**Artist Recognition for Fine Art Paintings with Deep Learning**

CS688: Pattern Recognition (Fall 2017)

Bhavika Tekwani, Jaecheon Jeong, Melanie Baker

**Abstract**

In this paper, we present a series of experiments to explore large scale classification of fine-art paintings from the Wikiart dataset. Our objectives are two-fold: first, we want to train classical machine learning models like Random Forests & Support Vector Machines to classify Impressionist paintings. Secondly, we want to train a convolutional neural network (CNN) end-to-end in order to use the deep learning approach that has gained popularity in the image classification domain. As we see later on, identifying paintings by artist is more challenging than detecting faces in images or classifying objects. We also explore model interpretability by exposing unlabeled images (not of paintings) to our trained models and trying to understand how machines perceive & recognize artistic style.

**Dataset**

We used the Wikiart dataset released by the authors of the ICIP2016 paper on deep convolutional networks for fine-art paintings classification [5]. The dataset in its entirety is 27 GB in size and contains over 80,000 images spanning 2108 artists and 27 genres.For our artist classification task, we use only the Impressionism genre (13060 images). Furthermore, to quickly prototype several methods and iterate on them, we limit all our experiments to 15 artists chosen in no particular order. For each artist, we select a set of 40 images. Therefore, our dataset for this task contains 600 images.

The complete list of artists is as follows:

|  |  |
| --- | --- |
| **Artist** | **Class Label** |
| Adam Baltatu | 0 |
| Alfred Sisley | 1 |
| Antoine Blanchard | 2 |
| Arkhip Kuindzhi | 3 |
| Armand Guillaumin | 4 |
| Auguste Rodin | 5 |
| Berthe Morisot | 6 |
| Camille Pissarro | 7 |
| Childe Hassam | 8 |
| Claude Monet | 9 |
| Constantin Artachino | 10 |
| Cornelis Vreedenburgh | 11 |
| Edgar Degas | 12 |
| Edouard Manet | 13 |
| Eugene Boudin | 14 |

**Table 1**: List of Impressionist artists whose paintings were used for training and testing

**Setup**

Hardware:

We did not use any specialized hardware for running our experiments. Available memory ranged from 4 GB - 8 GB on i5 and i7 processors. Although we used a deep learning approach, we did not use GPUs to train our convolutional neural networks.

Software:

We used Python 3 for all our experiments. The libraries we used for machine learning & deep learning include *pandas*, *scikit-learn, numpy, PIL, OpenCV, Tensorflow, Keras & PyTorch*.

**Literature Review**

A literature review of several gave us insight into what features and methods have worked best for similar tasks. Those papers are below with our general findings.

A. Blessing and K. Wen’s paper was written in 2010, which explains why a deep learning network was not part of their architecture. The author’s initially implemented their solution using Naive Bayes to create their features, but found it wasn't sufficient enough and they went to an SVM. The main thing we got out of this was different features (ex. SIFT, HOG, color histograms, etc.) we could focus our algorithm on if we didn't implement our solution as a CNN [1].

In B. Saleh and A. Elgammal’s paper, the authors’ goal was to develop an algorithm that was “able to make aesthetic-related semantic-level judgments, such as predicting a painting’s style, genre,and artist, as well as providing similarity measures optimized based on the knowledge available in the domain of art historical interpretation”. The main takeaway, since our main goal is to identify paintings by artist, is that for Artist Classification, their CNN that learned using the Information-Theoretic Metric Learning (ITML) trained the fastest and had the best accuracy [2].

Y. Hong and J. Kim’s paper was published in 2017 and is very similar in our original approach to the artist classification problem. It talks about each of the different CNN architectures they came up with and their results. With CNNs doing so well in image recognition, it's clear this is the more modern approach and we think if we each developed a CNN we could improve our success on this task [3].

A. Elgammal’s paper shows another option for building a learning architecture if we didn't want to use python and focus more on classifying the more general painting style. This person was able to use the MATLAB image processing libraries and create an SVM that got up to 60% classification on painting style. It's worse than a CNN, but if we had time we could do an experiment [4].

After a review of the above, we conclude that our approach would be unique in how we architect our CNN and the features we could use for the SVMs & Random Forests. We also want to try different learning rates for the deep nets and see the effect on classification accuracy per class.

**Approach**

Our experimental setup is divided into 3 parts - one is machine learning with models like Random Forests, & SVMs, the second is an 18 Layer Convolutional Neural Network built in Tensorflow and Keras, the third is a 2 Layer Convolutional Neural Network built with PyTorch.

The diagram below shows the division of those 3 steps. As is evident from the 3 branches, using deep convolutional nets helps avoid certain steps like feature selection & cross validation. As fine art painting classification is a difficult task, we assume that feature engineering would require some creativity & domain knowledge. We were hoping deep learning would help in this case since no feature engineering is required.



**Figure 1**: Flow chart providing an overview of each architecture/pipeline developed for this project. On the left, there is the diagram for our traditional method and in the middle and right are the diagrams for the 2-layer CNN & ResNet and the 12-layer CNN, respectively.

For all our experiments, we keep the distribution of data constant. We have 600 paintings by 15 artists which we split into a train and test dataset with a 60-40% distribution. Our training data now has 360 paintings and unsequestered test data has 240 paintings.

Feature Engineering & Selection

Impressionist paintings had an interplay of light & shadows, from the spectrum they used colors like yellow, orange, vermilion, crimson, violet, blue and green. These give us some ideas for feature selection.

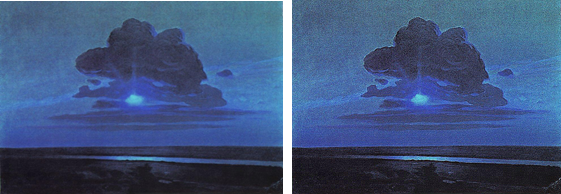
Although these kinds of characters may not relate to grasping artists’ drawing tendency, it may help to recognize the artist throughout their painting training of Support Vector Machine or Random Forest. Total features are consisted of ten features and several features such as blurriness, brightness, and edge are calculated using functions in OpenCV. The rest of traits are implemented by ImageFilter in Python Imaging Library(PIL). Some features such as smooth, edge are, additionally, provided as forms of \*\*\_enhance and \*\*\_enhance\_more, where refined features can support original features.  
 First, brightness means the degree of the painting brightness. One of authors’ outstanding characteristic in Impressionism distinguishing the form by using light and shade. Thus, this brightness can have an important role of distinct feature. This feature can be implemented for using Image.convert(*‘L’*) in PILLOW class. ‘L’ mode means translating a color image to black and white, the library uses the ITU-R 601-2 luma transform: L = R \* 299/1000 + G \* 587/1000 + B \* 114/1000. The default method of converting a greyscale (“L”) or “RGB” image into a bilevel (mode “1”) image uses Floyd-Steinberg dither to approximate the original image luminosity levels.   
 Second, the edge is the statistical value of image filtered with ImageFilter.CONTOUR. This feature is implemented in the cvtColor(img, cv2.COLOR\_BGR2GRAY) and cornerHarris(gray, 2, 3, 0.04) method in OpenCV. cvtColor function converts an input image from one color space to another. In this case, after color image convert gray color image since it is the intersection of two edges, it represents a point in which the directions of these two edges change. Hence, the gradient of the image (in both directions) have a high variation, which can be used to detect it. Consider a grayscale image . We are going to sweep a window (with displacements in the x direction and in the right direction) and will calculate the variation of intensity.  
Third, contour means the outline of a figure or body. i.e. value of image filtered with ImageFilter.CONTOUR. Contours are often obtained from edges, but they are aimed at being object contours. Thus, they need to be closed curves as boundaries. Contours can actually do a bit more than just detect edges. The algorithm does indeed find edges of images, but also puts them in a hierarchy. This means that you can request outer borders of objects detected in your images. Such a thing would not be possible if you only check for edges. The contours is mostly used for object recognition, where the canny edge detector is a more "global" operation. The contour algorithm uses some sort of canny edge detection.  
 Fourth, emboss is the statistical figure of image filtered with ImageFilter.EMBOSS mode. Each pixel of an image is replaced either by a highlight or a shadow, depending on light/dark boundaries on the original image. Low contrast areas are replaced by a gray background. The filtered image will represent the rate of color change at each location of the original image. This filter stamps and carves the active layer or selection, giving it relief with bumps and hollows. Bright areas are raised and dark ones are carved. You can vary the lighting.  
 Fifth, blurriness is a result of calculation of the Laplacian of the image. This feature of paintings may represent Impressionism authors’ painting tendency because they didn’t want to create the object as it is. This item is implemented that Laplacian measure the 2nd derivative of an image. The Laplacian highlights regions of an image containing rapid intensity changes and the Laplacian is often used for edge detection. The assumption here is that if an image contains high variance then there is a wide spread of responses, both edge-like and non-edge like, representative of a normal, in-focus image. But if there is very low variance, then there is a tiny spread of responses, indicating there are very little edges in the image. The more an image is blurred, the less edges there are. This notion of method would be computing the Fast Fourier Transform of the image and then examining the distribution of low and high frequencies — if there are a low amount of high frequencies, then the image can be considered blurry. If the variance falls below a pre-defined threshold, then the image is considered blurry; otherwise, the image is not blurry.  
 Sixth, smooth is the measure of image figure filtered with ImageFilter.SMOOTH mode. it is implemented by openThis characteristic , in image processing, is to create an approximating images that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. In smoothing, the data points of a signal are modified so individual points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal. Smoothing may be used in two important ways that can aid in data analysis (1) by being able to extract more information from the data as long as the assumption of smoothing is reasonable and (2) by being able to provide analyses that are both flexible and robust.(cited from <https://en.wikipedia.org/wiki/Smoothing>)  
 Seventh, detail is the measure of image figure filtered with ImageFilter.SMOOTH mode. Unlike Blurriness, this filter provide the exact figure value. Although the benefit from this feature may be little due to opposite to Impressionism traits, it can be one of considerations for author recognition.



**Image 1**: Original Imag **Image 2**: Brightness  **Image 3**: Edge

****

**Image 4**: Degree of Blurriness : 89.10 **Image 5**: Emboss

****

**Image 6**: Smoothness **Image 7**: Detail

Models

1. Random Forests

Data preprocessing: We resized all images to 224 \* 224.

Feature Selection:

1) Method : RFECV(Recursive Feature Elimination Cross Validation) method

2) Parameters used

-. estimator : RandomForestClassifier(n\_jobs=10, random\_state=0)

-. n\_jobs : 10 (Number of cores to run in parallel while fitting across folds.)

3) Selected Features : [1] = blur, [2] = edge, [4] = edge\_enhance\_more, [5] = contour, [6] = emboss

4) Selected Feature ranking (execution time : 4.88 sec)

|  |  |  |
| --- | --- | --- |
| ID | Feature Name | Ranking |
| 0 | brightness | 3 |
| 1 | blur | 1 |
| 2 | edge | 1 |
| 3 | edge\_enhance | 6 |
| 4 | edge\_enhance\_more | 1 |
| 5 | contour | 1 |
| 6 | emboss | 1 |
| 7 | detail | 4 |
| 8 | smooth | 5 |
| 9 | smooth\_more | 2 |

**Table 2**: Feature Ranking for Random Forest

Tuned Parameters

1) Method : GridSearchCV(Grid Search Cross Validation)

2) Parameter Candidates :

-. "max\_depth": [3,None],

-. "min\_samples\_split": [2,5,10,16],

-. "min\_samples\_leaf": [1,3,9],

-. "bootstrap":[True, False],

-. "criterion":["gini","entropy"]}

3) Best parameters set found on development set

-. {'bootstrap': False, 'criterion': 'gini', 'max\_depth': None, 'min\_samples\_leaf': 1,

'min\_samples\_split': 16}

4) Execution time : 3.47 sec

Prediction Result

1) Cross Validation method : StratifiedKFold

-. folds : 5

2) Train Datasets : train - X\_train, y\_train, predict - X\_test, y\_test

3) Classifier : ExtraTreesClassifier with tuned parameters

-. Tuned parameters

{'bootstrap': False, 'criterion': 'gini', 'max\_depth': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 16}

4) Result with all features

-. Accuracy : 25.0

-. Recall : 26.22

-. Precision : 24.12

-. Execution time : 0.68 sec

5) Result with selected features

-. Accuracy : 33.75

-. Recall : 35.05

-. Precision : 35.71

-. Execution time : 0.69 sec

6) Accuracy per artists - See Table 7

1. Support Vector Machines

Data preprocessing: We resized all images to 32 \* 32.

Feature Selection:

1) Method: Recursive Feature Elimination Cross Validation (RFE CV)

2) Parameters used

-. estimator : svm.SVC(C=0.01)

-. n\_jobs : 10 (Number of cores to run in parallel while fitting across folds.)

3) Selected Feature ranking

|  |  |  |
| --- | --- | --- |
| ID | Feature Name | Ranking |
| 0 | brightness | 1 |
| 1 | blur | 1 |
| 2 | edge | 1 |
| 3 | edge\_enhance | 1 |
| 4 | edge\_enhance\_more | 1 |
| 5 | contour | 1 |
| 6 | emboss | 1 |
| 7 | detail | 1 |
| 8 | smooth | 1 |
| 9 | smooth\_more | 1 |

**Table 3**: Feature Ranking for SVM

Tuned Parameters

1) Method : GridSearchCV(Grid Search Cross Validation)

2) Datasets : X\_train, y\_train

3) Best parameters set found on development set is that kernel value is linear and C is 0.01.

It takes 23722.60 sec(

Prediction Result

1) Cross Validation method : StratifiedKFold (folds : 3)

2) Datasets : train datasets : X\_train, y\_train, prediction datasets : X\_test, y\_test

3) Classifier : svm.SVC with tuned parameters

-. Tuned paramers : {'kernel': linear, 'C': 0.01}

4) Prediction result with all features

-. Accuracy : 19.16%

-. Recall : 19.16%

-. Precision : 14.98%

-. Execution time : 1652.10 sec

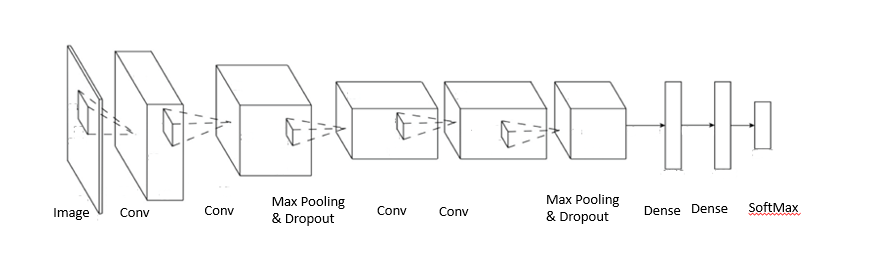
1. CNN - 12 Layer

In order to compare and possibly improve upon our 2-layer CNN accuracy, we wanted to experiment with a more complex network similar to ones we’ve seen used in research for classifying large data sets. Therefore, we utilized a pre-built 12 layer CNN architecture using Keras that had performed classification well for the CIFAR-10 dataset. The network architecture is detailed as such:

|  |
| --- |
| **Layer** |
| Conv2D [out=32, kernel = 3, stride = 1, padding = 0, act=relu] |
| Conv2D [out=32, kernel=3, stride=1, padding=0, act=relu] |
| MaxPool2D [kernel =2, stride = 2] |
| Dropout[rate =.25] |
| Conv2D [out=64, kernel = 3, stride = 1, padding = 0, act=relu] |
| Conv2D [out=64, kernel = 3, stride = 1, padding = 0, act=relu] |
| MaxPool2D [kernel =2, stride = 2] |
| Dropout[rate =.25] |
| Flatten[ ] |
| Linear [out = 512, act=relu] |
| Dropout[rate =.5] |
| Linear [out = 15, act=softmax] |

**Table 4**: 12-layer CNN architecture

Visually, this neural net can be represented as shown below. We changed the input to be 210 x 140 sized images of the paintings and the network was trained using various learning rates of .01, .001, and .0001, a batch size of 15 and initially 100 epochs. The loss function used was cross-entropy loss. The .0001 learning rate produced the best model. I varied the number of epochs as well from 80, 100 and 125, but 100 proved best, achieving ~78% accuracy on the training set. Training time using the above parameters took ~3.5 hours.



**Figure 2**: 12-layer CNN architecture diagram

1. CNN - 2 Layer

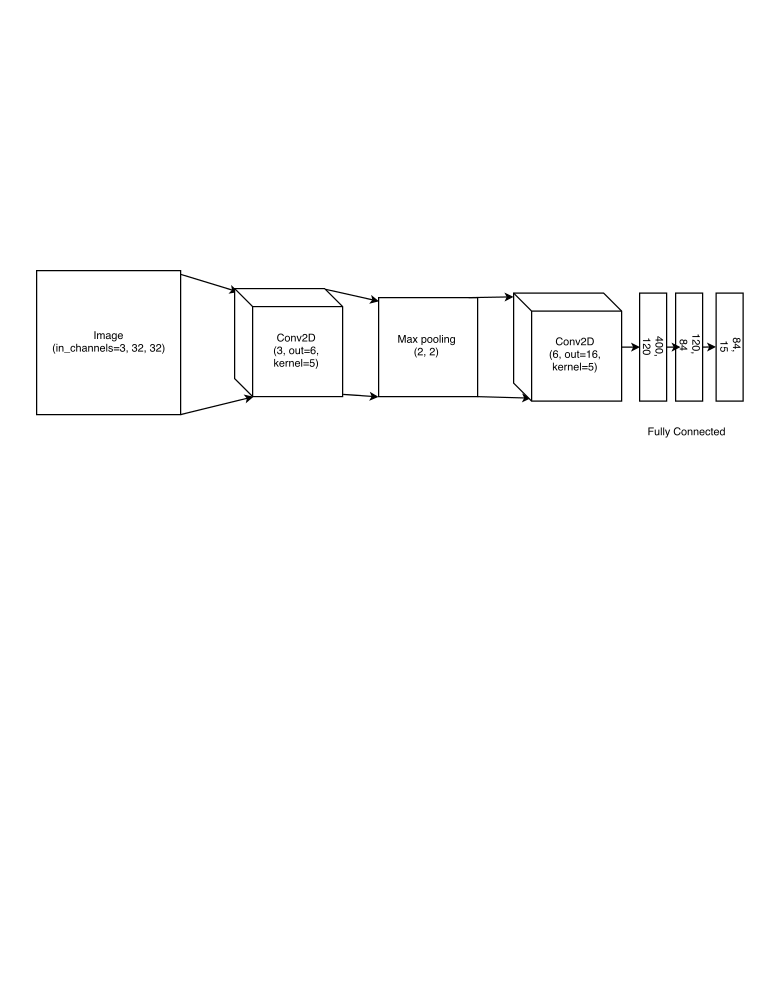
We implemented a simple 2 layer convolutional neural network in PyTorch and trained it with different learning rates. Every image used as input to this network was resized to 32 \* 32. The network architecture is detailed below:

|  |
| --- |
| **Layer** |
| Conv2D [in=3, out=6, kernel=5, stride = 1, padding = 0] |
| MaxPool2D [kernel =2, stride = 2] |
| Conv2D [in=6, out=16, kernel=5, stride=1, padding=0] |
| Linear [in = 400, out = 120] |
| Linear [in = 120, out = 84] |
| Linear [in = 84, out = 15] |

**Table 5**: 2-layer CNN architecture

Visually, this neural net can be represented as shown below. We use this 2 layer CNN as a baseline to measure other CNN architectures. The last fully connected layer emits predictions for 15 classes.

The loss function used is Cross Entropy Loss. We tune the weights in the CNN using the SGD optimizer with varying learning rates but keeping the momentum constant at 0.9. The learning rates were varied from 1e-5 to 1e-1, but the optimal learning rate for this network was found to be 0.001. Since PyTorch allows data to be processed in batches, we set the batch size to 4 since our training dataset was small & the network wasn’t too deep. The network was trained over 100 epochs which was excessive since we observed that the loss increases instead of decreasing after 50-60 epochs. Again, this can be attributed to the size of the training data.



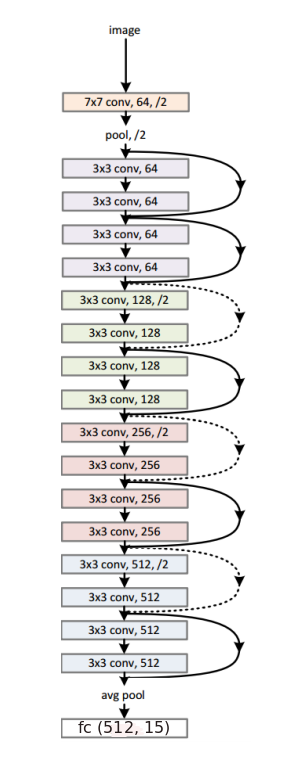
**Figure 3**: 2-layer CNN architecture diagram

1. ResNet-18 (transfer learning)

Since it is quite uncommon to train complex deep architectures from scratch, we decided to experiment with some popular architectures that have been known to perform well on image classification problems. One such architecture is ResNet which has several variants that performed well on the ImageNet dataset that contained 1000 classes. Of the architectures available, we use the simplest one - ResNet-18. It has 18 convolutional layers & several pooling & dropout layers. For our task, we simply modify the last fully connected layer to give us the predictions for 15 classes instead of 1000.

ResNet-18 needs an input size of 224 \* 224, so we resized each image to these dimensions in order to make the network reusable. Other than the resizing, we do not perform any transformations on the original paintings.

The modified architecture with a change in the last fully connected layer is shown below.



**Figure 4**: ResNet-18 architecture diagram

1. ResNet-18 (fine tuned)

Another approach we tried was fine-tuning the ResNet-18 model to adjust the weights in each layer for our specific task of classifying paintings. To do this, we used the pretrained model in PyTorch and retrained it with different learning rates again varying them in the range 1e-5 to 1e-1.

The batch size was set to 4 & we trained the model over 100 epochs. As observed with the 2 layer CNN, the loss tends to oscillate between 0.007 and 0.004 once we reach 40-50 epochs. The best learning rate was found to be 0.001.

Model Interpretability

Deep learning models are often hard to debug & tend to appear as black boxes. For our task of fine art classification, we wanted to understand if some of the ways humans perceive art and objects depicted in art are reflected in deep learning models. To do this, we selected a few images that were not works of art & did not have any artist labels present in our training data. Once we trained our models, we used each model and asked it to predict an artist for one of these random images.

Below, we present the new test images (not actual paintings) and the labels our model predicted for them. What we hope to observe here is not accuracy but rather the similarity between the test image & some of the training images for the predicted artist. This similarity can be in terms of the subject of the painting (human, nature, physical object, animals, etc.), overall color, tone, etc.

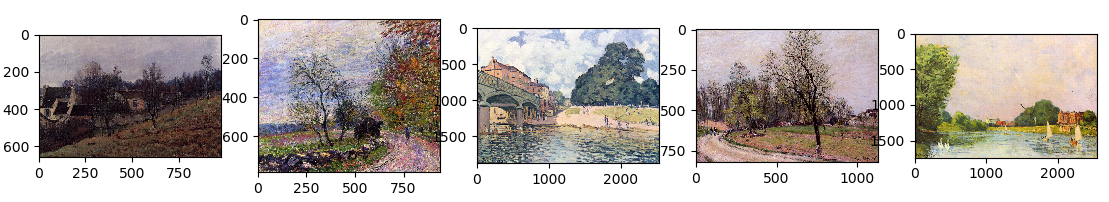
Random Forest

Case 1: Prediction result for real image chosen from Google(title : landscape.jpg)

input



**Image 9**: Google image of nature-trees



**Image 10:** Paintings by Alfred Sisley used for training

Input 1:



**Image 11:** Google image of baby playing in grass

**CNN - 2 Layer**

Predicted artist: Édouard Manet

**Image 12:** Paintings by Edouard Manet used for training

**ResNet18 fine-tuned**

Predicted artist: Armand Guillaumin



**Image 13:** Paintings by Armand Guillaumin used for training

Input 2:



**Image 14:** Google image of nature-mountains

**CNN - 2 Layer**

Predicted artist: Arkhip Kuindzhi



**Image 15:** Paintings by Arkhip Kuindzhi used for training

**ResNet-18 fine-tuned**

Predicted artist: Camille Pissarro  


**Image 16:** Paintings by Camille Pissarro used for training

Input 3:



**Image 17:** Google image of boat on water

**CNN - 2 Layer**

Predicted artist: Claude Monet



**Image 18:** Paintings by Claude Monet used for training

**ResNet-18 fine-tuned**

Predicted artist: Cornelis Vreedenburgh



**Image 19:** Paintings by Cornelis Vreedenburgh used for training

As we observe from the demo inputs & paintings we trained on, both models seem to do well when it comes to understanding the subject of the painting - a human in input 1, nature/scenery in input 2 and

a water body & boat in input 3.

Input 1:



**Image 20:** Google image of nature-waterfall

**CNN - 12 Layer**

Predicted artist: Camille Pissarro



**Image 21:** Paintings by Camille Pissarro used for training

Input 2:



**Image 22:** Google image of a city landscape

**CNN - 12 Layer**

Predicted artist: Childe Hassam



**Image 23:**Paintings by Childe Hassam used for training

Input 3:



**Image 24:** Google image of baby in grass

**CNN - 12 Layer**

Predicted artist: Claude Monet



**Image 25:** Paintings by Claude Monet used for training

Looking at the demo inputs and paintings we trained the 12-layer CNN on, the model seemed able to recognize images based on similar subject matter (i.e a city landscape, nature) and similar spatial aspects (i.e. water or greenery on bottom) of the image.

**Results**

We summarize the results for each model in Table 6.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** |
| Random Forest | 33.75% | 35.71% | 35.01% |
| Support Vector Machine | 19.16% | 19.16% | 14.98% |
| CNN - 2 Layer | 8.75% | 76.56% | 8.75% |
| CNN - 18 Layers | 32% | 39% | 32% |
| ResNet-18 (transfer learning) | 7.9% | 63.7% | 7.9% |
| ResNet-18 fine-tuned | 10.83% | 53.26% | 10.83% |

**Table 6**: Accuracy, Precision, and Recall for each Model

The table below shows classwise accuracy for artists listed in Table 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| Random Forest | 30 | 50 | 6.6 | 40 | 16 | 46 | 6.6 | 45 | 33 | 46 | 0 | 41 | 10 | 0 | 22 |
| SVM | 0 | 0 | 6.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5.2 | 0 | 0 | 0 |
| CNN - 2 Layer | 8 | 30 | 25 | 6 | 6 | 25 | 4 | 10 | 3 | 12 | 0 | 0 | 0 | 0 | 0 |
| CNN - 18 Layers | 19 | 38 | 100 | 6 | 13 | 38 | 19 | 56 | 31 | 19 | 25 | 0 | 56 | 38 | 44 |
| ResNet-18 (transfer learning) | 25 | 33 | 33 | 20 | 33 | 43 | 25 | 0 | 21 | 33 | 31 | 25 | 20 | 20 | 25 |
| ResNet-18 fine-tuned | 34 | 25 | 50 | 43 | 25 | 25 | 25 | 29 | 37 | 0 | 50 | 18 | 31 | 25 | 25 |

**Table 7**: Classwise accuracy for each Model

**Conclusions**

The highest accuracy we were able to achieve was 38% using random forest, with the next being 32% using the 12-layer CNN, and 7% using the 2-layer CNN. This is far away from the current state-of-the-art and given our results, we’ve concluded that artist recognition is an extremely difficult problem in machine learning that may require more data preparation than we used. To improve our model, we could have used paintings from only well known artists with unique styles or increased the number of training examples per artist. However, the latter could risk further model confusion.

**Further Work**

While we could not achieve very good results on our existing dataset, we believe increasing the amount of training data could help us get better results with deep learning. The current size of the dataset makes it better suited for classical machine learning approaches & the results reflect that. However, another challenge we faced is that not all of the artists in our dataset have very distinct & evolved artistic styles. If we chose influential artists from the whole Wikiart dataset & not just the Impressionism genre, there might be more room for the model to learn to identify those artists. Additionally, framing this as a hierarchical classification problem where we first predict the genre & then the artist within that genre would be very interesting. With better infrastructure (we did not use GPUs), we could process the whole dataset & implement some image transformations to observe the effect of those changes on accuracy metrics.

**References**

[1] A. Blessing and K. Wen, Using machine learning for identification of art paintings,

Tech. Rep., 2010.

[2] B. Saleh and A. Elgammal, Large-scale classification of fine-art paintings: Learning

the right metric on the right feature, ArXiv Prepr. ArXiv150500855, 2015.

[3] Y. Hong and J. Kim, Art Painting Identification using Convolutional Neural

Network, Int. J. Appl. Eng. Res., vol. 12, no. 4, pp. 532–539, 2017.

[4] Creating Computer Vision and Machine Learning Algorithms That Can Analyze

Works of Art - MATLAB & Simulink. [Online]. Available:

<https://www.mathworks.com/company/newsletters/articles/creating-computer-vision-and-machine-learning-algorithms-that-can-analyze-works-of-art.html>. [Accessed: 18-Sep-2017].

[5] ICIP2016, Wei Ren Tan, Chee Seng Chan, Hernan E. Aguirre and Kiyoshi Tanaka, Ceci n'est pas une pipe: {A} deep convolutional network for fine-art paintings classification. <http://web.fsktm.um.edu.my/~cschan/doc/ICIP2016.pdf>