Melissa Rutherfoord

MSDS458

Assignment 4

Final Project CNN:

Detecting Van Gogh's Painting Style

After running many models using 200x200 pixel tiles, and averaging about 60% accuracy with my validation set, I decided to move up to a more compositional level of the paintings by using 400x400 pixel tiles. I was a little concerned that I might not get a large enough training set if I overlapped the tiles by only fifty percent, so I altered my code to overlap by twenty-five percent instead. With these parameters, I was able to create a slightly larger dataset. This may not only be because of the increased overlapping, but also because the images are a larger proportion relative to the full-sized images. With a larger size tile, it could mean that the tiles have a better chance of containing more non-redundant information, and therefore that their entropy scores are more likely to be high enough to be included in the set. For this project, I ended up with a training set containing 2837 images, a validation set of 709 images, and a test set containing 395 images. Here is a sample of what the dataset looks like:

For this project, I ran four models: two with a five-layer architecture, and two with a four-layer architecture. Based on my previous experiments, I decided to continue to use a relu activation function for all layers except the output, and a single node with a sigmoid function for my output. For my convolutional layers, I used padding to make sure all parts of the image were included in the filter. In all of the architectures, I used two dense layers. For every network I trained, I experimented with the number of filters and nodes. Based on the previous assignment, I felt that using 32 filters for the convolutional layers would probably give me the best results, but I still wanted to try training a network using 16 filters on at least one of the models. I also experimented with the number of nodes in the dense layers, using 64, 32, and 16 nodes.

Model 1:

Model 1 is a five-layer CNN. The architecture and results are shown below.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 400, 400, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 200, 200, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 200, 200, 32)      9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 100, 100, 32)      9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 50, 50, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 80000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2560032
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 2,579,457
Trainable params: 2,579,457
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/4
2837/2837 [==============================] - 2961s 1s/step - loss: 0.3502 - acc: 0.8418 - val_loss: 0.9008 - val_acc: 0.6763
Epoch 2/4
2837/2837 [==============================] - 2854s 1s/step - loss: 0.1076 - acc: 0.9585 - val_loss: 1.7398 - val_acc: 0.6601
Epoch 3/4
2837/2837 [==============================] - 2782s 981ms/step - loss: 0.0452 - acc: 0.9847 - val_loss: 1.9247 - val_acc: 0.623
9
Epoch 4/4
2837/2837 [==============================] - 2789s 983ms/step - loss: 0.0357 - acc: 0.9880 - val_loss: 2.0890 - val_acc: 0.675
1
```

Already I can see that using the 400x400 tiles results in much better generalization.

Model 2:

Model 2 is also a five-layer CNN, but I wanted to see how far I could push reducing the complexity of the model. Here I used 16 filters, and only 16 nodes in the dense layer.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 400, 400, 32)      896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 200, 200, 32)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 200, 200, 16)      4624
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 100, 100, 16)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 100, 100, 16)      2320
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 50, 50, 16)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 40000)             0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                640016
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 647,873
Trainable params: 647,873
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/6
2837/2837 [==============================] - 2182s 769ms/step - loss: 0.6890 - acc: 0.5477 - val_loss: 0.6884 - val_acc: 0.548
4
Epoch 2/6
2837/2837 [==============================] - 2198s 775ms/step - loss: 0.6884 - acc: 0.5487 - val_loss: 0.6884 - val_acc: 0.548
8
Epoch 3/6
2837/2837 [==============================] - 2132s 751ms/step - loss: 0.6886 - acc: 0.5481 - val_loss: 0.6885 - val_acc: 0.548
1
Epoch 4/6
2837/2837 [==============================] - 2133s 752ms/step - loss: 0.6885 - acc: 0.5484 - val_loss: 0.6883 - val_acc: 0.549
4
Epoch 5/6
2837/2837 [==============================] - 2131s 751ms/step - loss: 0.6885 - acc: 0.5483 - val_loss: 0.6884 - val_acc: 0.548
5
Epoch 6/6
2837/2837 [==============================] - 2121s 748ms/step - loss: 0.6885 - acc: 0.5483 - val_loss: 0.6884 - val_acc: 0.548
6
```

This model did not train well. This tells me that the network must have bottlenecked, resulting in an underfitting. Sixteen filters and nodes are too few for the network to train with.

Model 3:

Model 3 is a four-layer network with two convolutional layers with 32 filters, and a dense layer with 64 nodes. This model was my most successful model, with the accuracy pushed to over 70%.

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 400, 400, 32)	896
max_pooling2d_7 (MaxPooling2	(None, 200, 200, 32)	0
conv2d_8 (Conv2D)	(None, 200, 200, 32)	9248
max_pooling2d_8 (MaxPooling2	(None, 100, 100, 32)	0
flatten_3 (Flatten)	(None, 320000)	0
dense_5 (Dense)	(None, 64)	20480064
dropout_3 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 1)	65

Total params: 20,490,273
Trainable params: 20,490,273
Non-trainable params: 0

```
Epoch 1/5
2837/2837 [==============================] - 3042s 1s/step - loss: 0.3131 - acc: 0.9108 - val_loss: 1.2102 - val_acc: 0.7145
Epoch 2/5
2837/2837 [==============================] - 3081s 1s/step - loss: 0.0248 - acc: 0.9925 - val_loss: 1.3187 - val_acc: 0.7239
Epoch 3/5
2837/2837 [==============================] - 2933s 1s/step - loss: 0.0199 - acc: 0.9950 - val_loss: 1.7173 - val_acc: 0.7443
Epoch 4/5
2837/2837 [==============================] - 3020s 1s/step - loss: 0.0079 - acc: 0.9980 - val_loss: 2.4946 - val_acc: 0.6917
Epoch 5/5
2837/2837 [==============================] - 2984s 1s/step - loss: 0.0131 - acc: 0.9965 - val_loss: 1.5015 - val_acc: 0.7313
```

Model 4:

Model 4 is a four-layer network that is identical to Model 3, except the dense layer has only 32 nodes instead of 64.

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 400, 400, 32)	896
max_pooling2d_9 (MaxPooling2	(None, 200, 200, 32)	0
conv2d_10 (Conv2D)	(None, 200, 200, 32)	9248
max_pooling2d_10 (MaxPooling	(None, 100, 100, 32)	0
flatten_4 (Flatten)	(None, 320000)	0
dense_7 (Dense)	(None, 32)	10240032
dropout_4 (Dropout)	(None, 32)	0
dense_8 (Dense)	(None, 1)	33

Total params: 10,250,209
Trainable params: 10,250,209
Non-trainable params: 0

```
Epoch 1/4
2837/2837 [==============================] - 2710s 955ms/step - loss: 1.2451 - acc: 0.5799 - val_loss: 0.7168 - val_acc: 0.555
1
Epoch 2/4
2837/2837 [==============================] - 2707s 954ms/step - loss: 0.1738 - acc: 0.9185 - val_loss: 2.9051 - val_acc: 0.631
2
Epoch 3/4
2837/2837 [==============================] - 2753s 970ms/step - loss: 0.0585 - acc: 0.9805 - val_loss: 2.8113 - val_acc: 0.644
9
Epoch 4/4
2837/2837 [==============================] - 2927s 1s/step - loss: 0.0323 - acc: 0.9900 - val_loss: 3.4513 - val_acc: 0.6353
```

This experiment tells me that the number of nodes in the dense layer is very important for accuracy for this project.

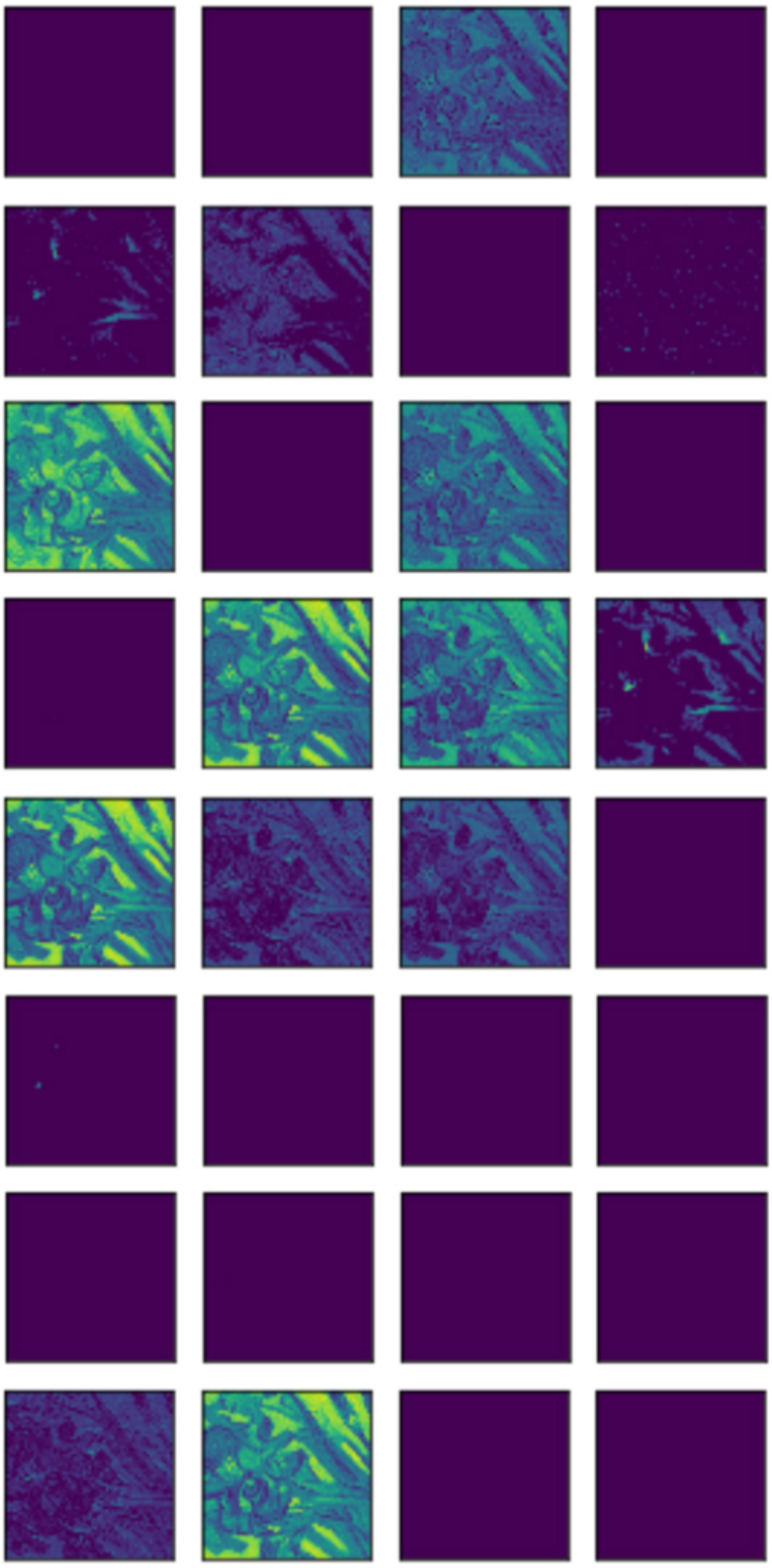
Model 3 Evaluation:

Since Model 3 performed the best, I used my test set to see how well the model generalized. The model worked surprisingly well, returning an 80% accuracy score.

```
In [24]: model400_3.evaluate_generator(test_set, steps=25)
Out[24]: [1.2833971189169944, 0.7974683542794819]
```

I also wanted to visualize the two convolutional layers for this model. I used the following image from the test set:

The first convolutional layer was difficult to make out what was happening with all 32 filters because the feature maps were quite dark, but here is a visualization from the second layer:



With so many more feature maps, it looks like many of them are not in use for this image, but I can make out some that are activating for textures, the background, etc. With thirty-two maps on two layers, it is harder for me to interpret them, unlike the three layers of sixteen maps I used for Assignment 3.

Conclusion

I am happy to see that my results improved greatly by using 400x400 sized tiles.. Not only was I able to obtain a larger dataset (which I'm sure helped with my training), it seems like the network can learn to recognize the difference between a painting composed by Van Gogh and similar artists. I am definitely interested in taking this project further by obtaining a better dataset (perhaps using landscapes or portraiture) in order to continue to investigate whether it is the composition that defines an artist, or if the hand of the artist is unique enough for a machine to discern the difference. This project has been fun, interesting, and an eye-opening experience which I hope to continue to expand upon in the future.