學號：r12227118　　系級：心理所碩二　姓名：張瑋宸

1. (0.5%) CNN model
    a. 貼上private submission所使用的CNN model程式碼。

```
class FaceExpressionNet(nn.Module):
  def __init__(self):
    super().__init__()

    self.conv = nn.Sequential(
      nn.Conv2d(1, 256, kernel_size=3, stride=1, padding=1),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=2, stride=2),
      nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=2, stride=2),
      nn.Dropout(p=0.3)
    )

    self.fc = nn.Sequential(
      nn.Linear(32 * 16 * 16, 128),
      nn.ReLU(),
      nn.Linear(128, 7)
    )

  def forward(self, x):
    x = self.conv(x)
    x = x.view(-1, 32 * 16 * 16)

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

2. (1%) Data Augmentation (ref: https://pytorch.org/vision/stable/transforms.html)
    a. 貼上這部分的程式碼，並說明為何使用這些augmentation。
    b. 畫出使用/不使用 augmentation時的training/validation loss curve，比較並說明其差異。

a. 採用水平翻轉 (HorizontalFlip(p=0.5)) 、20度的隨機旋轉(RandomRotation(20))，是為了讓CNN接受的臉部表情資料有更多的角度，不是只有正面or 特定側面，不採用更大的旋轉角度是因為大部分的表情仍沒有如上下顛倒的資料，輸入大角度甚至上下顛倒的表情可能會讓模型更難找到重要的feature。另外還加上RandomPerspective，可以讓正面臉的資料也可能產生不同視角看到的側面，增加資料多樣性。

```
train_tfm = v2.Compose([
  v2.ToImage(), # Convert to tensor, only needed if you had a PIL image
  v2.ToDtype(torch.uint8, scale=True), # Ensure input is uint8
  v2.RandomHorizontalFlip(p=0.5), # Horizontal flip with 50% probability
  v2.RandomRotation(20), # Random rotation between -20° to 20°
  v2.RandomPerspective(distortion_scale=0.3, p=0.4), # Random perspective distortion
  v2.ToDtype(torch.float32, scale=True), # Convert to float
])
eval_tfm = v2.Compose([
  v2.ToImage(), # Convert to tensor, only needed if you had a PIL image
  v2.ToDtype(torch.float32, scale=True), # Convert to float
])
```

b. 如下圖，左圖是不使用augmentation，右圖是使用augmentation後，可以看到不使用augmentation會使training 的時候很快產生overfitting的狀況，training loss下降很快但是vaidation loss則下降一段時間後就開始上升。相對而言，Data augmentation後訓練過程就較無overfitting。

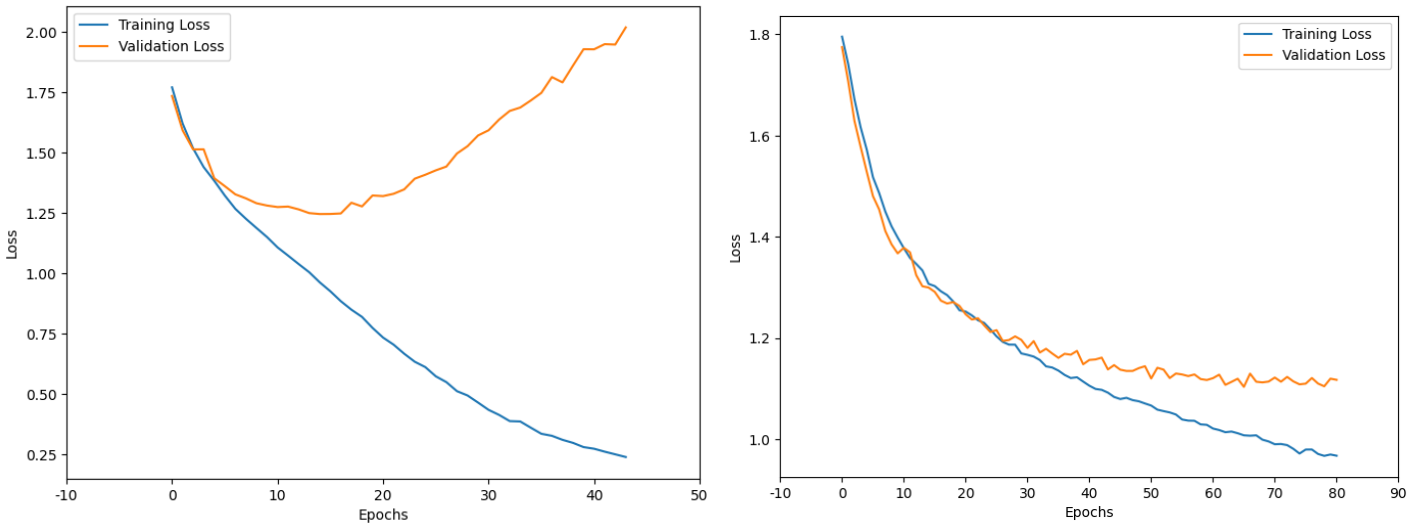

圖 1 Training/Validation Curve

3. (0.5%) Confusion Matrix (ref: https://en.wikipedia.org/wiki/Confusion_matrix)
    a. 貼上這部分的程式碼。
    b. 分析哪些類別的圖片容易使 model 搞混，並分析可能的原因。

a.

```
# Assited by Gemini
def draw_confusion_matrix(model, valid_loader):
  predictions, labels = [], []
  model.to(device)
  model.eval()
  with torch.no_grad():
    for img, lab in tqdm(valid_loader):
      img = img.to(device)
      output = model(img)
      predictions += torch.argmax(output, dim=-1).tolist()
      labels += lab.tolist()
    cm = np.zeros((7, 7), dtype=int)
    for label, prediction in zip(labels, predictions):
      cm[label, prediction] += 1
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
  classes = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
  # Plot the confusion matrix using matplotlib
  plt.figure(figsize=(8, 6))
  plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
  plt.title('Confusion Matrix')
  plt.colorbar(fraction=0.046, pad=0.04)

  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation=45, ha='right') # Rotate x-axis labels
  plt.yticks(tick_marks, classes)

  # Improved text display:
  thresh = cm.max() / 2.
  for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
     plt.text(j, i, format(np.round(cm[i, j],2)),
         horizontalalignment="center",
         color="white" if cm[i, j] > thresh else "black",
         fontsize=10 # Adjust font size as needed
         )
  plt.tight_layout()
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  plt.show()

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

b. 就以下圖2 confusion matrix中非主對角線中數值>0.2的格子舉例: Fear 較容易被分成Sad, Sad較容易被分成Neutral (而Neutral亦較容易被分成Sad).
猜測原因可能是兩類的資料本身就較類似。挑出training data中標籤為Sad/Neutral的資料進行觀察(圖3)，的確發現有些sad的資料表情較不誇張時，的確跟Neutral看起來十分類似。

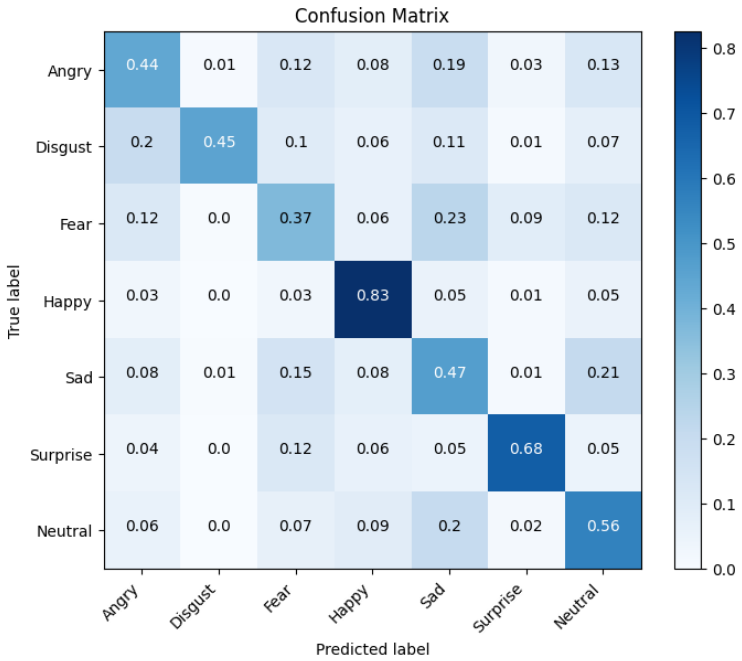

圖 2 : Confusion Matrix

圖 3: Sad (左49張)與Neutral(右49張)之training data sample