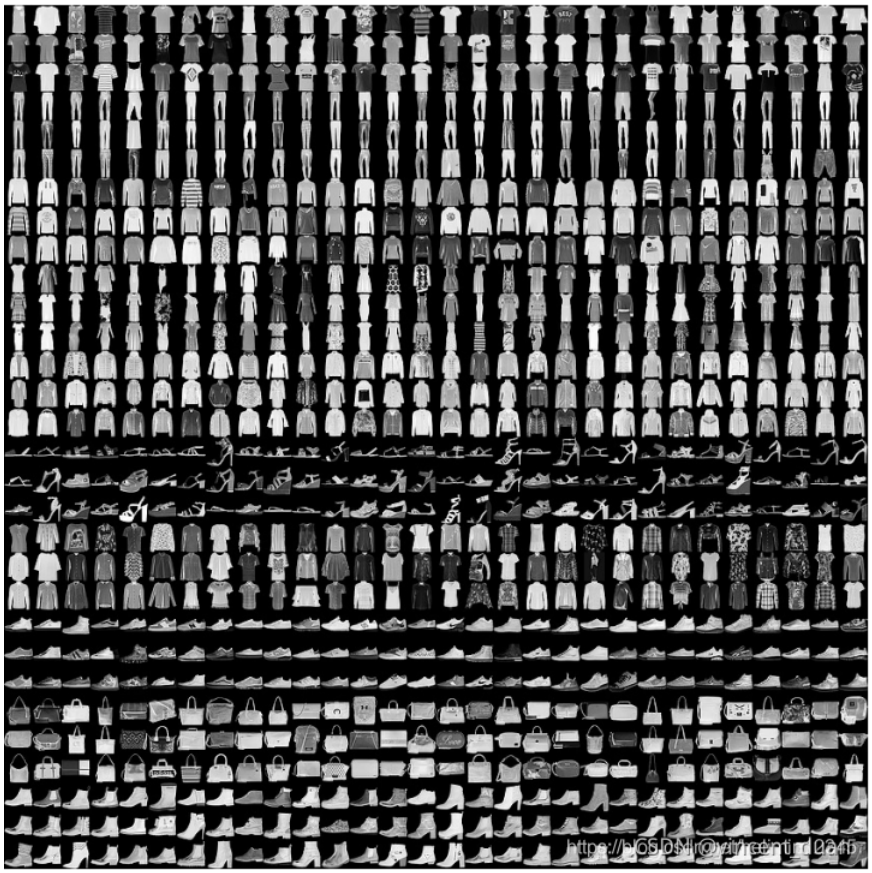
LENET 量化分析

FashionMNIST 資料庫

衣服、鞋子等服饰组成資料庫，經常被使用在機器學習便是的領域，每一張圖片為28*28大小，這個數據庫當中包含60000筆訓練影像和10000筆測試影像。



- 下載資料庫

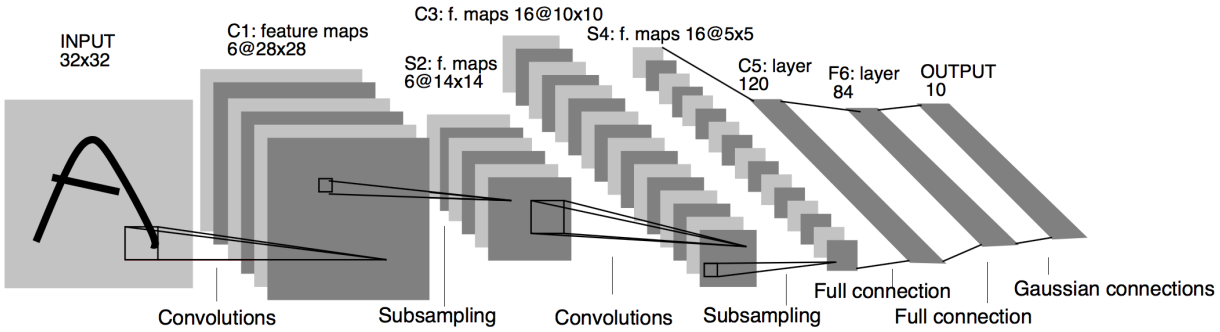
```
mnist_train = torchvision.datasets.FashionMNIST(root='~/Datas
mnist_test = torchvision.datasets.FashionMNIST(root='~/Datase

def load_data_fashion_mnist(mnist_train, mnist_test, batch_si
    num_workers = 0
    train_iter = torch.utils.data.DataLoader(mnist_train, bat
```

```
    test_iter = torch.utils.data.DataLoader(mnist_test, batch
    return train_iter, test_iter

batch_size = 256
train_iter, test_iter = load_data_fashion_mnist(mnist_train,
```

LENET 模型



- 利用 PyTorch 建立模型

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 6, 5), # in_channels, out_channels,
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2), # kernel_size, stride
            nn.Conv2d(6, 16, 5),
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2)
        )
        self.fc = nn.Sequential(
            nn.Linear(16*4*4, 120),
            nn.Sigmoid(),
            nn.Linear(120, 84),
            nn.Sigmoid(),
```

```
            nn.Linear(84, 10)
        )

    def forward(self, img):
        feature = self.conv(img)
        output = self.fc(feature.view(img.shape[0], -1))
        return output

net = LeNet()
```

- 訓練、測試模型

```
def evaluate_accuracy(data_iter, net, device=None):
    if device is None and isinstance(net, torch.nn.Module):
        # 如果没指定device就使用net的device
        device = list(net.parameters())[0].device
    acc_sum, n = 0.0, 0
    with torch.no_grad():
        for X, y in data_iter:
            net.eval() # 评估模式, 这会关闭dropout
            acc_sum += (net(X.to(device)).argmax(dim=1) == y
            net.train() # 改回训练模式
            n += y.shape[0]
    return acc_sum / n


def train(net, train_iter, test_iter, batch_size, optimizer,
    net = net.to(device)
    print("training on ", device)
    loss = torch.nn.CrossEntropyLoss()
    for epoch in range(num_epochs):
        train_l_sum, train_acc_sum, n, batch_count, start = (
        for X, y in train_iter:
            X = X.to(device)
            y = y.to(device)
```

```
            y_hat = net(X)
            l = loss(y_hat, y)
            optimizer.zero_grad()
            l.backward()
            optimizer.step()
            train_l_sum += l.cpu().item()
            train_acc_sum += (y_hat.argmax(dim=1) == y).sum(
            n += y.shape[0]
            batch_count += 1
        test_acc = evaluate_accuracy(test_iter, net)
        print('epoch %d, loss %.4f, train acc %.3f, test acc
              % (epoch + 1, train_l_sum / batch_count, train_

lr, num_epochs = 0.001, 5
optimizer = torch.optim.Adam(net.parameters(), lr=lr)
train(net, train_iter, test_iter, batch_size, optimizer, devi
```

- 訓練後的結果

```
training on  cpu
epoch 1, loss 1.8429, train acc 0.326, test acc 0.584, time 7.0 sec
epoch 2, loss 0.9517, train acc 0.632, test acc 0.679, time 6.2 sec
epoch 3, loss 0.7781, train acc 0.715, test acc 0.723, time 6.1 sec
epoch 4, loss 0.6862, train acc 0.742, test acc 0.747, time 6.2 sec
epoch 5, loss 0.6306, train acc 0.757, test acc 0.745, time 6.1 sec
```

- 將訓練完的模型轉換成 onnx 格式

```
# Export the model to ONNX
dummy_input = torch.randn(1, 1, 28, 28, device=device)
onnx_path = "lenet_fashion_mnist.onnx"
torch.onnx.export(net, dummy_input, onnx_path,
                  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'}}
```

- 測試 onnx 格式的準確率

```
# Evaluate accuracy on the test set
correct = 0
total = 0

for images, labels in tqdm(test_loader):
    # Preprocess input data
    images = images.numpy()

    # Run the model (inference)
    ort_inputs = {ort_session.get_inputs()[0].name: images}
    ort_outs = ort_session.run(None, ort_inputs)
    outputs = ort_outs[0]

    # Calculate accuracy
    predicted = np.argmax(outputs, axis=1)
    total += labels.size(0)
    correct += (predicted == labels.numpy()).sum().item()

accuracy = correct / total
print(f'Accuracy of the ONNX model on the test dataset: {accu
```

  - 測試結果

```
Accuracy of the ONNX model on the test dataset: 0.7453
```

- 量測時間

```
# Load the quantized model (or the original model for compari
model_path = "lenet_fashion_mnist.onnx"  # Use your quantized
session = ort.InferenceSession(model_path)
```

```
# Create a random input tensor with the same shape as the mod
input_name = session.get_inputs()[0].name
output_name = session.get_outputs()[0].name
dummy_input = np.random.rand(1, 1, 28, 28).astype(np.float32)

# Warm-up inference (optional, helps in stabilizing measureme
for _ in range(10):
    _ = session.run([output_name], {input_name: dummy_input})

# Measure inference time
start_time = time.time()
for _ in range(100):  # Run multiple inferences to get an ave
    result = session.run([output_name], {input_name: dummy_in
end_time = time.time()

# Calculate the average inference time
total_time = end_time - start_time
average_inference_time = total_time / 100

print(f"Average inference time: {average_inference_time * 100
```

- 量測結果

```
Average inference time: 0.1050 ms
```

- 利用 onnx runtime 量化模型，由於是 CNN 模型，因此採用官方建議的 static quantization

```
# Define a custom calibration data reader
class MnistDataReader(CalibrationDataReader):
    def __init__(self, data_loader, session):
        self.data_loader = data_loader
        self.data_iter = iter(data_loader)
        self.input_name = session.get_inputs()[0].name
```

```
    def get_next(self):
        try:
            batch = next(self.data_iter)
        except StopIteration:
            return None
        return {self.input_name: batch[0].numpy()}

    def rewind(self):
        self.data_iter = iter(self.data_loader)

# Load the ONNX model
onnx_model_path = "lenet_fashion_mnist.onnx"
model = onnx.load(onnx_model_path)

# Simplify the model (optional but recommended)
import onnxsim
model_simp, check = onnxsim.simplify(onnx_model_path)
simplified_model_path = "lenet_fashion_mnist_simplified.onnx"
onnx.save(model_simp, simplified_model_path)

# Create an ONNX Runtime session for the simplified model
session = ort.InferenceSession(simplified_model_path)

# Define the data loader for calibration
transform = transforms.Compose([
    transforms.ToTensor(),
])

calibration_dataset = datasets.FashionMNIST(root='~/Datasets/
calibration_loader = torch.utils.data.DataLoader(calibration_

# Create the calibration data reader with the session
calibration_data_reader = MnistDataReader(calibration_loader,

# Perform static quantization
quantized_model_path = "lenet_fashion_mnist_quantized.onnx"
```

```
quantize_static(
    simplified_model_path,  # Use the simplified model path
    quantized_model_path,
    calibration_data_reader,
    weight_type=QuantType.QInt8  # Quantize weights to 8-bit
)
```

- 量測準確率

```
Accuracy of the ONNX model on the test dataset: 0.7463
```

- 量測時間

```
Average inference time: 0.0678 ms
```

總結

	original	static quantization
accuracy	0.7453	0.7463
inference time	0.1050 ms	0.0678 ms