling sections to follow.

Additional preprocessing involved normalizing the pixel intensity values to 255 either by simple NumPy vectorized division or by use of Keras `ImageDataGenerator` **class. The** `ImageDataGenerator` **class was also used to augment the painting images via random transformations. Typically these transformations allow for better model generalizations, as this lets the model learn variations in orientations, rotation, etc. However, whether it is appropriate to apply such transformations on this dataset is something that warrants further experimentation. A concern is that the transformations influence the CNN's ability to discern the artist's style. In particular, one of the** `ImageDataGenerator` **arguments,** `fill_mode,` **which determines the method in which to fill empty pixels caused by the transformations, causes a streaking effect in the image when set to "nearest". Using "reflect" is likely a**

better bet as it does not cause this streaking appearance which potentially could be wrongly perceived as an artistic style by the CNN.

Baseline Models

As my project aims to generate images, I created a baseline “generative” model by averaging together images to create composite images of subsets of the data. Generally, I found that averaging together all images of a particular painting category/grouping creates very fuzzy image — there is likely a sweet spot where averaging a certain number of similar images would result in a "better" result (I had considered using KMeans to group similar images within higher level groupings such as genre and may try such an approach in the future just as a fun experiment). Interestingly, averaging together all portrait paintings reveals a discernible pattern — a central area where faces likely occur and a circular/oval frame popular in certain art history periods (for example Italian Renaissance). Averaging together all landscape, Impressionism, Cezanne, and van Gogh paintings essentially results in something just very muddied overall. To be clear, averaging together these images is a terrible baseline “generative” model and the muddied appearance was to be expected.

For my baseline classification model, I trained a Random Forest classifier to discern between Cezanne and van Gogh. I didn’t try anything fancy and just used the default scikit-learn model parameters. This resulted in a Random Forest classifier with an accuracy of 0.65, which is better than guessing. As the data has class imbalance accuracy is not the best evaluation metric, so I also looked at f1-score which was also 0.65.

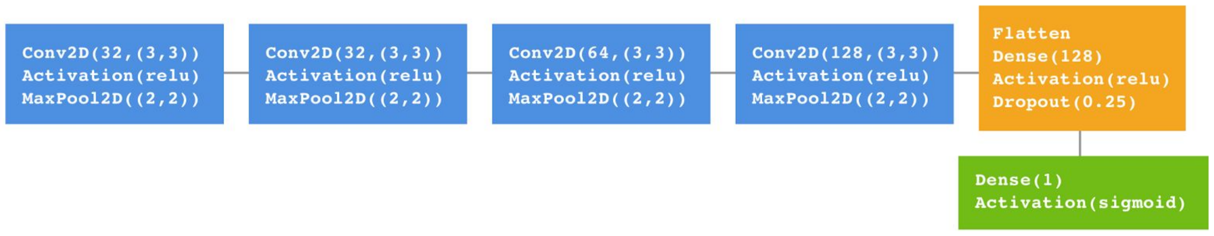
Looking into error analysis and examining which paintings the model classified most easily, which it was most fooled by, and most confused by hints to the model's understanding of the difference between the two artists. The Random Forest classifier tends to over-predict Vincent van Gogh even though there are slightly more Cezanne paintings (~20) in the training set. A cursory examination seems to suggest blues and reds confuse the model into thinking a painting is a van Gogh. I also returned feature importances from the Random Forest classifier which I initially found very interesting. In the case of images, the feature importances, as I understand, are the pixel location in the image. While the importances are very low values, there are about 10 locations that seem more important than other positions. I suspect these pixel locations are somewhere near the center of the image. However after further consideration, this this interpretation has it’s shortcomings as looking at precise pixel values is likely too granular to generalize across different images.

Simple Convolutional Neural Network Model

Next I implemented a very simple CNN and overfit on a small subset of the data (3 random paintings from Cezanne and van Gogh). This allowed me to quickly prototype a CNN architecture with my dataset. The simple CNN architecture consists of one convolutional layer outputting 32 filters with a rectifier linear unit (ReLU) as the activation function followed by one max-pooling layer, then a flatten layer followed by the one and only final dense layer with a sigmoid activation as this is a binary classification problem. For the same reason I also compiled this model using binary cross-entropy for the loss function. For the gradient descent optimizer I used RMSprop. This model's accuracy stabilizes to 1.0 after about 7 epochs. Once I had this architecture established, I trained the model using the entire dataset for 50 epochs arriving at an accuracy of 0.70 and a f1-score 0.68. Surprisingly, this is only slightly better than the Random Forest classifier. Curious to see if adjusting for class imbalance improves matters at all, I randomly sampled the larger class based on the smaller classes size. Running this data through the same simple CNN resulted in 0.71 for both accuracy and f1-score. This only slightly better performance over using the class imbalanced dataset.

"Deeper" Convolutional Neural Network Model

Building on the above simple CNN, I added three more convolutional and max-pooling layers, first holding the number of convolutional filters at 32 and then doubling for the additional two convolutional layers. I continue to use ReLU activation and added an extra flatten and dense layer with 128 units (neurons) as well as employed a dropout layer set at 0.25. The dropout layer randomly deactivates some neurons in a given layer helping to ameliorate potential overfitting to some extent. Again, I used both sigmoid as the final activation and compiled with binary cross-entropy loss. While I did not perform gridsearch (but do want to want to try out Sacred in the future), I did manually experiment with different hyperparameters, varying the number of hidden layers and trying out different activation functions (for example, ELU and Hard Sigmoid) but did not see any improvements in either accuracy or f1-score with these hyperparameters. However, using Adam for the gradient descent optimizer did yield a higher f1-score in my experimentation. See the diagram below for the full architecture.



I trained this model for 100 epochs on a Google Cloud instance with 16 vCPUs, 104 GB memory, and 2 NVIDIA Tesla K80 GPUs using the image provided by Stanford's CS231n course. Training for longer epochs did not improve accuracy and can even result in overfitting. In evaluating this model's performance I came across an important "gotcha" in Keras when using the `.flow_from_directory` method on an `ImageDataGenerator` object — the order of the classes which map to the label indices are alphanumeric. This initially caused a mix up in how I decoded the probabilities and class labels returned when evaluating the model on the validation set as I had previously been arbitrarily labeling van Gogh as class 0 and Cezzane as class 1. Essentially my evaluation metrics were inverted as I had the class labels switched. After clearing up this mix up, I'm happy to report the model's accuracy is 0.80 with a f1-score of 0.76, which is an improvement over the simple CNN previously mentioned.

In looking into error analysis, nothing stood out except for a few instances when the model confidently predicted Cezanne but the actual artist was van Gogh. In these instances I was struck by the similarity in style between the two artists. I further validated the model by testing two additional unseen images that are in fact not even actual paintings from either artist but are actually "fake" images generated by applying the artist's style derived from a neural style transfer onto photos. Testing a canonical but "fake" Cezanne, my model accurately labels it as a Cezanne (with a probability of 0.17 which is good — the more confident the model is the painting is a Cezanne the closer the probability should be to zero, as Cezanne is labeled as class 0). My model also accurately labels a canonical but "fake" van Gogh but only by a very small margin (here the probability was 0.53 which is not so good — if it were more confident this probability would be closer to 1.0). Nevertheless, considering the small dataset this model was trained on, overall the model performs quite well. Obviously training on more paintings from each artist would likely lead to better performance. Another idea is to employ transfer learning, training a model on a larger subset of the painting data, for example the top five painting genres and perhaps even paintings styles, then leveraging the weights learned to help improve the binary classification of Cezanne and van Gogh. Additional transfer learning to consider is making use of much larger pre-trained models such as VGG-16 or Inception (though I'm not quite sure how well that would work with this particular dataset as they are not photo-realistic images).

An Attempt at a Generative Adversarial Network

The final component of my project, but certainly not the least, is attempting to build a GAN that produces images which reasonably resemble paintings from the Kaggle painting dataset. I experimented with a "vanilla" GAN architecture (as opposed to other GAN architectures such as AC-GAN, StackGAN, or WGAN). In my vanilla GAN, the discriminator was comprised of four convolutional layers, each with leaky ReLU activation and generous dropout. The use of leaky ReLU helps to prevent "saturated gradients", a phenomenon that happens easily in GAN architectures. Saturated gradients occur as gradients (or the change in the model weights) are multiplied together (via chain rule) during backpropagation and through activation either become so small they tend towards zero or become very large they tend toward infinity. This effectively causes the neuron to no longer learn anything. A leaky ReLU allows for small negative values that attempts to fix the "dying ReLU" problem. Similar to the CNN built in my binary classification problem, the discriminator outputs a 1-dimensional probability vector via a flatten and dense layer with sigmoid activation.

The generator in this architecture takes as input a uniformly distributed noise vector. From this fake images are generated from essentially the inverse of convolution, employing several transposed convolutional (deconvolutional) layers. The generator also makes use of some more unfamiliar layers (at least to me) such as upsampling (which as the name suggests, increases the image dimensions) and batch normalization which stabilizes learning by normalizing the activations of the previous layer. Again similar to the discriminator, the generator also gets a generous dose of leaky ReLU and dropout, and ends with a flatten and dense layer again with sigmoid activation. Each model is then compiled using RMSprop as the optimizer and binary cross-entropy as the loss. Then the generator and discriminator are the finally stacked together to create the GAN.

While I was able to implement the above GAN architecture to produce reasonable results using MNIST within 1,000 training steps, I was unable to yield satisfactory results using the Kaggle painting dataset. Essentially the generator images still look too much like noise. At this time my best guess is this is due to the aforementioned issue of saturated gradients (according to some sources leaky ReLU is inconsistent in solving this issue) and warrants further exploration. One newer GAN architecture that seems promising is called a Wasserstein GAN (WGAN) and leverages the Wasserstein metric, which is a distance function between probability distributions. As I understand, the addition of this distance function helps stabilize the GAN. In short, I was not able to get reasonable decent results from the architecture I employed and plan to investigate other GAN architectures such as the WGAN in the future.

References

[1] Building powerful image classification models using very little data, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[2] Commonly used activation functions, http://cs231n.github.io/neural-networks-1/

[3] GAN by Example using Keras on Tensorflow Backend, https://medium.com/towards-data-science/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0

[4] GANGogh: Creating Art with GANs, https://medium.com/towards-data-science/gangogh-creating-art-with-gans-8d087d8f74a1

[5] Generative Adversarial Nets, https://arxiv.org/pdf/1406.2661.pdf

[6] Machine Learning is Fun Part 7: Abusing Generative Adversarial Networks to Make 8-bit Pixel Art, https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7