Final Results:

I have significantly changed my model from my first deliverable. I switched from using PCA to using a CNN because I wanted to limit the distortion of my data. For this reason rather than reshape my images, I decided to crop them.

I played around with different pre-built CNN architectures from Keras, but settled on using the VGG19 architecture as my based. Below is the structure of my model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 400, 400, 3)]     0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 400, 400, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 400, 400, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 200, 200, 64)      0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 200, 200, 128)     73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 200, 200, 128)     147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 100, 100, 128)     0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 100, 100, 256)     295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 100, 100, 256)     590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 100, 100, 256)     590080    
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 100, 100, 256)     590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 50, 50, 256)       0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 50, 50, 512)       1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 50, 50, 512)       2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 50, 50, 512)       2359808   
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 50, 50, 512)       2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 25, 25, 512)       0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 25, 25, 512)       2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 25, 25, 512)       2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 25, 25, 512)       2359808   
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 25, 25, 512)       2359808   
_________________________________________________________________
```

```
block5_pool (MaxPooling2D)   (None, 12, 12, 512)       0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_18 (Dense)             (None, 1024)              525312
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                10250
=================================================================
Total params: 20,559,946
Trainable params: 20,559,946
Non-trainable params: 0
```

After the VGG19 layers, I added an average pooling layer and a dropout layer, which drops 20% of the input. I also tried different optimizers for this architecture. I ran each one for one epochs to get a sense of which one was best.

Validation accuracy for optimizers after 1 epoch of training

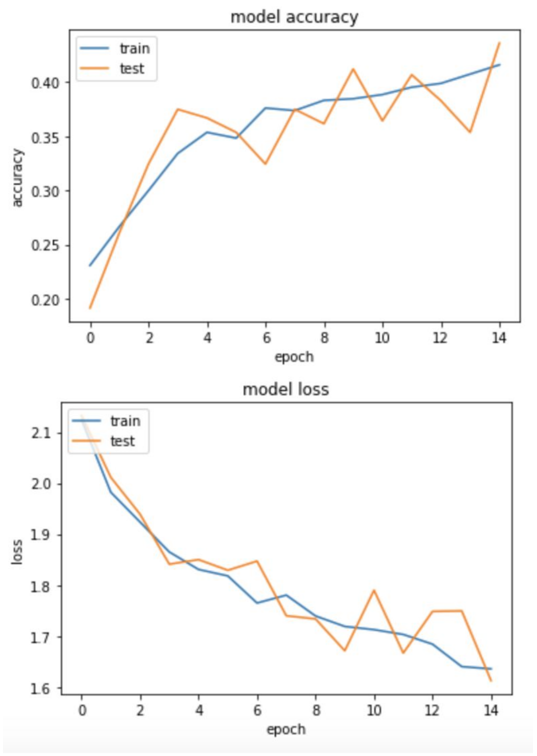
SGD:	0.2660
Adam:	0.2048
RMSprop:	0.1543
Adagrad:	0.2048
Adadelta	0.2048

Based on the results, I decided to use the SGD optimizer.

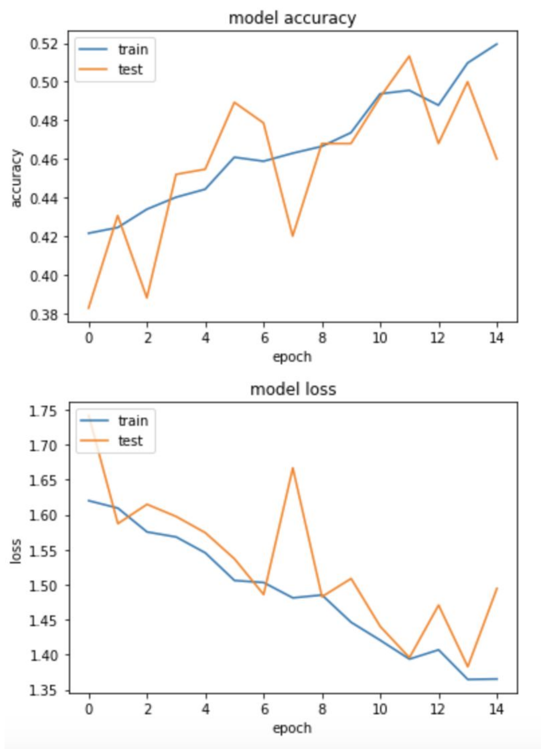
During testing I noticed that my model was predicting much more accurately for some classes than others. This was because I had very imbalanced data, 800 images for some artists and only 250 for others. To address this issue I added class weights.

I ran my model through 4 sets of 15 epochs and the following images are the training history.

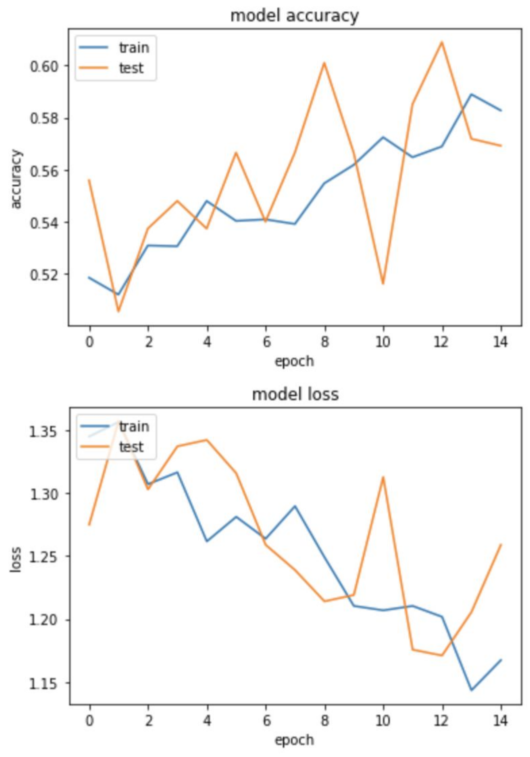
Epochs 1-15



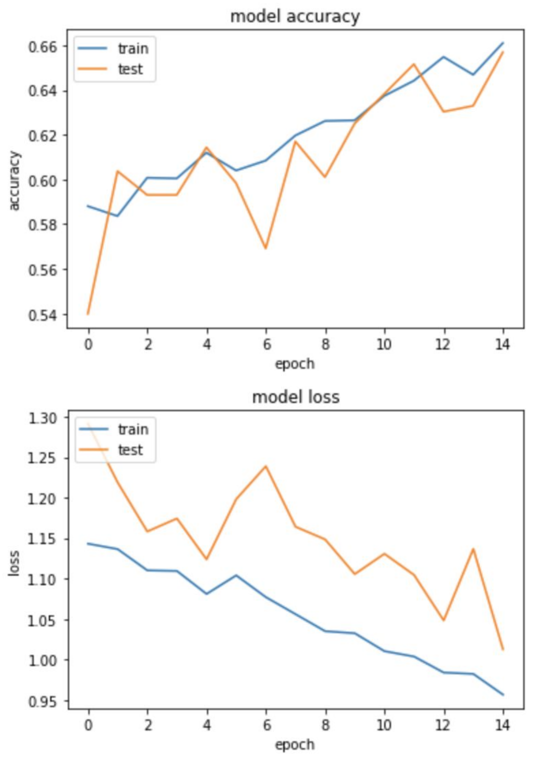
Epochs 16-30



Epochs 31-45



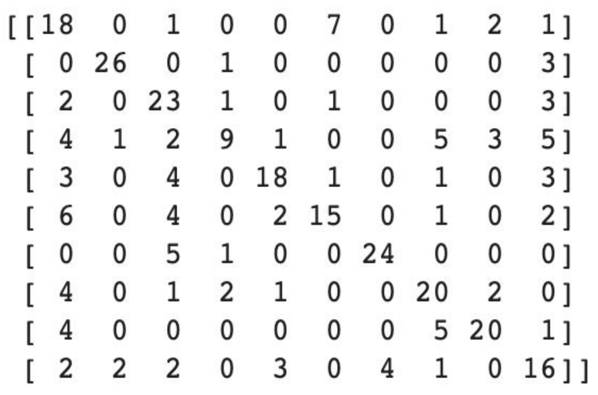
Epochs 46-60



The model predicted the test set with a 63% accuracy.

This is the confusion matrix (x axis is predictions, y axis is actual class)

Classes:



```
Degas       [[18  0  1  0  0  7  0  1  2  1]
Durer        [ 0 26  0  1  0  0  0  0  0  3]
Gaugin       [ 2  0 23  1  0  1  0  0  0  3]
Picasso      [ 4  1  2  9  1  0  0  5  3  5]
Renoir       [ 3  0  4  0 18  1  0  1  0  3]
Sisely       [ 6  0  4  0  2 15  0  1  0  2]
Titian       [ 0  0  5  1  0  0 24  0  0  0]
Rembrandt    [ 4  0  1  2  1  0  0 20  2  0]
Van Gogh     [ 4  0  0  0  0  0  0  5 20  1]
             [ 2  2  2  0  3  0  4  1  0 16]]
```

<u>Poster Presentation:</u>

Problem:

My goal was to create an artificial intelligence that could quantify artistic style and categorize 2D artwork by artist.

Hypothesis:

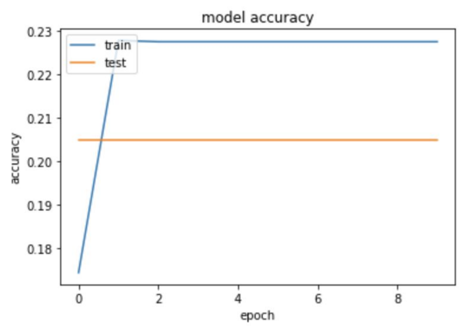
I predicted that I would be able to create a CNN that could classify paintings by 10 artists with a 70% accuracy. Although I did not quite reach my goal of 70%, I believe my results are pretty significant considering the fact that the task is difficult even for humans.

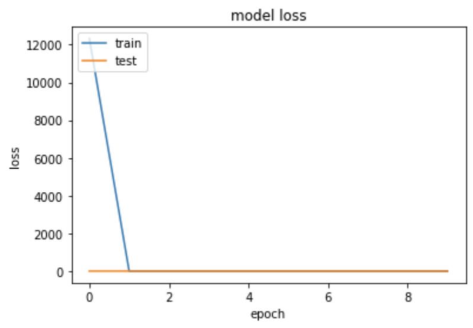
Keys in Implementation:

Two things that significantly improved my models performance was adding a dropout layer and using class weights. By far, I had the most data for Van Gogh and as you can see from the confusion matrix, the model is not predicting Van Gogh paintings much more accurately than other paintings.

Comparison to Baseline:

A simple neural network, with only once dense layers only predicts with a 10% accuracy, which is to be expected since I have 10 classes. Below is the training history for the baseline model. Note that the loss does not go down to 0. It only reaches about 2.2. The line looks flat, but it is not. It is just because the initial loss was very high, so the y axis is very long.





Real World Application:

Perhaps a model like this or a variation of it that classifies paintings by artistic styles or time periods could be used by art historians to study paintings from unknown artists/origins. It could also be used to identify similarities in different artistic styles which might help historians better understand how artistic styles changed overtime and spread throughout the world.