ADARLAB AI Training Course

Lec2 Homework Report

110511118 陳孟頡

Part I. HW1

- **Task :**

  In lec2_hw1.py, calculate the input/output size of each layer in NetHW and the total # of parameters of this model.

- **Input/output size :**

1. conv_1 (in_channel = 3, out_channel = 8, kernel_size = 3, stride = 1, padding = 1):

   原始輸入大小為

   $$input\ size = B * 3(\text{channel}) * 32(\text{height}) * 32(\text{width}) = 3072$$

   計算經過第一層 convolution layer 後輸出大小

   $$out_w = out_h = \frac{32 - 3 + (2 * 1)}{1} + 1 = 32$$

   $$channel: 3 \rightarrow 8$$

   $$output\ size = B * 8 * 32 * 32 = 8192$$

   總結而言，第一層 convolution layer 的 input size 為 B*3*32*32，output size 則為 B*8*32*32。

2. maxpool 1 (kernel_size = 2) :

   所使用 max pooling 之 kernel 大小為 2*2，會將輸入進行 down-sampling，長寬各縮減 1/2，計算經過第一層 max pooling 後輸出大小

   $$out_w = out_h = \frac{32}{2} = 16$$

   $$output\ size = B * 8 * 16 * 16 = 2048$$

   總結而言，第一層 max pooling layer 的 input size 為 B*8*32*32，output size 則為 B*8*16*16。

3. conv_2 (in_channel = 8, out_channel = 8, kernel_size = 5, stride = 1, padding = 2) :
   計算經過第二層 convolution layer 後輸出大小

$$out_w = out_h = \frac{16 - 5 + (2 * 2)}{1} + 1 = 16$$

$$channel : 8 \rightarrow 8$$

$$output\ size = B * 8 * 16 * 16 = 2048$$

   因此，第二層 convolution layer 的 input size 和 output size 同為 B*8*16*16。

4. maxpool 2 (kernel_size = 2) :
   計算經過第二層 max pooling 後輸出大小

$$out_w = out_h = \frac{16}{2} = 8$$

$$output\ size = B * 8 * 8 * 8 = 512$$

   總結而言，第二層 max pooling layer 的 input size 為 B*8*32*32，output size 則為 B*8*16*16。

5. conv_3 (in_channel = 8, out_channel = 16, kernel_size = 3, stride = 1, padding = 1) :
   計算經過第三層 convolution layer 後輸出大小

$$out_w = out_h = \frac{8 - 3 + (2 * 1)}{1} + 1 = 8$$

$$channel : 8 \rightarrow 16$$

$$output\ size = B * 16 * 8 * 8 = 1024$$

   因此，第三層 convolution layer 的 input size 為 B*8*16*16，output size 則為 B*16*8*8。

6. maxpool 3 (kernel_size = 2) :
   計算經過第二層 max pooling 後輸出大小

$$out_w = out_h = \frac{8}{2} = 4$$

$$output\ size = B * 16 * 4 * 4 = 256$$

   總結而言，第二層 max pooling layer 的 input size 為 B*16*8*8，output size 則為 B*16*4*4。

7. flatten :

   為了使讀入的二維資料能進到全連接層(fully connected layer)，因此需先將其 channel、height、weight 攤平為一維，因此，flatten layer 的 input size 為 B*16*4*4，output size 則為 B*256。

8. fc_1 (in_feature = 4*4*16, out_feature = 10) :

   全連接層將 in_feature 的所有 neurons 連接到 out_feature 的 neurons，因此輸出的數量為 out_feature 數。由此可知，fully connected layer 的 input size 為 B*256，output size 為 B*10。

- **Number of parameters :**

  Conv_1 : $3*8*3^2 = 216$

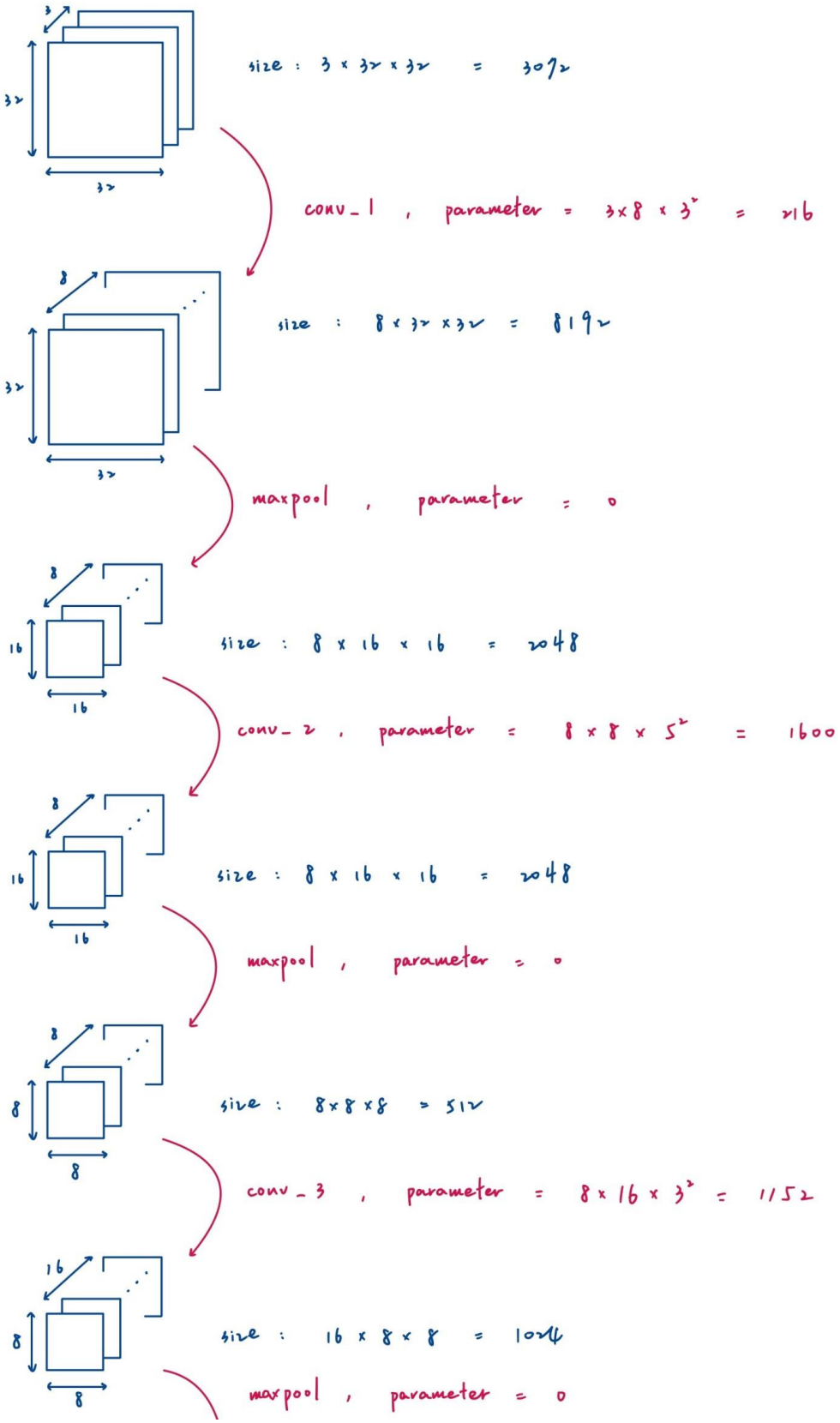
  Conv_2 : $8*8*5^2 = 1600$

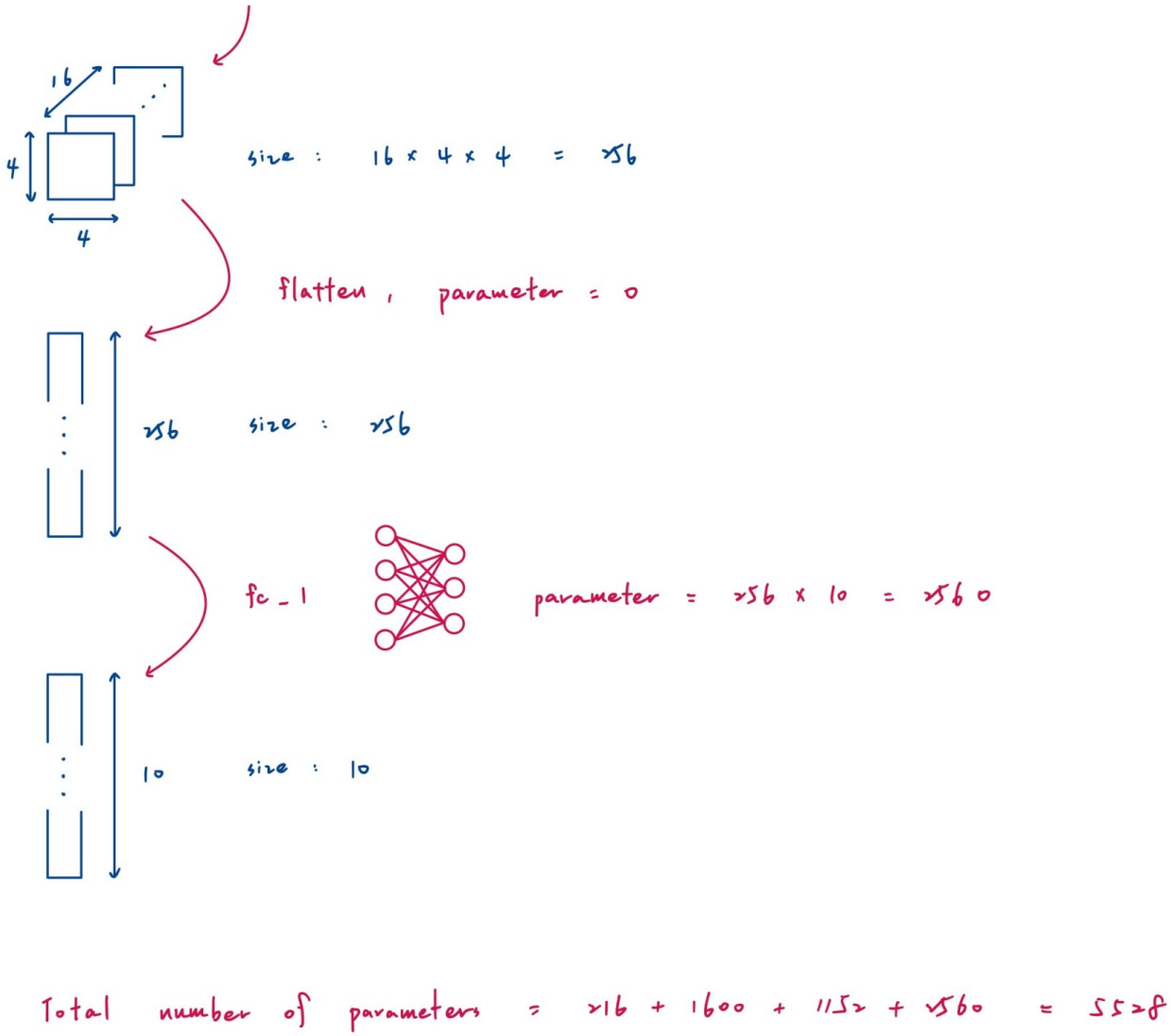
  Conv_3 : $8*16*3^2 = 1152$

  Fc_1 : $256*10 = 2560$

  Total number of parameters = 5528

- Manual calculation





- Tool calculation

```
After conv_1: torch.Size([1, 8, 32, 32])
After maxpool: torch.Size([1, 8, 16, 16])
After conv_2: torch.Size([1, 8, 16, 16])
After maxpool: torch.Size([1, 8, 8, 8])
After conv_3: torch.Size([1, 16, 8, 8])
After maxpool: torch.Size([1, 16, 4, 4])
After flatten: torch.Size([1, 256])
After fc_1: torch.Size([1, 10])
Total number of parameters: 5528
```

Part II. HW2

- **Problem 1 :**

  Please explain what is the residual block (two types) and give the pros and cons of each.

Answer :

在 neuron network 中，隨著架構的層數增加，再進行 back propagation 時，會有梯度減少的問題，使得訓練時間增加，甚至產生難以收斂的情形。因此，當使用的 neuro 斯 network 架構較多層時，需加入 residual block(skip connection block)，使進行 back propagation 參數趨近於 0 時，可選擇跳過某些層數，將 gradient 回傳至來源層。

Residual block 有兩中連接方式，一是 addition，二是 concatenation。Addition 中，normal connection 直接和 skip connection 進行 element-wise 相加，因此兩者 feature maps 的 tensor 維度皆需相同。Concatenation 則是將 normal connection 及 skip connection 的 feature maps 進行 channel 的疊加，因此兩者只需要 feature maps 的 height、weight 相同即可。

以下表格整理 residual block 的優缺點:

Pros	1. 解決梯度下降問題，使訓練較深的神經網路變容易 2. 使用 residual block 的架構通常有較佳的 performance 3. 架構對於各種資料型態的適應能力較佳
Cons	1. 增加計算量 2. 記憶體需求提升 3. 設計複雜度提高 4. 訓練收斂速度較慢

- **Problem 2 :**

Please explain what is the receptive field and how to adjust the receptive field in the neural network.

Answer :

Receptive field(感知範圍)代表 output feature map 所能涵蓋 input feature map 的範圍。

提升感知範圍可根據架構使用不同方法，以下列出其中三種:

1. 增加 kernel size :

   增加 kernel 的大小(以 5*5 代替 3*3)，能使單位 output feature 涵蓋更多的 input feature 資訊(9 單位增加到 25 單位 input feature map / 單位 output feature map)。

2. 增加 stride :

   增加 stride 數會相對減少 output feature map 大小，因此單位 output feature map 會涵蓋更多 input feature map 資訊，達到提升 receptive field 效果。

3. 增加 dilated rate :

   當要維持小 kernel size 來提升 receptive field 時，可增加 dilated rate，也可相對提升 output feature map 涵蓋的 input feature map 資訊。

需注意的是，提升 receptive field，會縮小 output feature map 大小，也會同時損失部分 input 資訊。

- **Problem 3 :**

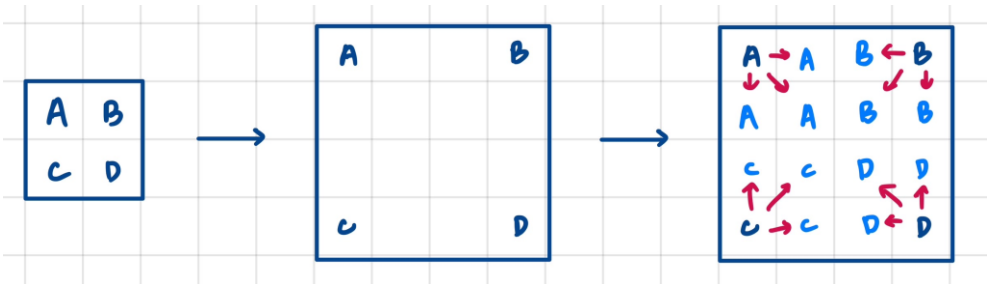
  Please give some methods to achieve feature map upsampling. Explain them with codes and images.

Answer :

現今有許多將 feature map 放大的做法，以下針對較常見的做法 Nearest Neighbor Interpolation、Bilinear Interpolation 以及 Transposed Convolution 進行介紹:

1. Nearest Neighbor Interpolation (最近鄰居插值法)

   最近鄰居插值法做法如其名，是將放大後空白的 pixel 填入相鄰的數值，如下圖:

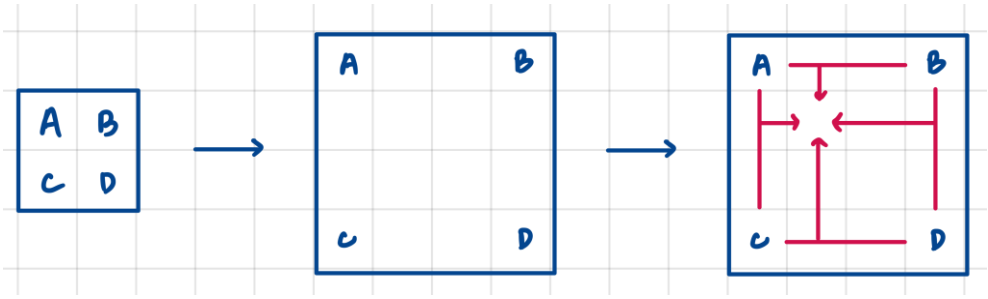


Pytorch 程式範例如下:

```
import torch
import torch.nn.functional as F

# 設定 dummy input tensor 形狀 (batch_size, channels, height, width)
input_tensor = torch.randn(1, 3, 64, 64)

# 將 upsampling scale 設定為 2 進行 Nearest Neighbor Interpolation
output_tensor = F.interpolate(input_tensor, scale_factor=2,
mode='nearest')
```

2. Bilinear Interpolation (雙線性插值法)

雙線性插值法會考慮周圍四個點，並依照距離比例決定插入的數值，圖例如下:

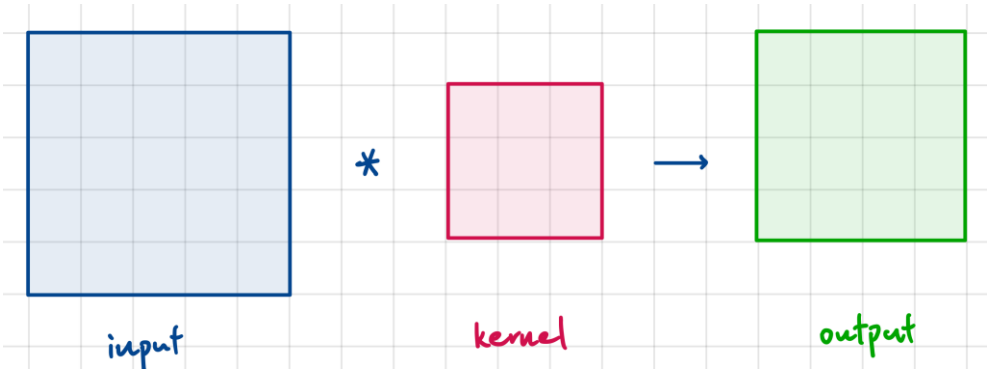


使用雙線性插值法的程式如下:

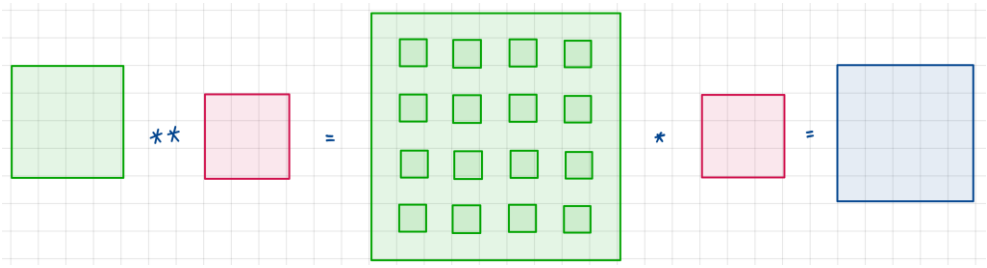
```
import torch
import torch.nn.functional as F

input_tensor = torch.randn(1, 3, 64, 64)

output_tensor = F.interpolate(input_tensor, scale_factor=2,
mode='bilinear', align_corners=True)
```

3. Transposed Convolution (轉置卷積)

轉置卷積會將每個 pixel 或是資料點，透過學習而來的 kernel 線性轉換成多個資料，達到 upsampling 的效果。一般 convolution 示意圖:



進行轉置卷積時，需將 input 進行 padding 等調整，再進行 convolution 運算，示意圖如下:



進行轉置卷積之程式如下:

```
import torch
import torch.nn.functional as F
import torch.nn as nn

# 設定轉置矩陣層
trans_conv = nn.ConvTranspose2d(in_channels=3, out_channels=3,
kernel_size=4, stride=2, padding=1)

# 使用 transposed convolution 進行 upsample
output_tensor = trans_conv(input_tensor)
```