The Art of Neural Nets

Marco Tavora

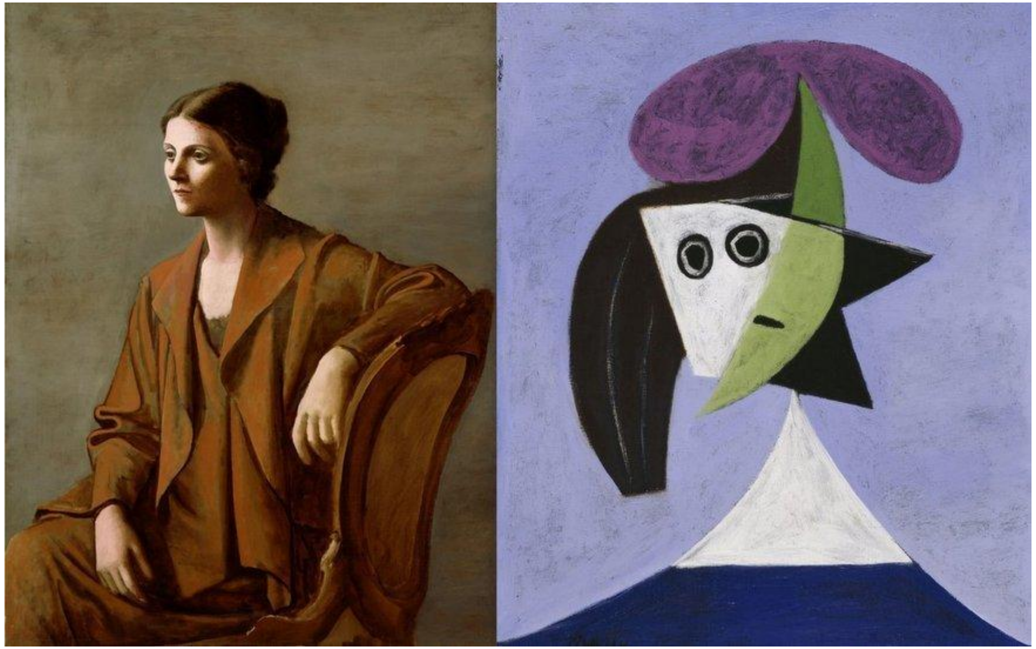
marcotav65@gmail.com

Preamble

The challenge of recognizing artists given their paintings has been, for a long time, far beyond the capability of algorithms. Recent advances in deep learning, specifically the development of convolutional neural networks, have made that task possible.

Challenge

Given a painting can machines identify the painter?

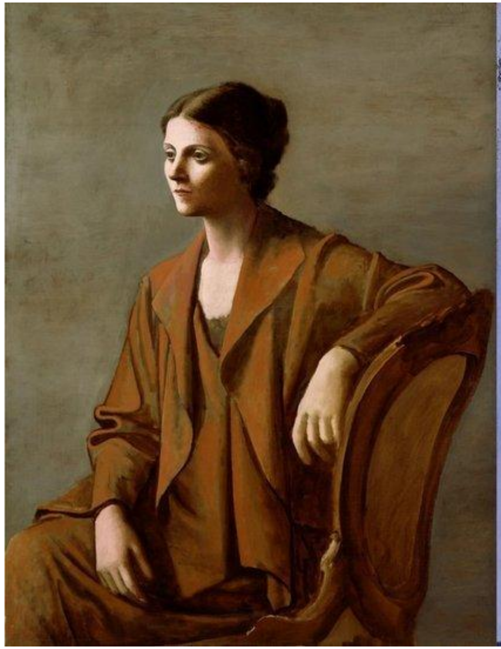


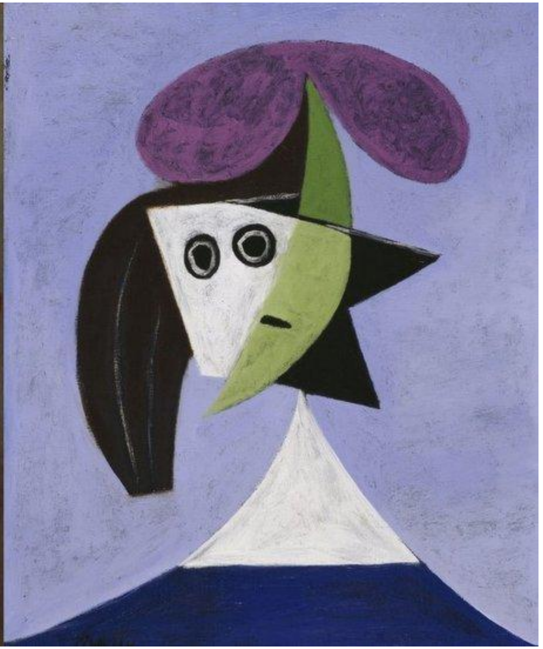
Can you identify these painter or maybe their period/style?

🤔

Challenge

Given a painting can machines identify the painter?


Woman in Hat [Olga] by Pablo Picasso, 1923


Portrait of Olga Picasso, by Pablo Picasso, 1935

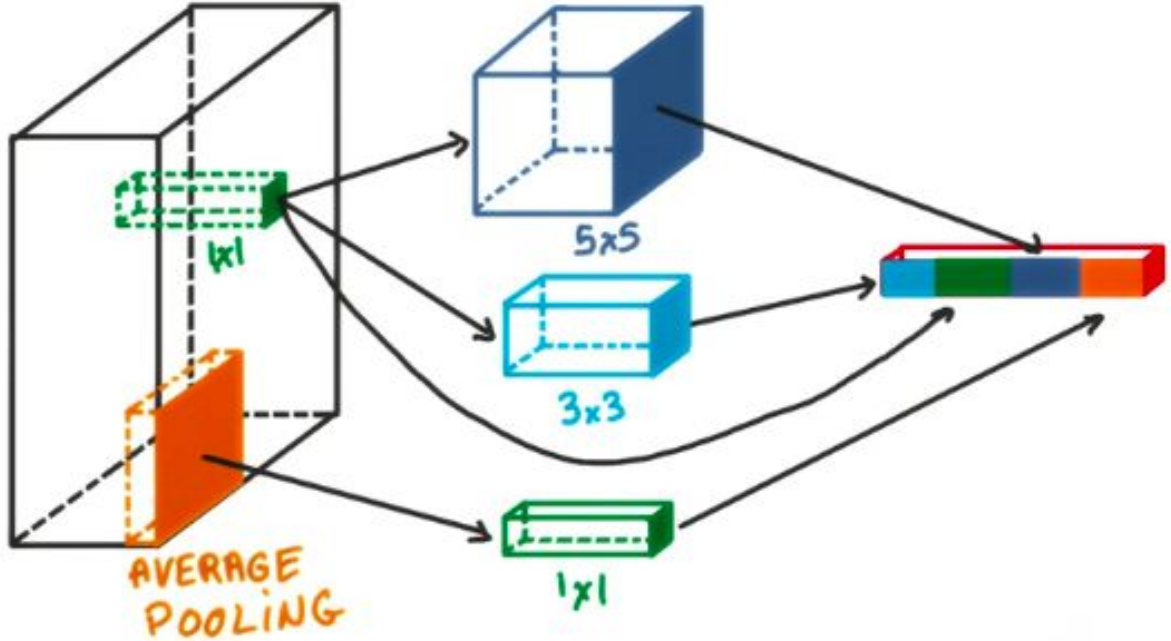
Both are from Picasso (1923, 1935)

Challenge

Complex but useful problem:

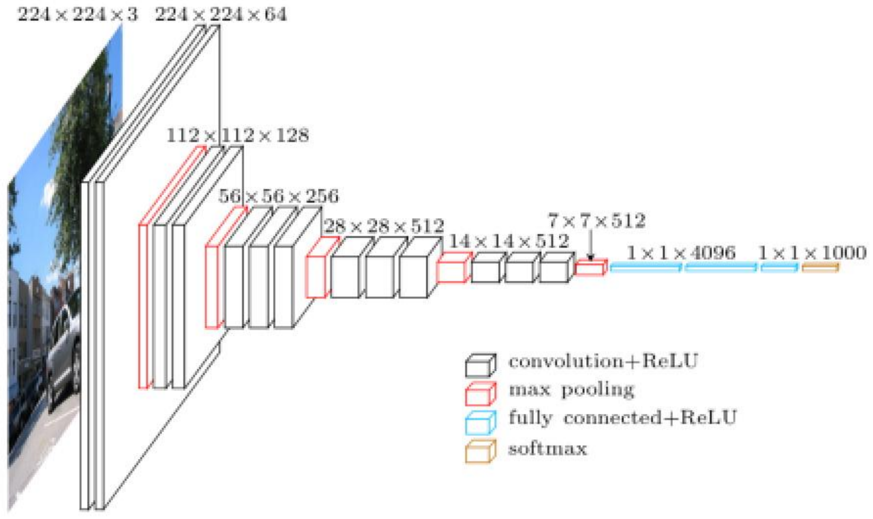
- Painters change styles along their career
- It is not just object or face recognition (painters can paint many different things)
- Methods employed by specialists can be invasive and cause damage do the painting
- High performance classifiers can also help identify forgeries

Deep learning to the rescue!



Deep learning

- DL methods of image processing can help.
- Most previous work on artist identification start by choosing the painters' characteristics as features
- Since NN do not need "feature hand-crafting" they are ideal for this task

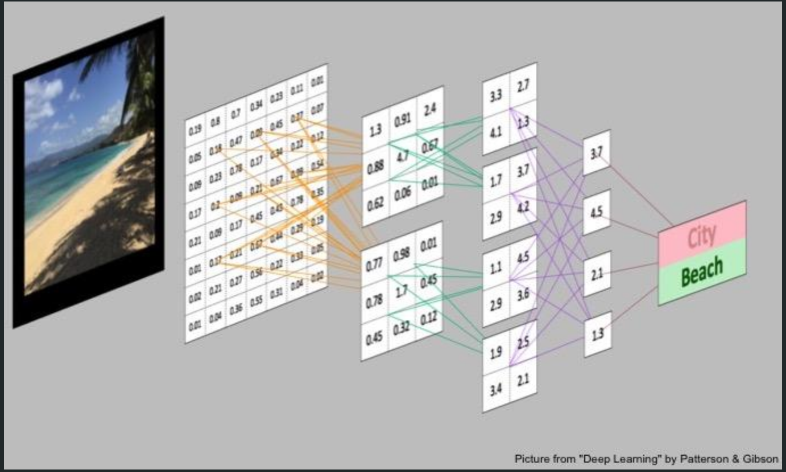


VGGNet Architecture

Methodology: Convolutional Neural Networks (CNNs)

Particularly proficient at image classification

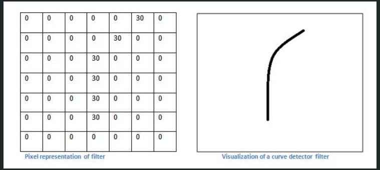
They make assumption that inputs are images and that vastly reduces number of parameters



Methodology: Convolutional Neural Networks (CNNs)

Layers are feature identifiers:

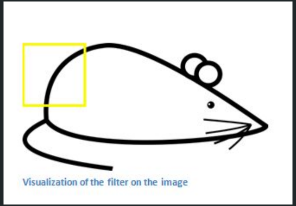
Convolutional layers filter input and they identify edges, curves, …



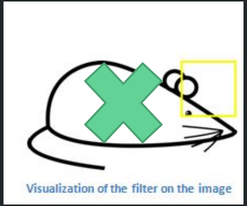
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Visualization of a curve detector filter

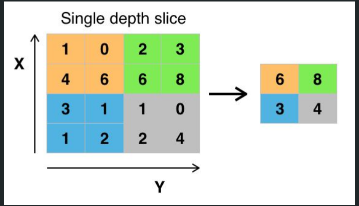


Visualization of the filter on the image



Visualization of the filter on the image

Pooling layers shrink input dimension by an integer factor

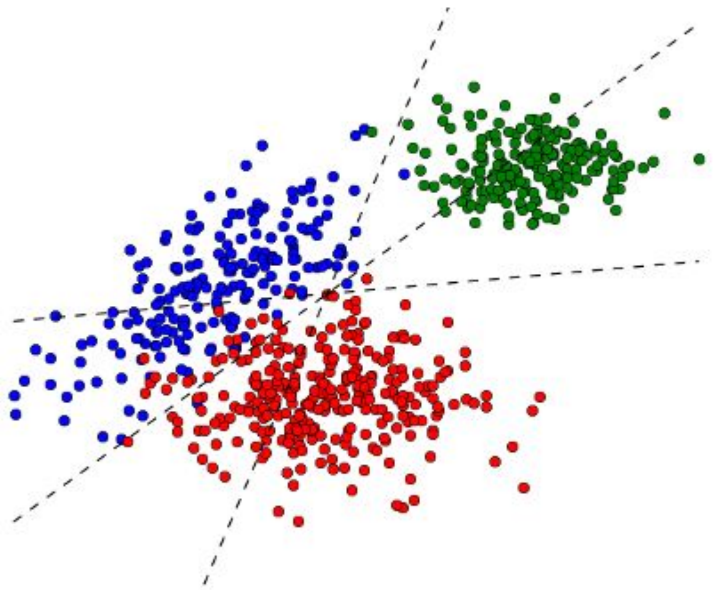


1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

6	8
3	4

Procedure

- Multi-class classification problem
- Painters must have a minimum number of paintings in the training set to ensure sufficient sample size
- Running time increases when the number of classes increases, so I ended up choosing 3 painters (contrast with 1 painter = binary classification)



Procedure

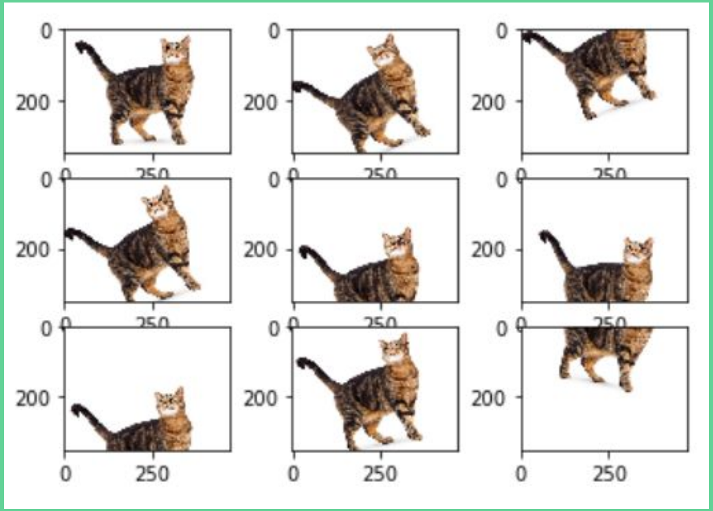
- We have a small sample (for image recognition standards)
- Tactics to attack this problem of lack of data:
  - Image augmentation
  - **Transfer learning**

Image Augmentation

- Combats high expense of getting new data by creating new data from existing data

- Keras very easily manipulates images to get many altered versions of the same image

- Modifications include rotations, shifts, shears, flips, whitening and others



What is Transfer Learning?

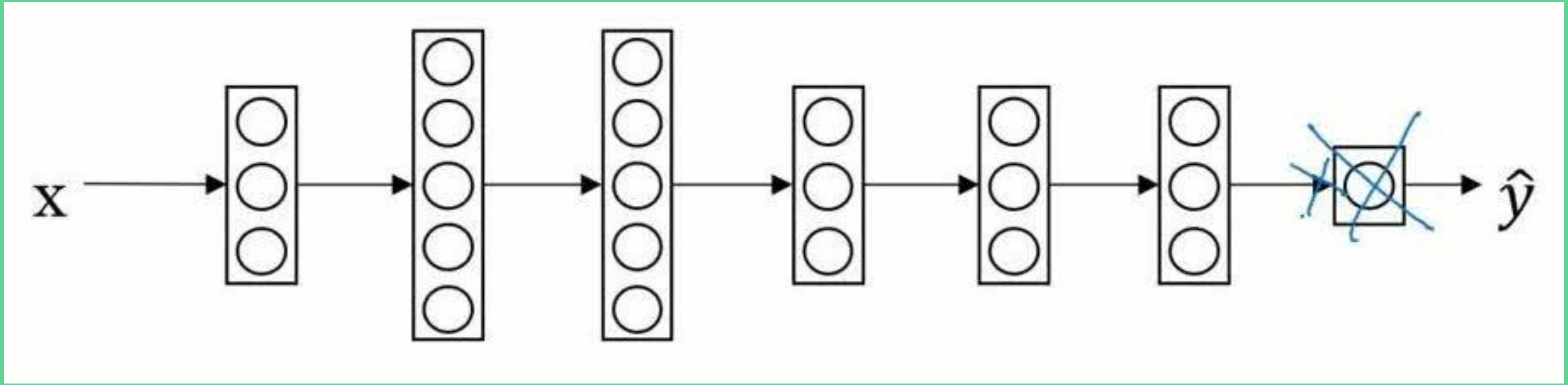
- When knowledge gained during training in one type of problem is used to train in other type of problem (e.g. knowledge gained learning to recognize cars applies when trying to recognize trucks)

- TL and CNNs: nowadays, training CNNs from scratch is quite rare because of sample size limitations

- Pre-train a CNN on a large dataset (e.g. ImageNet with 1.2M images, 1K categories) and apply it to our task

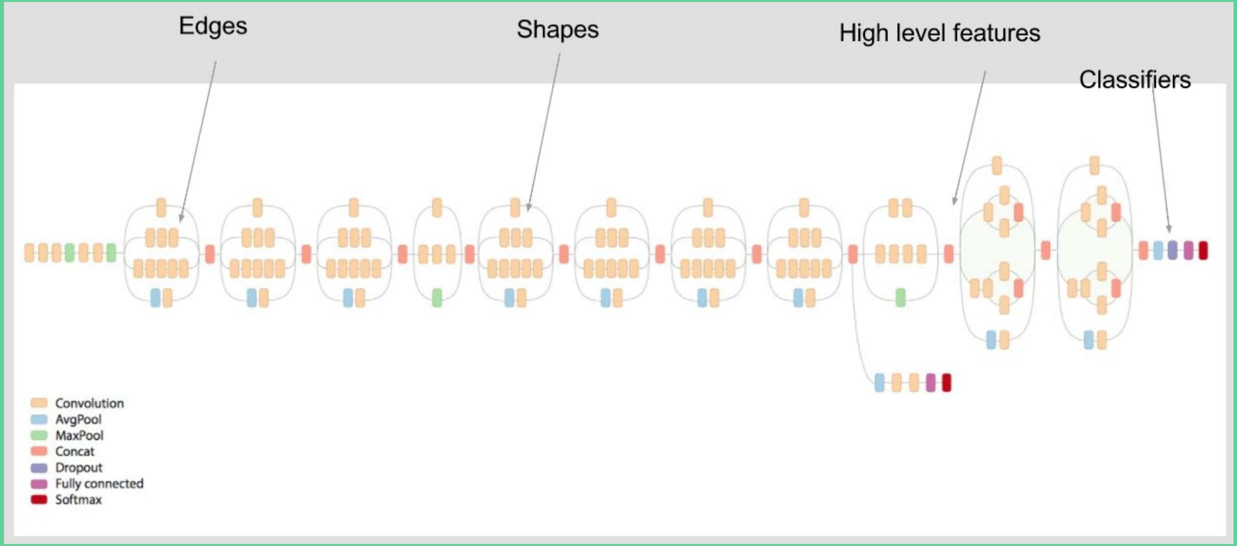
Feature Extractor

- We remove only the last full-connected layer of the pre-trained CNN and treat the rest as a fixed feature extractor
- I replace the final fully-connected layer with a new one to obtain a score for each painter in my dataset instead of a score for ImageNet classes.

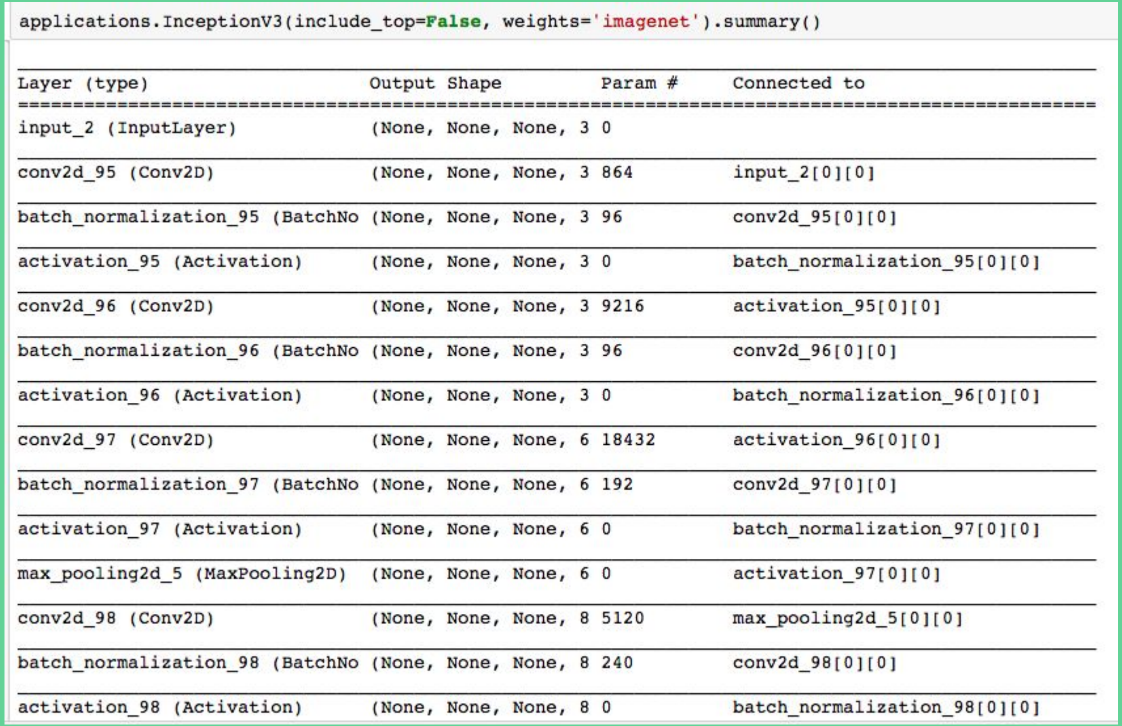


Feature Extractor

To verify if features from Google's InceptionV3 were useful as a starting point, I used it directly on my training data



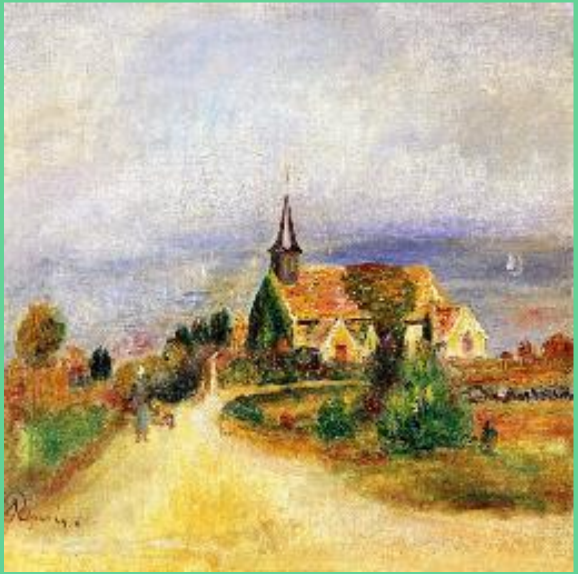
Inception V3 in Keras

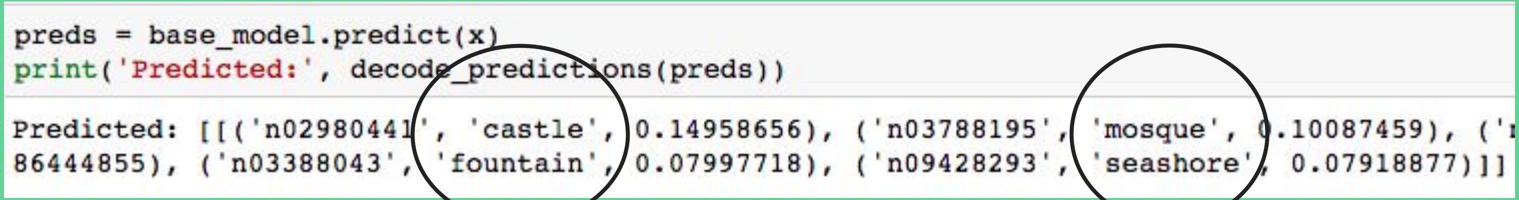


```
applications.InceptionV3(include_top=False, weights='imagenet').summary()
```

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_2 (InputLayer)             (None, None, None, 3 0
____________________________________________________________________________________________________
conv2d_95 (Conv2D)               (None, None, None, 3 864         input_2[0][0]
____________________________________________________________________________________________________
batch_normalization_95 (BatchNo (None, None, None, 3 96          conv2d_95[0][0]
____________________________________________________________________________________________________
activation_95 (Activation)       (None, None, None, 3 0           batch_normalization_95[0][0]
____________________________________________________________________________________________________
conv2d_96 (Conv2D)               (None, None, None, 3 9216        activation_95[0][0]
____________________________________________________________________________________________________
batch_normalization_96 (BatchNo (None, None, None, 3 96          conv2d_96[0][0]
____________________________________________________________________________________________________
activation_96 (Activation)       (None, None, None, 3 0           batch_normalization_96[0][0]
____________________________________________________________________________________________________
conv2d_97 (Conv2D)               (None, None, None, 6 18432       activation_96[0][0]
____________________________________________________________________________________________________
batch_normalization_97 (BatchNo (None, None, None, 6 192         conv2d_97[0][0]
____________________________________________________________________________________________________
activation_97 (Activation)       (None, None, None, 6 0           batch_normalization_97[0][0]
____________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)   (None, None, None, 6 0           activation_97[0][0]
____________________________________________________________________________________________________
conv2d_98 (Conv2D)               (None, None, None, 8 5120        max_pooling2d_5[0][0]
____________________________________________________________________________________________________
batch_normalization_98 (BatchNo (None, None, None, 8 240         conv2d_98[0][0]
____________________________________________________________________________________________________
activation_98 (Activation)       (None, None, None, 8 0           batch_normalization_98[0][0]
```

Feature Extractor





```
preds = base_model.predict(x)
print('Predicted:', decode_predictions(preds))
```

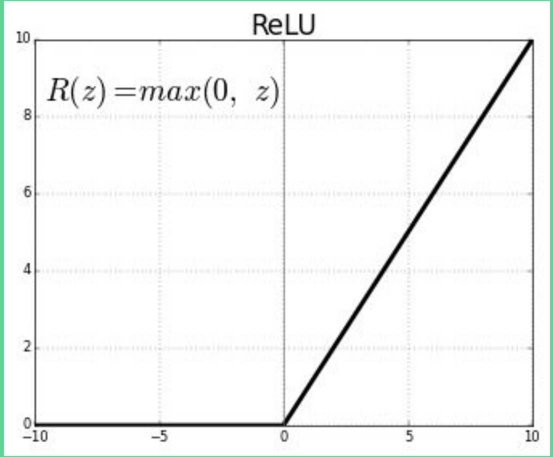
```
Predicted: [[('n02980441', 'castle', 0.14958656), ('n03788195', 'mosque', 0.10087459), ('
86444855), ('n03388043', 'fountain', 0.07997718), ('n09428293', 'seashore', 0.07918877)]]
```

Implementation

Metrics: I looked at the accuracy and (binary cross-entropy) loss for both training and validation set over the training epochs

I only used rectified linear activation functions (reLU) and the Adam optimizers

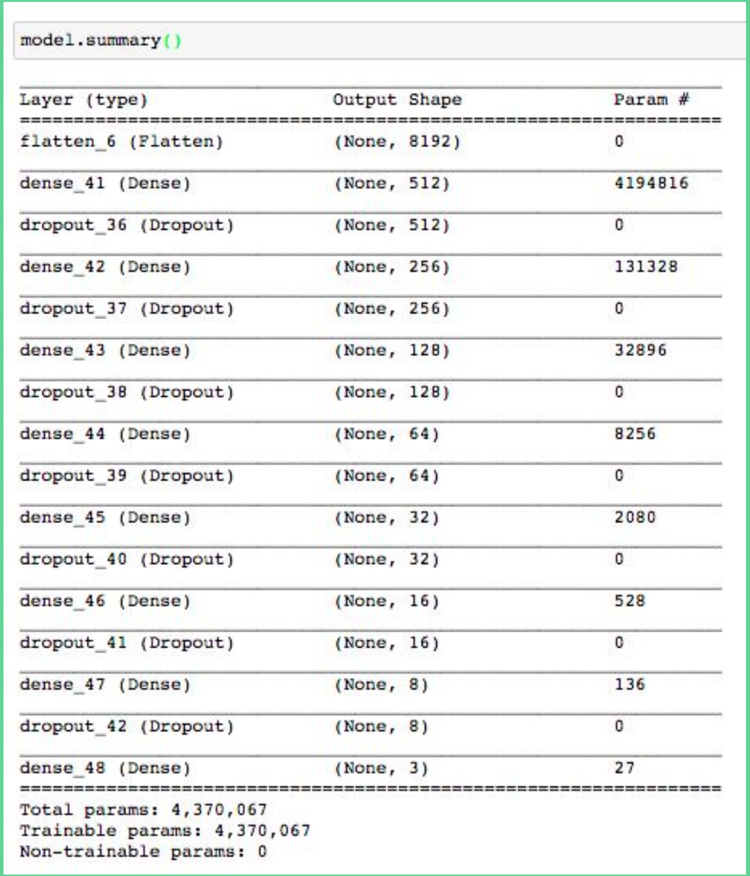
$$J = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$



Implementation

My fully-connected top model had convolutional layers with dimensions 512, 256, 128,..., 16 and 8 with 50% drop-outs in between. Adding layers was very helpful with both accuracy and losses.

Then I inserted a small CNN on top of the pre-trained network and trained it on those features get the classification I needed.

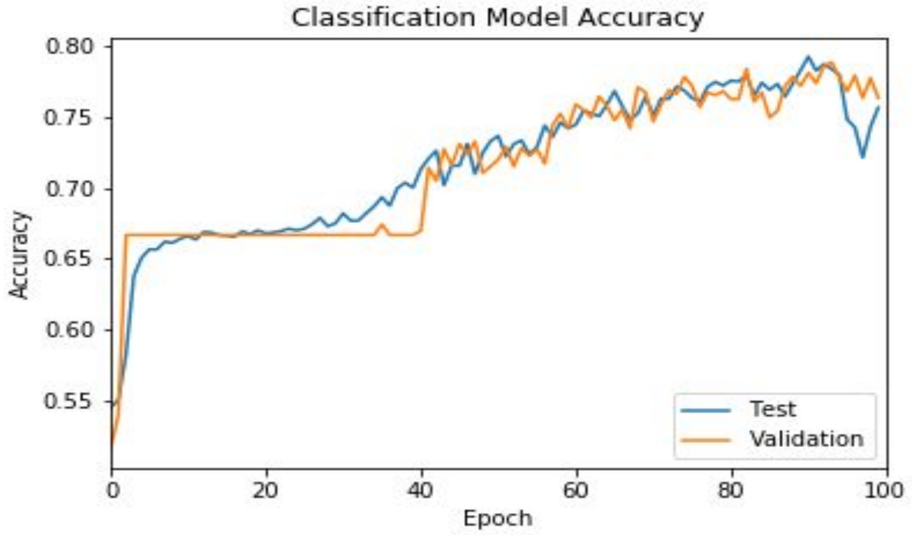


```
model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_6 (Flatten)          (None, 8192)              0         
_________________________________________________________________
dense_41 (Dense)             (None, 512)               4194816   
_________________________________________________________________
dropout_36 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_42 (Dense)             (None, 256)               131328    
_________________________________________________________________
dropout_37 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_43 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_38 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_44 (Dense)             (None, 64)                8256      
_________________________________________________________________
dropout_39 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_45 (Dense)             (None, 32)                2080      
_________________________________________________________________
dropout_40 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_46 (Dense)             (None, 16)                528       
_________________________________________________________________
dropout_41 (Dropout)         (None, 16)                0         
_________________________________________________________________
dense_47 (Dense)             (None, 8)                 136       
_________________________________________________________________
dropout_42 (Dropout)         (None, 8)                 0         
_________________________________________________________________
dense_48 (Dense)             (None, 3)                 27        
=================================================================
Total params: 4,370,067
Trainable params: 4,370,067
Non-trainable params: 0
```

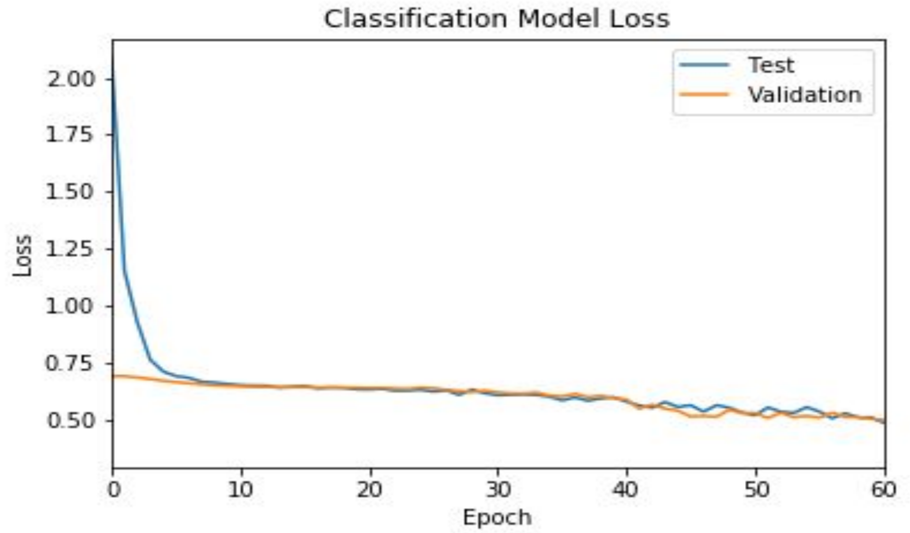
Metrics I

Accuracy for each epoch approaches 76% and baseline is 33.3%. Also train and validation accuracies track very close indicating little overfitting



Metrics II

Loss for each epoch - training and validation are also very close together (binary cross-entropy)



Conclusion I

- I used CNNs to identify the artist of a given painting. The dataset contains (after filtering) around 2000 paintings by 3 painters. Code structure allows this number to be any value (I started with 27 painters but it was too slow)
- I used transfer learning, fine-tuning the top layers of a deep pre-trained networks (Inception V3) and obtained:
    - reasonably high accuracy 76%
    - No overfitting

Conclusion II

- Confusion matrix analysis (partially done but not included in the slides)
- More thorough analysis using the KerasClassifier( ) wrapper for a classification neural network created in Keras and used in the scikit-learn library (keras.wrappers.scikit_learn)
- More painters, more paintings, larger networks, etc