Optimizer and regularization comparison experiment

1. Optimizer
   - SGD
   - Momentum
   - RMSProp
   - Adam
2. Regularization
   - none（不使用正則化）
   - L1 正則化
   - L2 正則化
   - Dropout
   - Batch Normalization

   本次實驗對所有所有組合（4 種優化器 × 5 種正則化 = 20 組）進行訓練來觀察比較不同優化器和正則化下的模型表現。

3. Code

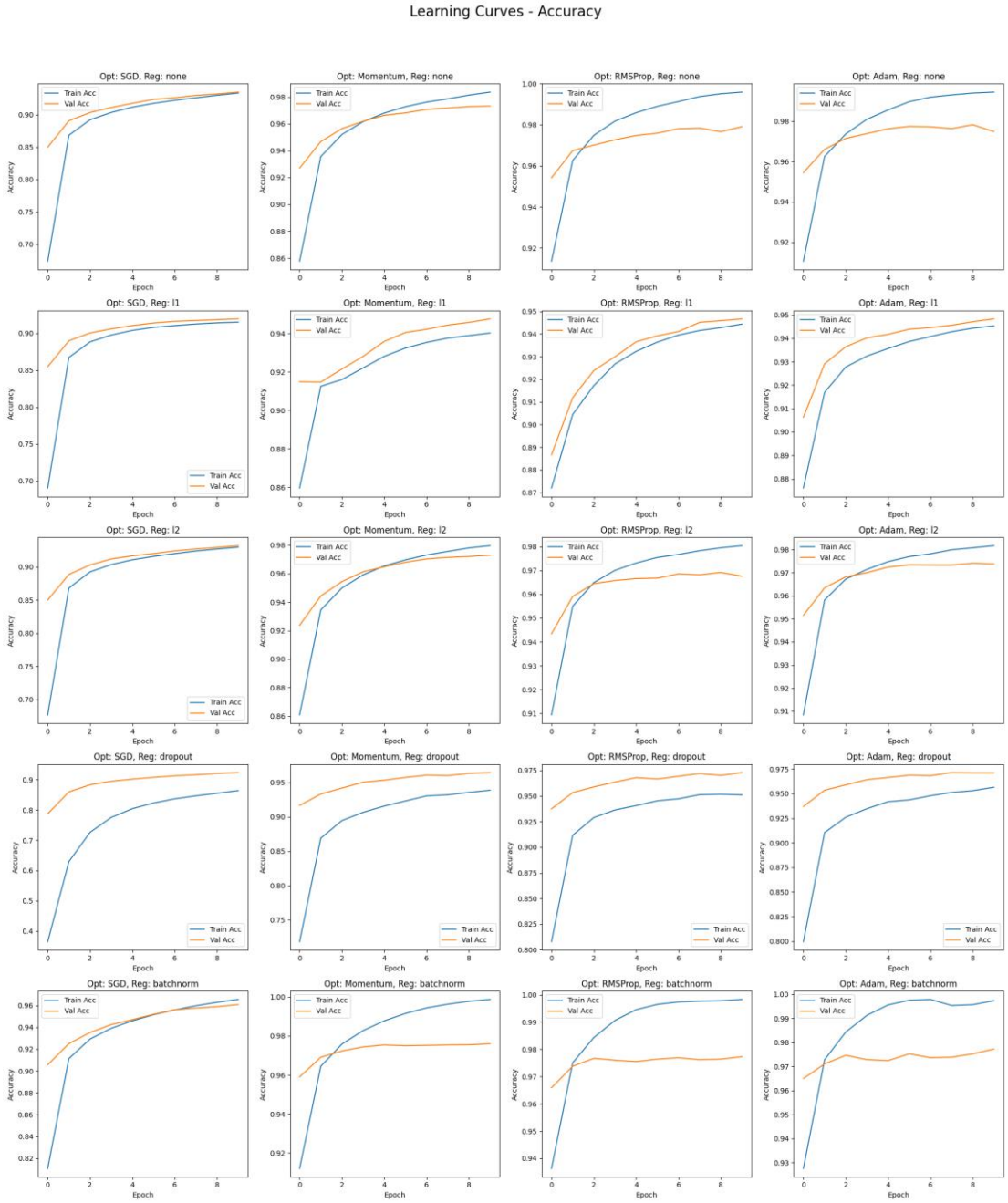
```python
def create_model(reg_type="none"):

    model = tf.keras.Sequential()
    if reg_type == "batchnorm":
        model.add(tf.keras.layers.Dense(128, use_bias=False, input_shape=(28*28,)))
        model.add(tf.keras.layers.BatchNormalization())
        model.add(tf.keras.layers.Activation('relu'))
    else:
        if reg_type == "l1":
            reg = tf.keras.regularizers.l1(0.001)
        elif reg_type == "l2":
            reg = tf.keras.regularizers.l2(0.001)
        else:
            reg = None

        model.add(tf.keras.layers.Dense(128, activation='relu', kernel_regularizer=reg, input_shape=(28*28,)))

    if reg_type == "dropout":
        model.add(tf.keras.layers.Dropout(0.5))

    if reg_type == "batchnorm":
        model.add(tf.keras.layers.Dense(128, use_bias=False))
        model.add(tf.keras.layers.BatchNormalization())
        model.add(tf.keras.layers.Activation('relu'))
    else:
        if reg_type in ["l1", "l2"]:
            reg = tf.keras.regularizers.l1(0.001) if reg_type=="l1" else tf.keras.regularizers.l2(0.001)
        else:
            reg = None
        model.add(tf.keras.layers.Dense(128, activation='relu', kernel_regularizer=reg))

    if reg_type == "dropout":
        model.add(tf.keras.layers.Dropout(0.5))

    model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

    return model
```

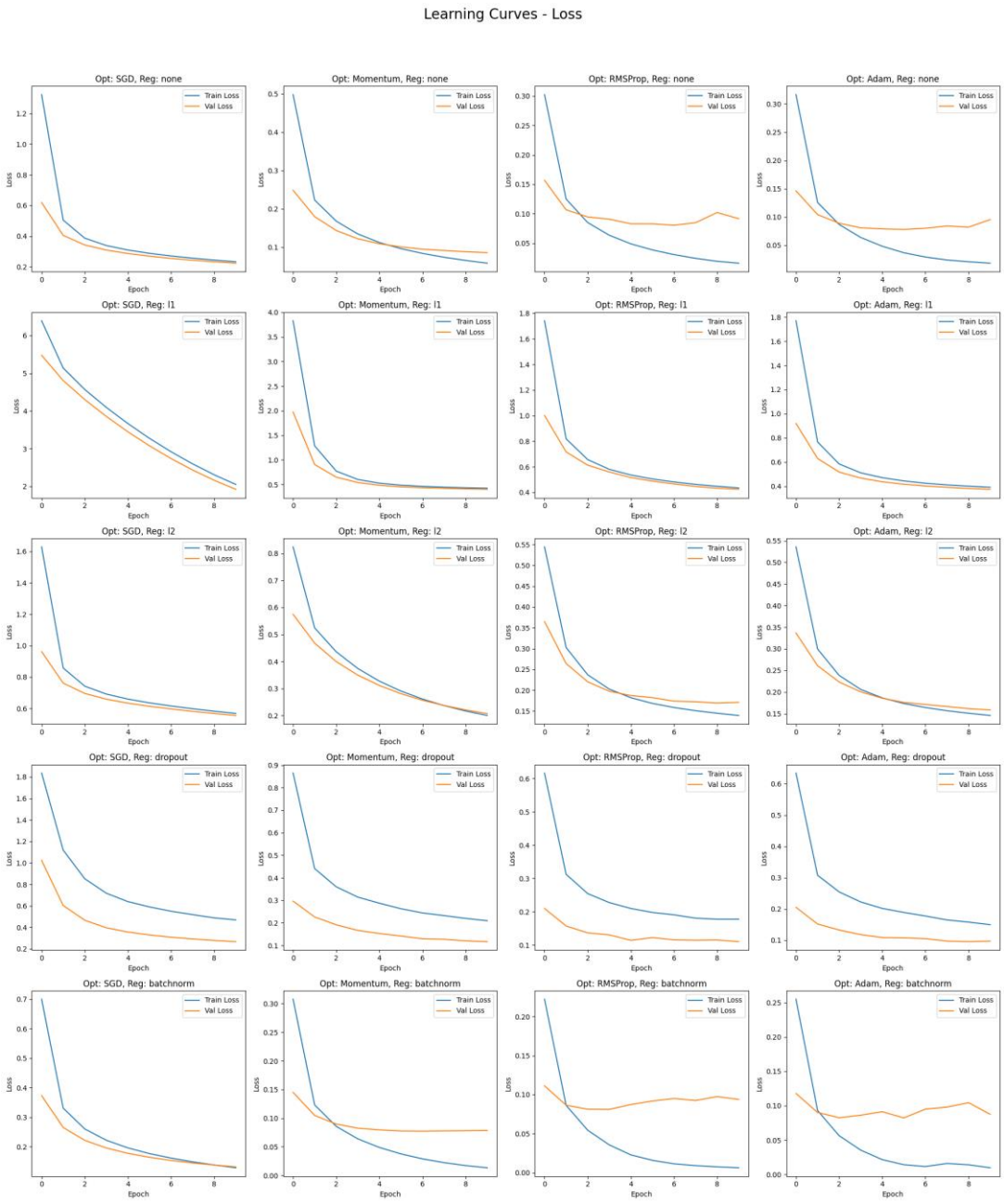
兩個隱藏層的簡單神經網路，透過 reg_type 來決定不同的正則化技術。

```python
optimizer_dict = {
    "SGD": lambda: tf.keras.optimizers.SGD(),
    "Momentum": lambda: tf.keras.optimizers.SGD(momentum=0.9),
    "RMSProp": lambda: tf.keras.optimizers.RMSprop(),
    "Adam": lambda: tf.keras.optimizers.Adam()
}

reg_types = ["none", "l1", "l2", "dropout", "batchnorm"]

history_dict = {}

epochs = 10
batch_size = 128


for opt_name in optimizer_dict:
    for reg in reg_types:
        print(f"Training with Optimizer: {opt_name}, Regularization: {reg}")
        model = create_model(reg_type=reg)
        optimizer = optimizer_dict[opt_name]()
        model.compile(optimizer=optimizer,
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])

        history = model.fit(x_train, y_train,
                            validation_data=(x_test, y_test),
                            epochs=epochs,
                            batch_size=batch_size,
                            verbose=0)
        history_dict[(opt_name, reg)] = history

n_rows = len(reg_types)
n_cols = len(optimizer_dict)
```

optimizer_dict 用來選擇不同的優化器，epochs = 10、batch_size = 128。

4. Result


Learning Curves - Accuracy
Opt: SGD, Reg: none
Opt: Momentum, Reg: none
Opt: RMSProp, Reg: none
Opt: Adam, Reg: none
Opt: SGD, Reg: l1
Opt: Momentum, Reg: l1
Opt: RMSProp, Reg: l1
Opt: Adam, Reg: l1
Opt: SGD, Reg: l2
Opt: Momentum, Reg: l2
Opt: RMSProp, Reg: l2
Opt: Adam, Reg: l2
Opt: SGD, Reg: dropout
Opt: Momentum, Reg: dropout
Opt: RMSProp, Reg: dropout
Opt: Adam, Reg: dropout
Opt: SGD, Reg: batchnorm
Opt: Momentum, Reg: batchnorm
Opt: RMSProp, Reg: batchnorm
Opt: Adam, Reg: batchnorm
Train Acc
Val Acc
Accuracy
Epoch



5. Conclusion

優化器的影響

- Adam 收斂最快，但 overfitting 風險較高。
- Momentum 與 RMSProp 速度適中。
- SGD 收斂最慢，但配合 BatchNorm 可以顯著提升效果。

正則化的影響

- L1, L2：能有效降低 overfitting，L2 效果好於 L1。
- Dropout：更能有效降低 overfitting，但影響較大，適合更深的網路。
- BatchNorm：幫助不明顯，可能適合更深的網路。

6. 心得

這次的實驗原本要跑非常多次要花很多的時間，但是我這次寫好程式碼一次就全部跑完，節省了很多時間跟精力。透過這次實驗讓我瞭解到了具體這些優化器跟正則化技術的效能差異，這樣在之後的其他實驗我就能更好知道如何組合。