說明：請各位使用此template進行Report撰寫，如果想要用其他排版模式也請註明題號以及題目內容（請勿擅自更改題號），最後上傳前，請務必轉成PDF檔，並且命名為report.pdf，否則將不予計分。

學號：r14921A13　系級：電機所碩一　姓名：鄭皓中

1. (0.5%) CNN model

(a)Paste the complete code of the CNN used in your submission.

```python
class FaceExpressionNet(nn.Module):        You, 20 小時前 • add hw2 …
    def __init__(self):
        super().__init__()

        self.relu = nn.LeakyReLU(0.1)

        # === 卷積區塊 (與您的原始程式碼相同) ===
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=3, padding=1), nn.BatchNorm2d(64), self.relu,
            nn.Conv2d(64, 64, kernel_size=3, padding=1), nn.BatchNorm2d(64), self.relu,
            nn.MaxPool2d(kernel_size=2, stride=2) # 尺寸: 64x64 -> 32x32
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.BatchNorm2d(128), self.relu,
            nn.Conv2d(128, 128, kernel_size=3, padding=1), nn.BatchNorm2d(128), self.relu,
            nn.MaxPool2d(kernel_size=2, stride=2) # 尺寸: 32x32 -> 16x16
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=3, padding=1), nn.BatchNorm2d(256), self.relu,
            nn.Conv2d(256, 256, kernel_size=3, padding=1), nn.BatchNorm2d(256), self.relu,
            nn.MaxPool2d(kernel_size=2, stride=2) # 尺寸: 16x16 -> 8x8
        )

        # 傳統全展平: 256 * 8 * 8 = 16384
        INPUT_SIZE = 256 * 8 * 8  # 16384

        self.fc_layers = nn.Sequential(
            nn.Dropout(0.3),
            # ⭐ FC 層輸入尺寸設為 16384
            nn.Linear(INPUT_SIZE, 256),
            nn.BatchNorm1d(256),
            self.relu,
            nn.Dropout(0.3),
            nn.Linear(256, 7)
        )

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)

        # ⭐ 傳統展平操作，保留所有空間資訊
        x = x.view(x.size(0), -1) # 尺寸: BATCH x 16384

        x = self.fc_layers(x)
        return x
```

(b) Describe the structure of your model:

- How many convolutional layers?
  3 convolutional layers

- Did you use batch normalization or dropout, are these useful to have better performance, explain why or why not ?
  我使用了 BatchNorm2d及LeakyReLU，使用LeakyReLU避免梯度小於或等於0時造成的死亡ReLU問題，BatchNorm2d則是穩定整體輸出並且加速收斂的速度。
  最後有經過Norm的模型表現較好，但是差別並沒有到很大。

- How did you design or modify the output layer?
  由於在沒有使用dropout的時候有觀察到後面的epoch雖然train的loss持續下降，validation的loss卻停滯沒有變化，所以使用dropout來防止overfitting狀況發生。
  後面使用Flatten直接將高維的Input展開成16384，再投影到256。
  BatchNorm1d則是提高模型輸出的穩定性。

(c) If you used a pretrained model, answer: No pretrained model.

2. (1%) Data Augmentation

(a) Paste the code for the data augmentation you implemented

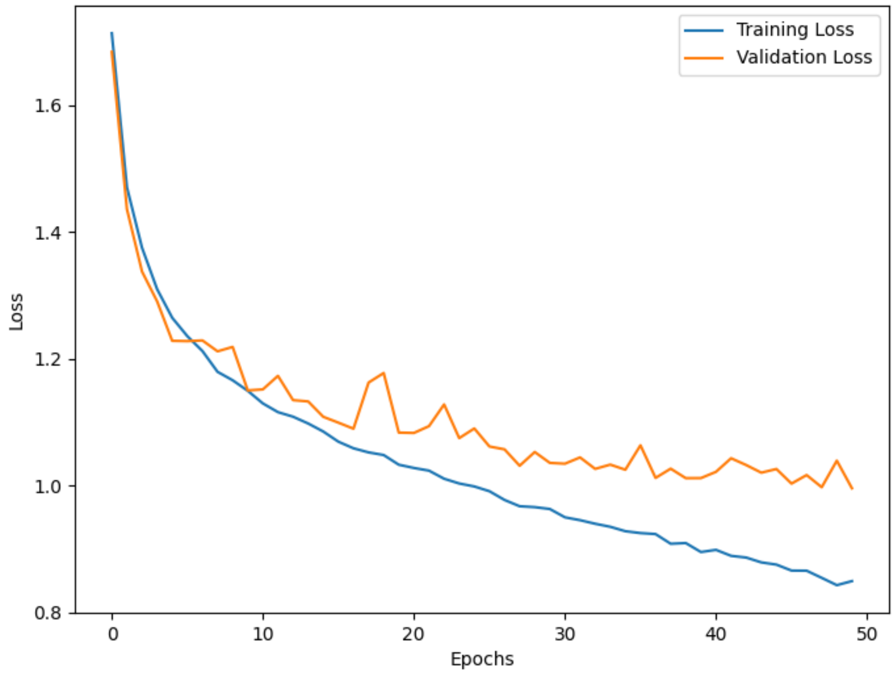
```
train_tfm = T.Compose([
    # 灰度圖片通常是 48x48，但原始讀取是 3 通道灰度圖，需要處理
    T.Grayscale(num_output_channels=1),
    T.Resize((64, 64)),
    # 1. 資料增強
    T.RandomHorizontalFlip(p=0.5),  # 左右翻轉
    T.RandomRotation(15),           # 隨機旋轉
    T.RandomAffine(degrees=10, translate=(0.1, 0.1), scale=(0.8, 1.2)), # 仿射變換

    # 2. 轉換與標準化
    T.ToTensor(),
    T.RandomErasing(p=0.25, scale=(0.02, 0.2), ratio=(0.3, 3.3)),

    # 對單通道灰度圖進行標準化 (請使用您資料集實際的 Mean/Std，這裡使用常見值)
    T.Normalize(mean=MEAN, std=STD),
])

eval_tfm = T.Compose([
    T.Grayscale(num_output_channels=1),
    T.Resize((64, 64)),
    T.ToTensor(),
    T.Normalize(mean=MEAN, std=STD),
])
```

(b)Explain the reasoning behind your chosen augmentation methods.

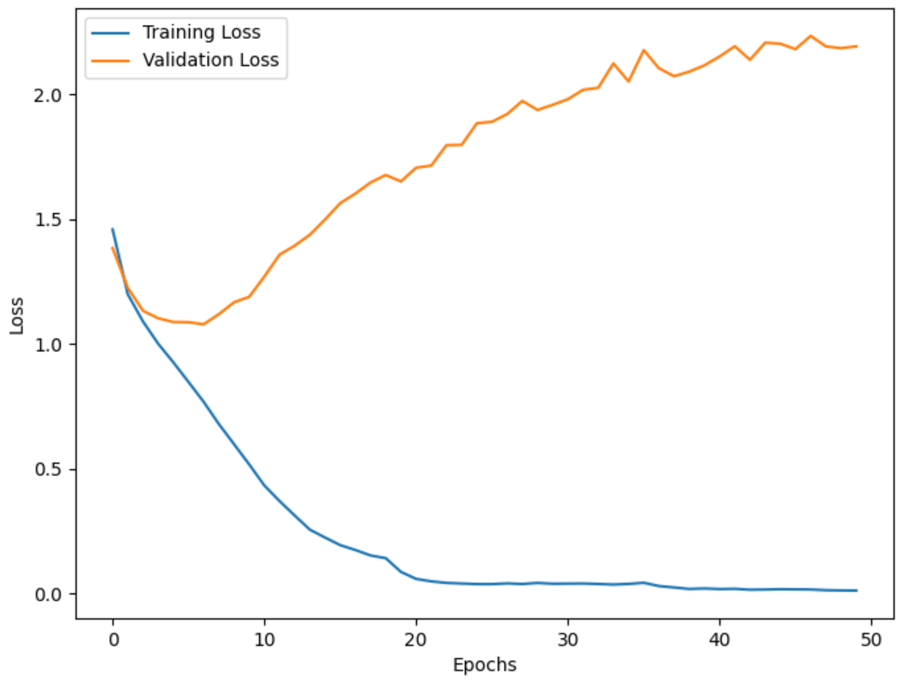
1. RandomHorizontalFlip, RandomRotation, RandomAffine: 透過對圖片進行旋轉、翻轉、變換的操作，避免Model只學習到正臉照片而overfitting，也讓Model可以更多的學到不同角度的識別
2. RandomErasing: 為了讓Model不要過度注意某些細節，使用RandomErasing會將部分的圖片進行遮擋，讓Model能夠更關注群體特徵。
3. Normalize: 將整個圖片進行Normalize，一樣是為了讓Model關注整體特徵而非特定的部分。

(c) Provide two sets of training/validation loss curves:

- With augmentation.



- Without augmentation.

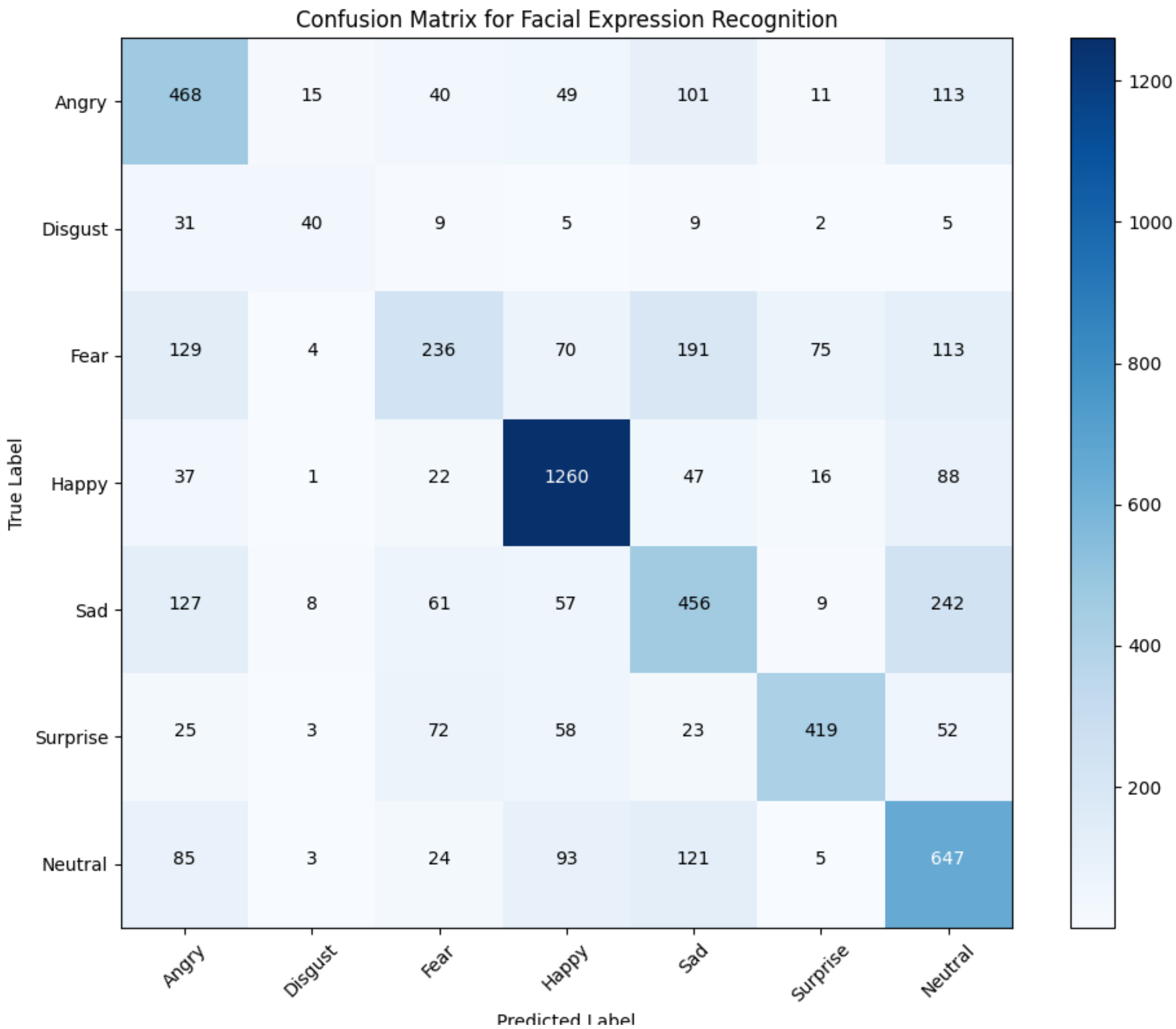


(d) Compare and explain the differences between the two settings.
觀察with/without augmentation的loss圖可以發現，經過augmentation的data得到的loss圖validation跟Train的趨勢比較相近，而沒有經過augmentation的data在Train的表現很好，但Validation卻反而loss增加，可以知道augmentation確實避免了overfitting的狀況發生，也達成了我預期中避免過擬合的結果。

3. **(0.5%) Confusion Matrix**

(a) Paste the code used to generate the confusion matrix and include the resulting figure(confusion matrix).

```python
# 將列表轉換為 NumPy 陣列
y_true = np.array(labels)
y_pred = np.array(predictions)

# 2. 手動計算混淆矩陣 (代替 sklearn.metrics.confusion_matrix)
num_classes = 7 # 您的情緒類別數量
cm = np.zeros((num_classes, num_classes), dtype=int)

for true_label, pred_label in zip(y_true, y_pred):
    # cm[i, j] 是真實標籤 i 被預測為 j 的次數
    cm[true_label, pred_label] += 1

# 假設情緒類別 (0-6) 對應的標籤名稱
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']

# 3. 繪製混淆矩陣
plt.figure(figsize=(10, 8))

# 使用 imshow 繪製顏色熱力圖
# cmap=plt.cm.Blues 是 Matplotlib 內建的藍色色階
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Facial Expression Recognition')
plt.colorbar() # 添加顏色條

tick_marks = np.arange(num_classes)
plt.xticks(tick_marks, emotion_labels, rotation=45) # 設置 X 軸標籤
plt.yticks(tick_marks, emotion_labels)              # 設置 Y 軸標籤

# 4. 在每個格子上添加數值標註 (Annotation)
thresh = cm.max() / 2.0
for i in range(num_classes):
    for j in range(num_classes):
        # 根據背景顏色，調整文字顏色 (深色背景用白色字，淺色背景用黑色字)
        color = "white" if cm[i, j] > thresh else "black"

        # 使用 plt.text 函式，將數值放置在格子中央
        plt.text(j, i, format(cm[i, j], 'd'), # 'd' 表示整數格式
                 horizontalalignment="center",
                 color=color)

plt.tight_layout() # 自動調整佈局，避免標籤重疊
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```



(b) Analyze which classes are most frequently misclassified and explain possible reasons.
Fear是最容易被misclassified的，從confusion matrix可以看到Fear有較高的比例被誤判成angry, sad, neutral，這可能是因為在經過資料增強及處理之後Fear跟其他幾類有較為相似的特徵。