ณสิต ผลัญชัย 65010273

วิวัตร เตชะโกศล 65011001

Lab7_2

```python
def load_data(path, class_names):
    ### START CODE HERE ###
    transform = transforms.Compose([
        transforms.Resize((28, 28)),
        transforms.ToTensor(),
        transforms.Grayscale(num_output_channels=1),
        transforms.Pad(27, padding_mode='reflect'),
        transforms.RandomRotation(degrees=15),
        transforms.CenterCrop(28)
    ])

    dataset = ImageFolder(path, transform=transform)

    class_to_idx = dataset.class_to_idx
    selected_classes = [class_to_idx[c] for c in class_names]

    indices = [i for i, (path, label) in enumerate(dataset.samples) if label in selected_classes]
    subset_dataset = Subset(dataset, indices)
```

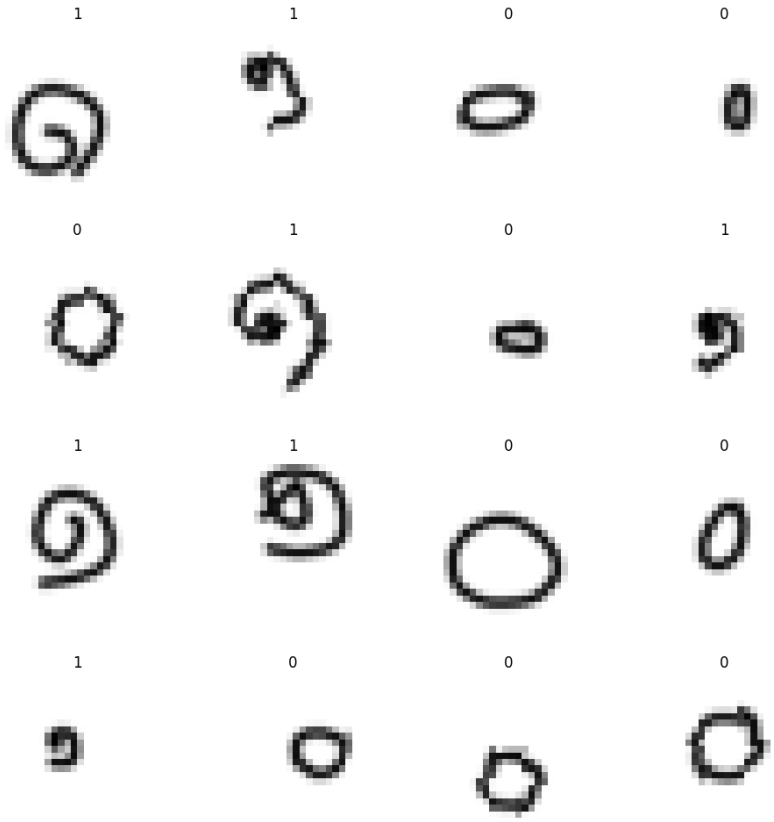
```python
test_split = 0.3
test_size = int(test_split * len(subset_dataset))
train_size = len(subset_dataset) - test_size

train_dataset, test_dataset = torch.utils.data.random_split(subset_dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, pin_memory=True)
```

สร้าง function load_data() คือ function ที่ใช้ในการโหลดภาพมาใช้ train และ test โดยจะใช้ parameters path คือ path ของรูปภาพและ class_names ใช้สำหรับกำหนดว่าจะนำภาพจาก folders ไหนมา train และ test บ้างในกรณีนี้โจทย์ให้ใช้ pair of classes เราก็จะเลือก class 0 และ 1 โดยใส่ [‘0’, ‘1’] เข้าไป ซึ่งภายใน function ก็จะมีการ preprocess ภาพด้วยคำสั่งต่างๆ เช่น resize ปรับขนาดให้เป็น (28, 28) Grayscale ปรับให้เป็นภาพขาวดำ 1 channel และหมุนภาพ หลังจาก preprocess ขั้นตอนต่างๆ ก็ทำการแบ่งขนาด data ของส่วนที่ train และ test โดยให้เป็นอัตราส่วน 70:30

```
### START CODE HERE ###

path = r'C:\Users\Nickv\Documents\ImageProcessing\Week7\lab7_2_thai-handwriting-number\t
class_names = ['0', '1']
train_loader, test_loader = load_data(path, class_names)
batch, labels = next(iter(train_loader))
imshow_grid(batch, labels)

### END CODE HERE ###
```

ทำการใช้ function ที่อธิบายด้านบนจะได้ผลลัพท์ดังรูปด้านล่าง

```
Train Dataset Information:
        Number of images in class 0: 268
        Number of images in class 1: 276
        Total number of training samples: 544

Test Dataset Information:
        Number of images in class 0: 120
        Number of images in class 1: 113
        Total number of test samples: 233
```



```python
class CustomLeNet(nn.Module):
    def __init__(self):
        super(CustomLeNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(32 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 2)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))

        x = x.view(-1, 32 * 4 * 4)

        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))

        x = self.sigmoid(self.fc3(x))

        return x
```

ทำการสร้าง model โดยมีการ convolution 2 รอบ รอบแรกจะรับ input เป็น 1 channel (ภาพขาวดำ) และมีการ Max Pooling ก่อนนำมาเข้า fully connected layer 1 ซึ่งจะรับ input เป็น flattened feature map จาก layer ก่อนหน้าและได้รับ output เป็น 120 neurons โดยจะผ่าน fully connected layer 2-3 จะเหลือ 2 neurons ซึ่งคือ binary classification

```python
def train(model,opt,loss_fn,train_loader,test_loader,epochs=10,checkpoint_path=None,device='cpu'):
    ### START CODE HERE ###
    print("🚂 Training on", device)
    model = model.to(device)

    for epoch in range(epochs):
        model.train()
        avg_train_loss = 0.0
        step = 0
        train_bar = tqdm(train_loader, desc=f'🚀 Training Epoch [{epoch+1}/{epochs}]', unit='batch')

        for images, gt in train_bar:
            images = images.to(device)
            gt = gt.to(device).float().view(-1, 1)
            opt.zero_grad()
            output = model(images)
            loss = loss_fn(output, gt)
            loss.backward()
            opt.step()
```

```python
        avg_train_loss += loss.item()
        step += 1

        train_bar.set_postfix(loss=loss.item())

    avg_train_loss /= step

    model.eval()
    avg_test_loss = 0.0
    test_bar = tqdm(test_loader, desc='📄 Testing', unit='batch')

    for images, gt in test_bar:
        images = images.to(device)
        gt = gt.to(device).float().view(-1, 1)
        output = model(images)
        loss = loss_fn(output, gt)
        avg_test_loss += loss.item()

    avg_test_loss /= len(test_loader)

    print(f'Epoch [{epoch+1}/{epochs}], Train Loss: {avg_train_loss:.4f}, Test Loss: {avg_test_loss:.4f}')
```

```python
    with torch.no_grad():
        sample_images, sample_gt = next(iter(test_loader))
        sample_images = sample_images.to(device)
        sample_gt = sample_gt.to(device)
        sample_output = model(sample_images)
        sample_preds = torch.argmax(sample_output, dim=1)

        print(f"Sample Predictions: {sample_preds.cpu().numpy()}, Sample Targets: {sample_gt.cpu().numpy()}")

    if checkpoint_path:
        torch.save(model.state_dict(), f"{checkpoint_path}_epoch_{epoch+1}.pth")

print("🏁 Training completed.")
```

สร้าง function train() โดยนำมาจาก Lab5_2

```
### START CODE HERE ###
import torch.optim as optim

model = CustomLeNet()
opt = optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.MSELoss()
train(model, opt, loss_fn, train_loader, test_loader, epochs=20, checkpoint_path='checkpoint/model',

### END CODE HERE ###
```

เริ่ม train model โดยใช้ optimizer เป็น Adam และ loss_fn เป็น MSELoss() ซึ่ง train โดยใช้ 20 epochs

```
def compute_confusion_matrix(model, test_loader, device):
    all_preds = []
    all_targets = []

    with torch.no_grad():
        for images, gt in test_loader:
            images = images.to(device)
            gt = gt.to(device)

            output = model(images)
            preds = torch.argmax(output, dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_targets.extend(gt.cpu().numpy())

    num_classes = len(set(all_targets))
    cm = compute_confusion_matrix_manual(all_preds, all_targets, num_classes)
    return cm
```

```
### START CODE HERE ###
def compute_confusion_matrix_manual(preds, targets, num_classes):
    confusion_matrix = np.zeros((num_classes, num_classes), dtype=int)

    for p, t in zip(preds, targets):
        confusion_matrix[t][p] += 1

    return confusion_matrix
```

สร้าง function compute_confusion_matrix_manual() เพื่อใช้คำนวณหา confusion matrix

```
Confusion Matrix:
[[ 70  44]
 [ 11 108]]
```

โดยตำแหน่ง row 1 col 1 เป็นจำนวนครั้งที่ model predict class 0 ถูก

ตำแหน่ง row 1 col 2 คือ จำนวนครั้งที่ model predict class 0 ผิด

ตำแหน่ง row 2 col 1 คือจำนวนครั้งที่ model predict class 1 ผิด

ตำแหน่ง row 2 col 2 คือจำนวนครั้งที่ model predict class 1 ถูก

```python
def compute_metrics(confusion_matrix):
    metrics = {}

    true_positives = np.diag(confusion_matrix)
    total_samples = confusion_matrix.sum()
    metrics['accuracy'] = true_positives.sum() / total_samples

    precision = np.zeros(confusion_matrix.shape[0])
    recall = np.zeros(confusion_matrix.shape[0])
    f1_score = np.zeros(confusion_matrix.shape[0])

    for i in range(confusion_matrix.shape[0]):
        precision[i] = true_positives[i] / confusion_matrix[:, i].sum() if confusion_matrix[:, i].sum() > 0 else 0
        recall[i] = true_positives[i] / confusion_matrix[i].sum() if confusion_matrix[i].sum() > 0 else 0
        f1_score[i] = 2 * (precision[i] * recall[i]) / (precision[i] + recall[i]) if (precision[i] + recall[i]) > 0 else 0

    metrics['precision'] = precision
    metrics['recall'] = recall
    metrics['f1_score'] = f1_score

    metrics['macro_precision'] = precision.mean()
    metrics['macro_recall'] = recall.mean()
    metrics['macro_f1_score'] = f1_score.mean()

    return metrics
```

สร้าง function compute_metrics() เพื่อใช้คำนวณค่า Accuracy Precision Recall แล้วนำมาคำนวณค่า F1

```
Metrics:
Accuracy: 0.7639
Precision per class: [0.86419753 0.71052632]
Recall per class: [0.61403509 0.90756303]
F1 Score per class: [0.71794872 0.79704797]
Macro Precision: 0.7874
Macro Recall: 0.7608
Macro F1 Score: 0.7575
```

```python
model = CustomLeNet()
checkpoint = torch.load('checkpoint/model_epoch_20.pth')
model.load_state_dict(checkpoint)
model.eval()
```

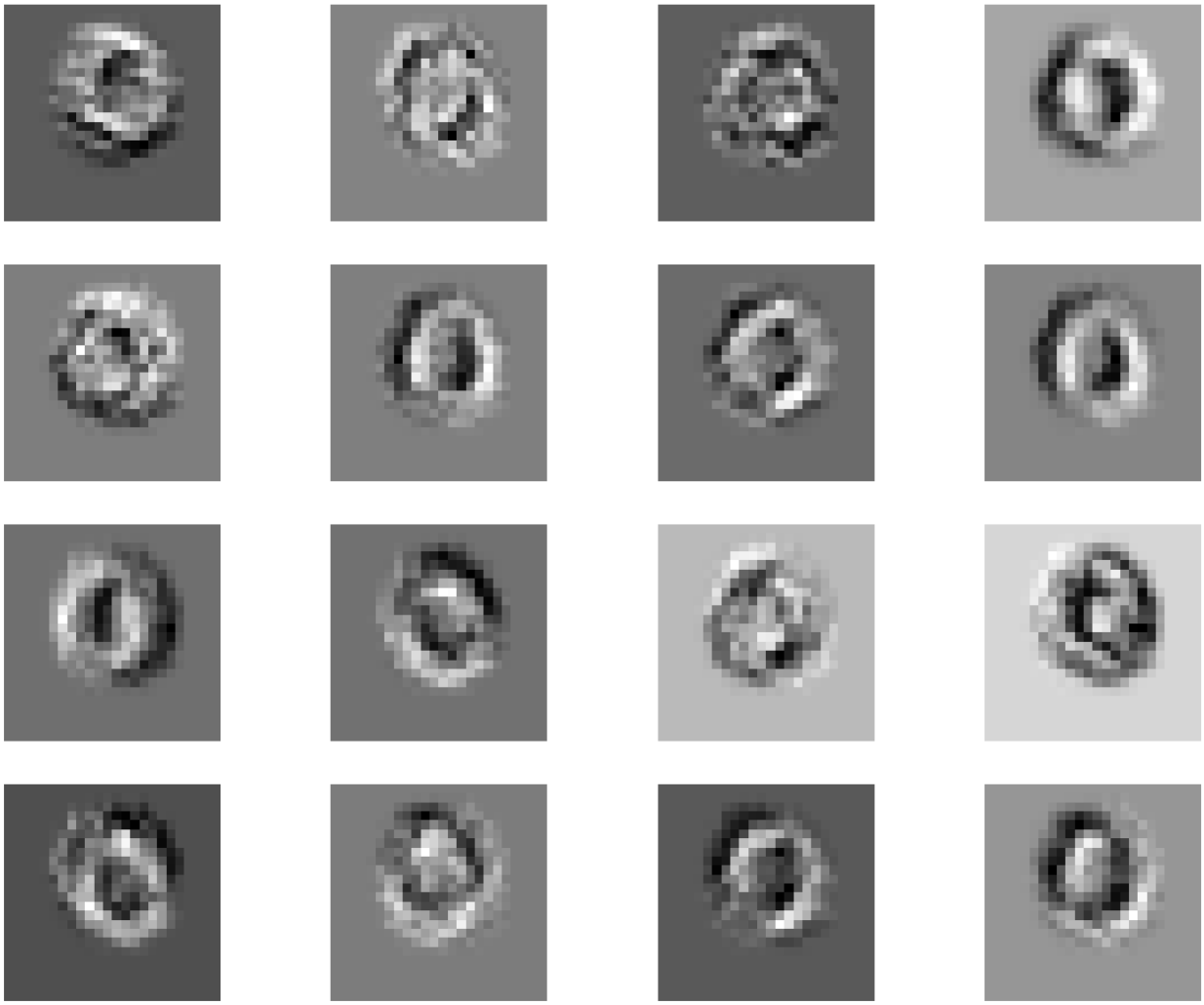
ทำการโหลด model ที่เซฟไว้มาใช้ดู features map

```python
feature_maps = []

def hook_fn(module, input, output):
    feature_maps.append(output.cpu().detach())

model.conv1.register_forward_hook(hook_fn)
model.conv2.register_forward_hook(hook_fn)

sample_image, _ = next(iter(test_loader))

device = 'cuda' if torch.cuda.is_available() else 'cpu'
sample_image = sample_image.to(device)
model.to(device)

with torch.no_grad():
    model(sample_image)
```

```python
def visualize_feature_maps(feature_maps):
    for i, fmap in enumerate(feature_maps):
        num_channels = fmap.shape[1]
        cols = 4
        rows = (num_channels + cols - 1) // cols

        fig, axes = plt.subplots(rows, cols, figsize=(15, rows * 3))
        fig.suptitle(f"Feature Maps for Layer {i+1}", fontsize=16)

        axes = axes.flatten() if rows > 1 else axes

        for j in range(num_channels):
            ax = axes[j]
            ax.imshow(fmap[0, j].cpu().detach().numpy(), cmap='gray')
            ax.axis('off')

        for ax in axes.flat[num_channels:]:
            ax.axis('off')

        plt.savefig(f'fmap/feature_map_layer_{i+1}.png')
        plt.show()

visualize_feature_maps(feature_maps)
```

จะได้ผลลัพท์ดังนี้โดยเป็นภาพที่ผ่านการ conv 2 รอบ

Feature Maps for Layer 1



Feature Maps for Layer 2