A Comparative Evaluation of Deep Learning Frameworks

CSI 680: Master's Project

Author: Jinyu Tian

Advisor: Jeong-Hyon Hwang

Department of computer science,

University at Albany – State University of New York

jtian4@albany.edu

GitHub: https://github.com/JinyuJinyuJinyu/MasterProject

December 5, 2020

Abstract:

As development of deep learning, there is a growing number of computing libraries, tools and frameworks. These frameworks have different designing philosophies. For example, TensorFlow uses static Computational Graphs while PyTorch uses dynamic Computational Graphs. So, some frameworks may perform better than others in a specific case. I tried to compare TensorFlow and PyTorch on vgg16 and resnet18. I have found that TensorFlow is better on vgg16 in terms of speed, while PyTorch is better on resnet18 in terms of accuracy and speed.

Table of Contents

1. Introduction

As deep learning develops, more and more applications merge as well as frameworks, such as TensorFlow, PyTorch. These frameworks are designed and working in different philosophies. TensorFlow is designed by Google and adopts a dataflow graph model. PyTorch has large community support and its goal is to achieve maximum flexibility and speed. They have significant differences as they use different hyper parameters. Some existing benchmarks, as Yanzhao Wu et al. [1] pointed out, have not taken a holistic approach to study the impact of computing libraries on deep learning frameworks performance.

To find out better frameworks in specific cases, I have tried to compare these two frameworks with vgg16 and resnet18.In this work, I used vgg16 and resnet18 models to run in frameworks TensorFlow and PyTorch separately. Two models are different in how layers are connected. Vgg is a 'plain' neural network, but for renset, it is a residual network, layers can be skipped by 'shortcut connection'. For datasets, I used CIFAR10[2] and ImageNet [3]. CIFAR10 is 32x32 colors image dataset and contains 80 thousand images with 10 categories. ImageNet is a hierarchical color image dataset.

2. Background

Paper 'DAWNBench: An End-to-End Deep Learning Benchmark' [4] proposed DAWNBench, a benchmark focuses on TTA (Time-To-Accuracy). It shows how different optimizers, stochastic depth, hyperparameters, multi-GPU training impact the training performance.

Paper 'Metrics for Machine Learning Workload Benchmarking' [5] argued that TTA of MLPerf benchmark is not enough to give a full view of machine learning performance. It proposed a new metric, ATTMT, to benchmark multi thresholds. It is unlike TTA sensitive to quality threshold, like seed value.

Paper 'Benchmarking neural network robustness to common corruptions and perturbations'[6] introduced IMAGENET-P dataset. Classifiers can perform well on a clean dataset, but it does not guarantee that classifiers will have the same performance on a

perturbation dataset. The IMAGENET-P dataset is a set of perturbed or subtly differing ImageNet images. It allows to benchmark model's perturbation robustness.

Paper 'Benchmarking State-of-the-Art Deep Learning Software Tools' [7] compared popular deep learning frameworks running performance on different specs of platform. It has made two contributions: first one is that the benchmark results can be used as a guide for selecting platforms; Second one is that the benchmark results point out the possible way to optimize running performance.

3. Environment and method

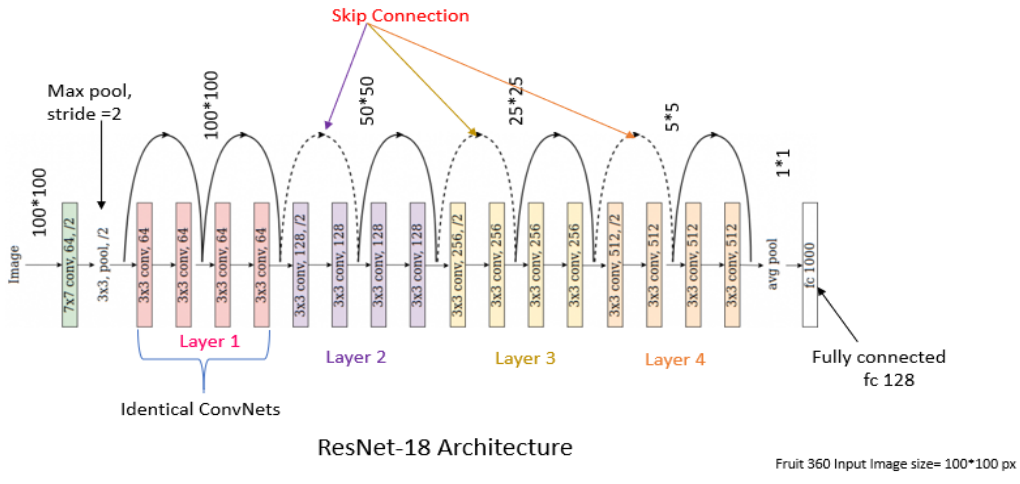
3.1 environment

Ubuntu 18.04 OS. Python 3.6. TensorFlow 2.1.0 version. PyTorch 1.4.0 version. CPU i7-8750H, GPU GTX1060 Max-Q. i7-8750H base frequency has 2.20 GHz with bus speed 8 GT/s. GTX1060 Max-Q has 1280 CUDA cores and 192-bit bus width.

3.2 Measurements

3.2.1 Models

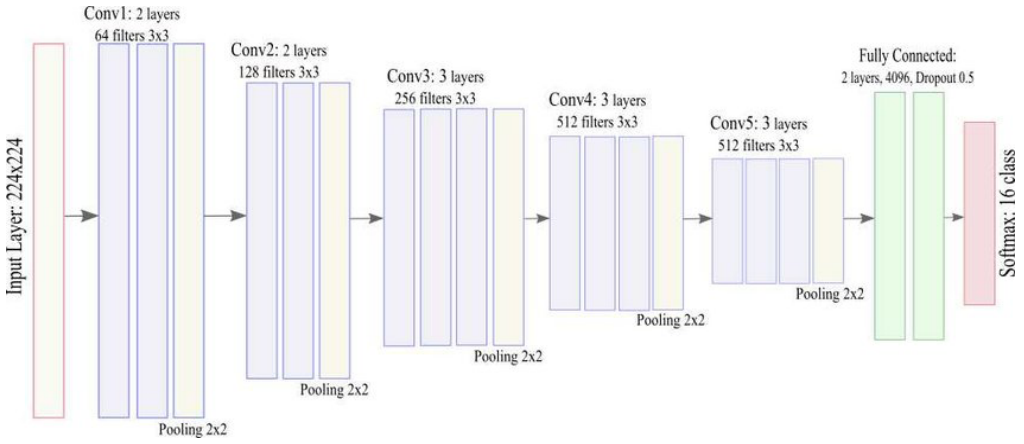
Resnet18[8] (resnet_tr.py, resnet_tf.py):



Resource: www.pluralsight.com/guides/introduction-to-resnet

The residual network is proposed by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. The goal of deep residual learning is to solve vanishing or exploding gradients in deep neural networks by adding “shortcut connections” between layers. The Resnet configuration has 18, 34, 50, 101,152 weight layers, the width of the convolution layer starts from 64 to 2048. For this work, Resnet18 is used.

VGG16[9] (vgg16_tr.py, vgg16_tf.py):



Resource: www.researchgate.net/figure/VGG16-Architecture-with-Softmax-layer-replaced-used-as-the-base-model-for-each-classifier_fig1_322787849

VGG is proposed by K. Simonyan and A. Zisserman. Their model can achieve 6.8 % top-5 test error in Image-Net. The model is designed for a fixed size 224x224 RGB image and contains 11-19 weight layers. The width of the convolution layer starts from 64 in the first layer, and then width is increased by a factor of 2, to 512 width.

3.2.2 measure method

In measuring time, I use Python built-in function time.time(). The first epoch is skipped, because the framework may take some time to initialize the model in the first epoch. To measure times, start timing at the beginning of the training loop, and timing validation loop separately. Total training time will be total time subtract validation time.

In measuring accuracy, since I need the confusion matrix to visualize model prediction results, I calculate the trace of confusion matrix for validation accuracy.

```
model_validation():
 val_logits = resnet18(x_batch_val, training=False)
 y_onehot_val = tf.one_hot(y_batch_val, depth=10)
 prob = tf.nn.softmax(val_logits,axis=1)
 preds = tf.argmax(prob, axis=1)
 preds = tf.cast(preds, dtype=tf.int32)
 mtx = tf.math.confusion_matrix(y_batch_val, preds, num_classes=classes)
 accu = tensor.trace(mtx)
```

```
total_time = time()
for epoch in epochs:

  model_train()

  val_time = time()
  model_validation()
  validation_time = time()-val_time()

 training_time = total_time - validation_time
```

4. Results

4.1 Resnet18:

In training resnet18, I use batch size 64, epoch 200, learning rate 0.001. In the figure 1, either optimizer SGD or Adam of PyTorch has faster converge rate than TensorFlow. And the model trained in PyTorch with optimizer Adam has the highest accuracy of 83.07%. In this case, TensorFlow in general has slower converge rates and its both optimizers validation accuracy are less than PyTorch. In the figure 2, 3, 4, PyTorch has less time in training, inference and first epoch than TensorFlow. PyTorch is 18% faster than TensorFlow in training, 58.7% faster in inference, 18.5% faster in first epoch. TensorFlow took an estimated 8.68 seconds longer than its average training time. However, PyTorch took an estimated 2.6 seconds longer than its average training time.

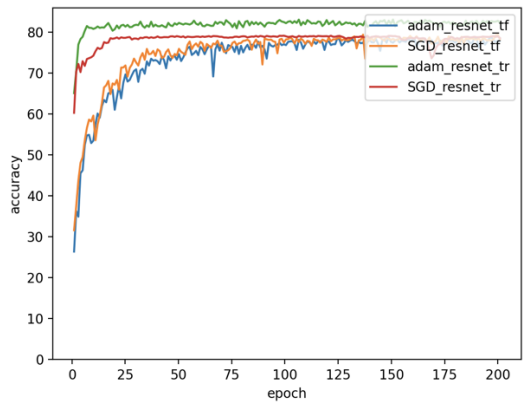


Figure 1

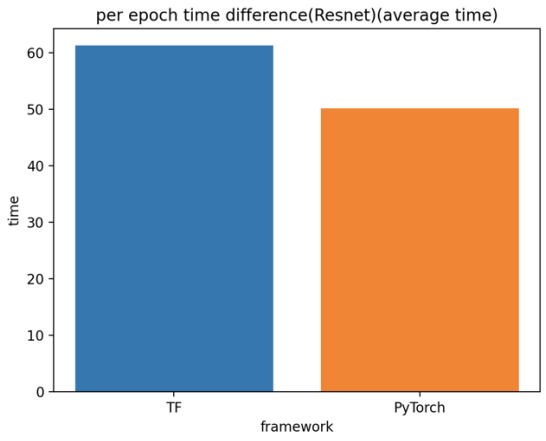


Figure 2

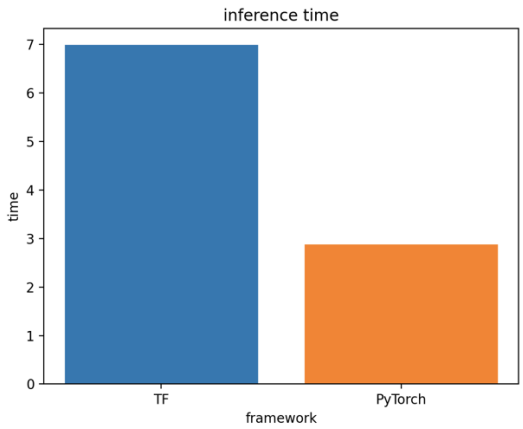


Figure 3

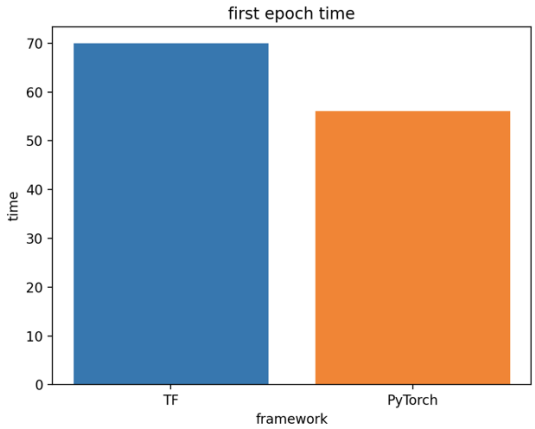


Figure 4

4.2 VGG16:

In training vgg16, I use batch size 32, epoch 200, learning rate 0.001. The reason to adjust batch size from 64 to 32 is limitation of GPU memory. In figure 5, training accuracy is very poor. There might be two reasons. One is that the training image is downsized to 80x80 not 224x224, because there is not enough RAM to load images. another one is only used in a very little part of the dataset (training dataset, validation, test dataset). Validation. In figure 6, 7, 8, TensorFlow outperformed PyTorch in training and inference. TensorFlow is 58.3% faster than Pytorch in training, 29.3% faster in inference. PyTorch took an estimated 17.9 seconds longer than its average training time.

However, TensorFlow took an estimated 14.54 seconds longer than its average training time.

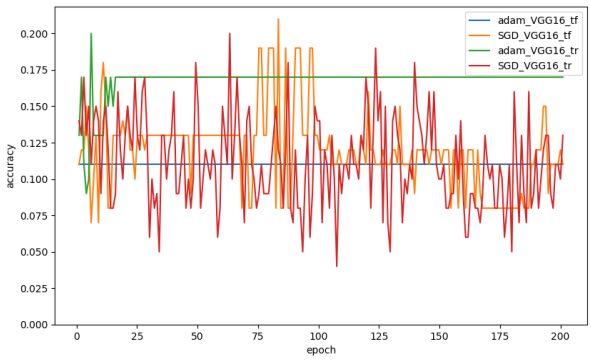


Figure 5

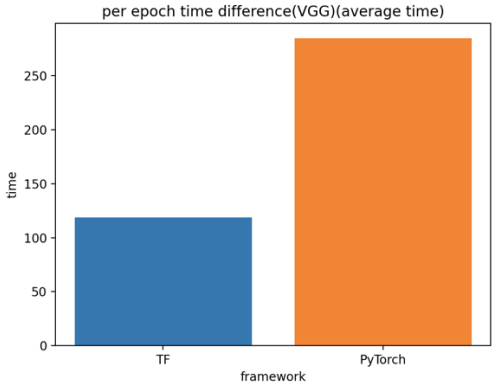


Figure 6

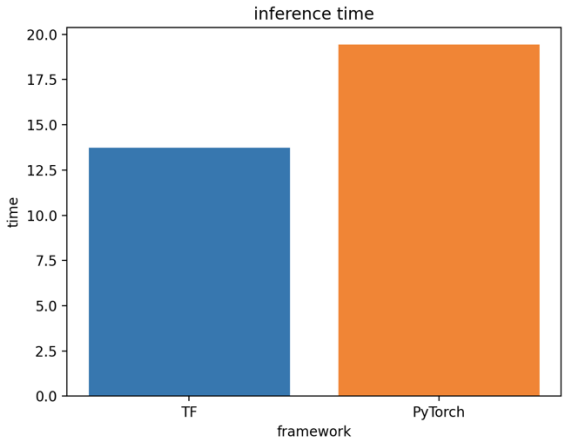


Figure 7

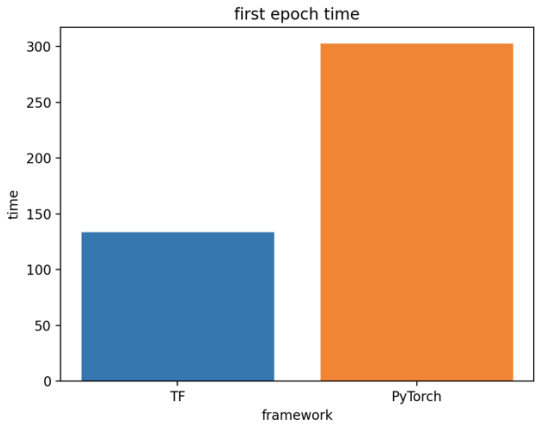


Figure 8

4.3 Summary

To sum up, PyTorch has better performance in resnet18 than TensorFlow, but TensorFlow is better than PyTorch in vgg16. In the training accuracy of Resnet, I observed that PyTorch in general has better performance than TensorFlow. To find the reason, I tried to set seed and global float point precision in the same value. The results remain the same, Pytorch has better performance. One reason might be that the loss functions are not equivalent, the loss function in PyTorch is CrossEntropyLoss(), the loss function in TensorFlow is categorical_crossentropy().

5. Limits

There are some limitations of this work. For example, limited GPU RAM, I do not have enough time to fix problems and limited experience on training neural networks. Basically, limitations can be sorted into two groups. One is training, another one is implementation.

5.1 Training

1. Fail to reproduce state of art accuracy.

For Resnet training, should strictly stick with the training method proposed in Deep Residual Learning for Image Recognition. Using weight decay of 0.0001, initial learning rate of 0.1, momentum of 0.9, batch size of 128. Because of the limits of GPU memory, I only used batch size 64. I did not set a weight decay of 0.0001, instead of a learning rate of 0.001.

For Vgg16, in order to reproduce the same result on paper, image dimension 224x224 is needed. Since limited RAM could not load all train images, I have downsized images to 80x80. Besides, considering very long training time, I only used the ILSVRC-2010 validation dataset for training. Moreover, data augment is needed according to the paper, such as random horizontal flipping and random RGB colour shift.

2. TensorFlow GradientTape and model.fit converge rates different

GradientTape is a low-level API, model.fit is a high-level API to train neural networks. In this work, models are trained by the GradientTape method. When compared with training results of model.fit, observed a significant difference in accuracy on Vgg16. The model.fit API has better accuracy than GradientTape. I did not find out the reason caused this difference.

In case of training resnet18, as figure 9 shows, the model.fit() function converges faster than GradientTape before epoch of 20. However, the highest accuracy achieved by model.fit() is 72% accuracy which is 7.3% less than GradientTape.

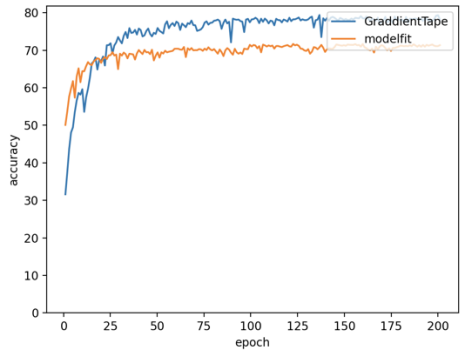


Figure 9

5.2 Implementation

I did not successfully implement a profiler in TensorFlow. When I ran code with a profiler (Tensorboard), I encountered the 'CUPTI_ERROR_INSUFFICIENT_PRIVILEGES' error. In order to measure training and inference time, I use Python built-in function time.time(). Tensorboard makes visualization and tooling in machine learning more convenient. It enables developers to visualize metrics, model graphs, histograms of weights, biases, etc.

6. Future works to do

1. Change the training schemas

   For training resnet18 and vgg16, I did not strictly follow the approaches provided by authors. For training vgg16, I did not use the full dataset, because of limited computational resources. For the next step, training schemas and dataset will be resolved.

2. VGG16 training on CIFAR10[10].

   I have tried training vgg16 on CIFAR10, but the results are very bad. The paper proposed an approach that modifies the vgg network to train on CIFAR dataset. The approach is to add 'dropout' and batch normalization layers after convolution layers. The dropout rate is set from 0.3 to 0.5.

3. To implement Tensorboard

   During this work, I did not have enough time to solve all issues. In the future, I can try different environments and approaches to solve the 'CUPTI_ERROR_INSUFFICIENT_PRIVILEGES' error.

4. Professional machine

Training models on a laptop is not time efficient and RAM is a bottleneck to use large image dataset. If using a professional machine, the problem of limited RAM, long training time can be solved easily.

7. Conclusion

As results show, TensorFlow and PyTorch have advantages in training different models. TensorFlow has better performance on vgg16, faster training and inference speed. PyTorch has faster training, inference speed and higher accuracy on resnet18.

There are many limitations of this work. For example, fail to reproduce state of art accuracy, need to adjust training schema. Still more work needed to be done in the future.

8. Reference

[1] Y. Wu et al., "A Comparative Measurement Study of Deep Learning as a Service Framework," in IEEE Transactions on Services Computing

[2] A. Torralba, R. Fergus and W. T. Freeman, "80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 11, pp. 1958-1970, Nov. 2008

[3] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, 2009, pp. 248-255

[4] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, Matei Zaharia, "DAWNBench: An End-to-End Deep Learning Benchmark and Competition," available: dawn.cs.stanford.edu/benchmark/

[5] Snehil Verma, Qinzhe Wu, Bagus Hanindhito, Gunjan Jha, Eugene B. John, Ramesh Radhakrishnan, Lizy K. John, "Metrics for Machine Learning Workload Benchmarking," available: researcher.watson.ibm.com

[6] Dan Hendrycks, Thomas Dietterich, "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations," ICLR 2019, available: arxiv.org/abs/1903.12261

[7] S. Shi, Q. Wang, P. Xu and X. Chu, "Benchmarking State-of-the-Art Deep Learning Software Tools," 2016 7th International Conference on Cloud Computing and Big Data (CCBD), Macau, 2016, pp. 99-104

[8] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778

[9] Karen Simonyan, Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2015, available: arxiv.org/abs/1409.1556

[10] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," 2015 3rd IAPR Asian Conference on pattern Recognition (ACPR), Kuala Lumpur, 2015, pp. 730-734