Classifying Painters Using a Convolutional Nueral Network

Kyle Nelson
Auburn University
KEN0018@auburn.edu

Henry Roden
Auburn University
HDR006@auburn.edu

Kevin Alfrod
Auburn University
KJA0018@auburn.edu

Abstract

In our final project, we will learn what a convolutional neural network is. Then we will apply what we have learned to classify paintings based on who painted them.

1 Introduction

For our project we wanted to use machine learning of some kind. However we wanted to do something kind of fun. After a lot of discussion we talked about ways that an AI could eventually determine what makes art special and what makes certain artists so iconic. That, however, was a big task that we did not know how to even start on. Eventually we decided that the first step in this process would be to create some kind of neural network that could look at a painting and determine what artist it came from. If we could make an AI 'see' what features of a painting belong to a specific artist, then we would have a network that was on its way to understanding art. After some research we found that the best way to classify an image based on the features in it would be to create a Convolutional Neural Network. In this paper we will talk about how a CNN works as well as how we successfully implemented one to determine artists with a high accuracy.

2 Related Works

There has been some previous work done with using CNNs detecting if a painting was done by a specific artist, specifically from Nitin Viswanathan at Stanford [1]. In their report they used similar data to us (discussed in section 3) and they used the same type of model. They used 300 images each from 57 different artists, pruning the pictures as they went to only use the best from each artist. In addition they used three different CNN models. One was a baseline CNN and two were built off the ResNet architecture. They were able to get 62% testing accuracy on their baseline CNN and 89% on their best ResNet model. Our project differs in that we want to build our own model with unique architecture. The author of the Stanford paper seemed very experienced in using CNNs and we are not. We felt that by trying to create our own architecture we can come away with a better understanding of how CNNs work and if we just used a pre-existing one, such as ResNet, then we will not have the opportunity to learn the models as well as we could by making our own. This sets us with the goal of creating a model architecture that can accurately classify artists.

3 Data

We gathered 1239 images of artwork for Picasso, 785 for Dali, 884 for Van Gogh, 827 for Monet. From each artist, we created a .pk file that had all of their artwork in it normalized to a 256 x 256 image to feed to our network. We specifically went with 256 x 256 because that was the highest resolution our computers could handle when running the network.

On any given run through our network would use 500 of these images from each artist to learn from, leaving a total of 2000 pictures for the model to use. The reason we used 500 as opposed

Figure 1: A Sample of the data, one from each artist

to all of them was because any more gave us memory errors. From there we split the data into three categories: Train (consisting of 1750 images), Validation (consisting of 150 images), and Test (consisting of 100 images). The train set was what the model used to actually learn from, this is why it was by far the biggest dataset. Validation was used during runtime of the training process. At the end of each epoch, the model predicts based on the validation set, but does not learn, and predicts an accuracy. This is helpful in hyperparameter fine tuning because this is a good indicator of if the model is overfitting or not. Finally, Test is a dataset kept completely separate from the training process and is used to show the overall accuracy of our model. Because the model has never seen any of these images before, this makes it a good metric for how well it learned.

4 Convolutional Nueral Networks

4.1 How They Work

For our project, we found that a Convolutional neural network would be the best way to input images and classify them based on different features in the image. How it works is fairly simple, the network has a filter that it drags across the image. This filter is usually square and fairly small, between 2 and 7 pixels in length, and as it passes over each subsection of the picture, it reads in the pixel values for every point it sees and calculates a number based on the product of pixels and weight values to represent that grid in an output image. Thus shrinking the picture and returning a new image that is smaller and more importantly it can contain information that a filter was looking for. For example if you had a picture of a face and the filter for a nose, then the filter would scan the image in every possible grid that fits the bounds, and at each point it would calculate how big the chance is that a nose is in that grid. Then with proper training, the output would contain very small values except for the spot corresponding to where the nose was, which would contain a relatively high value. Now this would just be for one feature, if we increase the dimensions of the filter and train them all with different weight values then at this one layer we can output multiple dimensions with each one highlighting different features. Going back to the face example, if we added a dimension for eyes and another one for mouth, we would be able to take the initial image (assuming it has a single dimension for this case) and it would output a new set that was slightly smaller in height and width of the initial but would be three dimensions deep with nose, eyes, and mouth being what is showing in their respective layers and locations.

We did not stop there, because there are a lot of extra points and noise that get generated from this filter, we can use activation function and max pooling layers to normalize the output and reduce noise. Max pooling is a fairly simple concept, by sliding a small filter (usually 2 by 2) across a set of features it can take the max of them and output will only be the highest for that square so it can reduce the size of the image and still keep the important values. This keeps the image intact while also reducing the computational load of the network. Next we have activations. Because the outputs can vary in scale we can run an activated image on a set of values to normalize them and make them more usable. An easy example of this would be Relu which returns the max of the input or 0. So by running this it would take away all the negative values in the feature set leaving the positive ones untouched. Another example would be a hyperbolic function such as tanh. These map the values to a range of [-1, 1] with the high positive ones being highly positive and the lower ones being highly negative. This scale makes the numbers easier to work with and can help prevent overflow. By

activating a feature after max pooling the higher values will be weighted more than the lower ones for the next layer or classification.

Now we need to cover how the model learns, this is done through a process called back propagation. After the model has reached the end of the training for a batch, it runs a classifier method to calculate the probability of each class. Then the index with the highest value corresponds to the class that the model thinks the object belongs to. One popular method for classification is softmax. From there the model will check the correct answer and see the probability it had for the correct class. By calculating the loss you can apply the principle of scholastic gradient descent to update the weights for each filter going backwards through the model and adjusting them slightly. After many run throughs and slight adjustments, all of the weight values will be distant and based on those the model will have been trained to correctly classify its input.

4.2 Our Model

Our model was a basic convolutional neural network. We have a collection of pictures for each of our artists, and at the beginning of each run we selected 500 of these to be used. We experimented using larger amounts of pictures but due to the limitations of our computers 500 of each was near the maximum that we could use for our network. We also considered using images of different sizes. We landed on 256x256 pixels as some of our images were too small to allow for this. So we were limited by the data that we had. Using larger images would likely give a better result, but this would greatly increase the computational requirements of the program. For this new 2000 image dataset 1750 of the pictures are used as training data, 150 as evaluation data, and 100 as testing data. Now that we have our data we move on to the actual network part of our program. Our group did not have very much experience using either neural networks or convolutional neural networks so a lot of these upcoming parameters were found using trial and error of what would give the best results. But because of our lack of experience, we decided to use tensorflow as a tool to create our model.

```
In [5]: model.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 255, 255, 15)      195
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 15)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 126, 126, 30)      1830
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 30)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 62, 62, 45)        5445
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 31, 31, 45)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 30, 30, 60)        10860
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 15, 15, 60)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 75)        18075
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 75)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 90)          27090
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 90)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2, 2, 105)         37905
_________________________________________________________________
flatten (Flatten)            (None, 420)               0
_________________________________________________________________
dense (Dense)                (None, 64)                26944
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 260
=================================================================
Total params: 128,604
Trainable params: 128,604
Non-trainable params: 0
```

Figure 2: This is the printout of our model. As you can see every Conv2d layer the dimentions of the image expand by a multiple of 15 until the final product is flattend and run through the dense layer

Tensorflow provided an efficient implementation of the methods we needed while still giving us control of how the neural network would be made. Ultimately our network had 7 convolutional layers and 6 max pooling layers. Each of our convolutional layers used a 2x2 filter, and increased the dimensions it outputs by 15 for every layer. We also used the tanh activation function for all of our layers along with bias. We trained our data using 8 epochs, as we found that any training beyond this point would overtrain the model for our dataset and give us less than optimal results. Each of the 8 epochs trained the data in 55 batches of images. With all of these inputs our model ended up having 128,604 total trainable parameters.

5 Experiment

5.1 Baselines

For our project we used multiple baselines from a variety of different sources. The first of which was random selection, the easiest to determine. Because we were using four different artists, the chance of our model being correct was 25% so if we were able to train one that could get higher then we would have done the bare minimum.

The next baseline we used as an intermediate was a three layer fully connected network. This network was trained with 50 epochs and batches of size 50. The layers were made up with two hidden, both of size 16, and were activated with a relu activation and at the end were classified with a softmax activation. The highest accuracy we got with this model was 48% but we had made some adjustments afterwards. At this stage in the project we were only using three different artists, so the baseline for this model being completely random was 33%. The model was also trained on images that were of size 32 x 32 (we later found more success with training our model on images with higher resolution but that was after we found this baseline). Regardless, there was a 15% increase in accuracy from pure random, so we felt like the 48% accuracy this model gave made this baseline usable.

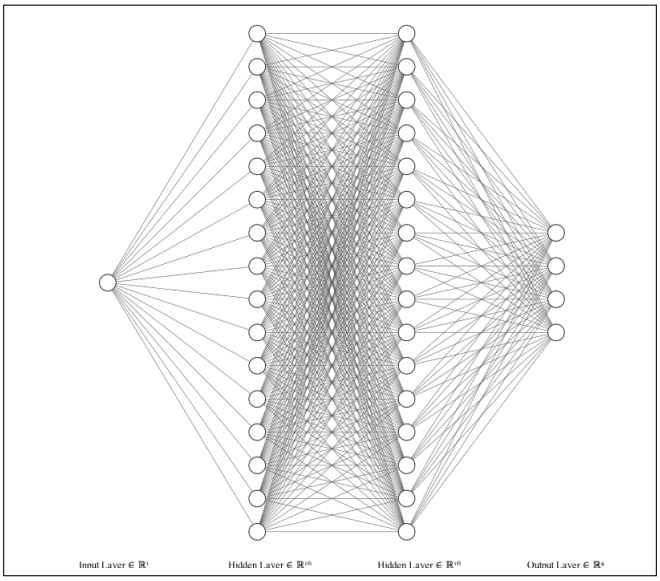

Figure 3: A Diagram of the 3 layer fully connected network

Our final baseline was based on general human knowledge, this one also became the goal we wanted to surpass. We created a google form quiz where people were asked to guess the correct artist based off of a painting. We felt that because these artists were so famous and well known that a quiz would be fairly accurate with classifying the paintings. This turned out to be correct with the average accuracy of the participants being 72%. While this is not extremely high, considering that these are average people who took the quiz, we felt like if our model was better than the average person then we would have succeeded in creating an AI that could classify paintings.

5.2 Hyperparameter Fine Tuning

As stated before we had minimal knowledge of neural networks so our tuning of parameters was a lot of trial and error to find what worked best. We tested our program using tanh, relu, and sigmoid activation functions. Out of these we found that tanh was giving us the best results. We decided to test these three parameters as we found that these were some of the more simple activation functions, and without a greater knowledge of neural networks, using a complex activation function would not help us get the best result. On deciding the amount of epochs to use we tried a multitude of numbers. We determined 8 would give us the best model accuracy, as when we ran the program with greater than 8 epochs, we found that the accuracy and validation accuracy would begin to diverge around the 9th epoch. This meant that we have overtrained for our data. Lastly, we needed to train our layers. This was the most time consuming part of parameter tuning. We attempted using different methods for this, first using fewer layers but having each layer view larger sections of the images and tested how many dimensions would be ideal. We found that having as many layers as we could gave us the most optimal outcome and thus for each layer, our filter only viewed a 2 x 2 area. For the dimensions of the image, we found that the one we used, increasing by 15 every layer, gave us the best outcome of those that we tried. Using fewer dimensions seemed to limit the network's ability to view everything it wanted, while using a greater number of dimensions was found to be unnecessary as many of the dimensions were not picking up any data. With a more theoretical knowledge of neural networks we likely would have been able to tune these parameters quicker and to a higher quality, but we found what we believe to be a good combination of these parameters for our data and our model.

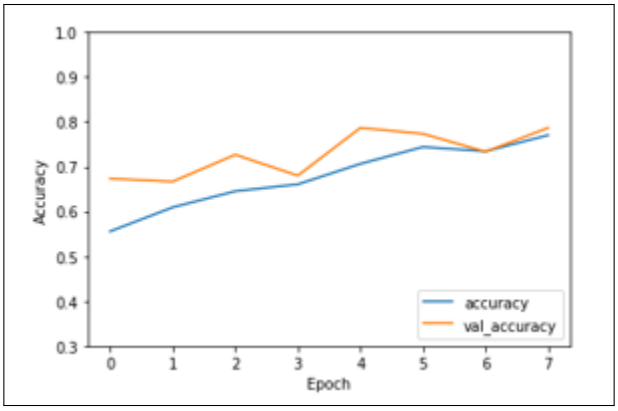

Figure 4: Here shows a graph of our accuracy over time versus our validation accuracy over time. As you can see they are closely corrolated so our model never overfit while still gradually improving

6 Results

After we finished tuning the parameters of our model we ended up with an accuracy of 81% which was greater than both our final baseline and the base CNN from the related work. Our data set makes it difficult to get a higher accuracy. Artists' styles changed over their careers so the body of work for each individual artist varied quite a bit, making it difficult for a network to learn the intricacies of each artist. The most artwork we had for an artist was Picasso with 1239 pieces. Ideally with trying to implement image recognition in our network we would have thousands of images at minimum to train on for each artist; however, with deceased artists there is a limited supply of artwork and there won't ever be more. Our computers were only powerful enough to run the network with 256 x 256 images which makes it inevitable that some features of the art won't be picked up since the resolution is so low. In short, if we were able to run the network with higher resolution images from a larger, more uniform data set we would likely see more accurate predictions.

6.1 Feature Maps

One example we have for showing how our model has learned is from feature maps. These are the resulting images that come from expanding the dimension in the convolutional layer described in section 3.1. We got the output from our first layer which expanded the image by 15 and gave a black and white feature map to show what areas got activated. In this example Figure 5 is the original image before the model and Figure 6 is the image after the first layer. Everywhere there is white represents the area where the neuron got activated. For example, the far left in the middle has activated only the lake, this means that the model found something important here and felt the need to record only lakes. We can see in the middle in the second to last column that the ground has all been activated which means that this neuron has been trained to see the ground. Moving onto the balck squares, we used a diverse set of artworks (not from one genre), not all of them are landscape painting. These black squares could be looking for features found in still lifes, sculptures, or portraits and because the original image was none of these, these neurons did not fire.

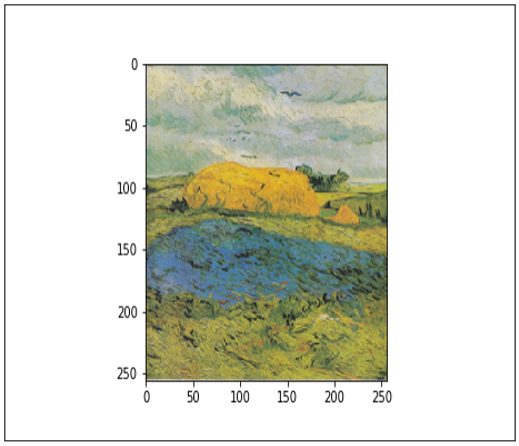

Figure 5: The Origional Image before going through the model

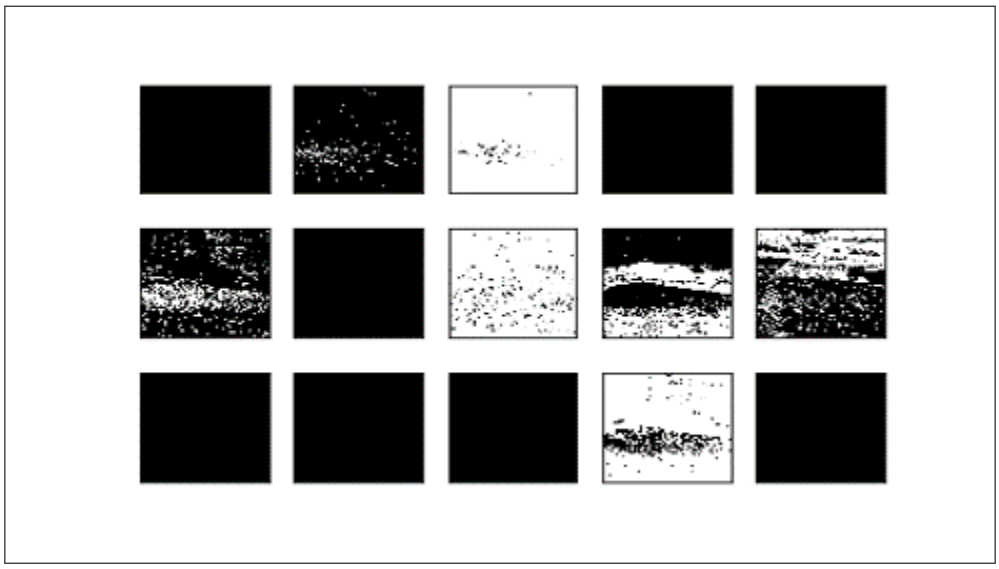

Figure 6: The feature map generated after going through the first convolutional layer

7 Conclusion

Overall we believe that our network was successful in what we set out to do. The model we created had an accuracy that was 9% better than the average human score we found, and given the limitations of our data and dataset, we feel good about the model we made. We successfully implemented a Convolutional Neural Network that was able to recognize painters based on the features in their paintings. In addition, by gathering the data ourselves and not using a prebuilt dataset, we got a better understanding of how neural networks take in data and how they learn from what they have taken in. If we were to do this project again we could make our dataset styles instead of individual painters so that we could have as many training points as possible, and we would be able to make a very accurate network thanks to what we learned by making this network.

As stated previously our model was a relatively basic implementation of a CNN. As this was our first time using any kind of neural network we focused a lot on trying to learn neural networks, whereas as you can see in the Stanford paper [1], they were going for the best model accuracy they could find. The Stanford model, which attempted to address this problem, used a more complex CNN than we did in our implementation (one which helped them achieve a better model accuracy). It's also worth noting that our baseline CNN performed to a higher degree than their baseline CNN along with their base RESNET-18. It was only when they added "Transfer Learning" to their RESNET-18 model that they were able to achieve a higher level of accuracy. Thus we feel as though we achieved a high level of success considering the complexity of our model and our relative inexperience in dealing with these types of models.

References

[1] Nitin Viswanathan, Stanford University, accessed via http://cs231n.stanford.edu/reports/2017/pdfs/406.pdf

[2] Claude Monet Paintings, accessed via https://www.wikidata.org/wiki/Wikidata:WikiProject_sum_of_all_paintings/Catalog/Catalogue_raiso

[3] Van Gogh Paintings, accessed via https://www.wikidata.org/wiki/Wikidata:WikiProject_sum_of_all_paintings/Creator/Vincent_van_Gogh

[4] Salvador Dali Paintings, accessed via https://www.wikiart.org/en/salvador-dali/all-works#!%23filterName:Style_surrealism,resultType:detailed

[5] Pablo Picasso Paintings, accessed via https://www.wikiart.org/en/pablo-picasso/all-works#!#filterName:all-paintings-chronologically,resultType:masonry

[6] End to End Machine Learning, accessed via https://e2eml.school/how_convolutional_neural_networks_work.html

[7] Simran Bansari, Data Driven Investor, accessed via https://medium.com/datadriveninvestor/introduction-to-how-cnns-work-77e0e4cde99b

[8] Tensorflow Tutorial, accessed via https://www.tensorflow.org/tutorials/images/cnn