data

MDPI

Article

The Visualization of Deep Neural Network

Yi Liu [1], Juan Wang [2], and Yukai Wu [3]

[1] Affiliation 1; yliu126@fiu.edu

[2] Affiliation 2; jwang090@fiu.edu

[3] Affiliation 3; ywu048@fiu.edu

Abstract: Deep Neural network models have recently demonstrated impressive performance on every deep learning application. Despite this encouraging progress, there is still little insight into the internal operation and behavior of these complex models, or how they achieve such good performance. From a scientific standpoint, this is deeply unsatisfactory. In this paper we consider image transformation problems, where an input image is transformed into an output image. The approach for solving this task is to train a feedforward convolutional neural network in a supervised manner, using perceptual loss functions that depend on high-level features from a pre-trained loss network (VGG19) to measure the difference between output and ground-truth images. In order to visualize the operation details and performance of the deep neural nets, we introduce a visualization technique, a multi-layered Deconvolutional Network (deconvnet), which reveals the input stimuli that excite individual feature maps at any layer in the model. It also allows us to observe the evolution of features during training and to diagnose potential problems with the model. Finally, we use TensorBoard to display the whole neural network architecture, these hidden layer details, intermediate results and the image generated after style transformation.

Keywords: Neural Nets visualization, Deconvolutional Network, style transformation, deep learning

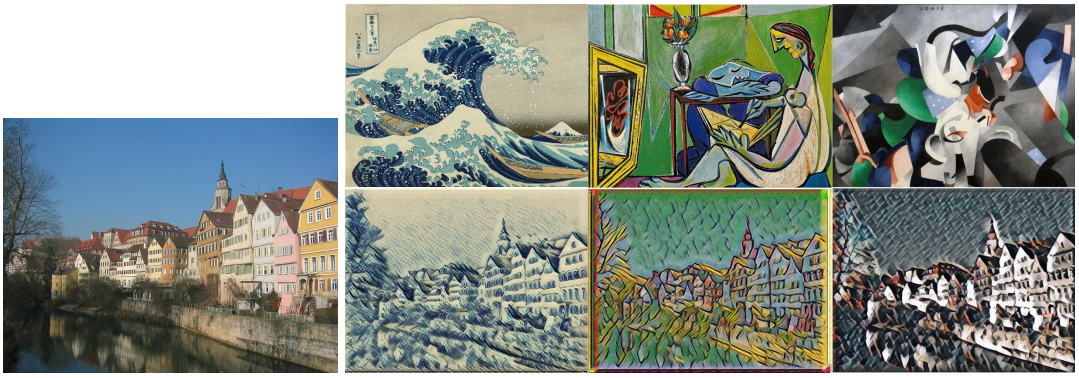
1. Introduction

Since the deep neural networks introduced in the early 1990's, they all have demonstrated excellent performance at tasks such as image classification, face detection, and image style transformation. Image style conversion is an indispensable part of AI art and is becoming more and more popular. The Image Style Transformation Model provides an algorithm for converting your own photos into similar world famous paintings. It uses a neural network to combine the original image with the style image to produce a new image. This kind of picture change is a magical process. Our goal is to present the intermediate process through data visualization techniques and observe the entire transformation process. At the same time, it also shows the image features extracted from the VGG19 network. In this experiment, we used the image-style transformation algorithm of transform-net (Justin Johnson et al., 2016). In the display of the feature picture of the picture, we use deconvnet technology to convert the picture feature map in the neural network into a human-understandable picture, so as to observe which part of the picture is extracted by the VGG neural network. We add many parts of visualizations in our algorithm. These visualizations allow us to find model architectures, analyze the loss functions and weights distribution. All these visualized results can be collected and displayed in the Tensor Board, a tool operates by reading TensorFlow events files, which contain summary data that you can generate when running TensorFlow.

2. Results

In the experiment, we training three different style model, la_muse, wave_crop and udnie. We use tensorflow to implement the experiment and use tensorboard to display our outcome.

2.1. style transform outcome



(a) (b)

Figure 1. This figure is show our input and output. The output image combine the style image and original image. (a): original photo, (b): style image(top) and output image(bottom). first is wave crop style, second is la muse style, final is udnie style.

2.2. Intermediate data in the neural network

Our visualization of the deep neural network contains four parts, the change of photos in the transform-net, the architecture of the network, the loss value, and the variable in the transform-net, such as weights in the convolution layer and scale, shift in the batch normalization layer.

1. Image in the neural network

   Understanding the operation of a convnet requires interpreting the feature activity in intermediate layers. We map these activities back to the input pixel space, showing what input pattern originally caused a given activation in the feature maps. We perform this mapping with a Deconvolutional Network (deconvnet) . A deconvnet can be thought of as a convnet model that uses the same components (filtering, pooling) but in reverse, so instead of mapping pixels to features does the opposite. Here, they are not used in any learning capacity, just as a probe of an already trained convnet.
   We use deconvnet to convert the image feature maps to an image, and then we can see how the change happen in each layer of the deep neural network.

   - Transform-net

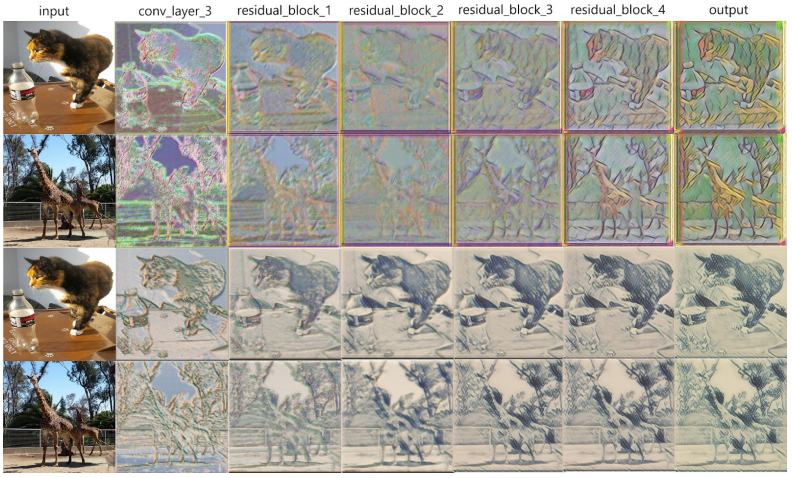


Figure 2. the figure show the processes how an original image change to the image with style from left to right. We get these image from several layers in transforms-net. layers' name is on the top of the picture.

- VGG19

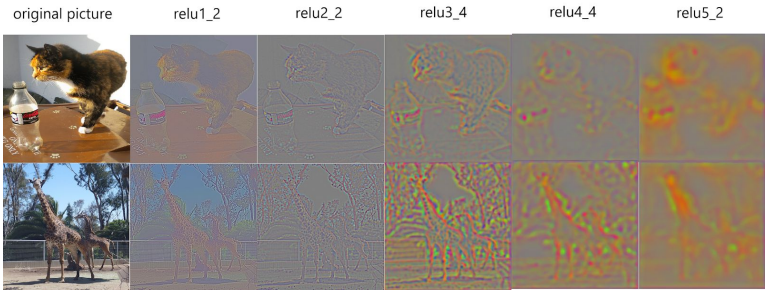


Figure 3. the figure show how the VGG19 learned from the content images. We can see the VGG19 can recognize the profile of the animals in the images.

2. Architecture

We show the architecture of the neural network by tensorboard. When we create a neural network by tensorflow, we can write the graph data into a log file. the tensorboard reads the log file, and then show the graph by web.

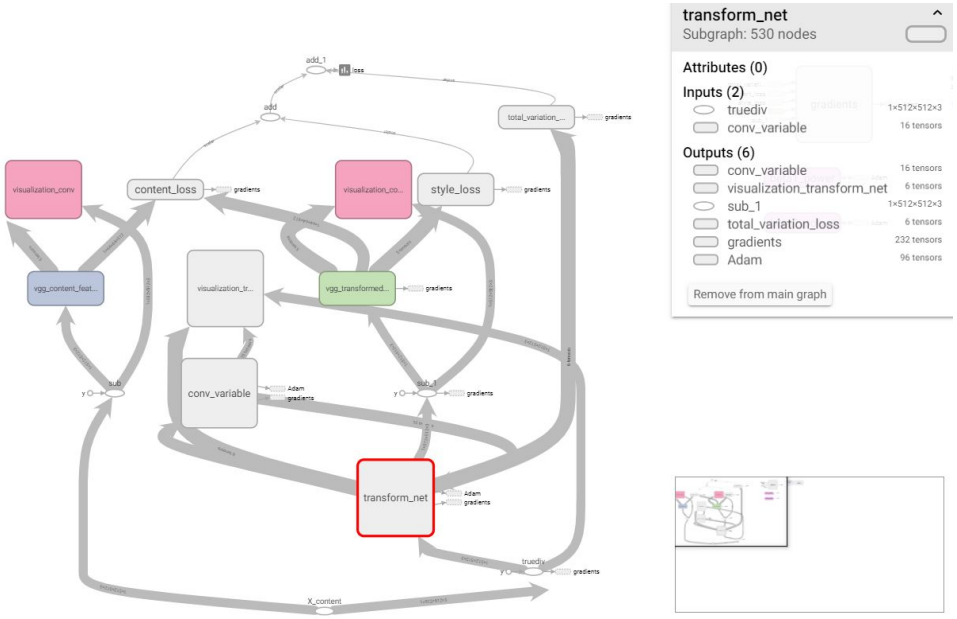


Figure 4. This figure show the total architecture of our neural network. We can see transform_net, vgg_transformed, vgg_content, visualization part and the computation of loss function and how data go through this neural network in the graph. When we select one block, the input, output and down-stream node will display in the right top of the picture. In the bottom, there is our input node "X_content", and the top is the total loss value.

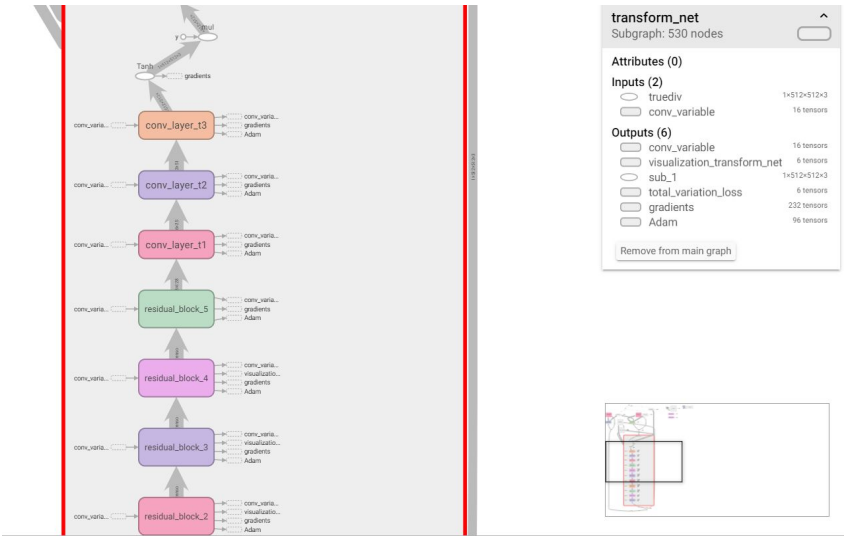


Figure 5. The figure is the detail of the transform-net after zooming in. we can see the transform-net contains three convolution layers, five residual block and three deconvolution layers.

3. Loss value
We collect the loss value when the neural network is training. These figure show how the loss value change during the training period.
    - Loss

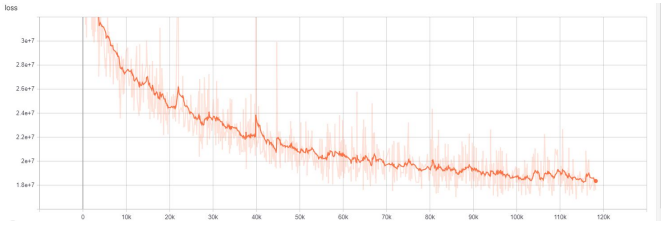


Figure 6. The total loss value is the content loss plus the style loss during the network training period. The trender of the loss value is reducing.

- Content Loss

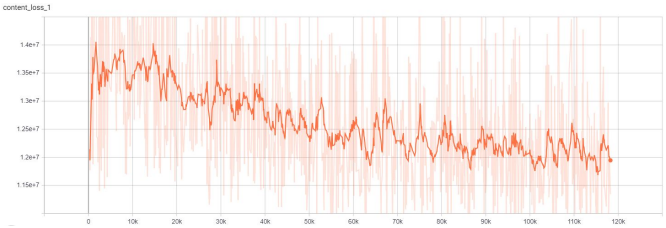


Figure 7. Content loss during the network training period

- Style Loss

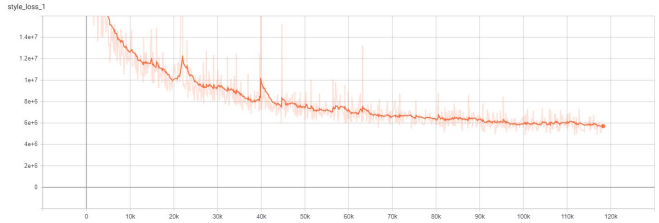


Figure 8. Style loss during the network training period

4. Variable

In the neural network, there are a lot of variables that need train to fit the data. We use Distribute Graph and Histograms to display these variables.

- Distribute Graph

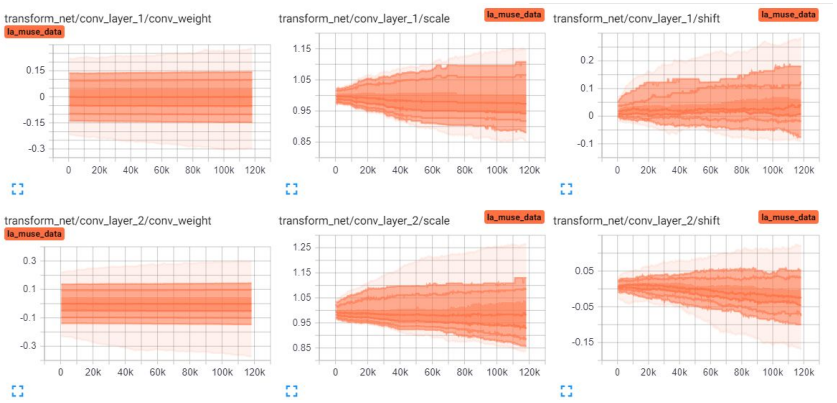


Figure 9. The distribution of weights' value in the transform-net, the x axis is the number of step during training period, the y axis is the value of weights.

- Histograms

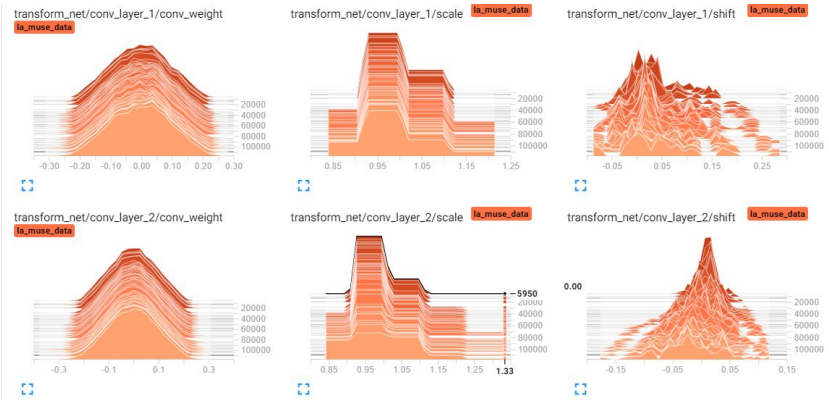


Figure 10. The distribution of weights' value in the transform-net for each step. In the right is the number of steps, in the bottom is the weights' value. The high and low of this graph mean the number of weights' value that equal to the value in the bottom.

2.3. Formatting of Mathematical Components

1. The loss Value

Our neural network consists of two components: an image transformation network f_W and a loss network $\varnothing$ that is used to define several loss functions $l_1, l_2, l_3, \ldots, l_k$. The image transformation network is a deep residual convolutional neural network parameterized by weights W; it transforms input images x into output images $\hat{y}$ via the mapping $\hat{y} = f_W(x)$. Each loss function computes a scalar value $l_1(\hat{y}, y_i)$ measuring the difference between the output image $\hat{y}$ and a target image y_i. The image transformation network is trained using Adam to minimize a weighted combination of loss functions:

$$W^* = \arg\min_W \mathbf{E}_{x,\{y_i\}} \left[\sum_{i=1} \lambda_i \ell_i(f_W(x), y_i) \right] \quad (1)$$

The loss network $\varnothing$ is used to define a feature reconstruction loss $l^{\varnothing}_{feat}$ and a style reconstruction loss $l^{\varnothing}_{feat}$ style that measure differences in content and style between images.

2. Feature Reconstruction Loss

Let $\varnothing_j(x)$ be the activations of the jth layer of the network $\varnothing$ when processing the image x; if j is a convolutional layer then $\varnothing_j(x)$ will be a feature map of shape $C_j \times H_j \times W_j$. The feature reconstruction loss is the (squared, normalized) Euclidean distance between feature representations:

$$\ell_{feat}^{\phi,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\phi_j(\hat{y}) - \phi_j(y)\|_2^2 \quad (2)$$

3. Style Reconstruction Loss

As above, let $\varnothing_j(x)$ be the activations at the jth layer of the network $\varnothing$ for the input x, which is a feature map of shape $C_j \times H_j \times W_j$. Define the Gram matrix $G_j^{\varnothing}(x)$ to be the $C_j \times C_j$ matrix whose elements are given by

$$G_j^{\phi}(x)_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \phi_j(x)_{h,w,c} \phi_j(x)_{h,w,c'}. \quad (3)$$

The style reconstruction loss is then the Frobenius norm of the difference between the Gram Matrices of the output and target images:

$$\ell_{style}^{\phi,j}(\hat{y}, y) = \|G_j^{\phi}(\hat{y}) - G_j^{\phi}(y)\|_F^2. \quad (4)$$

3. Discussion

Once training is complete, projecting each separately down to pixel space reveals the different structures that excite a given feature map, hence showing its invariance to input deformations. The projection from each layer show the hierarchical nature of the features in the network.

We also visualize the progression during training of the strongest activation within a given feature map projected back to pixel space.

Feature visualization is an active area of research and there are some other feature visualization techniques for future research.

Occlusion can be used in feature visualization by blocking out selective parts of an image and seeing how a network responds. The result should be a heatmap that shows the predicted class of an image as a function of which part of an image was occluded. The reasoning is that if the class score for a partially occluded image is different than the true class, then the occluded area was likely very important!

Salience can be thought of as the importance of something, and for a given image, a saliency map asks: Which pixels are most important in classifying this image? Not all pixels in an image are needed or relevant for classification. Saliency maps aim to show these important pictures by computing the gradient of the class score with respect to the image pixels. A gradient is a measure of change, and so, the gradient of the class score with respect to the image pixels is a measure of how much a class score for an image changes if a pixel changes a little bit.

4. Materials and Methods

Our algorithm consists of two parts, transform-net and VGG19. Transform-net is mainly used to convert the style of photos. His input is a normal photo and the output is a converted style photo. VGG19 is used as a Loss network, which is mainly used to obtain feature maps of style photos,

feature maps of content photos, and feature maps of converted pictures. The feature maps of these photos are used to calculate content loss and style loss. The calculated loss value is placed in the Adam optimizer and iteratively calculated to minimize the loss value. We used a deconvolution network for visualization technology. The deconvolution network first inverts and reconstructs the entire neural network, then derives the feature map of the image from the middle layer and inputs it into the deconvolution network, and finally obtains an image of the original size that can be observed. Through these pictures, we can know what VGG19 learned from the content picture during the training process, and how the converted picture is gradually converted from the original picture. This allows us to clearly understand the training process throughout the network.

We use tensorboard to display the overall architecture and internal variable data of the network, and use the API provided by tensorflow to export the data and save it to a log file, then read the data saved in the log file through tensorboard and display it on the web.

We use COCO dataset for my training dataset. It contains 130k difference images. you can download it by the link. http://images.cocodataset.org/zips/train2017.zip

5. Conclusions

In this paper we have combined the benets of feed-forward image transformation tasks and optimization-based methods for image generation by training feed-forward transformation networks with perceptual loss functions. We have applied this method to style transfer where we achieve comparable performance and drastically improved speed compared to existing methods.

After, we presented a novel way to visualize the activity within the model. With clear understanding of how and why those neural nets work, the development of better models will never be reduced to trial-and-error. The techniques are actually the basis for applications like Style Transfer and Deep Dream that compose images based on layer activations and extracted features. Perhaps most importantly, those visualizations give you a way to show and communicate to other people what your networks have learned.

References

[1] Github Neural Doodle. Available online: https://gitlab.com/GovanifY/neural-doodle (accessed on 10/04/2019)
[2] Github Neural Doodle. Available online: https://github.com/alexjc/neural-doodle (accessed on 10/04/2019)
[3] Github fast-neural-style-keras. Available online: https://github.com/misgod/fast-neural-style-keras (accessed on 10/04/2019)
[4] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Available online: https://arxiv.org/abs/1409.1556 (accessed on 10/04/2019)
[5] AlexNet. Available online: https://en.wikipedia.org/wiki/AlexNet (accessed on 10/04/2019)
[6] Convolutional neural network. Available online: https://en.wikipedia.org/wiki/Convolutional_neural_network (accessed on 10/04/2019)
[7] Deep learning. Available online: https://en.wikipedia.org/wiki/Deep_learning#Deep_neural_networks (accessed on 10/04/2019)
[8] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. CoRR, abs/1508.06576, 2015.
[9] F. Liu. GitHub Style-transfer. Available online: https://github.com/fzliu/ style-transfer
[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.
[11] VGG-19 Pre-trained Model for Keras. Available online: https://www.kaggle.com/keras/vgg19/home (accessed on 10/04/2019)
[12] MathWorks-VGG19. Available online: https://www.mathworks.com/help/deeplearning/ref/vgg19.html (accessed on 10/04/2019)
[13] Asha Anoosheh; Rishi Kapadia; Jared Rulison. Image Transformation via Neural Network Inversion. Available online: https://rulison.github.io/CS_280_Final_Report.pdf (accessed on 10/04/2019)
[14] GitHub-VGG19. Available online: https://gist.github.com/baraldilorenzo/8d096f48a1be4a2d660d (accessed on 10/04/2019)
[15] GitHub COCO Dataset. Available online: https://github.com/cocodataset/cocoapi (accessed on 10/04/2019)
[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, p. 436, 2015.

[17] Nikulin, Y., and Novak, R. 2016. Exploring the neural algorithm of artistic style. CoRR abs/1602.07188.
[18] Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556.
[19] Yang, Y.; Zhao, H.; You, L.; Tu, R.; Wu, X.; and Jin, X. 2015. Semantic portrait color transfer with internet images. Multimedia Tools and Applications 1–19.
[20] Hertzmann, A.; Jacobs, C.; Oliver, N.; Curless, B.; and Salesin, D. 2001. Image analogies. SIGGRAPH Conference Proceedings.
[21] Li, C., and Wand, M. 2016. Combining markov random fields and convolutional neural networks for image synthesis. abs/1601.04589.
[22] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, in Advances in neural information processing systems, 2015, pp. 91–99.
[23] Alex J. Champandard. Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artworks. Available online:: https://arxiv.org/abs/1603.01768 (accessed on 10/04/2019)
[24] Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.