Good or Bad Style Transfer*

Course Project†

Chukwunonso Wogu
Chuka Ezema
7824811
7804575

ABSTRACT

This paper explains the process of combining two independent images through Neural Style Transfer. It also shows the procedure used to create a working image classifier.

ACM Reference Format:
Chukwunonso Wogu and Chuka Ezema. 2018. Good or Bad Style Transfer: Course Project. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As sited from medium, "Neural style transfer is an optimization technique used to take three images, a content image, a style reference image (such as an artwork by a famous painter), and the input image you want to style, and blend them together such that the input image is transformed to look like the content image, but " painted" in the style of the style image. "

The goal of the project was to Implement a neural style transfer algorithm where we optimized cost functions to get pixel values and an Image Classifier to determine if the image gotten from the neural style transfer is good artistically or not. Neural style transfer merges two images, namely, a content image and a style image, to create a generated image. The generated image combines the content of the image with the style of image. We used a previously trained convolutional network and built on top of it using transfer learning. We used a 19-layer version of the VGG network. The model was already trained on a very large ImageNet database and thus had learned to recognize a variety of low-level features (at the earlier layers) and high-level features (at the deeper layers). We built the neural style transfer by implementing various functions such as Content cost, Style cost, Gram matrix/Style matrix, Total cost. A noisy image is then generated from the content image and its pixel values are now adjusted to match that of the content image and style image by using values gotten from our cost functions. The second part of our project was to predict if the image is good artistically or not, and we built the image classifier using Keras.

*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 NEURAL STYLE TRANSFER

"Neural style transfer is an optimization technique used to take three images, a content image, a style reference image (such as an artwork by a famous painter), and the input image you want to style, and blend them together such that the input image is transformed to look like the content image, but "painted" in the style of the style image." In Deep Neural Networks the most powerful in image processing tasks are Convolutional Neural Networks. Convolutional Neural Networks consist of layers of small computational units that process images hierarchically in a feed-forward manner. Each layer of units can be understood as a collection of image filters, each of which extracts a certain feature from the input image[2]. Thus, the output of a given layer consists of feature maps which are differently filtered versions of the input image. When Convolutional Neural Networks are trained on object recognition, they develop a representation of the image that makes object details more distinct. Therefore, the image is transformed into representations that increasingly care about the actual content of the image compared to its detailed pixel values[2]. We can directly visualize the information each layer contains about the input image by reconstructing the image only from the feature maps in that layer. Higher layers in the network capture the high-level content in terms of objects and their arrangement in the input image but do not constrain the exact pixel values of the reconstruction. In contrast, reconstructions from the lower layers simply reproduce the exact pixel values of the original image. The generated images were created by finding an image that simultaneously matches the content representation of the content and the style of the artwork. The image content and style cannot be completely disconnected[2]. When combining an image that combines the content of one image with the style of another, there usually does not exist an image that perfectly matches both at the same time. However, the loss function we minimize during image combination contains two terms for content and style respectively, that are well separated. A strong emphasis on style will result in images that match the appearance of the artwork, effectively giving a texturized version of it, but hardly show any of the contents detail. When placing a strong emphasis on content, one can clearly identify the image, but the style of the painting is not as well-matched. For a specific pair of source images, one can adjust the trade-off between content and style to create good Images.



2.1 Implementation

We used a pre-trained 19 layer VGGNetwork. We then computed various functions to help us with the image matching using Tensorflow. The given input image x was encoded in each layer of the CNN by the filter responses to that image. A layer with Nl distinct filters has Nl feature maps each of size Ml, where M is the height times the width of the feature map. So the responses in a layer l can be stored in a matrix Fl belongs to R N*M where Flij is the activation of the ith filter at position j in layer l. We then performed gradient descent on a noise image to find another image that matches the feature responses of the original image. We then defined the squared-error loss between the two feature representa-

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2 .$$

tions

The derivative of this loss with respect to the activations in layer

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^{l}} = \begin{cases} \left(F^{l} - P^{l}\right)_{ij} & \text{if } F_{ij}^{l} > 0 \\ 0 & \text{if } F_{ij}^{l} < 0 \,. \end{cases}$$

l equal

From which the gradient with respect to the image x can be computed using standard error back-propagation. Thus we can change the initially random image x until it generates the same response in a certain layer of the CNN as the original image p.

On top of the CNN responses in each layer of the network, we built a style representation that computes the correlations between the different filter responses

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}.$$

To generate a texture that matches the style of a given image, we then used gradient descent from the noise image to find another image that matches the style representation of the original image. This is done by minimizing the mean-squared distance between the entries of the Gram matrix from the original image and the Gram matrix of the image to be generated[2]. The contribution of that layer to the total loss is then

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^{l} - A_{ij}^{l}\right)^2$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

and the total loss is

The loss function we minimize is $\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$

2.2 Image Classifier

We then proceeded to build our image classifier to determine if the image gotten from the neural style transfer is good artistically or not. The image classifier was built using Keras and we encountered several challenges like 1) Gathering a large enough dataset for the image classifier. The main bottleneck with this project was the lack of a predefined dataset, requiring us to generate our own and label several images. 2) Getting the neural network to identify distinct features because the content and style of each image varies which then creates a different generated image with distinct features.

2.3 Collecting the Dataset

In order to train our machine, we needed to compile a relatively large dataset so that our model can learn from them by identifying out certain relations and common features related to the generated images. We started off by generating the images manually which was taking an unreasonable amount of time but fortunately we stumbled upon a collection of images involving neural style transfer which was available on the internet.

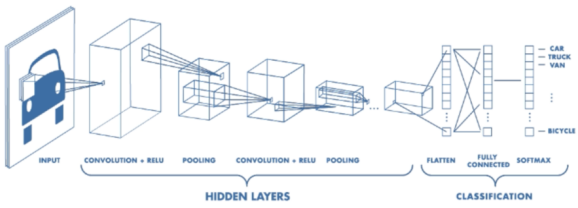
2.4 Importing Libraries, Splitting the Dataset, and Building the CNN

We imported several libraries to enable us to build our image classifier. After importing the libraries, we then needed to split our data into two parts - training set and test set. We then moved on to building our network. The network consists of three parts, 1) Convolution, 2) Polling, and 3) Flattening. The primary purpose of Convolution is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using input data. The image can be considered as a matrix and the convolution of that matrix with feature yields a matrix to be included in the feature map. An additional operation called ReLU is used after every Convolution operation. The next step is of pooling. Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. We used max pooling which takes the largest element from the rectified feature map within a window.

2.5 Full Connection, Data Augmentation, and Training our Network

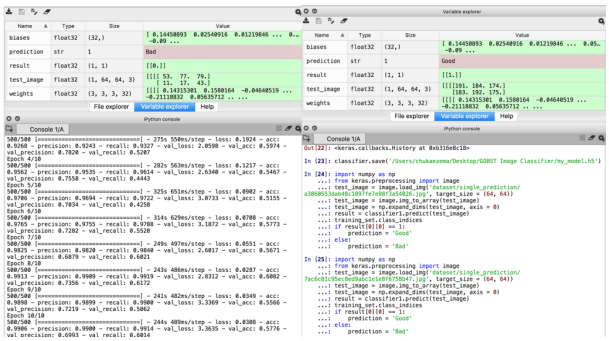
In this step, we apply full connection which is connecting our convolutional network to a neural network and then compiling our network. We began with a 1 layer neural network with a sigmoid function as an activation function for the last layer which was needed to find the probability of the generated image being good or bad and later grew the network using transfer learning. While training your data, we applied data argumentation which is a way we can reduce overfitting on models, where we increase the amount of training data using the information only in our training data. We then proceeded to train our model. We did not have access to a gpu but we used 8GB RAM CPU which took about 50 minutes, training it on 10 epochs, 500 steps per epoch, and 250 validation steps as our hyperparameters. This produced an accuracy of 99 percent on the training set and a 57.76 percent accuracy on the validation/test set. We also got a Recall value 99.14 percent on the training set and 60.14 percent on the validation/test set. We got a Precision value of 99 percent on our training set and 69.93 percent on the validation/test set. This was a result we predicted when embarking on the project as the feature of each generated image

varies widely thus not providing the neural network with enough features to learn with. The results of the two-layer model did not improve on the training set and improved minutely on the test set to 60.17 percent.



2.6 Testing

We then tested our model using random images and the result varied depending on the quality of the images and other features it may have learned. The prediction based on pixel quality made sense as most of the images given to the network were high-quality images so it classified low-quality images as bad



2.7 Components Used

Pre-trained VGG (enormous neural network of 19 layers), More than 700 images, Tensorflow (to calculate different losses for the content and style cost.), Keras

2.8 Challenges

Gathering a large enough dataset for the image classifier. The main bottleneck with this project was the lack of a predefined dataset, requiring us to generate our own. Getting the neural network to identify distinct features because the content and style of each image varies. This created a different generated image with distinct features.

2.9 Conclusion

So, we were able to successfully implement a neural style transfer algorithm and an image classifier to determine if the image generated was good or bad artistically. The image classifier we built can be applied to a diverse range of images depending on the application. This can be easily done by changing the dataset with what is required.

2.10 References

[1] Home. (n.d.). Retrieved from https://www.deeplearning.ai/

[2] A Neural Algorithm of Artistic Style. (n.d.). Retrieved from https://arxiv.org/pdf/1508.06576.pdf

[3] Gupta, V. (2017, November 29). Home. Retrieved from https://www.learnopen classification-using-convolutional-neural-networks-in-keras/

[4] Agarwal, Y. (2018, July 08). Create your first Image Recognition Classifier using CNN, Keras and Tensorflow backend. Retrieved from https://medium.com/nybles/create-your-first-image-recognition-classifier-using-cnn-keras-and-tensorflow-backend-6eaab98d14dd

[5] Cornelisse, D. (2018, April 24). An intuitive guide to Convolutional Neural Networks. Retrieved from https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050