IBM Research Data Science Course

srihari sridharan

Agenda

Neural Nets (Final Bits)

- Regularization (L1, L2)
- Dropout
- Miscellaneous improvements (Data Preprocessing)
- Convolutional Neural Network

Derivative of Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Derivative of Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \\
&= \frac{d}{dx}\left(1+\mathrm{e}^{-x}\right)^{-1} \\
&= -(1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x})-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x) \cdot (1-\sigma(x))
\end{aligned}$$

Loss Function

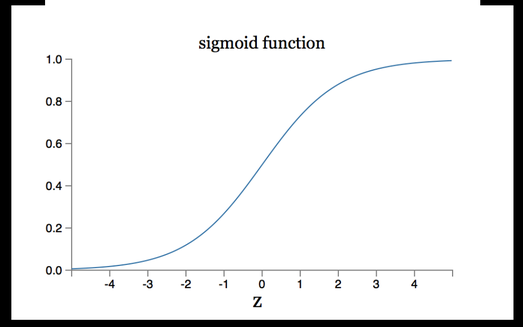
Quadratic Loss function

$$\text{Loss} = \sum \frac{1}{2}(\texttt{target} - \texttt{output})^2$$

z = wx + b
a = σ(z)

$$C = \frac{(y-a)^2}{2}$$

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$
$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$



Cross-entropy Loss function

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\frac{\partial \sigma}{\partial w_j}$$
$$= -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\sigma'(z)x_j.$$

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z)-y).$$

Loss Function

Quadratic Loss function

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$
$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$

Results in slow down

Find a Loss function without that slowdown.

Eliminate that term and let's see

$$\frac{\partial C}{\partial w_j} = x_j(a-y)$$
$$\frac{\partial C}{\partial b} = (a-y).$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a}\sigma'(z).$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a}a(1-a).$$

$$\frac{\partial C}{\partial a} = \frac{a-y}{a(1-a)}.$$

Loss Function

Quadratic Loss function

$$\frac{\partial C}{\partial w} = (a - y)\sigma'(z)x = a\sigma'(z)$$
$$\frac{\partial C}{\partial b} = (a - y)\sigma'(z) = a\sigma'(z),$$

Results in slow down

Find a Loss function without that slowdown.

Eliminate that term and let's see

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a}\sigma'(z).$$
$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a}a(1 - a).$$
$$\frac{\partial C}{\partial a} = \frac{a - y}{a(1 - a)}.$$

Integrating with respect to a

$$C = -[y \ln a + (1 - y) \ln(1 - a)] + \text{constant},$$

Improving Neural Networks (Regularisation)

- Several *W* possible for a model (Several *W* to predict input)
- Encode some preference for a certain set of weights W
- Preventing overfitting of data.
- Reduce complexity of weights during ‘learning’/training of data.
- Keeping the weights constrained
- Common ways : L1/L2 Regularization, Dropouts

$$\text{Total Loss} = \text{Data Loss} + \lambda R(W)$$

Regularisation Loss

Regularisation Strength

Improving Neural Networks (Regularisation)

L2 Regularisation (Ridge Regression)

- Penalising the squared magnitude of all parameters in the Loss

$$\text{Total Loss} = \text{Data Loss} + (1/2)*\lambda*W^2$$

To keep the gradient λW and not $2\lambda W$

- Penalising peaky weight vectors and preferring diffuse weight vectors
- Choose your λ wisely. Too small values : Negligible effect ; Too large values : Will set your weights to 0

```
from keras import regularizers
model.add(Dense(600, input_dim=784, kernel_regularizer=regularizers.l2(0.01)))
```

Improving Neural Networks (Regularisation)

L1 Regularisation (Lasso Regression)

$$\text{Total Loss} = \text{Data Loss} + \lambda * W$$

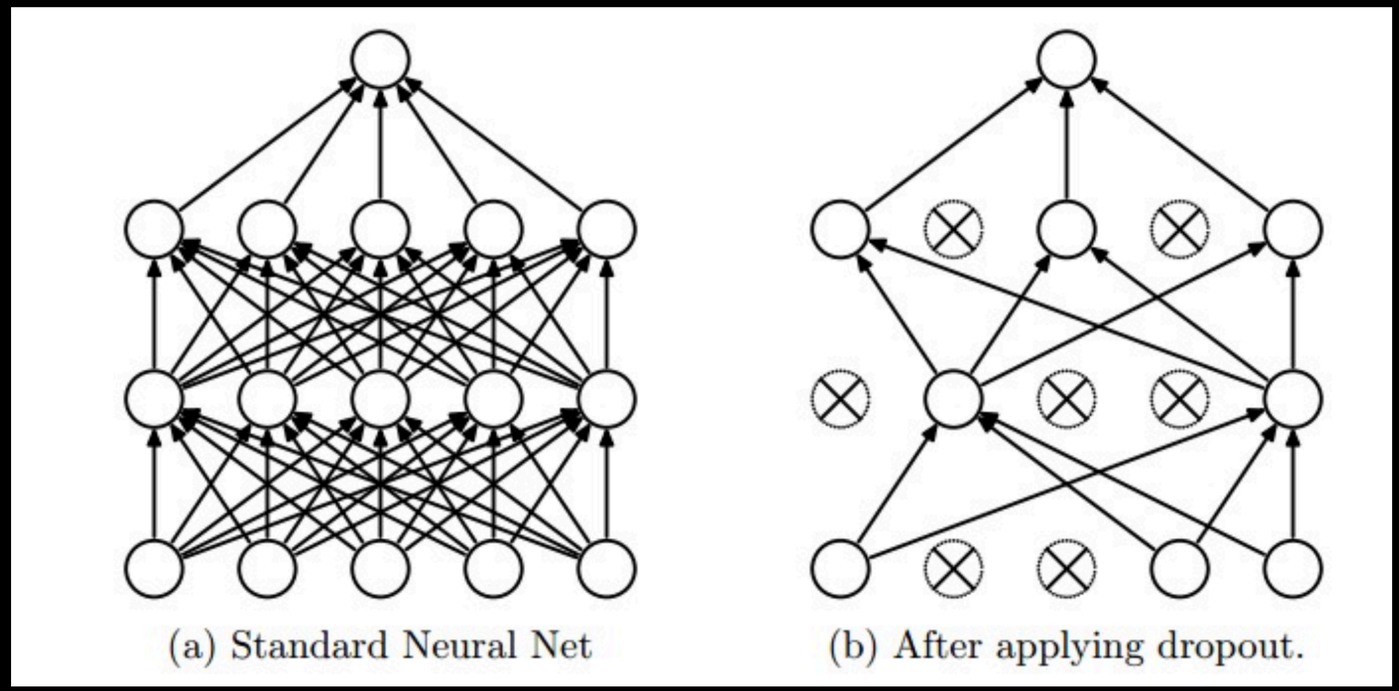
- Weight vectors become sparse during optimization.
- Useful when you have a large number of features

```
from keras import regularizers
model.add(Dense(600, input_dim=784, kernel_regularizer=regularizers.l2(0.01)))
```

- The key difference between these techniques is that **Lasso shrinks the less important feature's coefficient** to zero thus, removing some feature altogether.
- Works well for **feature selection** in case we have a huge number of features.

Improving Neural Networks (Dropouts)

- Extremely effective, simple and recently introduced regularization technique.
- Keeping a neuron active with some probability *p,* otherwise set it to 0.



// Srivastava et. al Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014

Improving Neural Networks (Dropouts)

- Training Process

```
p = 0.5 #probability
def train_step(X):
  H1 = np.dot(W1, X) + b1
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.dot(W2, H1) + b2
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3
  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

- During prediction

```
def predict(X):
  # ensembled forward pass
  H1 = np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

- We use all the weights during predict.
- If p=0.5, neurons must halve their outputs at test time to have the same output as they had during training time. So need to scale by p

Improving Neural Networks (Dropouts)

- Training Process

```
p = 0.5
def train_step(X):
  H1 = np.dot(W1, X) + b1
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.dot(W2, H1) + b2
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3
  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

- During prediction

```
def predict(X):
  # ensembled forward pass
  H1 = np.dot(W1, X) + b1 # no scaling necessary
  H2 = np.dot(W2, H1) + b2
  out = np.dot(W3, H2) + b3
```

Improving Neural Networks (Dropouts)

- Keras

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

Improving Neural Networks (Dropouts)

- Tensorflow

Model

```
XX = tf.reshape(X, [-1, 28*28])

Y1 = tf.nn.relu(tf.matmul(XX, W1) + B1)
Y1d = tf.nn.dropout(Y1, pkeep)

Y2 = tf.nn.relu(tf.matmul(Y1d, W2) + B2)
Y2d = tf.nn.dropout(Y2, pkeep)

Y3 = tf.nn.relu(tf.matmul(Y2d, W3) + B3)
Y3d = tf.nn.dropout(Y3, pkeep)

Y4 = tf.nn.relu(tf.matmul(Y3d, W4) + B4)
Y4d = tf.nn.dropout(Y4, pkeep)

Ylogits = tf.matmul(Y4d, W5) + B5
Y = tf.nn.softmax(Ylogits)
```

Improving Neural Networks (Dropouts)

- Tensorflow

Model

```
XX = tf.reshape(X, [-1, 28*28])

Y1 = tf.nn.relu(tf.matmul(XX, W1) + B1)
Y1d = tf.nn.dropout(Y1, pkeep)

Y2 = tf.nn.relu(tf.matmul(Y1d, W2) + B2)
Y2d = tf.nn.dropout(Y2, pkeep)

Y3 = tf.nn.relu(tf.matmul(Y2d, W3) + B3)
Y3d = tf.nn.dropout(Y3, pkeep)

Y4 = tf.nn.relu(tf.matmul(Y3d, W4) + B4)
Y4d = tf.nn.dropout(Y4, pkeep)

Ylogits = tf.matmul(Y4d, W5) + B5
Y = tf.nn.softmax(Ylogits)
```

Training Process

```
def training_step(i, update_test_data):
    # training on batches of 100 images with 100 labels
    batch_X, batch_Y = mnist.train.next_batch(100)
    if update_test_data:
        a, c, im = sess.run([accuracy, cross_entropy, It],
                            feed_dict={X: mnist.test.images, Y_: mnist.test.labels, pkeep: 1.0})
    # the backpropagation training step
    sess.run(train_step, {X: batch_X, Y_: batch_Y, pkeep: 0.75, step: i})
```

Notice the pkeep : 1 during testing (Keep all the input data)

Improving Neural Networks (Data Preprocessing)

Mean Subtracting

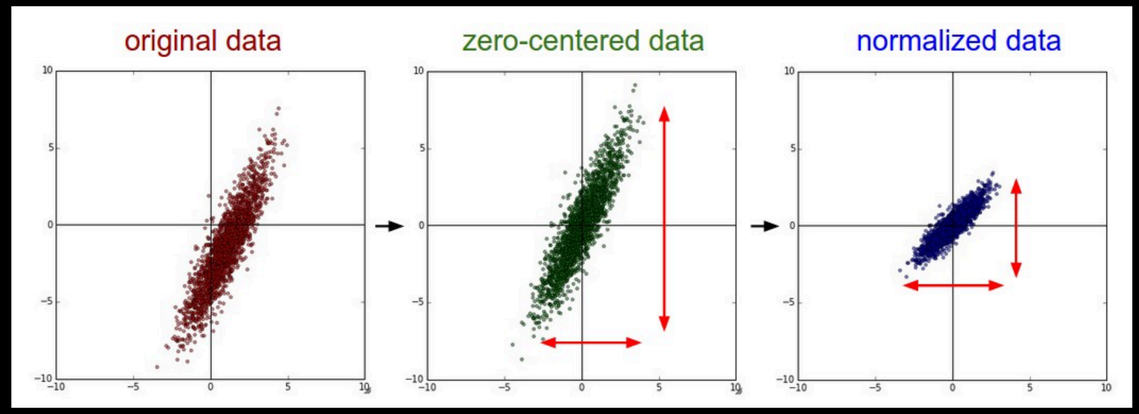
- Subtracting the mean across all the individual feature in data
- Centering the cloud around the origin along every origin

```
#X = [N x D] N: Number of data points ; D : Dimension of Data
X -= np.mean(X, axis = 0)
```

Normalising

- When data is in different scales (units).
- Divide each dimension by its standard deviation.

```
X /= np.std(X, axis = 0)
```



Improving Neural Networks (Miscellaneous)

Varying the Learning Rate (Learning Decay)

- Avoid wasting computations with slow learning rate. (It's expensive!)
- Don't vary too quickly either. (You'll miss the minima)

```
lr = 0.0001 +  tf.train.exponential_decay(0.003, step, 2000, 1/math.e)
train_step = tf.train.AdamOptimizer(lr).minimize(cross_entropy)
```

Weight Initialisation

- Initialising all weights to zero would significantly impede learning given that no weight would initially be active.
- Initialising in an appropriate way can be significantly influential to how easily the network learns from the training set.
- Xavier(Glorot) initialization : Weights from a probability distribution with Variance = $\frac{2}{n_{in}+n_{out}}$

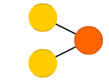
```
model.add(Dense(600, input_dim=784, kernel_regularizer=regularizers.l2(0.01), kernel_initializer='glorot_uniform'))
```

A mostly complete chart of

Neural Networks

- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

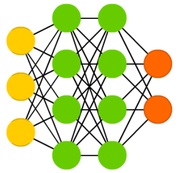
Perceptron (P)



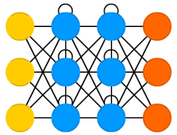
Feed Forward (FF)

Radial Basis Network (RBF)

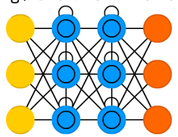
Deep Feed Forward (DFF)



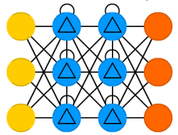
Recurrent Neural Network (RNN)



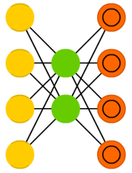
Long / Short Term Memory (LSTM)



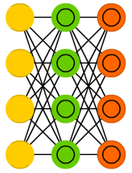
Gated Recurrent Unit (GRU)



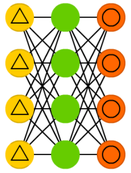
Auto Encoder (AE)



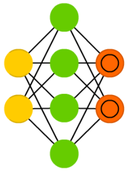
Variational AE (VAE)



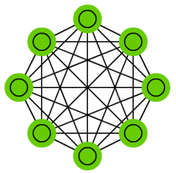
Denoising AE (DAE)



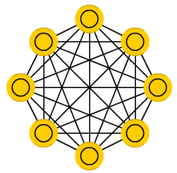
Sparse AE (SAE)



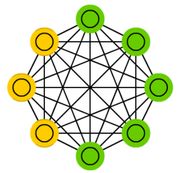
Markov Chain (MC)



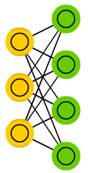
Hopfield Network (HN)



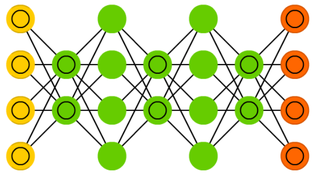
Boltzmann Machine (BM)



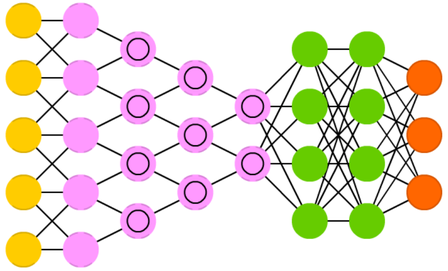
Restricted BM (RBM)



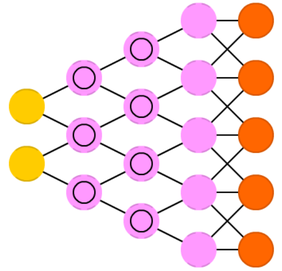
Deep Belief Network (DBN)



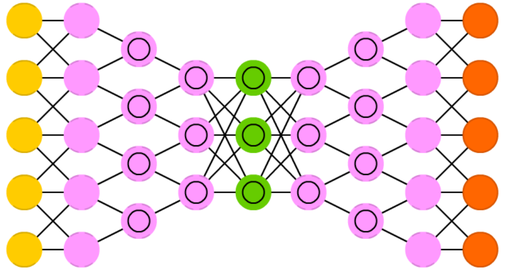
Deep Convolutional Network (DCN)



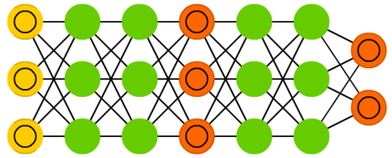
Deconvolutional Network (DN)



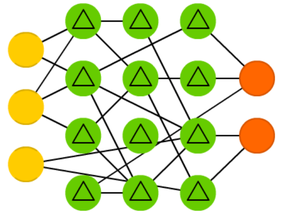
Deep Convolutional Inverse Graphics Network (DCIGN)



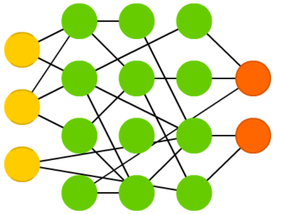
Generative Adversarial Network (GAN)



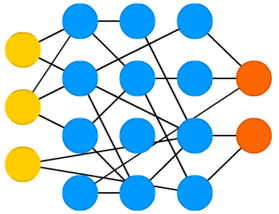
Liquid State Machine (LSM)



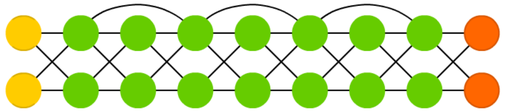
Extreme Learning Machine (ELM)



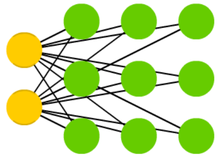
Echo State Network (ESN)



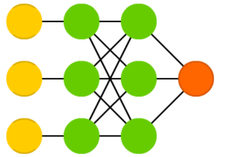
Deep Residual Network (DRN)



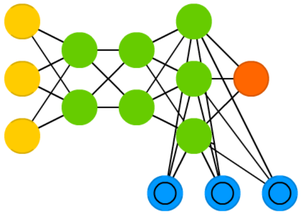
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)



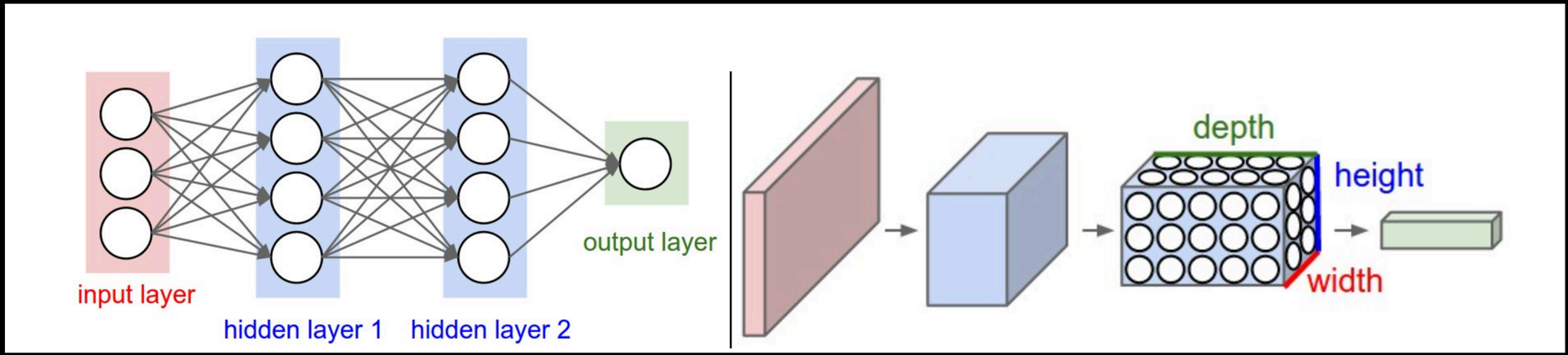
Convolutional Neural Networks

- Just like good old fashioned MLPs
- Trainable Weights and Bias
- Dot Product and Non-Linearity
- Single (Differentiable) Cost function.
- Pixels to Class values
- **Difference** : Developed with images in mind.

- Regular NNs doesn’t scale well with larger input.
  - CIFAR-10 : 32*32*3 = 3072 dimensions
  - Regular 200*200*3 image = 120000 dimensions!!

Convolutional Neural Networks

- Images have localised spatial structures. Exploit that.
- 3D Data : CIFAR 10 : [width, height, depth] = [32, 32, 3]
- Neurons **connect only to a portion** of the image. (Instead of all neurons. Imagine the computations)



// http://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks

- New types of layers :
  - Convolution

```
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
```

  - Compute the output of neurons that are connected to local regions in the input, each computing a **dot product between their weights and a small region they are connected to in the input volume**.
  - This may result in volume such as [32x32x12] if we decided to use 12 filters.

  - Pooling

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(AveragePooling2D(pool_size=(2,2)))
```

Convolutional Neural Networks

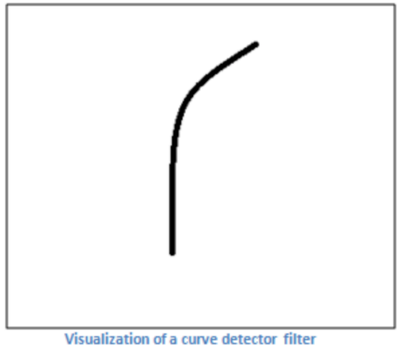
Convolutional Layer

- Consists of learnable filters.
- Spatially small, but **extends to the depth** of image
- During forward pass :
  - Slide across width and height of the whole volume. Calculate the dot product between filter and input.
  - Produces **2D map of activations** (dot product)
- Network ‘learns’ filters that activate similar visual feature. (Like edges or colour)
- Each filter produces one activation map.

Convolutional Layer

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

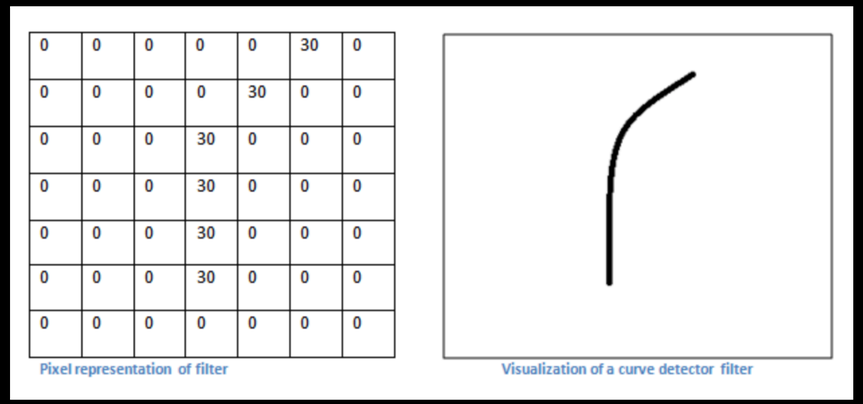
Pixel representation of filter

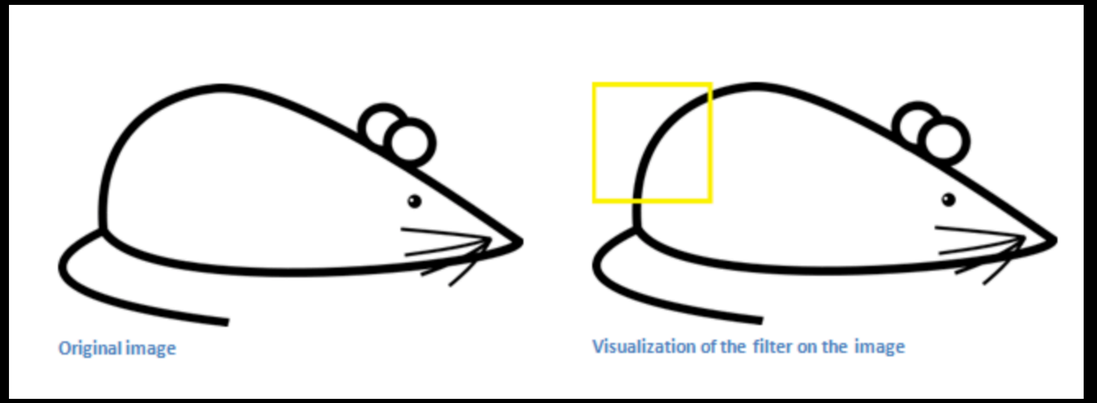


Visualization of a curve detector filter

// https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolutional Layer



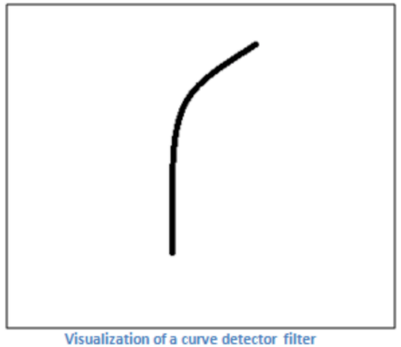


// https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolutional Layer

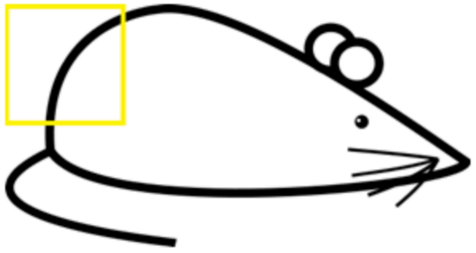
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

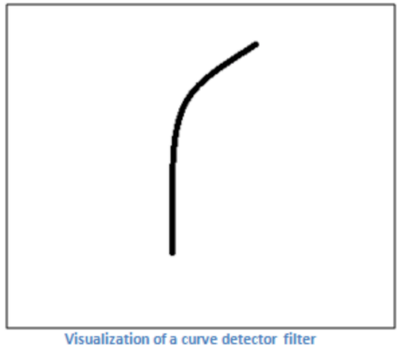
Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

// https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolutional Layer

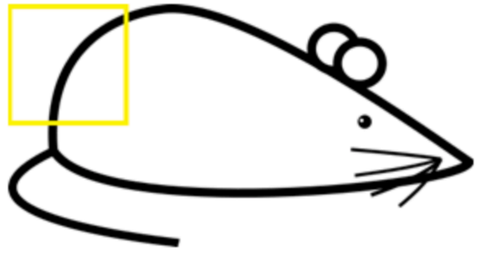
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

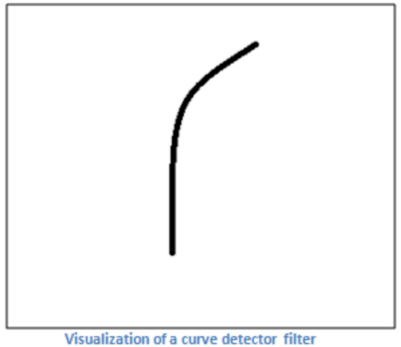
Multiplication and Summation = 0

// https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolutional Layer

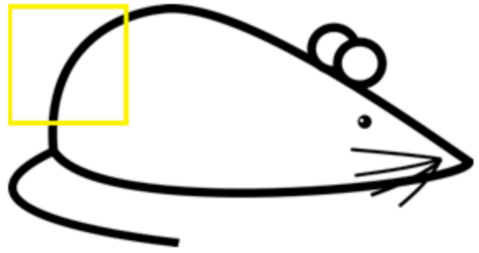
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

// https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

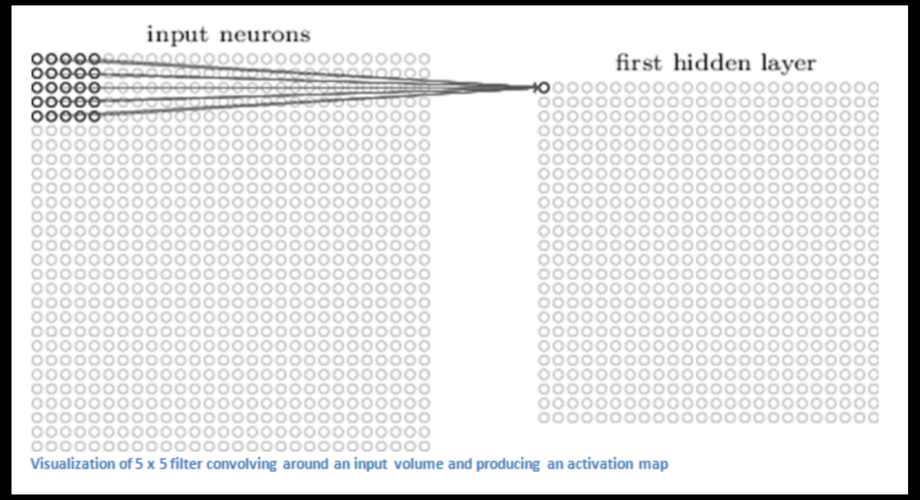
Convolutional Layer

Local Connectivity

- Impractical to connect to all the neurons in previous layer.
- Connect only to a local region.
- Spatial extent = receptive field = filter size
- Extent along the depth axis = depth of input volume

If input is [32x32x3], and filter size = (5,5) : Each filter is 5x5x3 size.
If input is [16x16x20], filter size = (5,5) : Each filter is 5x5x20 size

Single filter (1D) output :



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

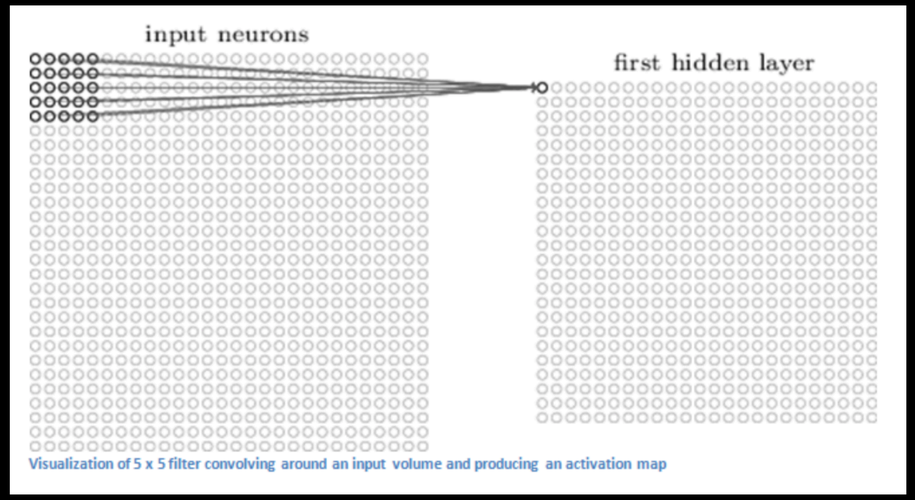
Convolutional Layer

Local Connectivity

- Impractical to connect to all the neurons in previous layer.
- Connect only to a local region.
- Spatial extent = receptive field = filter size
- Extent along the depth axis = depth of input volume

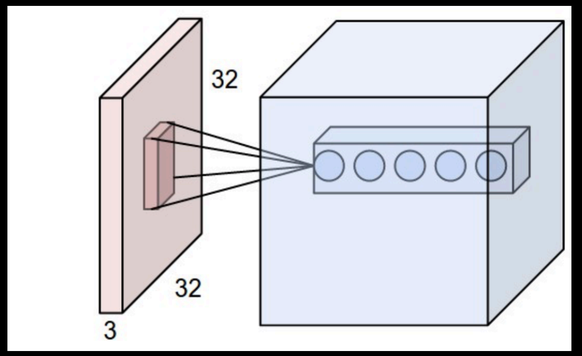
If input is [32x32x3], and filter size = (5,5) : Each filter is 5x5x3 size.
If input is [16x16x20], filter size = (5,5) : Each filter is 5x5x20 size

Single filter (1D) output :



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

Single filter (3D) output :



We can select the number of filters we want.
5 in above case.

Convolutional Layer

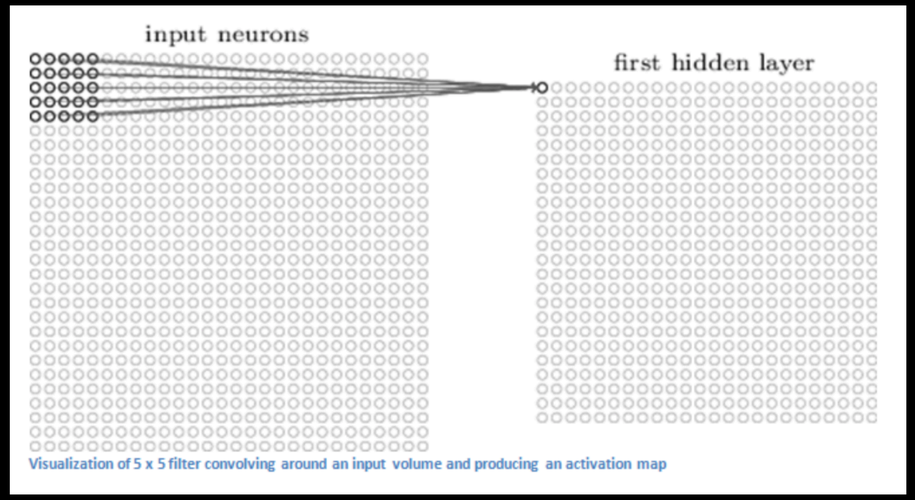
Spatial Arrangement

- Hyper-parameter : Depth of Volume output (No. of filters in output)
- Each filter output (2D layer) = Full multiplication in 3D volume.

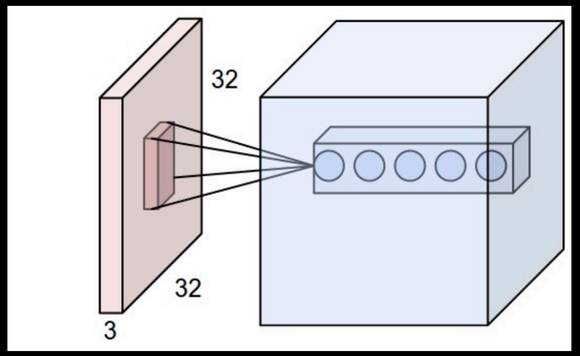
```
sum([5x5x3] (portion of input image) * [5x5x3] (filter)) = 1 number
```

- Filter Depth != Volume Depth
- **Filter Depth = Depth of input image**
- **Volume Depth = What you want to choose = Number of Filters**

Single filter (1D) output :



Single filter (3D) output :



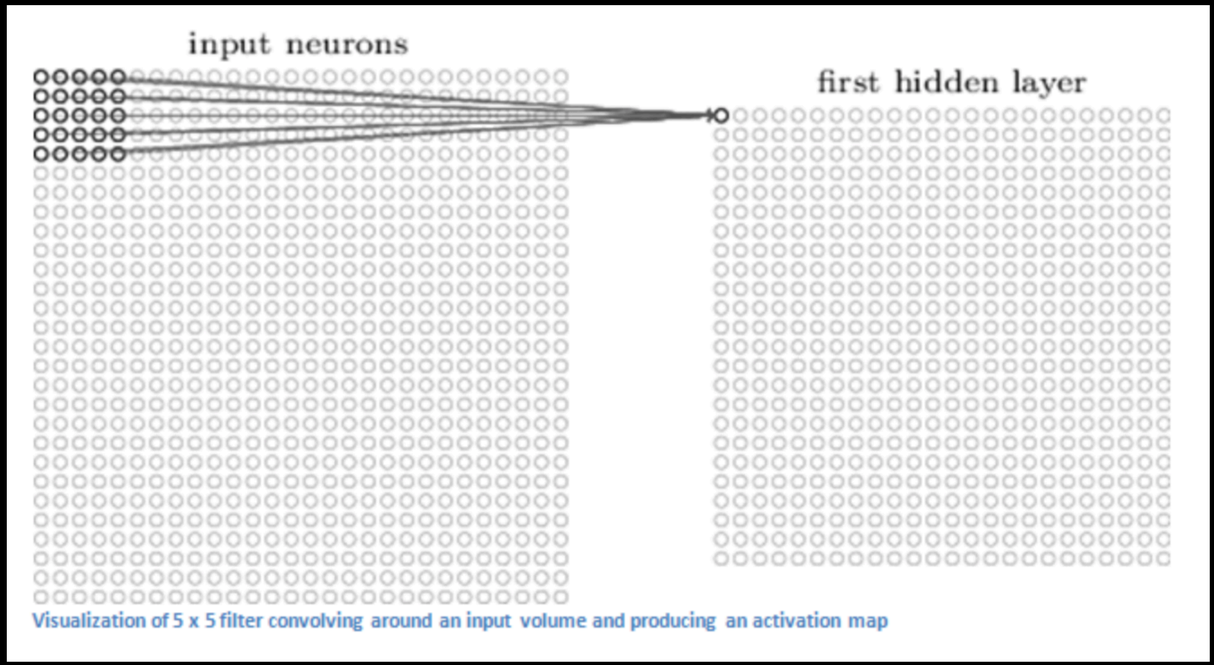
We can select the number of filters we want.
5 in above case.

Convolutional Layer

Spatial Arrangement

- Hyper-parameter : Strides
- No. of pixel jumps during sliding.

- Stride = 1 : Move 1 pixel ahead.



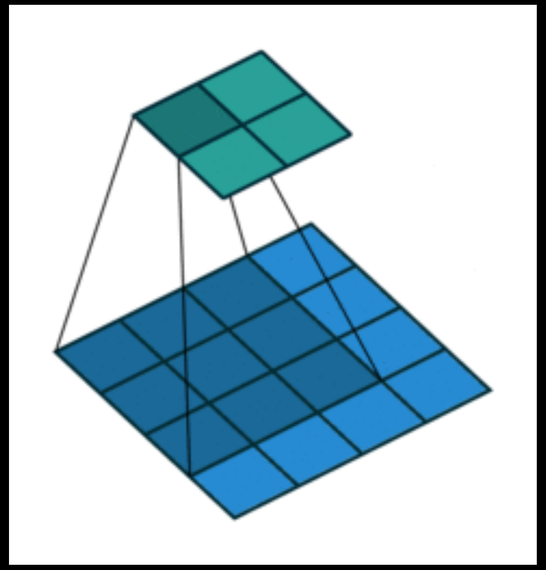
Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

Convolutional Layer

Spatial Arrangement

- Hyper-parameter : Strides
- No. of pixel jumps during sliding.

- Stride = 1 : Move 1 pixel ahead.



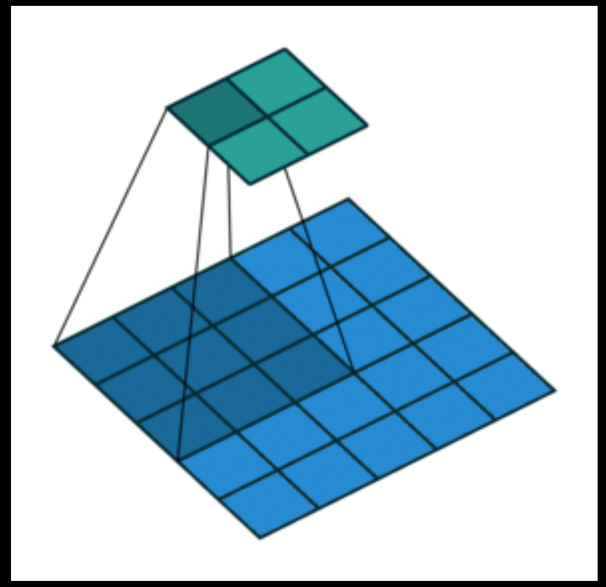
// https://github.com/vdumoulin/conv_arithmetic

Convolutional Layer

Spatial Arrangement

- Hyper-parameter : Strides
- No. of pixel jumps during sliding.

- Stride = 2 : Move 2 pixel ahead.

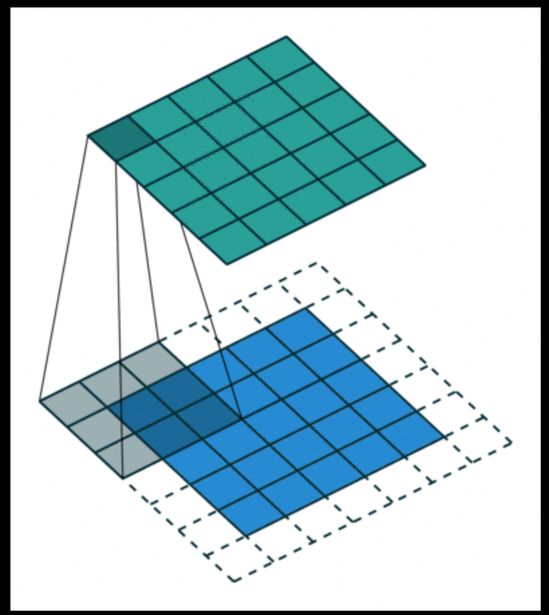


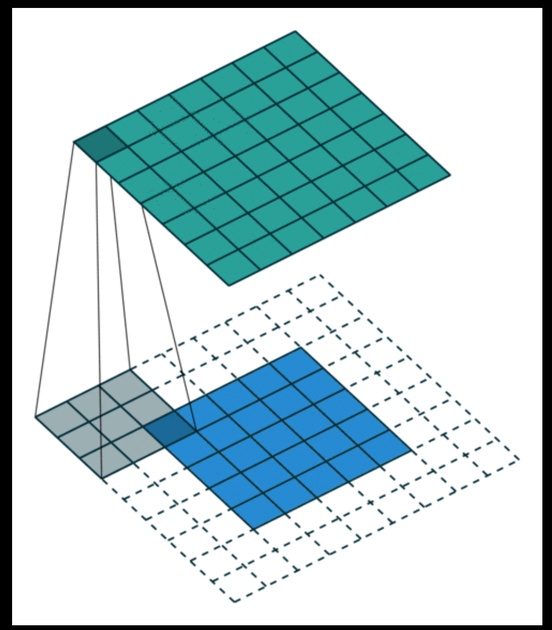
// https://github.com/vdumoulin/conv_arithmetic

Convolutional Layer

Spatial Arrangement

- Hyper-parameter : Padding
- Adding zeros around the image.
- Convenient for controlling spatial size of output.





// https://github.com/vdumoulin/conv_arithmetic

Convolutional Layer

Spatial Size of output

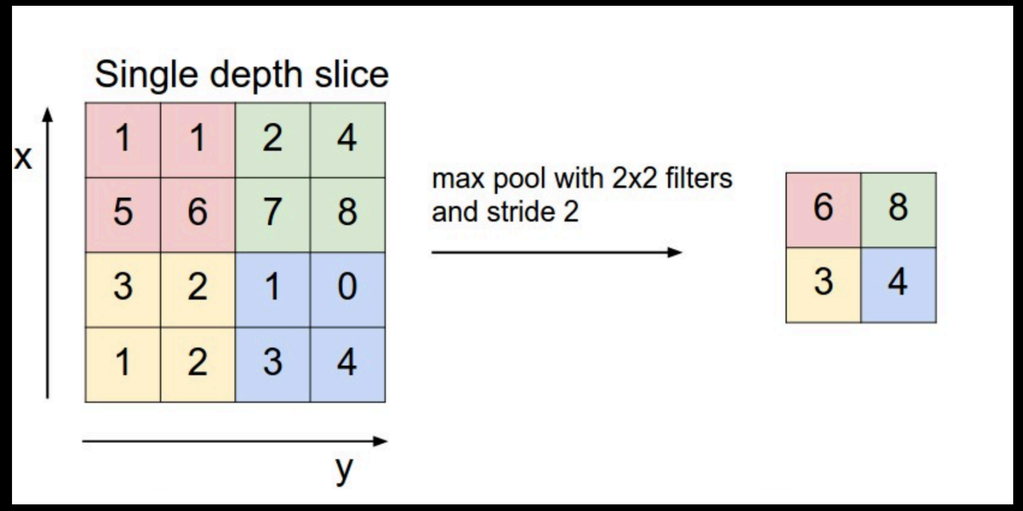
$$(W - F + 2P)/S + 1$$

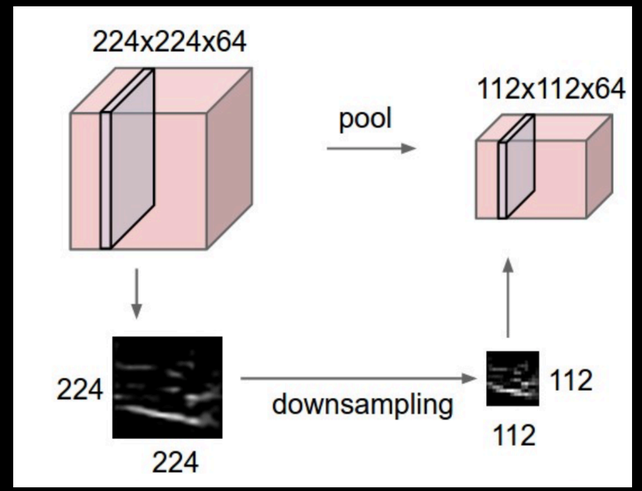
```
W : Input Volume Size
F : Filter Size
P : Padding Size
S : Stride
```

// https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Pooling Layer

- Progressively **reduce the spatial size** of the representation
- **Reduce the amount of parameters** and computation in the network.





// http://cs231n.github.io/convolutional-networks/

Pooling Layer

```
// coding time
```

Any Questions?

email id : harisris@gmail.com