Dissecting Neural Style Transfer

Xi Du

Australian National University, Australia
u6559090@anu.edu.au

Abstract. [2]

Keywords: Neural network, Deep learning, Small data, Interpretability
.

1 Introduction

2 Literature Review

3 Method

3.1 Deep Neural Network

For the purpose of implementing and even extending neural style transfer algorithms, it is not necessary to understand how a neural network is trained, because we can use pretrained model for example vgg19 [**?**].

It is necessary to point out that a deep neural network that neural style transfer is concerend about is just a sequence of functions $F_1, F_2, F_3, ...$ whose inputs and outputs are all multi-dimensional arrays, as illustrated in Figure 1. Because

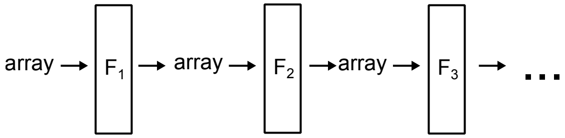

Fig. 1. A deep neural network

the arrays have usually more than two numbers of dimensions they are not really matrices. The arrays are called tensors (as in TensorFlow or torch.Tensor in PyTorch) but actual tensor algebra in a mathematical sense is rarely relevant. The functions need to be differentiable, which is handled automatically by modern deep learning frameworks. Other than that we can treat the functions as blackboxes because we are not concerned of training the model. A "layer" in a deep neural network refers to a few consecutive functions together. It is not particularly necessary to consider "layers" in this work. Though it is important to note that in our illustrations there are functions instead of layers.

3.2 vgg19 model

For the vgg19 model that most neural style transfer implementations available are based on, The inputs and outpus of each layer are all arrays with 4 dimensions. The word “dimension” here may mean something slightly different from what “dimension” means in for example “3-dimension vector”. Some people call the number of dimensions “rank” which would then raise another confusion with the rank of matrices. We give an illustration for the vgg19 model in Figure 2

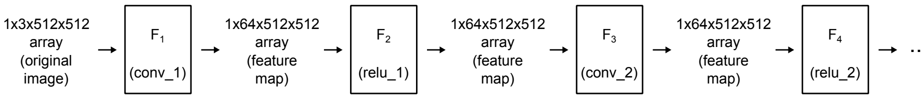

Fig. 2. The VGG19 model with an 512×512 RGB image as its input

The 1st dimension is the “batch” dimension, which means that you can put several images through the series of functions. In neural style transfer implementations it is usually just one image each time. So the 1st dimension is always 1, even between the F functions, throughout this work.

The 2nd dimension is the “feature” dimension. For a raw RGB image, it is 3. Intermediate arrays between, for example, F_3 and F_4, usually have a size much larger than 3, such as 64 or 128.

The 3nd and 4th dimensions are just spatial locations. For a 512×512 image they are 512 and 512. For intermediate arrays, these 2 dimensions are sometimes scaled down to half or even $\frac{1}{4}$ of the sizes of the image.

The functions $F_1, F_2, F_3, ...$ now have meaningful names such as relu_1, conv_2 now. [Not actually necessary to know but insert your elaborations here]

The arrays after the initial image are called feature maps, because their value means “how much a feature x exists at position y and x”. For example in a $1 \times 64 \times 512 \times 512$ feature map, it means there are 64 types of features. and its [1, 17, 111, 222] element, would then refer to the extent that the 17th type feature exists at the position of the 222th column of the 111th row. This “extent” could also be negative.

3.3 Vanilla neural style transfer

The vanillay neural style transfer algorithm [1] is structured as iteratively solving an optimisation problem. The argument to optimise is the image as a multidimensional array, of size $1 \times 3 \times 512 \times 512$ for example. The interative solver can be many, but in our case it is L-BFGS [Maybe elaborate on L-BFGS here]. The initial value of the argument could be either white noise or the content image, but the latter appeared to make the optimisation much easier.

The key issue here is still how to structure the optimisation target. The value to minimise is a linear combination of a “style loss” and “content loss”. Although these appeared to be two weights, actually only their ratio mattered. We simply fixed the weight of the content loss to 1 and leave the weight of the style loss as a adjustable hyperparameter k.

$$\underset{\text{image}}{\operatorname{argmin}}(kL_{\text{style}} + L_{\text{content}}) \tag{1}$$

Testing Phase

3.4 Data Set

Format and Preparation

(Not Really) Time-Series

Rationale

3.5 Hyperparameters and Experiments

For the vanilla style-transfer algorithm, there were a few hyper-parameters that affected the result.

- Number of iterations in the optimization. We did not have to expriment with different numbers of iterations though. It was more sensible to run the algorithm for more-than-enough iterations and store the intermediate results.
- Choice of initial values. Some sensible choices were:
  - White noise.
  - White.
  - Black.
  - The model mean.
  - The content image.
  - The style image.
  - The average of the content image and the style image.

Learning Rate

Hidden Layer Size

3.6 Evaluating Prediction

Binarisation and Choice of Loss

Amount of Network Reduction

Mechanism

Execution The program was developed under and are compatible with:

- Python 3.6
- PyTorch 1.0.1.post2
- CentOS 7 x86_64

To run the code, execute shell commands like

```
python36 0.py 5000
```

where 5000 is the number of desired training cycles to reach. The different hyperparameters will be automatically covered. The program automatically picks up stored models. To start fresh, clear the stored models and outputs by

```
rm out/*/*.*
```

but do not remove the directories.

4 Results and Discussion

5 Conclusion and Future Work

References

1. Gatys, L., Ecker, A., & Bethge, M. (2016). A neural algorithm of artistic style. Journal of Vision, 16(12), 326. doi:10.1167/16.12.326
2. Gedeon, T.D. and Harris, D. (1991) "Network Reduction Techniques," Proceedings International Conference on Neural Networks Methodologies and Applications, AMSE, San Diego, vol. 1: 119-126.
3. Penington, Jocelyn & Dow, R.J.F.. (1988). Neural net pruningWhy and how. Proceedings of the IEEE Conference on Neural Networks. 1. 325 - 333 vol.1. 10.1109/ICNN.1988.23864.
4. Doshi-Velez, F., & Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning.
5. Han, S., Mao, H., & Dally, W.J. (2016). Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. CoRR, abs/1510.00149.
6. Sun, X., Ren, X., Ma, S., Wei, B., Li, W., & Wang, H. (2018). Training Simplification and Model Simplification for Deep Learning: A Minimal Effort Back Propagation Method. CoRR, abs/1711.06528.
7. Cheng, Yu & Wang, Duo & Zhou, Pan & Zhang, Tao. (2017). A Survey of Model Compression and Acceleration for Deep Neural Networks.
8. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in PyTorch.