ณสิต ผลัญชัย 65010273

วิวัตร เตชะโกศล 65011001

Lab8

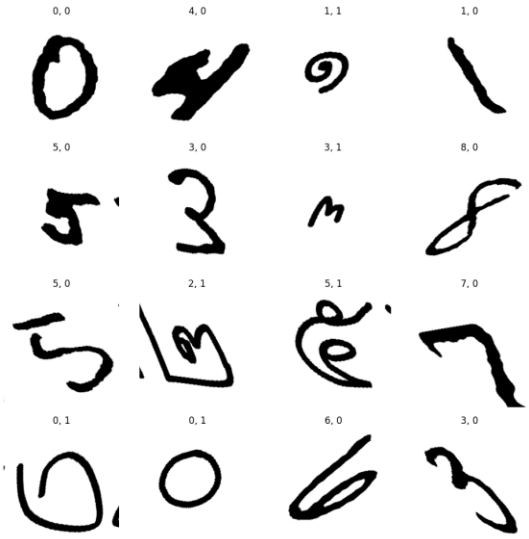
```python
### START CODE HERE ###
class MultiLanguageHandwrittenDataset(Dataset):
    def __init__(self, root_dirs, languages, transforms=None):
        self.root_dirs = root_dirs
        self.languages = languages
        self.transform = transforms
        self.samples = []
        for root_dir, language in zip(self.root_dirs, self.languages):
            for subdir in os.scandir(root_dir):
                if subdir.is_dir():
                    label = int(subdir.name)
                    for file in os.scandir(subdir.path):
                        if file.is_file():
                            self.samples.append((file.path, label, language))
        print(self.samples)

    def __len__(self):
        return len(self.samples)

    def invert(self, image):
        return Image.fromarray(255 - np.array(image).astype(np.uint8))
```

```python
    def __getitem__(self, idx):
        image_path, label, language = self.samples[idx]

        image = Image.open(image_path).convert('L')

        image_array = np.array(image)
        mean_pixel_value = np.mean(image_array)

        if mean_pixel_value < 128:
            image = self.invert(image)

        if self.transform:
            image = self.transform(image)

        label = torch.tensor(label)
        language = torch.tensor(language)

        return image, label, language
### END CODE HERE ###
```

สร้าง class MultiLanguageHandwrittingDataset() เพื่อไว้จัดการโหลดภาพและ label ว่าเป็นเลขอะไรรวมถึงภาษาอะไร ซึ่งจากการตรวจสอบ Dataset เบื้องต้นพบว่ามีบางภาพที่ background เป็นสีขาวตัวหนังสือสีดำและบางภาพ background สีดำตัวหนังสือสีขาว เมื่อนำไป train อาจทำให้ model สับสนได้จึงทำการตรวจสอบด้วยการหาค่าเฉลี่ยของภาพถ้าน้อยกว่า 128 จาก 255 มีโอกาสที่จะเป็น background สีดำมากกว่าก็ให้ทำการ invert ภาพจาก background สีดำตัวหนังสือสีขาวให้กลายเป็นภาพที่ background เป็นสีขาวตัวหนังสือสีดำทั้งหมด

```
### START CODE HERE ###
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((224, 224), antialias=True),
    transforms.Pad(223, padding_mode='reflect'),
    transforms.RandomAffine(degrees=15, shear=45),
    transforms.CenterCrop(224),
])

root_dirs = [r'C:\Users\Nickv\Documents\ImageProcessing\Week8\data\eng-handwritten-dataset', r'C:\Users\N
languages = [0, 1] #English = 0, Thai = 1

dataset = MultiLanguageHandwrittenDataset(root_dirs=root_dirs, languages=languages, transforms=transform)
dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

batch, label, language = next(iter(dataloader))
imshow_grid(batch, label, language)

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

ทำการ transform เปลี่ยนภาพเป็น tensor, Resize ภาพเป็น (224, 224) เพื่อให้เข้ากับ model vgg16 จากนั้นทำการ random หมุนภาพเพื่อเพิ่มความหลากหลายให้กับ dataset จากนั้นทำการ padding ภาพ เพื่อให้ภาพที่หมุนแล้วไม่เหลือขอบดำๆแล้วทำการ centercrop ตัดภาพให้เหลือขนาด (224, 224) เท่าเดิม จะได้ผลดังรูป



```
### START CODE HERE ###
class customVGG16(nn.Module):
    def __init__(self, add_feat_dims=None, h_dims=None, num_classes=10, input_size=(1, 224, 224), trainable_layers_idx=None):
        super(customVGG16, self).__init__()
        self.vgg16 = models.vgg16(pretrained=True)

        self.vgg16.features[0] = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

        #Freeze Feature Extractor Layers
        for param in self.vgg16.features[:].parameters():
            param.requires_grad = False

        #Unfreeze Classifier Layers
        for param in self.vgg16.classifier.parameters():
            param.requires_grad = True

        if trainable_layers_idx is not None:
            for idx in trainable_layers_idx:
                for param in self.vgg16.features[idx].parameters():
                    param.requires_grad = True
```

สร้าง class customVGG16() ซึ่งเป็น class ที่เราจะทำการ transfer learning นำ model ของ VGG16 มา train เพิ่มเติม โดยจะเปลี่ยนเฉพาะ feature layer แรกให้รับภาพ 1 channel ได้ เนื่องจากเราใช้ภาพ Grayscale จากนั้นทำการ Freeze parameters ของ features layers ที่เหลือไว้ แล้ว unfreeze parameters เฉพาะ classifier layers

```
    if add_feat_dims:
        features = []
        for i, out_ch in enumerate(add_feat_dims):
            if i == 0:
                features.append(nn.Conv2d(512, out_ch, kernel_size=3, padding=1))
            else:
                features.append(nn.Conv2d(add_feat_dims[i-1], out_ch, kernel_size=3, padding=1))
            features.append(nn.ReLU())
        self.vgg16.features = nn.Sequential(self.vgg16.features, *features)

    input_size_fc1 = self._get_input_size_fc(input_size)

    if h_dims:
        layers = []
        for i, hdim in enumerate(h_dims):
            if i == 0:
                layers.append(nn.Linear(input_size_fc1, hdim))
            else:
                layers.append(nn.Linear(h_dims[i-1], hdim))
            layers.append(nn.Dropout(0.4))
            layers.append(nn.ReLU())
        self.vgg16.classifier = nn.Sequential(*layers, nn.Linear(h_dims[-1], num_classes))
    else:
        self.vgg16.classifier = nn.Linear(input_size_fc1, num_classes)
```

```
    def _get_input_size_fc(self, input_shape):
        with torch.no_grad():
            x = torch.zeros(1, *input_shape)
            x = self.vgg16.features(x)
            x = self.vgg16.avgpool(x)
            x = torch.flatten(x, 1)

            return x.size(1)

    def forward(self, x):
        x = self.vgg16.features(x)
        x = self.vgg16.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.vgg16.classifier(x)
        return x

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

โดย class นี้ยังสามารถใช้ trainable_layers_idx ในการ unfreeze layers ที่ต้องการ ใช้ add_feat_dims ในการเพิ่ม features layers และใช้ h_dims ในการเพิ่ม fully connected layers ได้ และสุดท้าย num_classes ใช้เพื่อบอกจำนวน class output ของ model

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

def train(model, opt, loss_fn, train_loader, val_loader, epochs=10, writer=None, checkpoint_path=None, device='cuda', task='digit'):
    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, digits, languages in train_bar:
            images = images.to(device)

            if task == 'digit':
                digits = digits.to(device)
                outputs = model(images)
                loss = loss_fn(outputs, digits)
            elif task == 'language':
                languages = languages.to(device)
                outputs = model(images)
                loss = loss_fn(outputs, languages)
```

Function train() เหมือนในแลปก่อนๆ เพิ่มเติม parameter task เพื่อบอกว่า train ในส่วนของ digit หรือ language ทำให้ function เดียวสามารถใช้ได้ทั้งคู่ รวมถึงมีการเพิ่ม writer ซึ่งผมใช้ SummaryWriter ของ tensor board โดยจะทำการ save ค่า train loss และ validation loss เก็บไว้ใน path ที่เราตั้งและแสดงผลทาง tensorboard

```
            opt.zero_grad()
            loss.backward()
            opt.step()

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

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

        avg_train_loss /= step

        model.eval()
        avg_val_loss = 0.0
        val_bar = tqdm(val_loader, desc='📄 Validation', unit='batch')
```

```
        with torch.no_grad():
            for images, digits, languages in val_bar:
                images = images.to(device)

                if task == 'digit':
                    digits = digits.to(device)
                    outputs = model(images)
                    val_loss = loss_fn(outputs, digits)
                elif task == 'language':
                    languages = languages.to(device)
                    outputs = model(images)
                    val_loss = loss_fn(outputs, languages)

                avg_val_loss += val_loss.item()

        avg_val_loss /= len(val_loader)

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

        if writer:
            writer.add_scalar('Loss/Train', avg_train_loss, epoch + 1)
            writer.add_scalar('Loss/Validation', avg_val_loss, epoch + 1)

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

    print("▩▩ Training completed.")
```

ตัวอย่างการเรียกใช้งาน ในการ train model1 digits classification

```
### START CODE HERE ###
import torch.optim as optim
▶ Launch TensorBoard Session
from torch.utils.tensorboard import SummaryWriter

model1 = customVGG16()

writer = SummaryWriter(log_dir='./runs/digit_experiment')
opt = optim.Adam(model1.parameters(), lr=0.001)
loss_fn = nn.CrossEntropyLoss()
train(model=model1, opt=opt, loss_fn=loss_fn, epochs=20, train_loader=train_loader, val_loader=val_loader,
writer.close()

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

```python
def evaluate_task(y_true, y_pred, target_names="Task"):
    clf_report = classification_report(y_true, y_pred, target_names=target_names, zero_division=1)
    conf_matrix = confusion_matrix(y_true, y_pred)
    print("\nClassification Report:\n", clf_report)

    plt.figure(figsize=(8, 6))
    plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()

    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names, rotation=45)
    plt.yticks(tick_marks, target_names)

    thresh = conf_matrix.max() / 2.
    for i, j in np.ndindex(conf_matrix.shape):
        plt.text(j, i, format(conf_matrix[i, j], 'd'), ha="center", va="center",
                 color="white" if conf_matrix[i, j] > thresh else "black")

    plt.ylabel('Actual')
    plt.xlabel('Predicted')

    plt.tight_layout()
    plt.show()
```

Function evaluate_task() เป็นฟังก์ชันที่ใช้แสดงผลลัพธ์ในการ test ของ model ซึ่งจะแสดง report และ confusion matrix

```python
dataset_size = len(dataset)
train_size = int(0.7 * dataset_size)
val_size = int(0.15 * dataset_size)
test_size = dataset_size - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

ทำการแบ่งสัดส่วน train validation test เป็น 70:15:15 และ random_split()

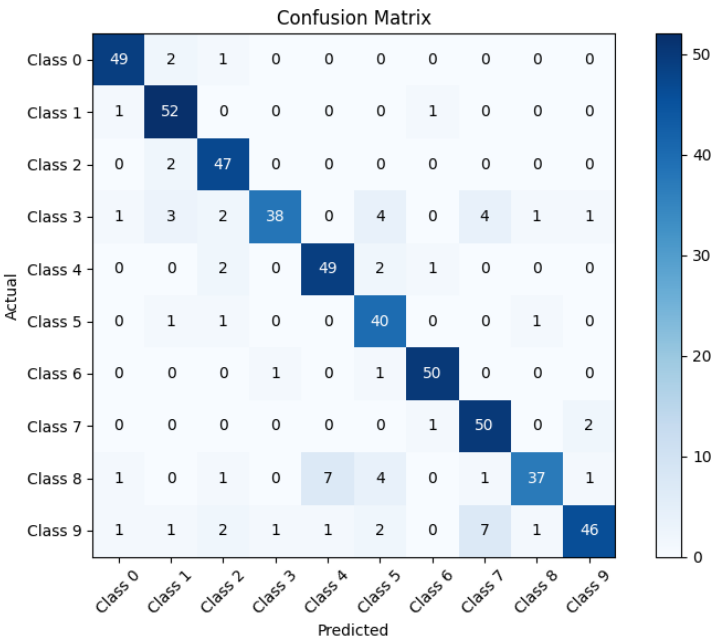
```
🚀 Training Epoch [16/20]: 100%|██████████| 154/154 [01:44<00:00,  1.47batch/s, loss=0.00707]
📄 Validation: 100%|██████████| 33/33 [00:22<00:00,  1.45batch/s]
Epoch [16/20], Train Loss: 0.3382, Val Loss: 0.4836
🚀 Training Epoch [17/20]: 100%|██████████| 154/154 [01:44<00:00,  1.47batch/s, loss=0.296]
📄 Validation: 100%|██████████| 33/33 [00:22<00:00,  1.47batch/s]
Epoch [17/20], Train Loss: 0.3204, Val Loss: 0.4902
🚀 Training Epoch [18/20]: 100%|██████████| 154/154 [01:44<00:00,  1.47batch/s, loss=0.0193]
📄 Validation: 100%|██████████| 33/33 [00:22<00:00,  1.47batch/s]
Epoch [18/20], Train Loss: 0.2931, Val Loss: 0.5032
🚀 Training Epoch [19/20]: 100%|██████████| 154/154 [01:44<00:00,  1.47batch/s, loss=0.659]
📄 Validation: 100%|██████████| 33/33 [00:22<00:00,  1.48batch/s]
Epoch [19/20], Train Loss: 0.2905, Val Loss: 0.4375
🚀 Training Epoch [20/20]: 100%|██████████| 154/154 [01:44<00:00,  1.47batch/s, loss=0.25]
📄 Validation: 100%|██████████| 33/33 [00:22<00:00,  1.48batch/s]
Epoch [20/20], Train Loss: 0.2785, Val Loss: 0.4920
🏁 Training completed.
```

หลังจากทำการ train model1 20 epochs เสร็จสิ้น

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

device = 'cuda' if torch.cuda.is_available() else 'cpu'
model1 = model1.to(device)
all_labels = []
all_predictions = []

for images, labels, _ in test_loader:
    images = images.to(device)
    labels = labels.to(device)
    with torch.no_grad():
        outputs = model1(images)
        _, predicted = torch.max(outputs, 1)

    all_labels.extend(labels.cpu().numpy())
    all_predictions.extend(predicted.cpu().numpy())

all_labels = np.array(all_labels)
all_predictions = np.array(all_predictions)

evaluate_task(all_labels, all_predictions, target_names=[f'Class {i}' for i in range(10)])

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

ทำการแสดงผลด้วย function evaluate_task() ซึ่งจะรับค่า predictions ของ model และคำตอบที่ถูกต้องจาก test_loader มาแสดงผลดังด้านล่าง

```
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.92      0.94      0.93        52
     Class 1       0.85      0.96      0.90        54
     Class 2       0.84      0.96      0.90        49
     Class 3       0.95      0.70      0.81        54
     Class 4       0.86      0.91      0.88        54
     Class 5       0.75      0.93      0.83        43
     Class 6       0.94      0.96      0.95        52
     Class 7       0.81      0.94      0.87        53
     Class 8       0.93      0.71      0.80        52
     Class 9       0.92      0.74      0.82        62

    accuracy                           0.87       525
   macro avg       0.88      0.88      0.87       525
weighted avg       0.88      0.87      0.87       525
```



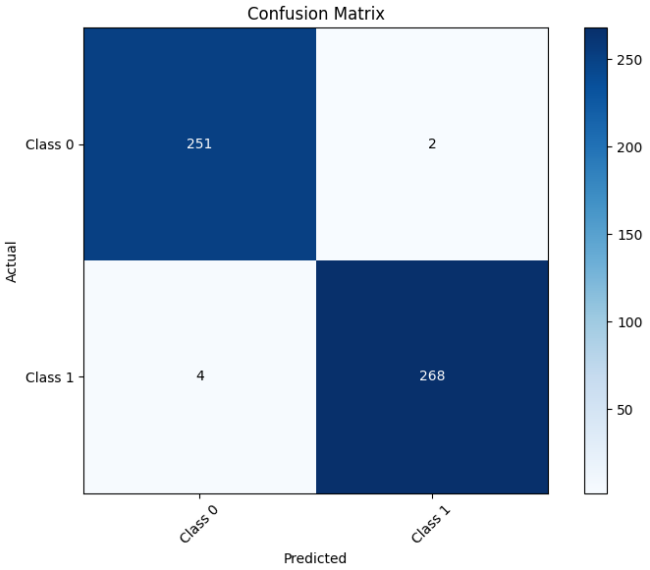
```
### START CODE HERE ###
import torch.optim as optim
▶ Launch TensorBoard Session
from torch.utils.tensorboard import SummaryWriter

model2 = customVGG16(num_classes=2)

writer = SummaryWriter(log_dir='./runs/language_experiment')
opt = optim.Adam(model2.parameters(), lr=0.001)
loss_fn = nn.CrossEntropyLoss()
train(model=model2, opt=opt, loss_fn=loss_fn, epochs=10, train_loader=train_loader,
writer.close()

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

ทำการ train model2 languages classification 10epochs

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

device = 'cuda' if torch.cuda.is_available() else 'cpu'
all_labels = []
all_predictions = []
model2 = model2.to(device)

for images, _, languages in test_loader:
    images = images.to(device)
    languages = languages.to(device)

    with torch.no_grad():
        outputs = model2(images)
        _, predicted = torch.max(outputs, 1)

    all_labels.extend(languages.cpu().numpy())
    all_predictions.extend(predicted.cpu().numpy())

all_labels = np.array(all_labels)
all_predictions = np.array(all_predictions)

evaluate_task(all_labels, all_predictions, target_names=['Class 0', 'Class 1'])

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

แสดงผลลัพธ์การ predict ของ model2

```
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.98      0.99      0.99       253
     Class 1       0.99      0.99      0.99       272

    accuracy                           0.99       525
   macro avg       0.99      0.99      0.99       525
weighted avg       0.99      0.99      0.99       525
```



```python
class customVGG16_multitask(nn.Module):
    def __init__(self, add_feat_dims=None, h_dims=None, input_size=(1, 224, 224),
        super(customVGG16_multitask, self).__init__()
        self.vgg16 = models.vgg16(pretrained=True)

        self.vgg16.features[0] = nn.Conv2d(in_channels=1, out_channels=64, kernel_

        #Freeze Feature Extractor Layers
        for param in self.vgg16.features[:].parameters():
            param.requires_grad = False

        #Unfreeze Classifier Layers
        for param in self.vgg16.classifier.parameters():
            param.requires_grad = True

        if trainable_layers_idx is not None:
            for idx in trainable_layers_idx:
                for param in self.vgg16.features[idx].parameters():
                    param.requires_grad = True
```

```python
    if add_feat_dims:
        features = []
        for i, out_ch in enumerate(add_feat_dims):
            if i == 0:
                features.append(nn.Conv2d(512, out_ch, kernel_size=3, padding=1))
            else:
                features.append(nn.Conv2d(add_feat_dims[i-1], out_ch, kernel_size=3, padding=1))
            features.append(nn.ReLU())
        self.vgg16.features = nn.Sequential(self.vgg16.features, *features)

    input_size_fc1 = self._get_input_size_fc(input_size)

    if h_dims:
        self.digit = self._create_fc_layers(input_size_fc1, h_dims, 10)
        self.lang = self._create_fc_layers(input_size_fc1, h_dims, 2)
    else:
        self.digit = nn.Linear(input_size_fc1, 10)
        self.lang = nn.Linear(input_size_fc1, 2)
```

สร้าง class customVGG16_multitask() เพื่อทำ multitask learning โดยใช้ model vgg16 เหมือนเดิมซึ่งภายใน class ส่วนใหญ่จะเหมือนเดิมแต่จะมี classifier เปลี่ยนไปโดยจะแยกเป็นของ digit และ language ส่วนที่เหลือเป็น shared layers

```python
def _create_fc_layers(self, input_dim, h_dims, output_dim):
    layers = []
    for i, hdim in enumerate(h_dims):
        if i == 0:
            layers.append(nn.Linear(input_dim, hdim))
        else:
            layers.append(nn.Linear(h_dims[i-1], hdim))
        layers.append(nn.Dropout(0.4))
        layers.append(nn.ReLU())
    layers.append(nn.Linear(h_dims[-1], output_dim))
    return nn.Sequential(*layers)

def _get_input_size_fc(self, input_shape):
    with torch.no_grad():
        x = torch.zeros(1, *input_shape)
        x = self.vgg16.features(x)
        x = self.vgg16.avgpool(x)
        x = torch.flatten(x[0])
        return x.size(0)
```

```python
def forward(self, x):
    x = self.vgg16.features(x)
    x = self.vgg16.avgpool(x)
    x = torch.flatten(x, 1)

    out_digit = self.digit(x)
    out_lang = self.lang(x)

    return out_digit, out_lang
```

Return output สองตัวคือของ digit และ language

```python
def train_multi(model, opt, loss_fn, train_loader, val_loader, epochs=10, writer=None, checkpoint_path
    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, digits, languages in train_bar:
            images = images.to(device)
            digits = digits.to(device)
            languages = languages.to(device)

            out_digits, out_languages = model(images)

            loss_digits = loss_fn(out_digits, digits)
            loss_languages = loss_fn(out_languages, languages)
            loss = loss_digits + loss_languages

            opt.zero_grad()
            loss.backward()
            opt.step()

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

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

        avg_train_loss /= step
```

```python
model.eval()
avg_val_loss = 0.0
val_bar = tqdm(val_loader, desc='📄 Validation', unit='batch')

with torch.no_grad():
    for images, digits, languages in val_bar:
        images = images.to(device)
        digits = digits.to(device)
        languages = languages.to(device)

        out_digits, out_languages = model(images)

        loss_digits = loss_fn(out_digits, digits)
        loss_languages = loss_fn(out_languages, languages)
        val_loss = loss_digits + loss_languages

        avg_val_loss += val_loss.item()

avg_val_loss /= len(val_loader)

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

if writer:
    writer.add_scalar('Loss/Train', avg_train_loss, epoch + 1)
    writer.add_scalar('Loss/Validation', avg_val_loss, epoch + 1)
```

Function train_multi() เหมือนกับ function train ปกติแค่มี output 2 ตัว เวลาคิด loss ต้องนำ loss ของทั้งสอง task มาบวกกัน

```
### START CODE HERE ###
import torch.optim as optim
▶ Launch TensorBoard Session
from torch.utils.tensorboard import SummaryWriter

model = customVGG16_multitask()

dataset_size = len(dataset)
train_size = int(0.7 * dataset_size)
val_size = int(0.15 * dataset_size)
test_size = dataset_size - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

writer = SummaryWriter(log_dir='./runs/custom_vgg16_experiment')
opt = optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.CrossEntropyLoss()
train_multi(model=model, opt=opt, loss_fn=loss_fn, train_loader=train_loader, val_loader=val_loader, epochs=10,
writer.close()

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

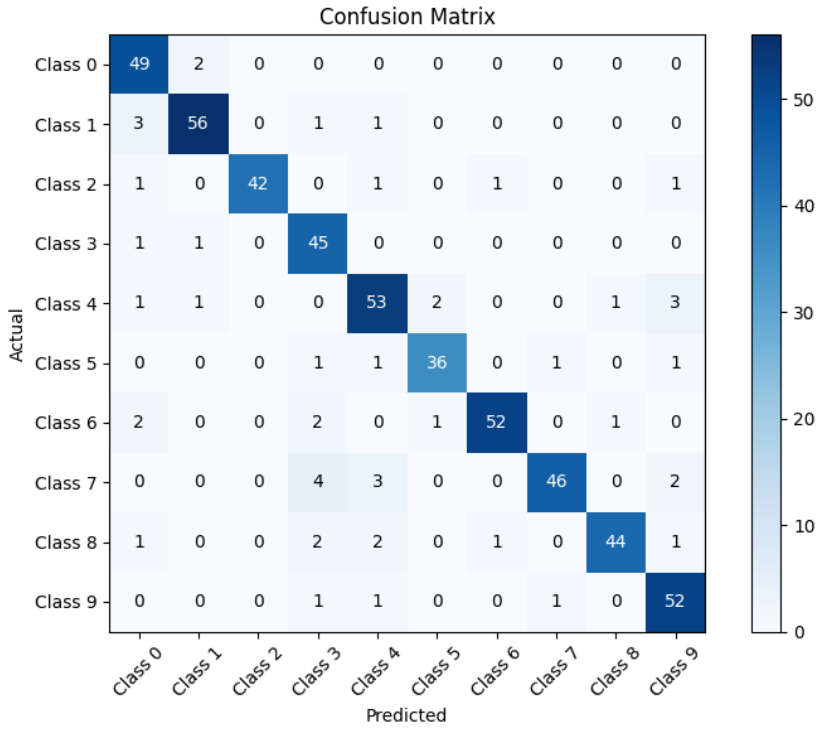
ทำการ train model multitask 10 epochs จากนั้นให้ model predict และเก็บผลของทั้งสอง task ไว้

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

device = 'cuda' if torch.cuda.is_available() else 'cpu'
all_digits = []
all_digits_pred = []
all_languages = []
all_languages_pred = []
model = model.to(device)

for images, digits, languages in test_loader:
    images = images.to(device)
    digits = digits.to(device)
    languages = languages.to(device)

    with torch.no_grad():
        out_digit, out_lang = model(images)
        _, digit_pred = torch.max(out_digit, 1)
        _, language_pred = torch.max(out_lang, 1)

    all_digits.extend(digits.cpu().numpy())
    all_digits_pred.extend(digit_pred.cpu().numpy())
    all_languages.extend(languages.cpu().numpy())
    all_languages_pred.extend(language_pred.cpu().numpy())

all_digits = np.array(all_digits)
all_digits_pred = np.array(all_digits_pred)
all_languages = np.array(all_languages)
all_languages_pred = np.array(all_languages_pred)

evaluate_task(all_digits, all_digits_pred, target_names=[f'Class {i}' for i in range(10)])
evaluate_task(all_languages, all_languages_pred, target_names=[f'Class {i}' for i in range(2)])

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

นำมาแสดงผลด้วย function evaluate_task() จะได้ผลลัพธ์ดังนี้

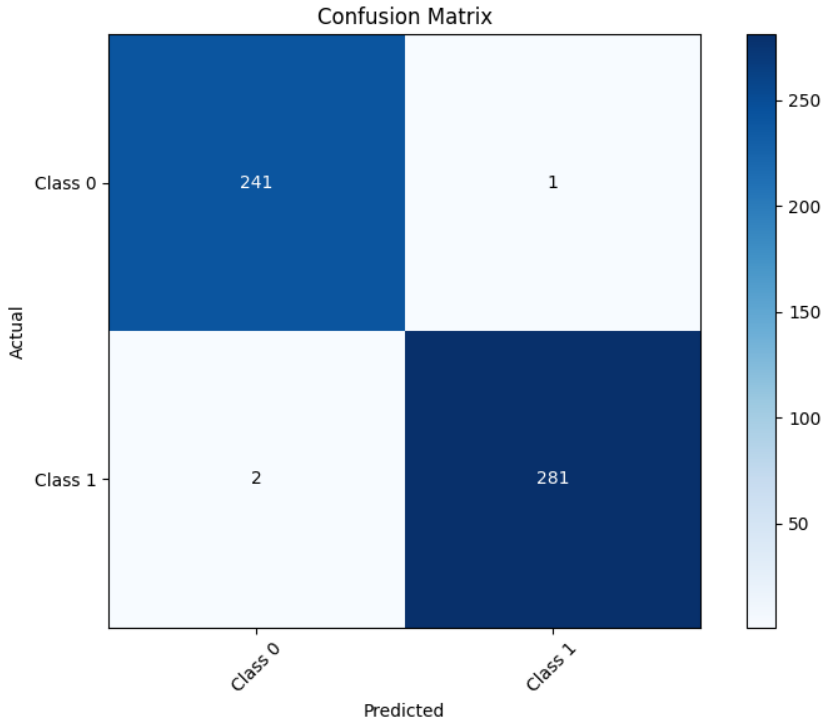
Digits classification

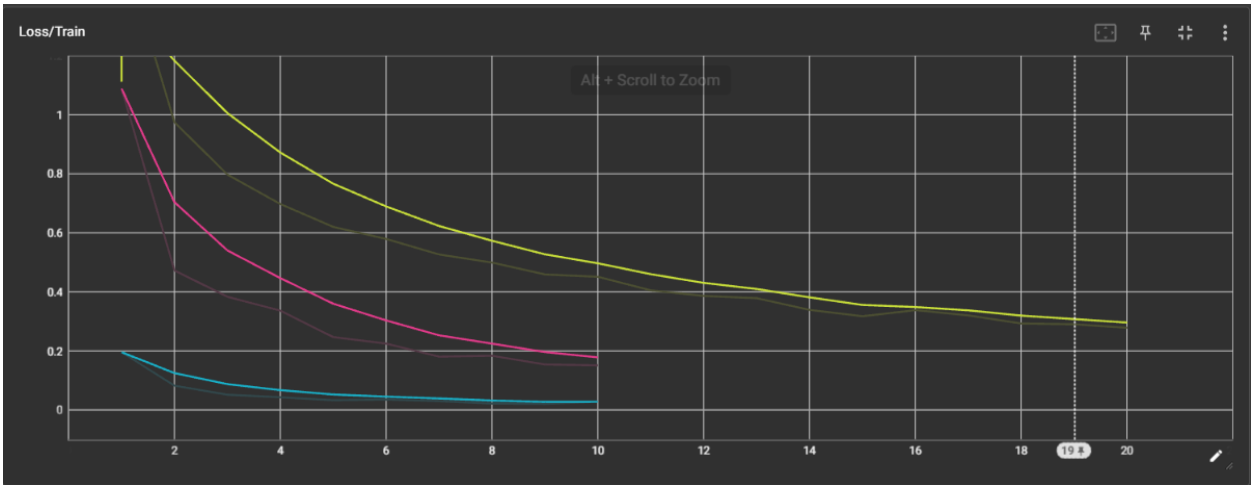
```
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.84      0.96      0.90        51
     Class 1       0.93      0.92      0.93        61
     Class 2       1.00      0.91      0.95        46
     Class 3       0.80      0.96      0.87        47
     Class 4       0.85      0.87      0.86        61
     Class 5       0.92      0.90      0.91        40
     Class 6       0.96      0.90      0.93        58
     Class 7       0.96      0.84      0.89        55
     Class 8       0.96      0.86      0.91        51
     Class 9       0.87      0.95      0.90        55

    accuracy                           0.90       525
   macro avg       0.91      0.91      0.91       525
weighted avg       0.91      0.90      0.91       525
```

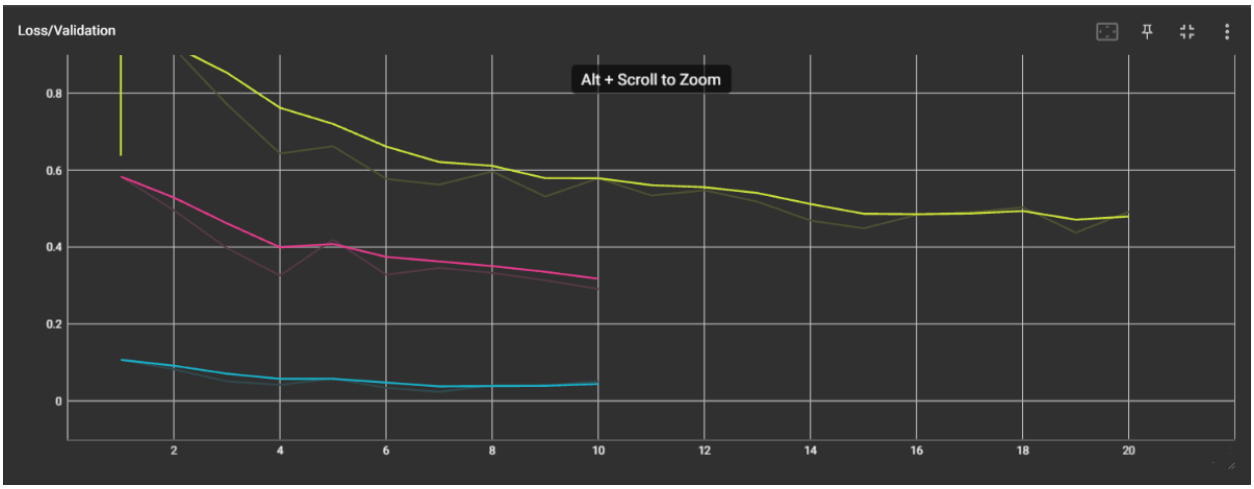


Languages Classification

```
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.99      1.00      0.99       242
     Class 1       1.00      0.99      0.99       283

    accuracy                           0.99       525
   macro avg       0.99      0.99      0.99       525
weighted avg       0.99      0.99      0.99       525
```







ผลลัพธ์ Train loss และ Validation loss จาก tensorboard โดย สีเขียวคือกราฟของ Digits classification model สีชมพูคือของ languages classification model และสีฟ้าคือของ Multitask model