Artistic Movement Recognition

Abhishek Kumar
University of Massachusetts Amherst
abhishekk@umass.edu

Arpit Singh
University of Massachusetts Amherst
arpitsingh@umass.edu

Abstract

The project is about automatically recognizing the influence of artistic movements on digital paintings using deep convolutional neural networks. While a single image can have characteristics of multiple movements, we focussed on single label prediction. We explored transfer learning using models trained on ImageNet dataset like VGG, ResNet and also tried a couple of custom model architectures to classify the images containing the art paintings into different artistic styles. Our ensemble gave an accuracy of 0.5229.

1. Introduction

In the recent times, the art museums are trying to create online records of their paintings. This opens a lot of opportunities to use technology in this area. One of the things that can be done is to list/sort the paintings by their artistic styles. We are working on automatically classifying the digital paintings into artistic styles.

It is interesting because for the paintings, it is hard to come up with a set of manual features that can be put into the classification model for this kind of task. So, we want to explore how to use deep neural network models for this which can learn its own features from the dataset and predict the results. In this project, we explored how we can tweak the pre-trained models like VGG, ResNet, etc. to obtain better results. We experimented using features from different layers of these mentioned networks and also tried a couple of custom VGG based residual-style architecture which computes spatially invariant statistics similar to the gram matrix of neural-style transfer and uses it to fine-tune multiple layers via backpropagation.

2. Related Work

The researchers have previously used various techniques to automatically recognize the influence of artistic movements on digital paintings. However, they mainly used to focus on identifying the set of features and applying traditional machine learning models on those features. For instance, in the paper by Khan et al, 2014 [4], the authors used features like GIST, LBP, etc. and put them into bag-of-words framework. Their dataset consisted of 4266 images out of which they were able to accurately predict for over 60% of the images.

The accuracy of such traditional machine learning models decreases when applied on larger dataset as it is difficult to come up with such diverse features that can sufficiently capture the model. The deep neural networks are able to learn the features by themselves. Hence, in recent times, the deep neural networks are being actively researched in this area. In the paper by Florea et al, 2017 [2], the authors used features like color

structure and the novel topographical features and put them into an adapted boosted ensemble of SVM (support vector machine) models. On the same dataset as ours, they achieved a recognition rate of 50.1%. But on using transfer learning based on AlexNet originally trained on ImageNet, they were able to achieve a recognition rate of 56.5%.

By taking inspiration from their paper, in this project, we explored transfer learning using models trained on ImageNet dataset like VGG and ResNet and also tried a few custom model architectures to classify the influence of artistic movements on digital paintings.

3. Dataset

We will be using Pandora 18K dataset for our project which can be found at the link: `http://imag.pub.ro/pandora/pandora_download.html`[2]. The dataset consists of 18038 images distributed among 18 classes namely Byzantine Iconography, Early Renaissance, Northern Renaissance, High Renaissance, Baroque, Rococo, Romanticism, Realism, Impressionism, Post Impressionism, Expressionism, Symbolism, Fauvism, Cubism, Surrealism, Abstract art, Nave art and Pop art. Each genre is illustrated by a number of images varying from 700 to more nearly 1200.

The database was verified by engineers to ensure that only the relevant part of the images are shown and by an art expert to make sure that artistic movement annotation is true.

4. Approach

For carrying out our experiments we used KERAS which is an open source neural network library written in Python [1]. Keras was chosen due to its simplicity and low learning threshold. TensorFlow was used as the backend. We were not able to load all the training data in the memory due to its large size. Keras enabled us to get past this issue as it supports DataGenerators where one can generate samples in batches which can then be passed to various models. We used 20 as our batch size. With Intel i7-7500U CPU [2.70GHz * 4] and 8 GB ram, which was our initial setup's configuration, the training of deep models was very slow. So we decided to switch to Google Colaboratory which provides python environment requiring no setup to use and runs entirely in the cloud. Keras ,with GPU support provided by Google Colab, enabled us to experiment with multiple models and architecture in short time.

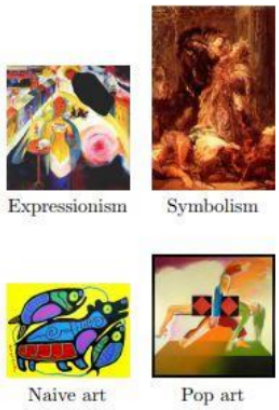

Figure 1: Sample images for a few classes of the dataset

First we used conventional machine learning models and image features to get an idea of how they perform. This worked as our baseline. We then applied transfer learning where we took deep convolutional networks pre-trained on ImageNet dataset like VGG19, ResNet,etc. and used features from different layers to train a set of new fully connected layers generating prediction over 18 classes. We also tried a couple of VGG based custom architecture which try to capture style of the images and use it for improving prediction.

4.1. Baseline

We were able to achieve a reasonable mean classification accuracy of 0.217 from our base-

line model using Histogram of Oriented Gradients feature and Linear SVM. This reported accuracy is better than what [2] reports for Hog feature with SVM. We also tried some other features like color histograms with KNN but got lower accuracy of 0.15. HOG was chosen as it is resistant to small deformations and illumination changes. Since the images in the dataset are large, we had to resize them to smaller sizes (tried 100*100 (accuracy of 0.217 reported above is for this case) and 300*300). We trained and tested our baseline model using a 4:1 train/test split of the dataset.

4.2. Transfer Learning

In our project, our focus was to explore transfer learning using pre-trained models like VGG, ResNet, etc. We experimented by tweaking at various levels of depth in VGG.

VGG19 originally has a total of 24 layers (including layers without weights). Initially, we took the output of the 19th layer and fed it to an FC layer consisting of 1024 units. We also used dropout to reduce overfitting as there was sufficient gap between train and test accuracy for models where deeper features were used to train the FC layers. So in our model (let's call it as VGG19) we had 19 layers of VGG followed by Flatten layer, FC layer containing 1024 units, Dropout layer (with p=0.5), FC layer containing 1024 units and softmax layer.

For our next model (we will call it as VGG17), we fed the output of the 17th layer of VGG to our FC layer (with the same layout as the model above). After this, for our next model (we will call the model as VGG15), we went above and fed the output of the 15th layer of VGG to our FC layer.

We also tried ResNet and fed it in our FC layer in the similar manner as the above models.

4.3. Custom architecture

Since we are trying to classify paintings, we thought that capturing style and texture of the image may help us in better prediction. We took inspiration from the work of Leon A. Gatys et al. [3], which uses a 2D representation called Gram matrix which for a layer is mean of the inner product of its feature maps. This captures the pairwise feature activations. Since in style transfer, we have a single style image, this works fine. But, in our problem, we needed to capture the style of all the images and a 2D representation for each of them would need too much memory. Also since these need to generated while training models, the training process would be significantly slower. What is important here is some statistics of activation which is spatially invariant. So we used a 1D representation to capture styles of images where to capture style using a layer's output, we took mean across space which gives spatially invariant representation of the intermediate layer outputs similar to the gram matrix. This was also discussed in the lecture: `https://www.youtube.com/watch?v=GHVaaHESrlY&t=3921s`.

We tried two custom models using VGG which are similar to the residual networks. The custom layers in these models compute the above mentioned spatially invariant representation of the intermediate layer outputs.

In the first model we computed the 1D statistics from intermediate layers 9, 4 and 16 and then directly fed them to the fully connected layers after concatenating them. The FC layers were also receiving flattened output of the standard VGG19 stack. Just using the style representation for classification didn't give us good results. So we decided to merge the VGG output and the computed style representations to train the FC layers. Later we use "Custom VGG 1" to refer to this model. Table 1 and figure 2 below show this

architecture.

In the second model, we followed a different strategy which was executed in two phases. We first passed all the training images through pre-trained VGG and stored classwise 1D style representations representing the style of the class. In the second phase we used a modified VGG model with two inputs and two losses, where the first input was the normal batch of training images and the second ones were the mean activations of the the corresponding true classes computed in phase 1. The two losses were softmax which is used in standard VGG and a custom L2 loss to account for the difference in the style representations generated for the current batch and the mean style representation of true class. The idea was that if all the images belonging to the same class generate similar style representations while going through VGG, then the model will better be able to place them in the same category and predict same labels for them. The softmax loss made sure that the prediction also matches with the true class. Since we wanted better prediction and also the two losses were different in magnitude, we gave different weights to the two losses. We refer to this model as "Custom VGG 2". Figure 3 shows this architecture.

In both the figures the lambda layers calculate the 1D representation capturing style.

5. Experiments and Results

5.1. Transfer Learning

We tried various hyper parameters for training our models. A good way to do this is to monitor loss and accuracy for few epochs. Below we talk about the individual models separately. We used categorical cross-entropy as a loss function for classification in all of these models.

We trained the VGG19 model with 19 layers for 20 epochs using SGD optimizer (with learning rate=0.00001 and momentum=0.5) on our dataset and achieved an accuracy of 0.46. For VGG17, we used 20 epochs and SGD optimizer (with learning rate=0.00001 and momentum=0.5) which gave an accuracy of 0.3776. We noticed that there was a decrease in accuracy. After training the VGG15 model for 20 epochs using SGD optimizer (with learning rate=0.001 and momentum=0.5), we obtained an accuracy of

Table 1: Cutom VGG 1

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(224, 224, 3)	0	
block1_conv1 (Conv2D)	(224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Conv2D)	(224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Conv2D)	(112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Conv2D)	(112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Conv2D)	(56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Conv2D)	(56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Conv2D)	(56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Conv2D)	(28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Conv2D)	(28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Conv2D)	(28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Conv2D)	(14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Conv2D)	(14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Conv2D)	(14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(7, 7, 512)	0	block5_conv3[0][0]
flatten_7 (Flatten)	(25088)	0	block5_pool[0][0]
dense_25 (Dense)	(1024)	25691136	flatten_7[0][0]
lambda_19 (Lambda)	(256)	0	block3_conv3[0][0]
lambda_20 (Lambda)	(512)	0	block4_pool[0][0]
lambda_21 (Lambda)	(512)	0	block5_conv2[0][0]
dropout_13 (Dropout)	(1024)	0	dense_25[0][0]
concatenate_7 (Concatenate)	(2304)	0	lambda 19,20,21,13
dense_26 (Dense)	(1024)	2360320	concatenate_7[0][0]
dropout_14 (Dropout)	(1024)	0	dense_26[0][0]
dense_27 (Dense)	(1024)	1049600	dropout_14[0][0]
dense_28 (Dense)	(18)	18450	dense_27[0][0]

Table 2: Custom VGG 2

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(224, 224, 3)	0	-
block1_conv1 (Conv2D)	(224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Conv2D)	(224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Conv2D)	(112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Conv2D)	(112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Conv2D)	(56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Conv2D)	(56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Conv2D)	(56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Conv2D)	(28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Conv2D)	(28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Conv2D)	(28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Conv2D)	(14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Conv2D)	(14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Conv2D)	(14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(7, 7, 512)	0	block5_conv3[0][0]
flatten_1 (Flatten)	(25088)	0	block5_pool[0][0]
dense_1 (Dense)	(1024)	25691136	flatten_1[0][0]
lambda_1 (Lambda)	(128)	0	block2_conv2[0][0]
lambda_2 (Lambda)	(256)	0	block3_conv2[0][0]
lambda_3 (Lambda)	(512)	0	block4_conv1[0][0]
lambda_4 (Lambda)	(512)	0	block4_conv3[0][0]
lambda_5 (Lambda)	(512)	0	block5_conv2[0][0]
dropout_1 (Dropout)	(1024)	0	dense_1[0][0]
concatenate_1 (Concatenate)	(1920)	0	lambda 1,2,3,4,5
aux_input (InputLayer)	(1920)	0	-
dense_2 (Dense)	(1024)	1049600	dropout_1[0][0]
subtract_1 (Subtract)	(1920)	0	concatenate_1[0][0],aux_input[0][0]
LabelDiff (Dense)	(18)	18450	dense_2[0][0]
activationDiff (Lambda)	(1920)	0	subtract_1[0][0]

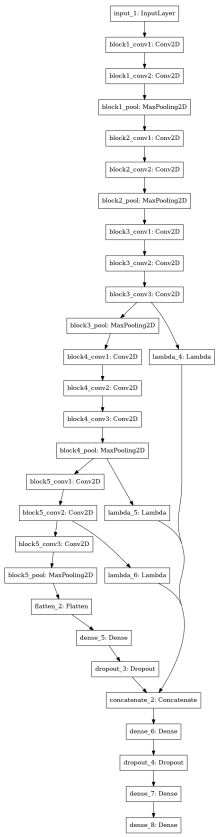

Figure 2: Custom VGG 1

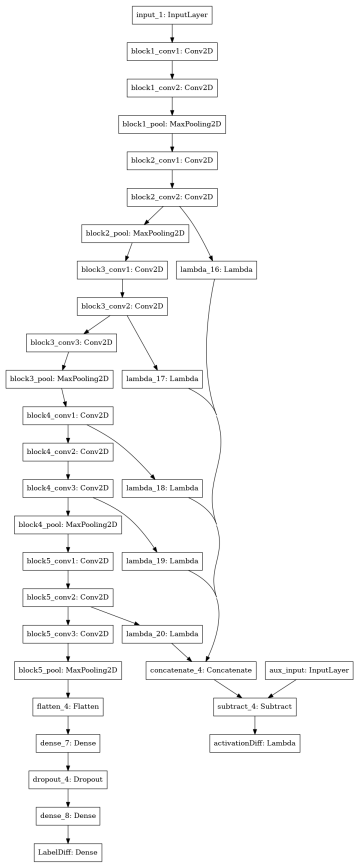

Figure 3: Custom VGG 2

0.1588 on the testing data which is even further less.

We think that since the bottom layers contain more general features like edges, etc., they are not able to capture the model that well and that is why, using the deeper layers to train for our data results in worse accuracy. So, deeper the layer that is fed to our FC layer that we added, lesser is the accuracy of the corresponding model. We felt that going below this (by feeding the output of a further deeper layer to our FC layer) seems to make no point as those layers will be able to capture even more general features.

The VGG19 with 19 layers gave the best result among all the models that we tried while leveraging transfer learning. Also, we can see in figure 7 that even after introducing dropouts between the FC layers, there is significant overfitting.

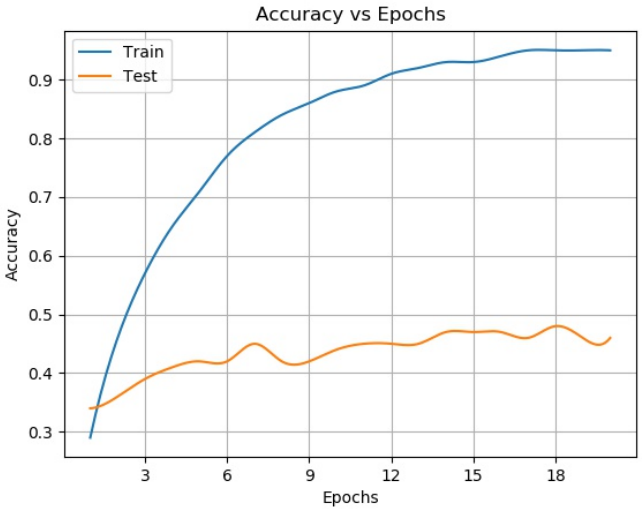

Figure 4: Accuracy with VGG depth 19

With VGG depth 17 also there is overfitting. The test accuracy in this case is lower than what we got with using all the VGG layers in earlier case.

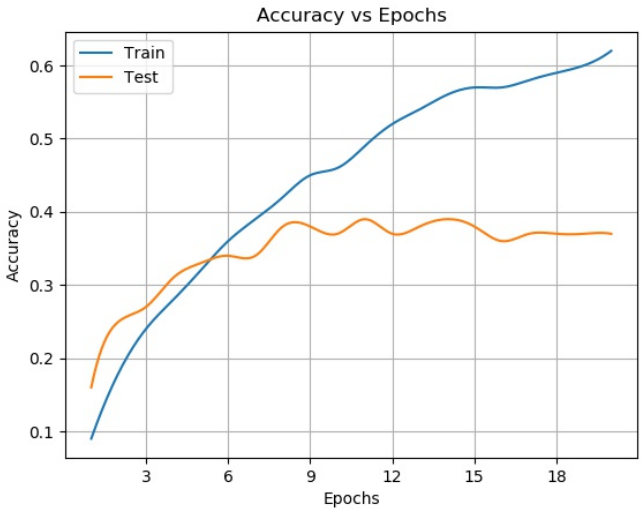

Figure 5: Accuracy with VGG depth 17

In case of VGG depth 15, both the train and test accuracies come out to be poorer than the earlier two models. We see a trend of decrease in test accuracies as we remove more layer from the bottom of the VGG19 stack which may be happening due to the issue discussed in the earlier paragraphs.

We also tried ResNet and when trained for 40 epochs For ResNet, using SGD optimizer (with learning rate=0.0001 and momentum=0.8), we were able to get an accuracy of 0.4261. With ResNet50, we were expecting to beat all the VGG based models, but it didn't happen. With 20 epochs, we got an accuracy of around 0.4 which is lower then that of VGG depth 19. There isn't a large gap between the test and train accuracies suggesting low overfitting.

5.2. Custom Architecture

We wanted to test the performance of our custom architectures for more number of epochs to see how they perform. So for Custom VGG 1, we used 80 epochs and as can be seen in plot 8 there was a continuous increase in both the train and test accuracies over epochs. This might be attributed to the architecture which is more complex than the perviously used models as in it there is an extra FC layer which learns a 1D representation from the final layer of VGG19 before it goes to the final 2 FC layers along with the 2304 dimensional style vector. This style vector is the

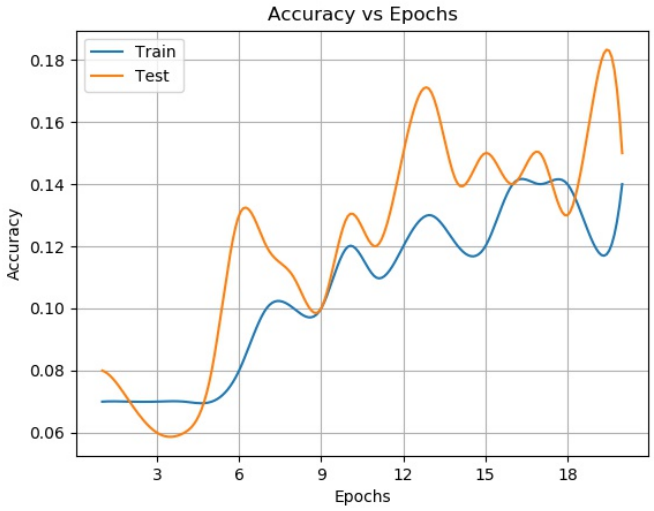

Figure 6: Accuracy with VGG depth 15

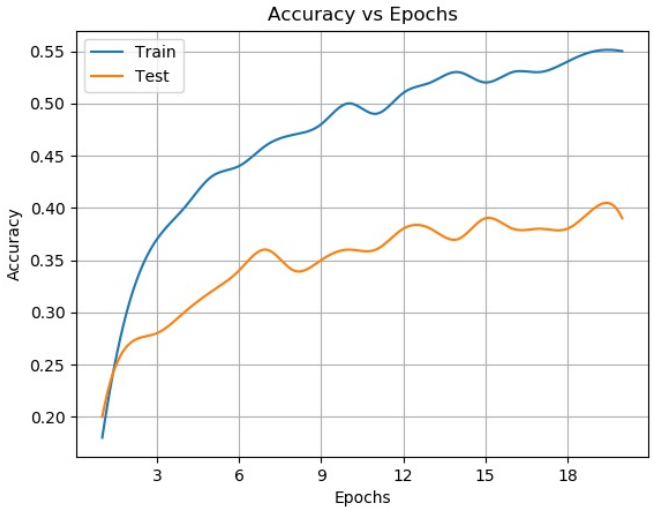

Figure 7: Accuracy with ResNet50

concatenation of the style representations of the various intermediate layers as show in the above sections. At the end of the training, we were able to get a train accuracy of 0.6 and a test accuracy of 0.51. For this custom model we used adam optimizer (with learning rate=0.000005, beta1 = 0.9, beta2 = 0.999, epsilon=None and decay=0)

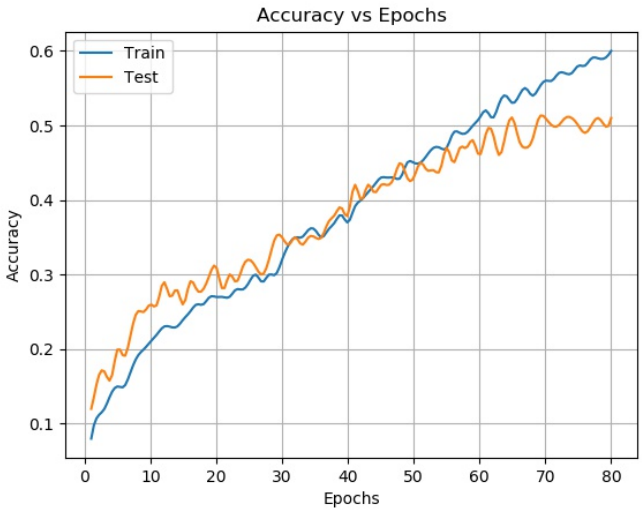

Figure 8: Accuracy with Custom VGG 1

For the custom VGG 2 we were able to just use 20 epochs as the training was considerably slower. Here we allowed even the intermediate layers which were generating style representations to update their weights via back propagation. We used adam optimizer (with learning rate = 0.0000005, beta1 = 0.9, beta2 = 0.999, epsilon = None and decay=0). The learning rate here was kept $1/10$ of the learning rate used for Custom VGG 1 as here we are allowing the update in the intermediate layers also and using larger rate was proving disruptive.

5.3. Ensembling

After this, we took a voting-based ensemble of VGG19, ResNet and Custom VGG 1. We were able to boost the accuracy to 0.5229 which is our final accuracy.

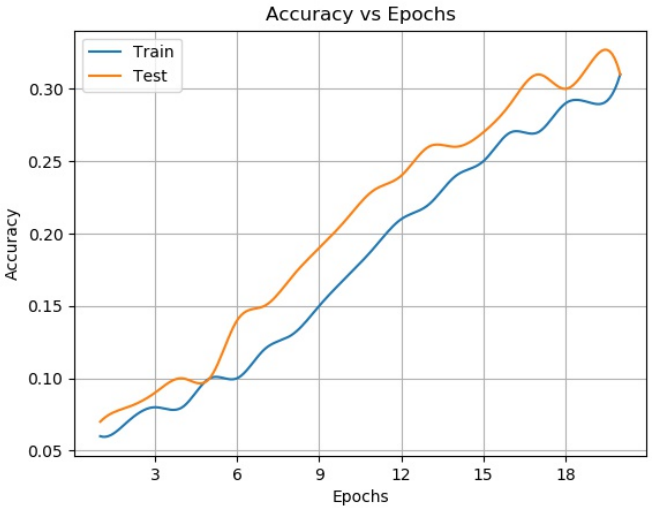

Figure 9: Accuracy with Custom VGG 2

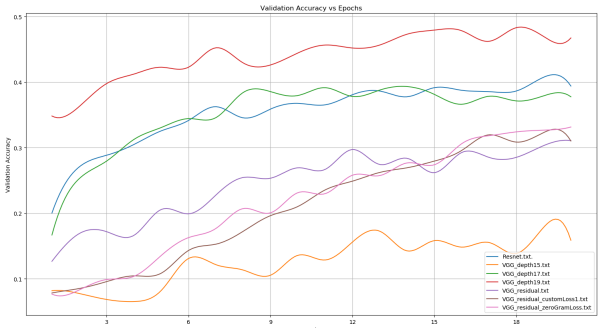

Figure 10: Validation accuracy of different models

Table 3: Test accuracy of different models.

Model [Default: 20 Epochs]	**Val Accuracy**
SVM with HOG (baseline)	0.217
VGG_depth_19	0.46
ResNet [40 Epochs]	0.4261
VGG_depth_17	0.3776
VGG_depth_15	0.1588
Custom VGG 1	0.3108
Custom VGG 2	0.3102
Custom VGG 1[80 Epochs]	0.51
Ensemble (final)	0.5229

The table above shows the test accuracies for the different models that we experimented.

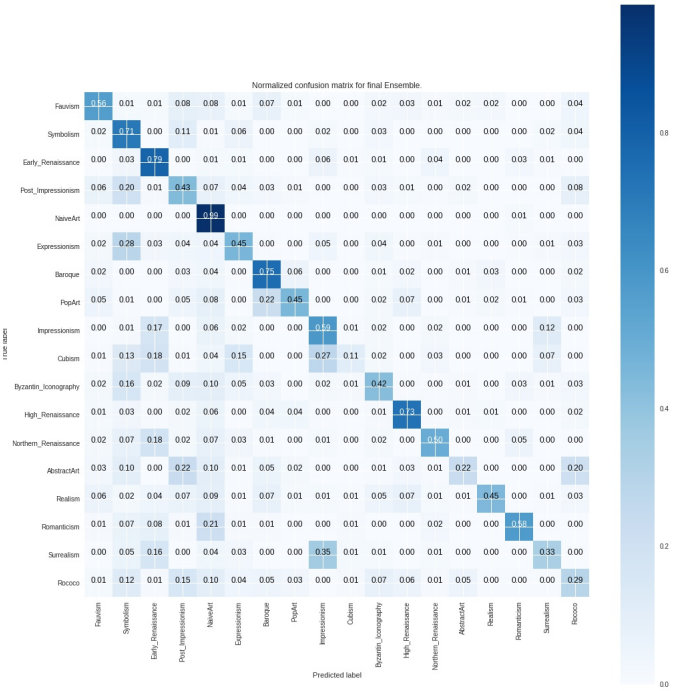

Figure 11: Confusion matrix for the final ensemble.

As we see here, for some classes the prediction accuracy is very high and for some it's very low. Like for Naive art, it's 0.99 and for Cubism, it's 0.11. This seems because of the nature of the content of paintings of the two classes as shown in the **??**. Naive art has simple objects which resembles to the classes on which VGG and other deep models have been trained using ImageNet.

(a) Naive art (b) Cubism

6. Conclusion and Future Work

We see that classification of the paintings into artistic styles is inherently tougher than a normal image classification task because of the highly diverse features they possess. We also see that the deep neural networks are able to learn such features. When trained with a larger dataset and for numerous epochs, such deep networks give good results. In our project, we explored transfer learning using models trained on ImageNet dataset like VGG, ResNet and also tried a couple of custom model architectures to classify the images. Our experiments showed that we can tweak the pre-trained deep models like VGG, ResNet, etc. to obtain better classification results.

Regarding the future directions, we can experiment even more custom models by feeding the outputs of multiple layers at the other different levels into the FC layer. We can try finding better hyper parameters for our models which may result in the improvement of the accuracy of our current models even further. Also, we currently trained our models for either 20 or 40 epochs. We can train them for many more epochs which will probably result in the improvement of the results. Currently, we experimented with VGG and ResNet. In the future, we can explore using other pre-trained models like GoogLeNet, etc.

References

[1] F. Chollet et al. Keras. `https://keras.io`, 2015.

[2] C. Florea, C. Toca, and F. Gieseke. Artistic movement recognition by boosted fusion of color structure and topographic description, 2017.

[3] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks.

[4] F. S. Khan, S. Beigpour, J. van de Weijer, and M. Felsberg. Painting-91: a large scale database for computational painting categorization, Aug 2014.