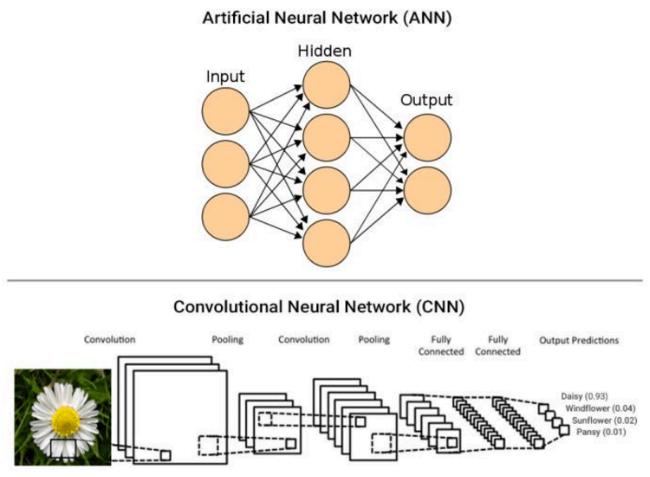
Extension III: Neural Network - Adding Convolution and Pooling Layers.

1. Introduction:
In homework 8, we were asked to modify the neural network implementation we discussed in class. What we did in homework:
(a) Add a regularization term to the cost function
(b) Try using the ReLU activation function
(c) Try using the tanh activation function

2. Extension:
The extension we made on Neural Network was to add convolution and pooling layers to it and made Artificial Neural Network to Convolutional Neural Network.
DataSet used:
(a). load_digits from sklearn.datasets (homework)
(b). http://yann.lecun.com/exdb/mnist/ (test)

3. Explanation (how it works):
The reason why we decided to add convolution and pooling layers to our dataset is below:
Generally speaking, an ANN is a collection of connected and tunable units (a.k.a. nodes, neurons, and artificial neurons) which can pass a signal (usually a real-valued number) from a unit to another. A CNN, in specific, has one or more layers of convolution units. A convolution unit receives its input from multiple units from the previous layer which together create a proximity.
Thus, the relationship between each neurons in CNN will be much more bounded than ANN. Therefore, the accuracy will be better. (see photo below)
// reference: Borhan Kazimipour, https://ai.stackexchange.com/questions/5546/what-is-the-difference-between-a-convolutional-neural-network-and-a-regular-neur



// photo credit: Sathiesh Kumar, "Flower species recognition system using convolution neural networks and transfer learning"

Implementations:

(a). add convolution operation

```python
def convolution(image, filt, bias, s=1):
    (n_f, n_c_f, f, _) = filt.shape # filter dimensions
    n_c, in_dim, _ = image.shape # image dimensions

    out_dim = int((in_dim - f)/s)+1 # calculate output dimensions


    out = np.zeros((n_f,out_dim,out_dim))

    # convolve the filter over every part of the image, adding the bias at each step.
    for curr_f in range(n_f):
        curr_y = out_y = 0
        while curr_y + f <= in_dim:
            curr_x = out_x = 0
            while curr_x + f <= in_dim:
                out[curr_f, out_y, out_x] = np.sum(filt[curr_f] * image[:,curr_y:curr_y+f, curr_x:curr_x+f]) + bias[cu
                curr_x += s
                out_x += 1
            curr_y += s
            out_y += 1

    return out
```

(b). add max pooling

```python
def maxpool(image, f=2, s=2):
    n_c, h_prev, w_prev = image.shape

    h = int((h_prev - f)/s)+1
    w = int((w_prev - f)/s)+1

    downsampled = np.zeros((n_c, h, w))
    for i in range(n_c):
        curr_y = out_y = 0
        while curr_y + f <= h_prev:
            curr_x = out_x = 0
            while curr_x + f <= w_prev:
                downsampled[i, out_y, out_x] = np.max(image[i, curr_y:curr_y+f, curr_x:curr_x+f])
                curr_x += s
                out_x += 1
            curr_y += s
            out_y += 1
    return downsampled
```

5. Result:

	NN (HW8)	**NN with convolution and pooling layers**
Accuracy (load_digits)	87%	90.5%
Accuracy (the mnist database)	75.6%	88%

Reasonable analysis:

As shown on the table, adding convolution and pooling layers did improve the accuracy in two datasets (both homework dataset and the mnist database). As the CNN process breaks down the image into couple of features and analyze them independently. The neurons are fully connected so that they are more bounded. Thus, the final classification will be more precise.