Final Project Report

Aditya Karanjkar
Cristhian Cruz

Image recognition with Convolutional Neural Networks

Image recognition is the task of determining whether an image contains a specific object, feature or activity. Image recognition is one of the primary tasks in the cross disciplinary field of computer vision and machine learning. Image recognition can be broadly classified in the following categories:

- Object classification: This is the set of tasks where the image of an object is classified into one of the many predefined classes. Eg: Classifying the image of an animal as a mammal, amphibian or a reptile etc.
- Identification: An object in the provided image is identified. Eg: identifying a person based on a photograph.
- Detection: Searching the image for the presence of one or more objects. Eg: Detecting the presence of tumors from a medical image.

Convolutional Neural Network (CNN), also known as a ConvNet, is a machine learning technique that is currently among the most popular methods for image recognition. CNNs have demonstrated that they can be used for image recognition with very high accuracy, approaching those of humans.

Artificial neural networks

An Artificial Neural Network (ANN) is a type of a biologically inspired computational model that is analogous to the neurons in the human brain. An ANN consists of interconnected layers of artificial neurons, also known as perceptrons. A perceptron is the basic building block of a neural network. A perceptron in a neural network consists of a set of weights for each one of the inputs to the neuron and an activation function. The final output of the neuron is calculated by applying the activation function to the weighted sum of the inputs to the neuron:

$$y = \varphi\left(\sum_{1}^{n} w_i x_i\right)$$

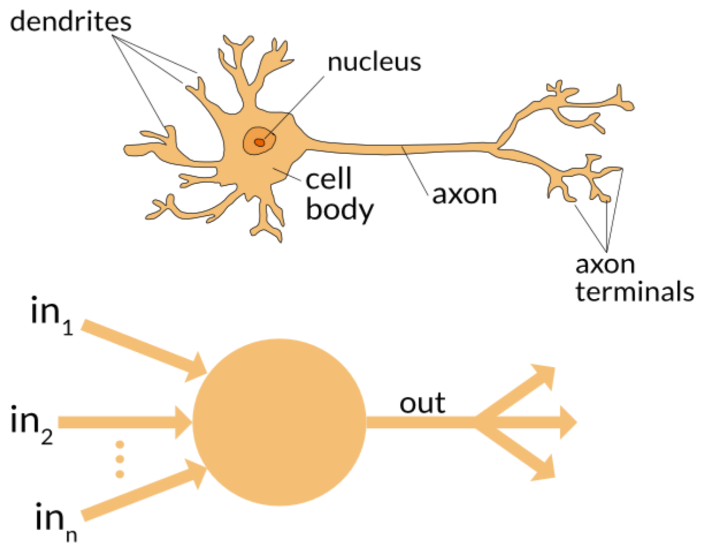

Figure 1: A biological neuron and an artificial neuron

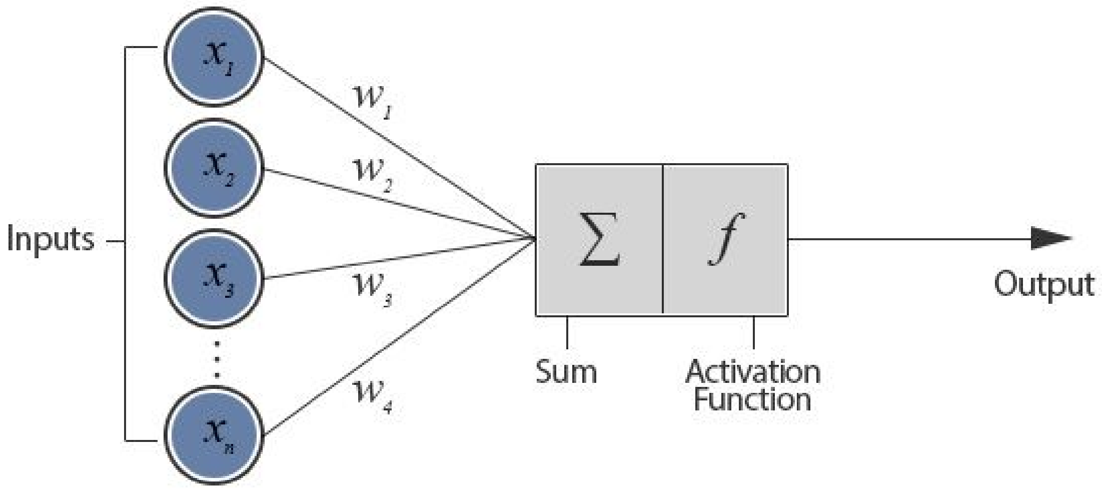

Figure 2: Output of a neuron

In typical neural network structures, the outputs of one layer of perceptrons serve as the inputs to the perceptrons in the next layer. The first layer of neurons of the neural network is the input layer. The outputs of the input layer serve as the inputs to another set of neurons known as a hidden layer. A neural network may have one or more of such hidden layers. The outputs of the hidden layer serve as the inputs of the final layer known as the output layer.

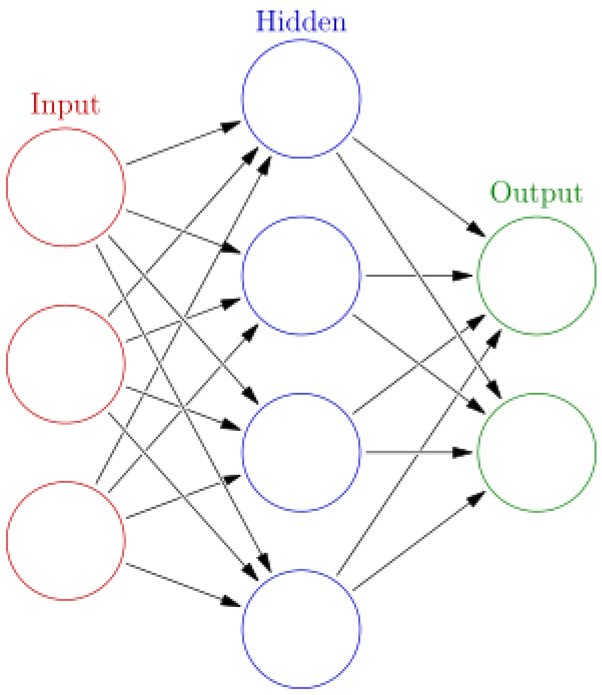

Figure 3: A typical network structure

ANNs are typically trained using the backpropagation algorithm in conjunction with an optimizer like gradient descent. When an input is provided to the network, it is calculated and propagated through the network until it produces an output after the output layer. The output of the network is compared to the desired output and the error for each neuron is calculated using a loss function. The error is then propagated backwards through the network to calculate the gradient of the loss function with respect to each neuron in the network. The optimization function is then used to adjust the weights of the network to minimize the loss function.

Convolutional neural networks

Convolutional neural networks (CNN) are a specialized type of feedforward artificial neural network, the main use of CNNs is classification and they have proven to be very useful in image recognition tasks. The name of convolutional comes from the fact that CNNs use a convolution operation instead of matrix multiplication[6]. Training in CNN is similar to the training of a regular ANN, but some modifications are made to the backpropagation algorithm to fit the network design.
The idea behind CNNs is that it shares a set of weights across the whole input set, finding the same kind of feature in different sections of the input data; this set of weights is often call a filter. To visualize this, let’s use an image as an input example for a CNN. The filter will identify a feature of section of an image, for example an horizontal edge, and searches for that particular feature in other sections of the image. In Figure 4 we can see how the filter is applied to a 5x5

section of the image and produces the output for the hidden layer. To continue with feature extraction we slide the input section by one (or more) pixel to the right and repeat the process. This is usually called the convolution stage. The result of applying the filter to image is called a feature map (the hidden layer in Figure 4). To actually achieve image recognition a set of features map is needed, one for a each filter applied to the image.

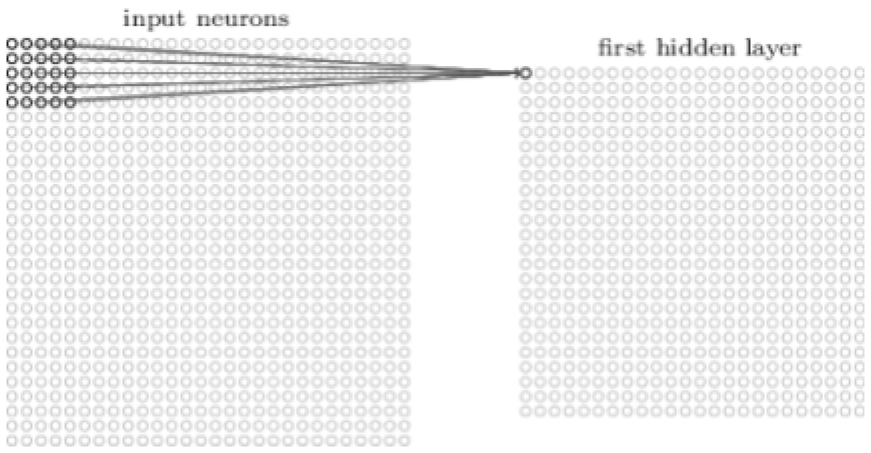

Figure 4: Applying a filter to a section of the input

CNNs also use a technique known as pooling, pooling is used to generalize the input further. Pooling takes a section of the input (similarly as the convolutional stage does) and applies a function to it, usually just extract the maximum value (max-pooling). See the image below for a graphical representation. The intuition behind pooling is that the networks just cares if certain feature is present in a region of the image, but it does not care about the exact position of the feature.

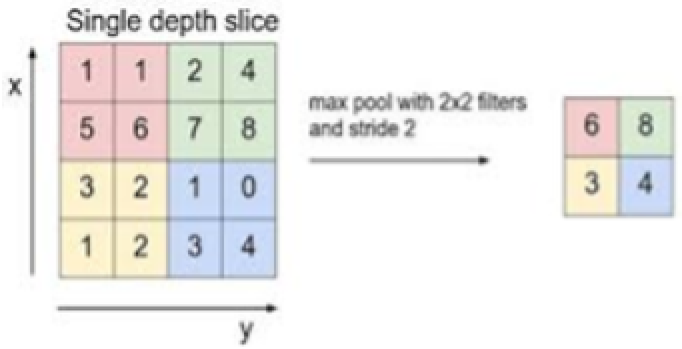

Figure 5: Max-pooling

The ideas mentioned above make up a convolutional layer. A convolutional layer starts with the convolutional stage to extract features, then the output of each convolutional stage is passed through activation function (usually a rectified linear function) to increase its nonlinear properties. Finally the data can optionally pass through a pooling stage to generalize the data. The pooling stage in some literature is considered part of the convolutional stage, in other literature it is considered a stage on its own. Below is a graphical representation of a typical convolutional layer.

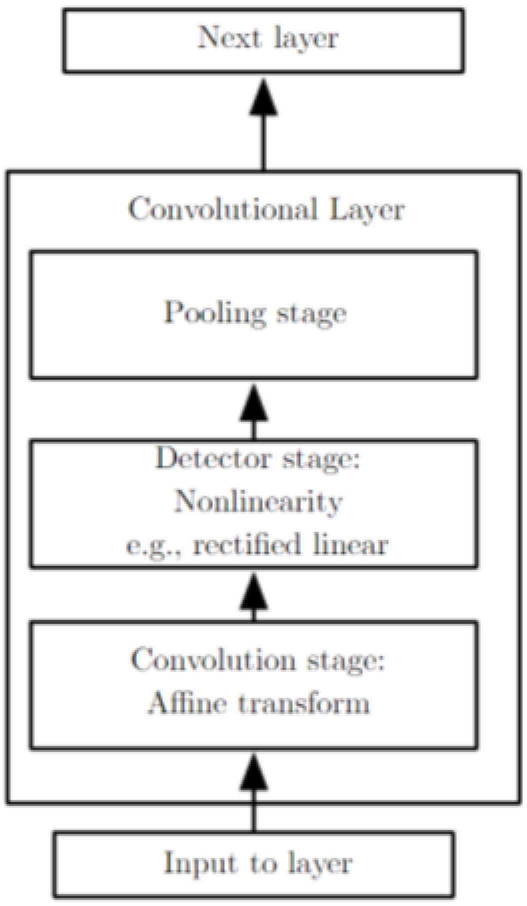

Figure 6: Structure of a convolutional layer

After specifying what a convolutional layer is, we can put together a CNN. A CNN has one or more convolutional layers, and usually the last layer of the network is a fully connected layer. Below is a diagram of CNN. From the input image after a convolutional layer the result is a set of feature maps, one for each of the filters applied to the network. Each feature map is reduced using a pooling layer, finally the input is rearranged in a linear manner and that becomes the input for the fully connected layer.

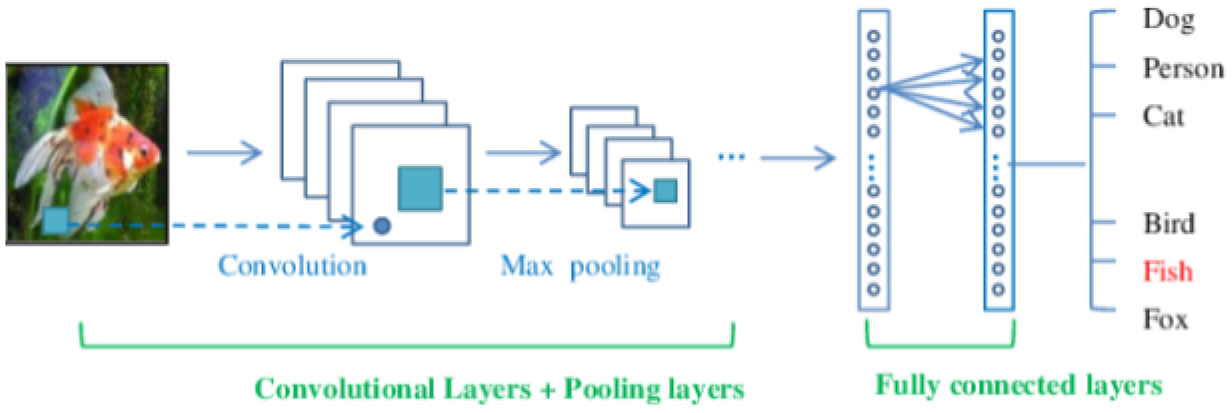

Figure 7: Structure of a convolutional neural network

Due to their accuracy in the task of image recognition, CNNs are widely used by Internet technology companies such as Google and Facebook. We have a brief introduction to two of these platforms below:

Facebook Lumos

Facebook's Lumos is a platform built for image and video understanding. It is a self-serve platform built on top of FBLearner Flow. FBLearner Flow is Facebook's internal backbone AI platform. Lumos allows teams within Facebook to harness the power of computer vision for their products and services with prior expertise. Using Lumos, Facebook has implemented the ability to search photographs by what is contained within them. This is achieved by training a deep CNN on their large dataset of photographs. After training the model, the search functionality is achieved by essentially matching the search terms to features extracted from photographs. In order avoid displaying the same images with slight changes in zoom and angles, weights are added in to prioritize diversity in the search results.

Google Tensorflow

Google Tensorflow is based on Google DistBelief which was a proprietary machine learning platform based on deep neural networks. After DistBelief gained popularity within Google, it was refactored into a more robust, application grade library and came to be known as Tensorflow. Tensorflow was released as an open source software on November 9, 2015. Tensorflow provides APIs for Python, C++, Java and Go. Computations are expressed in TensorFlow using dataflow graphs. The nodes in a dataflow graph represent mathematical operations whereas the edges represent multidimensional data (tensors) that flow between them. Due to it's open source nature, we chose to build our CNN implementation using TensorFlow.

MNIST database

To implement the CNN for image recognition we have used the MNIST database. The MNIST (Modified National Institute of Standards and Technology) database is a large database of handwritten digits. It is a popular dataset that is used very commonly to train algorithms for image recognition. The MNIST database was remixed from the original samples from the database by National Institute of Standards and Technology. The database contains 60000 samples for training of which 5000 samples are used for validation, and 10000 samples for testing. Each image in the database is a monochrome image of 28 by 28 pixels. Each sample in the database consists of data points which represent the image and a label which indicates the class of the image.

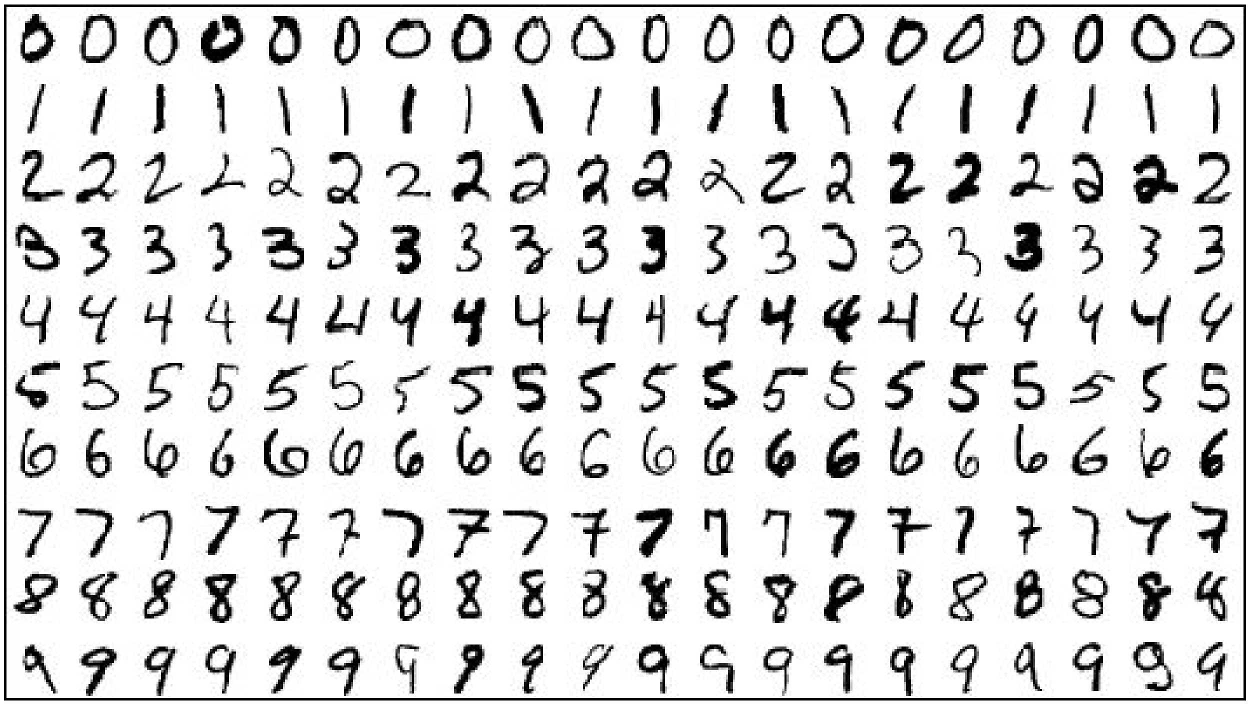

Figure 8: Sample of MNIST data

CUDA

Graphical processor units (GPU) are popular in the field of neural networks. GPU was originally made with the purposes of graphical processing, especially for video games, to perform matrix multiplication and video rendering operations in parallel. GPU has evolved to be able to perform general purpose computations. CUDA is one of the most popular GPU languages used, and Tensorflow makes use of it. Since CNNs uses the same filter in different parts of the input, the process can be parallelize. Another advantage of using GPUs over CPUs is that GPUs have a higher memory bandwidth with is useful for the large number of parameter CNNs have. Some research has shown that GPU implementations of CNNs can be up to 40 times more time efficient that CPU ones[5].

Implementation in Tensorflow

The following table shows the structure of the network used in our implementation. Using our implementation we were able to achieve an accuracy of 97% on the test images. The input and output columns shows the input and output of the layer respectively in the format(batch_size, height, width, feature_maps). In our implementation the batch size was 128, the height and width represent the dimension of the image in process, and feature_maps is the number of features maps at each stage of the network. The filter column represents the size of the filter use in each layer.

Layer	Input	Output	Filter
conv1	(batch_size, 32, 32, 1)	(batch_size, 28, 28, 32)	5*5
pool1	(batch_size, 28, 28, 32)	(batch_size, 14, 14, 32)	2*2
conv2	(batch_size, 14, 14, 32)	(batch_size, 14, 14, 64)	5*5
pool2	(batch_size, 14, 14, 64)	(batch_size, 7, 7, 64)	2*2
Flat	(batch_size, 7, 7, 64)	(batch_size, 3136)	N/A
fc1	(batch_size, 3136)	(batch_size, 1024)	N/A
fc2	(batch_size, 1024)	(batch_size, 10)	N/A

Table 1: Structure of our implementation

A TensorFlow program consists of two discrete sections:

1. Building the graph
2. Running the graph

Building the graph

We begin by initializing the input to the network and providing the training labels as placeholders:

```
# Input and output
```

```
x = tf.placeholder(tf.float32, [None, input_size])
y = tf.placeholder(tf.float32, [None, output_size])
keep = tf.placeholder(tf.float32)
```

The weights are then initialized as variables according to our network shape:

```
# Weights
# 5*5 filter, 1 input, 32 outputs
conv1_weights = tf.Variable(tf.random_normal([5, 5, 1, 32]))
# 5*5 filter, 32 input, 64 outputs
conv2_weights = tf.Variable(tf.random_normal([5, 5, 32, 64]))
# Fully connected layer, 7*7*64 inputs, 1024 outputs
fc_weights = tf.Variable(tf.random_normal([7 * 7 * 64, 1024]))
# Output
output_weights = tf.Variable(tf.random_normal([1024, output_size]))
```

We then initialize the biases as per our network shape:

```
# Biases
conv1_bias = tf.Variable(tf.random_normal([32]))
conv2_bias = tf.Variable(tf.random_normal([64]))
fc_bias = tf.Variable(tf.random_normal([1024]))
output_bias = tf.Variable(tf.random_normal([output_size]))
```

The forward flow of the network is defined as a function:

```
# Convolutional network
def convolutional_network(input_data):
    # Reshape the input picture
    input_data = tf.reshape(input_data, [-1, 28, 28, 1])

    # Convolutional layer 1
    # Convolution
    conv1_output = tf.nn.conv2d(input_data, conv1_weights, [1, 1, 1, 1], 'SAME')
    conv1_output = tf.nn.bias_add(conv1_output, conv1_bias)
    conv1_output = tf.nn.relu(conv1_output)
    # Pooling
    conv1_output = tf.nn.max_pool(conv1_output, [1, 2, 2, 1], [1, 2, 2, 1], 'SAME')

    # Convolutional layer 2
    conv2_output = tf.nn.conv2d(conv1_output, conv2_weights, [1, 1, 1, 1], 'SAME')
    conv2_output = tf.nn.bias_add(conv2_output, conv2_bias)
    conv2_output = tf.nn.relu(conv2_output)
    # Pooling
    conv2_output = tf.nn.max_pool(conv2_output, [1, 2, 2, 1], [1, 2, 2, 1], 'SAME')

    # Fully connected layer
    fc1_output = tf.reshape(conv2_output, [-1, fc_weights.get_shape().as_list()[0]])
    fc1_output = tf.add(tf.matmul(fc1_output, fc_weights), fc_bias)
    fc1_output = tf.nn.relu(fc1_output)
    fc1_output = tf.nn.dropout(fc1_output, keep)

    # Output layer
    output_output = tf.add(tf.matmul(fc1_output, output_weights), output_bias)
    return output_output
```

We then define the cost function and use an optimizer to reduce the cost function. We have used the Adam optimizer in our implementation:

```
# Define cost function
# Softmax activation on output layer and calculate cross entropy. Then reduce mean
cost_function = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
# Define optimizer
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost_function)
```

After our graph structure has been initialized, we initialize all the variables:

```
# Initialize variables
init = tf.global_variables_initializer()
```

Running the graph
With the network graph initialized, we can now run the graph.

```
with tf.Session() as sess:
    # initialize session
    sess.run(init)

    step = 1
    # We have a batch size of 128
    while step * batch_size < epochs:
        batch_x, batch_y = mnist_data.train.next_batch(batch_size)
        sess.run(optimizer, {x: batch_x, y: batch_y, keep: dropout})

        if step % display_step == 0:
            loss, acc = sess.run([cost_function, accuracy], {x: batch_x, y: batch_y, keep:
dropout})
            print("Iteration: ", step * batch_size, " Loss: ", loss, " Accuracy: ", acc * 100)
        step += 1
    # Test accuracy on the test images
    print(sess.run(accuracy, {x: mnist_data.test.images[:256], y:
mnist_data.test.labels[:256], keep: 1}))
```

References

[1] https://techcrunch.com/2017/02/02/facebooks-ai-unlocks-the-ability-to-search-photos-by-whats-in-them/
[2] http://colah.github.io/posts/2014-07-Conv-Nets-Modular/
[3] https://www.tensorflow.org/tutorials/image_recognition
[4] https://code.facebook.com/posts/1259786714075766/building-scalable-systems-to-understand-content/

[5] Machine Learning, Tom Mitchell, McGraw Hill, 1997
[6] https://devblogs.nvidia.com/parallelforall/cuda-spotlight-gpu-accelerated-deep-neural-networks/
[7] http://www.deeplearningbook.org/contents/convnets.html