Deep Learning Image recognition with TensorFlow

Băia Alina-Elena

Deep learning

- is a branch of **machine learning** based on set of **algorithms** that attempt to model high level abstractions in data

- it is based on **feature learning** (a set of techniques that learn a feature) and allows a machine to both learn a specific task or a specific feature.

Deep learning architectures in computer vision

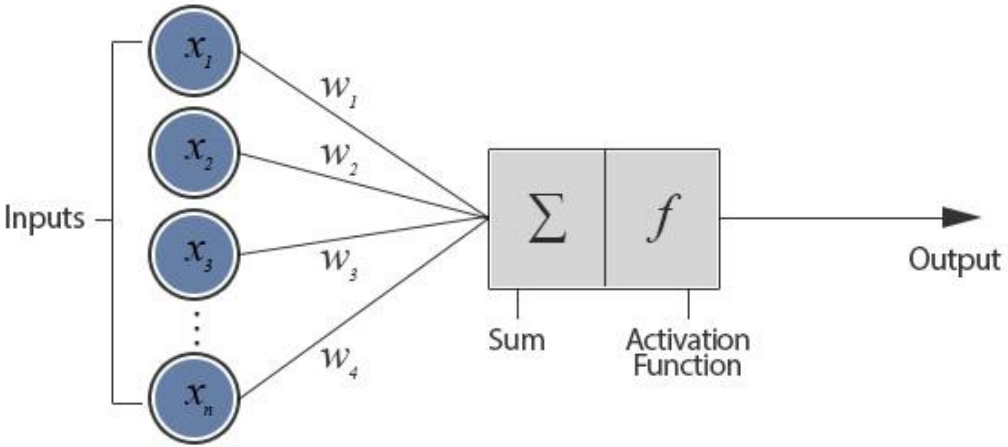
- Deep neural networks
- Convolutional deep neural networks
- Deep belief networks
- Recurrent neural networks

Artificial neural network

Is a computational approach designed to solve any problem by trying to mimic the structure and the function of human nervous system.

Neural networks are based on simulated neurons, which are joined together in a variety of ways to form networks.

Artificial neuron model

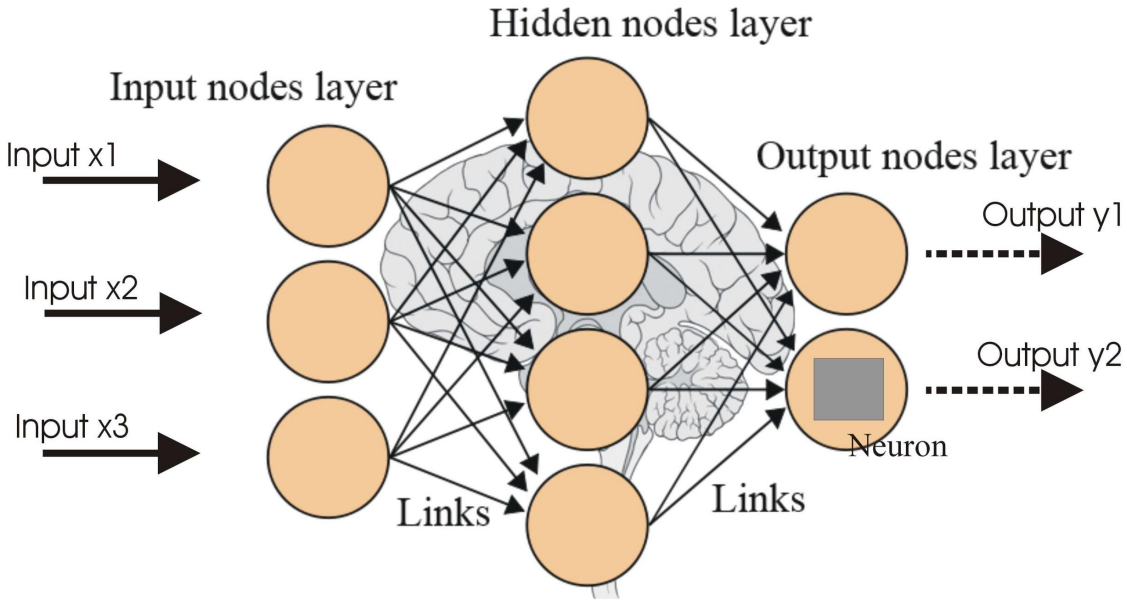


- inputs to the network are represented by the mathematical symbol, x_n
- each of these inputs are multiplied by a connection weight, w_n

  Sum = $w_1 x_1 + \ldots + w_n x_n$

- activation function **f(Sum)**

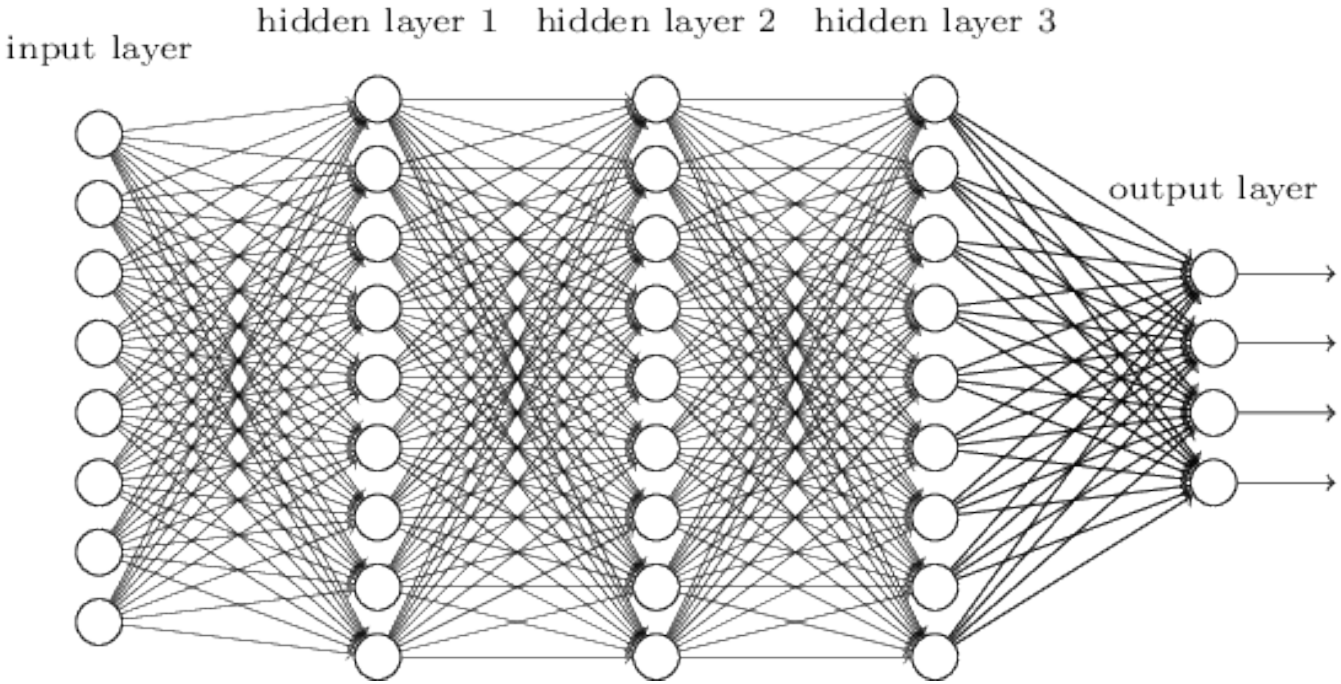
Artificial neural network model



An ANN is typically defined by three types of parameters:

1. The interconnection pattern between the different layers of neurons
2. The learning process for updating the weights of the interconnections
3. The activation function that converts a neuron's weighted input to its output activation.

Deep neural network



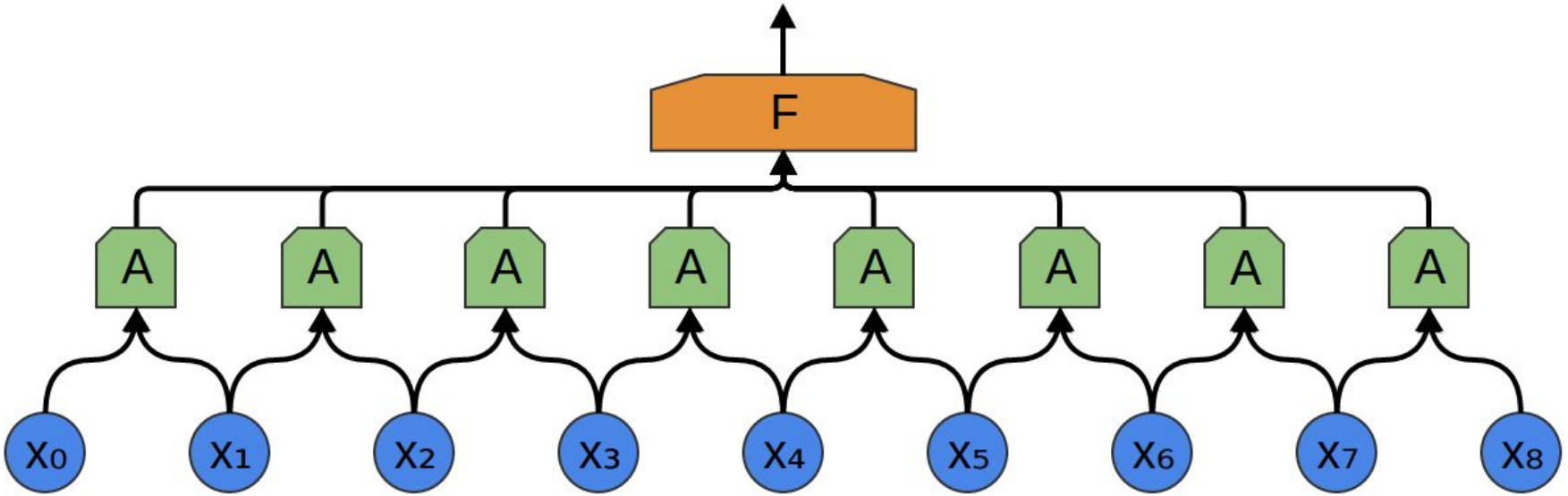
A *deep neural network* (DNN) is an artificial neural network with multiple hidden layers of units between the input and output layers.

Convolutional Neural networks (CNN)

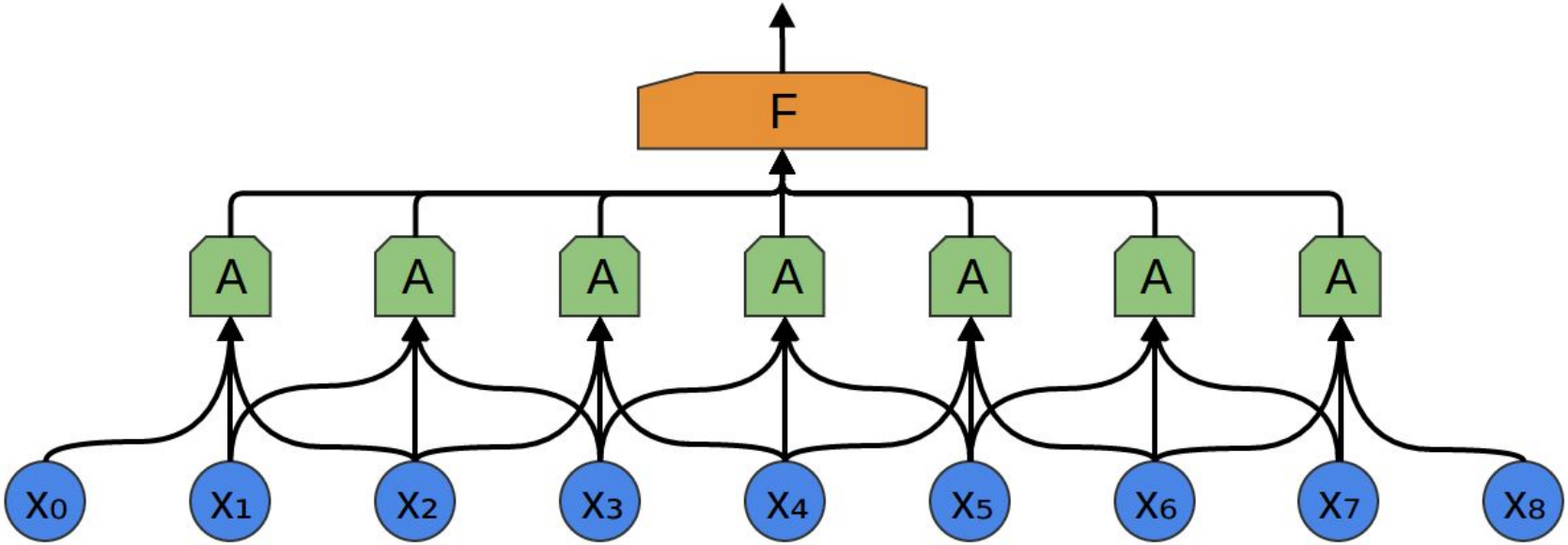
CNN are made up of neurons that have **learnable** weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the **raw image** pixels on one end to **class** scores at the other.

CNN architectures make the explicit assumption that the inputs are images.

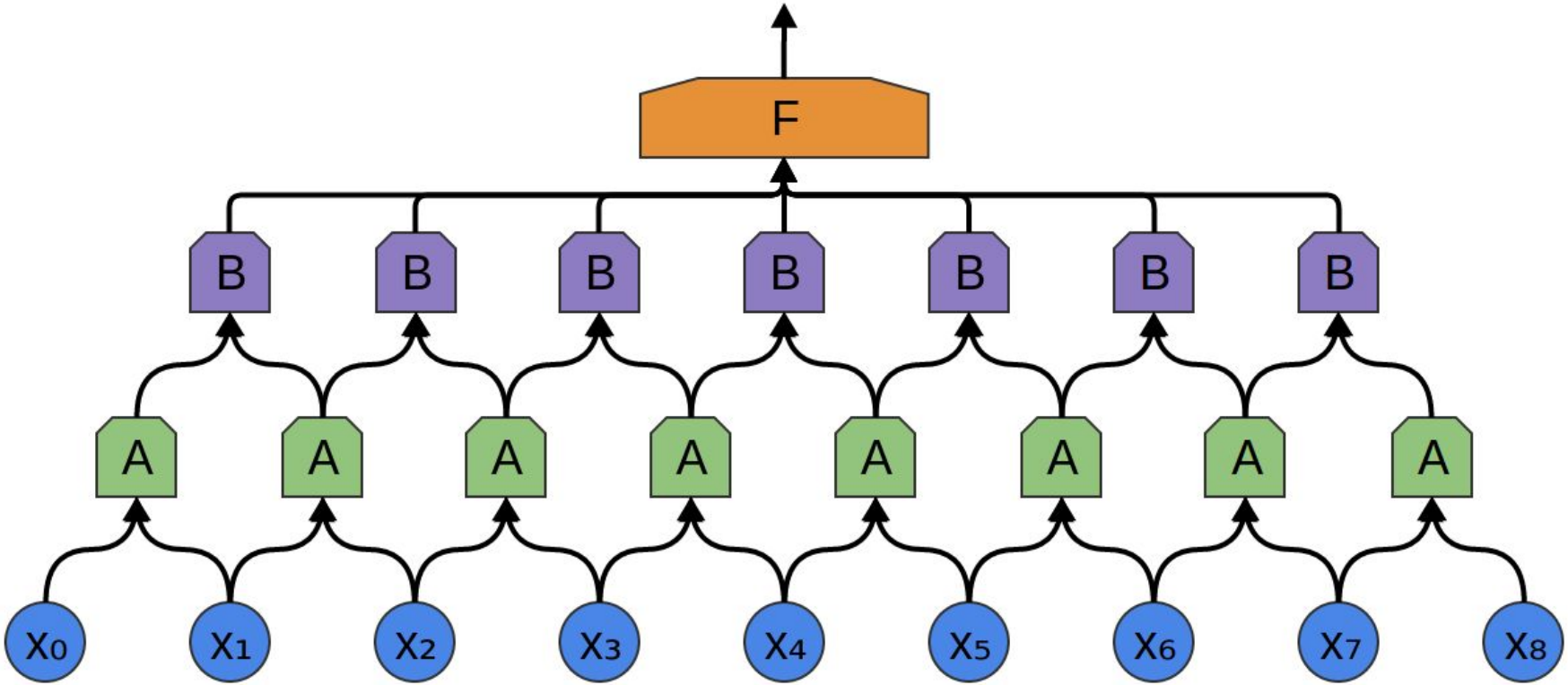
Structure of CNN



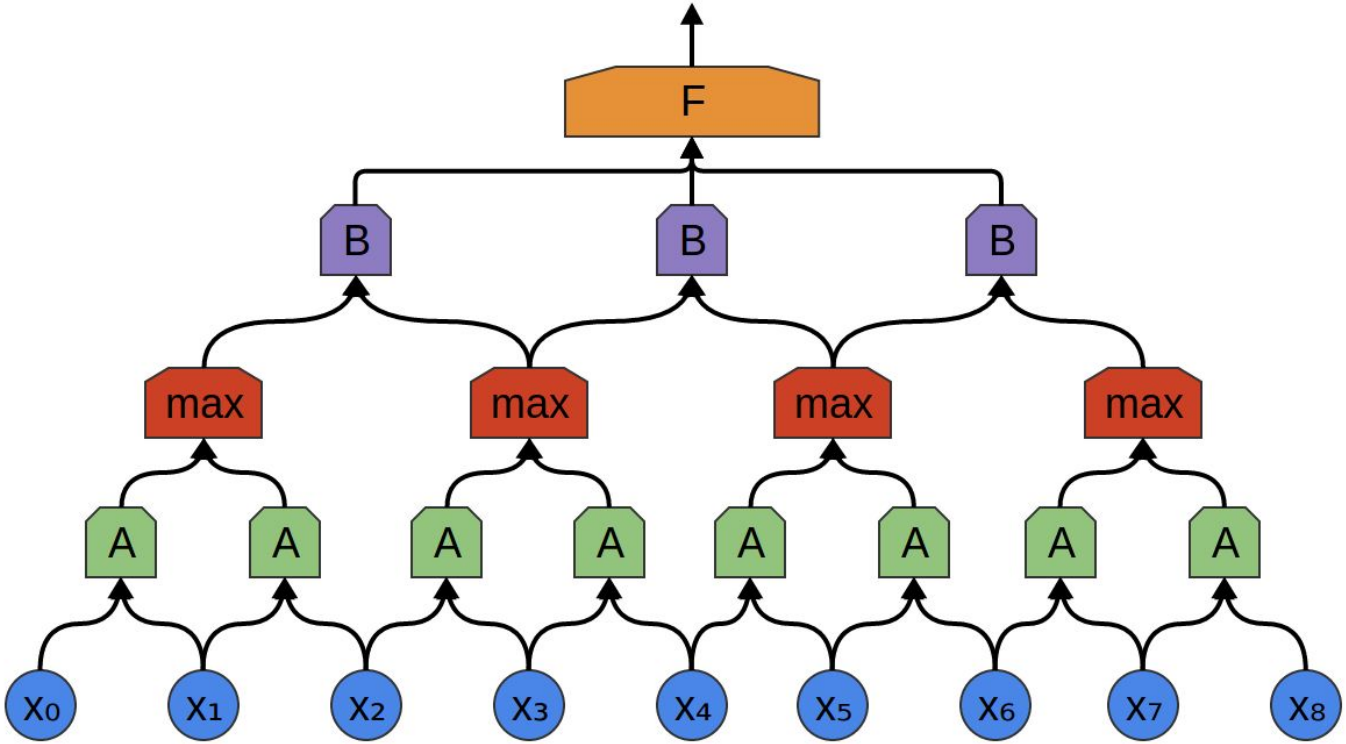
Structure of CNN



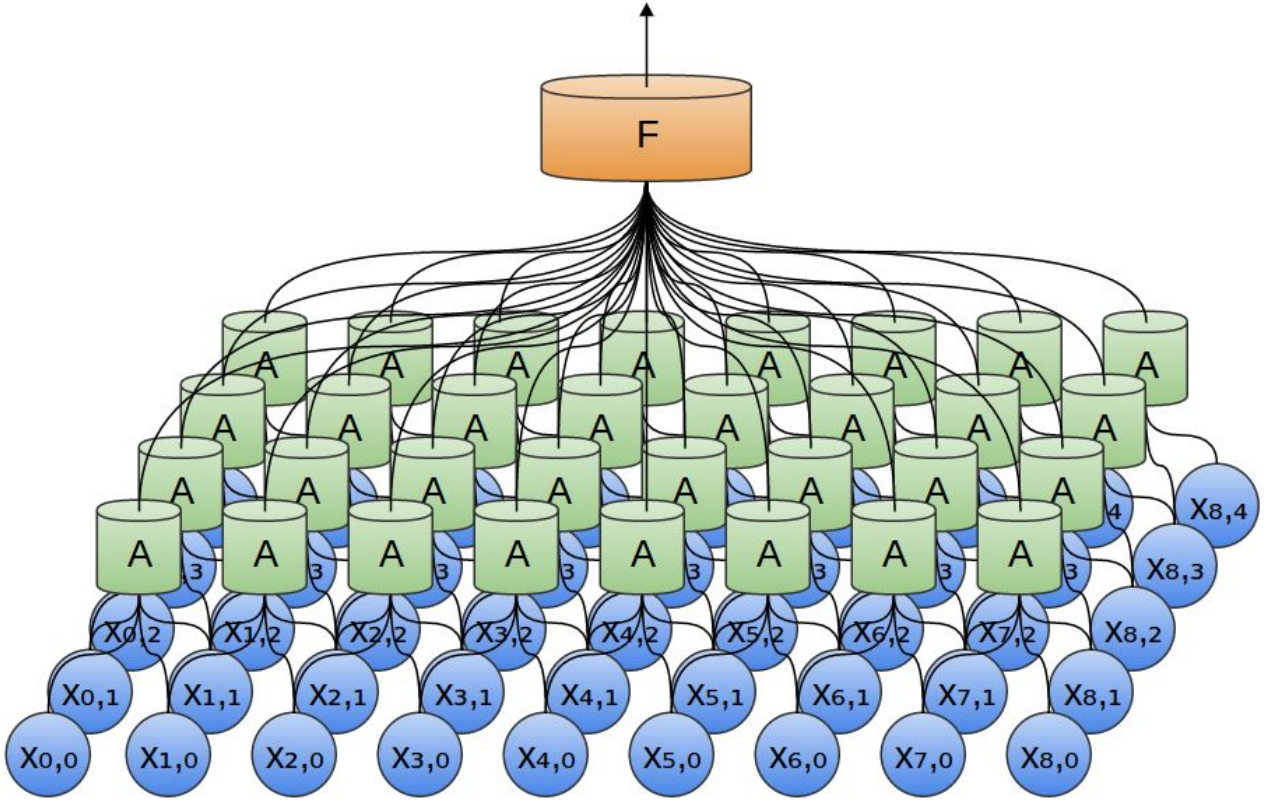
Structure of CNN



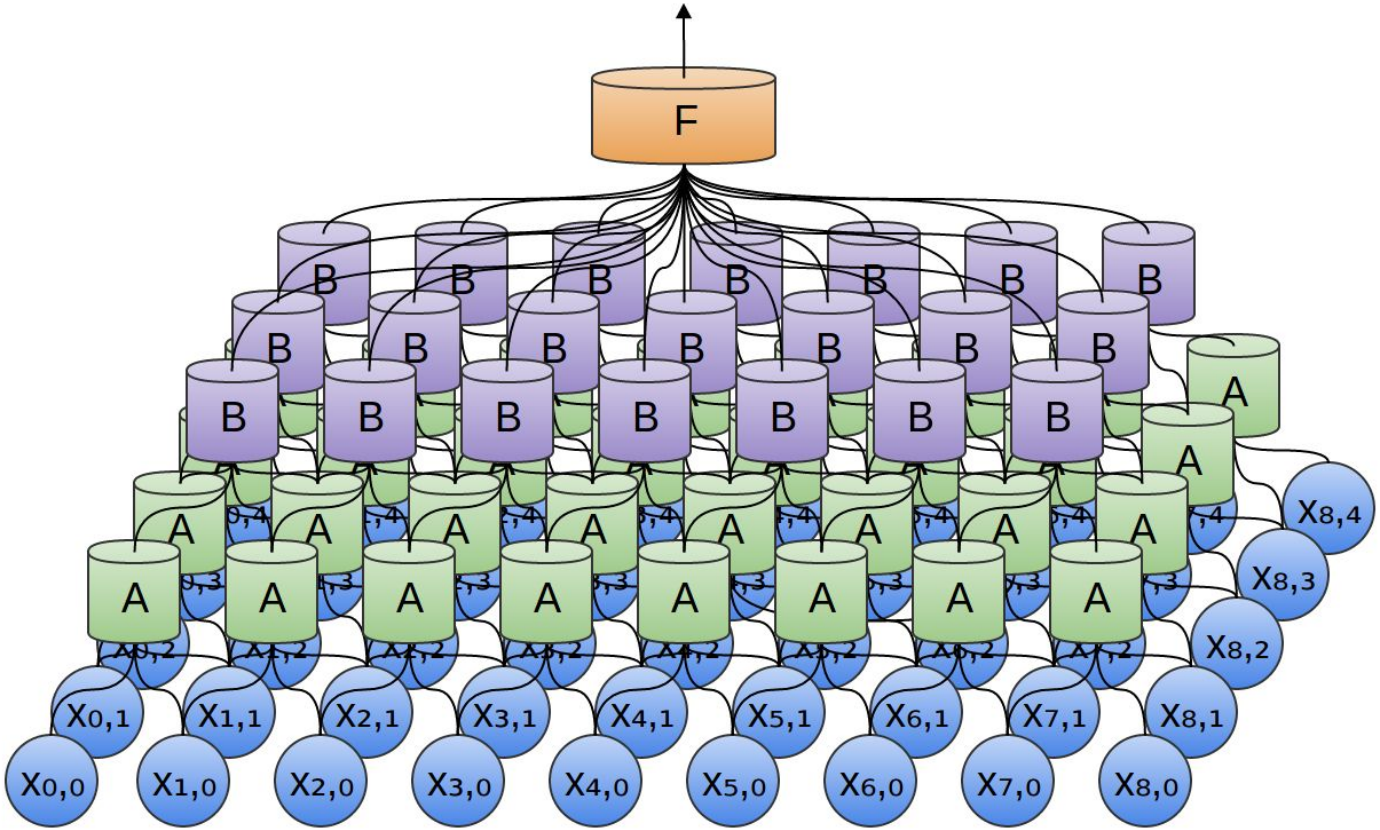
Structure of CNN



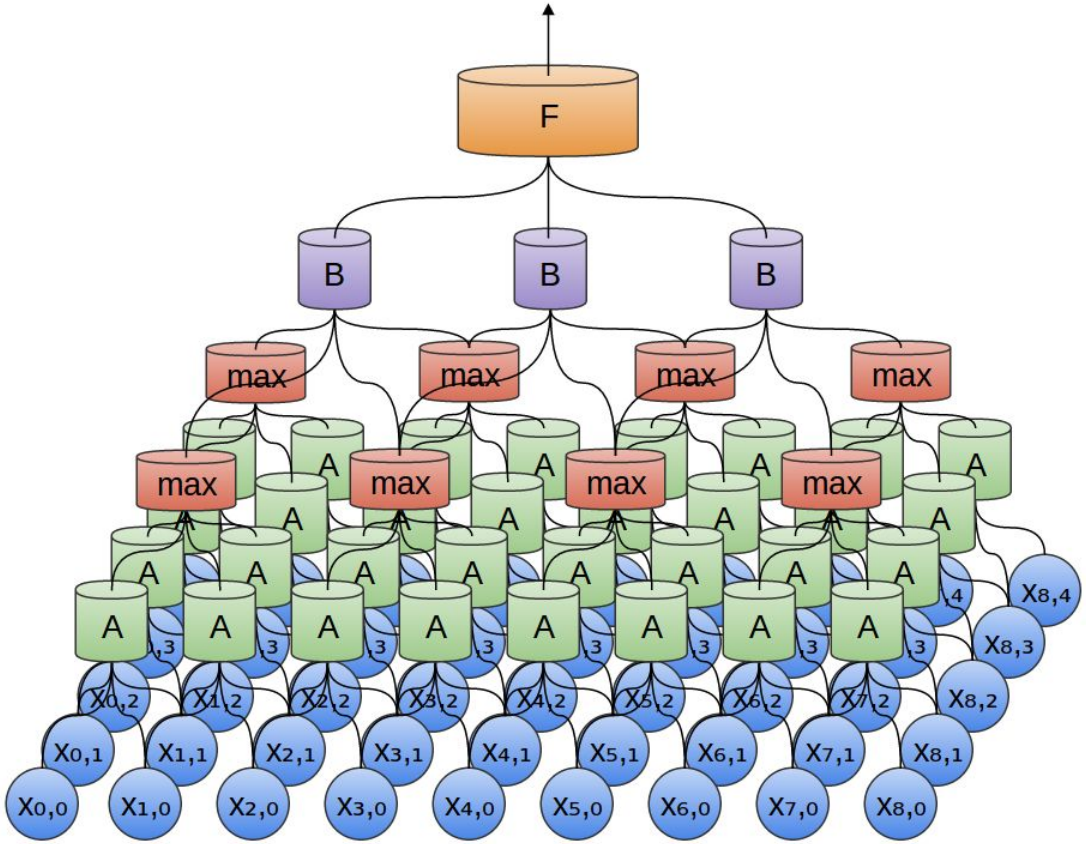
Structure of CNN



Structure of CNN



Structure of CNN



Layers used to build CNN

Convolutional Layer

ReLU Layer

Pooling Layer

Fully-Connected Layer

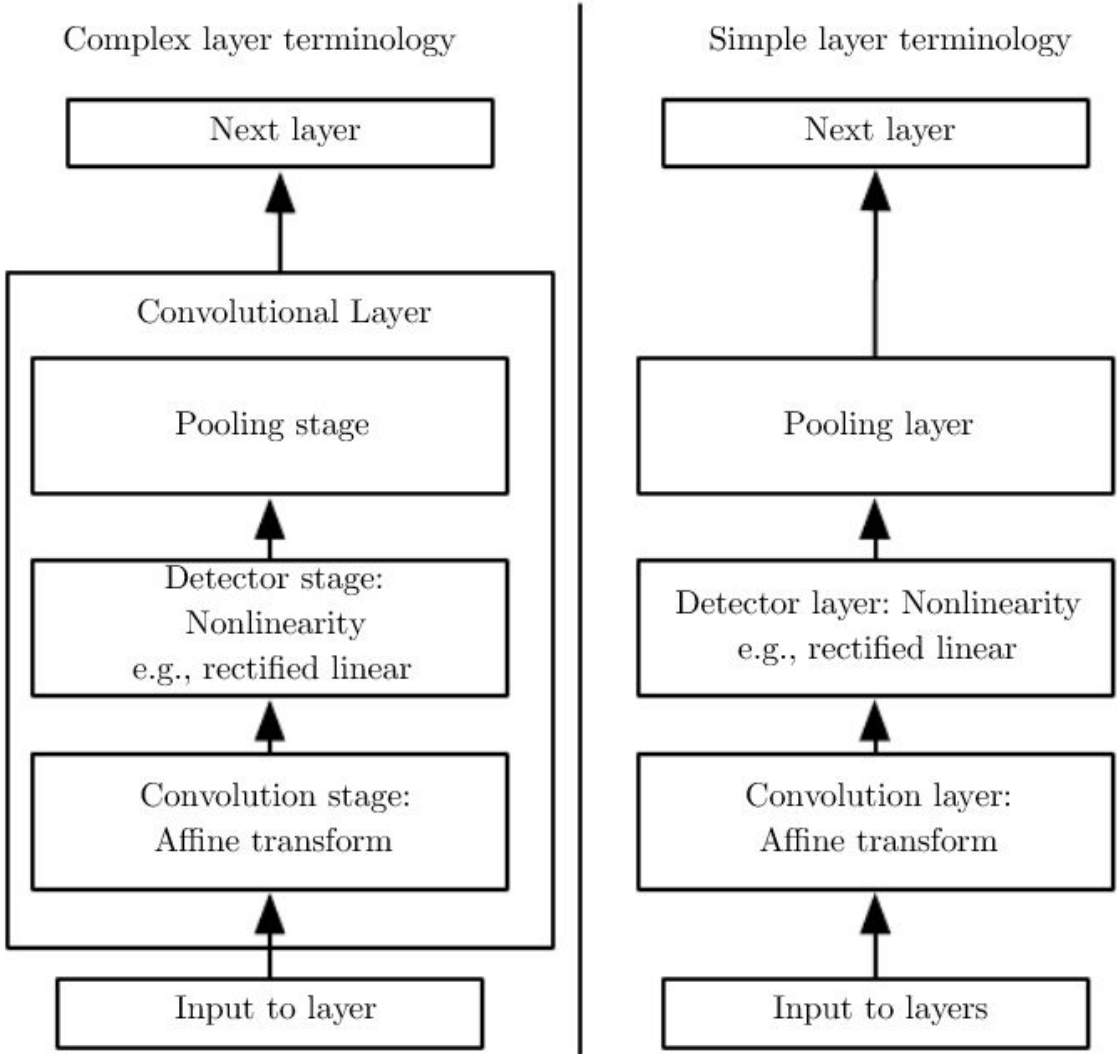
Four main operations in the CNN

Convolution

Non Linearity (ReLU)

Pooling (subsampling)

Classification


Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage:
Nonlinearity
e.g., rectified linear
Convolution stage:
Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity
e.g., rectified linear
Convolution layer:
Affine transform
Input to layers

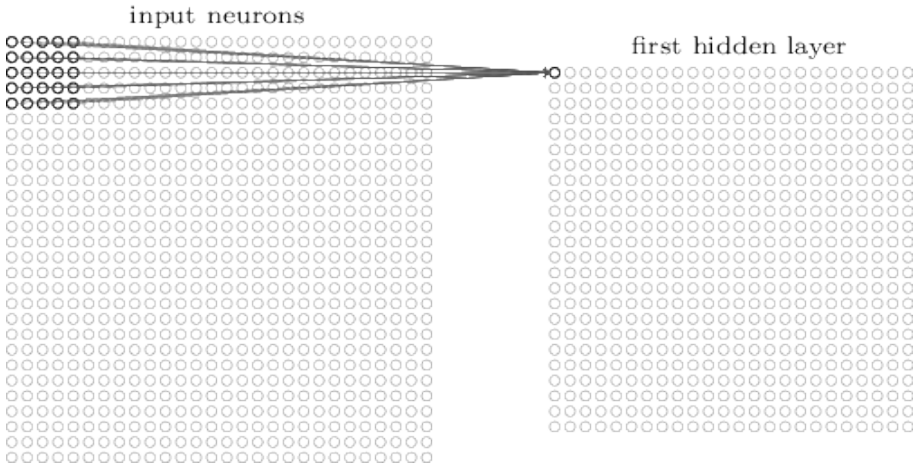
The Convolutional step

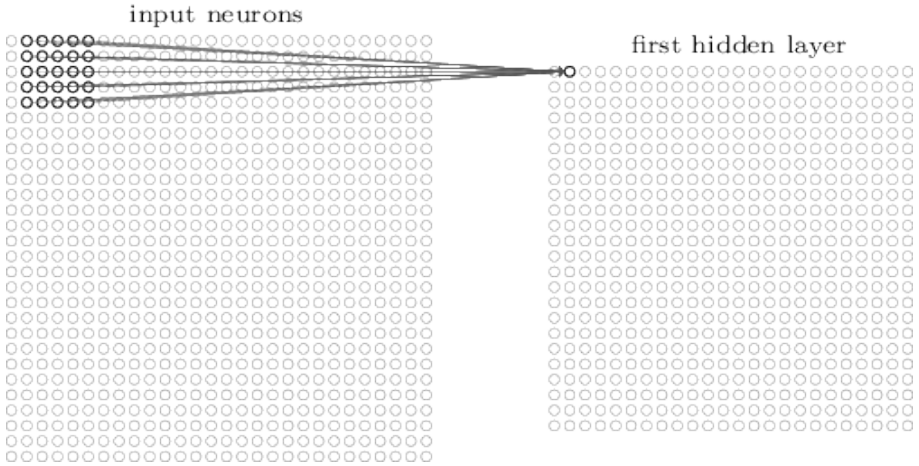
The primary purpose of Convolution in case of a CNN is to **extract features** from the input image.

Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

Basic ideas:

- local receptive fields and stride length
- shared weights and bias (kernel or filter)





Local Receptive Fields

Example:
28 x 28 pixels image
5 x 5 region => 25 input pixels
stride length = 1

Result : 24 x 24 neurons in the hidden player

for the *j*,*k* th hidden neuron, the output is:

$$\sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\right)$$

Where σ is the activation function , *b* is the shared bias, *w* is the array of shared weights, *a* denotes the input activation

Shared weights and bias

In our Example each hidden neuron has a bias and 5 x 5 weights connected to its local receptive field.

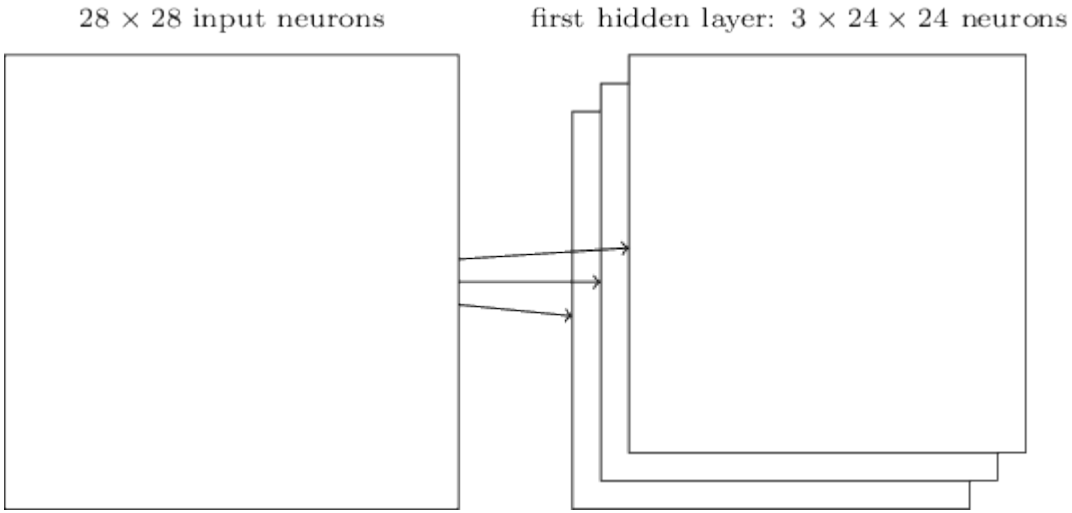
We are using the same weights and bias for each of the 24 x 24 hidden neurons.

This means that all the neurons in the first hidden layer detect exactly the same feature, just at different locations in the input image.

We obtain a feature map.

Feature map

To do image recognition we'll need more than one feature map.
And so a complete convolutional layer consists of several different feature maps:



Convolutional Layer

Each feature map : 5 x 5 = 25 shared weights
One single shared bias.
So each feature map requires 26 parameters.
If we have 20 feature maps that's a total of 20×26=520 parameters defining the convolutional layer.

Fully connected Layer

Imagine we have a fully connected layer with 28 x 28 = 784 input neurons, and a relatively modest 30 hidden neurons. That's a total of 784×30 weights, plus an extra 30 biases, for a total of 23,550 parameters.

40 times more parameters!

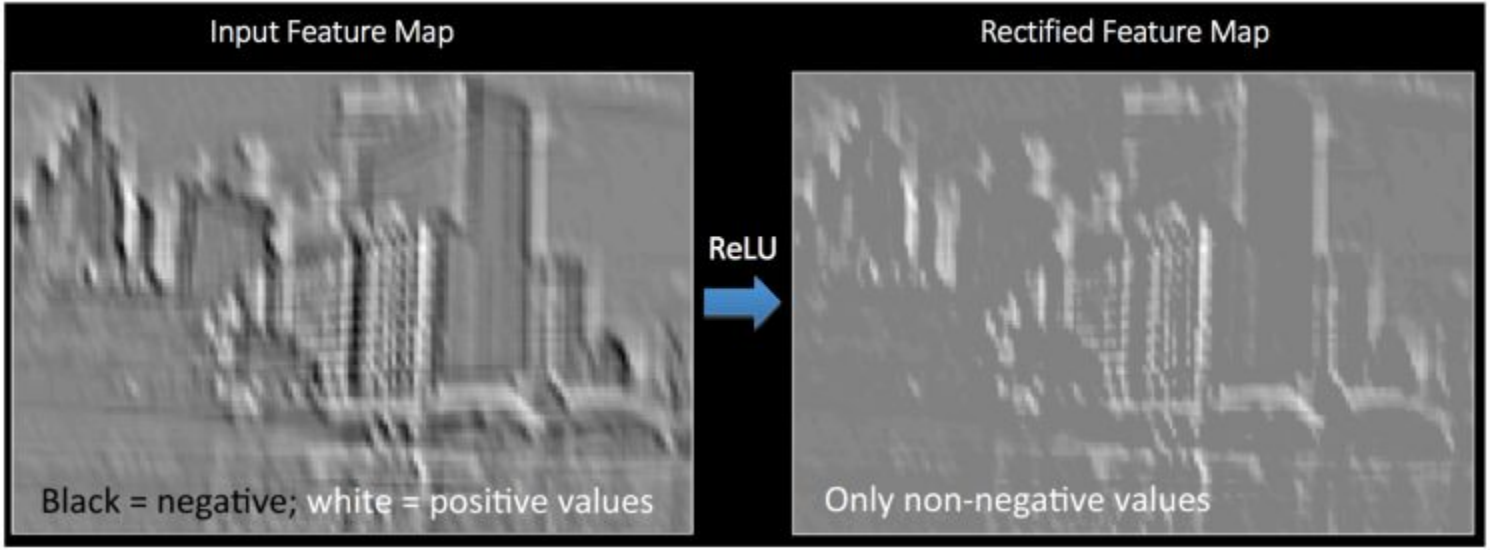
ReLU (Rectified Linear Unit)

$$f(x) = Max(0, x)$$

ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our CNN, since most of the real-world data we would want our CNN to learn would be non-linear.

(Convolution is a linear operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

ReLU (Rectified Linear Unit)



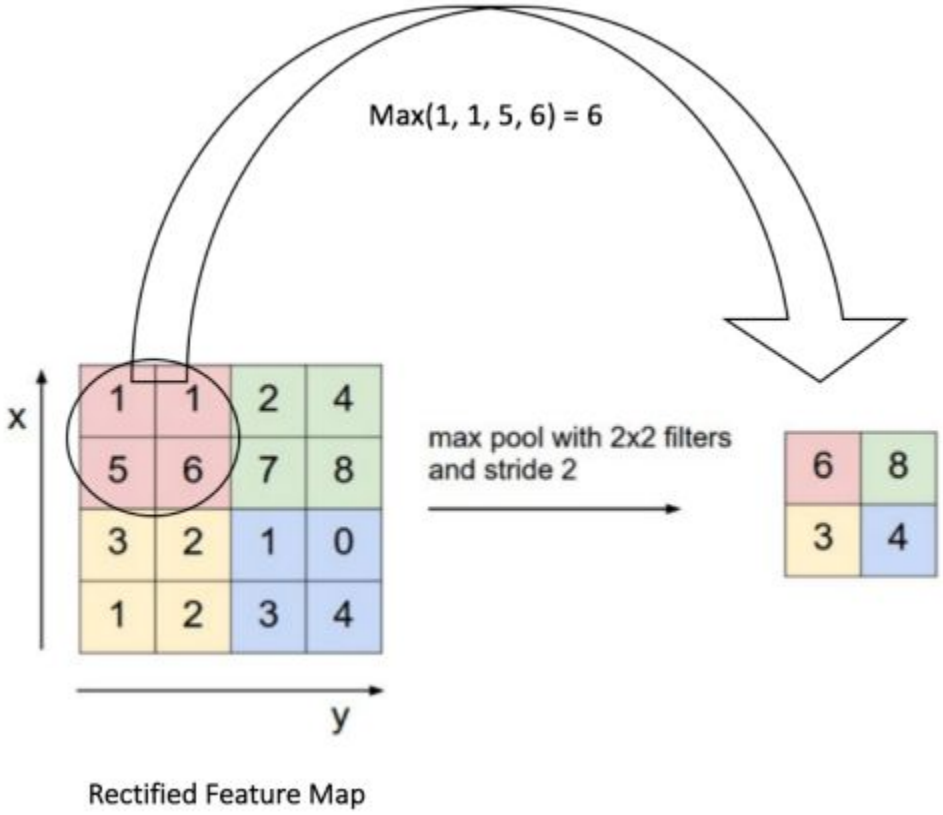
The Pooling step

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information.

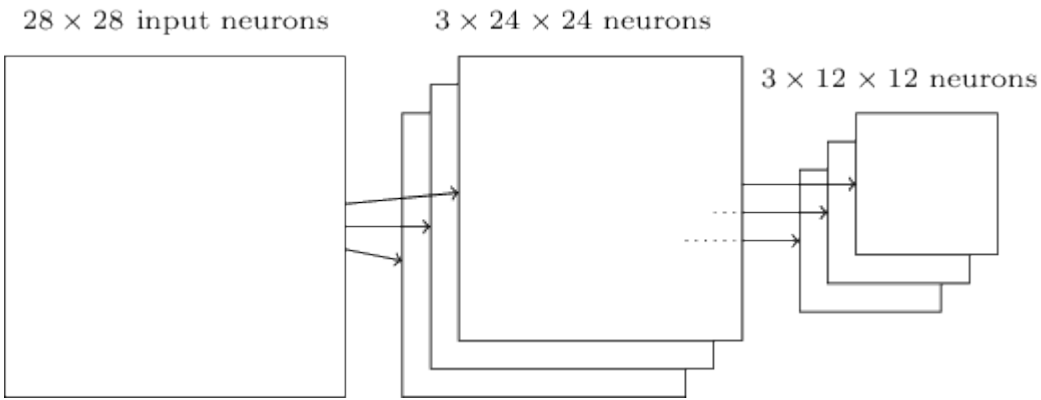
A pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map.

Spatial Pooling can be of different types: Max, Average, Sum etc.

Max Pooling


Max(1, 1, 5, 6) = 6
x
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
y
max pool with 2x2 filters
and stride 2
6 8
3 4
Rectified Feature Map

Max Pooling for each feature map



Fully connected layer

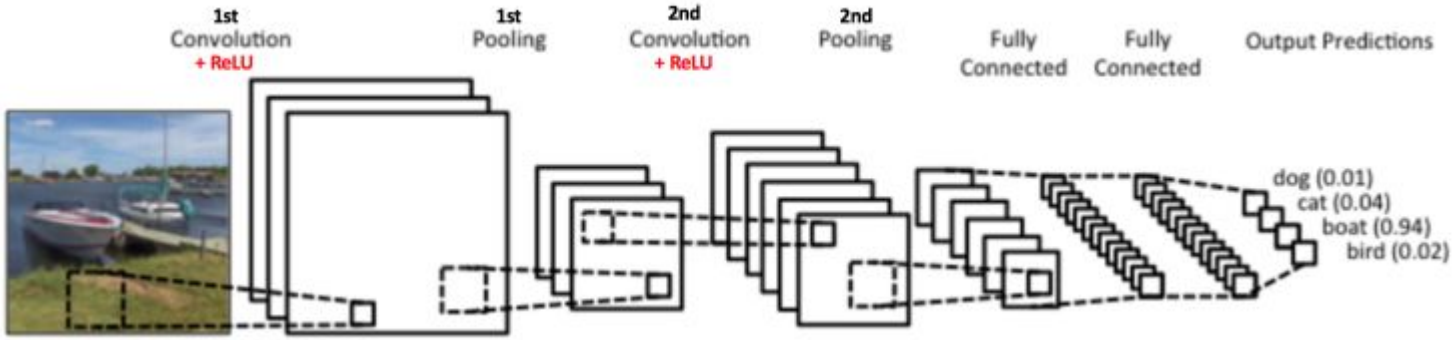
The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.

The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

The Fully Connected layer uses a softmax activation function in the output layer.

The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one.

Putting it all together



The overall training process of the Convolution Network may be summarized as below:

Step1: We initialize all filters and parameters / weights with random values
Step2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
Step3: Calculate the total error at the output layer
Step4: Use Backpropagation to update all filter values / weights and parameter values to minimize the output error.

The weights are adjusted in proportion to their contribution to the total error.

Step5: Repeat steps 2-4 with all images in the training set.

Backpropagation Algorithm:

Initially all the edge weights are randomly assigned. For every input in the training dataset, the CNN is activated and its output is observed. This output is compared with the desired output that we already know, and the error is “propagated” back to the previous layer. This error is noted and the weights are “adjusted” accordingly.

This process is repeated until the output error is below a predetermined threshold.

TensorFlow

TensorFlow is an *open source* software library for numerical computation using *data flow* **graphs**.

Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (**tensors**) communicated between them.

TensorFlow
Features

- **Flexibility**: if you can express your computation as a data flow graph, you can use TensorFlow
- **Portability**: desktop, server, mobile
- **More languages**: Python, C++, Java, Go…
- **Performance**: CPU/GPU, threads, queues, asynchronous computation
- **Community:** GitHub repository and StackOverflow channel

TensorFlow
Hands-on

Steps to follow:

- Install TensorFlow
- Get the dataset (COIL-20: https://goo.gl/cSWyPb)
- Write a dataset reader
- Create CNN model
- Train the model
- Test the results

Install TensorFlow

We used TensorFlow on Windows 10.

Windows is the most recent added platform and for the moment TensorFlow supports only 64-bit Python 3.5.

Once you have a VM Python 3.5 installed on your system the installation of the framework is a simple line:

```
C:\> pip install --upgrade
https://storage.googleapis.com/tensorflow/windows/cpu/
tensorflow-0.12.1-cp35-cp35m-win_amd64.whl
```

Get the dataset COIL-20

COIL-20 is a collection of images (*128x128*) from *20* objects made by Columbia University.

Is available online as a compressed file to download.

Write a dataset reader

We cannot process directly the images organized in that way, because is a simple folder that contains all the images.

The dataset reader deals with these points:

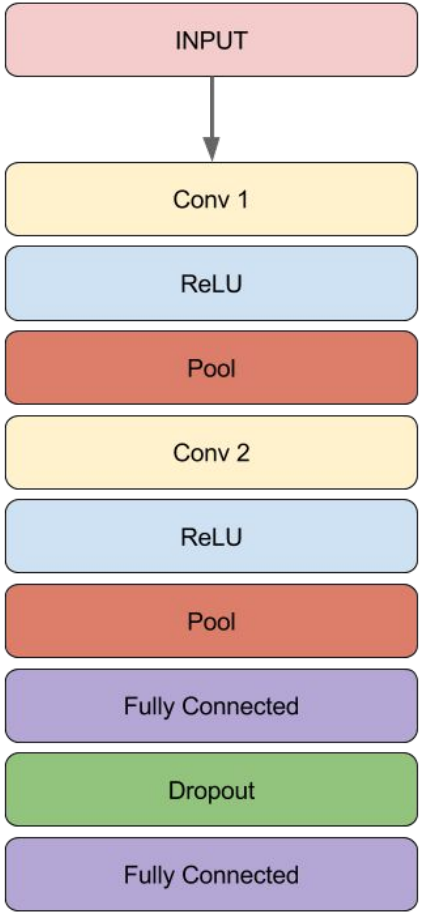
- Read all the images inside the folder
- Convert them to **numpy array** (to match the TensorFlow tensors structure)
- Split the dataset in **Train set** and **Test set**
- Store the objects associated with each image (labeling or class memorization)
- Export the data in **binary files** ready to be processed by TensorFlow

Create CNN model

The model is created and used inside the program like an object.

We have some support functions that help us during the creation but all the layers are already implemented in TensorFlow, we only need to use them.

Particularly attention must to be given to the layer sizes.



Create CNN model

Code to create the layers:

```
# Conv 1 + ReLU + Pool
model.W_conv1 = weight_variable([5, 5, 1, 32])
model.b_conv1 = bias_variable([32])
model.x_image = tf.reshape(model.x, [-1, 128, 128, 1])
model.h_conv1 = tf.nn.relu(conv2d(model.x_image,
                                  model.W_conv1) + model.b_conv1)
model.h_pool1 = max_pool_2x2(model.h_conv1)

# Conv 2 + ReLU + Pool
model.W_conv2 = weight_variable([5, 5, 32, 64])
model.b_conv2 = bias_variable([64])
model.h_conv2 = tf.nn.relu(conv2d(model.h_pool1,
                                  model.W_conv2) + model.b_conv2)
model.h_pool2 = max_pool_2x2(model.h_conv2)
```

Create CNN model

Code to create the layers:

```
# Fully connected 1 + Dropout
model.W_fc1 = weight_variable([32 * 32 * 64, 1024])
model.b_fc1 = bias_variable([1024])
model.h_pool2_flat = tf.reshape(model.h_pool2, [-1,
                                                32*32*64])
model.h_fc1 = tf.nn.relu(tf.matmul(model.h_pool2_flat,
                                   model.W_fc1) + model.b_fc1)
model.keep_prob = tf.placeholder(tf.float32)
model.h_fc1_drop = tf.nn.dropout(model.h_fc1,
                                 model.keep_prob)

# Fully connected 2
model.W_fc2 = weight_variable([1024, 20])
model.b_fc2 = bias_variable([20])
model.y_conv = tf.matmul(model.h_fc1_drop, model.W_fc2)
                         + model.b_fc2)
```

Train the model

To train the model we define the train step operation and we will use it inside a batch cycle. Each batch has size 50, so it will process 50 images in a single step of the training.

```
model.cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(
        model.y_conv, model.y_))
model.train_step = tf.train.AdamOptimizer(
    1e-4).minimize(model.cross_entropy)
```

After the training the graph state is saved for future uses with a special object available in TensorFlow.

```
saver.save(sess, "models/coil.chk")
```

Test the results

The test is made on images that the model did not see during the training phase.

Those images are extracted from the dataset reader that took 12 random pictures for each object (class) to test later the success.

The model is not bullet proof but we reached an 85% of accuracy with not much effort and using a small dataset.

```
model.correct_prediction = tf.equal(
    tf.argmax(model.y_conv, 1),
        tf.argmax(model.y_, 1))
model.accuracy = tf.reduce_mean(
    tf.cast(model.correct_prediction, tf.float32))
```

From script to UI

We created a simple and useful User Interface to interact with our model.

The program uses PyQt5 and Matplotlib, along with the support of TensorFlow to improve the user experience with a direct access to the model created before.

COIL tester program