HW4

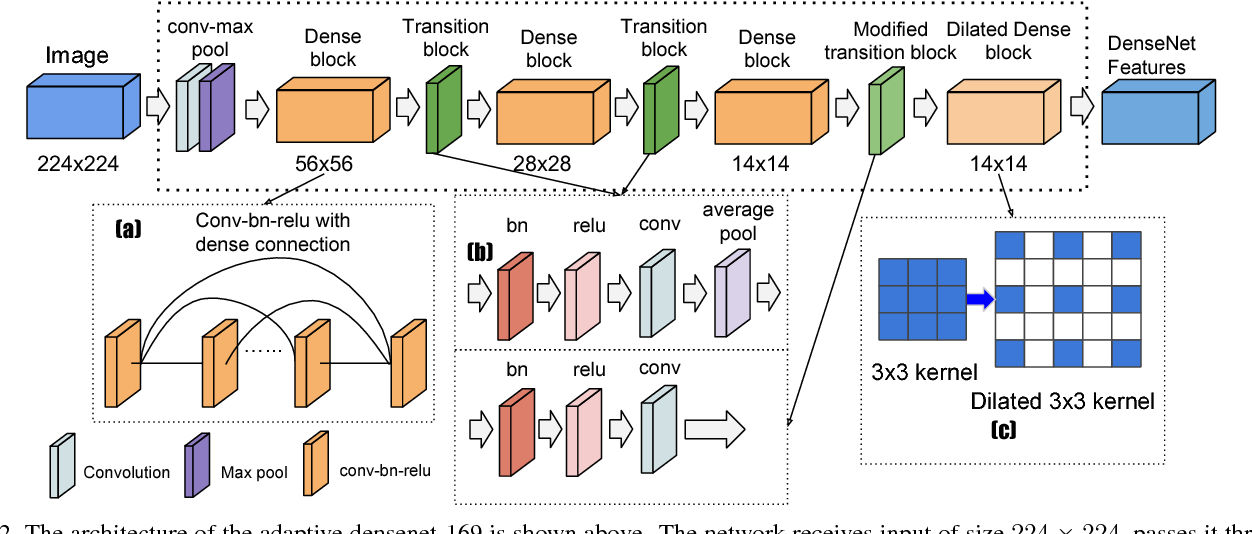
**使用densenet169模型實作**

CNN越深則效果越好，但是，會面臨梯度彌散的問題，經過層數越多，則前面的信息就會漸漸減弱和消散。為了確保網絡中最大的信息流通，讓每層都與改層之前的所有層都相連，即每層的輸入，是前面所有層的輸出的CONCAT。

（1）Highway Network,Residual Network通過前後兩層的殘差連結使信息儘量不丟失

（2）Stochastic depth通過隨機drop掉Resnet的一些層來縮短模型

（3）FractalNets通過重複組合一些平行的層序列來保證深度的同時減輕這個問題。



1. 模型架構定義
2. 多少層convolution：in\_channel = 3，out\_channel = 64，kernel\_size = 7，stride = 2，padding = 3，輸出（122 \* 122 \* 64），batch normalization計算，輸入與輸出維度不變（122 \* 122 \* 64），激活函數relu計算，輸入與輸出維度不變（122 \* 122 \* 64）。
3. 池化層poollig計算：kenel\_size = 3，stride = 2，padding = 1，輸出（56 \* 56 \* 64）。
4. Residual：1.Batch Normalization， 輸出（56\*56\*64，ReLU ，輸出（56\*56\*64），Bottleneck，是可選的，為了減少 feature-maps的數量，過程如下3步-1x1 Convolution, kernel\_size=1, channel = 4k, 則輸出為（56\*56\*128），-Batch Normalization（56\*56\*128），-ReLU（56\*56\*128），4.Convolution, kernel\_size=3, channel = k （56\*56\*32），5.Dropout,可選的,用於防止過擬合（56\*56\*32）。
5. dense connection：Dense Block有L層dense layer組成layer 0:輸入（56\*56\*64）->輸出（56\*56\*32），layer 1:輸入（56\*56\*(32\*1))->輸出（56\*56\*32），layer 2:輸入（56\*56\*(32\*2))->輸出（56\*56\*32），layer L:輸入（56\*56\*(32\*L))->輸出（56\*56\*32），注意，L層dense layer的輸出都是不變的，而每層的輸入channel數是增加的，因為如上所述，每層的輸入是前面所有層的拼接。Transition Block是在兩個Dense Block之間的，由一個卷積+一個pooling組成（下面的數據維度以第一個transition block為例）：

輸入：Dense Block的輸出（56\*56\*32）

1.Batch Normalization 輸出（56\*56\*32）

2.ReLU 輸出（56\*56\*32）

3.1x1 Convolution，kernel\_size=1，此處可以根據預先設定的壓縮係數（0-1之間）來壓縮原來的channel數,以減小參數，輸出（56\*56\*(32\*compression)）

4.2x2 Average Pooling 輸出（28\*28\*(32\*compression)）

在ImageNet的數據集上，構建的densenet是由4個Dense Block，和3個Transition構成，按照上文講述的過程，數據流的演變過程應該是：

Dense Block1：輸入（56\*56\*64）,輸出（56\*56\*32）

Transition1：輸入（56\*56\*32）,輸出（28\*28\*32）

Dense Block2：輸入（28\*28\*32）,輸出（28\*28\*32）

Transition2：輸入（28\*28\*32）,輸出（14\*14\*32）

Dense Block3：輸入（14\*14\*32）,輸出（14\*14\*32）

Transition3：輸入（14\*14\*32）,輸出（7\*7\*32）

最後一步是ClassificationBlock，這一步將原來3維的數據拉平成一維，再接上全連接層，以準備做softmax。計算過程如下：

輸入：Transition3的輸出（7\*7\*32）

1.Batch Normalization, 輸出（7\*7\*32）

2.ReLU, 輸出（7\*7\*32）

3.poolling, kernel\_size=7, stride=1,輸出（1\*1\*32）

4.flatten,將（1\*1\*32）鋪平成（1\*32）

5.Linear全連接，輸出（1\*classes\_num) classes\_num為待分類的數目

1. 參數設定 (所有參數結開放調整)

pretrain = true 或pretrain = false參數都一樣

data\_loader = DataLoader(dataset=train\_set, batch\_size=16, shuffle=True, num\_workers=1)

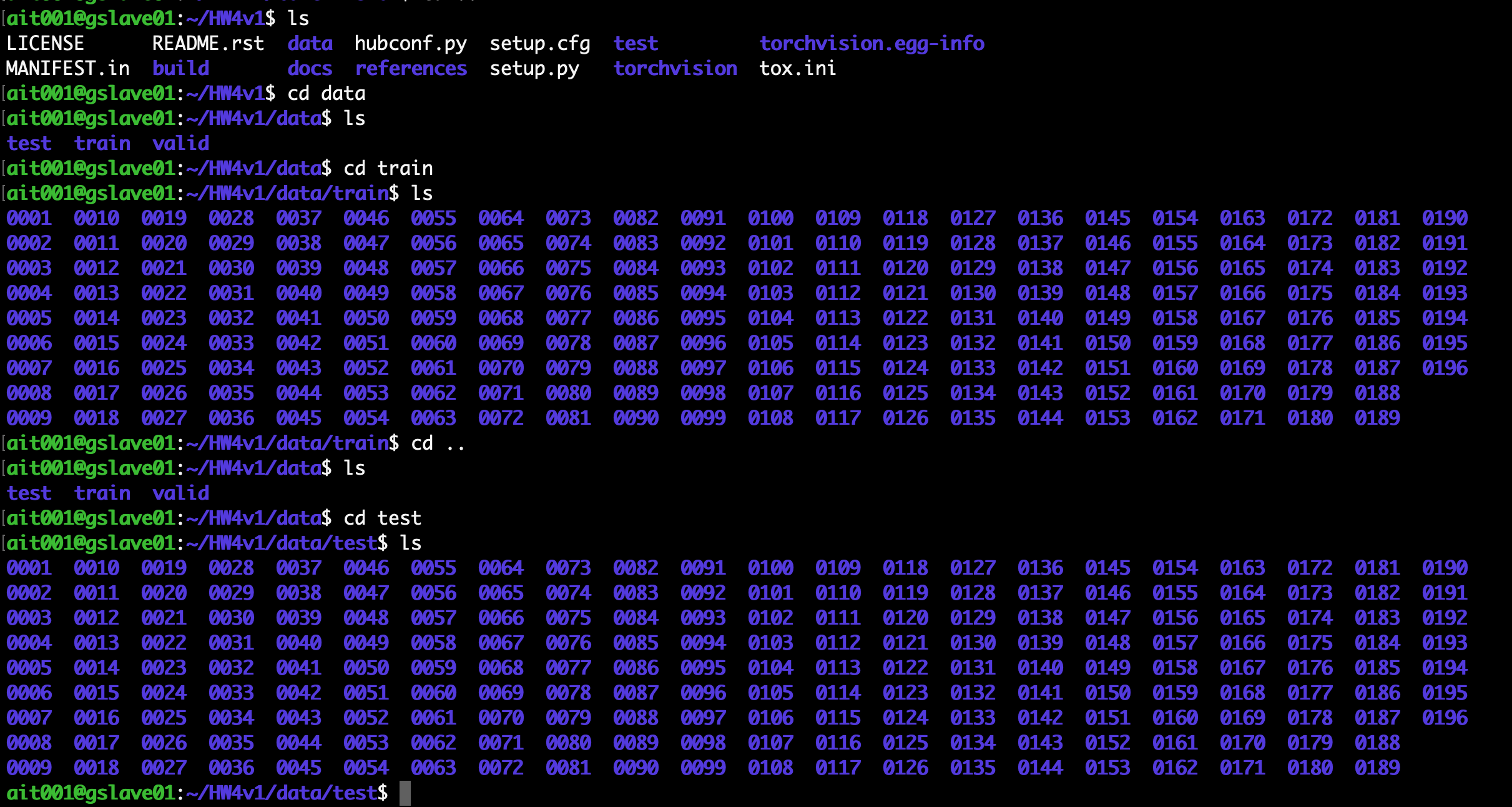
best\_acc = 0.0

num\_epochs = 20

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(params=model.parameters(), lr=0.001, momentum=0.9)

1. 資料處理 (data augmentation, background subtraction,...)

與HW2相同，使用mat座標將train 照片corp出來並分為196個class，test資料也區分為196個class。

完成模型訓練與測試

Pretrained= False

Epoch: 1/20

-----------

Training loss: 5.2630 accuracy: 0.0091

Epoch: 2/20

-----------

Training loss: 5.1398 accuracy: 0.0174

Epoch: 3/20

-----------

Training loss: 5.0516 accuracy: 0.0222

Epoch: 4/20

-----------

Training loss: 4.9673 accuracy: 0.0322

Epoch: 5/20

-----------

Training loss: 4.8917 accuracy: 0.0404

Epoch: 6/20

-----------

Training loss: 4.8085 accuracy: 0.0462

Epoch: 7/20

-----------

Training loss: 4.7186 accuracy: 0.0538

Epoch: 8/20

-----------

Training loss: 4.6295 accuracy: 0.0621

Epoch: 9/20

-----------

Training loss: 4.5198 accuracy: 0.0745

Epoch: 10/20

------------

Training loss: 4.4047 accuracy: 0.0872

Epoch: 11/20

------------

Training loss: 4.2839 accuracy: 0.0932

Epoch: 12/20

------------

Training loss: 4.1497 accuracy: 0.1130

Epoch: 13/20

------------

Training loss: 4.0179 accuracy: 0.1235

Epoch: 14/20

------------

Training loss: 3.8824 accuracy: 0.1386

Epoch: 15/20

------------

Training loss: 3.7585 accuracy: 0.1545

Epoch: 16/20

------------

Training loss: 3.6198 accuracy: 0.1785

Epoch: 17/20

------------

Training loss: 3.4867 accuracy: 0.1880

Epoch: 18/20

------------

Training loss: 3.3532 accuracy: 0.2150

Epoch: 19/20

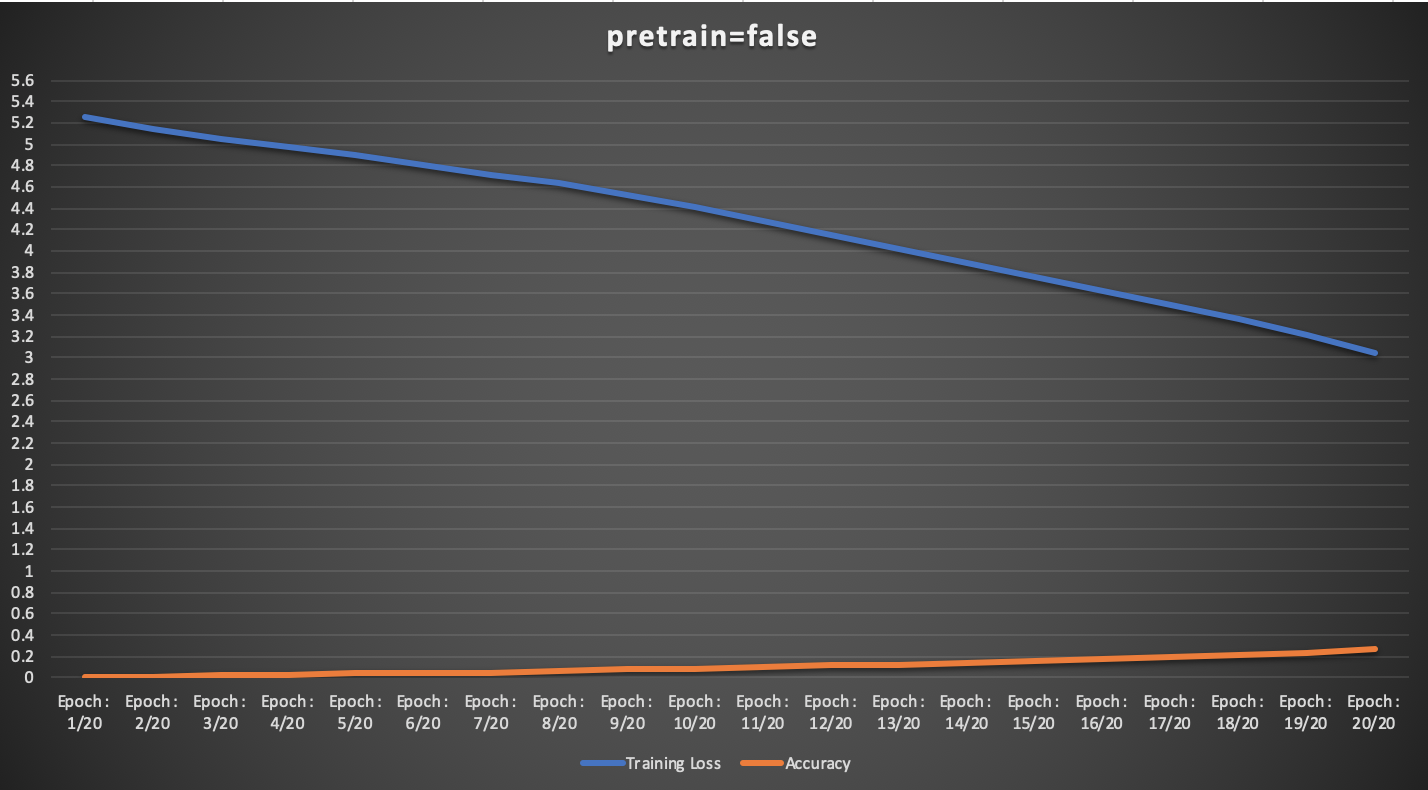
------------

Training loss: 3.2106 accuracy: 0.2377

Epoch: 20/20

------------

Training loss: 3.0515 accuracy: 0.2706



Accuracy on the ALL test images: 14 %

Accuracy of 0150 : 47 %

Accuracy of 0120 : 11 %

Accuracy of 0012 : 30 %

Accuracy of 0067 : 2 %

Accuracy of 0178 : 0 %

Accuracy of 0157 : 52 %

Accuracy of 0193 : 0 %

Accuracy of 0078 : 8 %

Accuracy of 0096 : 2 %

Accuracy of 0177 : 20 %

Accuracy of 0059 : 2 %

Accuracy of 0033 : 2 %

Accuracy of 0004 : 9 %

Accuracy of 0114 : 6 %

Accuracy of 0186 : 0 %

Accuracy of 0031 : 9 %

Accuracy of 0145 : 4 %

Accuracy of 0087 : 0 %

Accuracy of 0105 : 2 %

Accuracy of 0181 : 0 %

Accuracy of 0126 : 9 %

Accuracy of 0160 : 40 %

Accuracy of 0144 : 0 %

Accuracy of 0046 : 16 %

Accuracy of 0085 : 18 %

Accuracy of 0196 : 17 %

Accuracy of 0027 : 0 %

Accuracy of 0143 : 0 %

Accuracy of 0073 : 15 %

Accuracy of 0130 : 2 %

…………. 以下省略完整檔nopretraintest

Pretrained= True

Epoch: 1/20

-----------

Training loss: 4.5581 accuracy: 0.1029

Epoch: 2/20

-----------

Training loss: 2.5550 accuracy: 0.5119

Epoch: 3/20

-----------

Training loss: 1.2771 accuracy: 0.7834

Epoch: 4/20

-----------

Training loss: 0.6764 accuracy: 0.9012

Epoch: 5/20

-----------

Training loss: 0.3699 accuracy: 0.9548

Epoch: 6/20

-----------

Training loss: 0.2070 accuracy: 0.9827

Epoch: 7/20

-----------

Training loss: 0.1318 accuracy: 0.9907

Epoch: 8/20

-----------

Training loss: 0.0909 accuracy: 0.9955

Epoch: 9/20

-----------

Training loss: 0.0685 accuracy: 0.9964

Epoch: 10/20

------------

Training loss: 0.0549 accuracy: 0.9978

Epoch: 11/20

------------

Training loss: 0.0432 accuracy: 0.9975

Epoch: 12/20

------------

Training loss: 0.0372 accuracy: 0.9977

Epoch: 13/20

------------

Training loss: 0.0305 accuracy: 0.9979

Epoch: 14/20

------------

Training loss: 0.0283 accuracy: 0.9977

Epoch: 15/20

------------

Training loss: 0.0242 accuracy: 0.9983

Epoch: 16/20

------------

Training loss: 0.0227 accuracy: 0.9980

Epoch: 17/20

------------

Training loss: 0.0204 accuracy: 0.9979

Epoch: 18/20

------------

Training loss: 0.0194 accuracy: 0.9979

Epoch: 19/20

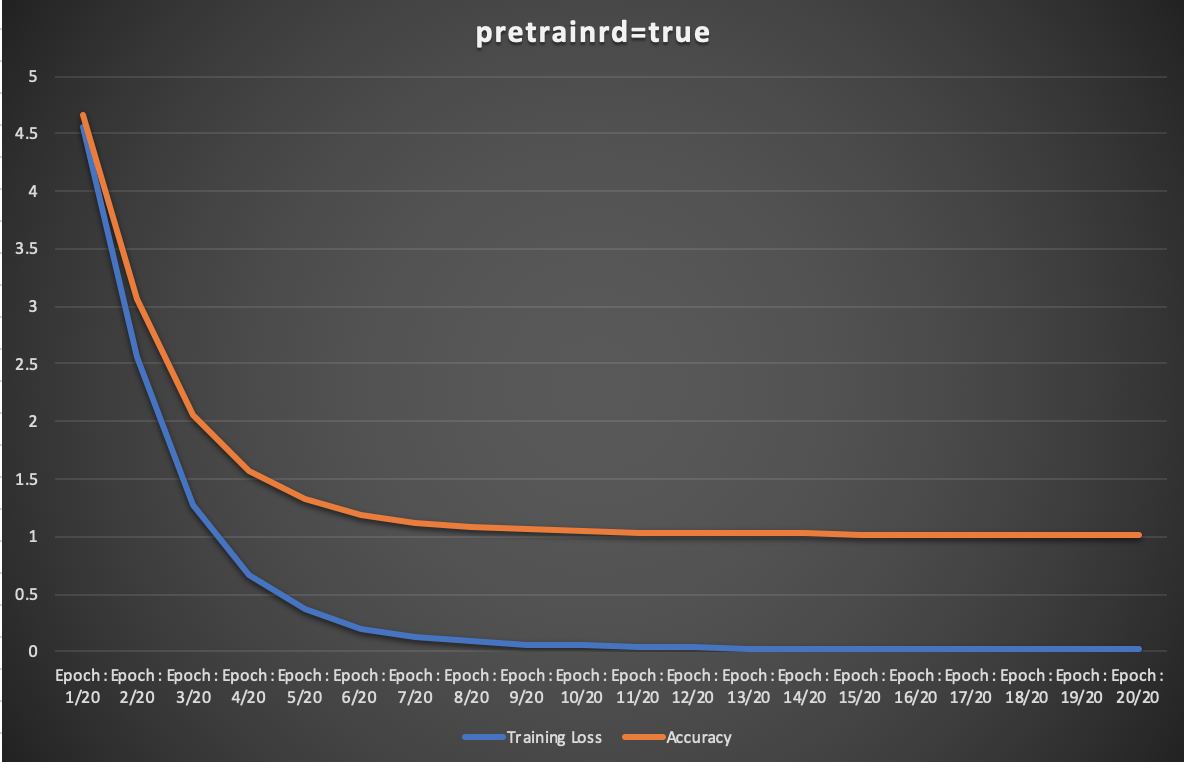
------------

Training loss: 0.0177 accuracy: 0.9983

Epoch: 20/20

------------

Training loss: 0.0168 accuracy: 0.9979



Accuracy on the ALL test images: 88 %

Accuracy of 0150 : 97 %

Accuracy of 0120 : 85 %

Accuracy of 0012 : 83 %

Accuracy of 0067 : 89 %

Accuracy of 0178 : 97 %

Accuracy of 0157 : 94 %

Accuracy of 0193 : 92 %

Accuracy of 0078 : 83 %

Accuracy of 0096 : 87 %

Accuracy of 0177 : 84 %

Accuracy of 0059 : 81 %

Accuracy of 0033 : 92 %

Accuracy of 0004 : 90 %

Accuracy of 0114 : 95 %

Accuracy of 0186 : 89 %

Accuracy of 0031 : 70 %

Accuracy of 0145 : 95 %

Accuracy of 0087 : 77 %

Accuracy of 0105 : 93 %

Accuracy of 0181 : 92 %

Accuracy of 0126 : 97 %

Accuracy of 0160 : 100 %

Accuracy of 0144 : 80 %

Accuracy of 0046 : 81 %

Accuracy of 0085 : 97 %

Accuracy of 0196 : 97 %

Accuracy of 0027 : 94 %

Accuracy of 0143 : 97 %

Accuracy of 0073 : 90 %

Accuracy of 0130 : 95 %

Accuracy of 0195 : 93 %

Accuracy of 0035 : 87 %

Accuracy of 0032 : 92 %

Accuracy of 0098 : 89 %

Accuracy of 0136 : 91 %

…………. 以下省略完整檔pretraintest

比較實驗結果

有比作業二的準確度高嗎?

沒有

結果討論與原因推論

ResNet模型的核心是通過建立前面層與後面層之間的“短路連接”（快捷方式，跳過連接），這有助於訓練過程中梯度的反向傳播，從而能訓練出更深的CNN網絡。DenseNet模型，它的基本思路與ResNet一致，但是它建立的是前面所有層與後面層的密集連接（密連），它的名稱也是由此而來。HW2使用Resnet-101，本作業使用DenseNet-169模型由圖可知Resnet-101的錯誤較低。

參考資料：<https://zhuanlan.zhihu.com/p/37189203>

