Understanding Visual Art with CNNs

Michael Baumer
Stanford University
Department of Physics
mbaumer@stanford.edu

Derek Chen
Independent Researcher
https://feature.engineering
derekchen14@gmail.com

Abstract

Previous work in computational visual art classification has relied on fixed, pre-computed features extracted from raw images. In this work, we use convolutional neural networks (CNNs) to classify images from a collection of 40,000 digitized works of art according to artist, genre, and location. After some simple pre-processing and downsampling, we employ a modified VGGNet architecture to achieve better than state-of-the-art results on artist and genre classification (we could not find a literature benchmark for location identification). We obtain a test set accuracy of 62.2% on artist classification, 68.5% on genre classification, outperforming the best algorithms in the literature by 3% and 10%, respectively.

Our results represent a step forward in the computational understanding of visual art. We present a confusion matrix for our artist classifier that reveals interesting quantitative similarities between different artists. In addition, we produce class optimization images for art of various genres which elucidate the quantitative features that distinguish various genres of art. We end by discussing possible extensions that have potential to improve these results even further.

1. Introduction

Many art museums have amassed and digitized very large collections of human artwork. This presents an immense curatorial challenge to both understand famous pieces and attribute pieces of unknown origin based on characteristic features of individual artists and time periods. An algorithm to identify the likely artist behind a work would be useful to both museum curators and auction houses as one way of validating an acquisition of unknown provenance.

In addition, while expert art historians can recognize a particular artist's style or estimate the date of an unknown piece, it is difficult for amateurs like ourselves to understand the image features that distinguish works of art. The main motivation for this project is to apply quantitative techniques to increase our understanding of visual art.

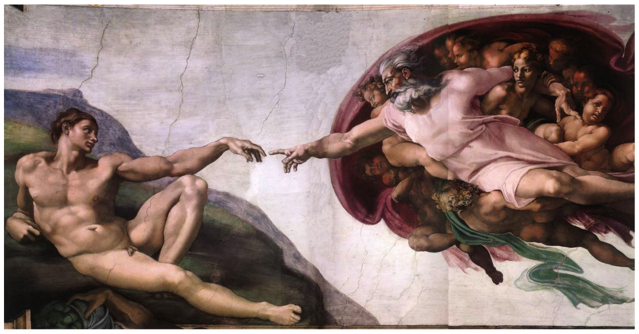
Figure 1. Example artwork found in our input catalog. Artist, genre and location truth labels for this image are 'Michelangelo', 'Religious', and 'Italian', respectively.

To do this, we take images found in a catalog of $\sim 40,000$ works from the Web Gallery of Art (WGA) and use CNNs based on a modified VGG-Net architecture to predict the artist, genre, and location (see Table 1 for examples of each) of a given work of art. An example input image, along with relevant truth data, from the WGA catalog is shown in Figure 1.

Our first goal is to achieve as high of accuracies as possible in classifying works of art by artist, genre, and location, and compare to results from literature. Our second goal is to use methods learned in this course to better understand the features of input images that lead to the classifications we make.

To tackle these goals, we review previous work in section 2, 3 outline a modified VGGNet architecture in section 4, describe our cross-validation strategy in section 5, describe our classification results from our artist-, genre-, and location-classifying networks in section 6, discuss the implications of this work on artistic understanding in section 7, and propose future extensions and improvements to this work in section 8.

Artist	Technique	Medium	Genre	Location
Michelangelo	Painting	Oil on canvas	Religious	Italian
Giotto	Sculpture	Fresco	Portrait	French
Rembrandt	Graphics	Marble	Landscape	Dutch
Albrecht Dürer	Illumination	Photograph	Mythological	Flemish
van Gogh	Architecture	Oil on wood	Still life	German

Table 1. The top five occurrences of several categories of artworks in our dataset, illustrating the diversity of the artworks in our dataset. The table includes our target categories of artist, genre, and location.

2. Previous Work

Within the field of art history, the primary method of classifying art is qualitative (see, e.g. [9]). Recently, however, several computational groups have endeavored to use tools from modern computer vision to advance the quantitative understanding of visual art, comparing the difficulty of identifying objects in art vs. photographs [5], and analyzing networks of artistic influence [18]. These collective efforts have made significant enough inroads that some have begun to consider the impact of these new quantitative paradigms on the field of art history [21].

Work on direct classification of artworks, however, has focused on using fixed feature-extraction methods. We were inspired to take on this challenge by work presented in [15], which used Fischer vector formalism to classify works from the collection of the famous Rijksmuseum. However, only their pre-computed feature vectors, rather than the full set of images, were available in a a convenient truth-matched format. In addition, as a museum focused on Dutch art, the catalog was not particularly diverse, so we decided to work on a different dataset.

The previous work we can most directly compare our results to are the classification analyses presented in [19]. They built a classifier to infer artist, genre, and artistic movement (e.g. impressionism, abstract, baroque, etc.) using hand-engineered features as input to a support vector machine (SVM) classifier. These pre-computed features included scale-invariant feature transforms (SIFT) and histogram of oriented gradients (HOG), as well as features derived from the outputs of a pre-trained four-layer CNN.

We benchmark our results against theirs, expecting some improvement given that we use precomputed features from a deeper CNN as well using a three-layer fully-connected network to classify on these features rather than a simple SVM.

The key question we will attempt to answer in this project will be whether or not fully or partially training a more complex CNN will enable us to outperform these results on similar tasks, the primary success metric being classification accuracies on a test set of withheld images.

3. Data

The WGA hosts a gallery of 39,302 rectangular RGB images in varying sizes ($\sim 1000 \times 1000$ pixels) in JPG format. It also provides a .csv catalog of accompanying metadata, giving the artist, year, technique, medium, genre, and artistic school of each work. The five most common entries for each of these categories is given in Table 1, demonstrating that this data set contains a diverse collection of well-known artists and styles. The images in the WGA catalog are all licensed for educational use.[1]

Since the images are rather large and irregularly sized, we convert the images to a standard 224×224 by downsampling along the shorter dimension as much as necessary to reach 224 pixels, downsampling the same amount along the longer dimension, and cropping the remainder. As an example, Figure 2 shows the 224x224 downsampling of the image shown in Figure 1. We restricted our attention to 20 well-known artists who were well-represented in the catalog, for a total of 5763 works.

For the genre and location tasks, we chose the 6 top genres and the 13 top locations represented in the catalog. In these categories, we had many more images (33,324 and 38,670, respectively) than in the artist classification task and could not load the full dataset into CPU memory on our EC2 instance. Keras lacks a memory-efficient data loader (its `train_on_batch` method just samples from an in-memory array) to load batches on-the-fly from disk. Instead of spending time implementing our own, we decided to downsample the images further to 96x96, as illustrated in Figure 3.

While not an optimal solution, it allowed us to explore the robustness of art classification at varying input scales. Some stylistic information is no doubt present in the small scales we are smoothing over, so a future extension of this work would be to implement ensembling of networks that train on data at varying scales, or novel layers like spatial pyramid pooling [10] or scale-invariant convolutional layers [13].

[1] http://www.wga.hu/frames-e.html?/legal.html

Figure 2. The 224x224 downsampling of Michelangelo's *Creation of Adam* used in the artists classifier.

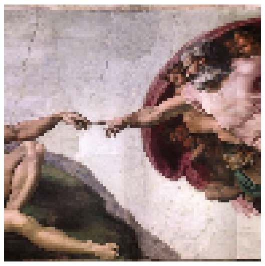

Figure 3. The 96x96 downsampling of the same work, used in genre and location classfication.

4. Methods

We implement our classifier using Keras, a recently-developed framework for building and training deep learning models [7]. We chose Keras because of its simple API to a Theano backend [2] [4], as well as the availability of pre-trained weights for a VGGNet architecture [1]. Another advantage of using Keras was that the code runs on both CPUs and GPUs without any manual adaptation, which allowed us to iterate quickly on our local machines before running larger jobs on Amazon EC2 g2.2xlarge instances. These instances use NVIDIA GRID K520 GPUs, with data access provided by an external Elastic Block Store (EBS) volume.

As a baseline model, we use a modified VGGNet architecture (based on configuration "D" as recommended in class) [20] initialized with fixed pre-trained weights from [1] in all the convolutional layers. We modify the vanilla VGG architecture by adding dropout and batch normalization layers between the fully-connected (FC) layers. A diagram of our network architecture is shown in Figure 4.

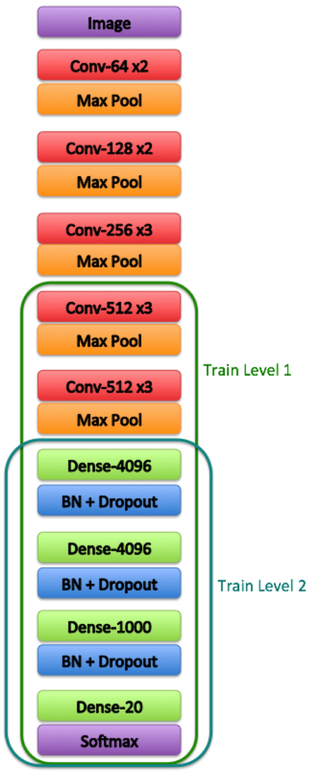

Figure 4. A schematic diagram of our modified VGGNet architecture. ReLU layers are omitted for brevity.

The convolutional blocks of our model (shown in red in Figure 4) are formed from sequential blocks zero-padding layers, convolution layers, and activation layers. The benefit of using convolution blocks over fully-connected hidden layers is these blocks take into account the spatial layout of the input data, as well as reducing the number of parameters in the model.

Since VGGNet uses 3x3 filters with stride 1, we add one row and column of zero-padding to the image before each convolutional layer to maintain the size of each image as it is passed through the network.

Finally, after each convolution layer, we use rectified linear units (ReLU) activation layers, which have been shown to be superior to sigmoidal activations when training deep networks [16]. ReLU units introduce non-linearity into the network by clipping negative activations to zero and never saturating positive activations:

$$\text{ReLU}(x) = max(0, x) \tag{1}$$

In addition to providing non-linearity, ReLU units, in contrast to sigmoidal and tanh units which squash large values to ≈ 1, where the resulting gradients are nearly zero, help avoid this vanishing gradient problem during backpropagation by continually increasing the gradient in conjunction with its final output.

The pooling blocks in our architecture (orange in Figure 4) implement 2x2 max-pooling to progressively reduce the spatial size of the images by retaining only the maximum value in a 2x2 activation block that slides across the input volume with stride 2. Pooling reduces the number of parameters needed for all downstream layers, which has the benefit of lowering memory costs in addition to reducing the potential for overfitting.

The second component of our network is blocks of dense layers that consist of fully-connected (FC) units, a batch normalization layer, a ReLU activation, and a dropout layer. The FC layers compute a linear combination of input features $x_{out} = W^T x_{in}$, as in a traditional neural network. The weight matrix W contains the learned parameters of the layer, initialized according to the prescription given in [11]. What makes the FC layers so effective in this context is that the inputs from the upstream convolutional filters contain spatial structure from the input image that FC layers alone would not be able to extract.

The batch normalization layer [12] serves to renormalize the activations of a given layer to a unit Gaussian distribution. This helps the later non-linearity perform better by ensuring that any weights that might have grown too large or too small are brought back to a reasonable size. The batch normalization formula for a single batch is given by:

$$\hat{x} = \frac{x_{in} - E\left[x_{in}\right]}{\sqrt{Var\left(x_{in}\right)}} \tag{2}$$

$$x_{out} = \gamma\hat{x} + \beta \tag{3}$$

where γ and β are learnable parameters of each batch that allow the network to undo the normalization forced by the first equation if training shows it preferable. This can be accomplished by setting $\gamma = \sqrt{Var\left(x_{in}\right)}$ and $\beta = E\left[x_{in}\right]$. It is important to note that since both of these formulas are differentiable, we can backpropagate through them like any other layer.

Lastly, these dense blocks also include two forms of regularization. The fully-connected layers contain L2 regularization which penalizes weight vectors by adding a term to the loss function (see Equation 5) of the form

$$\lambda \sum_{\text{FC layers } k} \sqrt{\sum_{i,j} (W_{ij}^{(k)})^2} \tag{4}$$

where λ is a hyperparameter that sets the regularization strength.

The second form of regularization is dropout, which makes the network more robust by randomly (with some probability p_{drop}) setting the activations of each node of the upstream layer to zero. p_{drop} is a hyperparameter that we include in our cross-validation. By removing nodes from the model at each step, each remaining weight must now be more robust in its contribution to the final prediction. Therefore, as with L2 regularization, we prefer weights with more evenly distributed values.

The final component of our network is the loss function. We have chosen a softmax classifier which uses the cross-entropy loss function:

$$L = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{e^{f_{y_i}}}{\sum e^{f_j}}\right) \tag{5}$$

This loss function will allow learning to continue even from examples it has already classified correctly. This is in contrast to the SVM loss function, which returns zero loss once the target class score is greater than all others by a fixed margin.

To optimize this loss function, we initially started with vanilla SGD with Nesterov momentum [17]. At the time, we were performing classification on three artists, and achieved a top validation accuracy of 77.6%. We decided to switch over to using the Adam optimizer [14], which keeps a decaying average of past gradients, as well as squares of gradients. These values are used to calculate the first-order and second-order moments of the past gradients. Using Adam on the initial 3-class task, we ended up with a top validation accuracy of 81.3% after 10 epochs, so we continued using Adam for all future optimizations to save cross-validation time.

5. Training and Cross-Validation

Our cross-validation experiments consisted of sampling three hyperparameters—learning rate, regularization strength, and dropout probability—over a broad range of scales. Learning rate and L2 regularization strength were sampled uniformly in log-space, while dropout probability was sampled directly from a uniform distribution to ensure unbiased exploration of parameter space.

We started by using random search rather than grid search to sample the parameter space more efficiently [3]. During this time, a coarse search was made by training at least 20 models per classification task with parameters chosen from a wide range of values, as described in Table 2. Because the goal was to simply determine what options displayed favorable properties, such as a continuously decreasing validation loss, these networks were only run for just four epochs.

Once we identified subsets of these ranges where we obtained reasonable results, we sampled more densely within a smaller range of possible values, as shown in Table 2. At this stage, we were more interested in seeing consistent

Hyperparameter	Coarse Range	Fine Range	Artists Opt.	Genres Opt.	Locations Opt.
Learning Rate	$10^{(-7,-2)}$	$10^{(-6,-4)}$	1×10^{-4}	1.2×10^{-5}	8×10^{-5}
L2 Regularization Strength	$10^{(-7,-2)}$	$10^{(-5,-4)}$	8×10^{-4}	1×10^{-3}	5×10^{-5}
Dropout Probability	(0.1,0.9)	(0.3,0.5)	0.40	0.35	0.45

Table 2. Ranges of tested hyperparameters for both coarse-grain and fine-grain cross-validation, along with the optimal values found for each classification task.

progress in the right direction, so we ran each model up to ten epochs.

The final stage of optimization involved tuning by hand. If the loss function appeared to be dropping to slowly, we pushed up the learning rate. If the training and validation results started diverging, we tried increasing the regularization strength. We also performed several targeted grid searches to assess changes in performance with respect to changes in a single parameter, with the other two held fixed.

After these three optimization steps, we obtained the loss functions illustrated in Figure 5.

For the artist-classifying network, the loss descends in a nice exponential, but the training and validation accuracies diverge after a few epochs. Despite our best efforts, we could not find a combination of hyperparameters that could make this go away, making it possible that we should have done more coarse sampling. The genres net has a nicely descending loss, and the training and validation accuracies increase together (albeit slowly). It is possible this network could have benefitted from additional capacity, although as described below, we were unable to find suitable hyperparameters to make this work.

The locations network might have benefitted from a slightly higher learning rate, as its loss descends more linearly than exponentially, but we were unable to find a better tradeoff with regularization strength that gave good generalization results.

Overall, though imperfect, our loss functions show that our three classifiers converged sufficiently to be useful. We describe two unsuccessful efforts to improve these results (which turned out to beat the best results in the literature anyway, as described in Section 6) below.

5.1. Attempted extensions

In an effort to boost our results, we also implemented a data augmentation pipeline that consisted of performing horizontal flipping and random crops on the image inputs. With 10 random crops per image, we effectively increased the amount of input data by a factor of 20. Using a customized implementation of a Keras ImageGenerator, the images were augmented during a pre-processed step, and then incrementally fed into the final training function. However, as before, we ran into CPU memory limitations when we discovered that the Keras data augmentation framework attempts to store the entire augmented array in memory, and then produce random indices to sample from it during training. In the interest of focusing on our primary project goals, we decided to proceed without data augmentation, but it would be one of the first things we would implement given additional time.

Secondly, motivated by the idea that CNNs should perform better if allowed to change the convolutional features, we decided to test the effect of opening up more layers for training. Our results up to this point yielded high training accuracy, but overfitting prevented us from seeing these quality results on the validation or testing data. We had only been training the fully connected layers as highlighted by Train Level 2 in Figure 4, leaving the weights of all convolutional layers fixed to their pre-trained value.

By opening up an additional six convolutional layers for training, we hoped we would be able to add more flexibility to the network without necessarily increasing overfitting because no new parameters would be introduced (they were already present in the model as pre-trained weights).

As Keras does not allow variable learning rates for each layer (the optimizer, with fixed learning rate, is instantiated separately from the model), we split up the training into two parts.

First, we trained a network that had all the layers in Train Level 1 open for optimization using lower learning rates (to not train too far away from the pre-trained weights). We performed grid search of learning rates in $[2 \times 10^{-6}, 5 \times 10^{-6}]$, regularization strengths in $[8 \times 10^{-4}, 2 \times 10^{-3}]$ and dropout probabilities in $[0.4, 0.5]$. The best result we achieved was a validation accuarcy of 30.3% after 10 epochs. This represented a higher starting point for training the second network, which would often initialize around 20% after the first epoch.

The second half of this process kept all the convolution layers fixed, and varied only the weights in fully-connected layers with higher learning rates. We performed grid search of learning rates in $[5 \times 10^{-5}, 8 \times 10^{-6}, 2 \times 10^{-6}]$, regularization strengths in $[8 \times 10^{-4}, 2 \times 10^{-3}]$ and dropout probabilities in $[0.4, 0.5]$. However, after this limited cross-validation, the final results only went up to 35.2%, much lower than the 60% threshold we reached training just the bottom level layers. Lacking the resources to perform a full cross-validation on these expanded networks, we returned to our original results.

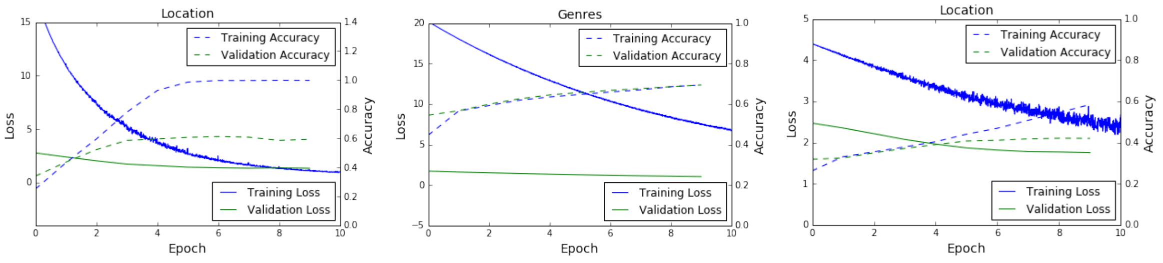

Figure 5. Loss functions from the training of our three classifiers.

6. Results

We express our classification results using confusion matrices computed from a withheld test set of images. The confusion matrices on a withheld test set for artist, genre, and location classification are shown in Figures 6, 7, and 8, respectively.

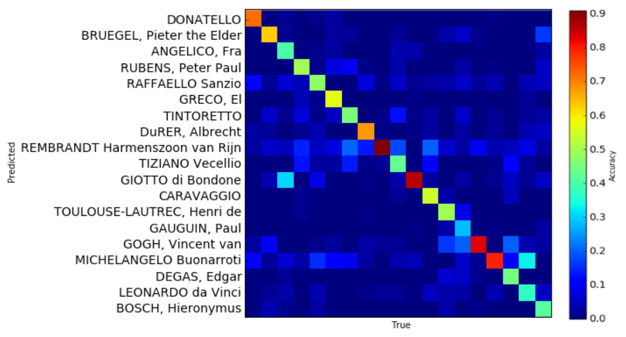

Figure 6. Confusion matrix for artists classification on a test set. X-axis labels follow the Y-axis labels, and were omitted due to space limitations.

Figure 6 shows that our artist classifier overall performs well, as it is strongly diagonal. We obtain best performance on the most popular artists in our dataset: Michelangelo, Rembrandt, van Gogh, and Giotto. The lines in confusion matrix for these artists show that, to some extent, the network is misclassifying other artists as these four, probably due to their frequent occurrence in the training set.

It is artistically interesting to note that Fra Angelico and Giotto are frequently confused, as Fra Angelico was inspired by Giotto and followed in his tradition of placing images of Jesus and Mary into scenes from everyday life [8]. Additionally, Michelangelo and Leonardo da Vinci are also frequently confused, which makes sense as they are both Italian Renaissance artists.

For our genres classifier, the results are more mixed. The overall test accuracy is high (see Table 3). However, mythological and genre paintings—that is, paintings of scenes from everyday life—are both mainly classified as religious art. These two categories had the fewest training examples, and so were frequently classified as religious art, most

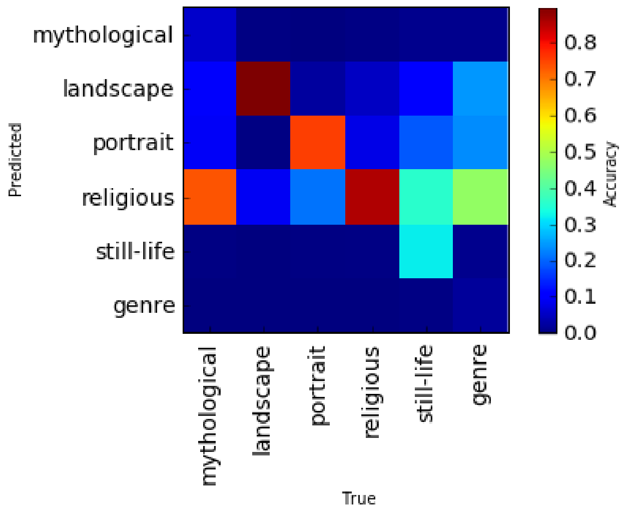

Figure 7. Confusion matrix for genres classification on a test set.

likely because of the predominance of religious works in our dataset.

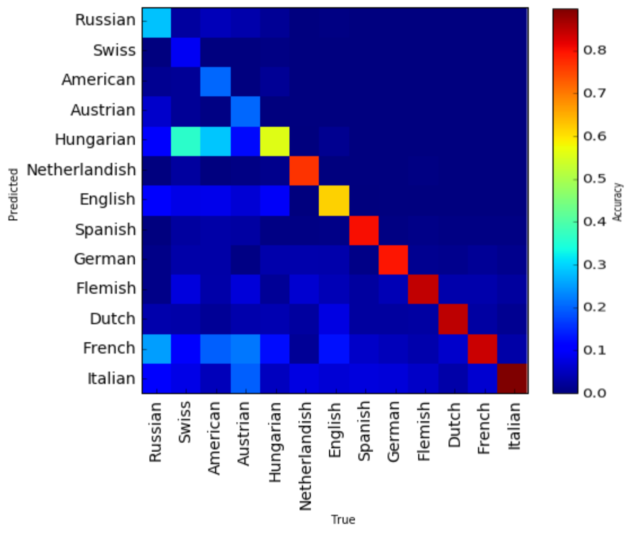

Figure 8. Confusion matrix for genres and location classification on a test set.

For our locations classifier, we perform well for the categories for which we have many training examples (lower right corner), but performance drops off as we consider categories with fewer training examples, which are frequently misclassified as more popular categories.

Task	Best previous	Our performance
Artist	59.3%	62.2%
Genre	58.3%	68.5 %
Location	—	43.1%

Table 3. Test accuracies from literature compared to our performance. We show a substantial improvement on state of the art performance!

We summarize our performance relative to the results from the literature in Table 3. Overall, we exceed the performance of previous work on artists and genre classification. We could not find a performance benchmark for location classification to compare to in the literature, but based on our confusion matrix in Figure 8, we believe our method is promising for this task as well.

Relative to location classification, it is not too surprising that we performed so well on artist classification because this task was completed using 224x224 images whereas the location task was completed using 96x96 images. However, it is interesting that we did well classifying genres which was also forced to employ down-sampled 96x96 images due to memory constraints. Our hypothesis is that the characteristic features that define genre classes are attributes such as rivers, faces, etc. that lie at large spatial scales. In other words, genre is more about the large-scale composition of the painting rather than fine details.

On the other hand, our lesser performance on location classification may signify that the identifying features of one country's art from another is in the details of the brushwork rather than a work's large-scale composition. This makes sense, as portraits, still-lifes, and landscapes are all painted in many countries, yet with distinctive style. Regarding artists, we're clearly picking up distinctive features at small-to-medium spatial scales, although more tests would be needed to see where the majority of the constraining power lies.

7. Discussion

A goal of this project was to increase our *understanding* of visual art—in essence, to understand the quantitative features that are characteristic of various artists, genres, and artistic schools. In addition to considering the impact of varying spatial scales, as above, another way of investigating this is through class optimization.

To construct an image that optimizes for output as a given class, we begin by forward-propagating an image of random noise through our trained network [6]. We then backpropagate through the network, starting with gradients set to zero for every class score except the target class, whose starting gradient we set to 1. We update the input image via gradient ascent, which ensures that the class score of the input image will increase at every iteration. Results from applying this method to the genre-classifying network are shown in Figure 10.

The artist and location class examples are difficult to interpret. As an example, the class-optimized image for Vincent van Gogh is shown in Figure 9.

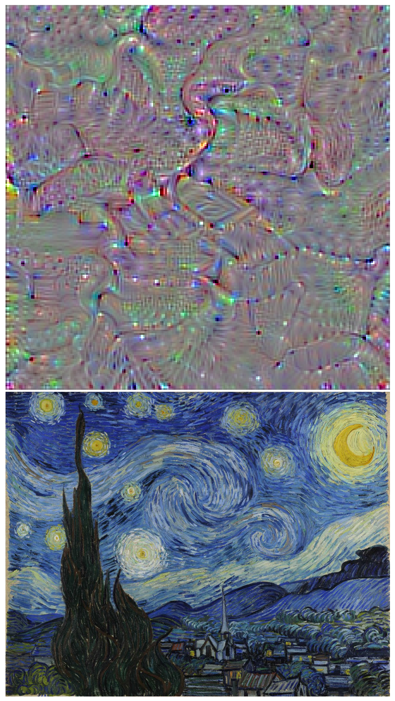

Figure 9. Class-optimized image for Vincent van Gogh, compared with his famous *Starry Night*.

Broad, swirling patterns are prominent in the class-optimized image, with brushstroke periodicities in between them, which correspond to the famous features of van Gogh's work as seen in *Starry Night*. However, it is difficult to make more definitive claims about the captured features without greater knowledge of specific artistic styles.

On the other hand, the genre class images are very interesting and informative. Figure 10 shows the results for four genre classes, which each have highly interpretable features. The portrait image has a head-like structure in the top-center of the image, with little characteristic structure elsewhere. In the landscape image, the horizon, a river, trees, clouds, and a mountain are all quite distinctive. The still life has grapes near the center surrounded by wicker baskets and more round fruits–a common still-life motif. Finally, the idealization of religious art is dominated by bearded faces, with what might be angel-like wings in the upper corners.

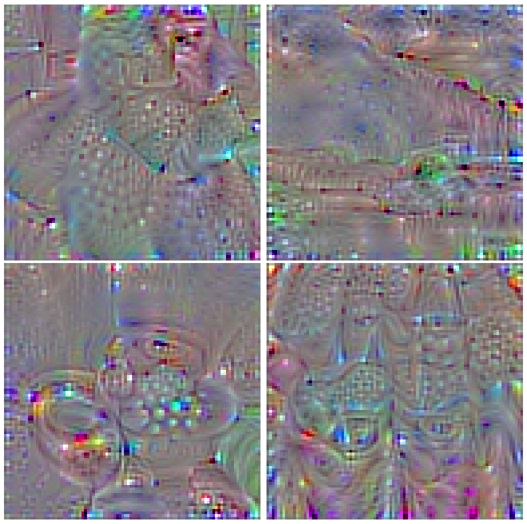
Figure 10. Clockwise from top-left, images from the portrait, landscape, religious, and still life class optimizations.

8. Conclusions and Future Work

After exploring our dataset and implementing three classifiers, we obtain results that improve significantly on the state of the art for artist and genre classification. Our results from analyzing images at varying spatial scales showed that while artistic genre can be inferred from highly downsampled images, the location of origin is likely best constrained by small-scale (brushstroke) information. In addition, we used class optimization to extract the quantitative features that define various genres of art.

In this work, we were frequently limited by memory constraints; it is clear that our next step in improving these results is to circumvent this resource limitation. Loading the training images on-the-fly will allow our server to handle all the data we have available at maximum image resolution. Although this I/O would introduce more overhead, the cost of the extra time needed for training should be offset by the ability to use the larger image sizes, extracting information at all spatial scales, rather than the downsampled 224x224 or 96x96 pixel images. Furthermore, with this on-the-fly loading, we would also be able to perform data augmentation to increase the number of effective training examples. We believe these changes have the highest potential for yielding significant gains.

References

[1] L. Baraldi. https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3.

[2] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[3] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, Feb. 2012.

[4] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[5] H. Cai, Q. Wu, T. Corradi, and P. Hall. The Cross-Depiction Problem: Computer Vision Algorithms for Recognising Objects in Artwork and in Photographs. *ArXiv e-prints*, May 2015.

[6] F. Chollet. http://blog.keras.io/how-convolutional-neural-networks-see-the-world.html.

[7] F. Chollet. Keras. https://github.com/fchollet/keras, 2015.

[8] G. Didi-Huberman and A. F. Angelico. *Fra Angelico: Dissemblance Figuration*. University of Chicago Press, 1995.

[9] P. DiMaggio. Classification in art. *American Sociological Review*, 52(4):440–455, 1987.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *ArXiv e-prints*, June 2014.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *ArXiv e-prints*, Feb. 2015.

[12] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv e-prints*, Feb. 2015.

[13] A. Kanazawa, A. Sharma, and D. Jacobs. Locally Scale-Invariant Convolutional Neural Networks. *ArXiv e-prints*, Dec. 2014.

[14] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, Dec. 2014.

[15] T. Mensink and J. van Gemert. The rijksmuseum challenge: Museum-centered visual recognition. 2014.

[16] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[17] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[18] B. Saleh, K. Abe, R. Singh Arora, and A. Elgammal. Toward Automated Discovery of Artistic Influence. *ArXiv e-prints*, Aug. 2014.

[19] B. Saleh and A. Elgammal. Large-scale Classification of Fine-Art Paintings: Learning The Right Metric on The Right Feature. *ArXiv e-prints*, May 2015.

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

[21] E. L. Spratt and A. Elgammal. Computational Beauty: Aesthetic Judgment at the Intersection of Art and Science. *ArXiv e-prints*, Sept. 2014.