HW9 Facial Keypoint Detection
B083040029 邱品諺

本次作業的目的為標示出臉部特徵的位置，將利用 FacialKeypointDataset 作為訓練、驗證、測試之資料集，此資料集的資料型態為 {“image”: image, “keypoints”: keypoints}，每張圖片有 15 個 keypoints 標示圖片中臉部器官的特徵，每個 keypoint 由 x, y 座標表示，且每張圖片為 96x96 的 gray scale 圖片。

Facial Keypoint Detection 可視為 Regression 的問題，目標為預測 30 個[-1, 1]的點並將預測值和 ground truth 利用 Mean Square Error 計算 Loss。

模型的架構主要先透過多層的 Convolution layer 擷取低階特徵、將低階特徵組合為高階特徵，再將組合後的高階特徵打平送入 Fully-connected layer。另外，在 Convolution layer 及 Fully-connected layer 後會通過 Non-linearity，且在 Convolution layer、ReLU 後會加上 Max pooling 以降低維度，最後在 CNN 後面加上 Dropout layer 並將機率設定為 0.5，以防止模型 overfitting。另外，由於通過的 Non-linearity 為 ReLU，因此在權重初始化的部分本程式使用 Kaiming initialization。下列為模型架構的程式：

```
self.cnn = nn.Sequential(
    nn.Conv2d(1, 16, 3, 1, 1),      # [16, 96, 96]
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),          # [16, 48, 48]

    nn.Conv2d(16, 32, 3, 1, 1),     # [32, 48, 48]
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),          # [32, 24, 24]

    nn.Conv2d(32, 64, 3, 1, 1),     # [64, 24, 24]
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),          # [64, 12, 12]

    nn.Conv2d(64, 128, 3, 1, 1),    # [128, 12, 12]
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),          # [128, 6, 6]
    nn.Dropout2d(0.5),

    nn.Flatten()
)
self.fc = nn.Sequential(
    nn.Linear(128 * 6 * 6, 256),
    nn.ReLU(),
    nn.Linear(256, 30)
)
```

```
def initialize_weights(m):
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_uniform_(m.weight.data,nonlinearity='relu')
        if m.bias is not None:
            nn.init.constant_(m.bias.data, 0)
    elif isinstance(m, nn.Linear):
        nn.init.kaiming_uniform_(m.weight.data)
        nn.init.constant_(m.bias.data, 0)
```

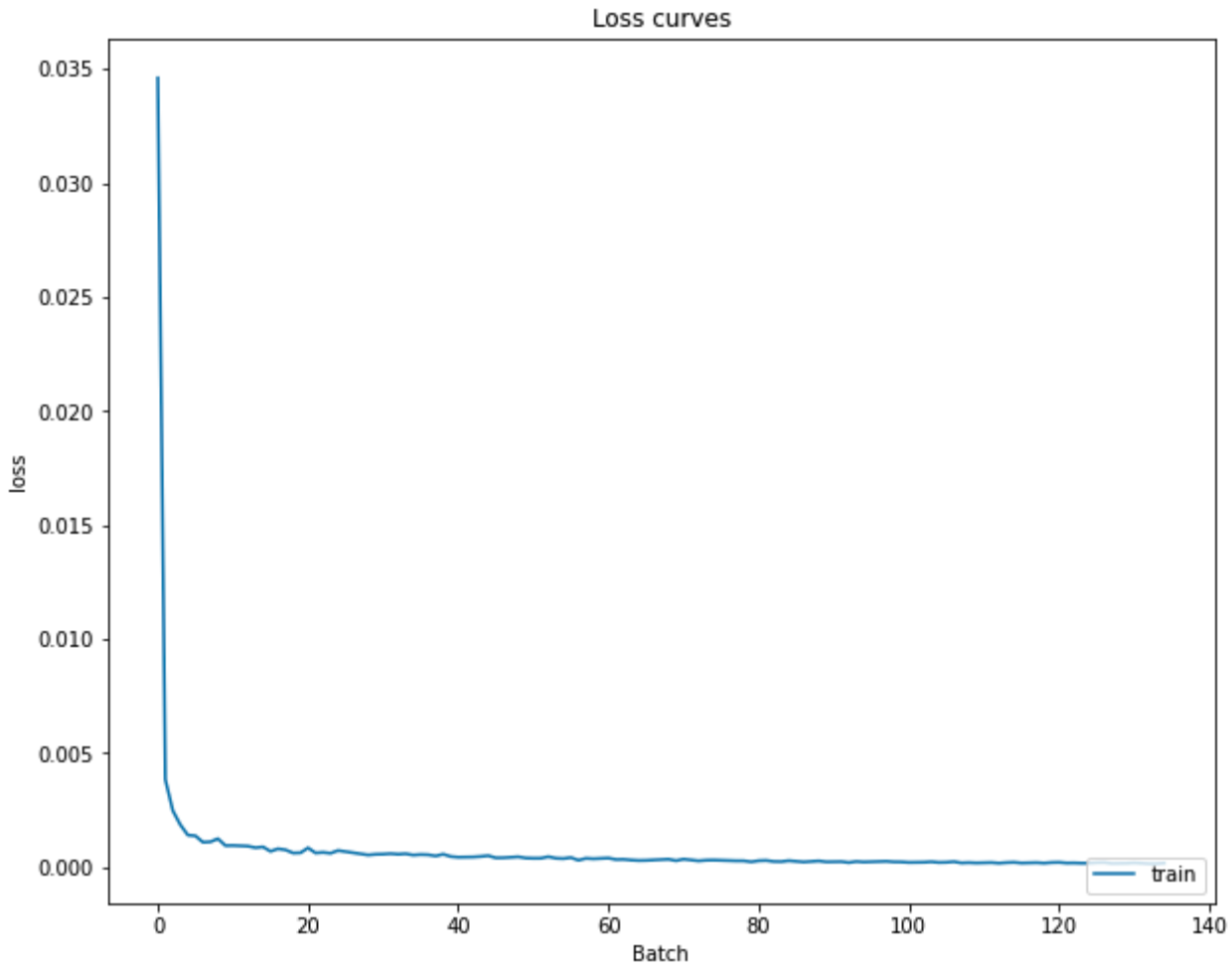
模型的 hyperparameters：

```
hparams = {
    # TODO: if you have any model arguments/hparams, define them here
    "lr": 0.001,
    "batch_size": 16,
    "num_epoch": 15
}
```

在模型訓練的部分，Loss function 使用 Mean Square Error，optimizer 使用 Adam，並設定 learning rate decay 使 learning rate 隨著 epoch 增加逐漸減小，下圖為模型訓練的程式：

```
################################################################################
# TODO - Train Your Model                                                      #
################################################################################
import torch.nn as nn
import torch.optim as optim

lr = hparams["lr"]
batch_size = hparams["batch_size"]
num_epoch = hparams["num_epoch"]

train_loss_history = []

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), weight_decay = 1e-7, lr=lr)
# scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=5)
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
# training sample: 1546

model.train()
for epoch in range(num_epoch):
    running_loss = 0.0
    for i, data in enumerate(train_dataloader):
        x, y = data["image"], data["keypoints"]
        # x, y = x.to(device), y.to(device)
        y = y.view(y.size(0), -1)

        y_pred = model(x)
        loss = criterion(y_pred, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # scheduler.step(1)

        running_loss += loss.item()

        if i % 10 == 9:
            running_loss /= len(train_dataloader)
            train_loss_history.append(running_loss)
            print("[Epoch %3d, Batch %3d] loss: %f" % (epoch+1, i+1, running_loss))
            running_loss = 0.0

################################################################################
#                           END OF YOUR CODE                                   #
################################################################################
```

由於使用 Adam 作為 Optimizer，由下圖可見初始 Loss 下降十分快速，但是訓練的後期有些許的上下波動，而非平緩的下降。



最後，由下圖結果數據可知 Loss 小於 0.005 且 Score 大於 100。另外，本程式有試過利用 Batch Normalization 加在每一層 Convolution layer 及 Fully-connected layer 後以取代 Dropout 防止 overfitting，但是訓練出的數據較使用 Dropout 還差。

```
loss: 0.001086
Score: 460.59505930009067
```