BS6207 Assignment3

- Result and formula of each function

No.	Function	Formula	RMSE
1	MaxPool2d	$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$	0.0
2	AvgPool2d	$out(N_i, C_j, h, w) = \frac{1}{kH * kW} \sum_{m=0}^{kH-1} \sum_{n=0}^{kW-1} input(N_i, C_j, stride[0] \times h + m, stride[1] \times w + n)$	0.0
3	Conv2d	$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$	4.5* e-07
4	ConvTrans-pose2d	This is the gradient of Conv2d with respect to its input, and also known as a fractionally-strided convolution or a deconvolution.	3.3* e-07
5	flatten	Flattening a zero-dimensional tensor will return a one-dimensional view.	0.0
6	sigmoid	$S(x) = \frac{1}{1 + e^{-x}}$	1.1* e-07
7	Roi pooling	Region of interest pooling is used for utilising single feature map for all the proposals generated by RPN in a single pass	0.0
8	batch_norm	$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$	2.8* e-06
9	Cross entropy	$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$	0.0
10	mse_loss	$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = (x_n - y_n)^2,$	2.3* e-07

- Explanation of code and algorithm of several functions

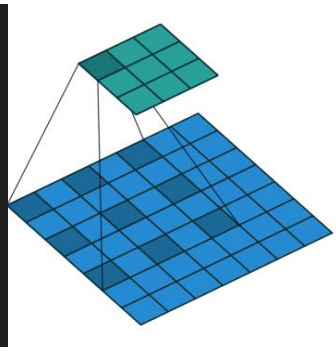
Several functions have relatively more complex algorithms, thus the corresponding explanation is needed.

1. Conv2d:

For the first one that kernel size is 3, dilation=1, the result is the sum of weight times input with the size 3*3 adding bias. In terms of the one with kernel size is 5, dilation=3, stride=2, which equates to kernel size = 9 with 0 filled in. To deal with stride, the index of output should be divided by 2.

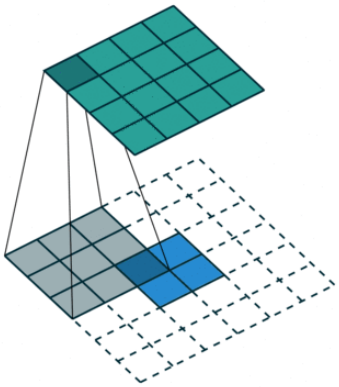
```
def M3_script(input,w,b):
# The script of Conv2d kernel_size=3, dilation=1
  res = []
  for k in range(6):
    img = np.zeros((30,30))
    for i in range(30):
      for j in range(30):
        temp = []
        for g in range(3):
          temp.append(input[0][g][i:i+3,j:j+3] * w)
        img[i][j] = np.sum(temp) + b
    #    new_img.append(new_row)
    res.append(img)
  return res
```

```
def M4_script(input,w,b):
  # the script of Conv2d kernel_size=5, dilation=2
  res = []
  for k in range(6):
    img = np.zeros((12,12))
    for i in range(0,24,2):
      # 24 = 32 - 9 + 1
      for j in range(0,24,2):
        temp = []
        for g in range(3):
          temp.append(input[0][g][i:i+9,j:j+9] * w)
        img[i//2][j//2] = np.sum(temp) + b
  res.append(img)
  return res
```



2. ConvTranspose2d

For this function, first we should transform the input into the one with paddings. Then we can do the same process with that of standard convolution operation. The purpose of this function is to add same weight of each pixel in the input.



```
# build new input by adding interpolation
# 36 = 32 +2+2 each side add 2 * 0
inputT = []
for k in range(3):
  tempT= np.zeros((36,36))
  for i in range(2,34):
    for j in range(2,34):
      tempT[i][j] = array[0][k][i - 2][j - 2]
  inputT.append(tempT)
inputT = np.array(inputT)
inputT.shape
```

```
def M5_script(input,w,b):
  # the script of ConvTranspose2d
  res = []
  for k in range(4):
    img = np.zeros((34,34))
    for i in range(34):
      for j in range(34):
        temp = []
        for g in range(3):
          temp.append(input[g][i:i+3,j:j+3] * w)
        img[i][j] = np.sum(temp) + b
    res.append(img)
  return np.array(res)
```

3. Cross_entropy

This function combines LogSoftmax and NLLLoss in one single class. It is useful when training a classification problem with multiple classes, especially given an unbalanced training set. In other words this function focus more on the classes.

LogSoftmax:

$$\text{LogSoftmax}(x_i) = \log\left(\frac{\exp(x_i)}{\sum_j \exp(x_j)}\right)$$

NLLLoss:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} x_{n,y_n}, \quad w_c = \text{weight}[c] \cdot 1\{c \neq \text{ignore\_index}\},$$

In my script, I combined these two algorithms.

```
import math
def M10_script(array,target):
  # The script of cross_entropy
  res= []
  for k in range(3):
    img = np.zeros((32,32))
    for i in range(32):
      for j in range(32):
        temp = -1 * math.log(np.exp(input[target[k,i],i,j]) / np.exp(input[:,i,j]).sum())
  return np.mean(res)
```