AI training HW4

STuser19 賴昱凱

1. **Please explain the pros and cons of "regular convolution" and "depth-wise separable convolution" ?**

(M: input channel, N: output channel, D_k: kernel size, D_f: feature map size)

Regular convolution:

對 input 做一般的 convolution 計算得到 output，其計算量為

$$\boldsymbol{D_k \times D_k \times M \times (N \times D_f \times D_f)}$$

因為 convolution 計算方式就是將 kernel 大小的部分一對一相乘再相加得到 output 的一個像素，當 kernel 像素總數$D_k \times D_k$，且 input 有 M 個 channel 時，每計算一個 output 像素的計算量就為$D_k \times D_k \times M$。若我們希望 output 的大小為$D_f \times D_f$，有 N 個 output channel 所需總計算量就為 $D_k \times D_k \times M \times (N \times D_f \times D_f)$。

Pros:

一、各大小區域皆被完整掃描，可以適應不同大小特徵類型的圖片。

二、有更多參數且 channel 不獨立，可以更有效地辨識複雜的特徵。

三、因為模型可以捕捉各種細節特徵，因此應用廣泛且表現佳。

Cons:

一、計算量大，需要更多的算力、記憶體。

二、訓練及計算效率低。

三、對電力耗能要求高。

depth-wise separable convolution:

對 input 做 depth-wise separable convolution，其計算量為

$$\boldsymbol{(D_k \times D_k + N) \times M \times D_f \times D_f}$$

depth-wise separable convolution 分成 Depth-wise 以及 pointwise convolution 兩部分，Depth-wise 為各 input channel 獨立計算出自己的 output channel，計算量為$D_k \times D_k \times (M \times D_f \times D_f)$。Pointwise 則是利用 M 個 1x1 的 kernel 對 depth-wise 輸出的 M 個 channel 做 convolution，因此有 N 個 output channel 就有$M \times D_f \times D_f \times N$個計算量。兩部分計算量共 $(D_k \times D_k + N) \times M \times D_f \times D_f$。

Pros:

一、計算量及參數少，可以更快的訓練及計算。

二、層數、activation function 多，可能使其在某些情況表現更佳。

三、記憶體、算力等訓練成本要求減少。

Cons:

一、由於參數減少且各 channel 獨立，對於捕捉特徵的能力減弱。

二、要求細節的特徵上無法完全替代 regular convolution。

三、小模型使用此方法反而會因無法正確捕捉特稱而表現不佳。

2. **Please report the parameters of AlexNet by manual calculations. Show the actual "FLOPS / parameters" reported by code. Attached with Screenshot**

CustomAlexNet structure:

```
class CustomAlexNet(nn.Module):
    def __init__(self, num_classes):
        super(CustomAlexNet, self).__init__()
        # Create your AlexNet
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=11, padding=2, stride=4)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=196, kernel_size=5, padding=2)
        self.conv3 = nn.Conv2d(in_channels=196, out_channels=384, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(in_channels=384, out_channels=256, kernel_size=3, padding=1)
        self.conv5 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(in_features=256*6*6, out_features=4096)
        self.fc2 = nn.Linear(in_features=4096, out_features=1024)
        self.fc3 = nn.Linear(in_features=1024, out_features=num_classes)

    def forward(self, x):
        # Connect your Model
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, kernel_size=3, stride=2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, kernel_size=3, stride=2)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = F.relu(self.conv5(x))
        x = F.max_pool2d(x, kernel_size=3, stride=2)
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, p=0.5)
        x = F.relu(self.fc2(x))
        x = F.dropout(x, p=0.5)
        x = self.fc3(x)
        return x
```

Total number of parameters:

Convolution layers: input channel × (kernel size)^2 × output channel + output channel

$$conv1: 3 \times 11^2 \times 64 + 64 = 23296$$

$$conv2: 64 \times 5^2 \times 196 + 196 = 313796$$

$$conv3: 196 \times 3^2 \times 384 + 384 = 677760$$

$$conv4: 384 \times 3^2 \times 256 + 256 = 884992$$

$$conv5: 256 \times 3^2 \times 256 + 256 = 590080$$

Fully connection layers: input feature × output feature + output feature

$$fc1: 256 \times 6 \times 6 \times 4096 + 4096 = 37752832$$

$$fc2: 4096 \times 1024 + 1024 = 4195328$$

$$fc3: 1024 \times 1000 + 1000 = 1025000$$

$$\boldsymbol{Total\ parameters: 23296 + 313796 + 677760 + 884992 + 590080 + 37752832 + 4195328 + 1025000 = 45463084}$$

Screenshot:

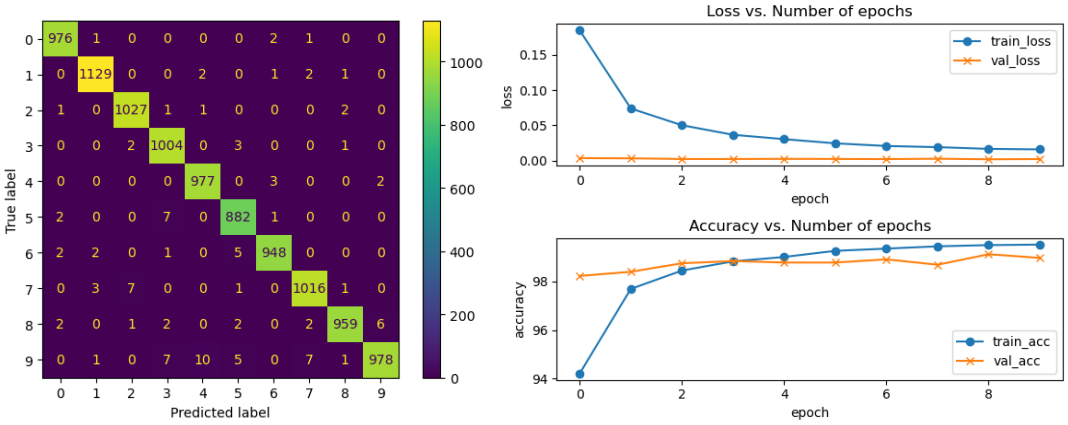
```
FLOPS: 705535424.0
Parameters: 45463084.0
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 55, 55]          23,296
            Conv2d-2         [-1, 196, 27, 27]         313,796
            Conv2d-3         [-1, 384, 13, 13]         677,760
            Conv2d-4         [-1, 256, 13, 13]         884,992
            Conv2d-5         [-1, 256, 13, 13]         590,080
            Linear-6                [-1, 4096]      37,752,832
            Linear-7                [-1, 1024]       4,195,328
            Linear-8                [-1, 1000]       1,025,000
================================================================
Total params: 45,463,084
Trainable params: 45,463,084
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 3.77
Params size (MB): 173.43
Estimated Total Size (MB): 177.77
----------------------------------------------------------------
```

3. With hw4.py, train a CNN-based model without pre-trained weights.

首先因為作業要求 without pre-trained weights，因此我更改原先模型 ResNet18 參數 pretrained=False 並將 param.requires_grad = False 拿掉，就跑出 accuracy = 98.96%，如下圖:

```
Test Set: Average loss: 0.0022, Accuracy: 98.96%

Training Finish
Accuracy: 0.9896
Precision: 0.9895178357582639
Recall: 0.989530859201621
F1 score: 0.9895019235691171
```



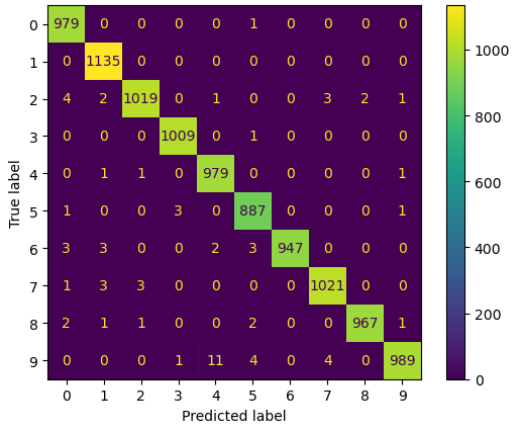
我推測是因為 MNIST 的手寫數字 input 圖檔只有$28\ pixels \times 28\ pixels$，且皆為灰階圖片(channel = 1)，結構簡單且辨識結果可能性只有 10 種，因此隨意的 CNN 模型就可以有很好的表現。

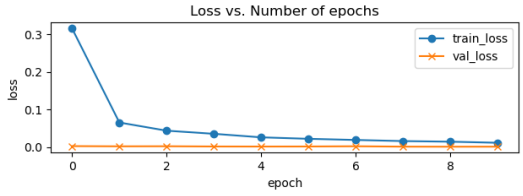
由於 ResNet18 只有一個 fully connection layer，因此我參考了 2011 年 Ciresan 等人在 ICDAR 上發表的 CNN-based model (其在圖像辨識尤其是 MNIST 手寫數字辨識上有極優的表現)，並將其改成 ResNet 的形式拿來與原模型比較，更改後結構為一層 convolution layer、3 個 residual blocks 及 3 層 fully connection layer，共 10 層神經網路，並在第一層 convolution layer 後及 fully connection layer 前使用 Maxpooling 讓計算量減少以簡短訓練時間(約 30 分鐘內降至約 7 分鐘)，結果如下:

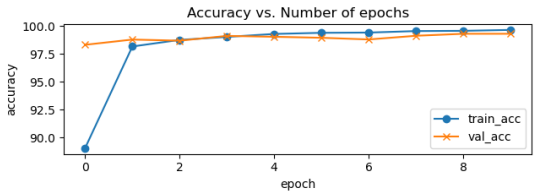
CiresanNet 之結構及 accuracy: 99.32%

```python
class CiresanNet(nn.Module):
    def __init__(self, num_classes=10):
        super(CiresanNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 48, kernel_size=5, stride=1, padding=2)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(48, 128, kernel_size=5, stride=1, padding=2)
        self.conv3 = nn.Conv2d(128, 192, kernel_size=5, stride=1, padding=2)
        self.conv4 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.conv5 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.conv6 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.conv7 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.fc1 = nn.Linear(192 * 3 * 3, 512)
        self.fc2 = nn.Linear(512, 512)
        self.fc3 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = F.relu(self.conv5(x))
        x = F.relu(self.conv6(x))
        x = F.relu(self.conv7(x))
        x = self.pool(x)
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
Test Set: Average loss: 0.0018, Accuracy: 99.32%

Training Finish
Accuracy: 0.9932
Precision: 0.993207413117578
Recall: 0.993143216553445
F1_score: 0.9931546440350664
```



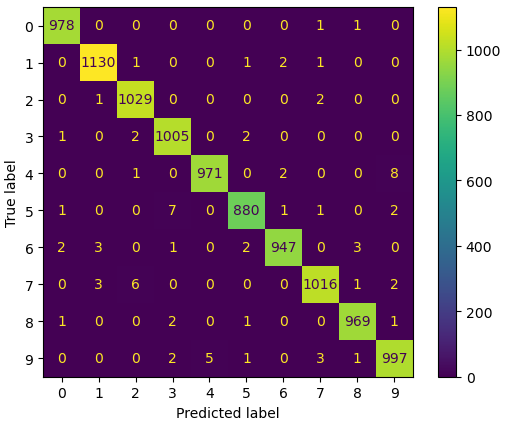


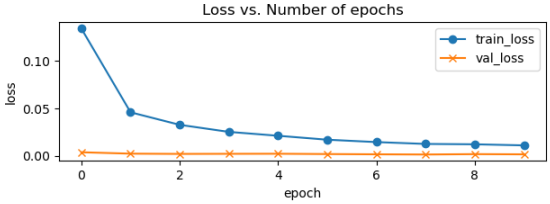


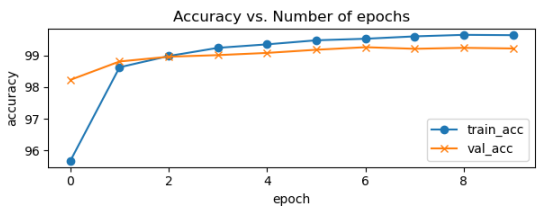
更改後 CiresanNet 之結構及 accuracy: 99.22%

```python
class CiresanNet(nn.Module):
    def __init__(self, num_classes=10):
        super(CiresanNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 48, kernel_size=5, stride=1, padding=2)
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(48, 128, kernel_size=5, stride=1, padding=2)
        self.conv3_1x1 = nn.Conv2d(128, 192, kernel_size=1, stride=1, padding=0)
        self.conv3 = nn.Conv2d(128, 192, kernel_size=5, stride=1, padding=2)
        self.conv4 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.conv5 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.conv6 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.conv7 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.fc1 = nn.Linear(192 * 7 * 7, 512)
        self.fc2 = nn.Linear(512, 512)
        self.fc3 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.maxpool(x)
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x) + self.conv3_1x1(x))
        x = F.relu(self.conv4(x))
        x = F.relu(self.conv5(x) + x)
        x = F.relu(self.conv6(x))
        x = F.relu(self.conv7(x) + x)
        x = self.maxpool(x)
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
Test Set: Average loss: 0.0016, Accuracy: 99.22%

Training Finish
Accuracy: 0.9922
Precision: 0.9922228473620104
Recall: 0.9920860041405895
F1_score: 0.9921486799542121
```



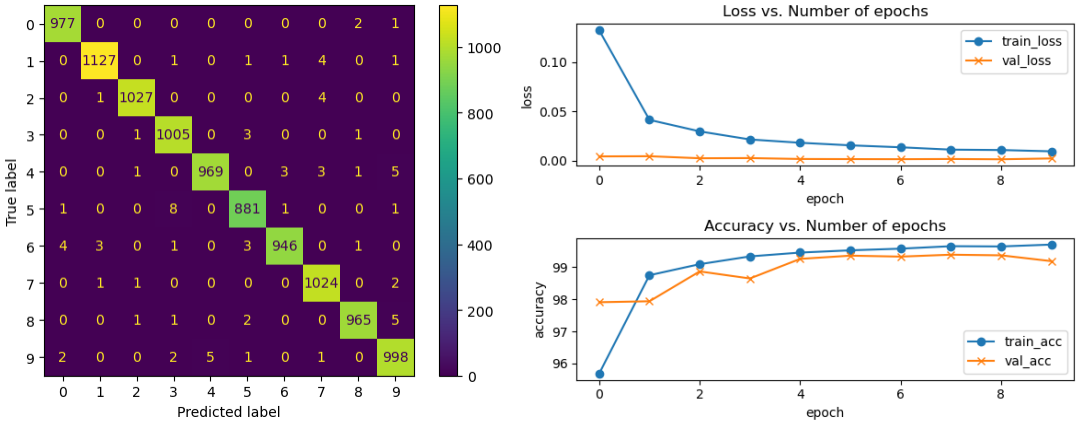




將 CiresanNet 改成 Residual blocks 表現稍差於 convolution + Maxpooling，但與同為 Residual blocks 架構的 ResNet18 比較起來表現明顯更佳，應該是 fully connection layer 由 1 層增為 3 層的表現彌補了 Residual blocks 的減少，以及 channel 數量變多的原因。因此我將 fully connection layer 以 2、4 層分別訓練得到以下結果: (原結構為 3 層，accuracy: 99.22%)

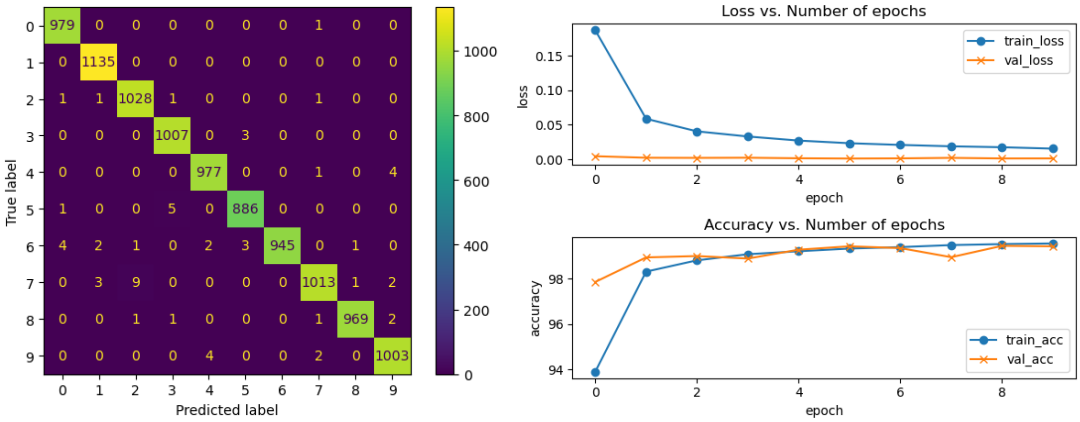
2 層 fully connection layers: 99.19%

```
Test Set: Average loss: 0.0021, Accuracy: 99.19%

Training Finish
Accuracy: 0.9919
Precision: 0.9918659760224587
Recall: 0.9917965457223724
F1_score: 0.9918237898754763
```



4 層 fully connection layers: 99.42%

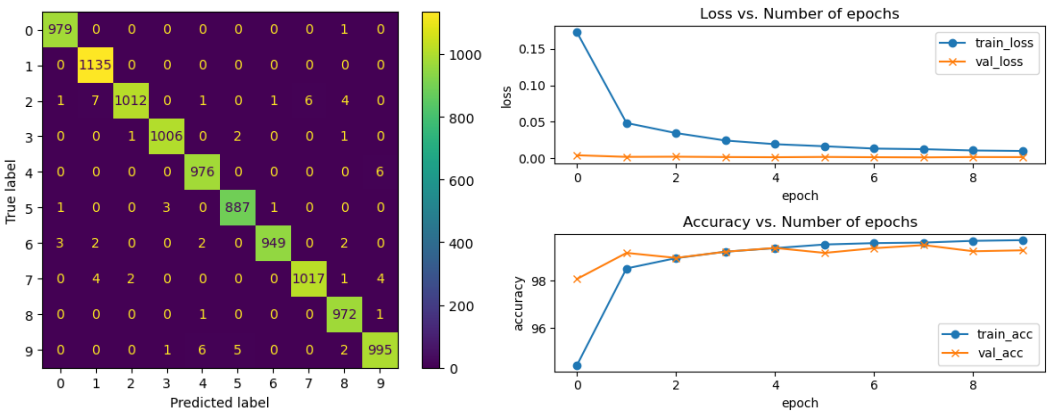
```
Test Set: Average loss: 0.0013, Accuracy: 99.42%

Training Finish
Accuracy: 0.9942
Precision: 0.9942468591562037
Recall: 0.9941073889772056
F1_score: 0.9941673544843912
```



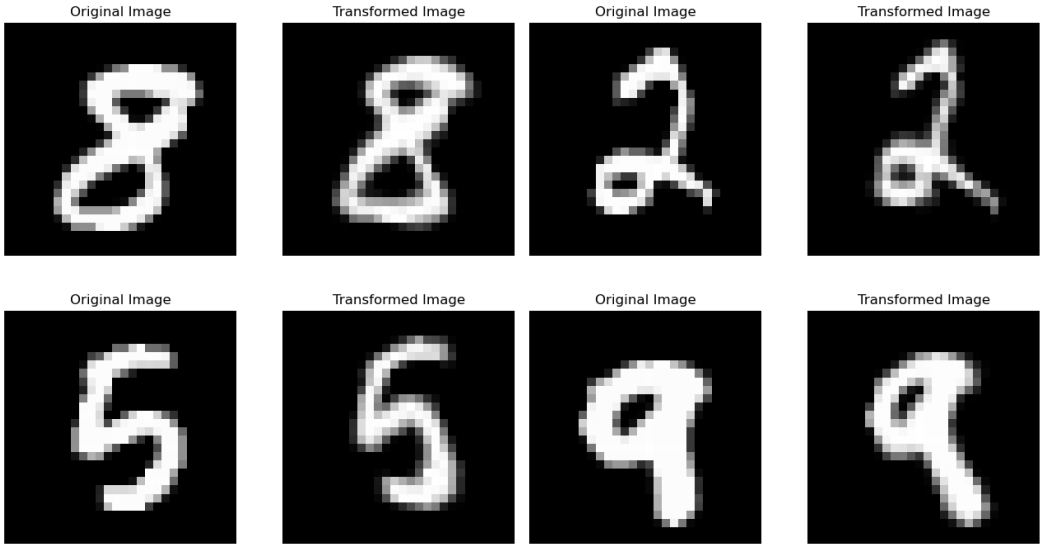
結果表示在 4 層以下 fully connection layer 的 accuracy 有正相關，但我將其增加至 5 層後便出現 98.98%的 accuracy，應該是參數過多導致 overfitting，因此 2~4 層是比較恰當的層數。

提到 overfitting，我就想到是否減少 convolution layer 層數對於簡單的圖形辨識可以有更佳的表現?因此我將 CiresanNet 減少至 4 層 convolution layers 及 3 層 fully connection layers，得到結果表現與原先的 10 層網路差不多:

```
Test Set: Average loss: 0.0015, Accuracy: 99.28%

Training Finish
Accuracy: 0.9928
Precision: 0.9928613851175913
Recall: 0.9927900516931076
F1_score: 0.9928104803504644
```

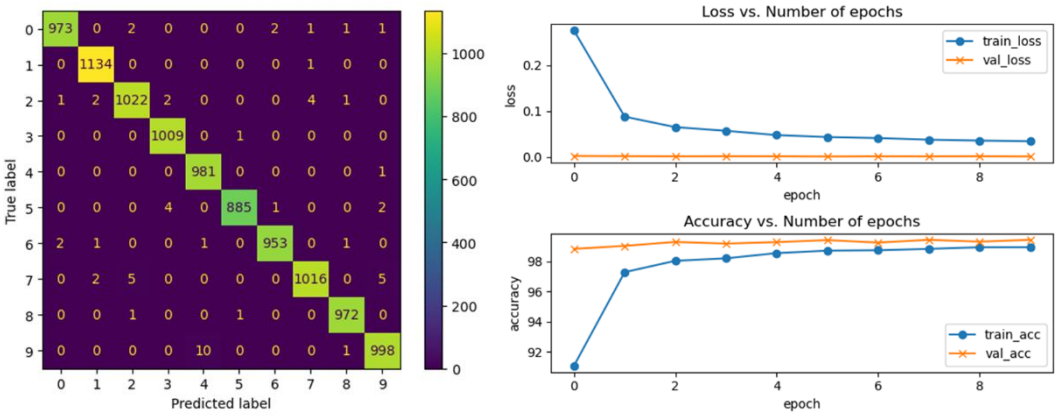


於是我參考許多研究都有使用的 elastic distortion 將輸入資料做前處理，會隨機扭曲圖片及平滑邊緣，如下圖:



並使用剛才使用的 7 層 CNN 模型，結果如下:

```
Test Set: Average loss: 0.0011, Accuracy: 99.43%

Training Finish
Accuracy: 0.9943
Precision: 0.9943469877039435
Recall: 0.9942582570952279
F1_score: 0.9942967000670555
```



Summary:

1. Fully connection layer 的層數適合 2~4 層，相較於 1 層可有效提升 accuracy。
2. Residual blocks 雖然可以避免梯度消失問題，但在此任務中效果不佳，我推測是因為任務過於簡單，本就不易遇到梯度消失，用 Residual blocks 反而造成 loss 更難抵達 minimum。
3. 簡單的任務若使用複雜的模型會很容易導致 overfitting，層數少的模型反而表現更佳。
4. Maxpooling 可以大幅減少訓練的計算及時間成本，且不會對於表現有太大影響。
5. Elastic distortion 對於 MNIST handwriting numbers recognition 有正向影響。

Reference:

https://yann.lecun.com/exdb/mnist/

https://blog.csdn.net/weixin_43112053/article/details/127374541

https://chih-sheng-huang821.medium.com/%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-mobilenet-depthwise-separable-convolution-f1ed016b3467

https://blog.csdn.net/u011622208/article/details/112566676