BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
PILANI CAMPUS
FIRST SEMESTER 2018 – 2019
PRINCIPLES OF PROGRAMMING LANGUAGES (CS F301)
ASSIGNMENT

Date: 24/10/2018 **Submission Due: 4/11/2018**
Weightage: 10 % (20 M) **Type: Open Book**

Instructions:

1. Deliverable should be in form of a single Scala file named as ID.scala (Use full BITS ID like F2016A7XXYYYYP.scala).
2. The student has to create the file in a package named "pplAssignment" and object should be named after students ID with added prefix “F” for example "F2016A7PSXXXXP" or ”F2016A7TSXXXXP” (Follow the same conventions for 2015 batch also).
3. **Do not use any function not used in the lab sheets (Please refer to lab sheets for exhaustive list of functions you can use).**
4. Write the program in a purely functional manner. Do not use mutable constructs (like arrays or var) and also do not use random access in list like listint[3],etc.
5. Use scala command line to check your assignment output. Install scala in Ubuntu 16.04 using command ***sudo apt-get install scala***. To test your attempt, Compile using : ***scalac <STUDENT_ID>.scala driver.scala*** and run as: ***scala pplAssignment.Driver t*.in > t*.check***
You can check our output with given output using ***diff t*.check t*.out***. (* stands for 0/1/2.)
6. Before running the driver replace the **<STUDENT_ID>** tag with your own ID prefixed by “F”.
7. Make sure your code terminates within 2 seconds on all the sample inputs provided. Check comments on line 107 in driver.scala
8. The maximum marks you get will depend upon the number of steps on which the output is correct (see Marking Scheme). However this does not mean you will get full marks on all such steps and this will depend upon the logic used and its implementation. Such decisions will depend solely on the discretion of the IC.
9. All submissions will be subjected to plagiarism check.
10. If a submission is found plagiarized, then it will be penalized with negative marking equal to the weightage of the part whose answer is found plagiarized. (Example: If the question is of 5 marks, there will be negative marking of 5 marks for plagiarized answer). Please don’t copy.
11. All submissions have to be made on Nalanda. No submissions will be accepted by email.

Problem Statement:

The aim of the assignment is to implement a two layered convolutional neural network, the assignment is broken down into the following 6 parts (READ THE ENTIRE ASSIGNMENT BEFORE STARTING):

1) **Convolution Layer** : Takes as input a matrix, size of matrix, kernel, size of kernel and returns the convoluted output.
   Part I:
   Dot Product: Takes two matrices and returns a single scalar value which is the dot product of both the matrices.
   Part II:
   Convolute: Uses the above dot product module to compute the final convoluted output.

2) **Activation Layer** : Takes a matrix and an activation function as input and returns the activated layer as output. How to take activation function as an input parameter is explained in further sections.
3) **Pooling Layer**: Takes a matrix, a pooling function, pooling region as input and returns the pooling layer as output. How to take pooling function as an input parameter is explained in further sections.
   Part I:
   Single Pooling
   Takes a K*M matrix(list of list) (K rows, M columns), pooling function ,pooling region(same as K in this function) as input and returns a list(1-D) of size M/K where each element is calculated by applying the pooling function over the input matrix (assume M is divisible by K and please don't ask if K can be 0).
   Part II
   Pooling Layer
   Use the above SinglePooling module to compute the final output.

   Note: Assume pooling region and matrix dimensions will be compatible

4) **Normalization** : Takes a matrix and outputs the normalized matrix with each value between 0 to 255 (Int type).
5) **Mixed layer**: Mixed layer takes a matrix(image), kernel, size of image, size of kernel, activation function, pooling function and returns the final output. The final output is calculated in the following manner:

   Image → Convolution Layer(Kernel) → Activation Layer (activation function) → Pooling Layer (pooling function) → **Output**
6) **Assembly** : The assembly function takes a matrix(image), image size, weight 1, weight 2, bias, kernel 1,kernel 2,kernel size 1,kernel size 2,pooling region and output the following result:

Image + Kernel 1 → Mixed Layer→ Temporary_output_1

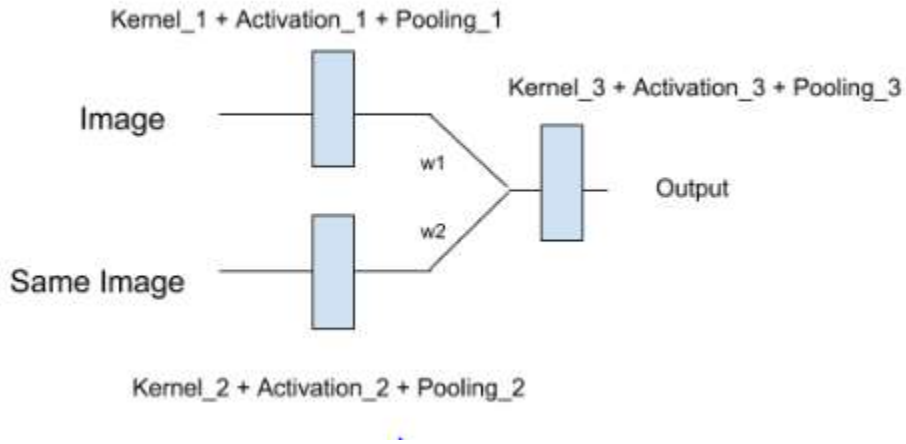
Image + Kernel 2 → Mixed Layer → Temporary_output_2

Temporary_output_1***weight_1*** + Temporarary_output_2***weight_2*** + **Bias** --> Temporary_output_3

Temporary_output_3 + Kernel 3 → Mixed Layer→ Temporary_output_4

Temporary_output_4 → Normalization → **Output**

<u>Please see the below photo to get a clearer understanding of the above process</u>



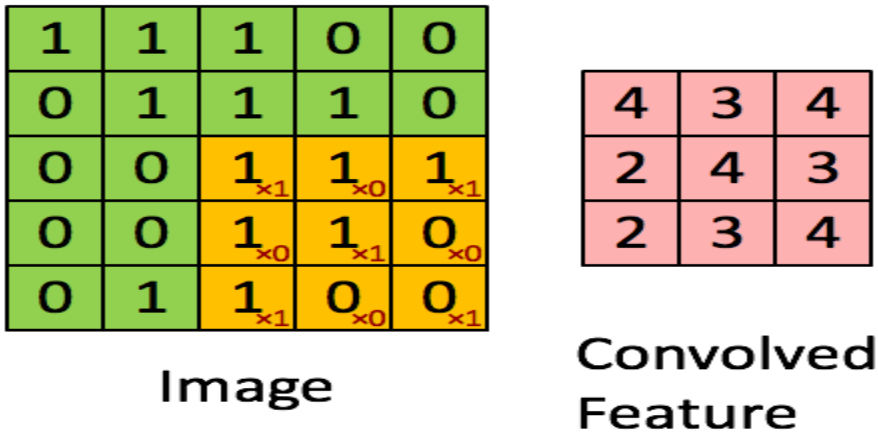
Note: The activation functions and pooling layer functions to be used are mentioned below.

Algorithm

The following sections explains the algorithms involved in the first 3 sections : Convolution, Activation and Pooling. The other 3 modules (Mixed, assembly and normalization) are self-explanatory and are dependent on the correct use of previously implemented modules. That is, Mixed layer depends on correctly using Convolutional, Activation and Pooling modules.

1) Convolutional Layer

   A convolution filter passes over all the pixels of the image in such a manner that, at a given time, we take 'dot product' of the convolution filter and the image pixels to get one final value output. We do this hoping that the weights (or values) in the convolution filter, when multiplied with corresponding image pixels, gives us a value that best represents those image pixels.



   Here the kernel used is [[1, 0, 1], [0, 1, 0], [1, 0, 1]]

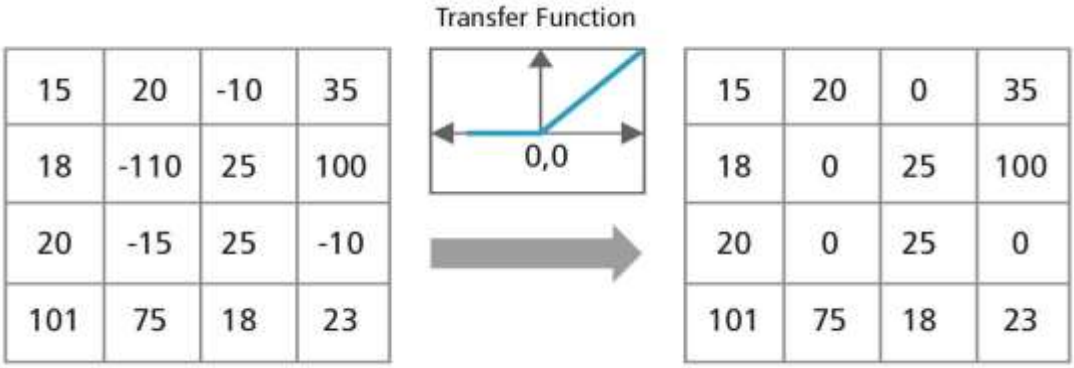
   For an intuitive understanding of the convolutional layer, please refer to the following link
   https://icecreamlabs.com/2018/08/19/3x3-convolution-filters%E2%80%8A-%E2%80%8Aa-popular-choice

2) Activation Layer

   The activation layer is used to introduce non-linearity in a neural network. It takes a matrix ‘M’ and an activation function ‘f’ (R→R), the output matrix ‘O’ is calculated by applying the function ‘f’ on each element of M.

   ReLu : max(0,x)
   Leaky ReLu : x if x>0 , 0.5*x if x < 0

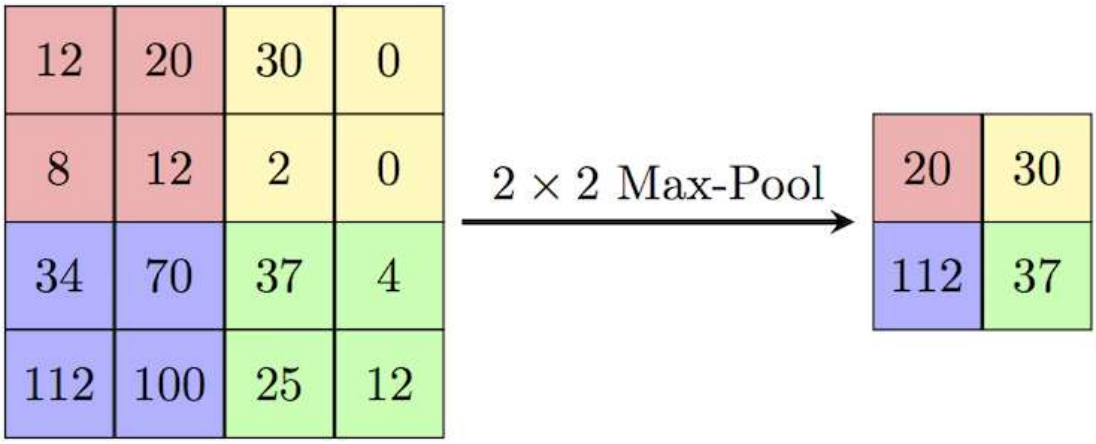
The following example is of **ReLu** activation : Apart from ReLu, you will be asked to implement the l**eaky ReLu.**



3) Pooling Layer

The pooling layer is used to decrease the feature map size. You will be asked to implement two pooling layer functions : Max and Average. Below you have been provided example of how a max pooling layer works.

Note : Some of you students who have done the Andrew Ng course maybe wondering if the pooling region will be fixed at 2*2, the answer is **No**. In your code you **cannot** hardcode your pooling layer function so as to only work for 2*2 pooling region. Sorry!



Steps:

Note : The parameters are mentioned in the order they must be passed. Please do not change the order in your implementations. Data Type must be **double** unless specifically mentioned.

1) Convolution Layer :

Part I

Name of function : dotProduct
Parameters :
matrix_1:List[List[Double]],
matrix_2: List[List[Double]]
Output :
Double

$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$

$$= (1 \times 0) + (0 \times -1) + (-1 \times -1) + (0 \times 0)$$
$$= 0 + 0 + 1 + 0 \quad = 1$$

Input :
Matrix 1
[[1, 0], [-1, 0]]

Matrix 2
[[0, -1], [-1, 0]]

Output:
1

Part II

Name of function : convolute
Parameters :
Image:List[List[Double]],
Kernel:List[List[Double]],
imageSize:List[Int],
kernelSize:List[Int]
Output :
List[List[Double]]

Note: imageSize will store the number of rows and column in the Image, that is imageSize.head and imageSize.tail.head will give the number of rows and columns respectively.

IMAGE (In a list of list format)

0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8
0 2 4 6 8 10 12 14 16
0 3 6 9 12 15 18 21 24
0 4 8 12 16 20 24 28 32
0 5 10 15 20 25 30 35 40
0 6 12 18 24 30 36 42 48
0 7 14 21 28 35 42 49 56
0 8 16 24 32 40 48 56 64

KERNEL (in a list of list format)

0 0 0
0 1 2
0 2 4

imageSize:
[9,9]

kernelSize:
[3,3]

Example Output: (in a list of list format)

25 40 55 70 85 100 115
40 64 88 112 136 160 184
55 88 121 154 187 220 253
70 112 154 196 238 280 322
85 136 187 238 289 340 391
100 160 220 280 340 400 460
115 184 253 322 391 460 529

2) Activation Layer:

Name of function : activationLayer
Parameters :
activationFunc:Double => Double,
Image:List[List[Double]]

Output :
List[List[Double]]

Image:

25 40 55 70
40 64 88 112
55 88 121 154
70 112 154 196

activationFunc: (If you have seen the second lab sheet or attended the lab, you must know what this is)
(x:Int)=>x-1

Note: Here activationFunc is a function passed as a parameter. The activationFunc takes double as input and returns a double. This activationFunc that is passed is applied on each element of the matrix to get the output.

Output:
24 39 54 69
39 63 87 111
54 87 120 153
69 111 153 195

3) Pooling Layer:

Part I

Name of function : singlePooling
Parameters :
poolingFunc:List[Double]=>Double,
Image:List[List[Double]] (Dimension will always be K * M, where M is multiple of K),
K:Int (This is the pooling region, for example if K=2 then the pooling region will be 2*2)
Output :
List[Double] (Dimension should be M/K)

Image:
25 40 55 70
40 64 88 112

poolingFunc: Max Pooling (see parameter signature)

K = 2

Note: Here poolingFunc is passed as a parameter. It takes a list of doubles and returns a single value. So for example if you need to run pooling layer with max pooling your poolingFunc should take a list of doubles and return the maximum value. In the above Image example, the poolingFunc will be used to take [25, 40, 40, 64] as input and will return 64 as the output.

Output:
64 112

Note: Use the color coding for intuitive understanding of what is happening. Orange matches to orange and green matches to green.

Part II

Name of function : poolingLayer
Parameters :
poolingFunc:List[Double]=>Double,
Image:List[List[Double]],
K:Int
Output:
List[List[Double]]

Image
24 39 54 69
39 63 87 111
54 87 120 153
69 111 153 195

PoolingLayer: Max Pooling

K = 2

Output:

63 111
111 195

4) Mixed Layer:
Name of function : mixedLayer

Parameters :

Image:List[List[Double]],
Kernel:List[List[Double]],
imageSize:List[Int],
kernelSize:List[Int],
activationFunc:Double => Double,
poolingFunc:List[Double]=>Double,
K:Int

Output :

List[List[Double]]

5) Normalisation

Name of function : normalise

Parameters :

Image:List[List[Double]]

Output :

List[List[Int]]

(IMAGE CAN HAVE INPUT NUMBER IN ANY RANGE NORMALIZE IT BY USING MAX NUMBER IN MATRIX AS 255 AND MIN AS 0. ROUND IT FOR OUTPUT. You can use .round method)

Image:
0.1, 0.4, 0.5
0.2, 0.8, 0.1
0.6, 0.9, 0.45

Output:
0, 95, 127
31, 223, 0
159, 255, 111

6) Name of function : assembly

Parameters :

Image:List[List[Double]],
imageSize:List[Int],
w1:Double,
w2:Double,

b:Double,
Kernel1:List[List[Double]],
kernelSize1:List[Int],
Kernel2:List[List[Double]],
kernelSize2:List[Int],
Kernel3:List[List[Double]],
kernelSize3:List[Int],
Size: Int

Output :

List[List[Int]]

CHECK TEST CASE FOR SAMPLE INPUT OUTPUT

Sample Input in terms of parameters passed to assembly function in driver

Sample output in terms of res 8 in testcase output

Note: Here activation function and pooling function are not taken as input parameters, therefore assume, use of ReLu for mixedLayer_1 and mixedLayer_2 and leaky ReLu for mixedLayer_3 and max pooling for mixedLayer_3 and avg pooling for mixedLayer_1 and mixedLayer_2.

GIVEN RESOURCES:

driver.scala - Don't Edit anything apart from replacing your Student Id. You can comment un-comment 109 and 110 lines to check time of code execution. Default Output is with 109 commented and 110 uncommented.

<Student_Id>.scala - Change file name to match given format. Eg - "F2015A7PS0127P.scala"

t*.in - testcase input . Input is read automatically in driver file and presented as multiple val's.

t*.out - testcase output . Should be matched exactly

MARKING SCHEME

Topic	**Marks**
Convolution Layer - part1	**2**
Convolution Layer - part2	**4**
Activation Layer	**2**
Pooling Layer - part 1	**4**
Pooling Layer - part 2	**2**
Normalization	**1**
Mixed Layer	**2**
Assembly	**3**
Total	**20**