Deploy Style Transfer on the Android Devices

1st 曾泓硯
Dept. of Engineering and Electronics
National Tsing Hua University
Taiwan
tony871107@cloud.nthu.edu.tw

2nd 廖宏儒
Dept. of Engineering and Electronics
National Tsing Hua University
Taiwan
hankliao1998@cloud.nthu.edu.tw

***Abstract*—Mobile devices equipped with Machine Learning and Deep Learning models are going to be the mainstream in the future. Among these models, we are interested in image style transfer. We aim to deploy PyTorch models to Android Platforms by TVM. Compare to our method, Tensorflow can also be deployed to different platforms by Tensorflow Lite. Although examples of style transfer in Tensorflow Lite haven't been released in public, we can still get benchmarks to compare with.**

***Index Terms*—style transfer, PyTorch, TVM, Android**

I. Introduction

Deploying style transfer on android devices can be separated into three parts – style transfer model, TVM, and Android. First, we apply PyTorch to construct our Image Transform Net and Loss Network to get good quality style transfered image. Second, we utilize TVM to cross complie our PyTorch model into ARM64 binary files. Last but not least, we implement Android program based on the application programming interface that TVM provided.

Comparing to the benchmark that TensorFlow lite released in its project homepage (500ms to transfer one image) [1], our app spends about two seconds to transfer one image. However, we haven't improved our performance by using GPUs on the smart phones. Besides, the smart phones that we experiment on (Samsung S7 Edge) is different to the benchmark (Pixel 4, iPhone XS).

II. Methodology

A. Style Transfer Model - PyTorch

We refer to the method by Justin Johnson for Perceptual Losses for Real-Time Style transfer [2] to train our model, which consist of two part of networks: Image Transform Net and Loss Network. We convert our model files into onnx files. (Source code)

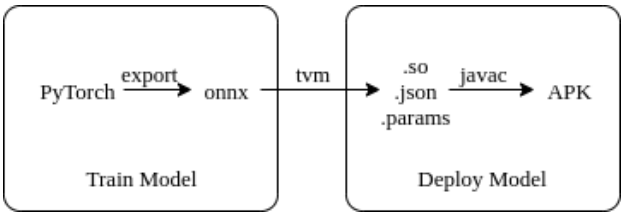

Fig. 1. Our work flow.

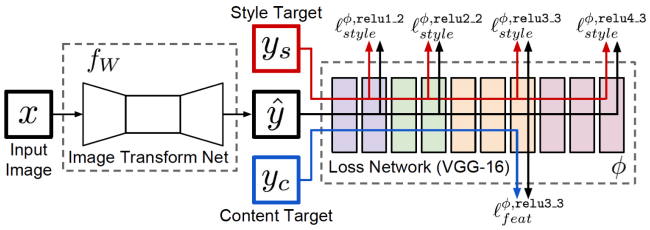

Fig. 2. Model of Perceptual Loss. [2]

- Image Transform Net:
  It consist of several convolutional layers and residual layers. We downsample the image by using convolutional layers, passing spatial information by residual layers. As for upsampling, we refer to the method by Wenzhe Shi for Is the deconvolution layer the same as a convolutional layer? [3] by combining interpolation, padding and convolutional layers. [4])
- Loss Network:
  It evaluates the content loss and style loss, the main idea is to utilize pre-trained VGG-16 network as loss function to calculate mean square error between result from transform network and input image, style image respectively. In VGG-16, there are four relu layers, and each of layers content different high dimensional information. In our project, we fetch the result from second relu layer and calculate Gram matrix to extract style feature from content feature. [2]

B. TVM

TVM (https://tvm.apache.org) is an open deep learning compiler stack for CPUs, GPUs, and specialized accelerators. It aims to close the gap between the productivity-focused deep learning frameworks, and the performance- or efficiency-oriented hardware backends. TVM cross compiles our onnx files into arm-64 binary files. (.so, .json, .params)

TABLE I
PERFORMANCE COMPARISON

Solution	Model Size	Device	CPU Time
Our solution	6 Mb/model	Samsung S7 Edge	2000ms
TensorFlow Lite [1]	0.2 Mb/model	Pixel 3	540ms
		Pixel 4	405ms
		iPhone XS	251ms

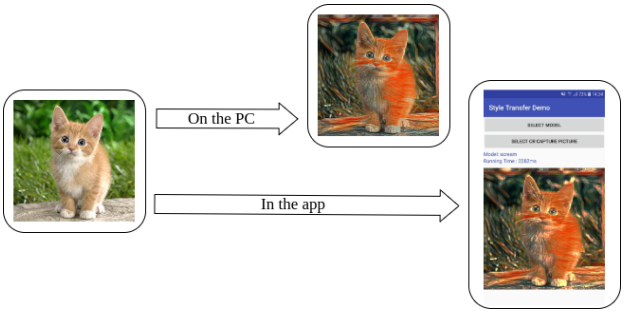

Fig. 3. The Figure shows that the results that we get from computer are as same as from our APP.

C. Android

We implement application based on the application programming interface that TVM provided. Firstly, we import module into our application by loading .so, .json, .params files. After that, we prepare the input of the model by resizing the input image to the size of our input model. Then we invoke the model. Afterwards, we adjust the output from the model by clipping the output to the limit of value of the standard ARGB.8888, and resizing the output image back to the original size. In the end, we show the image in the application.

III. EXPERIMENTAL RESULTS

Table I shows that both our performance and model size are worse than the benchmark on the style transfer example homepage of TensorFlow Lite.

The CPU time of the TensorFlow Lite is calculated on the application which used four threads. And it use a brand-new smart phone that there is no background program running. But we only run the model with one thread. And our smart phone is installed with several app which run some tasks in the background. The background program might increase the running time of the model. Therefore, the running time of our method might not be slower than that of TensorFlow Lite under same circumstances.

IV. CONCLUSION

We deploy the style transfer model to the Android device successfully. The performance of running the model on the Android is two seconds per image.

There are several drawbacks in our flow: Firstly, training model is time-consuming which takes approximate 6 hour for one style transfer model. Maybe we can modify the structure of the model to decrease the training time. On the other hand, prediction takes some time too. Perhaps, we can try to use the GPU to run the model on the mobile.

REFERENCES

[1] Artistic Style Transfer with TensorFlow Lite: https://www.tensorflow.org/lite/models/style_transfer/overview
[2] Justin, J., Alexandre, A. and Fei-Fei, L.: "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". ECCV (2016)
[3] Wenzhe, S., Jose, C. and Lucas, T.: "Is the deconvolution layer the same as a convolutional layer?".
[4] Odena, et al., "Deconvolution and Checkerboard Artifacts", Distill, 2016. http://doi.org/10.23915/dist