Joan Grau Noël - 172578

Practice 4 – Deep Learning

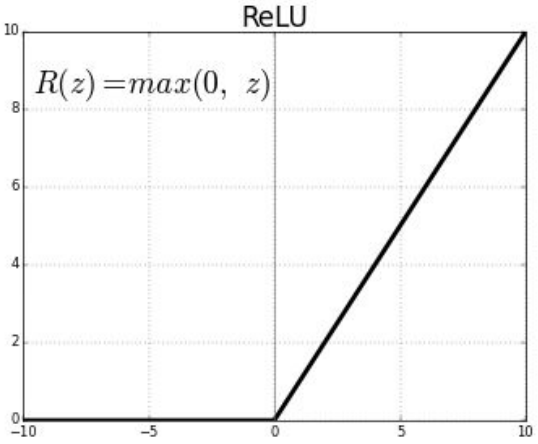
PART 1

- **First of all** we are asked to take a look into the layers that AlexNet has build into it. There is a total of 25 different layers. A lot are repeated more than once, so we don't have to name them all:

  - **Image Input**: this is the image input of the dataset, which is the raw images we load into our program

  - **Convolution**: It's the convolving layer, where we will convolve our filters with our image in order to obtain new features that our Network can learn from them. Each one has a different depth and number of filters in it. For example, this one just has three filters. The next one, has 96.

  -**ReLU**: It's the rectifier Linear function, which is actually a non-linear activation function for our network. What it does, is find the max between the pixel value of the filter output and zero, and then activates on the following manner (in this case, x is z):



- **Cross Channel Normalization**: Is a method used when using ReLu on our network. What it basically does, is amplify the neurons, or in this case, inputs with higher values, while lowering the outputs in its neighboring area of size N. It also does it between all the channels we found before.
- **Max Pooling**: This is the Max Pooling area, what it does is find on a local area the biggest value, and it keeps it while removing the rest. For Example:

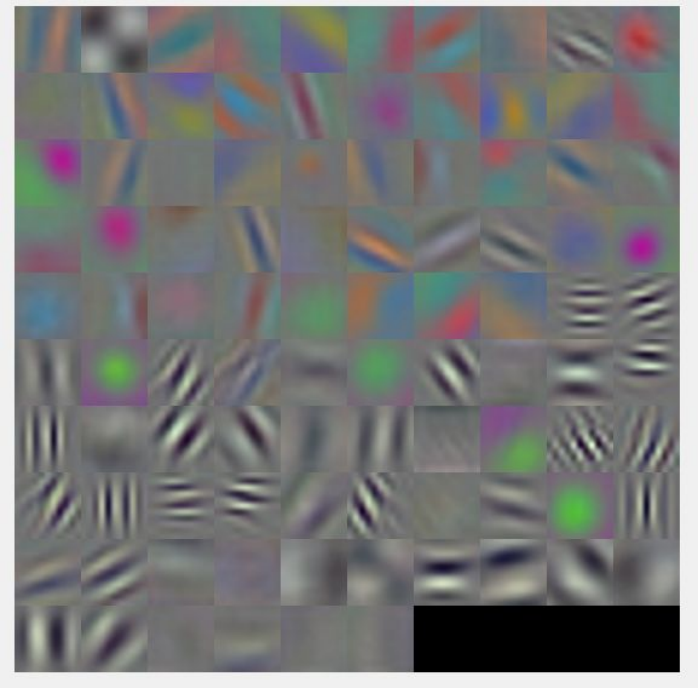
1	3
7	2

=

7

- **Fully Connected**: This is the fully connected layer, which is a standard NN made of neurons connected between different hidden layers, and with different weights between neurons, unlike the convolutional layers.
- **Dropout**: What this does is drop out units, which would be the neurons, in the outputs of the fully connected layer in order to avoid overfitting. It does so with a 50% rate, so oonly half of the outputs are kept.
- **Softmax**: It's a probablity based method that normalizes the probabilities accross all the outputs in order to classify.
- **Classification Output**: The final output class where the input is classified.

- **Plotting the first output weights**:



As we can see, the filters have different directions in all the cases. This means that these filters are searching for shapes in the image, so what they are trying to do is identify the objects by the way they look. In some cases, we can see how there is also colour added to these filters, this means that probably, in this case, the colour is a meaningful feature that takes part into the dataset, and must be used.

- **Getting the accuracy on our dataset**

By plotting the accuracy obtained in our dataset, we get one of **0.8624**, which is very good, considering that we used the seven different categories at the same time.

The only change we had to make in the code to make it work was change the value of number of images inside the reshape function.

PART 2

In this part we are asked to build our own CNN and see what results it yields.

First of all, we will not use the folders 2, 4 and 6 in the CKDB dataset, because they contain too little dataset inputs.

First of all, we build our dataset by loading the data into a datastore object, like on the example we were given:

```
imDatasetPath = fullfile(pwd, 'CKDB');
imData = imageDatastore(imDatasetPath, 'IncludeSubfolders',true,'LabelSource','foldernames');
```

Next, we will do so that the different labels or categories in this dataset, have all the same number of variables. That is made by searching for the label with the least number of data and then reducing the rest to the same number:

```
numlabels = countEachLabel(imData);
minSetCount = min(numlabels{:,2});
imData = splitEachLabel(imData, minSetCount, 'randomize');
```

Finally, we create a training set with half of the overall variables for each label:

```
NumTrain = int8(min(numlabels{:,2})/2);
[trainData, labelData] = splitEachLabel(imData,NumTrain,'randomize');
```

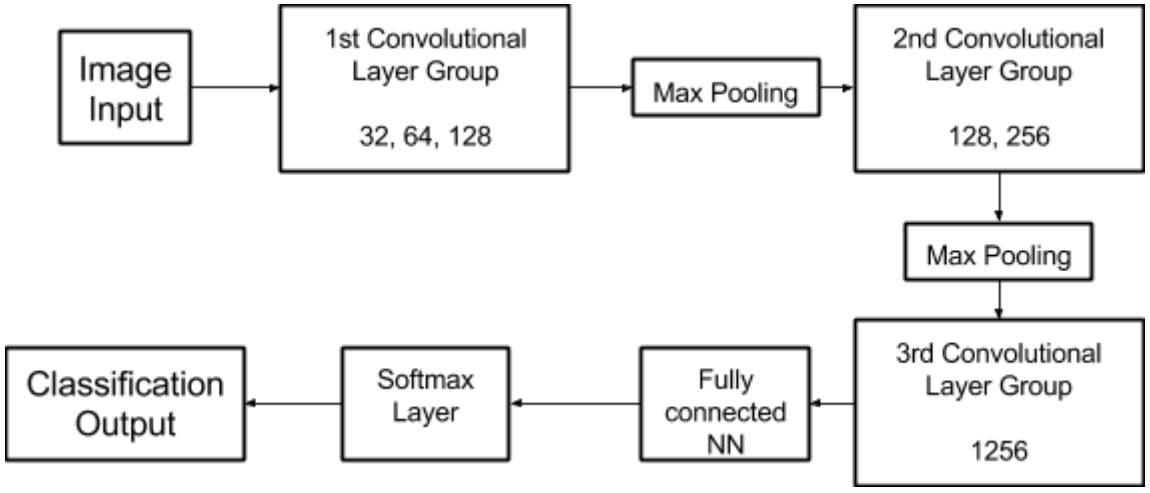
Now it's time to create our CNN architecture.

We will inspire our architecture with the AlexNet architecture that we used before, but since this is a smaller dataset, it's scale will also be smaller.

Also, we couldn't put the *Output* and *Cross Channel Normalization* layers, because they are not directly available in matlab.

What we thought it would work, is to create different convolving layers, with increasing number of filters on each one of them, and between the different groups of this layers, put a max Pooling layer between them.

The overall architecture it's the following:



We think it could work because by starting with low values of filters, and then increasing them, we think that what the Network will do is first focus on the big details and shapes of the problem and later on the details that are more important for classifying between classes.

Also, putting a Max Pool layer between them it's useful because it reduces the cost of computing a lot of filters on the dataset.

To add to this, after each Convolutional Layer, there is a ReLU layer.

- **Results**

We were asked to design an architecture that could yield the best results possible. The one we explained before it's the one that gave us the best results, with the following accuracy:

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.93	1.6097	18.26%	0.0100
30	30	25.82	1.5609	55.65%	0.0100

We know it's very far from the AlexNet accuracy, but this is also a smaller scale network, which also provides a good accuracy when classifying between 5 different classes.

We think that despite this accuracy, we should have gotten a better one for using a Neural Network. We think it's because we are missing some components here, like the Output layer, and the Cross Channel Normalization layer, which we have explained before.