Convolutional Neural Nets

Nishanth Nair

March 25,2020

Overview

Convolutional Neural Networks are a specialized kind of neural nets for processing data that has a known grid-like topology. Examples include time-series data (which can be thought of as a 1D grid taking samples at regular time intervals) and image data (a 2D grid of pixels). The network applies a special mathematical operation called convolution and thus the name.

Image filters and Kernels

- A kernel is a matrix applied to an entire image inorder to transform the image in a certain way.
- This operation of applying a kernel is called convolution.
- Example of applying a kernel on an image:

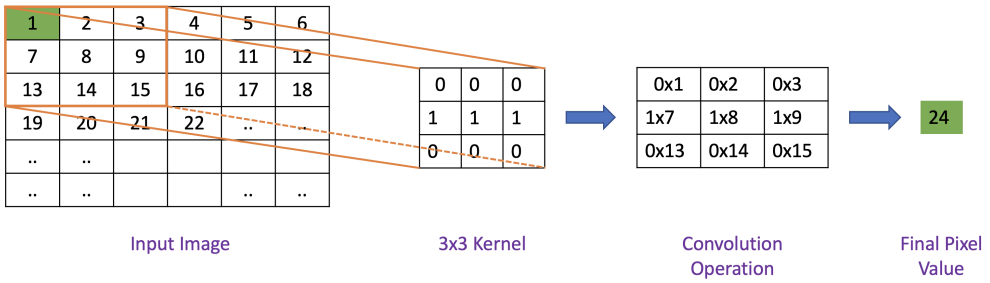


Figure 1: For first pixel

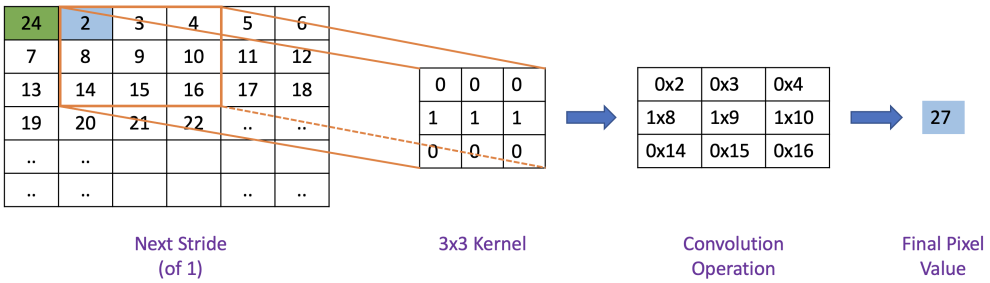


Figure 2: Stride of 1

- A filter is a concatenation of multiple kernels. (One kernel for each channel of the input)
- Filters are always one dimension more than the kernels.For example, in 2D convolutions, filters are 3D matrices.
- So for a CNN layer with kernel dimensions h x w and input channels k, the filter dimensions are k x h x w.

Padding

Padding is the process of adding a layer of zeros to the input.

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	0
0	7	8	9	10	11	12	0
0	13	14	15	16	17	18	0
0	19	20	21	22	..	..	0
0	..	..					0
0	..	..			..	..	0
0	0	0	0	0	0	0	0

Figure 3: Image padding

- In general, if "p_h" rows and "p_w" columns of padding are added, the input image now becomes:

$$(n_h + 2p_h) \times (n_w + 2p_w) \text{ (padded input shape)}$$

- In general, For input shape (n_h, n_w) and kernel size (f_h, f_w), output shape is:

$$(n_h - f_h + 1) \times (n_w - f_w + 1) \text{ (Convolution Output shape)}$$

- With padding,output becomes:

$$(n_h - f_h + 2p_h + 1) \times (n_w - f_w + 2p_w + 1) \text{ (Convolution Output shape with padding)}$$

- This shrinks the output and thus puts an upper limit on the number of times the convolution can be done.
- Pixels at the corner and edges are used much less than those in the middle:

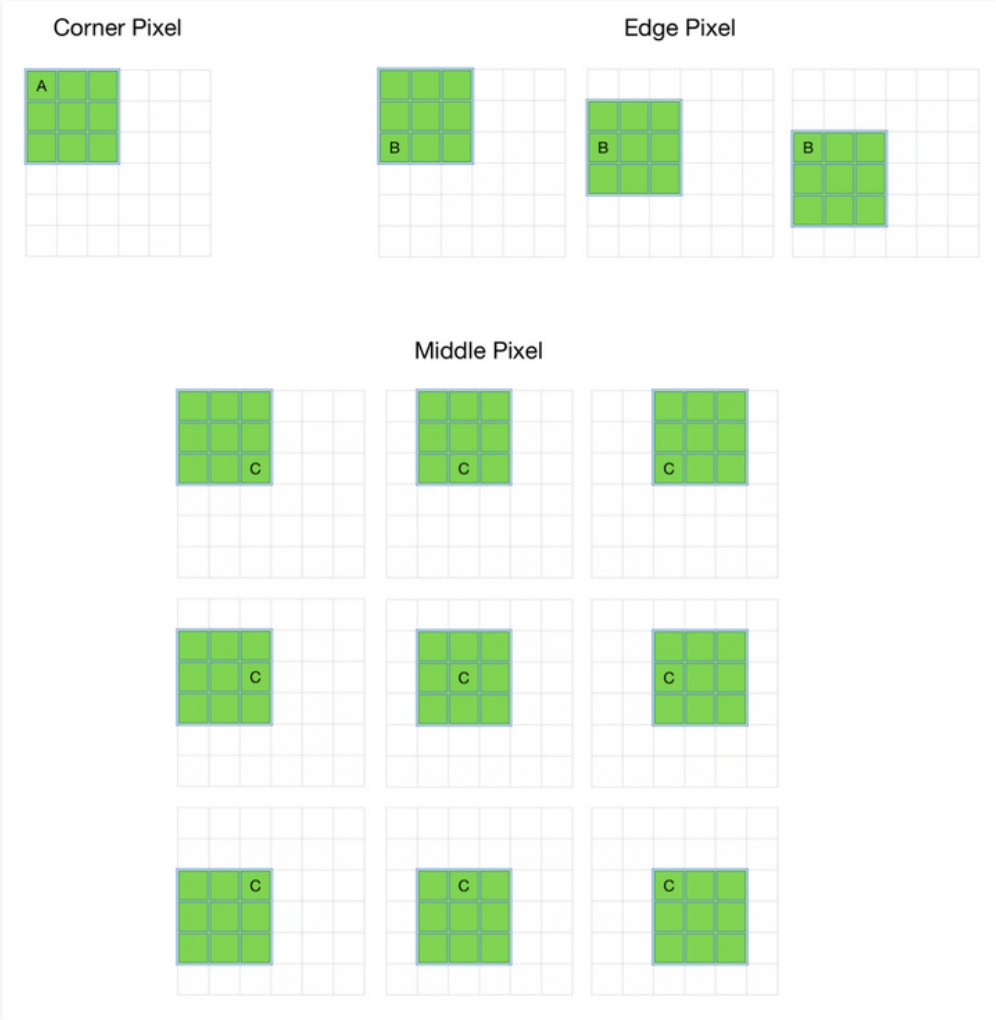


Figure 4: Pixel Contribution during convolution

- In the above,
  - A contributes to 1 convolution
  - B contributes to 3 convolutions
  - C contributes to 9 convolutions
- To overcome the above limitations,padding is used.
- Padding is the process of adding zeros to the input to overcome the above problems.
- **Valid Padding**:
  - Here, no padding is added and we get a smaller output than the input.
  - Last convolution is dropped if dimensions don't match.

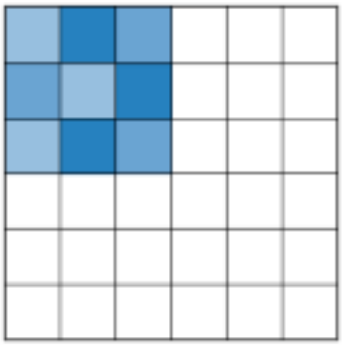


Figure 5: Valid Padding

- **Same Padding**:
  - Add (p_h, p_w) padding layers such that output size is mathematically convenient.
  - Also called "half" padding.
  - Typically, output size will be $\frac{n}{s}$, where n is input size and s is the stride.

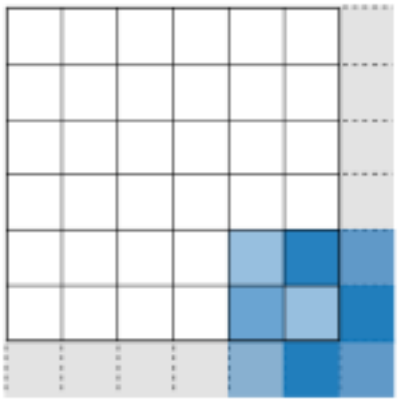


Figure 6: Same Padding

- If we asssume stride is 1, output size will equal input size.
  - To achieve this, padding is computed as follows:

$$n_h + 2p_h - f_h + 1 = n_h \implies p_h = \frac{f_h - 1}{2}$$

$$\text{similarly, } p_w = \frac{f_w - 1}{2}$$

  - For same padding, if kernel size is f x f, (f-1)/2 layers of padding should be added to get output of same size as input.
- **Full Padding**:
  - Maximum padding such that end convolutions are applied on the limits of the input

– Filter "sees" the input end-to-end

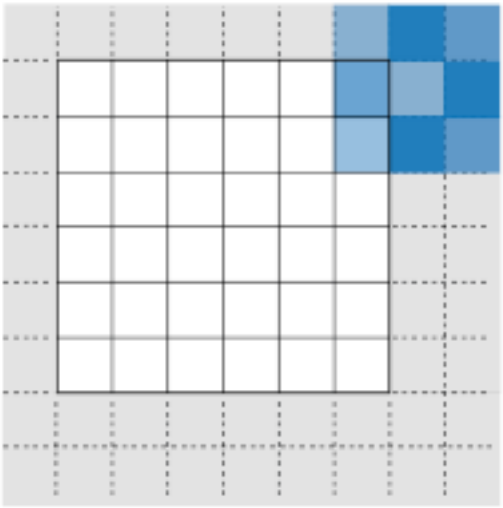


Figure 7: Same Padding

Strides

- The number of rows and columns traversed per slide is called the **stride**.
- When the stride for height is s_h and stride for width is s_w, the output shape is:

$$\frac{(n_h - f_h + 2p_h + s_h)}{s_h} \times \frac{(n_w - f_w + 2p_w + s_w)}{s_w} \text{ (output size)}$$

- Strides can be used to downsample the image
  - If we set $p_h = f_h - 1$ and $p_w = f_w - 1$, the output shape will be:

$$\frac{(n_h + s_h - 1)}{s_h} \times \frac{(n_w + s_w - 1)}{s_w} \text{ (output size)}$$

  - If the input height and width are divisable by stride height and width, output will be:

$$\frac{n_h}{s_h} \times \frac{n_w}{s_w} \text{ (output size)}$$

In summary,

- Padding can increase the height and width of the output.
  - This is often used to give the output the same height and width as the input.
- The stride can reduce the resolution of the output.
- Padding and stride can be used to adjust the dimensionality of the data effectively.

Motivation to use convolution

Convolution leverages three important ideas that help improve ML systems:

1. Sparse Interactions
2. Parameter Sharing
3. Equivariant Representations

Sparse Interactions (sparse connectivity or sparse weights)

- Traditional neural network layers use matrix multiplication with a seperate parameter describing the interaction between each input unit and each output unit. This means, every output interacts with every input.
- CNNs have sparse interactions which is achieved by using kernels smaller than the input.This implies,
  - We have fewer parameters to store which reduces memory requirements and improves statistical efficiency.
  - Computing output requires fewer operations.
- In the below example, one input (x3) affects 3 output units when kernel size is of width 3. In a regular neural net, every input would've affected every output unit.

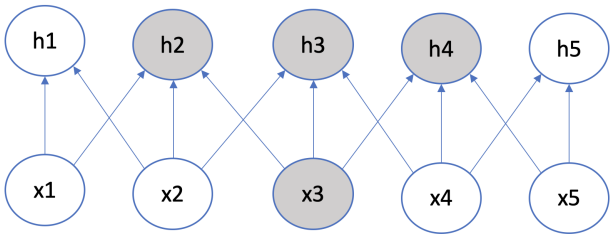


Figure 8: Sparse Connectivity: input -> output

- The input units that affect a particular output unit is called the **receptive field** of that unit.
- In the below example,the receptive field of h3 is depicted.In a convolution net, since kernel width is 3, the receptive field of h3 is 3 units. In a regular neural net, all inputs would've been in the receptive field of h3.

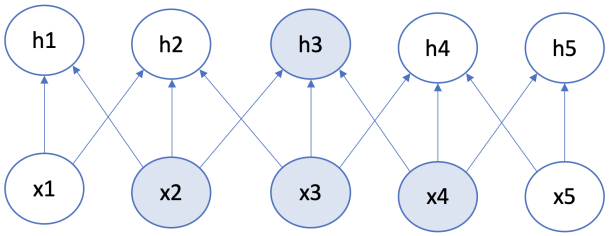


Figure 9: Receptive field of h3

- As depicted below, the receptive field of units in the deeper layers of a CNN is larger than the receptive field at the shallow layers.This means that eventhough direct connections in a CNN are very sparse, units in the deeper layer can be indirectly connected to all of the input image.

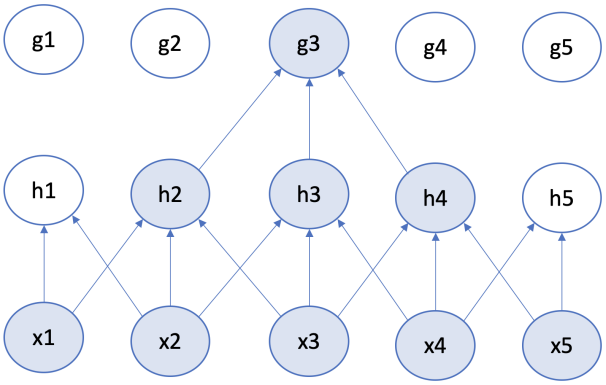


Figure 10: Receptive field of deeper layers

- The receptive field at layer k is the area denoted $R_k \times R_k$ of the input that each pixel of the k-th activation map can 'see'.
- By calling f_j the filter size of layer j and s_i the stride value of layer i and with the convention $s_0 = 1$, the receptive field at layer k can be computed with the formula:

$$R_k = 1 + \sum_{j=1}^{k} (f_j - 1) \prod_{i=0}^{j-1} s_i$$

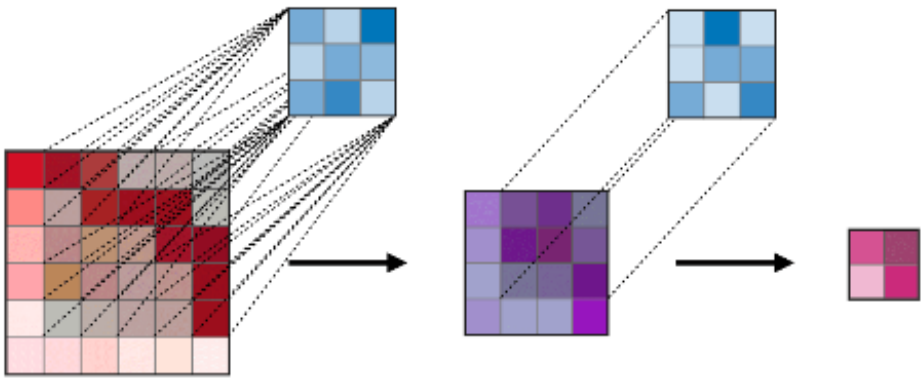


Figure 11: Receptive field example

- In the above example,$f_j = 3$,$s_i = 1$, so receptive field at layer 2, $R_2 = 1+(2\times1+2\times1) = 5$
- Implies, a neuron in the second layer covers 5x5 pixels from the input image.

Parameter Sharing (or weight sharing)

- This refers to using the same parameter for more than one function in a model.
- In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer.
- In a CNN,each member of the kernel is used at every position of the input (except boundaries depending on padding type).
  - This implies, instead of learning a seperate set of parameters for each location, only one set is learned.
  - This reduces storage requirements to k parameters.(where k is kernel size)

Equivariant Representations (or translation invariance)

- Parameter sharing described above causes the layer to have a property called **Equivariance to translation**
- A function $f(x)$ is equivariant to a function g,if $f(g(x)) = g(f(x))$
- For example,
  - When processing time-series data, convolution produces a timeline that shows when different features appear in the input.
    - If we move an event later in time in the input, the exact same representation will appear in the output, just later.
  - When processing image data, convolution creates a 2D map of where certain features appear in the input.
    - This makes object detection better, since its position in the image will not matter.

Convolution Layers

- CNNs are basically just several layers of convolutions with nonlinear activation functions like ReLU or tanh applied to the results.
- In a traditional feedforward neural network we connect each input neuron to each output neuron in the next layer. That's also called a fully connected layer, or affine layer.
- In CNNs we don't do that. Instead, we use convolutions over the input layer to compute the output.
- This results in local connections, where each region of the input is connected to a neuron in the output. Example below:

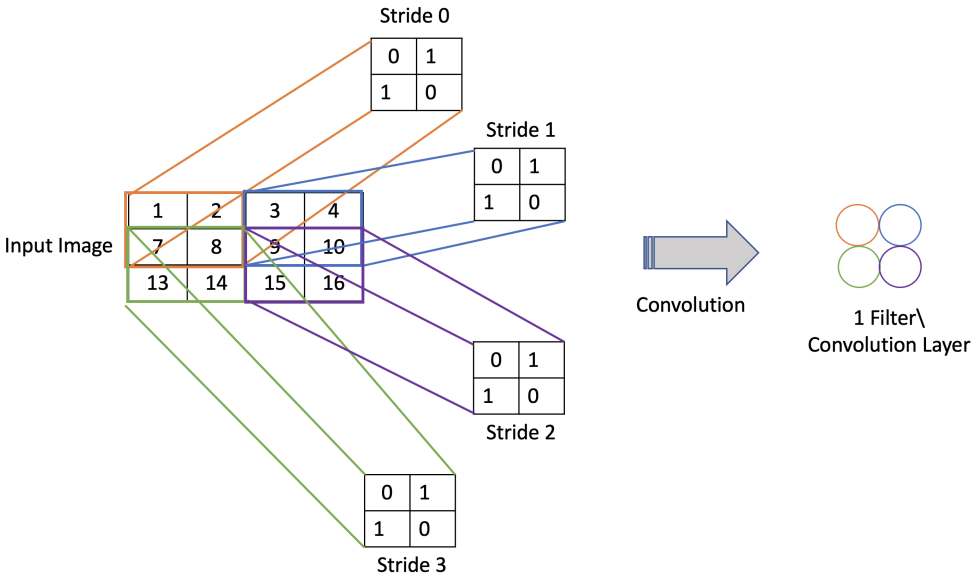


Figure 12: Convolution layer

- During the training phase, a CNN automatically learns the values of its filters based on the task you want to perform.

Pooling

- A key aspect of Convolutional Neural Networks are pooling layers, typically applied after the convolutional layers.
- A typical layer of a CNN consistes of 3 stages
  - First stage: Convolutions are done to produce a set of linear activations
  - Second stage: Each linear activation is run through a non-linear activation function such as ReLU.This stage is called the **detector stage.**
  - Third Stage: A pooling function is used to modify the output of the layer.

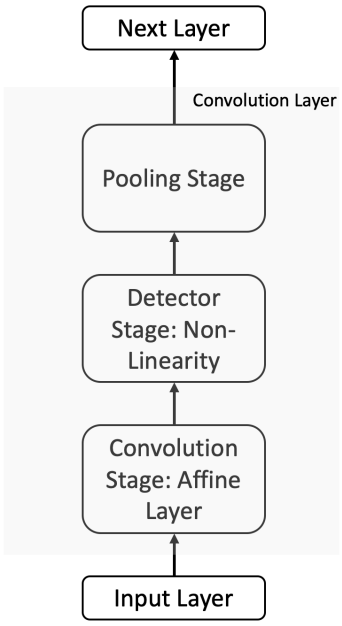


Figure 13: Components of a convolution layer

- A pooling function replaces the output of the previour layer at a certain location with a summary statistic of the nearby outputs.
  - **Max pooling** uses the maximum value from each of a cluster of neurons at the prior layer.
  - **Average pooling** uses the average value from each of a cluster of neurons at the prior layer.

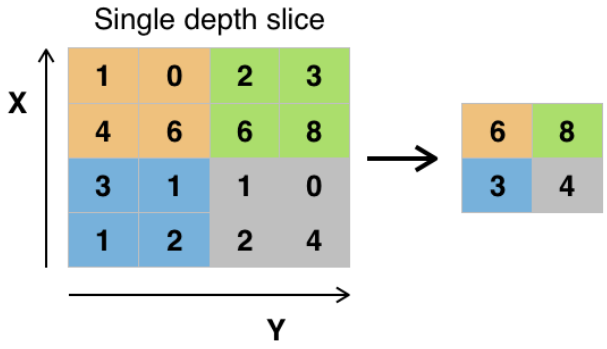


Figure 14: Example of max pooling

- Why Pooling?
  - It provides a fixed size output matrix.
    - This is typically required for classification.
    - Allows you to use variable size inputs, and variable size filters, but always get the same output dimensions to feed into a classifier.

- Pooling layers subsample their input.i.e reduces the output dimensionality but keeps the most salient information.
- Pooling helps make a representation approximately invariant to small translations of the input.
  * Invariance to local translation is useful when we care more about whether some feature is present than exactly where it is.
  * In image recognition,
    · When pooling over a region, the output will stay approximately the same even if you shift/rotate the image by a few pixels.
    · This is because the max operations will pick out the same value regardless.

Fully Connected Layer

A CNN is generally composed of the following layers:

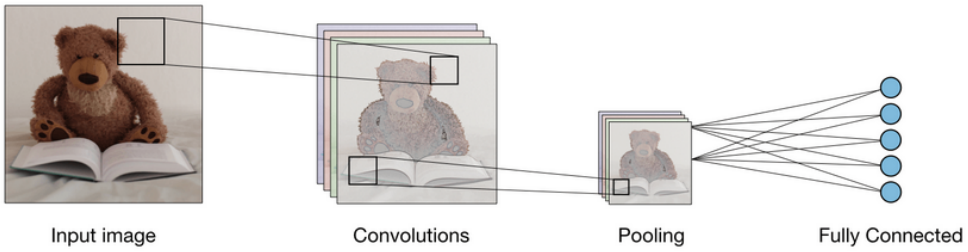
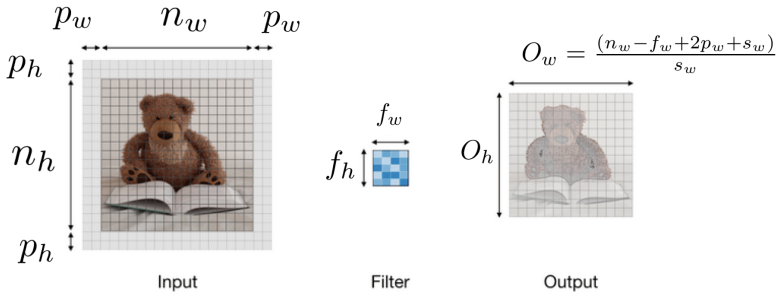


Figure 15: Example of max pooling

- The fully connected layer is a regular feedforward network that operates on a flattened output from the convolution layers.
- If present, the fully connected layers are usually found towards the end of the CNN.
- They are usually used to optimize the objective functions.

Model Complexity (Number of parameters)



	Convolution	Pooling	Fully connected
Input size	$(n_h + 2p_h) \times (n_w + 2p_w) \times C$	$I_w \times I_h \times K$	N_{in}
Output size	$\frac{(n_h - f_h + 2p_h + s_h)}{s_h} \times \frac{(n_w - f_w + 2p_w + s_w)}{s_w} \times K$	$O_{pw} \times O_{ph} \times K$	N_{out}
Number of parameters	$(f_w \times f_h \times C + 1) \times K$	0	$(N_{in} + 1) \times N_{out}$
Additional notes	• K is number of filters • C is number of channels • One bias per filter (that is +1)		• Input is flattened • One bias parameter per neuron

Figure 16: Computing Parameters

Example

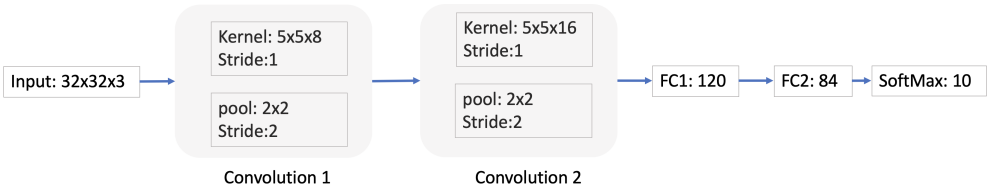


Figure 17: Example CNN

	Input	Output	Units	Parameters
Conv 1 (5x5x8,stride=1)	32 x 32 x 3	32-5+1=28 28 x 28 x 8	28 x 28 x 8 = 6272	(5 x 5 x 3 + 1) x 8 = 608
Conv 1 Pool (2x2,stride=2)	28 x 28 x 8	28-2+2/2=14 14 x14 x 8	14 x 14 x 8 = 1568	0
Conv 2 (5x5x16,stride=1)	14 x14 x 8	14-5+1=10 10 x 10 x 16	10 x 10 x 16 = 1600	(5 x 5 x 8 + 1) x 16 = 3216
Conv 2 Pool (2x2,stride=2)	10 x 10 x 16	10-2+2/2 = 5 5 x 5 x 16	5 x 5 x 16 = 400	0
FC1	400	120	120	(400 + 1) x 120 = 48120
FC2	120	84	84	(120 + 1) x 84 = 10164
SoftMax	84	10	10	(84 + 1) x 10 = 850
Total Parameters				**62958**

Figure 18: Parameter calculation

Backpropogation in CNN

TB

CNN Architectures

LeNet-5 (1998)

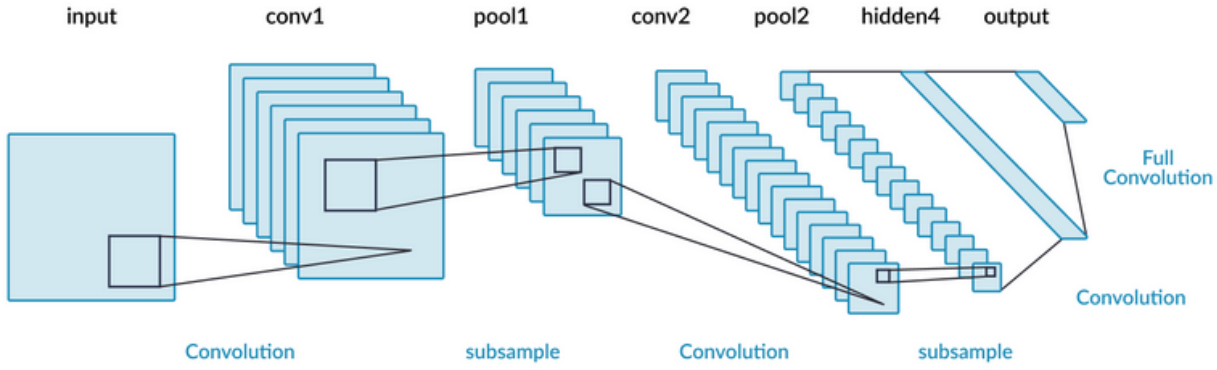


Figure 19: LeNet Architecture

AlexNet (2012)

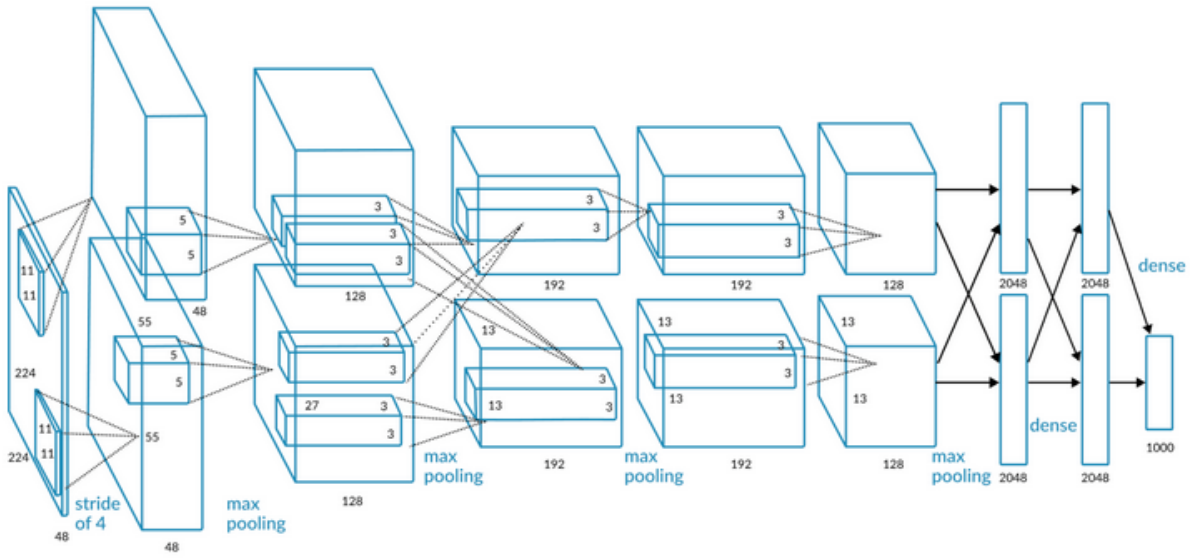


Figure 20: AlexNet Architecture

GoogleNet or Inception

Insight

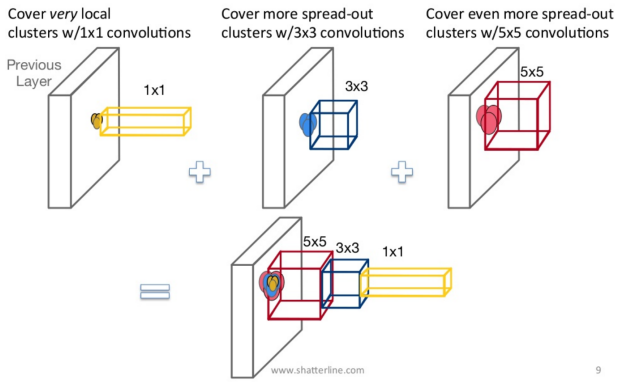


Figure 21: Insights for architecture

Inception Module

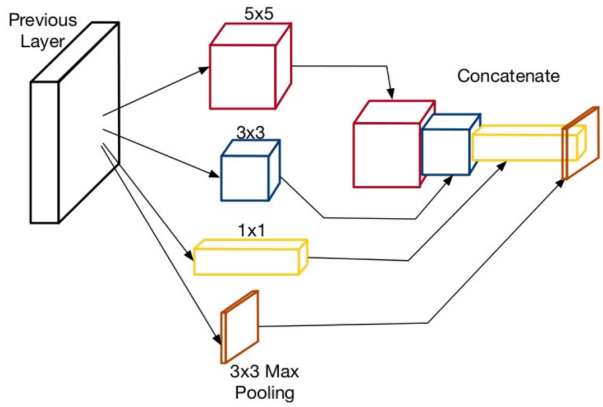


Figure 22: Overview of Inception module

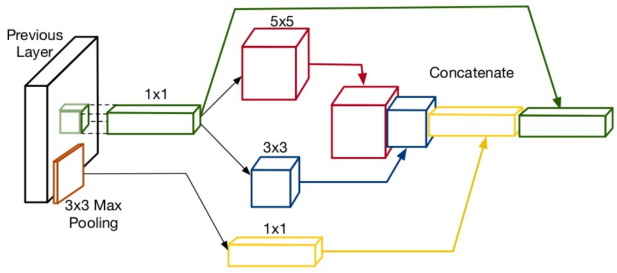


Figure 23: Inception module in practice

Architecture

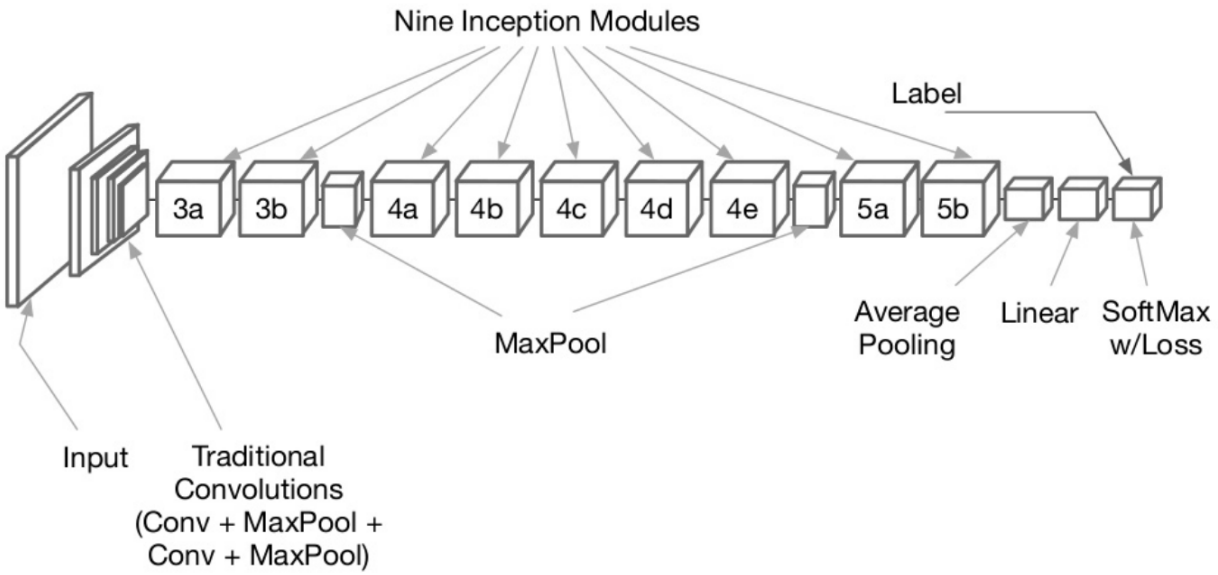


Figure 24: GoogleNet Architecture

CNNs for NLP

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/