**Implement Convolution and pooling from scratch**

You will be implementing the building blocks of a convolutional neural network! Implement each function mentioned below, show its usage with result and give your analysis for each function and visualization of the image.

Packages to be used: **numpy** and **matplotlib** **ONLY**

**Any plagiarism in code or your analysis found will lead to zero marks.**

Functions that you will implement are:

* Convolution functions, including:
  + **Zero Padding:**
    - Zero-padding adds zeros around the border of an image:

The main benefits of padding are the following:

* It allows you to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as you go to deeper layers. An important special case is the "same" convolution, in which the height/width is exactly preserved after one layer.
* It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.

**Exercise**: Implement the following function, which pads all the images of a batch of examples X with zeros. Use np.pad. Note if you want to pad the array "a" of shape (5,5,5,5,5)  with pad = 1 for the 2nd dimension, pad = 3 for the 4th dimension and pad = 0for the rest, you would do:

a = np.pad(a, ((0,0), (1,1), (0,0), (3,3), (0,0)), 'constant', constant values = (..,..))

* + **Convolve window:**

In this part, implement a single step of convolution, in which you apply the filter to a single position of the input. This will be used to build a convolutional unit, which:

* Takes an input volume
* Applies a filter at every position of the input
* Outputs another volume (usually of different size)
* In a computer vision application, each value in the matrix on the left corresponds to a single pixel value, and we convolve a 3x3 filter with the image by multiplying its values element-wise with the original matrix, then summing them up and adding a bias. Here, you will implement a single step of convolution, corresponding to applying a filter to just one of the positions to get a single real-valued output.
  + **Convolution forward:**

In the forward pass, you will take many filters and convolve them on the input. Each 'convolution' gives you a 2D matrix output. You will then stack these outputs to get a 3D volume:

**Exercise**: Implement the function below to convolve the filters W on an input activation A\_prev. This function takes as input A\_prev, the activations output by the previous layer (for a batch of m inputs), F filters/weights denoted by W, and a bias vector denoted by b, where each filter has its own (single) bias. Finally you also have access to the hyperparameters dictionary which contains the stride and the padding.

**Hint**:

* To select a 2x2 slice at the upper left corner of a matrix "a\_prev" (shape (5,5,3)), you would do:
  + - * a\_slice\_prev = a\_prev[0:2,0:2,:]

This will be useful when you will define a\_slice\_prev, using the start/end indexes you will define.

* To define a\_slice you will need to first define its corners vert\_start, vert\_end, horiz\_start and horiz\_end.
* Pooling function:
  + **Pooling forward:** The pooling (POOL) layer reduces the height and width of the input. It helps reduce computation, as well as helps make feature detectors more invariant to its position in the input.
  + Now, you are going to implement MAX-POOL and AVG-POOL, in the same function.
  + **Exercise**: Implement the forward pass of the pooling layer.

Note: Give your overall analysis (in your own words) on how the features are extracted using CNN model in image dataset. Describe the role of back propagation in CNN