OPTIMIZING VISION TRANSFORMER MODEL FOR DEPLOYMENT

Jeff Tang, Geeta Chauhan

Vision Transformer models apply the cutting-edge attention-based transformer models, introduced in Natural Language Processing to achieve all kinds of the state of the art (SOTA) results, to Computer Vision tasks. Facebook Data-efficient Image Transformers DeiT is a Vision Transformer model trained on ImageNet for image classification.

In this tutorial, we will first cover what DeiT is and how to use it, then go through the complete steps of scripting, quantizing, optimizing, and using the model in iOS and Android apps. We will also compare the performance of quantized, optimized and non-quantized, non-optimized models, and show the benefits of applying quantization and optimization to the model along the steps.

What is DeiT

Convolutional Neural Networks (CNNs) have been the main models for image classification since deep learning took off in 2012, but CNNs typically require hundreds of millions of images for training to achieve the SOTAresults. DeiT is a vision transformer model that requires a lot less data and computing resources for training to compete with the leading CNNs in performing image classification, which is made possible by two key components of of DeiT:

- Data augmentation that simulates training on a much larger dataset;
- Native distillation that allows the transformer network to learn from a CNN's output.

DeiT shows that Transformers can be successfully applied to computer vision tasks, with limited access to data and resources. For more details on DeiT, see the repo and paper.

Classifying Images with DeiT

Follow the README at the DeiT repo for detailed information on how to classify images using DeiT, or for a quick test, first install the required packages:

```
# pip install torch torchvision timm pandas requests
```

To run in Google Colab, uncomment the following line:

```
# !pip install timm pandas requests
```

then run the script below:

```
from PIL import Image
import torch
import timm
import requests
import torchvision.transforms as transforms
from timm.data.constants import IMAGENET_DEFAULT_MEAN, IMAGENET_DEFAULT_STD

print(torch.__version__)
# should be 1.8.0


model = torch.hub.load('facebookresearch/deit:main', 'deit_base_patch16_224', pretrained=True)
model.eval()

transform = transforms.Compose([
    transforms.Resize(256, interpolation=3),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(IMAGENET_DEFAULT_MEAN, IMAGENET_DEFAULT_STD),
])

img = Image.open(requests.get("https://raw.githubusercontent.com/pytorch/ios-demo-
app/master/HelloWorld/HelloWorld/image.png", stream=True).raw)
img = transform(img)[None,]
out = model(img)
clsidx = torch.argmax(out)
print(clsidx.item())
```

Out:

```
1.13.0+cu117
Downloading: "https://github.com/facebookresearch/deit/zipball/main" to /var/lib/jenkins/.cache/torch/hub/main.zip
Downloading: "https://dl.fbaipublicfiles.com/deit/deit_base_patch16_224-b5f2ef4d.pth" to
/var/lib/jenkins/.cache/torch/hub/checkpoints/deit_base_patch16_224-b5f2ef4d.pth

  0%|          | 0.00/330M [00:00<?, ?B/s]
  0%|          | 56.0k/330M [00:00<10:10, 567kB/s]
  0%|          | 312k/330M [00:00<03:16, 1.76MB/s]
  0%|          | 1.16M/330M [00:00<01:10, 4.90MB/s]
  1%|          | 3.27M/330M [00:00<00:30, 11.3MB/s]
  2%|1          | 5.14M/330M [00:00<00:24, 14.1MB/s]
  2%|2          | 6.75M/330M [00:00<00:22, 15.0MB/s]
  2%|2          | 8.22M/330M [00:00<00:22, 15.1MB/s]
  6%|5          | 18.2M/330M [00:00<00:07, 43.3MB/s]
  9%|8          | 28.9M/330M [00:00<00:04, 64.5MB/s]
 12%|#1         | 39.1M/330M [00:01<00:03, 77.6MB/s]
 14%|#4         | 46.5M/330M [00:01<00:04, 73.6MB/s]
 17%|#7         | 56.6M/330M [00:01<00:03, 82.5MB/s]
 20%|#9         | 64.5M/330M [00:01<00:03, 81.9MB/s]
```

The output should be 269, which, according to the ImageNet list of class index to labels file, maps to 'timber wolf, grey wolf, gray wolf, Canis lupus'.

Now that we have verified that we can use the DeiT model to classify images, let's see how to modify the model so it can run on iOS and Android apps.

Scripting DeiT

To use the model on mobile, we first need to script the model. See the Script and Optimize recipe for a quick overview. Run the code below to convert the DeiT model used in the previous step to the TorchScript format that can run on mobile.

```
model = torch.hub.load('facebookresearch/deit:main', 'deit_base_patch16_224', pretrained=True)
model.eval()
scripted_model = torch.jit.script(model)
scripted_model.save("fbdeit_scripted.pt")
```

Out:

```
Using cache found in /var/lib/jenkins/.cache/torch/hub/facebookresearch_deit_main
```

The scripted model file fbdeit_scripted.pt of size about 346MB is generated.

Quantizing DeiT

To reduce the trained model size significantly while keeping the inference accuracy about the same, quantization can be applied to the model. Thanks to the transformer model used in DeiT, we can easily apply dynamic-quantization to the model, because dynamic quantization works best for LSTM and transformer models (see here for more details).

Now run the code below:

```
# Use 'fbgemm' for server inference and 'qnnpack' for mobile inference
backend = "fbgemm" # replaced with qnnpack causing much worse inference speed for quantized model on this notebook
model.qconfig = torch.quantization.get_default_qconfig(backend)
torch.backends.quantized.engine = backend

quantized_model = torch.quantization.quantize_dynamic(model, qconfig_spec={torch.nn.Linear}, dtype=torch.qint8)
scripted_quantized_model = torch.jit.script(quantized_model)
scripted_quantized_model.save("fbdeit_scripted_quantized.pt")
```

Out:

```
/opt/conda/lib/python3.10/site-packages/torch/ao/quantization/observer.py:214: UserWarning:

Please use quant_min and quant_max to specify the range for observers.                     reduce_range will be deprecated in a
future release of PyTorch.
```

This generates the scripted and quantized version of the model fbdeit_quantized_scripted.pt, with size about 89MB, a 74% reduction of the non-quantized model size of 346MB!

You can use the `scripted_quantized_model` to generate the same inference result:

```
out = scripted_quantized_model(img)
clsidx = torch.argmax(out)
print(clsidx.item())
# The same output 269 should be printed
```

Out:

```
269
```

Optimizing DeiT

The final step before using the quantized and scripted model on mobile is to optimize it:

```
from torch.utils.mobile_optimizer import optimize_for_mobile
optimized_scripted_quantized_model = optimize_for_mobile(scripted_quantized_model)
optimized_scripted_quantized_model.save("fbdeit_optimized_scripted_quantized.pt")
```

The generated fbdeit_optimized_scripted_quantized.pt file has about the same size as the quantized, scripted, but non-optimized model. The inference result remains the same.

```
out = optimized_scripted_quantized_model(img)
clsidx = torch.argmax(out)
print(clsidx.item())
# Again, the same output 269 should be printed
```

Out:

```
269
```

Using Lite Interpreter

To see how much model size reduction and inference speed up the Lite Interpreter can result in, let's create the lite version of the model.

```
optimized_scripted_quantized_model._save_for_lite_interpreter("fbdeit_optimized_scripted_quantized_lite.ptl")
ptl = torch.jit.load("fbdeit_optimized_scripted_quantized_lite.ptl")
```

Although the lite model size is comparable to the non-lite version, when running the lite version on mobile, the inference speed up is expected.

Comparing Inference Speed

To see how the inference speed differs for the four models - the original model, the scripted model, the quantized-and-scripted model, the optimized-quantized-and-scripted model - run the code below:

```
with torch.autograd.profiler.profile(use_cuda=False) as prof1:
    out = model(img)
with torch.autograd.profiler.profile(use_cuda=False) as prof2:
    out = scripted_model(img)
with torch.autograd.profiler.profile(use_cuda=False) as prof3:
    out = scripted_quantized_model(img)
with torch.autograd.profiler.profile(use_cuda=False) as prof4:
    out = optimized_scripted_quantized_model(img)
with torch.autograd.profiler.profile(use_cuda=False) as prof5:
    out = ptl(img)

print("original model: {:.2f}ms".format(prof1.self_cpu_time_total/1000))
print("scripted model: {:.2f}ms".format(prof2.self_cpu_time_total/1000))
print("scripted & quantized model: {:.2f}ms".format(prof3.self_cpu_time_total/1000))
print("scripted & quantized & optimized model: {:.2f}ms".format(prof4.self_cpu_time_total/1000))
print("lite model: {:.2f}ms".format(prof5.self_cpu_time_total/1000))
```

Out:

```
original model: 318.75ms
scripted model: 346.39ms
scripted & quantized model: 251.74ms
scripted & quantized & optimized model: 219.66ms
lite model: 206.44ms
```

The results running on a Google Colab are:

```
original model: 1236.69ms
scripted model: 1226.72ms
scripted & quantized model: 593.19ms
scripted & quantized & optimized model: 598.01ms
lite model: 600.72ms
```

The following results summarize the inference time taken by each model and the percentage reduction of each model relative to the original model.

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'Model': ['original model','scripted model', 'scripted & quantized model', 'scripted & quantized & optimized
model', 'lite model']})
df = pd.concat([df, pd.DataFrame([
    ["{:.2f}ms".format(prof1.self_cpu_time_total/1000), "0%"],
    ["{:.2f}ms".format(prof2.self_cpu_time_total/1000),
     "{:.2f}%".format((prof1.self_cpu_time_total-prof2.self_cpu_time_total)/prof1.self_cpu_time_total*100)],
    ["{:.2f}ms".format(prof3.self_cpu_time_total/1000),
     "{:.2f}%".format((prof1.self_cpu_time_total-prof3.self_cpu_time_total)/prof1.self_cpu_time_total*100)],
    ["{:.2f}ms".format(prof4.self_cpu_time_total/1000),
     "{:.2f}%".format((prof1.self_cpu_time_total-prof4.self_cpu_time_total)/prof1.self_cpu_time_total*100)],
    ["{:.2f}ms".format(prof5.self_cpu_time_total/1000),
     "{:.2f}%".format((prof1.self_cpu_time_total-prof5.self_cpu_time_total)/prof1.self_cpu_time_total*100)]],
    columns=['Inference Time', 'Reduction'])], axis=1)

print(df)

"""
        Model                             Inference Time    Reduction
0   original model                             1236.69ms           0%
1   scripted model                             1226.72ms        0.81%
2   scripted & quantized model                  593.19ms       52.03%
3   scripted & quantized & optimized model      598.01ms       51.64%
4   lite model                                  600.72ms       51.43%
"""
```

Out:

```
                                    Model Inference Time Reduction
0                          original model       318.75ms        0%
1                          scripted model       346.39ms    -8.67%
2              scripted & quantized model       251.74ms    21.02%
3  scripted & quantized & optimized model       219.66ms    31.09%
4                              lite model       206.44ms    35.23%

'\n        Model                             Inference Time    Reduction\n0\toriginal model
1236.69ms           0%\n1\tscripted model                             1226.72ms        0.81%\n2\tscripted & quantized model
593.19ms       52.03%\n3\tscripted & quantized & optimized model      598.01ms       51.64%\n4\tlite model
600.72ms       51.43%\n'
```

Learn More

- Facebook Data-efficient Image Transformers
- Vision Transformer with ImageNet and MNIST on iOS
- Vision Transformer with ImageNet and MNIST on Android

Total running time of the script: (0 minutes 30.306 seconds)

Rate this Tutorial ☆ ☆ ☆ ☆ ☆