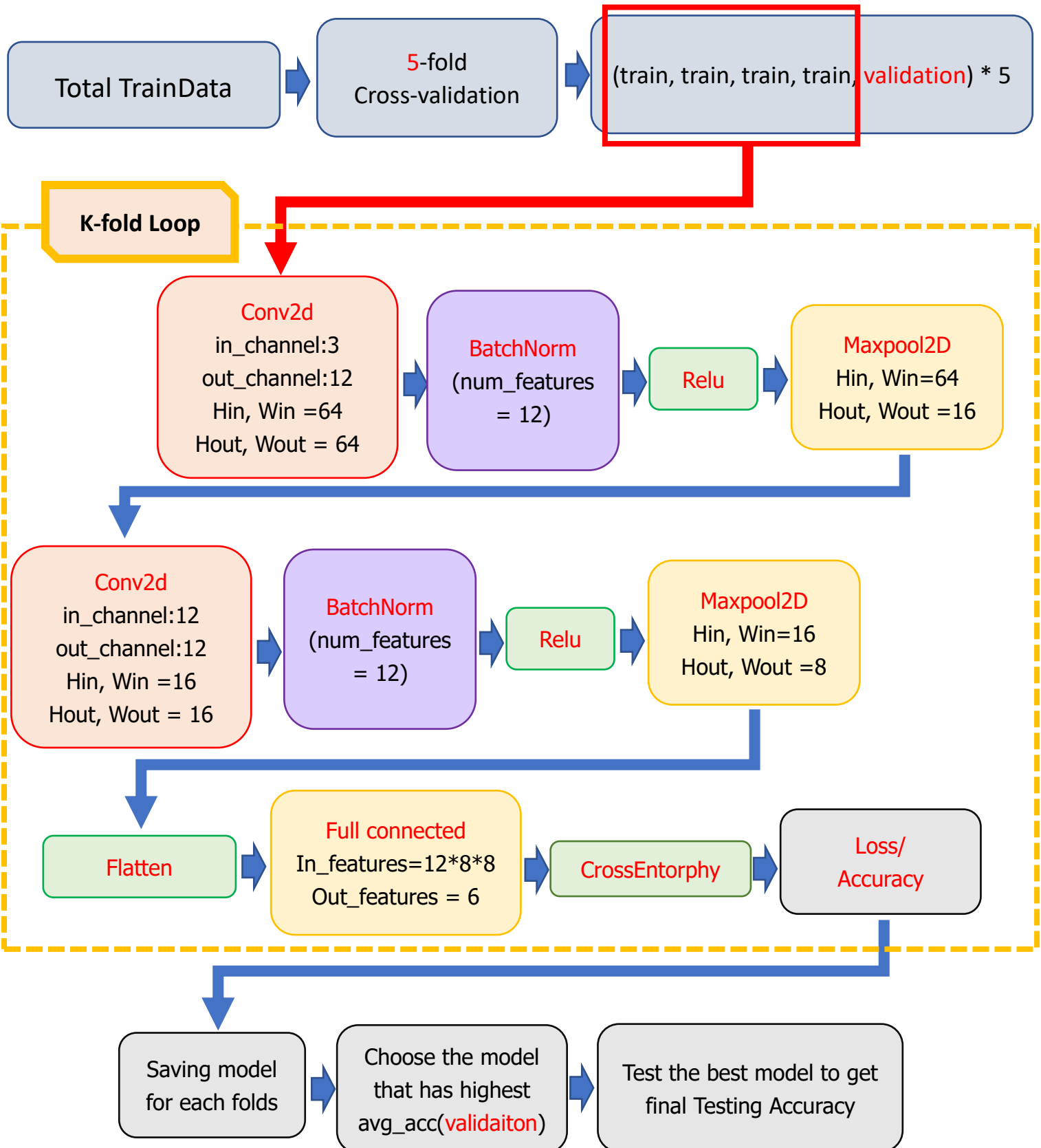


Problem1:

CNN Model architecture:



模型架構討論：

上圖為我的架構模型，是利用 pytorch 中兩層的 conv2D、BatchNorm2D、Relu、maxpool2D，來完成，之所以會使用 BatchNorm2D 的原因是為了將資料進行歸一化，避免資料中某些過大的數值而導致整個神經網路的不穩定。只使用兩層 conv2D 的原因是在觀察這次的資料特性(手勢判定)後，覺得主要圖片的特徵比較大，並沒有需要非常細微的特徵判斷，所以兩層的訓練效果應該就可以非常不錯，由最後的訓練以及測試成果也可以證明我的推論是正確的。

最後使用的 loss function 為 crossEntropy，較適合 Classification 的題目，以下為 hyperparameters 的調較(1)epochs=20 (2)batch_size=4 (3)learning_rate= 0.005。batch_size 選擇為 4，因為如果 batch_size 過大，本身電腦 GPU 的記憶體空間會不足。Epoch 基本上在 15 個 epochs 過後，training_acc 都已經達到 1.0 的效果，所以再多的 epochs 並不會更加地去優化模型，必須再去思考是否有其他方法來增進模型 testing 的準確性。如下圖為 5 個 models 中 best_model 的 testing accuracy(95%)。

```
Epoch : 7      train loss : tensor(0.0564, dtype=torch.float64, grad_fn=<DivBackward0>)      train acc : 0.9791666666666666
Epoch : 7      val loss : 0.10544260159148804      val acc : 0.9583333333333334
Epoch : 9      train loss : tensor(0.0282, dtype=torch.float64, grad_fn=<DivBackward0>)      train acc : 0.9942129629629629
Epoch : 9      val loss : 0.08525603734299718      val acc : 0.9583333333333334
Epoch : 11     train loss : tensor(0.0336, dtype=torch.float64, grad_fn=<DivBackward0>)      train acc : 0.9872685185185185
Epoch : 11     val loss : 0.2374507322469332      val acc : 0.9305555555555556
Epoch : 13     train loss : tensor(0.0400, dtype=torch.float64, grad_fn=<DivBackward0>)      train acc : 0.9895833333333334
Epoch : 13     val loss : 0.13302639538581898      val acc : 0.9351851851851852
Epoch : 15     train loss : tensor(0.0057, dtype=torch.float64, grad_fn=<DivBackward0>)      train acc : 1.0
Epoch : 15     val loss : 0.13591117012477477      val acc : 0.9629629629629629
Epoch : 17     train loss : tensor(0.0005, dtype=torch.float64, grad_fn=<DivBackward0>)      train acc : 1.0
Epoch : 17     val loss : 0.1480043917215203      val acc : 0.9629629629629629
Epoch : 19     train loss : tensor(0.0003, dtype=torch.float64, grad_fn=<DivBackward0>)      train acc : 1.0
Epoch : 19     val loss : 0.1514186345527614      val acc : 0.9629629629629629
the best model is the fold-5
Testing accuracy(%): 95.0
```

Problem2:

Fold-1_model:

time-27:12/09/22:21:27:56 Train loss. 0.0003 Train acc 1.0000 Val loss. 0.1872 Val acc 0.9537

Fold-2_model:

time-28:12/09/22:21:28:30 Train loss. 0.0005 Train acc 1.0000 Val loss. 0.1254 Val acc 0.9491

Fold-3_model:

time-29:12/09/22:21:29:03 Train loss. 0.0003 Train acc 1.0000 Val loss. 0.1915 Val acc 0.9583

Fold-4_model:

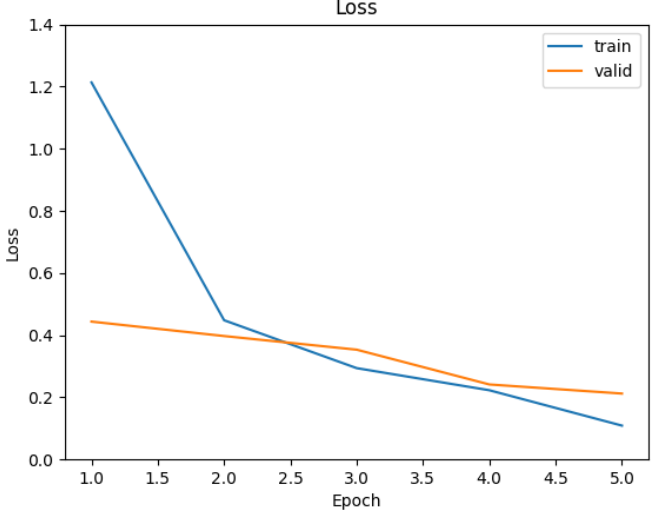
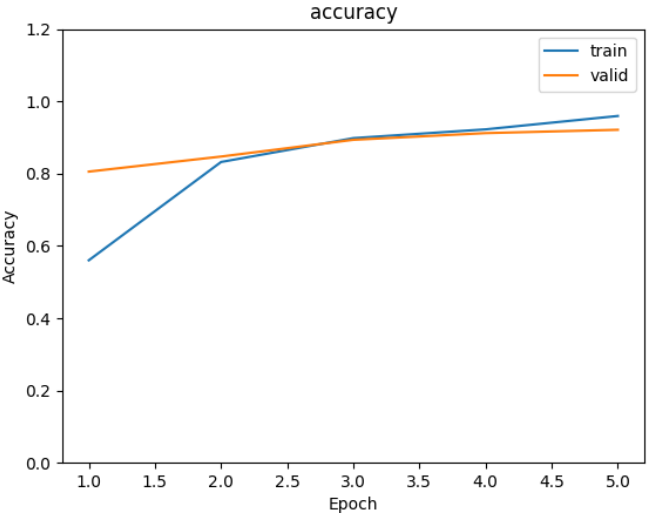
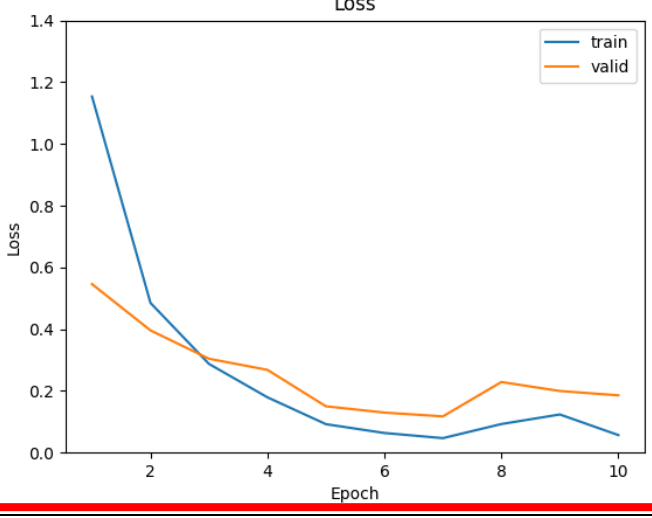
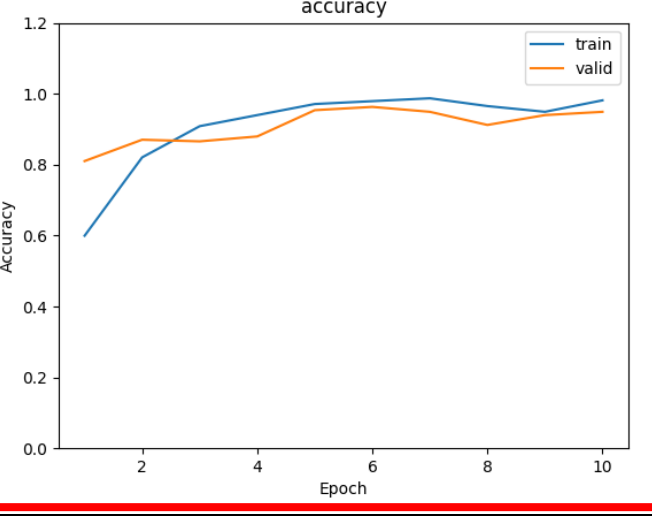
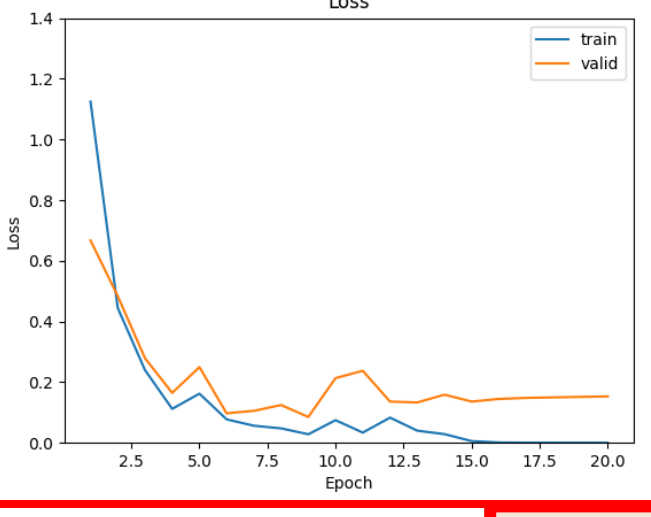
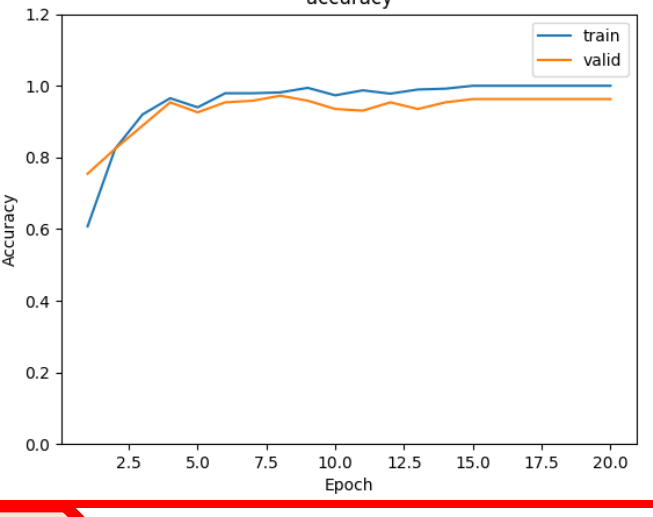
time-29:12/09/22:21:29:37 Train loss. 0.0004 Train acc 1.0000 Val loss. 0.3428 Val acc 0.9352

Fold-5_model(**best_model**):

time-30:12/09/22:21:30:11 Train loss. 0.0003 Train acc 1.0000 Val loss. 0.1530 Val acc 0.9630

Cross Validation 的優點是能夠完整的利用整份 data 來 training 我們的模型，因為在每個 fold 的訓練中，train data 以及 validation data 都不同，如此以來能夠訓練到整個 training data，並藉由不同的 validation data 來選出 best model，以此來獲得最好的 testing accuracy。最好的結果是每個 fold 都擁有相當的 accuracy，這可以指出 data 的完整性以及正確性。

Problem3: (validation)

epoch	Loss	Accuracy
5 Fold-1 (best-fold)	0.2121 	Val: 92.13%, testing = 93.33% 
10 Fold-2 (best-fold)	0.1855 	Val : 94.91% , testing = 94.17% 
20 Fold-5 (best-fold)	0.1530 	Val : 96.30%, testing = 95.0% 

Problem4:

將每個 fold 中 validation data 的 all epochs accuracy 去做平均值，接著從 5 個 folds 中的結果，選出最高 val_acc 的 model，已次來當作 best model，將 testing data 丟進 best model，得出最終的 Testing_accuracy，如下圖程式碼所展示。

```
total_val_acc_plot.append(val_acc_plot)    #val_acc_plot is all epochs accuracy in one fold

##### choose_model #####
total_acc_avg_per_fold = []
for i in range(len(total_val_acc_plot)):    #total folds
    avg_per_fold = np.sum(total_val_acc_plot[i]) / len(val_acc_plot) #calulate avg acc in total epochs
    total_acc_avg_per_fold.append(avg_per_fold)
max_idx = total_acc_avg_per_fold.index(max(total_acc_avg_per_fold)) #get the highest acc model in folds

best_model = My_Model()
best_model.load_state_dict(torch.load(f'./model-fold-{max_idx + 1}.pth'))    #load the best model
best_model.eval()
best_model = best_model.cuda()
best_model.double()
print("the best model is the fold-{}".format(max_idx + 1))
##### test_pred #####
test_pred = []
total_truth = []
with torch.no_grad():
    for data, label in test_loader:
        x_test, y_test = Variable(data), Variable(label)
        if torch.cuda.is_available():
            x_test = x_test.cuda()
            y_test = y_test.cuda()

        output_test = best_model(x_test)

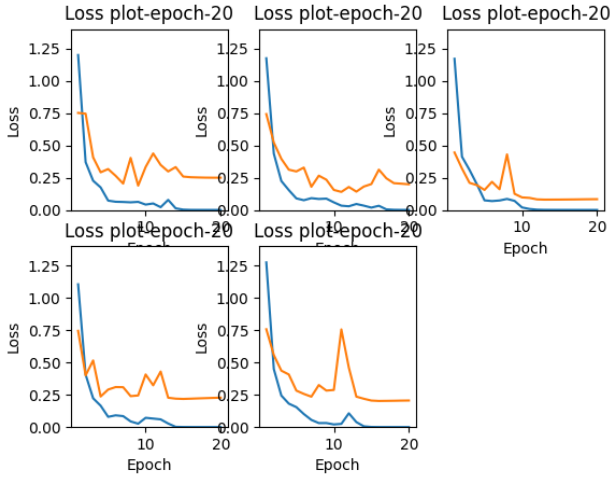
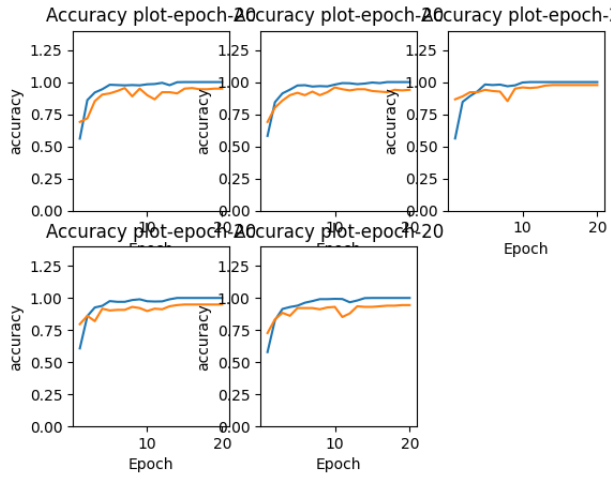
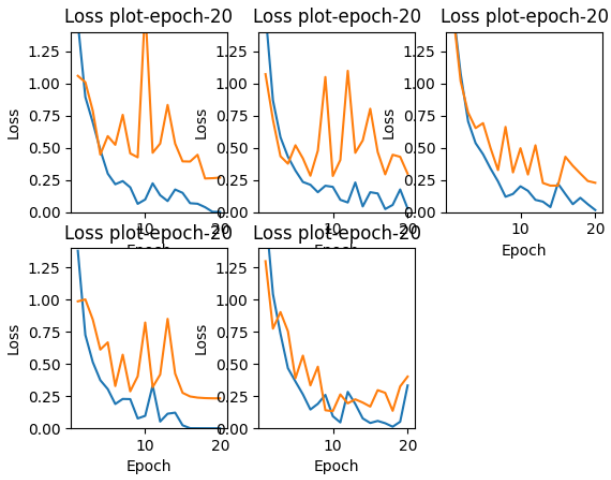
        # prediction for test set
        softmax = torch.exp(output_test).cpu()
        prob = list(softmax.detach().numpy())
        predictions = np.argmax(prob, axis=1)
        test_pred.extend(predictions)

        g_truth = y_test.cpu().detach().numpy()
        total_truth.extend(g_truth)

##### final prediction #####
prediction = test_pred# predict the first data is class 5, and second is class 0 ...
label = total_truth # first GT is class 5, and second is class 1 ...
Testing_accuracy = cal_accuracy(prediction, label)
```

Problem5: compare the pretrain_resnet with our own model

在這裡, 我使用 pretrained model – “resnet18” 來與我自己 scratch 出來的 model 作比較, 以下是相同條件下(epochs=20, batch_size=4), training 出來的結果(5 folds)。

Model Type	Loss	Accuracy
scratch	<p>0.1530 (best)</p> 	<p>Val : 96.30%, testing = 95.0% (best)</p> 
Resnet	<p>0.4022 (best)</p> 	<p>Val : 95.83%, testing = 93.33% (best)</p> 