Rizqy Asyraff Athallah

1103210158

Machine learning Task 6

- **BAB 9**
    1. **Tensorflow Model Optimization**
        1. **install library tensor**

```
%pip install -q tensorflow
%pip install -q tensorflow-model-optimization
import tensorflow as tf
import tensorflow_model_optimization as tfmot
from tensorflow import keras
import pathlib
import numpy as np
```

        2. **Memuat Dataset MNIST**

```
# Load MNIST dataset
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the input image so that each pixel value is between 0 and 1.
train_images = train_images / 255.0
test_images = test_images / 255.0
```

        3. **Mendefinisikan model CNN dan melatihnya pada kumpulan data MNIST.**

```
# Define the model architecture
model = keras.Sequential([
  keras.layers.InputLayer(input_shape=(28, 28)),
  keras.layers.Reshape(target_shape=(28, 28, 1)),
  keras.layers.Conv2D(filters=12, kernel_size=(3, 3), activation=tf.nn.relu),
  keras.layers.MaxPooling2D(pool_size=(2, 2)),
  keras.layers.Flatten(),
```

```
  keras.layers.Dense(10)
])

# Train the digit classification model
model.compile(optimizer='adam',
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
model.fit(
  train_images,
  train_labels,
  epochs=1,
  validation_data=(test_images, test_labels)
)
```

4. **Setelah melatih model, selanjutnya mengubahnya ke format TFLite dan kemudian melakukan kuantisasi selama konversi.**

```
tflite_models_dir = pathlib.Path("notebooks/Unit 9 - Model Optimization/models")
tflite_models_dir.mkdir(exist_ok=True, parents=True)
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# without quantization
tflite_model = converter.convert()
tflite_model_file = tflite_models_dir/"original_model.tflite"
tflite_model_file.write_bytes(tflite_model)

# with quantization
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quant_model = converter.convert()
tflite_model_quant_file = tflite_models_dir/"quantized_model.tflite"
tflite_model_quant_file.write_bytes(tflite_quant_model)
```

5. **Menampilkan isi direktori tempat model TensorFlow Lite disimpan**

```
%ls -lh {tflite_models_dir}
```

6. **Membandingkan model yang sudah terkuantisasi dengan model aslinya**

```
# A helper function to evaluate the TF Lite model using "test" dataset.
def evaluate_model(interpreter):
```

```
  input_index = interpreter.get_input_details()[0]["index"]
  output_index = interpreter.get_output_details()[0]["index"]

  # Run predictions on every image in the "test" dataset.
  prediction_digits = []
  for test_image in test_images:
    # Pre-processing: add batch dimension and convert to float32 to match with
    # the model's input data format.
    test_image = np.expand_dims(test_image, axis=0).astype(np.float32)
    interpreter.set_tensor(input_index, test_image)

    # Run inference.
    interpreter.invoke()

    # Post-processing: remove batch dimension and find the digit with highest
    # probability.
    output = interpreter.tensor(output_index)
    digit = np.argmax(output()[0])
    prediction_digits.append(digit)

  # Compare prediction results with ground truth labels to calculate accuracy.
  accurate_count = 0
  for index in range(len(prediction_digits)):
    if prediction_digits[index] == test_labels[index]:
      accurate_count += 1
  accuracy = accurate_count * 1.0 / len(prediction_digits)

  return accuracy


interpreter = tf.lite.Interpreter(model_path=str(tflite_model_file))
interpreter.allocate_tensors()
print("Original model accuracy = ", evaluate_model(interpreter))


interpreter_quant = tf.lite.Interpreter(model_path=str(tflite_model_quant_file))
```

```
interpreter_quant.allocate_tensors()
print("Quantized model accuracy = ", evaluate_model(interpreter_quant))
```

7. Mengurangi ukuran model dengan memangkas nya

```
prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude

# Compute end step to finish pruning after 2 epochs.
batch_size = 128
epochs = 2
validation_split = 0.1 # 10% of training set will be used for validation set.

num_images = train_images.shape[0] * (1 - validation_split)
end_step = np.ceil(num_images / batch_size).astype(np.int32) * epochs

# Define model for pruning.
pruning_params = {
    'pruning_schedule': tfmot.sparsity.keras.PolynomialDecay(initial_sparsity=0.50,
                                                             final_sparsity=0.80,
                                                             begin_step=0,
                                                             end_step=end_step)
}

model_for_pruning = prune_low_magnitude(model, **pruning_params)

# `prune_low_magnitude` requires a recompile.
model_for_pruning.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

print(model_for_pruning.summary())

callbacks = [
  tfmot.sparsity.keras.UpdatePruningStep(),
]

model_for_pruning.fit(train_images, train_labels,
```

```
              batch_size=batch_size, epochs=epochs, validation_split=validation_split,
              callbacks=callbacks)
```

8. **Setelah dipangkas tingkat keakuratan model meningkat sehingga sama dengan model asli nya**

```
_, baseline_model_accuracy = model.evaluate(
    test_images, test_labels, verbose=0)
_, model_for_pruning_accuracy = model_for_pruning.evaluate(
   test_images, test_labels, verbose=0)

print('Baseline test accuracy:', baseline_model_accuracy)
print('Pruned test accuracy:', model_for_pruning_accuracy)
```

9. **mengonversi model TensorFlow yang sudah di-pruning menjadi format TensorFlow Lite (TFLite)**

```
model_for_export = tfmot.sparsity.keras.strip_pruning(model_for_pruning)

pruning_converter = tf.lite.TFLiteConverter.from_keras_model(model_for_export)
pruned_tflite_model = pruning_converter.convert()
pruned_model_file = tflite_models_dir/"pruned_model.tflite"
pruned_model_file.write_bytes(pruned_tflite_model)
```

2. ONNX(Open Neural Network Exchange)

10. **install library ONNX**

```
%pip install torch torchvision
%pip install onnx onnxruntime
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import torch.quantization
import pathlib
import numpy as np
import torch.onnx
import onnx
import onnxruntime
```

```
from onnxruntime.quantization import quantize_dynamic, quantize_static, CalibrationDataReader, QuantType
```

11. Selanjutnya, melatih model CNN sederhana pada kumpulan data MNIST.

```
transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])

train_dataset = datasets.MNIST('./data', train=True, download=True,transform=transform)
test_dataset = datasets.MNIST('./data', train=False,transform=transform)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc = nn.Linear(12 * 13 * 13, 10)

    def forward(self, x):
        x = x.view(-1, 1, 28, 28)
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        output = F.log_softmax(x, dim=1)
        return output


train_loader = torch.utils.data.DataLoader(train_dataset, 32)
test_loader = torch.utils.data.DataLoader(test_dataset, 32)

device = "cpu"

epochs = 1

model = Net().to(device)
```

```
optimizer = optim.Adam(model.parameters())


model.train()


for epoch in range(1, epochs+1):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))


MODEL_DIR = pathlib.Path("./onnx_models")
MODEL_DIR.mkdir(exist_ok=True)
torch.save(model.state_dict(), MODEL_DIR / "original_model.p")
```

12. **Setelah training model, ekspor model ke format ONNX**

```
x, _ = next(iter(train_loader))
torch.onnx.export(model,
                  x,
                  MODEL_DIR / "mnist_model.onnx",
                  export_params=True,
                  opset_version=10,
                  do_constant_folding=True,
                  input_names = ['input'],
                  output_names = ['output'],
                  dynamic_axes={'input' : {0 : 'batch_size'},
                                'output' : {0 : 'batch_size'}})
```

13. **Selanjutnya, validasi model yang dikonversi dengan menjalankan inferensi dan membandingkan hasilnya dengan model PyTorch**.

```
torch_out = model(x)
```

```
onnx_model = onnx.load(MODEL_DIR / "mnist_model.onnx")
onnx.checker.check_model(onnx_model)

ort_session = onnxruntime.InferenceSession(MODEL_DIR / "mnist_model.onnx",
providers=["CPUExecutionProvider"])

def to_numpy(tensor):
    return tensor.detach().cpu().numpy() if tensor.requires_grad else tensor.cpu().numpy()

# compute ONNX Runtime output prediction
ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(x)}
ort_outs = ort_session.run(None, ort_inputs)

# compare ONNX Runtime and PyTorch results
np.testing.assert_allclose(to_numpy(torch_out), ort_outs[0], rtol=1e-03, atol=1e-05)

print("Exported model has been tested with ONNXRuntime, and the result looks good!")
```

14. Membandingkan model asli dengan model quantized

```
%ls -lh {MODEL_DIR}
```

15. Membandingkan akurasi nya juga

```
def test_onnx(model_name, data_loader):
    onnx_model = onnx.load(model_name)
    onnx.checker.check_model(onnx_model)
    ort_session = onnxruntime.InferenceSession(model_name)
    test_loss = 0
    correct = 0
    for data, target in data_loader:
        ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(data)}
        output = ort_session.run(None, ort_inputs)[0]
        output = torch.from_numpy(output)
        test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
        pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
        correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(data_loader.dataset)
```

```
    return 100. * correct / len(data_loader.dataset)

acc = test_onnx(MODEL_DIR / "mnist_model.onnx", test_loader)
print(f"Accuracy of the original model is {acc}%")

qacc = test_onnx(MODEL_DIR / "mnist_model_quant.onnx", test_loader)
print(f"Accuracy of the quantized model is {qacc}%")
```

3. OPTIMUM

16. install library yang dibutuhkan

```
%pip install optimum
!pip install --upgrade cupy-cuda11x  # Replace 11x with your CUDA version
!apt-get update
!apt-get install -y cuda # Install CUDA. Please ensure you have the correct version for your hardware.

# Set the LD_LIBRARY_PATH environment variable.
import os
os.environ['LD_LIBRARY_PATH'] = '/usr/local/cuda/lib64:' + os.environ.get('LD_LIBRARY_PATH', '')

# Reinstall CuPy. Choose the correct version for your CUDA installation.
!pip install --upgrade cupy-cuda11x # Replace 11x with your CUDA version if needed.
from optimum.onnxruntime import ORTQuantizer, ORTModelForImageClassification
from functools import partial
from optimum.onnxruntime.configuration import AutoQuantizationConfig, AutoCalibrationConfig
from onnxruntime.quantization import QuantType
from transformers import AutoFeatureExtractor
from PIL import Image
import requests
```

17. Memuat model ONNX Runtime dari Huggingface Hub. menggunakan Vision Transformer vit-base-patch16-224.

```
preprocessor = AutoFeatureExtractor.from_pretrained("optimum/vit-base-patch16-224")
model = ORTModelForImageClassification.from_pretrained("optimum/vit-base-patch16-224")
model.save_pretrained("models/vit-base-patch16-224")
```

18. **Untuk melakukan dynamic quantization, pertama-tama buat quantizer menggunakan kelas ORTQuantizer dan tentukan konfigurasi menggunakan AutoQuantizationConfig sebelum memanggil metode quantize() untuk mengkuantisasi model.**

```
quantizer = ORTQuantizer.from_pretrained(model)
dqconfig = AutoQuantizationConfig.avx512_vnni(is_static=False, per_channel=False)
dqconfig.weights_dtype = QuantType.QUInt8
model_quantized_path = quantizer.quantize(
    save_dir="models/vit-base-patch16-224-quantized-dynamic",
    quantization_config=dqconfig,
)
```

19. **Selanjutnya, memvalidasi model terkuantisasi dengan membandingkan hasil model asli dan model kuantisasi. Dimana membuat fungsi untuk melakukan inferensi ketika model, prosesor, dan gambar diberikan kemudian mengembalikan hasil klasifikasi berdasarkan label ImageNet**

```
def infer_ImageNet(classification_model, processor, image):
    inputs = processor(images=image, return_tensors="pt")
    outputs = classification_model(**inputs)
    logits = outputs.logits
    predicted_class_idx = logits.argmax(-1).item()
    return classification_model.config.id2label[predicted_class_idx]

# Get sample image
url = 'http://images.cocodataset.org/val2017/000000039769.jpg'
image = Image.open(requests.get(url, stream=True).raw)

res = infer_ImageNet(model, preprocessor, image)
print("Original model prediction:", res)

quantized_model = ORTModelForImageClassification.from_pretrained(model_quantized_path)
dq_res = infer_ImageNet(quantized_model, preprocessor, image)
print("Quantized model prediction:", dq_res)
display(image)
```

- **BAB 10**

1. Generating Synthetic Data Using a 3D Renderer

```
!pip install PyOpenGL PyOpenGL_accelerate
# Import packages after installing
import trimesh
import pyrender
import os
# Set the PYOPENGL_PLATFORM environment variable to 'egl' or 'osmesa' before importing pyrender
os.environ['PYOPENGL_PLATFORM'] = 'egl'

# Buat kubus 3D
mesh = trimesh.creation.box(extents=(2, 2, 2))

# Render gambar
scene = pyrender.Scene()
scene.add(pyrender.Mesh.from_trimesh(mesh))
camera = pyrender.PerspectiveCamera(yfov=1.5)
scene.add(camera, pose=[[1, 0, 0, 0], [0, 0, -1, -5], [0, 1, 0, 5], [0, 0, 0, 1]])

renderer = pyrender.OffscreenRenderer(800, 800)
color, _ = renderer.render(scene)

# Simpan hasil
from PIL import Image
image = Image.fromarray(color)
image.save("synthetic_image.png")
```

2. Synthetic Data Generation Using DCGAN

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms

# Generator
```

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(100, 64, 4, 1, 0, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.ConvTranspose2d(64, 1, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, x):
        return self.main(x)

# Discriminator
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(1, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.main(x)

# Contoh menghasilkan gambar sintetis
generator = Generator()
noise = torch.randn(1, 100, 1, 1)  # Noise input
synthetic_image = generator(noise)

print("Synthetic Image Generated:", synthetic_image.shape)
```

3. Synthetic Data Generation with Diffusion Models

```
import torch
import torch.nn as nn

# Noise Generator
class DiffusionModel(nn.Module):
    def __init__(self):
        super(DiffusionModel, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.Tanh()
        )

    def forward(self, x):
        return self.layers(x)

# Contoh generasi noise dan pembelajaran
diffusion_model = DiffusionModel()
noise = torch.randn(1, 256)  # Input noise
synthetic_data = diffusion_model(noise)

print("Synthetic Data Generated:", synthetic_data)
```

20. Library

```
pip install diffusers transformers accelerate
```

21. Menghailkan gambar dengan Stable Diffusion, salah satu model berbasis diffusion.

```
from diffusers import DDPMPipeline

# Memuat pipeline DDPM (diffusion sederhana)
pipeline = DDPMPipeline.from_pretrained("google/ddpm-cifar10-32")
pipeline = pipeline.to("cpu")  # Gunakan CPU
```

```
# Menghasilkan gambar dari noise
generated_image = pipeline(num_inference_steps=50).images[0]

# Menampilkan gambar
generated_image.show()

# Menyimpan gambar
generated_image.save("ddpm_synthetic_image.png")
print("Gambar telah disimpan sebagai ddpm_synthetic_image.png")
```

- **BAB 11**
  - **Zero-shot Learning**

```
import requests
from PIL import Image
from io import BytesIO
import urllib.parse

from transformers import pipeline

# 1. Buat pipeline untuk zero-shot image classification
zsl_pipeline = pipeline("zero-shot-image-classification", model="openai/clip-vit-base-patch16")

# 2. Muat gambar dari URL
# Original URL: This leads to a Bing search result page, not the image directly
url =
"https://www.bing.com/images/search?view=detailV2&ccid=tK83YsaQ&id=80EC431BF22EF94925D6C617C93BE6E08EC2A604&thid=OIP.tK83YsaQrO6a6_OEWqJZ0wHaFj&mediaurl=https%3a%2f%2fth.bing.com%2fth%2fid%2fR.b4af3762c690acee9aebf3845aa259d3%3frik%3dBKbCjuDmO8kXxg%26riu%3dhttp%253a%252f%252f2.bp.blogspot.com%252f-_4XsKSWULzo%252fUIZGhNiRgtI%252fAAAAAAAACZA%252fVV5fIy-d7_o%252fs1600%252fgambar%252bkucing%252blucu%252b3.jpg%26ehk%3dZVra8E7YUE206BC9qTqZkFR3VRm7m1u7eOZ6gqDgX0c%253d%26risl%3d%26pid%3dImgRaw%26r%3d0&exph=864&expw=1152&q=gambar+kucing&simid=607990550606460530&FORM=IRPRST&ck=F5421CBCC2ED3028D0C20E76B608CF09&selectedIndex=8&itb=0"
```

```
# Extract the actual image URL from the Bing search result URL
parsed_url = urllib.parse.urlparse(url)
query_params = urllib.parse.parse_qs(parsed_url.query)
image_url = query_params['mediaurl'][0] # Get the direct image URL

# Now, use the direct image URL
response = requests.get(image_url, stream=True)
response.raise_for_status()
image = Image.open(BytesIO(response.content))

# 3. Kelas kandidat
candidate_labels = ["cat", "dog", "car"]

# 4. Prediksi
# Pass candidate_labels as a keyword argument
predictions = zsl_pipeline(image, candidate_labels=candidate_labels)

# 5. Tampilkan hasil prediksi
print("Predictions:", predictions)
```

- **BAB 12**

Apa itu Etika dalam AI?

- Etika dalam AI adalah tentang memastikan teknologi kecerdasan buatan (termasuk model Computer Vision) digunakan secara adil, transparan, dan tidak merugikan siapa pun.
- Isu etika melibatkan bias, privasi, disinformasi, ketidaksetaraan akses, dan penggunaan AI untuk tujuan berbahaya.

Mengapa Etika Penting?

AI seringkali mencerminkan bias yang ada dalam data pelatihan. Misalnya, model Computer Vision dapat menunjukkan bias terhadap ras, gender, atau budaya karena data pelatihan yang tidak

seimbang. Model yang digunakan tanpa pertimbangan etika dapat memperburuk ketidaksetaraan sosial.

Inisiatif Utama oleh Hugging Face:

a) Model Card:

Setiap model di Hugging Face memiliki model card, yang menjelaskan:

- ✓ Tujuan penggunaan model.
- ✓ Batasan model.
- ✓ Risiko etika.

Contoh: Jika sebuah model dilatih pada dataset yang bias, ini akan disebutkan di model card.

b) **Datasets Card**:

Memberikan transparansi tentang asal, karakteristik, dan bias dalam dataset.

c) im Etika AI:

Hugging Face memiliki tim khusus untuk mempelajari dampak etika dari teknologi AI mereka. Mereka mempromosikan panduan tentang pengembangan AI yang bertanggung jawab.

Pedoman yang Diadopsi oleh Hugging Face:

- ✓ Responsible AI: Model harus digunakan untuk kebaikan sosial dan tidak boleh digunakan untuk tujuan berbahaya.
- ✓ Inclusive AI: Memastikan bahwa model tidak mendiskriminasi kelompok apa pun.
- ✓ Transparent AI: Memberikan penjelasan tentang cara kerja model, data pelatihan, dan batasan.