Machine Learning Engineer Nanodegree

Capstone Project

Shawn Gong
January 10th, 2020

I. Definition

Project Overview

The art industry is a field that is considered as a highly human judgement based area. When critics evaluate the artistic value of a painting, the comments are mostly centric on the emotion the painting expresses, the life experience of the painter himself of herself and many other contextual or abstract perspectives. With the development of deep learning and artificial intelligence, many other tasks have been facing the shock of automation, while art industry is seemed as one of the field that is too human-centric to be replace by automation, despite there are already artificial intelligence that can paint. [1] Taking the buyer's perspective, there is extremely limited applications of AI to evaluate the artistic value of paintings as of now. With the development of artificial intelligence and deep learning, it would be interesting to see what the computers see in the paintings. Can the machine see the same pattern as the human beings do when they see a painting?

This project focuses on classifying artwork with the artists who painted them, which is a fundamental step of evaluating the artistic value of paintings. But with more and more descriptive variables available and more data being accessible, potential application including:

1. Automating the artwork evaluation process, making art trading more accessible and more transparent to the general public.
2. For education purposes, help students learn about art style, artists, eventually even aid the art creation process for students.

The dataset being used is from kaggle.com: https://www.kaggle.com/ikarus777/best-artworks-of-all-time. The dataset contains 5576 paintings from 50 of the most famous artists from history. The classes are largely imbalance, however, with the most productive painters painted more than 800 paintings while the least one only produced 20-30 paintings throughout their life time. It also includes a csv dataset that describes each artist by their birth and death year, their genre, their nationality.

The image dataset is mainly used for classification and the csv file is mainly used for exploratory data analysis.

The images are in different sizes and are mostly rectangle, with many different genres. Since we are classifying on the artists, it would be interesting to see whether the classifier can differentiate the artists under the same genre (Impressionism for example). Here is a small subset of the data:

Problem Statement

The main objective is to use deep learning techniques to classify paintings to their painters. The model being used is a convolutional neural network that is used for image classification. The goal is to get high accuracy and high F1 score for each classification. To ensure good performance, preprocessing is essential since the dataset is largely imbalanced.

The solution to the problem is a convolutional neural network (CNN) using transfer learning. I plan to use pre-trained weight architecture that has been proven to be successful in image classification including Xception and ResNet50. Both models has relatively moderate number of parameters to train, lowering the need for computation resources and are proven to be successful in image classification tasks. Though ImageNet is largely different from the project dataset, but the transfer learning technique has demonstrated it powerful generalization capabilities in many different tasks. The resulting model that achieved highest accuracy will be used for classifying image based on their painters.

Evaluation Metrics

Since the task is classification, the base evaluation metrics is accuracy, which is the most direct way to evaluate the models' performance. Moreover, because of the multiclass and imbalanced nature of the task, F1-score is another important metrics to evaluate model's performance on multiple classes. We will be able to see which classes did the model do a particularly good job classifying and the classes that the model has limited capability identifying.

For evaluation purposes, I also built benchmark model, which is a shallow convolutional neural network that has not been pre-trained on task to showcase the efficiency and accuracy of the proposed solution. For image classification, convolutional neural network has been proven successful in many fields, especially computer vision. As we are capturing the general patterns of the paintings, a shallow neural network that has been built from scratch is an appropriate benchmark model to be used here.

II. Analysis

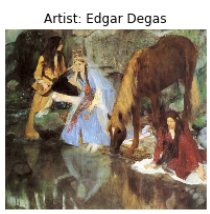
Data Exploration

The dataset I am using contains 5576 paintings that were painted by 50 artists throughout the history. These artists will be served as the classification targets in this project. They are:

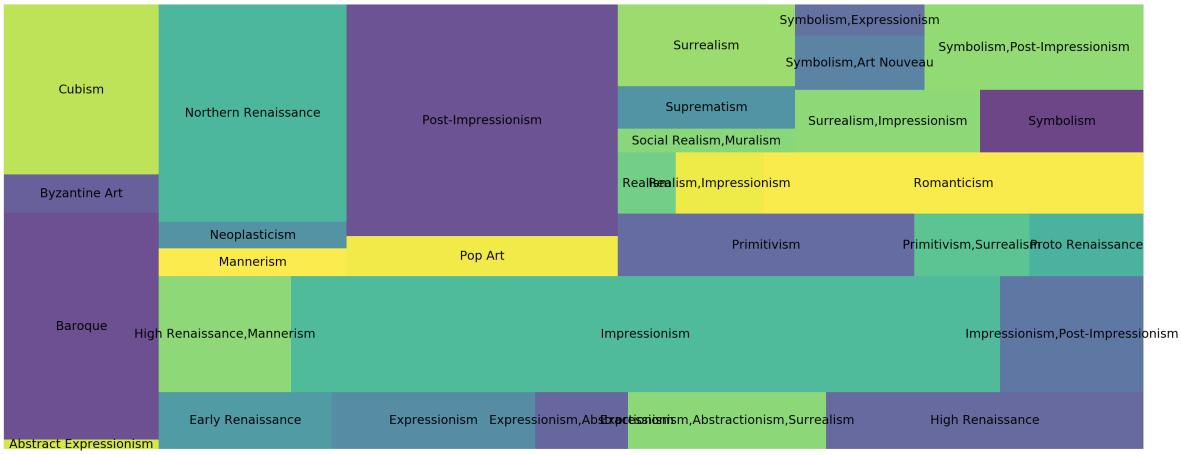
```
0             Amedeo Modigliani
1            Vasiliy Kandinskiy
2                  Diego Rivera
3                  Claude Monet
4                 Rene Magritte
5                 Salvador Dali
```

```
6                Edouard Manet
7                Andrei Rublev
8              Vincent van Gogh
9                  Gustav Klimt
10             Hieronymus Bosch
11             Kazimir Malevich
12               Mikhail Vrubel
13                Pablo Picasso
14            Peter Paul Rubens
15       Pierre-Auguste Renoir
16               Francisco Goya
17                  Frida Kahlo
18                     El Greco
19               Albrecht Dürer
20                Alfred Sisley
21               Pieter Bruegel
22                 Marc Chagall
23            Giotto di Bondone
24            Sandro Botticelli
25                   Caravaggio
26            Leonardo da Vinci
27              Diego Velazquez
28                 Henri Matisse
29                  Jan van Eyck
30                  Edgar Degas
31                    Rembrandt
32                       Titian
33   Henri de Toulouse-Lautrec
34              Gustave Courbet
35             Camille Pissarro
36               William Turner
37                  Edvard Munch
38                  Paul Cezanne
39             Eugene Delacroix
40               Henri Rousseau
41               Georges Seurat
42                    Paul Klee
43                Piet Mondrian
44                    Joan Miro
45                  Andy Warhol
46                 Paul Gauguin
47                      Raphael
48                  Michelangelo
49               Jackson Pollock
```

Here's a set of randomly printed paintings as an example to show the dataset:



We can see that the styles of the paintings are vastly different from one another. Just from the five images subset, we can see that there are Impressionism, Renaissance, Abstract Expressionism and Expressionism genres. The dataset covers genres including:

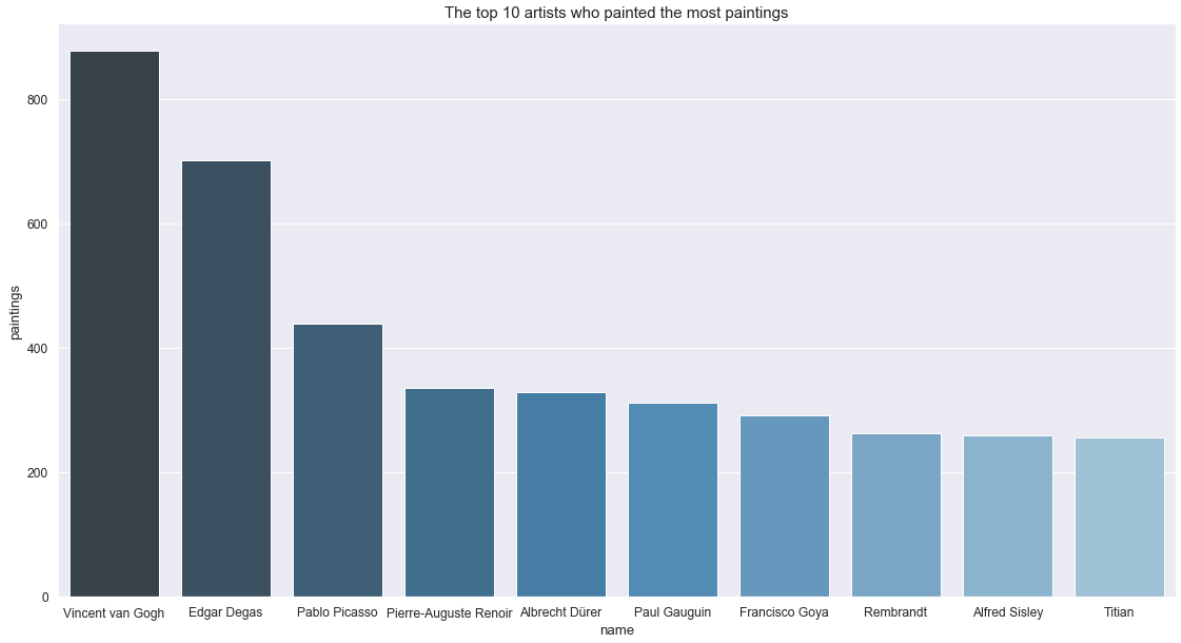


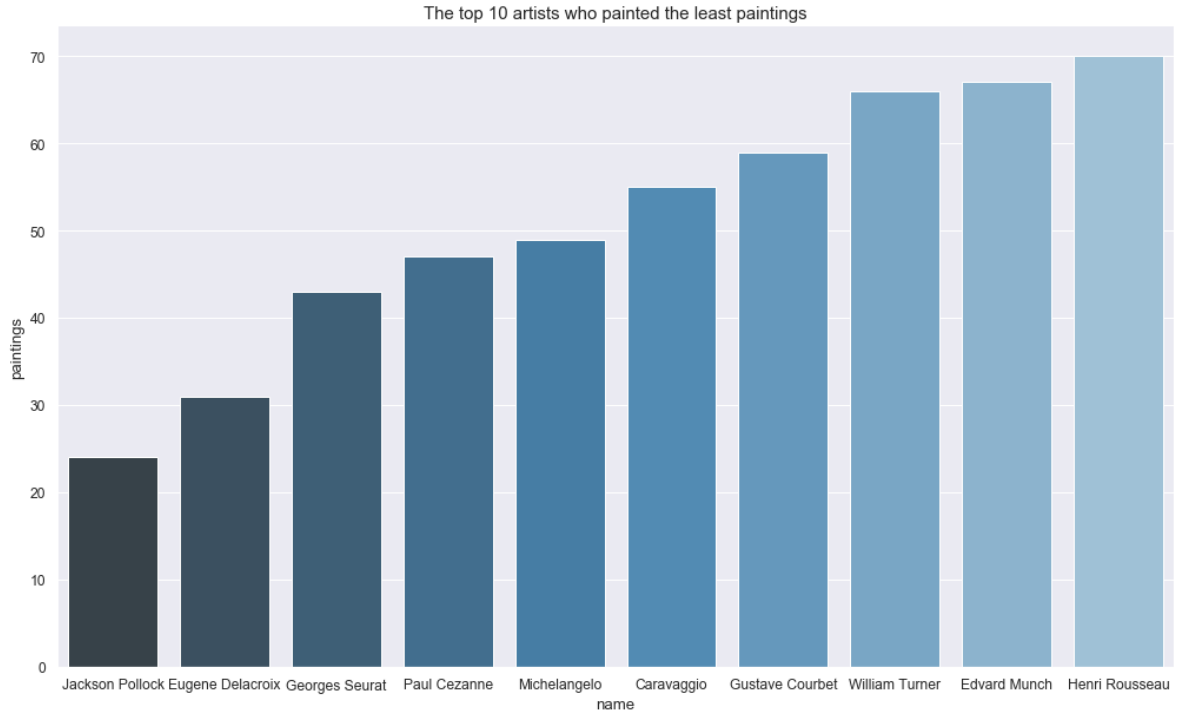
Above is a square plot of the paintings colored by each genre. As two of the most popular genres nowadays, we can see that Impressionism and Post-Impressionism has taken a large part of the dataset. Among the 50 artists, 15 of them are French or one of their nationality is French, 8 of them are Italian and 5 of them are Spanish, comprising more than half of the sample data.

	nationality	0
0	American	2
1	Austrian	1
2	Belgian	1
3	British	1
4	Dutch	4
5	Flemish	3
6	French	13
7	French,British	1
8	French,Jewish,Belarusian	1
9	German	1
10	German,Swiss	1
11	Italian	8
12	Mexican	2
13	Norwegian	1
14	Russian	4
15	Spanish	5
16	Spanish,Greek	1

Exploratory Visualization

As show in the bar plot, the gap between the top and the bottom 10 artists ranked by the number of paintings is very large. The champion, Vincent van Gogh, has more than 800 paintings available in the dataset, while the least of them all only has less than 25 paintings presented in the dataset. This demonstrates that the dataset is largely imbalance. In order to ensure the performance of the model, I will have to take a subset of the image dataset for this particular task.





Algorithms and Techniques

The Algorithm I plan to use is Convolutional Neural Network (CNN) for image classification. Convolutional Neural Network is a famous deep learning technique that has been proven to be successful in image classification task. It is based on MLP (Multi Layer Perceptron), which is commonly know as Feed Forward Neural Network as well. One of the reasons that CNN is widely used in image classification tasks is that it:

1. Is computationally efficient. CNN uses layers with kernels defined with height and width, which drastically reduces the number of parameters to train compare to MLP, significantly reduced the computational resources to be allocated on this task.
2. Feature extraction function. One of the basic component of CNN is kernel, which has a height and width and could scan over the image when training. With labels defined, the CNN could quickly memorize the patterns within the data and build connections with the class label.
3. Flexibility provided by hyperparameter tuning and layer architecture. There are many hyperparameters tunable in CNNs. The network also has many kinds of layers available for

data scientists to choose from based on the task. The example includes pooling layers, which reduces the number of parameters to train and improves the efficiency when training, batch normalization layers, which improves the speed and performance of CNNs and dropout layers, which randomly drop a proportion of parameters in each training epoch to prevent overfitting and make the model generalize better.

I will be using the transfer learning technique. This technique takes pre-trained CNN architectures trained on other image classification tasks and are proven to be useful. Through transfer learning, I will be able to reduce the training time significantly and reach a high performance benefitted from the knowledge pre-existed in the network architecture.

Benchmark

The benchmark model I used is a shallow CNN model trained with 20 epochs. The architecture is as following:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 111, 111, 16)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 109, 109, 32)      4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 54, 54, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 52, 52, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 26, 26, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 26, 26, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 128)       73856
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 128)       147584
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 5, 5, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 3, 3, 256)         295168
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
dropout_6 (Dropout)          (None, 1, 1, 256)         0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_2 (Dense)              (None, 18)                9234
=================================================================
Total params: 681,010
Trainable params: 681,010
Non-trainable params: 0
```

The accuracy of the benchmark model is 0.271 on the test set, which is significantly better than randomly guessing but can hardly be practical for classification. Although the model is already a shallow CNN, we can see that the model is already training nearly 700,000 parameters and takes 25 seconds to train one epoch on a RTX 2070.

III. Methodology

Data Preprocessing

For data preprocessing, I take the following steps to ensure the performance of the model:

1. Take subset of the data and remove the classes with less than 150 observations available.

I take a subset of the data and only selected artists who has more than 150 paintings available to ensure the model can generate adequate performance when classifying the images.

2. Assign class weight to the remaining classes to balance the

   One of the key evaluation metrics is F1-score:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

F1-score is an important evaluation metrics for imbalanced dataset classification. In imbalanced dataset, the accuracy measure loses its usefulness since the accuracy only consider the absolute correct and error but does not consider the uneven distribution of observations in each classes. Choosing model only based on accuracy can result in choosing very bad models that generates misleading results. However since I take a subset of the data, the imbalance of data is less of a concern. But I still calculated class weight to ensure the equality among classes. The calculation method is as following:

$$w_{class} = \frac{max(N_{painting})}{N_{artists} * \sqrt{N_{paintings}}}$$

Through this way, I can penalize more on the misclassification on underrepresented classes therefore balancing the performance of the model.

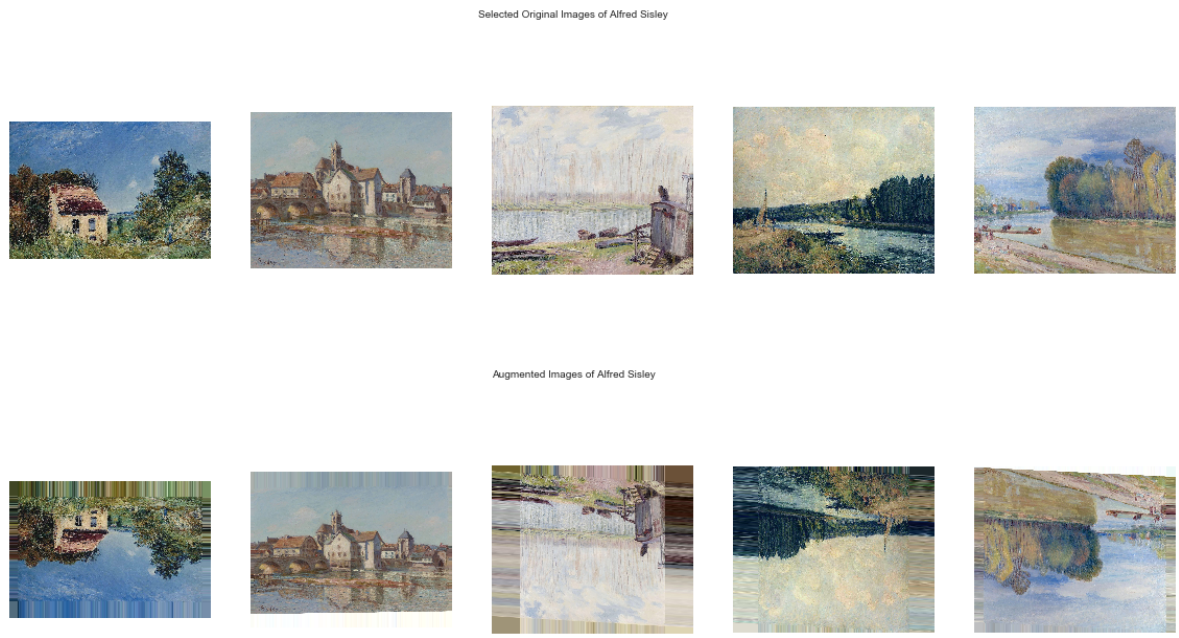
3. Image Augmentation

   I used image augmentation techniques to ensure the model can generalize well to more variation of the images it classifies on. The engine I used for image augmentation is as following:

```
# Augment data
batch_size = 16

# Input shape requirement of the specific architecture used in transfer learning
process
train_input_shape = (224, 224, 3)
n_classes = artists_subset.shape[0]

train_datagen = ImageDataGenerator(validation_split=0.25, #Split the training
data
                                   rescale=1./255.,
                                   #rotation_range=30,
                                   #width_shift_range=0.05,
                                   #height_shift_range=0.05,
                                   shear_range=5,
                                   zoom_range=[1.0, 1.4],
                                   #brightness_range = [0.9, 1.0],
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                  )
# Only rescalling needs to be done for test data
test_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(directory=images_dir_train,
                                                    class_mode='categorical',
```

```
 target_size=train_input_shape[0:2],
                                                    batch_size=batch_size,
                                                    subset="training",
                                                    shuffle=True,

 classes=artists_names.tolist()
                                                   )

valid_generator = train_datagen.flow_from_directory(directory=images_dir_train,
                                                    class_mode='categorical',

 target_size=train_input_shape[0:2],
                                                    batch_size=batch_size,
                                                    subset="validation",
                                                    shuffle=True,

 classes=artists_names.tolist()
                                                   )
test_generator = test_datagen.flow_from_directory(directory=images_dir_test,
                                                    class_mode='categorical',

 target_size=train_input_shape[0:2],
                                                    batch_size=batch_size,
                                                    shuffle=False,

 classes=artists_names.tolist())

STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST = test_generator.n//test_generator.batch_size
print("Total number of batches =", STEP_SIZE_TRAIN, "and", STEP_SIZE_VALID,
"and", STEP_SIZE_TEST)
```

Here's an example of the image augmentation:



Implementation

By using the transfer learning technique, I utilized the ResNet 50 and Xception architecture as the base model. Then I added two dense layers and the final classification layer. I set the optimizer as `rmsprop` with a learning rate of `1e-4` and accuracy as metrics to optimize the model. Here's an example for ResNet 50 transfer learning:

```
# Base-model ResNet 50
base_model = ResNet50(weights = "imagenet", include_top=False,
input_shape=train_input_shape)

for layer in base_model.layers:
    layer.trainable = True
output = base_model.output
output = Flatten()(output)
base_model = Model(base_model.input, output=output)

# Adding dense layers and compile the model for image classification
model = Sequential()
model.add(base_model)
model.add(Dense(512, activation='relu', input_dim=train_input_shape,
kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dense(32, activation='relu', kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dense(number_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer= optimizers.rmsprop(lr = 1e-4),
              metrics=['accuracy'])

# Train the model
checkpointer_resnet50 =
ModelCheckpoint(filepath='saved_models/weights.best.resnet50.hdf5',
                                verbose=1, save_best_only=True)
n_epoch = 20

CNN_1 = model.fit_generator(generator=train_generator,
steps_per_epoch=STEP_SIZE_TRAIN,
                               validation_data=valid_generator,
validation_steps=STEP_SIZE_VALID,
                               epochs=n_epoch,
                               shuffle=True,
                               verbose=1,
                               callbacks=[checkpointer_resnet50],
                               use_multiprocessing=False,
                               workers=16,
                               class_weight=class_weight
                              )
```

Refinement

During the model building process, I found that the model's performance remained stagnant starting from epoch 20. Initially, without any improvement, the cross validation accuracy is around 0.75.

```
Epoch 00018: val_loss did not improve from 0.88750
Epoch 19/100
209/209 [==============================] - 50s 239ms/step - loss: 0.8024 - acc:
0.9284 - val_loss: 0.9874 - val_acc: 0.7412
```

In order to refine the model, I added some callback method and some new layers to improve the performance, including:

1. Adding dropout layer

   Dropout layer can be very useful for improving the performance of the model and preventing overfitting. I set the dropout ratio at 0.3, which gives optimal performance compare to other ratios.

2. Adding callback method such as `ReduceLROnPlateau` and `Early Stop`.

   One thing that could cause the cross validation accuracy to remain stagnant is that the learning rate is too high after the model has been training for some epoch. In this case, reducing the learning rate when the learning rate becomes stagnant is a desirable solution to keep the model training properly.

   ```
   early_stop = EarlyStopping(monitor='val_loss', patience=20, verbose=1,
                              mode='auto', restore_best_weights=True)

   reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5,
                                 verbose=1, mode='auto')
   ```

   I set the patience argument as 5 so that the model will reduce learning rate by a factor of 0.1 when the loss hasn't been reducing for 5 epochs in a row. I also utilized `early_stop` callback method so that I can train for longer epochs until the model stops training after the model does not improve for 20 epochs in a row.

After setting up these refinements, I increased the number of epochs to train to 100 so that I can exhaust the space of improvement for the model. The model accuracy then increased to 0.84 on test set.

```
n_epoch = 100

CNN_1 = model.fit_generator(generator=train_generator,
steps_per_epoch=STEP_SIZE_TRAIN,
                            validation_data=valid_generator,
validation_steps=STEP_SIZE_VALID,
                            epochs=n_epoch,
                            shuffle=True,
                            verbose=1,
                            callbacks=[reduce_lr, checkpointer_resnet50,
early_stop],
                            use_multiprocessing=False,
                            workers=16,
                            class_weight=class_weight
                           )
```

```
Epoch 00037: val_loss did not improve from 0.56769
Epoch 38/100
209/209 [==============================] - 50s 239ms/step - loss: 0.2836 - acc:
0.9874 - val_loss: 0.5645 - val_acc: 0.8533
```

```
Epoch 00038: val_loss improved from 0.56769 to 0.56449, saving model to
saved_models/weights.best.resnet50.hdf5

Epoch 00057: val_loss did not improve from 0.56449
Epoch 58/100
209/209 [==============================] - 50s 240ms/step - loss: 0.2444 - acc:
0.9907 - val_loss: 0.5720 - val_acc: 0.8378

Epoch 00058: ReduceLROnPlateau reducing learning rate to 9.99999943962493e-12.

Epoch 00058: val_loss did not improve from 0.56449
Restoring model weights from the end of the best epoch
Epoch 00058: early stopping
```

IV. Results

Model Evaluation and Validation

The final model I arrived to is a transfer learning model based on Xception architecture, which has 20,861,480 parameters in the base model and 72,261,402 parameters after adding the dense layers. It reaches an overall accuracy of 0.84 on the test set that I separated from the original set before the training, which is considered a reasonable performance for this kind of task. I also created a classification report on the classes:

Classification report	Precision	recall	f1-score	support
Vincent van Gogh	0.85	0.89	0.87	175
Edgar Degas	0.89	0.84	0.86	140
Pablo Picasso	0.67	0.69	0.68	88
Auguste Renoir	0.91	0.88	0.89	67
Albrecht Dürer	0.97	0.95	0.96	66
Paul Gauguin	0.85	0.76	0.80	62
Francisco Goya	0.84	0.81	0.82	58
Rembrandt	0.87	0.90	0.89	52
Alfred Sisley	0.96	0.85	0.90	52
Titian	0.89	0.82	0.86	51
Marc Chagall	0.93	0.54	0.68	48
Rene Magritte	0.72	0.92	0.81	39
Amedeo Modigliani	0.86	0.92	0.89	39

Classification report	Precision	recall	f1-score	support
Paul Klee	0.78	0.95	0.86	38
Henri Matisse	0.76	0.78	0.77	37
Andy Warhol	0.82	0.92	0.87	36
Mikhail Vrubel	0.72	0.76	0.74	34
Sandro Botticelli	0.65	1.00	0.79	17
Accuracy			0.84	1099
Macro Average	0.83	0.84	0.83	1099
Weighted Average	0.84	0.84	0.84	1099

We can see that the model achieved good performance classifying the paintings overall, though it has some trouble classifying some of the artists' paintings. For example, Pablo Picasso only has a F1-score of 0.68, possibly due to his Abstract Expressionism painting style is hard for model to recognize patterns. Since the model is evaluated based on the test set that is separated before the training process, the model is considered to be robust enough for such problem. Overall, the model can achieve pretty good classification performance on the dataset.

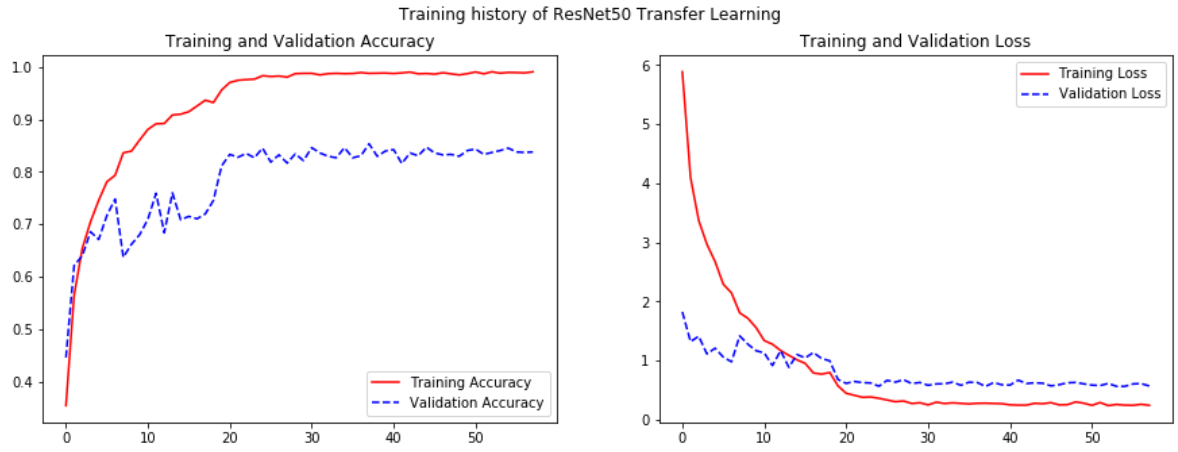
Justification

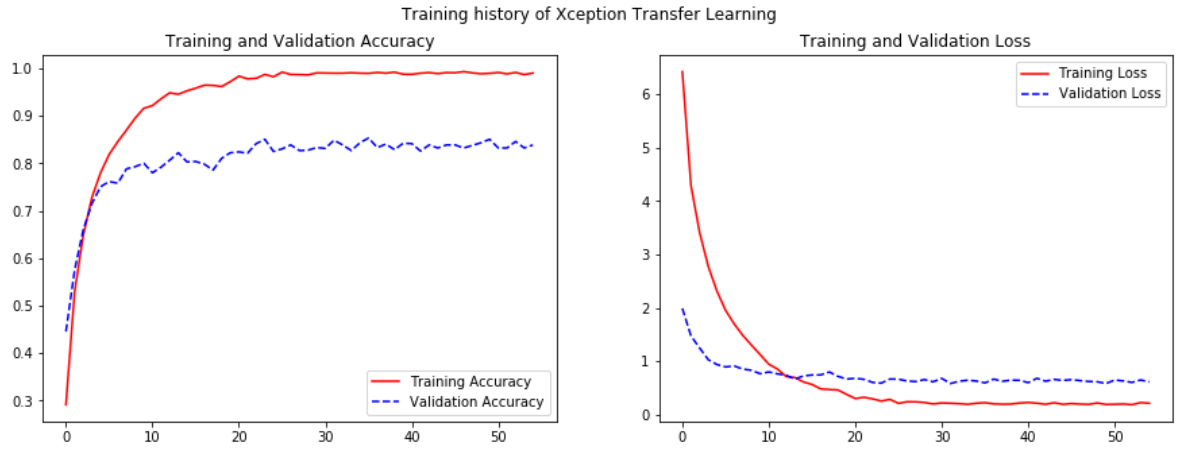
The model achieved 0.84 overall accuracy, significantly higher than the benchmark model's 0.271. The model performs well on the classification task for the subset of the original painting data.

However, the model faces some limitations in practical circumstances as well. In modern art, it is harder and harder to find patterns that could be used to identify an artists, and the contemporary artists take more "free-from" approach and a CNN that is trained mostly on identifying patterns and features could easily be confused by the free-form paintings that are commonly seen in contemporary art.

V. Conclusion

Free-Form Visualization





Above are the training history plot of ResNet50 and Xception based transfer learning, both using the same set of layers, callback method and training epoch. As we can see, the Xception architecture has a way smoother training history, is more quickly achieving a reasonable performance and has slightly higher validation accuracy compare that of ResNet50. Despite having similar number of parameters. Xception has a more superior performance compare to ResNet50. Although there are indeed more transfer learning options I can choose from, but Xception has significantly fewer parameters,and achieves a pretty good performance for this task.

Reflection

In this project, I have used transfer learning techniques to classify paintings to the artists who painted them. I started from exploratory data analysis and identifying potential problems that need to be solved. Then I process to build the models to compare the performance and selected the final Xception model that has better performance compare to the benchmark model and the ResNet50 based model. During the process, in order to improve the performance, I adjusted hyperparameters and refined the model and incorporated callback method to improve the performance. The aspect I found challenging was adjusting the hyperparameters to an ideal state. This is both applicable to the image preprocessing state and the model building stage. Even when pre-trained architecture provides performance that is good enough, there are still plenty of room for improvement. Because there are so many things to adjust, finding the ideal hyperparameters that could improve the performance is the most difficult task I found during working on this project.

This project is mainly for personal interest and for fun. In a way, this project reveals the undeniable presence of personal style in artwork creation and is a lot fun to work on. Moreover, this model could be practical applicable on a very small scale. Given today's booming art industry, it is hard to tell whether the model would be universally useful in classifying the artwork, but it is definitely a great fun to work on this project.

Improvement

There are more advanced and more complex architectures available for use. But my access to computational resource is limited. There are InceptionResNet50, NASNetLarge in keras alone that provide better performance on the ImageNet, which could translate to better performance for this task as well. I also attempted to train a proportion of the layers in the network while freeze other layers, but the performance is about the same as training on all the layers. I think there could be best practices to this technique, as well. Additionally, the dataset could also have more descriptive variables, as well. As of now, the dataset is only capable for a simple classification task.