Artistic Movement Recognition using Deep CNNs

Yangyang Yu, Olivier Jin, Daniel Hsu
Stanford University
[yyu10, ojin, dwhsu]@stanford.edu

Abstract

In this project we investigate using deep convolutional neural networks to classify artwork into artistic styles. Previous efforts have not used deep learning, and our goal is to design different CNN architectures as well explore transfer learning for this problem. Our custom hybrid VGG and Inception architecture obtained an accuracy of 31.2%, and our best result from transfer learning the InceptionV3 architecture obtained an accuracy of 56.6%, beating the original publications baseline of 56%.

1. Introduction

The digitization of artwork opens up many new possibilities in computing and identification. Classifying artwork is a non-trivial task, given the subjectivity and nuances separating different artistic categories. By correctly categorizing a piece of art in its proper movement, researchers can gain contextual information regarding that piece of art and its relation to surrounding works.

We approached this task by utilizing the Pandora18k dataset, a collection of 18,038 pieces of artwork across 18 different artistic categories. The movements range from Byzantine iconography to modern pop art, and include an even distribution in each category. The dataset ensures that only the relevant part of the artwork is shown in each image.

Our approaches rely on end-to-end deep convolutional neural networks (CNN) in to both extract a variety of features (potentially color palette, brush techniques, painting subject, etc) and identify the proper artistic category. We implemented one smaller network inspired by cells in Inception and VGG and trained from scratch for an efficient model. We also experimented with transfer learning on Inception by retraining the dense layers and the last convolution cell to achieve the best performance.

2. Related Work

Automatic artwork categorization has been an active research area since more than a decade ago.[9] Various techniques were applied to digitized artworks. Earlier approaches usually focus on manually analyzing the paintings to find relevant features and applying traditional machine learning classifiers such as Bayesian and SVM.[2][7][9][10] These methods were able to achieve relatively high accuracy that ranges from 62% to 91%. However, these benchmarks were generated on smaller datasets with fewer than 5000 paintings and no more than 13 classes. As we can see in Table 1, in general, the accuracy decreases as the dataset size and the class number increase, which is expected since a bigger dataset could imply more diverse features that are harder to define manually while a larger number of classes makes the problem fundamentally harder.

Table 1: Related work that used predefined features for artwork classification.

Research	**Classes**	**Dataset**	**RR**
Gunsel et al.[9]	3	107	91.66%
Arora and Elgammal[2]	7	490	65.4%
Khan et al.[10]	13	2338	62%
Condorovici et al.[7]	8	4119	72.24%

To be able to achieve better performance on larger datasets, recent research started to use deep neural networks for feature extraction. Bar et al.[3] collected a dataset of 47,724 paintings divided into 27 classes by crawling wikiart.org. They were able to achieve a recognition rate of 43% by combining a learned image descriptor PiCoDes[4] and features extracted by a deep CNN originally trained on ImageNet. Florea et al.[8] collected the Pandora18K dataset that consists of 18,038 paintings in 18 classes. The best recognition rate they achieved was 56% using transfer learning based an AlexNet[12] originally trained on ImageNet.

We chose to work with the Pandora18K dataset because the labels were more carefully examined and verified by experts.

Our work is also inspired by Inception[16] and VGG[14], which are both deep CNNs that performed very well in the ImageNet challenges[13]. The ImageNet challenges are a set of vision based tasks on the ImageNet

dataset.

3. Dataset and Features

As mentioned in earlier sections, we chose to use the Pandora18K dataset with 18,038 paintings categorized into 18 classes. Figure 1 shows some sample images from the dataset. As we can see, the dataset contains paintings in a wide range of topics, such as scenery, portrait, still life, etc.

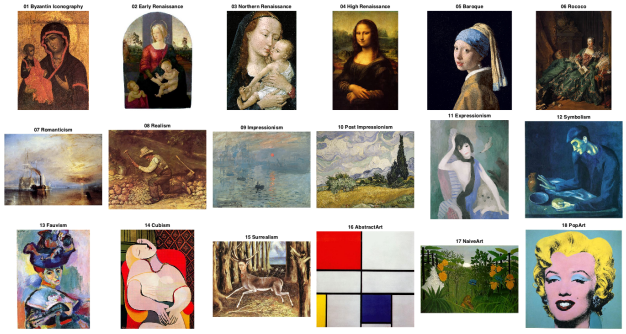


Figure 1: Pandora18K Dataset.

One restriction of most CNNs is that the inputs need to be of the same size since the fully-connected layers usually have a fixed number of parameters. However, the sizes of the images in our dataset are not constant. Therefore we need to resize the images to the same size by scaling and cropping. To determine the appropriate input size, we profiled the image sizes in our dataset. The results are presented below.

The scattered plot in Figure 2 gives an overview of the image sizes. Each data point represents an image in the dataset whose coordinates shows its width and height. As we can see, the width and height of most images are below 2000 pixels while only a few images have a very high resolution of over 5000 pixels in width and height.

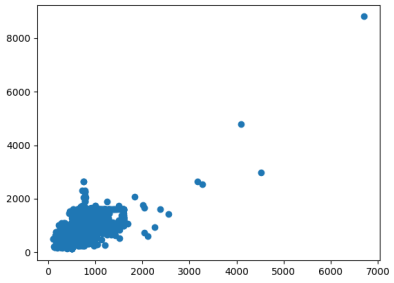
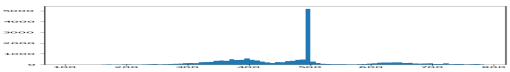


Figure 2: Image size scatter plot.

To better understand the distribution of image sizes, we generated the histogram of the images' width, height and aspect ratio respectively.

(a) Image width histogram of all 18,038 images.



(b) Image width histogram of the lower 17,000 images

(c) Image width histogram of the higher 1,038 images.

Figure 3: Image width distribution.

Figure 3 shows the width distribution. In addition to the overall histogram, we also plotted the lower 17,000 images separately to present more details. From the histogram, we can clearly observe that the majority of images have a width of around 500 pixels. The height distribution is very similar to the width distribution. The exact histograms are presented in Figure 4.

(a) Image height histogram of all 18,038 images.

(b) Image height histogram of the lower 17,000 images

(c) Image height histogram of the higher 1,038 images.

Figure 4: Image height distribution.

The histogram in Figure 5 shows that the majority of the images in our dataset have an aspect ratio of around 1:1. With this profiling, we found that the majority of the images in our dataset has an aspect ratio of around 1:1 and width/height of around 500 pixels. Therefore we decided to resize all our image inputs to 500x500x3 pixels.

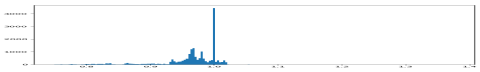


Figure 5: Image aspect ratio distribution.

3.1. Framework

We used the Keras API[6] with Tensorflow[1] as the backend to facilitate our network implementation. This is

a high level wrapper for Tensorflow that makes fast iterative development with our models possible. One major reason we chose to use Keras was its ability to stream in images using a data generator. The Keras data generators allow us to efficiently stream in images from a directory and resize them in real-time when training and testing. This saves us from having to spend time preprocessing our data and resizing images on our own. Furthermore, it allows us to perform our experiments with a greatly reduced memory footprint, since Keras builds tensors from batches of images it streams in, instead of building a prohibitively large tensor that contains all of our training samples. We referenced a Keras tutorial on transfer learning[5] when we set up our framework.

4. Custom CNN Architecture

The primary goal in designing a custom architecture was to achieve a high level of accuracy while minimizing resources. Our Google Cloud instance was constrained to 30GB and, rather than increasing the maximum capacity, we used this limit as a challenge to design a CNN which was both high-performing and lightweight. This decision greatly reduced our training time, allowing us to experiment with a wider range of configurations and hyperparameters.

Our final design is a hybrid network consisting of a compressed VGG-style layer, a single Inception-style layer, and a dense layer with Softmax output. The specific details are as follows:

Table 2: Hybrid CNN Architecture

Type	**Filters**	**Kernel**	**Strides**	**Activ.**
VGG-Style Sequence				
Input	-	-	-	-
Conv2D	8	(3x3)	(1x1)	Relu
Conv2D	16	(3x3)	(1x1)	Relu
Conv2D	32	(3x3)	(1x1)	Relu
Conv2D	64	(3x3)	(1x1)	Relu
MaxPool2D	-	(2x2)	(2x2)	-
Inception-style Branch				
Conv2D_{1x1}	32	(1x1)	(1x1)	Relu
Conv2D_{3x3}	16	(1x1)	(1x1)	Relu
Conv2D_{3x3}	64	(3x3)	(1x1)	Relu
Conv2D_{3x3}	32	(1x1)	(1x1)	Relu
Conv2D_{5x5}	8	(1x1)	(1x1)	Relu
Conv2D_{5x5}	32	(5x5)	(1x1)	Relu
Conv2D_{5x5}	32	(1x1)	(1x1)	Relu
Inception-style Concatenation				
Dense	256	-	-	Relu
Dense	64	-	-	Relu
Output	18	-	-	Softmax

Our final model was initially based on a simplified VGG architecture. VGG was attractive since it utilized increasing numbers of filters to extract different levels of information. However, VGG also required massive amounts of memory to train a single forward pass, which we could not accomplish given our limited compute resources. As a result, we implemented a "VGG-lite" design with simplified conv2D layers (32-32-64-64-128-128-256-256) filters and 2D max pooling layers between each pair of conv layers. This is in stark contrast to the bulky architecture specified by Simonyan and Zisserman. Using this modified configuration, we could achieve a reasonable accuracy of 31.2%.

Treating our VGG-lite model as a baseline, we experimented with enhancing our model using elements from other models as well. We found that adding an Inception-style layer resulted in the best outcomes, with our accuracy increasing to 35.8% overall. The strengths in this approach were the architectures ability to extract and concatenate multi-level features in parallel. In this case, our convolutional layer modeled the Inception 3(a) layer using 64 1x1, 128 3x3, and 32 5x5 convolutions along with corresponding reduction layers.

Both of our networks were trained using an Adam optimizer[11] with randomized learning rate, beta values, and other hyperparameters. We found that, rather than comprehensively sweeping through all possible combinations, a randomized approach would yield a greater range of potential accuracies.

5. Transfer Learning

In order to achieve high accuracy, we tried to leverage pre-trained networks by performing transfer learning on an InceptionV3 network trained on ImageNet. The first step of our transfer learning is to retrain the fully connected layers so that the network can categorize inputs into one of our 18 classes instead of the 1000 classes defined by the ImageNet challenge.

It is worth noticing that since we are freezing all the convolutional layers, the forward propagation through the convolution layers will remain the same during the entire training process. Therefore we can pre-generate the features extracted by the last convolution layer and feed these features to the fully connected layers during training instead of running each input image through the convolutional layers for each epoch. This way, we are able to accelerate the training process significantly.

We connected output features of the convolutional layers to two sets of dropout layer and fully connected layer pairs. We used the categorical softmax-cross-entropy loss function and Adam optimizer. We also used the dropout regularization technique[15] to reduce overfitting, since the Inception model is very complex which makes the model prone to overfitting. We performed a randomized hyper-

parameter search on the number of nodes of the first fully connected layer and on the dropout probabilities.

6. Fine-tuning Final Layer

In a similar fashion to the transfer learning technique described in the previous part, we expand the number of levels we train to incorporate both fully connected and convolutional layers. In this part, we fine tune the last convolutional module along with the last fully connected layers. This allows us to tune the weights of our model to fit our own problem better, since we force the last convolutional module to take on weights that classify our images better.

We set up our model by freezing all but the final fully connected layers, and the last convolutional module. Because we froze the vast majority of the Inception network, we can avoid running our entire network during the training phase because we know that training wont affect the frozen weights. What we do instead, is generate the features that we will feed into our trainable network. These features are the output of the frozen network.

Now with these features, we can feed these into our trainable model to fine tune its weights to our problem. One important thing to note is that we can reuse the results of our previous transfer learning experiment by initializing the fully connected layer weights with the weights we obtained in the previous part. But we still update and optimize these weights because when combined with a trainable convolutional module, its likely that a different combination of weights will yield even better results. Since we are reusing the fully connected layers trained from the last step, the hyperparameters in the fully connected layers, i.e. second fully connected layer size and dropout probability, are already determined. The only hyperparameter to be optimized is the learning rate. To avoid completely overwriting the previously learned features, we used the Adam optimizer with a very low learning rate. In addition to the hyperparameters, we also need to choose the appropriate initial weights. Weights that overfit the training set too much are not suitable since the training accuracy might already be so high that the optimizer would not be able to further optimize the model.

7. Results

7.1. Custom CNN Architecture

As stated before, the final validation accuracy of our hybrid network was 35.8%. This is more than 4% better than our previous VGG-lite architecture, and approximately 6x that of a pure random baseline (5.55%).

While not as accurate as the transfer learning and fine-tuning methods discussed in the following section, our custom network requires a much shorter time when training. For example, our hybrid model required only 300 seconds per epoch, while SqueezeNet (the next fastest network we tested) required 1200 seconds per epoch. Given this improvement, we were able to tune our hyperparameters much more effectively and efficiently, as well as iterate over more epochs in a shorter period of time. Another benefit of our model is its low memory requirement, requiring only 50k trainable parameters for all convolutional layers. In comparison, InceptionV3 has more than 21.8 million parameters, 21.7 million of which are trainable. Given these architectures, our hybrid model performs more than half as well as transfer learning using InceptionV3, while using less than 1% of the convolutional parameters.

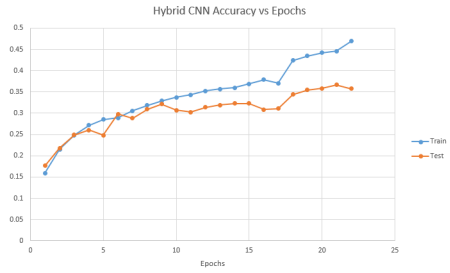


Figure 6: Custom CNN Accuracy vs.Epoch

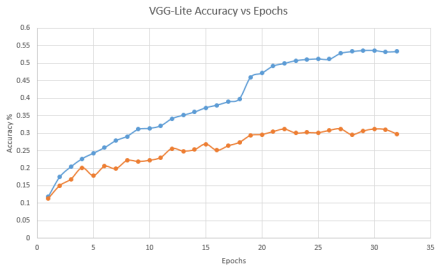


Figure 7: VGG-Lite Accuracy vs.Epoch

7.2. Transfer Learning

What we see in from transfer learning is a substantial improvement of 90.6% training accuracy , 57.84% validation accuracy and test accuracy of 56.6%. Our optimal hyperparameters were a learning rate of $1.95e^{-5}$, a dense layer node number of 650, and a dropout probability of 0.7. A plot of the training and validation accuracy across epochs is shown in Fig 8.

As we can see in the plot, this specific model overfits the training set significantly even with a high dropout probability, which could be caused by the large number of nodes in the dense layer, since the large dense layer increases the complexity of the model. We also trained some models with around 300 nodes in the first dense layer during hyperparameter search. Those models did not overfit as much but also produced slightly lower validation accuracy.

Our transfer learning efforts demonstrate how we can

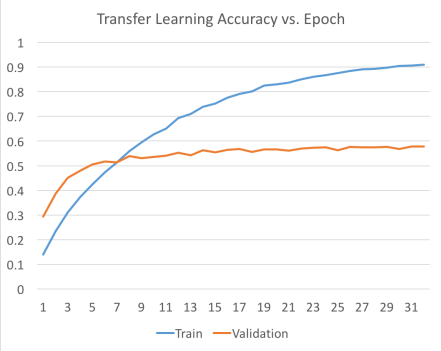


Figure 8: Transfer Learning Accuracy vs.Epoch

harness the power of a pretrained networks architecture and weights and adopt it to our task. This large boost in performance is expected, considering the sheer complexity of the InceptionV3 model specifically designed for image classification purposes. Given this models demonstrated success in image classification, we validate transfer learnings fundamental idea of transferring a pretrained networks feature extraction ability into a separate but related problem space.

7.3. Fine-tuning Final Layer

As mentioned above, before fine-tuning the last convolutional module, we need to initialize the model with weights learned from transfer learning. The model with the highest validation accuracy was not very suitable as the initial weights since it already achieved a very high training rate. Therefore, we used a model that demonstrated less overfitting compared to our initial weights. Figure 9 shows the training process of this model. It reached a final validation accuracy of 55.53%.

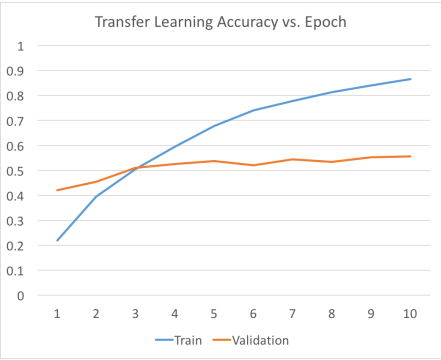


Figure 9: Transfer Learning Accuracy vs.Epoch

Figure 10 illustrates the fine-tuning process. As we can see, the accuracy did increase slightly although it did not end up being better than the best validation accuracy we achieved from dense layer transfer learning. Another point to notice is that the model did not overfit as much. It might be explained by the more effective feature extraction with the fine-tuned convolutional module. The fine-tuned convolutional module would suit our specific task better.

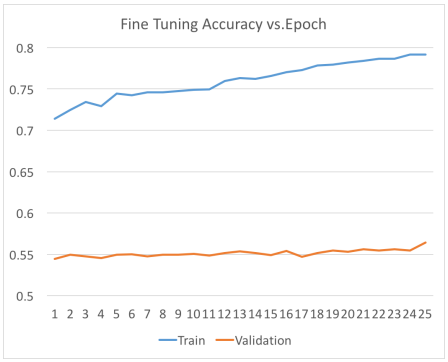


Figure 10: Fine Tuning Accuracy vs.Epoch

Due to the long training time caused by the very low learning rate, and the dependency on a trained initialization, we did not have the chance to examine the entire fine-tuning training landscape thoroughly. We do expect that with more time for training and more computational resources to speed up this process, unfreezing more layers like we did with the convolutional module should yield better and better results. As we allow more layers to be trained on our dataset, we train the weights in the architecture for our problem. To take this to the extreme would be to unfreeze all the layers, and retrain the entire network from scratch. This is computationally unfeasible for our purposes in this project, and so future work will involve finding an optimal balance between training more layers and handling the increasing computational load.

8. Discussion

We started with our own custom architecture in order to experiment with emulating different architectures and observing which ones worked well for our problem. What we found was that a combination of VGG and InceptionV3 worked extremely well. Our hybrid model of these two architectures was enough to obtain a test accuracy of 35.8 percent. This already came close to the baseline presented in Florea et al.[8], and demonstrated the usefulness of deep learning in artistic style classification. What was extremely impressive was that even with a relatively simple architecture of layer-layer-layer-etc, deep learning was able to extract and learn from features of various artistic styles.

Once we knew that InceptionV3 worked well, we pivoted to transfer learning in order to offload some of the heavy-lifting to predefined architectures and weights. We were able to obtain a 56.66% test accuracy from this technique by only tuning the fully connected layer at the end of the network. With more time and compute clusters, we

have no doubt that training more and more of the architecture will enable us to achieve even better performance from our transfer learning model.

What we can see from these experiments is that while the custom architecture performs well, the best results come from transfer learning. The theoretical basis for why transfer learning should drastically improve our accuracy comes from the idea that many image classification tasks, datasets, and image features are inherently similar. Once a deep learning model has learned how to extract meaningful features from one image dataset (in our case, pretrained on ImageNet), it requires substantially less time to adapt it to our problem compared to developing and training a new model from scratch.

Despite the impressive performance from our three approaches, there are still several important qualitative observations. In an effort to visualize where our model was struggling, we construct a confusion matrix from our best performing model in Figure 11.

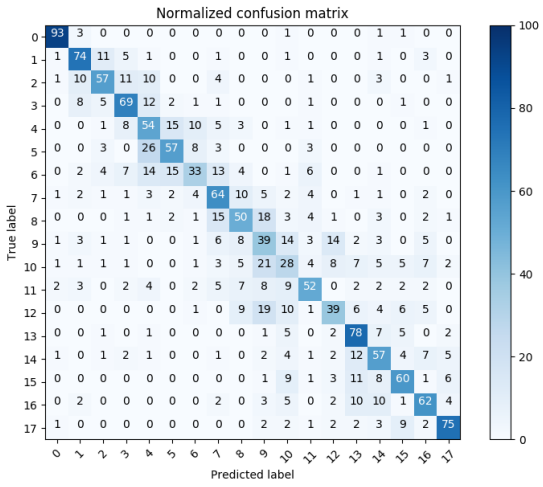


Figure 11: Confusion Matrix

We clearly see from the heat map visualization that the main diagonal has the highest values, indicating that our model generally performs very well. Some classes, such as the Byzantine artistic style, perform extremely well with very little confusion with its neighbors. Upon inspection, this isn't all that surprising. As shown in Figure 1, Byzantine pieces are noticeably different from other styles.

It is also likely that there was an unexpected, though still advantageous, feature of Byzantine art that allowed us to classify these pieces with such accuracy. From a historical perspective, Byzantine art is the oldest out of all our artistic categories, flourishing for nearly nine centuries under the Eastern Roman Empire. As such, many of these pieces are well over a thousand years old. Despite efforts in artistic community to preserve these pieces, they do bear marks of age. Many of these pieces are damaged and faded, which probably presented itself as a feature that our model picked up on.

Looking at our confusion matrix again, we can see that other classes are not so easily differentiated like the Byzantine class. Classes such as Impressionism and Post-Impressionism have a higher error rate as shown in the heatmap. However given how visually similar these two classes of images are, this is not surprising.

9. Conclusion

In this project we experimented with two main techniques for classifying artwork into different artistic styles. Our custom CNN architecture drew inspiration from VGG and Inception, and achieved an accuracy of 35.8%. We then moved onto more complex architectures by means of transfer learning, retraining on the last fully connected layers and with and without retraining on the last convolutional module. The best result from transfer learning achieved a test accuracy of 56.6%. Given these results, we can conclude that deep learning and image classification with regard to artistic style is a very suitable approach.

In the future we plan on exploring more with both of our techniques. For our custom model, we will add on more layers in order for our model to obtain a deeper understanding of our data, and we also plan on incorporating additional architectures into our hybrid model. Architectures such as SqueezeNet and AlexNet are worth looking into for our model. For transfer learning, we plan on retraining more than just the last convolutional module and the last fully connected module. With more time for training, it is very likely that fine tuning more layers from InceptionV3 will allow our model to better handle our specific problem, instead of general image recognition problems. Lastly, we plan on pursuing more data augmentation techniques. With only around a thousand images per class, our dataset is still relatively small. One way for us to make the most out of this data in the future is to apply data augmentation to enhance our training process.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] R. S. Arora and A. Elgammal. Towards automated classification of fine-art painting style: A comparative study. In

Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), pages 3541–3544, Nov 2012.

[3] Y. Bar, N. Levy, and L. Wolf. *Classification of Artistic Styles Using Binarized Features Derived from a Deep Neural Network*, pages 71–84. Springer International Publishing, Cham, 2015.

[4] A. Bergamo, L. Torresani, and A. W. Fitzgibbon. Picodes: Learning a compact code for novel-category recognition. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2088–2096. Curran Associates, Inc., 2011.

[5] F. Chollet. Building powerful image classification models using very little data. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html, June 2016.

[6] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[7] R. G. Condorovici, C. Florea, and C. Vertan. Automatically classifying paintings with perceptual inspired descriptors. *J. Vis. Comun. Image Represent.*, 26(C):222–230, Jan. 2015.

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

[9] B. Gunsel, S. Sariel, and O. Icoglu. Content-based access to art paintings. In *IEEE International Conference on Image Processing 2005*, volume 2, pages II–558–61, Sept 2005.

[10] F. S. Khan, S. Beigpour, J. Weijer, and M. Felsberg. Painting-91: A large scale database for computational painting categorization. *Mach. Vision Appl.*, 25(6):1385–1397, Aug. 2014.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. 12 2014.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. 12 2015.