**1. Theoretical Tasks**

Often enough, Convolutional Neural Networks (CNN) are built by stacking blocks composed of a convolutional layer followed by a nonlinearity and by a max pooling layer. At the end, it is common to have a fully connected layer to classify the extracted features. Finally softmax operation is performed. In the exercise below you have to calculate by hand each one of the steps.

**Task 1.1 Convolution**

**Theoretical Background**

Read the following two blog posts:

• What are convolutions?

• Convolutions and Neural Networks

**Practice**

Now consider the following image *I*, represented as a matrix *I*:

And the following kernel *k*, represented as a matrix *k*:

I = | 1 1 1 1 | k = | 0 1 0 |

| 1 1 2 1 | | 1 2 1 |

| 1 -3 -4 1 | | 0 1 0 |

| 1 1 1 1 |

Calculate a *same* convolution *I ∗ k* as described in Convolutions and Neural Networks above. Use zero padding for handling the margins. Since its a *same* convolution, use (1*,* 1) stride. DO NOT use a computer to compute this operation, do it by hand - yes, I know, I know. . . but it takes you 5 minutes and it’s good to do it at least *once* in life.

1. Pad the input image with zeros to preserve the spatial dimensions after applying the convolution:

| 0 0 0 0 0 0 |

| 0 1 1 1 1 0 |

| 0 1 1 2 1 0 |

| 0 1 -3 -4 1 0 |

| 0 1 1 1 1 0 |

| 0 0 0 0 0 0 |

I\*k=

(row 1, col 1): (00 + 01 + 00) + (01 + 12 + 11) + (00 + 01 + 00) = 3

(row 1, col 2): (00 + 11 + 10) + (11 + 12 + 21) + (10 + 11 + 10) = 5

(row 1, col 3): (00 + 11 + 10) + (11 + 12 + 11) + (10 + 11 + 1\*0) = 5

(row 1, col 4): (10 + 11 + 10) + (11 + 12 + 11) + (10 + 11 + 0\*0) = 3

(row 2, col 1): (00 + 11 + 10) + (01 + 12 + 11) + (00 + 11 + 1\*0) = 3

(row 2, col 2): (10 + 11 + 10) + (11 + 12 + 21) + (10 + 11 + 1\*0) = 2

(row 2, col 3): (10 + 11 + 10) + (11 + 12 + 11) + (10 + 21 + 1\*0) = 0

(row 2, col 4): (10 + 21 + 10) + (11 + 12 + 11) + (20 + 11 + 0\*0) = 3

(row 3, col 1): (10 + 11 + 10) + (11 + 12 + 11) + (10 + 11 + 1\*0) = 3

(row 3, col 2): (10 + 11 + 20) + (11 + -32 + -41) + (10 + 11 + 1\*0) = -10

(row 3, col 3): (10 + 21 + 10) + (11 + -32 + -41) + (10 + 11 + 10) = -9

(row 3, col 4): (20 + 11 + 10) + (11 + 12 + 11) + (10 + 11 + 0\*0) = 3

(row 4, col 1): (10 + 11 + 10) + (11 + 12 + -31) + (10 + 11 + 0\*0) = 3

(row 4, col 2): (10 - 31 - 40) + (11 + 12 + 11) + (00 + 11 + 1\*0) = 5

(row 4, col 3): (-30 - 41 + 10) + (11 + 12 + 11) + (00 + 11 + 1\*0) = 5

(row 4, col 4): (10 - 41 + 10) + (11 + 12 + 11) + (-30 + 11 + 0\*0) = 3

The resulting matrix is:

| 3 5 5 3 |

| 3 2 0 3 |

| 3 -10 -9 3 |

| 3 5 5 3 |

**Task 1.2 Non Linearity**

**Theoretical Background**

In general, Convolutional layers are followed by a nonlinearity i.e. a non linear function is applied to their output. Without the nonlinearity all layers could be collapsed to a single matrix multiplication (non desirable). These functions play a major role affecting the learning (or non learning!) of neural networks. Common ones are sigmoid-like (sigmoid, tanh, softsign, . . . ) and ReLU-like (ReLU, LeakyReLU, . . . ).

1

**Practice**

Apply the ReLU activation function to the result of previous task.

ReLU(C) = | 3 5 5 3 |

| 3 2 0 3 |

| 3 0 0 3 |

| 3 5 5 3 |

After applying the ReLU activation function, all negative values in the matrix are replaced with 0, and the positive values remain unchanged. This nonlinearity helps the neural network to learn complex patterns and prevents the network from collapsing into a single matrix multiplication.

**Task 1.3 Max Pooling**

**Theoretical Background**

Max Pooling divides the input image in several sections of a given size. This size is often referred to as *subsample size*, or *filter size*. Then, for each section, we only return the biggest value present in that section. For example in Figure 10 here the sections are of size (2*,* 2). It is easy to notice that the output image will have a smaller size than the input. In this case, because the filter size is (2*,* 2), the final size of the image half of the original image. If it were (3*,* 3), then the final size would be one third of the original image.

**Practice**

Apply *valid* Max Pooling with a filter size of (2*,* 2) on the result of previous task. Consider a stride of (2*,* 2). Notice that often enough if the stride is not provided it means it equal to the filter size (bad practice!).

Max Pooling with filter size (2, 2) and stride (2, 2):

First, divide the matrix into non-overlapping sections of size (2, 2):

Section 1: | 3 5 |

Section 2: | 5 3 |

Section 3: | 3 0 |

Section 4: | 0 3 |

| 3 2 | | 0 3 |

| 3 5 | | 5 3 |

For each section, return the maximum value:

Max of Section 1: 5

Max of Section 2: 5

Max of Section 3: 5

Max of Section 4: 5

Resulting matrix after Max Pooling:

| 5 5 |

| 5 5 |

The Max Pooling operation reduces the size of the matrix, in this case by a factor of 2, and retains the most significant features within each section. This helps to reduce the computation complexity and control overfitting in the neural network.

**Task 1.4 Flattening**

**Theoretical Background**

Flattening is the process of reshaping a matrix into a one dimensional vector e.g by putting all the rows of the image in one same line. For example, the image below:

will become:

*I* =

0 1 2 3

*If lattened* = [0*,* 1*,* 2*,* 3]*T*

**Practice**

Flatten the result of the previous task.

Flattened = [5, 5, 5, 5]

**Task 1.5 Fully Connected Layer**

**Theoretical Background**

After extracting the features a fully connected layer is used to perform classifi cation. This is consisting in a simple matrix multiplication where there weight matrix of the layer has are as many rows as classes and as many columns as features. For this reason the flattening operation the we just performed is very handy.

**Practice**

Calculate the output of a Fully Connected layer. Use the following *W*:

W = | 1 2 3 4 |

| 5 6 7 8 |

FC\_Output = W x Flattened = [50, 130]

**Task 1.6 SoftMax**

**Theoretical Background**

Finally, a softmax operation is performed to transform the raw output of the network into probabilities. Eventually, the highest number after the softmax operation will be selected as output class.

**Practice**

Apply a softmax to the output of the previous task and indicate which is the output class (1st or 2nd).

Calculate the exponentials:

exp(50) ≈ 5.18470553e+21

exp(130) ≈ 3.06666348e+56

Calculate the sum of exponentials:

sum\_exp ≈ 5.18470553e+21 + 3.06666348e+56 ≈ 3.06666348e+56

Divide each element's exponential by the sum of exponentials:

softmax(50) ≈ 5.18470553e+21 / 3.06666348e+56 ≈ 1.68924337e-35 ≈ 0

softmax(130) ≈ 3.06666348e+56 / 3.06666348e+56 ≈ 1

Softmax output: [0,1]

The output class is the one with the highest probability. In this case, it's the 2nd class, as its probability is 1.

3