CNN 方向預測報告

111511239 仇健安

1. 資料前處理：天線選擇

本實驗中使用的原始資料為多天線收發所得的雷達訊號，我從中擷取特定天線配置並保留關鍵資訊。具體配置如下：

- 接收端（Rx）：選取第 9 根至第 16 根天線，共 8 根
- 發送端（Tx）：選取第 5 根至第 12 根天線，共 8 根

這個選擇，可產生總共 $8 \times 8 = 64$ 組 Tx-Rx 組合，對應於輸入資料形狀為 (64, 64, 2)，其中最後一維代表實部與虛部（共兩通道）。

2. CNN 模型架構

我採用深度卷積神經網路來進行方向分類預測，設計上使用了 5 個卷積區塊，並搭配 Batch Normalization、GELU 激活函數與 Global Average Pooling，再經過多層全連接層進行輸出。模型結構如下：

```
model = models.Sequential([
    layers.InputLayer(input_shape=(64, 64, 2)),

    # Conv Block 1 ~ 4：逐漸增加 channel 數
    layers.Conv2D(64, 3, padding='same'),
    layers.BatchNormalization(),
    layers.Activation(gelu),
    layers.MaxPooling2D(pool_size=(2, 2)),  # 64 → 32

    layers.Conv2D(128, 3, padding='same'),
    layers.BatchNormalization(),
    layers.Activation(gelu),
    layers.MaxPooling2D(pool_size=(2, 2)),  # 32 → 16

    layers.Conv2D(256, 3, padding='same'),
    layers.BatchNormalization(),
    layers.Activation(gelu),
    layers.MaxPooling2D(pool_size=(2, 2)),  # 16 → 8

    layers.Conv2D(512, 3, padding='same'),
    layers.BatchNormalization(),
```

```
    layers.Activation(gelu),
    layers.MaxPooling2D(pool_size=(2, 2)),  # 8 → 4

    # Conv Block 5：新增 1024 維通道
    layers.Conv2D(1024, 3, padding='same'),
    layers.BatchNormalization(),
    layers.Activation(gelu),

    # Global feature
    layers.GlobalAveragePooling2D(),

    # Fully Connected
    layers.Dense(256, activation=gelu),
    layers.Dense(128, activation=gelu),
    layers.Dense(64, activation=gelu),
    layers.Dense(32, activation=gelu),
    layers.Dense(8, activation='sigmoid')
])
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 64)	1,216
batch_normalization (BatchNormalization)	(None, 64, 64, 64)	256
activation (Activation)	(None, 64, 64, 64)	0
max_pooling2d (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_1 (Conv2D)	(None, 32, 32, 128)	73,856
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 128)	512
activation_1 (Activation)	(None, 32, 32, 128)	0
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_2 (Conv2D)	(None, 16, 16, 256)	295,168
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 256)	1,024
activation_2 (Activation)	(None, 16, 16, 256)	0
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 256)	0
conv2d_3 (Conv2D)	(None, 8, 8, 512)	1,180,160
batch_normalization_3 (BatchNormalization)	(None, 8, 8, 512)	2,048
activation_3 (Activation)	(None, 8, 8, 512)	0
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 512)	0
conv2d_4 (Conv2D)	(None, 4, 4, 1024)	4,719,616
batch_normalization_4 (BatchNormalization)	(None, 4, 4, 1024)	4,096
activation_4 (Activation)	(None, 4, 4, 1024)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1024)	0
dense (Dense)	(None, 256)	262,400
dense_1 (Dense)	(None, 128)	32,896
dense_2 (Dense)	(None, 64)	8,256
dense_3 (Dense)	(None, 32)	2,080
dense_4 (Dense)	(None, 8)	264

Total params: 6,583,848 (25.12 MB)
Trainable params: 6,579,880 (25.10 MB)
Non-trainable params: 3,968 (15.50 KB)

3. 損失函數（Loss Function）

本任務屬於多標籤分類問題，因此我們選擇 binary_crossentropy 作為損失函數：

```
loss='binary_crossentropy'
```

這代表每一個輸出方向都是獨立的二元分類問題（是否為該方向），而非互斥的單一分類。

4. 正確率計算（Exact Match Accuracy）

每筆資料的真實標籤是長度為 8 的 0-1 向量，代表可能同時具有多個方向的目標，因此正確率必須考慮整體預測是否與真值接近，也就是給予高過某個門檻的預測值為目標所在地。

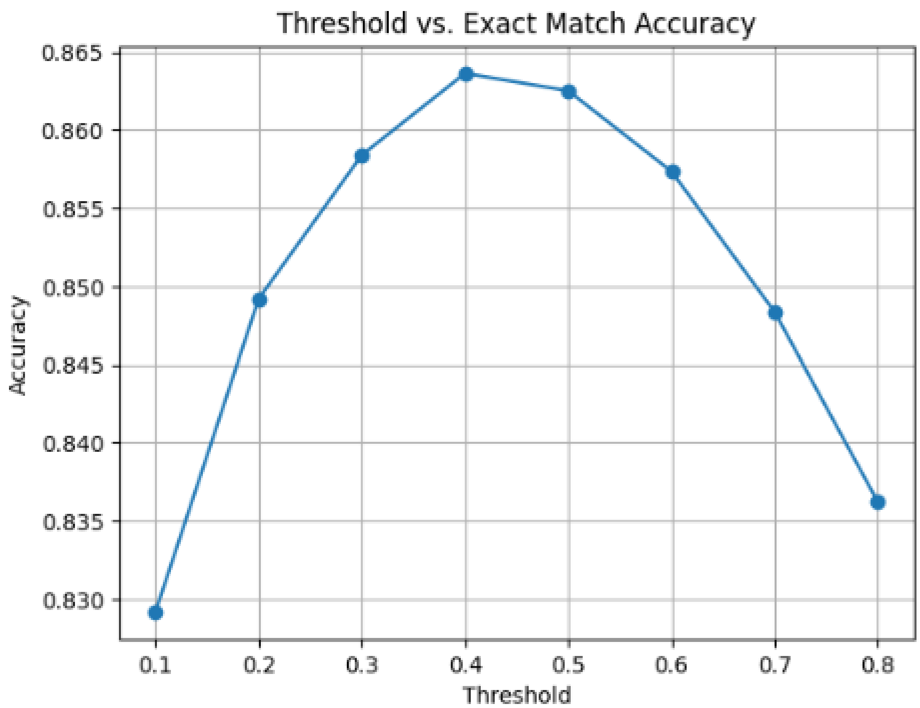
```
class ExactMatchThreshold(tf.keras.metrics.Metric):
    def __init__(self, threshold=0.5, name="exact_match_accuracy", **kwargs):
        super().__init__(name=name, **kwargs)
        self.threshold = threshold
        self.total = self.add_weight(name="total", initializer="zeros")
        self.correct = self.add_weight(name="correct", initializer="zeros")

    def update_state(self, y_true, y_pred, sample_weight=None):
        y_pred_bin = tf.cast(y_pred >= self.threshold, tf.int32)
        y_true_bin = tf.cast(y_true, tf.int32)

        match = tf.reduce_all(tf.equal(y_pred_bin, y_true_bin), axis=1)
        match = tf.cast(match, self.dtype)

        self.total.assign_add(tf.cast(tf.shape(y_true)[0], self.dtype))
        self.correct.assign_add(tf.reduce_sum(match))

    def result(self):
        return self.correct / self.total

    def reset_states(self):
        self.total.assign(0.0)
        self.correct.assign(0.0)
```

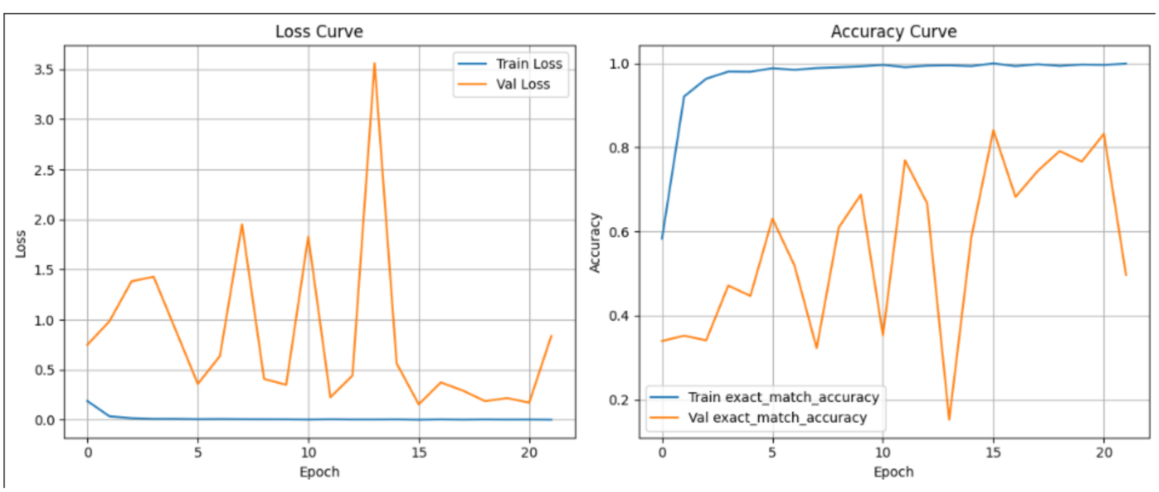
5. 預測結果與圖表（Exact Match Accuracy）

最後我用這個模型得到的最高正確率為: 0.8871

```
exact_match_accuracy: 0.8871
```

但因為我是睡覺的時候跑的，所以 server 關掉，沒有畫到圖，因此以下附上的是簡單跑幾次的 0.865 左右正確率的 Threshold vs Accuracy 圖和 loss/acc curve，基本上模型簡單跑就會有 0.86 的正確率。





Final Accuracy: 0.8871~=89%