VGG16 COMPRESSION TECHNICAL REPORT

Hoang Tran Ba

hoangtb203@gmail.com

Abstract

This report shows some common techniques and its application to reduce size to model. Using VGG16 as base model and CIFAR100 as dataset, I test several techniques to compress the VGG16 while keeping the error as small as possible. Knowledge distillation shows high potential but the results is not as expected. Quantization shows the best performance despite of small change in response time. Pruning shows the most ineffectiveness that make the model predict randomly with a small number of cutting nodes.

1 Introduction

In the recent years, remarkable advancements in hardware, such as Graphics Processing Units (GPUs), and the emergence of cutting-edge deep learning techniques have led to the creation of large and powerful AI models. These models have practical applications that support the human in unique ways. However, deploying these large models on common devices can be challenging. By employing effective size reduction techniques, it is possible to make AI models more efficient without compromising their accuracy significantly. This report aims to optimizing AI models and achieving the best possible trade-off between model size and performance.

2 Techniques

2.1 Knowledge distillation

Knowledge distillation refers to the process of transferring the knowledge from a large unwieldy model or set of models to a single smaller model that can be practically deployed under real-world constraints.

Knowledge distillation is performed more commonly on neural network models associated with complex architectures including several layers and model parameters. Therefore, with the advent of deep learning in the last decade, and its success in diverse fields including speech recognition, image recognition, and natural language processing, knowledge distillation techniques have gained prominence for practical real-world applications.

As shown in Figure 1, in knowledge distillation, a small "student" model learns to mimic a large "teacher" model and leverage the knowledge of the teacher to obtain similar or higher accuracy.

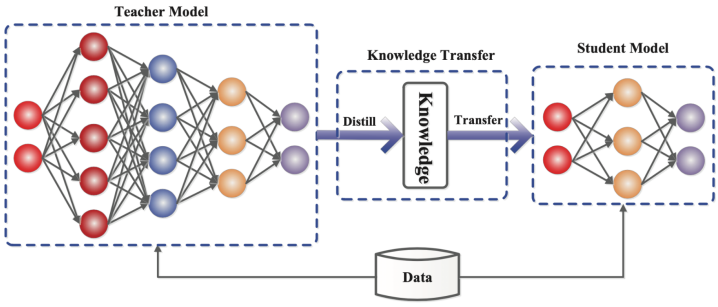

Figure 1: Knowledge Distillation

2.2 Quantization

Quantization improves performance and power efficiency by reducing memory access costs and increasing computing efficiency. Lower-bit quantization requires less data movement, which reduces memory bandwidth and energy consumption. Additionally, mathematical operations with lower precision consume less energy and improve computation efficiency, thereby reducing power consumption.

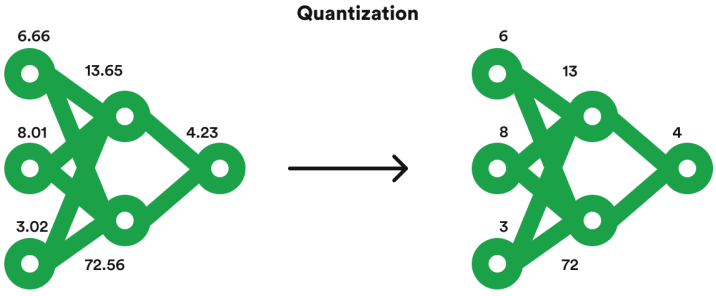

Figure 2: Quantization

2.3 Pruning

State-of-the-art deep learning techniques rely on over-parametrized models that are hard to deploy. On the contrary, biological neural networks are known to use efficient sparse connectivity. Identifying optimal techniques to compress models by reducing the number of parameters in them is important in order to reduce memory, battery, and hardware consumption without sacrificing accuracy. This in turn allows you to deploy lightweight models on device, and guarantee privacy with private on-device computation.

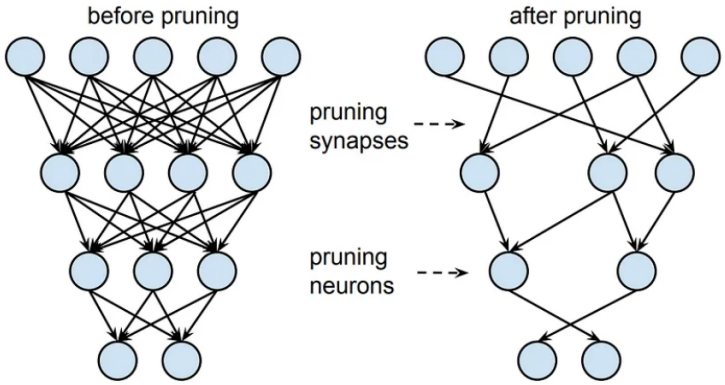

Figure 3: Pruning

3 Dataset

3.1 CIFAR100

This dataset has 100 classes containing 600 images each capturing things in the outside world. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. As a result, 60000 images is contained inside this dataset, make it large enough for my experiment.

CIFAR-100 is available at `https://www.cs.toronto.edu/~kriz/cifar.html`

3.2 Data pre-processing processes

Since PyTorch includes and splits CIFAR100 in its framework, I just have to call both the train and test data set within few lines of code. After that, I apply normalization transform to the tensors to accelerate the upcoming training process. Total, there are 50000 samples

Table 1: Platforms information

Name	GPU	Pytorch	CUDA
SageMaker Studio Lab	NVIDIA T4	2.0.1	11.8.89
Google Colaboratory	NVIDIA T4	2.0.1	11.8.89
Kaggle	NVIDIA Tesla P100	2.0.0	11.8.89

for training and 10000 samples for testing. In this experiment, validation samples are not in use.

4 Approach

In this experiment, I decide to approach from most-effort method to least-effort method, since I want to use the combination of all strategies mentioned. As a result, the step would be: Knowledge distillation, Quantization, then Pruning.

Since this experiment needs to have GPU to accelerate the process, I use Amazon SageMaker Studio Lab, Google Colaboratory, and Kaggle Platform to train the model and edit the code. Switching between platforms take time and redundant process, but this is the best way I can do as a student with no resources at all. Pytorch, CUDNN, and GPUs are different in each platform, but it's overcame with installation instructions.

Firstly, I choose VGG16 as my baseline model and train it to get the baseline scores. VGG16 refers to the VGG model, also called VGGNet. It is a convolution neural network (CNN) model supporting 16 layers. It is one of the top models from the ILSVRC-2014 competition. This model has 34.0M parameters and takes about half of a minutes for an epoch using NVIDIA T4. It allows me to try more ideas within limited time available.

Secondly, to find the ideal "student model", I make some different models with distinct number of parameters. These small models still use Convolution Neural Network and Linear Layer as components. After that, they are trained using modified loss function that take into account of both label and teacher model's outputs. I also change the optimizer to Adam to achieve faster converge.

Finally, pruning and quantization are careful examined to ensure that the different of top 1 error between current model and VGG16 is not larger than 4%.

5 Metrics

There are metrics to evaluate how well the model classifying the images. In terms of correctness, the main metric used is Top 1 error. This is the percentage of miss-classifying images on the test set. Also, I measure the top 5 errors as additional metric. This metric will decrease if the correct label is within the top five highest percentage labels, and increase otherwise.

$$\text{top 1 err} = \frac{1 - \text{correct predict label}}{samples}$$

Figure 4: Formula of the top 1 error

On the other hand, time response is considered to measure how long the model take to return an output for a given input. Since every sample generally take less than a second, Profiler is installed and used.

6 Results

6.1 Knowledge distillation

6.1.1 The first experiment

I have created some models and train them using specialized loss function. Motivated by the information gained in technical report of singh96aman, the loss function has the following formula:

$$\frac{1}{m}\sum_{j=0}^{m}\left\{2T^2\alpha D_{KL}\left(P^{(j)}, Q^{(j)}\right) - (1-\alpha)\sum_{i=1}^{c} y_i^{(j)} log\left(1-\widehat{y}_i^{(j)}\right)\right\}$$

Figure 5: Formula of the loss function

In this equation, D_{KL} is Kullback–Leibler divergence, m is total number of samples; P are the soft labels from teacher model, Q are student's softmax scores. I set the variables Temperature (T) = 1 and alpha = 0.9, which are the same in that article. After hours of training, the result comes with surprised results (see in Table 3). The 4th largest model (I call it 'v2') brings the best top 1 error score, while the largest model is disappointing. It is easy that sometimes larger model doesn't mean better results.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MyCompressNet2                           [1, 100]                  --
├─Conv2d: 1-1                            [1, 64, 32, 32]           1,792
├─BatchNorm2d: 1-2                       [1, 64, 32, 32]           128
├─Conv2d: 1-3                            [1, 128, 16, 16]          73,856
├─BatchNorm2d: 1-4                       [1, 128, 16, 16]          256
├─Conv2d: 1-5                            [1, 128, 16, 16]          147,584
├─BatchNorm2d: 1-6                       [1, 128, 16, 16]          256
├─Conv2d: 1-7                            [1, 256, 8, 8]            295,168
├─BatchNorm2d: 1-8                       [1, 256, 8, 8]            512
├─Conv2d: 1-9                            [1, 256, 8, 8]            590,080
├─BatchNorm2d: 1-10                      [1, 256, 8, 8]            512
├─Conv2d: 1-11                           [1, 256, 4, 4]            590,080
├─BatchNorm2d: 1-12                      [1, 256, 4, 4]            512
├─Conv2d: 1-13                           [1, 256, 4, 4]            590,080
├─BatchNorm2d: 1-14                      [1, 256, 4, 4]            512
├─Linear: 1-15                           [1, 1024]                 4,195,328
├─Linear: 1-16                           [1, 1024]                 1,049,600
├─Linear: 1-17                           [1, 256]                  262,400
├─Linear: 1-18                           [1, 256]                  65,792
├─Linear: 1-19                           [1, 128]                  32,896
├─Linear: 1-20                           [1, 100]                  12,900
==========================================================================================
```

Figure 6: The architecture of v2 model

Table 2: Model results

Name	Top 1 error	Top 5 error	Epochs	Parameters	Size of file
VGG16	0.2887	0.1054	200	34,015,396	132.9MB
v1	0.4159	0.1553	200	5,613,668	21.4MB
v2	0.3743	0.1640	200	7,910,244	30.2MB
v3	0.3828	0.1502	200	12,303,460	47MB
v4	0.5191	0.3781	200	25,307,492	96.6MB
v5	0.5352	0.4143	200	24,650,596	94.1MB

6.1.2 The second experiment

After seeing failures, I adopt some idea from other popular light networks that have small size but remarkable performances. One of them is the squeezeNet with the 'Fire' layers.

Knowing the 'v2' model is ideal for optimizing, I added some Fire layers, as well as remove some convolution layers. My goal was to test the effectiveness of the Fire module in boosting accuracy. I created two models—one with a few Fire layers and another with over 50% containing Fire layers—and ran them on Kaggle for approximately 2 hours to evaluate their performance. Also, the alpha variable is adjusted to 0.5, optimizer is Adam with learning rate 0.001.

Trained using discussed configuration, the next two version don't prove any significant increase in classifying task. Additionally, the changing of alpha variable doesn't make models

```python
class Fire(nn.Module):
    def __init__(self, inplanes: int, squeeze_planes: int, expand1x1_planes: int, expand3x3_planes: int) -> None:
        super().__init__()
        self.inplanes = inplanes
        self.squeeze = nn.Conv2d(inplanes, squeeze_planes, kernel_size=1)
        self.squeeze_activation = nn.ReLU(inplace=True)
        self.expand1x1 = nn.Conv2d(squeeze_planes, expand1x1_planes, kernel_size=1)
        self.expand1x1_activation = nn.ReLU(inplace=True)
        self.expand3x3 = nn.Conv2d(squeeze_planes, expand3x3_planes, kernel_size=3, padding=1)
        self.expand3x3_activation = nn.ReLU(inplace=True)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.squeeze_activation(self.squeeze(x))
        return torch.cat(
            [self.expand1x1_activation(self.expand1x1(x)), self.expand3x3_activation(self.expand3x3(x))], 1
        )
```

Figure 7: The Fire module class

Table 3: Model results in second experiment using knowledge distillation

Name	Top 1 error	Top 5 error	Epochs	Alpha	Parameters	Size of file
VGG16	0.2887	0.1054	200	0.9	34,015,396	132.9MB
v6	0.3763	0.1640	200	0.9	9,271,268	35.4MB
v6	0.3728	0.1600	200	0.5	9,271,268	35.4MB
v7	0.4150	0.1711	200	0.9	11,579,428	44.2MB
v7	0.4059	0.1761	200	0.5	11,579,428	44.2MB

fit better with data set. Clearly, there must be a problem in a domain that I don't figure out at the moment of making this report.

6.2 Quantization

To apply static quantization, I used 'torch.qint8' as my target dtype of model and let the Pytorch transforms the inside weights. After that, I test the new model with similar test function for original model. The top 1 error was down a little (0.2889) but the state dict file size was reduced dramatically (129.8 to 74.7 MB).

Table 4: Model results

Name	Top 1 error	Top 5 error	Epochs	Parameters	Size of file
VGG16	0.2887	0.1054	200	34,015,396	132.9MB
VGG16-qint8	0.2889	0.1055	n/a	14,723,136	74.7MB

Table 5: Time response

Name	Random input	Test images	Average
VGG16	123.578ms	53.756s	5.376ms
Quantization model	45.054ms	45.890s	4.589ms

Table 6: Model results after pruning

Name	Top 1 error	Top 5 error	Prune amount	Parameters
VGG16	0.2887	0.1054	0%	34,015,396
Pruning model v1	0.6559	0.3878	10%	33,989,482
Pruning model v2	0.4242	0.2037	10%	33,723,294

In time response test, however, it seems that there is no major reduction. Using profiler to measure the time difference, the quantization model show just around 15% less time than the original model (VGG16).

6.3 Pruning

Since I intend to use the combination of techniques, quantization model is used to be pruned. Due to unknown of amount of redundant nodes, 10% of nodes in few first convolution layers are cut. After running test script, it seems that model is nearly unable to predict correctly. It just randomly assign the predict class more than half of the sample passed. I then shift my focus to the middle layers and employed the L1Unstructured pruning technique instead of RandomUnstructured, hoping for better results. The result is improved but just as same as other bad knowledge distillation models.

7 Conclusion

Compression model is one of the most necessary but challenging task in the pipeline of model development. It requires a deep understand of the model we are working, as well as the advantages of each strategies to apply them properly.

Knowledge distillation shows some potentials to mimic the parent model with a smaller number of parameters. Despite the failure in keeping the epsilon of top 1 error not above 4 percent, there are rooms for improvement. I will try some other effective model architectures and apply some advanced tricks to diminish the error number.

Quantization proves its effectiveness in compressing the size of model. However, the time processing is not likely to behave as expected. Moreover, choices for number types in static quantization are limited. On the other hand, pruning is an interested idea, but somehow didn't work theoretically.

The state dicts are saved in this URL: https://drive.google.com/drive/folders/1K0FBD2wCnPfnRHLH7bLspiz0cczg8V1T?usp=sharing

The full modified fork repository is located in https://github.com/hoangbros03/pytorch-cifar100

8 References

singh96aman: "Compression-of-Deep-Neural-Networks" URL: https://github.com/singh96aman/Compression-of-Deep-Neural-Networks

Neptune.ai "Knowledge Distillation: Principles, Algorithms, Applications" URL: https://neptune.ai/blog/knowledge-distillation

rinf.tech: "5 Reasons Why Machine Learning Quantization is Important for AI Projects" URL: https://www.rinf.tech/5-reasons-why-machine-learning-quantization-is-important-for-ai-projects

Song et al.: "Learning both Weights and Connections for Efficient Neural Networks (2015)"