Compressed Deep Learning Model in Artistic Style Transformation on iOS

Chao-Ming Yen
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
cyen1@andrew.cmu.edu

Abstract

Recently it's shown that, with the proper use of convolutional neural network (CNN), the artistic style of an image can be extracted to combine with another image to create a new fine art that shares former's style and later's content [1]. In this project, I examine how compressed models can improve the speed of this kind of artistic style transformation, and how fast it can be when processed on mobile devices such as iPad2. The ultimate goal of this project is too see if the real-time artistic style can be realized based on existing techniques.

1. Introduction

A fine art consists of two distinct factors: style and content. Literally, it's trivial that the content of a fine art is what we see as it is, but the concept of style is somehow obscure. An artistic style could be treated as a higher-level abstraction of a fine art, and it's the style that differentiates painters. For example, we can easily tell the different artistic styles between Vincent van Gogh and Pablo Picasso, even if they both paint the same content. Amazingly, according to [1], the style of a fine art can be extracted with CNN in a forward pass. While it's well known that CNN is a powerful feature extractor, the concept of style requires some further definition based on feature map. Simply put, the artistic style of a fine art is defined as correlations among all the combinations of possible filter responses (activation values of feature map).

On the other hand, the CNN-encoded content is easier to understand, since feature maps encode most of distinct features for a given content, and one can verify two images by comparing their feature maps. Based on these metrics of both content and style encoded in CNN, given a fine art and a photo, one can randomly initiate a new image and optimize it by making its style close to fine art's style and its content close to the photo. The final image will become a new fine art, and it looks just like the painter (if exists) of the former fine art image just created a new masterpiece by paint the content according to the photo. I refer to this as "artistic style transformation."

2. Methods

2.1. Image representation in CNN

To further specify the encoding of image style and content in CNN, I follow the definition in [1] to represent encoded image style and content in CNN. Given $\vec{p}$ the input photo and $\vec{x}$ the image to be generated, the difference of content on layer l can be defined as square-error loss

$$L_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2}\sum_{i,j}(F_{ij}^{l} - P_{ij}^{l})^2 \quad (1)$$

where F_{ij}^{l} is the activation of ith filter at position (vectorized 2D image) j in layer l for image $\vec{x}$, and P_{ij}^{l} for photo $\vec{p}$. The derivative of loss with respect to the activations in layer i will be

$$\frac{\partial L_{content}}{\partial F_{ij}^{l}} = \begin{cases} F_{ij}^{l} - P_{ij}^{l} & if\ F_{ij}^{l} > 0 \\ 0 & if\ F_{ij}^{l} < 0 \end{cases} \quad (2)$$

On the other hand, given $\vec{a}$ the input fine art image and $\vec{x}$ the image to be generated, the style can be represented as correlations between different filter responses. Here, the feature correlations at layer l are given by Gram matrix, $G(\vec{a})_{ij}^{l} = \sum_k A_{ik}^{l} A_{jk}^{l}$, where A_{ij}^{l} is the activation of ith filter at position (vectorized 2D image) j in layer l for fine art image $\vec{a}$. Following the definition in [1], the difference of style between $\vec{a}$ and $\vec{x}$ can be taken as mean-square distance of two Gram matrices

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{i,j}(G(\vec{x})_{ij}^{l} - G(\vec{a})_{ij}^{l})^2 \quad (3)$$

and the total loss will be

$$L_{style}(\vec{a}, \vec{x}) = \sum_0^L w_l E_l \quad (4)$$

where w_l are weights to specify contribution from different layers. The derivative of E_l with respect to the activation in layer l thus become

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2}[(F^l)^T(G(\vec{x})^l - G(\vec{a})^l)]_{ij} & if\ F_{ij}^l > 0 \\ 0 & if\ F_{ij}^l < 0 \end{cases} \quad (5)$$

To generate a composed image, we can pose it as an optimization problem where we have to minimize the total loss from style and content. According to [1], the loss function we want to minimize for artistic style transformation given $\vec{p}$ (the input photo), $\vec{p}$ (the input fine art) and $\vec{x}$ (the image to be generated) is

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x}) \quad (6)$$

Here, the $L_{content}(\vec{p}, \vec{x})$ is the sum of loss of selected layer(s) $L_{content}(\vec{p}, \vec{x}, l)$. For example, the CNN used in [1] is VGG and the selected layer for content representation is 'conv4_2' and the selected layers for style representation are 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1' (set $w_L = \frac{1}{5}$ for L=1,2,3,4,5 and 0 for other layers). The α and β are weighting factors to decide the composition of content and style. Here, the ratio α/β is set to be 1×10^{-3}.

To minimize the loss, optimization algorithm such as SGD, L-BFGS or ADAM can be used. The following figure shows the example for 10 iterations of minimizing loss function as described in equation (6) using L-BFGS optimizer.

Figure 1: Example of artistic style transformation for 10 iterations. Top-left: input photo $\vec{p}$. Bottom-right: input fine art $\vec{a}$ (Pablo Picasso's self portrait in 1907). The generated image $\vec{x}$ is initialized to be the same as input photo.

2.2. Artistic style transformation on iOS

The artistic style transformation has been shown feasible on mobile device. There is already a popular app on app store called "Prisma," which is inspired by the same work in [1]. To implement the artistic style transformation on iOS, first we need a deep learning framework. In this project, I choice Torch as my main deep learning framework for its excellent ability in handling embedded systems, and there are lots of public available pre-trained torch models. For communication between Torch scripts and Objective-C, I use torch-ios though it's a little bit outdated and lack of maintenance.

Next, to get a workable CNN model, I decide to use the encoder part of SegNet [2], which is designed for pixel level semantic segmentation (figure 2). The encoder of SegNet is exactly the same structure of VGG used in ImageNet and same in [1]. The decoder layers are removed since they are not needed for artistic style transformation; only filter responses of forward pass are required. The reason why I choose semantic segmentation model over image classification model for feature extraction is because I assume the feature map learned with segmentation data should be more "informative" than one learned with classification data such as VGG in ImageNet.

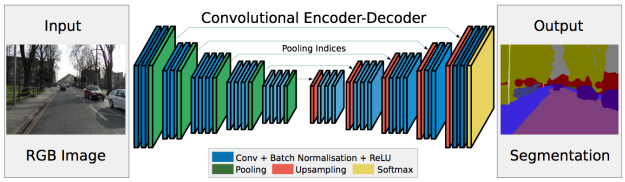


Figure 2: Model structure of SegNet as illustrated in [2]. The encoder structure is exactly the same as VGG.

Since currently there is no access of GPU on iOS granted for Torch, and the only way to access GPU is through Metal, which is very new and lack of available pre-trained models. For this reason and there is currently no software to communicate Torch and Metal, I decide to use CPU based scheme (library torch-nn, which uses BLAS of Accelerate Framework for fast matrix operations) for deep learning operations in this project, and slowness of processing speed is unfortunately inevitable.

The general scheme of the artistic style transformation on iOS is summed up in the following diagram (figure 3):

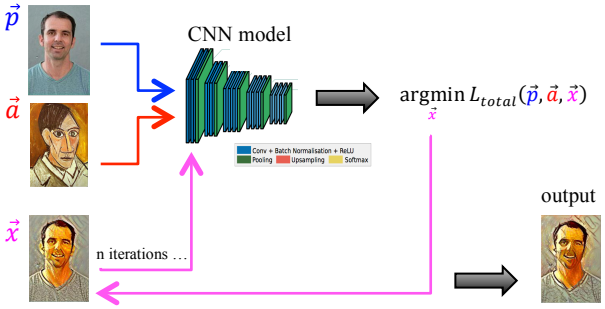


Figure 3: The general application scheme of artistic style transformation on iOS. The input $\vec{p}$ can be taken from camera and $\vec{a}$ is user choice.

2.3. Compression of deep learning model

When implementing artistic style transformation on iOS system, one must notice the restriction of limited memory and small disk space usage of the total application.

According to the implementation scheme of figure 3, the computation of artistic style transformation requires:

- 1 forward for input photo
- 1 forward for input fine art
- n forwards for minimization
- optimization cost for minimization process.

Therefore, if we can use a more economic model as feature extractor, it will not only increase the forward speed for totally n + 2 times of forwards, but also reduce the number of parameters in optimization process, which in turn make optimization faster. Based on this, I search some model compression techniques to find a more efficient compressed model in substitution of original VGG used in [1].

In this project, I experiment 2 different approaches for model compression. The first method is called "Knowledge Distillation" [3]. It uses a pre-trained model as a "teacher" to re-label training data into softmax label and retrains a "student" model using the softmax label and original training data. The soft label can be further "softened" by adjusting the temperature parameter T according to [3] :

$$q_i = \frac{\exp(z_j/T)}{\sum_j \exp(z_j/T)} \qquad (7)$$

The student model is the compressed version of teacher model, since lesser feature maps and neurons are needed for student model to reach the same accuracy as teacher model for this training scheme. In other words, the knowledge is "distilled" from teacher model through the softmax relabeling, and the student model learns on the distilled knowledge instead of original hard label. Figure 4 illustrates the general idea of knowledge distillation and its training on semantic segmentation data.

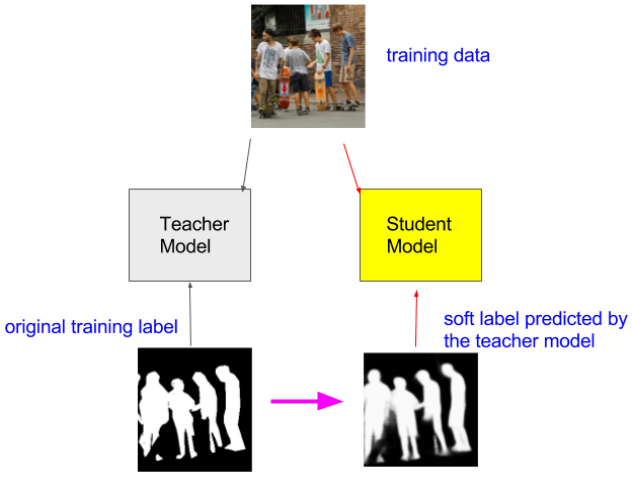
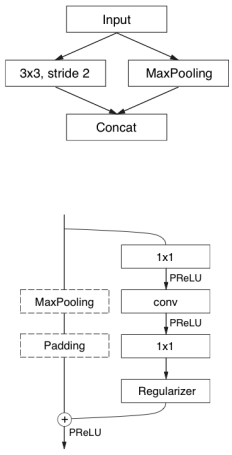


Figure 4: The idea of knowledge distillation. The soft label is the softmax with temperature T (see equation (7)).

The second model compression method is by proposing a more efficient architecture than VGG. The proposed ENet in [4] is a SegNet replacement designed for embedded system, which saves ~80 times of parameters compared to the original model in [2]. The ENet exploits early down-sampling and use dilated convolution on the bottleneck layers [5] to save the computational cost. See figure 5 and [4] for more detail about the ENet architecture.



Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
$4\times$ bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

Figure 5: The ENet model. Top-left: initial module. Bottom-left: the bottleneck module. The bottleneck 4.X and 5.X are decoders, which are unnecessary and removed in this project.

3. Results

3.1. Compression of deep learning model

For the training data, I use 10,000 pieces of the MS-COCO dataset [6] of human segmentation. In knowledge distillation method, the student mod is "taught" by the original pre-trained SegNet model but with only $\frac{1}{8}$ number of feature maps than original VGG encoder. In ENet model, same training dataset is used (see table 1).

	VGG	ENet	Distilled VGG
#parameters	7,444,896	181,506	263,046
Accuracy (%)	95.13	94.59	92.93

Table 1: The training results of compressed models and the original VGG. The accuracy is evaluated by computing the pixel-wise error on 1,000 validation data. Notice that the #parameter only counts the encoder part of model which is relevant in artistic style transformation, but the accuracy is evaluated using the whole model (encoder + decoder).

After the compressed models are ready, I tested them for artistic style transformation and get the following result (figure 6):

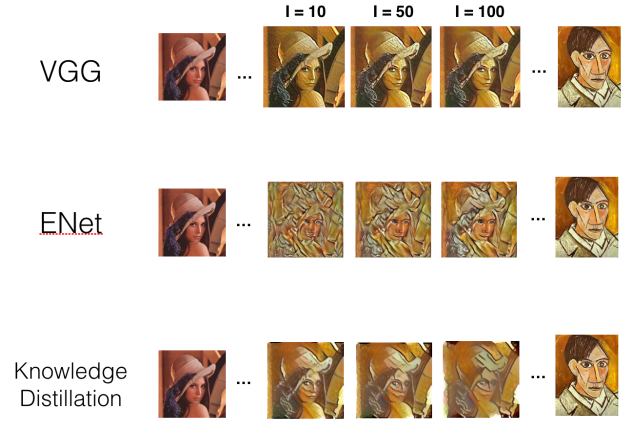


Figure 6: Comparison of results of artistic style transformation with different models. "I" indicates the number of iteration.

Though it's hard to define a metric to measure the how better the result is, we can still observer that the result of ENet might be less favored by users. On the other hand, the distilled model seems to work pretty well and converge faster than the original VGG (the result at iteration = 10 of distilled model is similar to the result at iteration = 100 of VGG model).

3.2. Speed test on iOS

To test the speed of compressed models on iOS for artistic style transformation tasks (see figure 3 for the general scheme of implementation), I randomly generate 256 x 256 images for input photo and fine art, test 100 times and measure the average running time of :

1. Forward, and
2. Single Iteration step of optimization (which contains one forward, see figure 3. and section 2.3 for more detail)

The result is shown in figure 7:

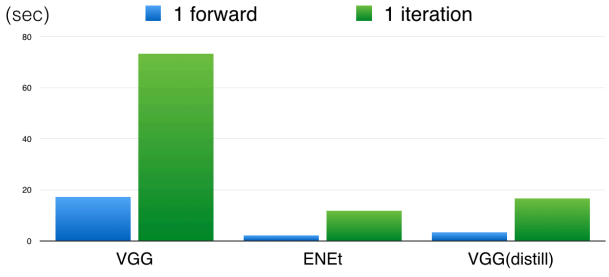


Figure 7: Test of compressed model on iPad2 for artistic style transformation. The optimizer used here is L-BFGS.

As shown in figure 7, the running speed of non-compressed VGG is painfully slower, maybe it's because it runs on CPU only. On the other hand, both compressed model seem to have a better running speed. Consider the artistic style result in figure 6, I recognize distilled VGG as my best model for artistic style transformation.

4. Discussion

In this project, only model compression is evaluated regarding to the speed of artistic style transformation. As the result shown in figure 7, I also notice that the speed of optimizer seems to be more important than the CNN model (iteration time takes longer than 2 x forward time), which is out of my expectation. In conclusion, if I am going to make a "real-time" artistic style transformation application on mobile devices, there are things other than model compression need to be considered.

The first is the ability of accessing GPU for deep learning operations on iOS. In this project, I only use CPU to forward CNN and perform optimization, which is very slow.

The second thing is I need to think a faster way for finding the solution of optimization problem as shown in equation 6. One possible solution is to use a bigger-step and perform only like 1 or 2 iterations to give the final result. Another possible solution is to try a more efficient optimizer (In this project I use L-BFGS for all cases).

The last thing is about XOR-Net, I haven't tried it out for this project because artistic style transformation is a little bit different from learning a model. But there might still be some possibility that binarized weight can take advantage in the optimization process.

References

[1] L. A. Gatys, A. S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. CVPR, pp. 2414-2423, 2016.

[2] V. Badrinarayanan, A. Handa, R. Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling. arXiv:1511.00561, 2015.

[3] G. Hinton, O. Vinyals, J. Dean. Distilling the Knowledge in a Neural Network. arXiv:1503.02531. 2015.

[4] A. Paszke. A. Chaurasia, S. Kim, E. Culurciello. ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation. arXiv:1606.02147. 2016.

[5] M. Lin, Q. Chen, S. Yan. Network in Network. arXiv:1312.4400. 2014.

[6] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, P. Dollár. Microsoft COCO: Common Objects in Context. arXiv:1405.0312. 2014.