Lec10 - Convolutional Neural Networks

CS 783 - Visual Recognition

Vinay P. Namboodiri

IIT Kanpur

7^{th} February 2019

Contents

Outline

Backpropagation algorithm

In the previous class we have covered the backpropagation algorithm and seen the equations in detail

- Used to update all the weights based on the update rule.

Backpropagation algorithm

In the previous class we have covered the backpropagation algorithm and seen the equations in detail

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate η

Backpropagation algorithm

In the previous class we have covered the backpropagation algorithm and seen the equations in detail

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate η
- There are different modes of update

Backpropagation algorithm

In the previous class we have covered the backpropagation algorithm and seen the equations in detail

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate η
- There are different modes of update
  - Online: After seeing each training sample

Backpropagation algorithm

In the previous class we have covered the backpropagation algorithm and seen the equations in detail

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate η
- There are different modes of update
  - Online: After seeing each training sample
  - Full: After seeing all the training samples

Backpropagation algorithm

In the previous class we have covered the backpropagation algorithm and seen the equations in detail

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate η
- There are different modes of update
  - Online: After seeing each training sample
  - Full: After seeing all the training samples
  - mini-batch: After seeing a small set of training samples. This is the most common case.

Backpropagation algorithm

In the previous class we have covered the backpropagation algorithm and seen the equations in detail

- Used to update all the weights based on the update rule.
- Weights are adapted based on the learning rate η
- There are different modes of update
  - Online: After seeing each training sample
  - Full: After seeing all the training samples
  - mini-batch: After seeing a small set of training samples. This is the most common case.
- Gradient descent with backpropagation is not guaranteed to find the global minimum of the error function, but only a local minimum

Outline

Sigmoidal Activation function

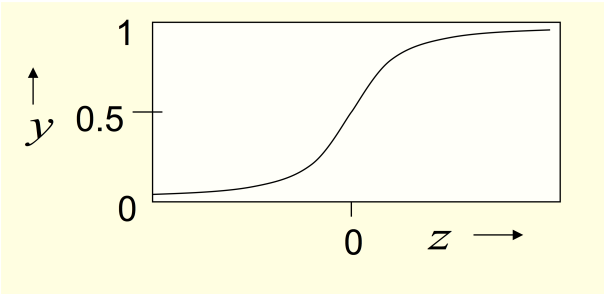


Figure: Sigmoidal Activation function

- We have also seen the sigmoidal activation function

Sigmoidal Activation function

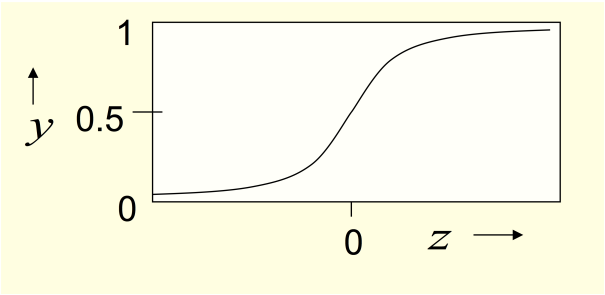


Figure: Sigmoidal Activation function

- We have also seen the sigmoidal activation function
- The sigmoidal activation function can be used (it approximates the discontinuous Heaviside function used in perceptrons).

Sigmoidal Activation function

- The sigmoidal activation function is given by

$$y_j = \frac{1}{1 + e^{-h_j}} \tag{1}$$

Sigmoidal Activation function

- The sigmoidal activation function is given by

$$y_j = \frac{1}{1 + e^{-h_j}} \tag{1}$$

- The derivative of the sigmoid is obtained as

$$\frac{\partial y_j}{\partial h_j} = y_j(1 - y_j) \tag{2}$$

as $\frac{\partial y_j}{\partial h_j} = \frac{-1}{(1+e^{-h_j})^2}(-e^{-h_j}) = (\frac{1}{1+e^{-h_j}})(\frac{e^{-h_j}}{1+e^{-h_j}}) = y_j(1 - y_j)$

Tanh Activation function

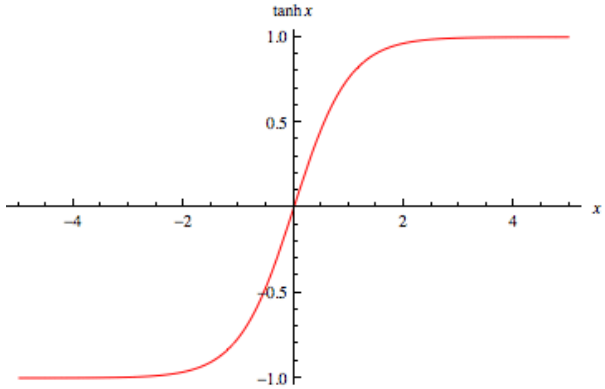


Figure: Tanh Activation function

- The tanh activation function is given by

$$g(z) = \frac{sinh(z)}{cosh(z)} \tag{3}$$

Derivative of Tanh Activation function

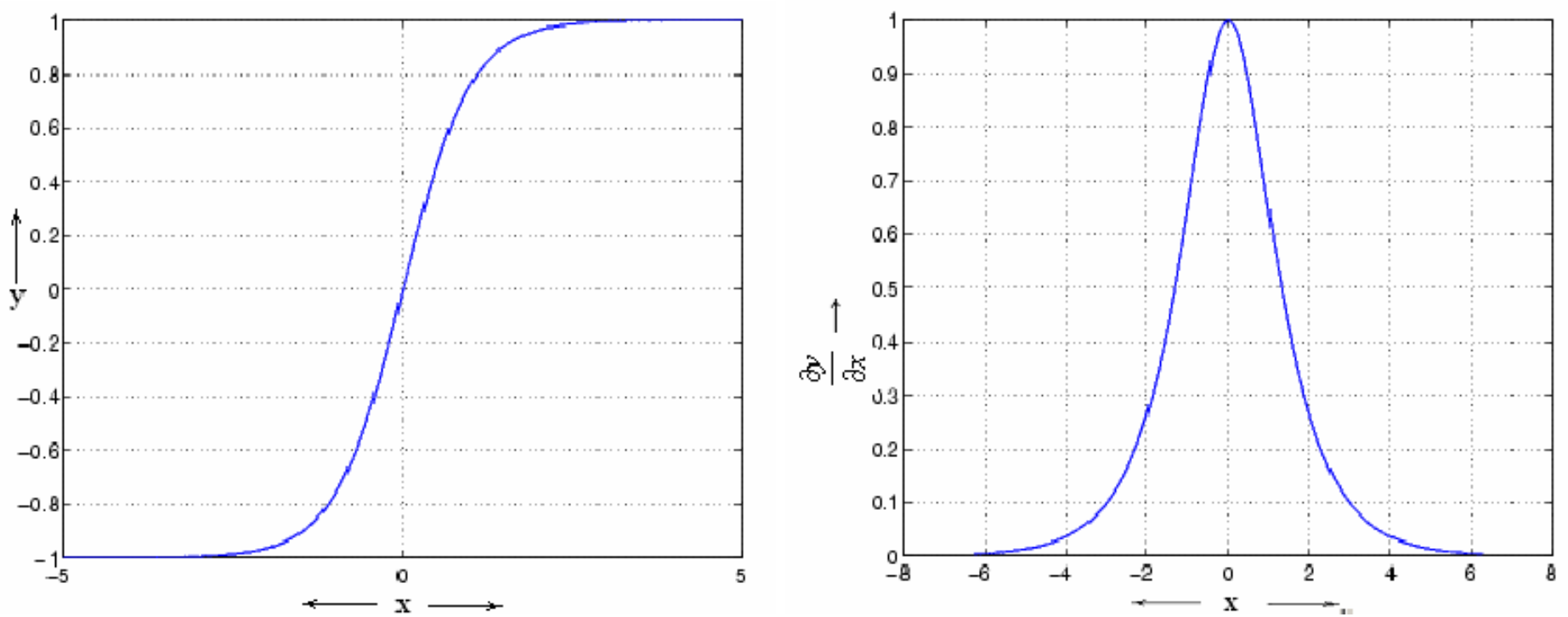


Figure: Derivative of Tanh Activation function

Derivative of Tanh Activation function

- The derivative of the tanh activation function is obtained as

$$g'_{tanh}(z) = \frac{\partial}{\partial z}\frac{sinh(z)}{cosh(z)} \tag{4}$$

$$= \frac{\frac{\partial}{\partial z}sinh(z) \times cosh(z) - \frac{\partial}{\partial z}cosh(z) \times sinh(z)}{cosh^2(z)} \tag{5}$$

$$= \frac{cosh^2(z) - sinh^2(z)}{cosh^2(z)} \tag{6}$$

$$= 1 - \frac{sinh^2(z)}{cosh^2(z)} \tag{7}$$

$$= 1 - tanh^2(z) \tag{8}$$

ReLU Activation function

- The previous activation functions had a problem as for the most part the gradient of the activation function would be zero for a large range of the values with a small non-zero operating range where the gradient would be non-zero

ReLU Activation function

- The previous activation functions had a problem as for the most part the gradient of the activation function would be zero for a large range of the values with a small non-zero operating range where the gradient would be non-zero
- This would result in the vanishing gradient problem as the gradients would vanish and the network would not learn.

ReLU Activation function

- The previous activation functions had a problem as for the most part the gradient of the activation function would be zero for a large range of the values with a small non-zero operating range where the gradient would be non-zero
- This would result in the vanishing gradient problem as the gradients would vanish and the network would not learn.
- The rectified linear unit function (commonly called ReLU) addresses this problem

ReLU Activation function

- The previous activation functions had a problem as for the most part the gradient of the activation function would be zero for a large range of the values with a small non-zero operating range where the gradient would be non-zero
- This would result in the vanishing gradient problem as the gradients would vanish and the network would not learn.
- The rectified linear unit function (commonly called ReLU) addresses this problem
- The ReLU function is given by

$$f(x) = max(0, x) \tag{9}$$

ReLU Activation function

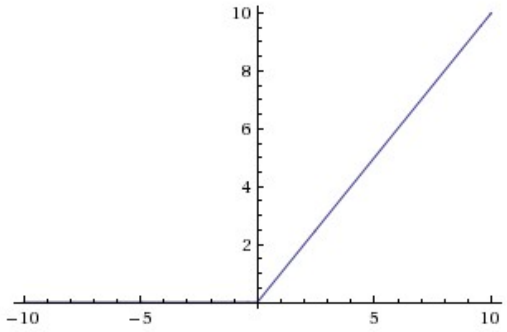


Figure: ReLU Activation function

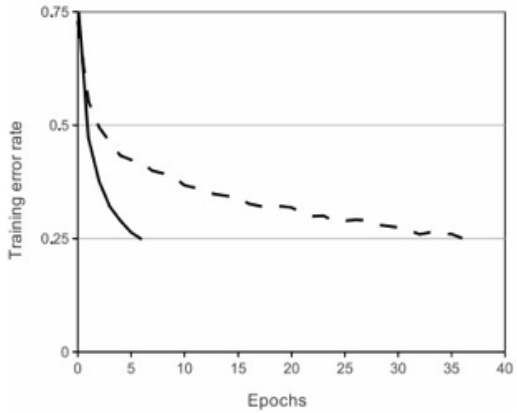


Figure: Effect of ReLU function on convergence

ReLU Activation function

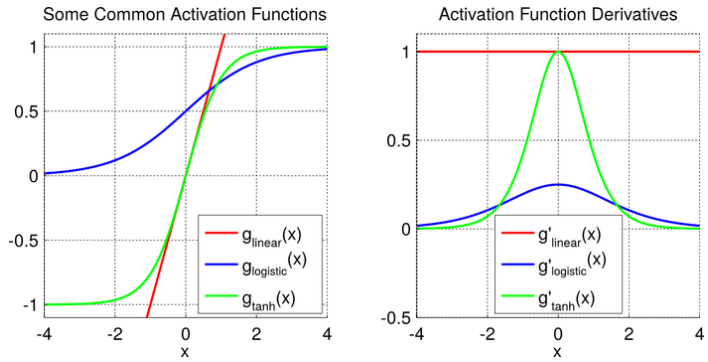


Figure: Comparison of Derivatives of functions

- The derivative of the ReLU function is obtained as

$$f'(x) = \begin{cases} 1 \text{ if } x > 0, \\ 0 \text{ otherwise} \end{cases} \tag{10}$$

Parameterized ReLU Activation function

- The ReLU function also has problems with vanishing gradient and occasionally specific neurons would not get updated.

Parameterized ReLU Activation function

- The ReLU function also has problems with vanishing gradient and occasionally specific neurons would not get updated.
- One way to address this is with a leaky ReLU function given by

$$f(x) = \begin{cases} x \text{ if } x > 0, \\ 0.0001x \text{ otherwise} \end{cases} \tag{11}$$

Parameterized ReLU Activation function

- The ReLU function also has problems with vanishing gradient and occasionally specific neurons would not get updated.
- One way to address this is with a leaky ReLU function given by

$$f(x) = \begin{cases} x \text{ if } x > 0, \\ 0.0001x \text{ otherwise} \end{cases} \tag{11}$$

- A related variant is to use a scalar parameter to define the ReLU activation function

$$f(x) = \begin{cases} x \text{ if } x > 0, \\ a\dot{x} \text{ otherwise} \end{cases} \tag{12}$$

where the parameter a is also learned

Parameterized ReLU Activation function

- The ReLU function also has problems with vanishing gradient and occasionally specific neurons would not get updated.
- One way to address this is with a leaky ReLU function given by

$$f(x) = \begin{cases} x \text{ if } x > 0, \\ 0.0001x \text{ otherwise} \end{cases} \tag{11}$$

- A related variant is to use a scalar parameter to define the ReLU activation function

$$f(x) = \begin{cases} x \text{ if } x > 0, \\ a\dot{x} \text{ otherwise} \end{cases} \tag{12}$$

  where the parameter a is also learned
- Another variant is the maxout activation function where two parameter vectors are used

$$f(x) = max(w_1^T x + b_1, w_2^T x + b2) \tag{13}$$

Illustration of effect of ReLU

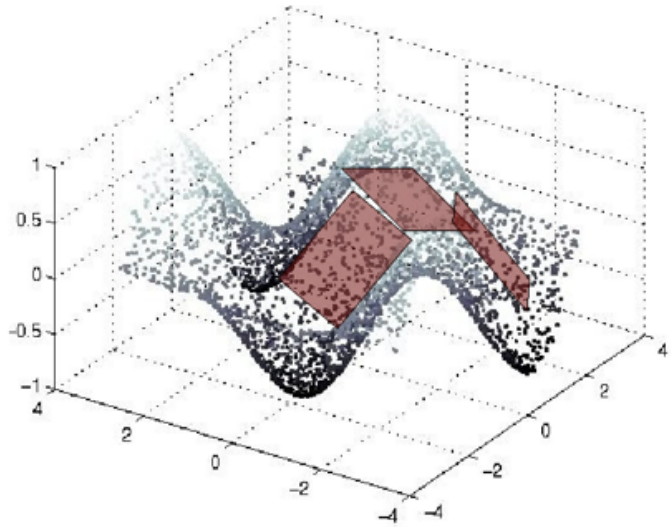


Figure: ReLU function with data (figure credit: Marc Aurelio Ranzato)

- The ReLU activation function results in piecewise linear planar regions that are learned and oriented to maximize the classification accuracy.

Illustration of effect of ReLU

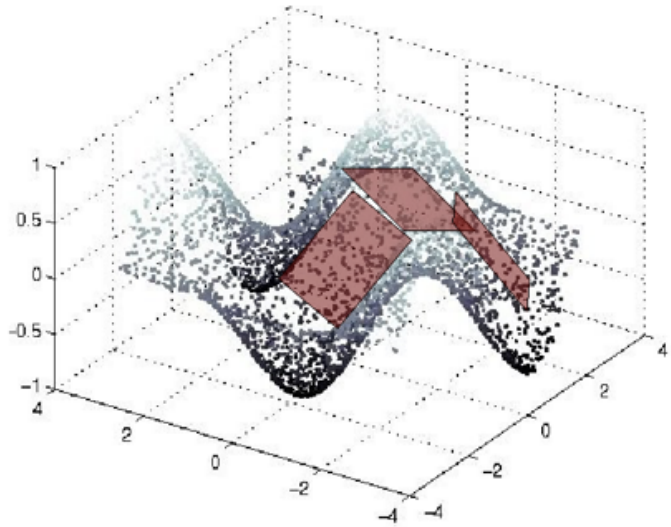


Figure: ReLU function with data (figure credit: Marc Aurelio Ranzato)

- The ReLU activation function results in piecewise linear planar regions that are learned and oriented to maximize the classification accuracy.
- It results in linear tiling of the feature space

Outline

MLP as a universal approximator

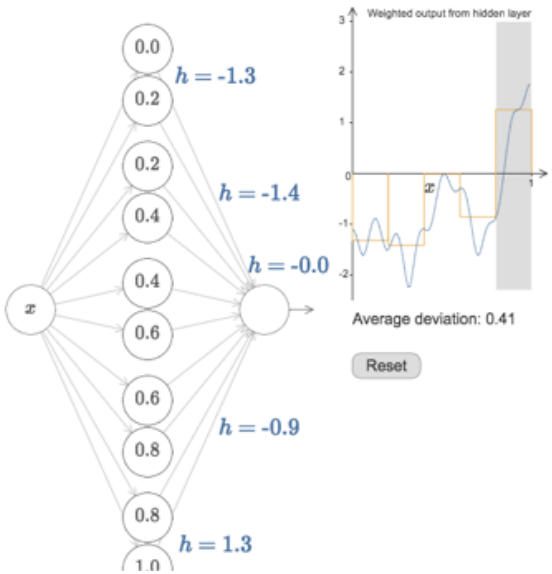


Figure: Illustration of an MLP approximating a function (figure credit: Michael Nielsen)

- According to the universal approximation theorem, any multi-layer perceptron with a single hidden layer can be used to approximate continuous functions on compact subsets of $\mathcal{R}^n$ under mild assumptions on the activation function.

Distributed Representations

A single layer representation does not make use of the distributed representation power and results in a high degree of redundancy and requires an exponential number of neurons.

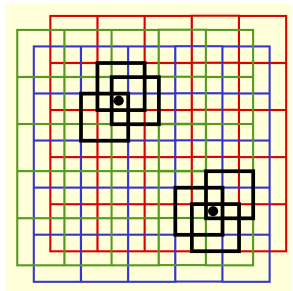


Figure: A fine cell representation through coarse cells (figure by Geoffrey Hinton)

- A fine cell representation can be obtained by distributing the representation over three coarse cells. If a point falls in a cell, all cells in which that point falls record that information.

Distributed Representations

A single layer representation does not make use of the distributed representation power and results in a high degree of redundancy and requires an exponential number of neurons.

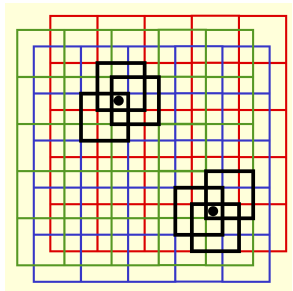


Figure: A fine cell representation through coarse cells (figure by Geoffrey Hinton)

- A fine cell representation can be obtained by distributing the representation over three coarse cells. If a point falls in a cell, all cells in which that point falls record that information.
- This information is recorded 3 times, however, it saves on a 3×3 discretisation that would be needed to record fine information.

Distributed Representations

This idea can be used to obtain a distributed representation across different functions reducing the redundancy

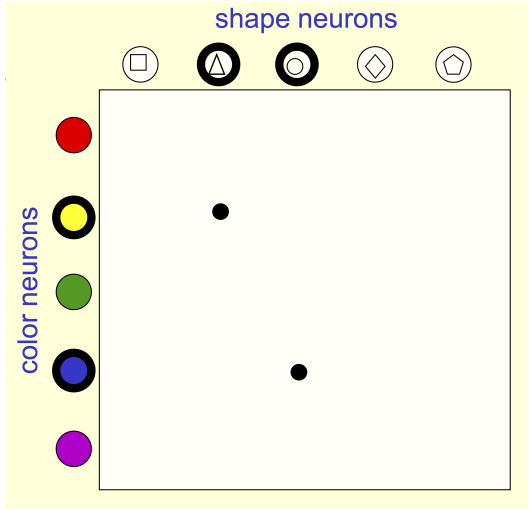


Figure: Distributing shape and color functions (figure by Geoffrey Hinton)

Distributed Representations

This idea can be used to obtain a distributed representation across different functions reducing the redundancy

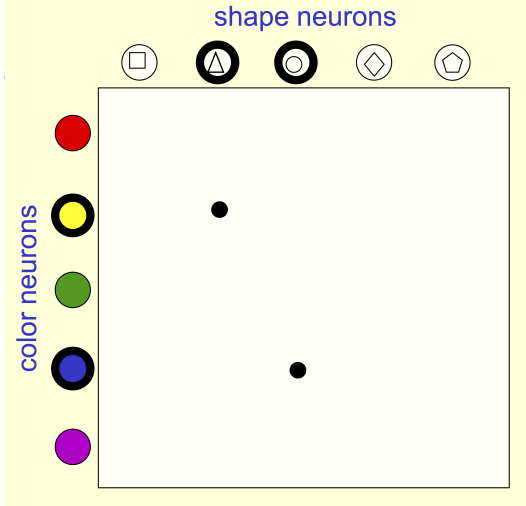


Figure: Distributing shape and color functions (figure by Geoffrey Hinton)

- A description of an object could be recorded by distributing it across shape and color functions, such as a blue circle or a yellow triangle.

Parameters for a fully connected network

As we use fully connected network with multiple layers to distribute the representation, the number of parameters required for a fully connected network can be huge

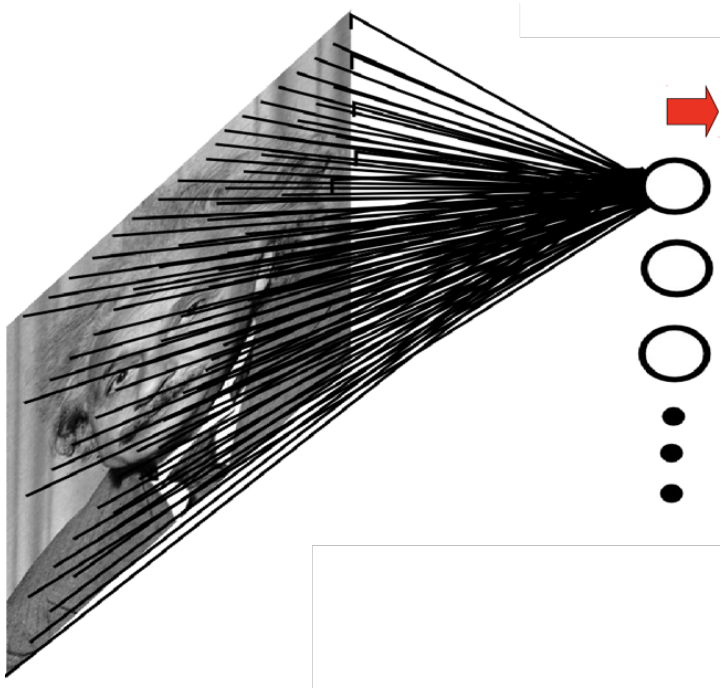


Figure: A 200× 200 image would require 4 Billion parameters (figure by Marc Aurelio Ranzato)

Parameters for a fully connected network

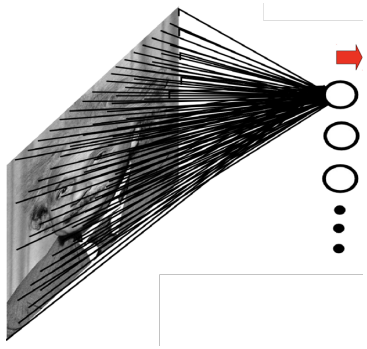


Figure: A 200× 200 image would require 4 Billion parameters (figure by Marc Aurelio Ranzato)

Parameters for a fully connected network

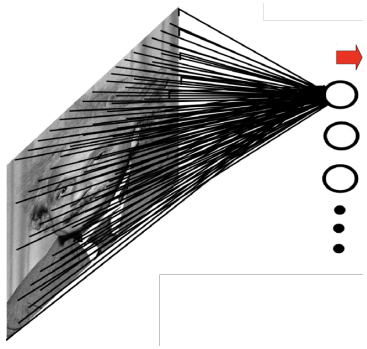


Figure: A 200× 200 image would require 4 Billion parameters (figure by Marc Aurelio Ranzato)

- As we consider a fully connected network through a distributed representation, it will require a huge number of parameters.

Parameters for a fully connected network

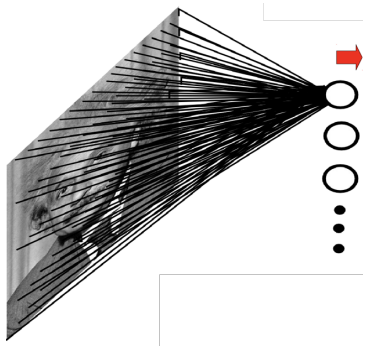


Figure: A 200× 200 image would require 4 Billion parameters (figure by Marc Aurelio Ranzato)

- As we consider a fully connected network through a distributed representation, it will require a huge number of parameters.
- Moreover, such a representation does not take into account that specific representation parts are usually local, for instance face can be represented in terms of local part representations such as eyes, nose, etc.

Locally connected network

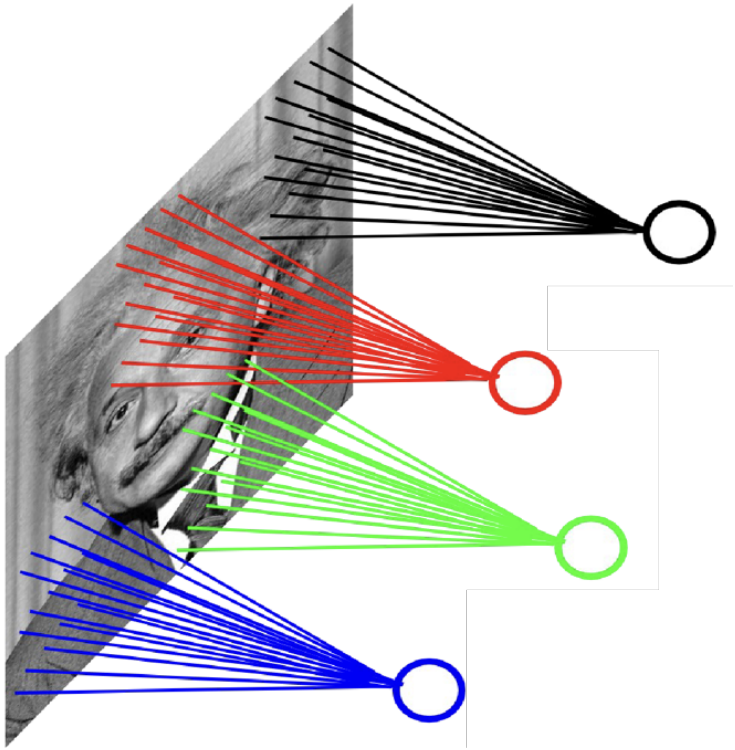


Figure: A 200× 200 image would require 4 million parameters in a locally connected (figure by Marc Aurelio Ranzato)

Locally connected network

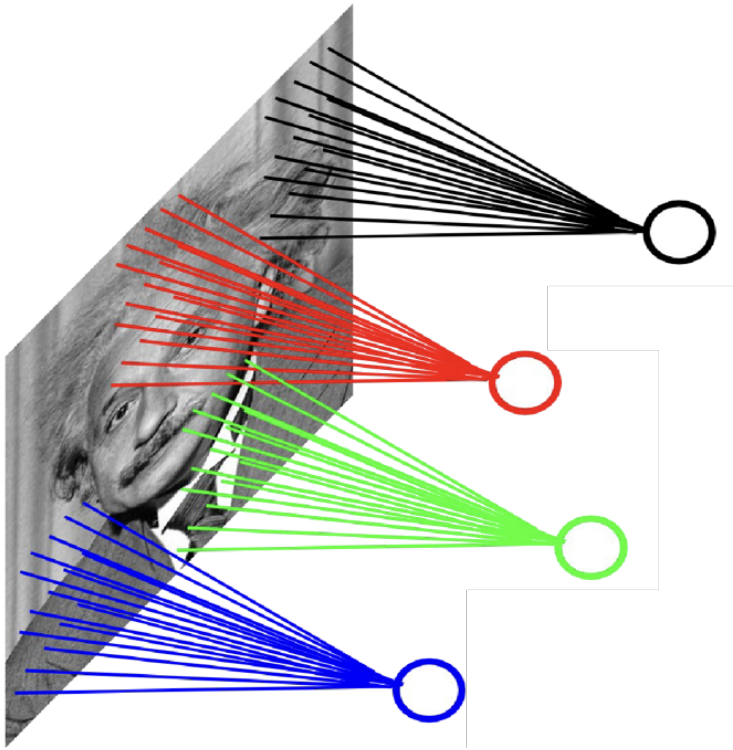


Figure: A 200× 200 image would require 4 million parameters in a locally connected (figure by Marc Aurelio Ranzato)

- A local representation could be used for representing the parts.

Locally connected network

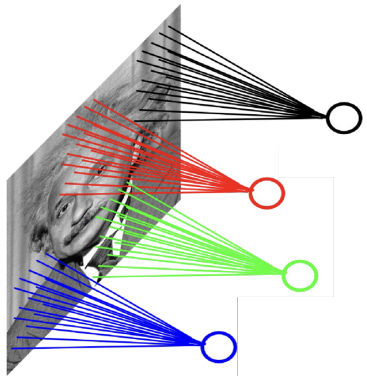


Figure: A 200× 200 image would require 4 million parameters in a locally connected (figure by Marc Aurelio Ranzato)

- A local representation could be used for representing the parts.
- This is possible more for rigidly registered cases such as faces where we know that the eyes and nose are fixed relative to other samples.

Convolutional network

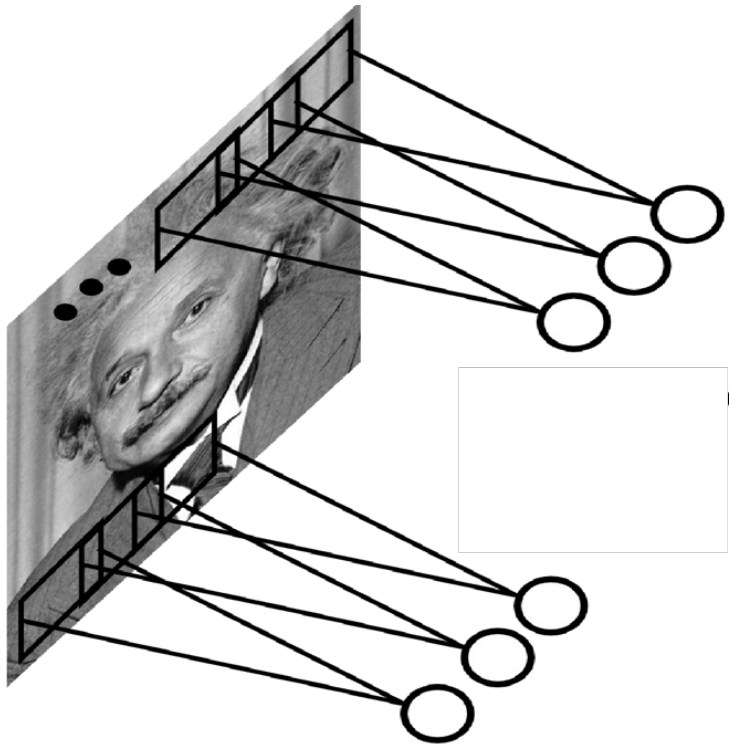
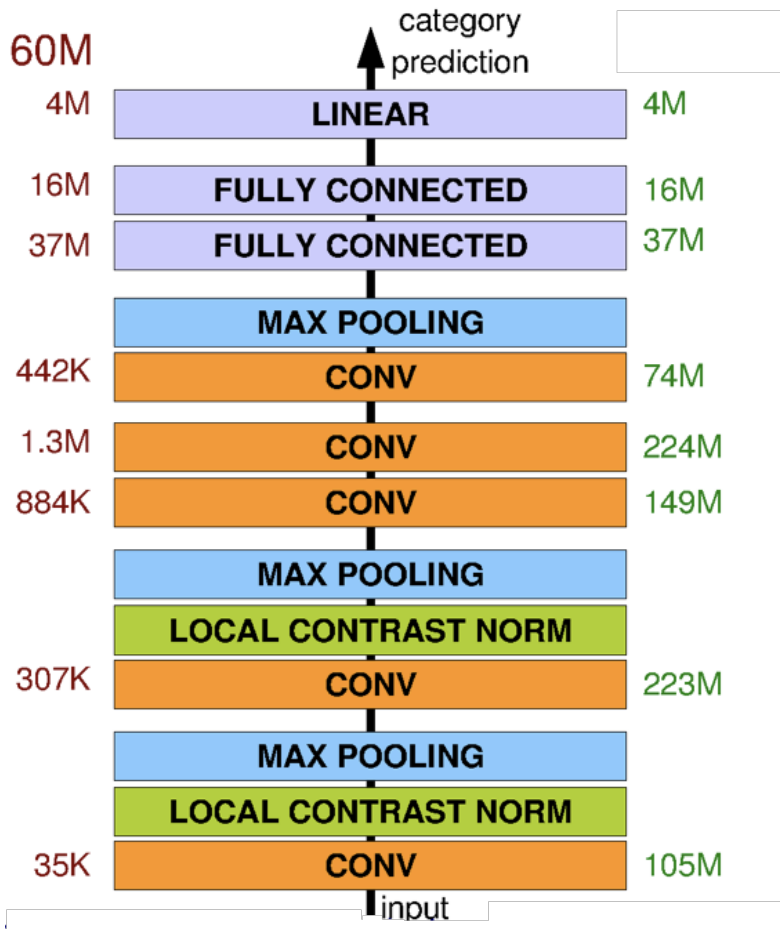


Figure: A convolutional network sharing parameters (figure by Marc Aurelio Ranzato)

- A convolutional network shares the parameter set across different locations.

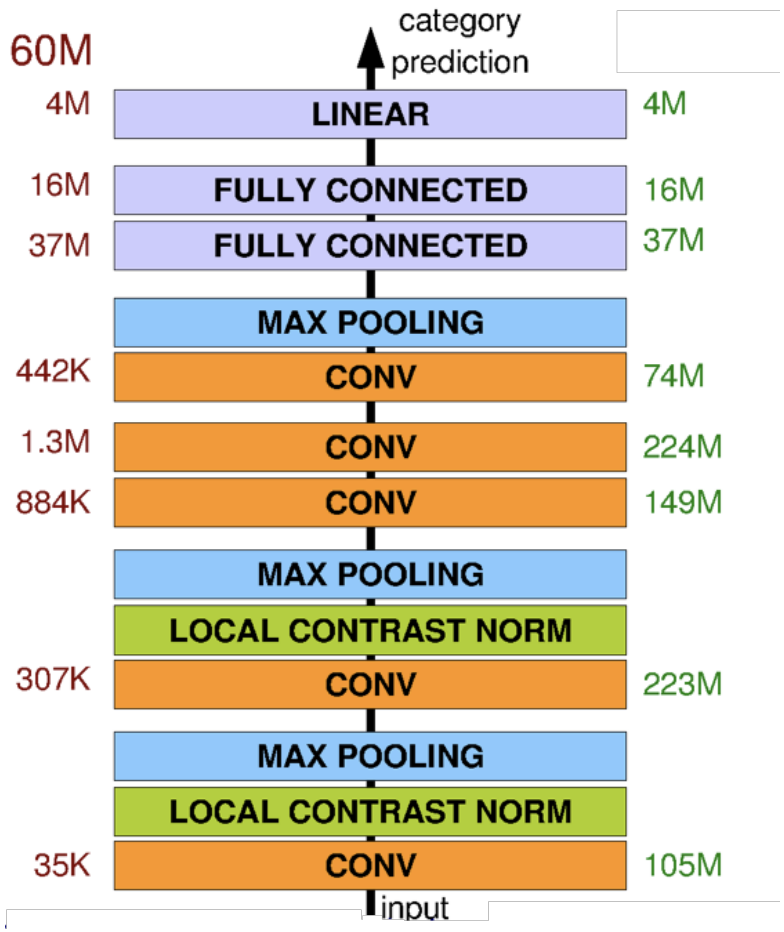
Architecture of a Convolutional network



- A convolutional network comprises of three kinds of layers that are combined in a network.

Figure: Alexnet CNN architecture, fig. credit. Ranzato

Architecture of a Convolutional network



- A convolutional network comprises of three kinds of layers that are combined in a network.
  - Convolutional Layer

Figure: Alexnet CNN architecture, fig. credit. Ranzato

Architecture of a Convolutional network

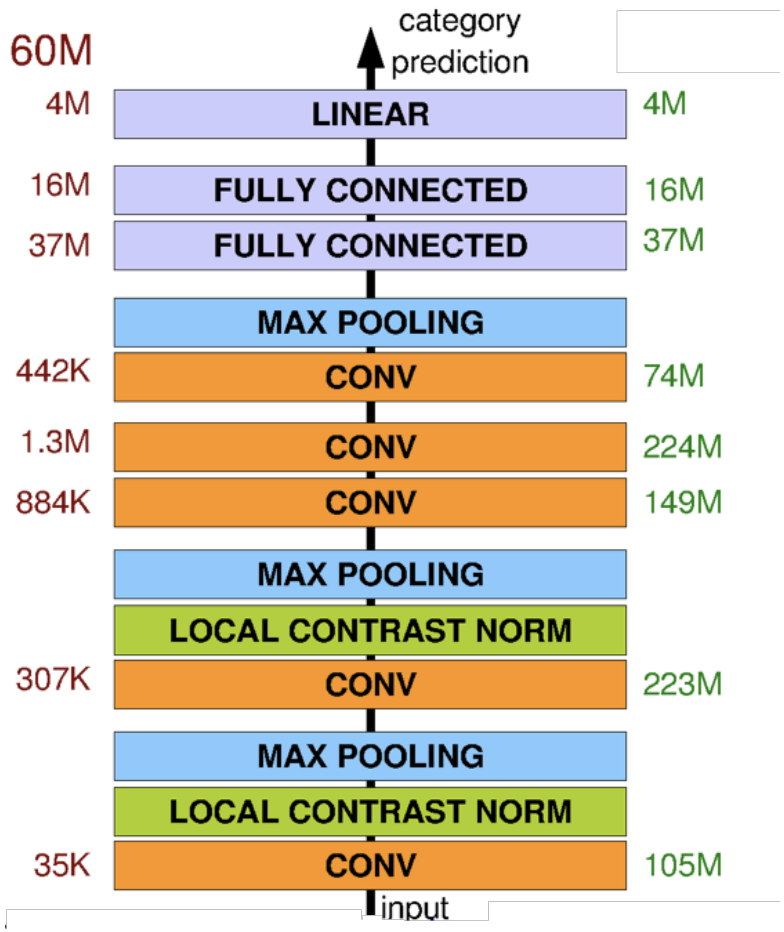


Figure: Alexnet CNN architecture, fig. credit. Ranzato

- A convolutional network comprises of three kinds of layers that are combined in a network.
  - Convolutional Layer
  - Pooling Layer

Architecture of a Convolutional network

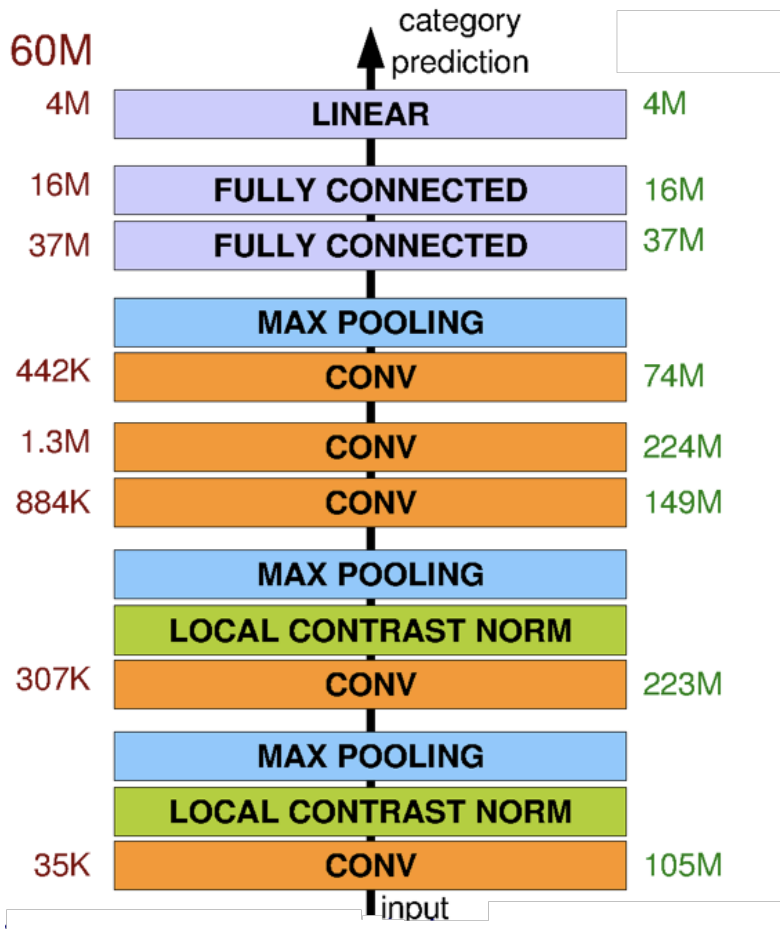


Figure: Alexnet CNN architecture, fig. credit. Ranzato

- A convolutional network comprises of three kinds of layers that are combined in a network.
  - Convolutional Layer
  - Pooling Layer
  - Normalization Layer

Architecture of a Convolutional network

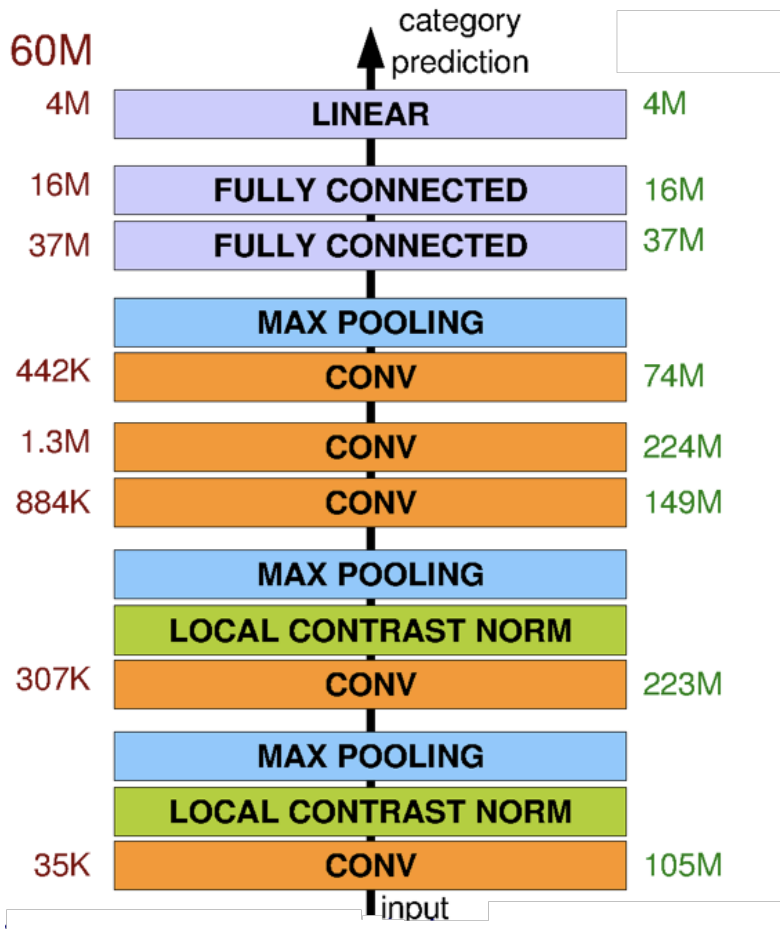


Figure: Alexnet CNN architecture, fig. credit. Ranzato

- A convolutional network comprises of three kinds of layers that are combined in a network.
  - Convolutional Layer
  - Pooling Layer
  - Normalization Layer
- We consider the role of each of these layers in detail

Convolutional Layer

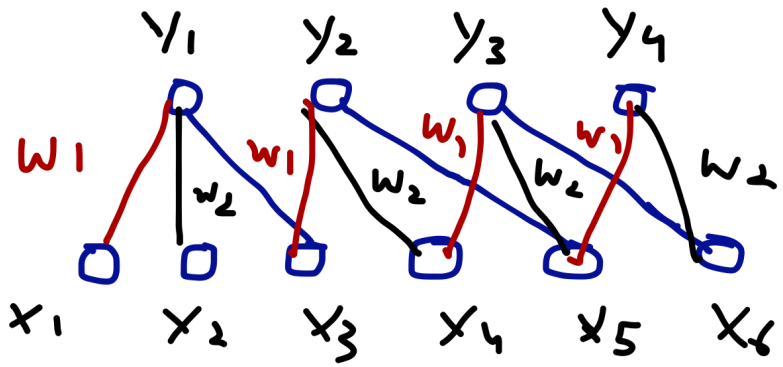


Figure: Illustration of a Convolution layer

- At its simplest form, a convolutional layer comprises a matrix that determines the weights that are used to obtain the output response given the input activations.

Convolutional Layer

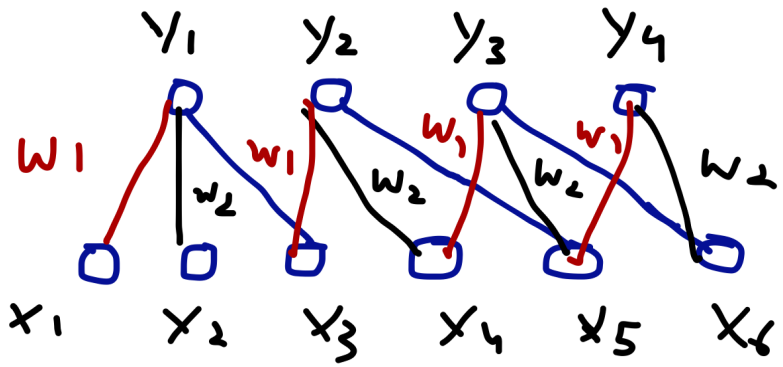


Figure: Illustration of a Convolution layer

- At its simplest form, a convolutional layer comprises a matrix that determines the weights that are used to obtain the output response given the input activations.
- These weights are learned by the network. The figure illustrates that the weights w_1, w_2, w_3 are shared between the input layer $X_1, X_2, \cdot$ and the output layer $Y_1, Y_2, \cdot$

Convolutional Layer

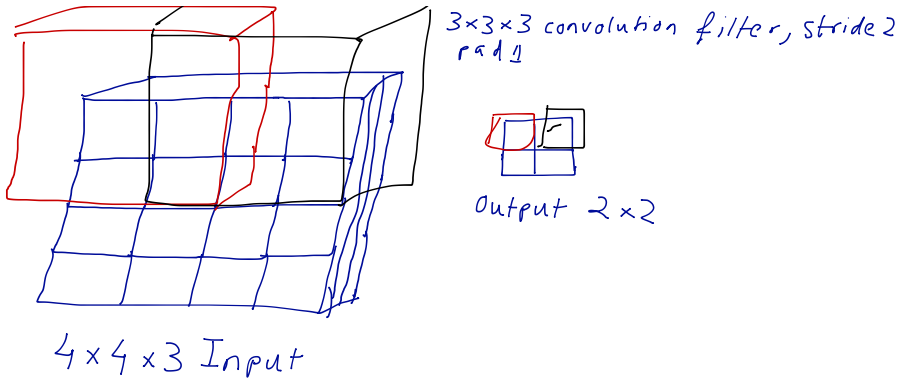


Figure: Illustration of input-output filters. Check out also the interactive demo at http://jsfiddle.net/yces4vn9/43/

- Each convolution layer converts an input 3D matrix to an output 3D matrix.

Convolutional Layer

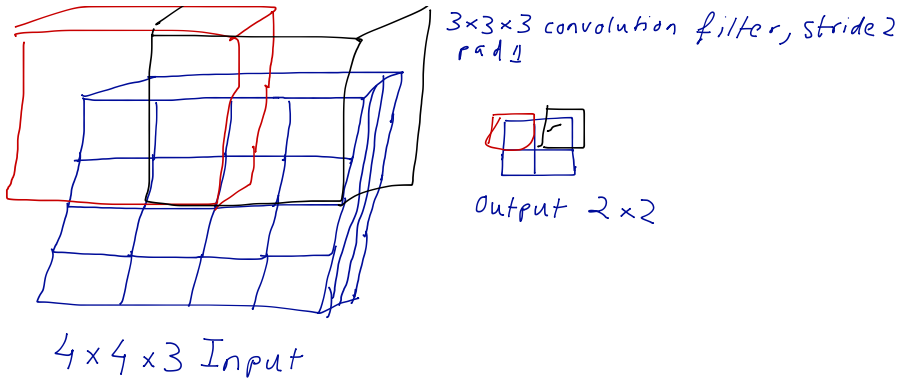


Figure: Illustration of input-output filters. Check out also the interactive demo at http://jsfiddle.net/yces4vn9/43/

- Each convolution layer converts an input 3D matrix to an output 3D matrix.
- The parameters of the layer are width w and height h of the kernel, the input channel depth d_i, the output channel depth d_o (that is obtained through d_o filters), the stride s, zero padding p and the dilation d for the filter.

Pooling Layer

The pooling layer can be either max-pooling or average pooling and are obtained by the following equation

Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

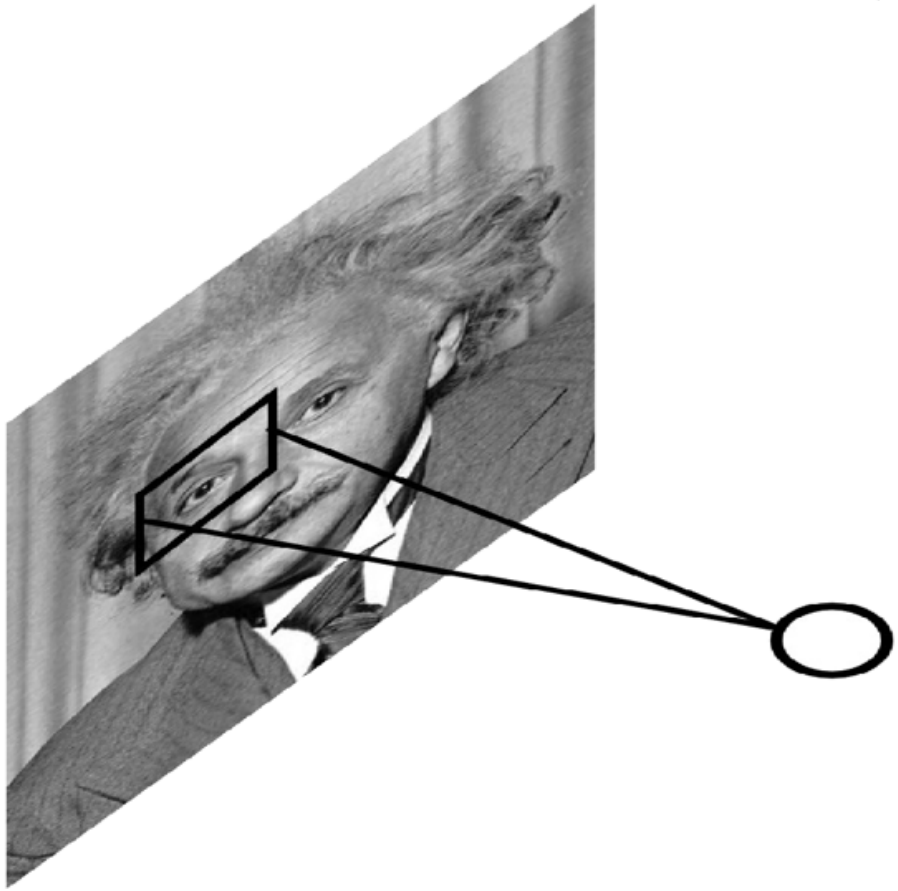
Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

If the convolutional filters have size KxK and stride 1 and pooling layers have pools of size PxP, then each unit in the pooled layer depends upon a patch of size (P+K-1)x(P+K-1). The effect can be visualized through the interactive demo available at http://jsfiddle.net/yces4vn9/43/

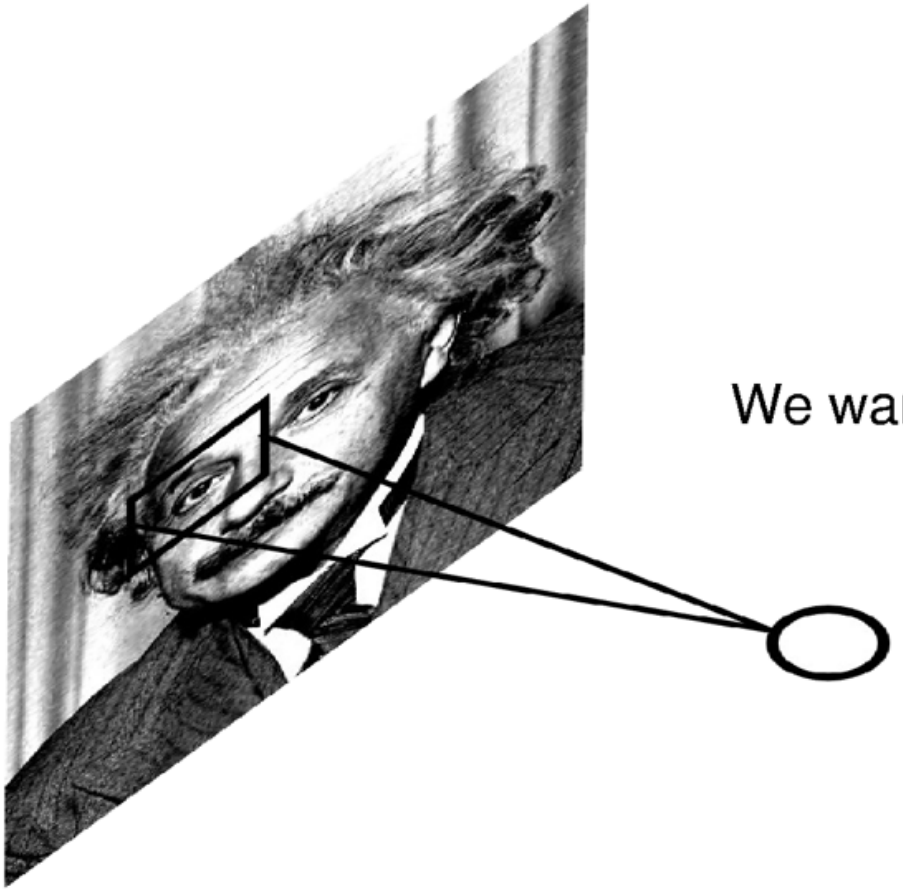
Contrast Normalisation Layer

$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{\sigma^i(N(x,y))}$$



Contrast Normalisation Layer

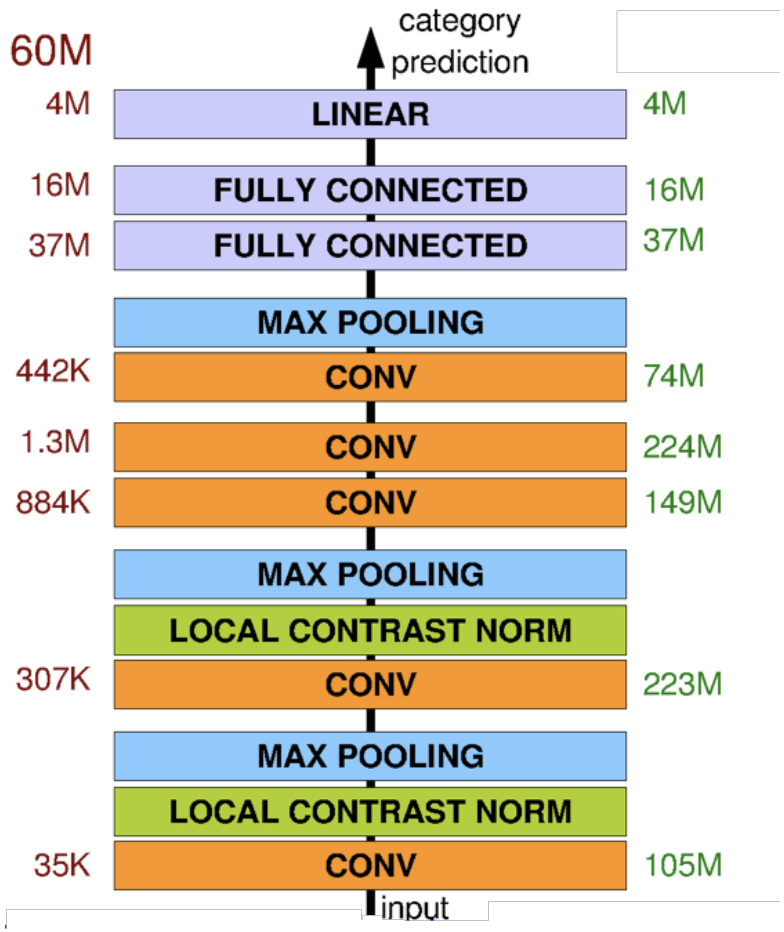
$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{\sigma^i(N(x,y))}$$



We want the same response.

Outline

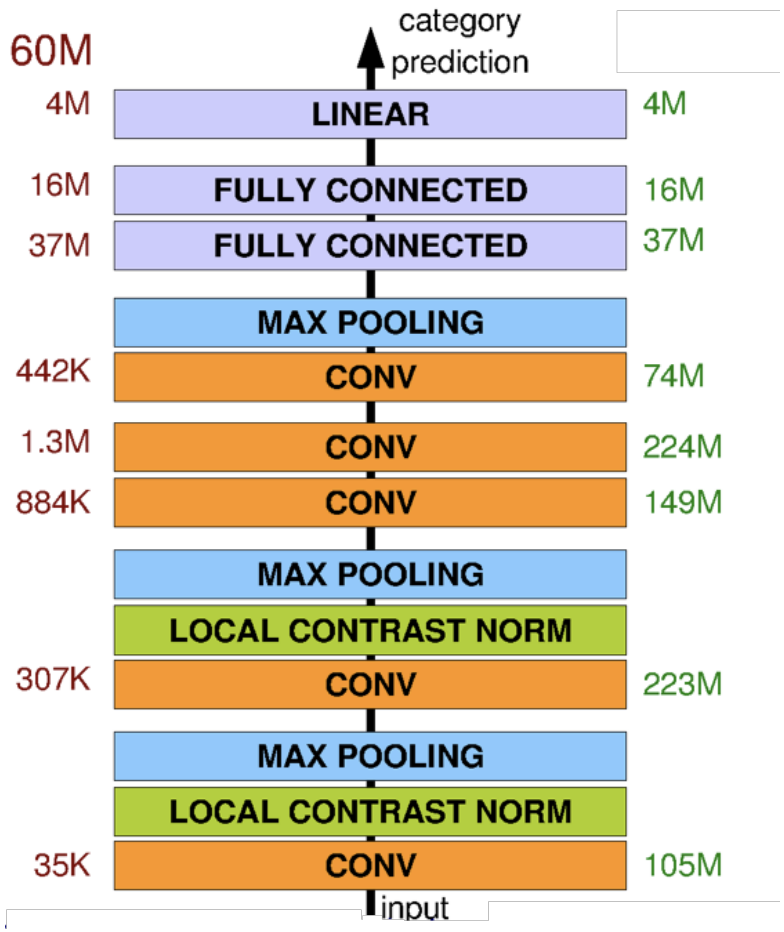
Architecture of a Convolutional network



- A convolutional network comprises of three kinds of layers that are combined in a network.

Figure: Alexnet CNN architecture, fig. credit. Ranzato

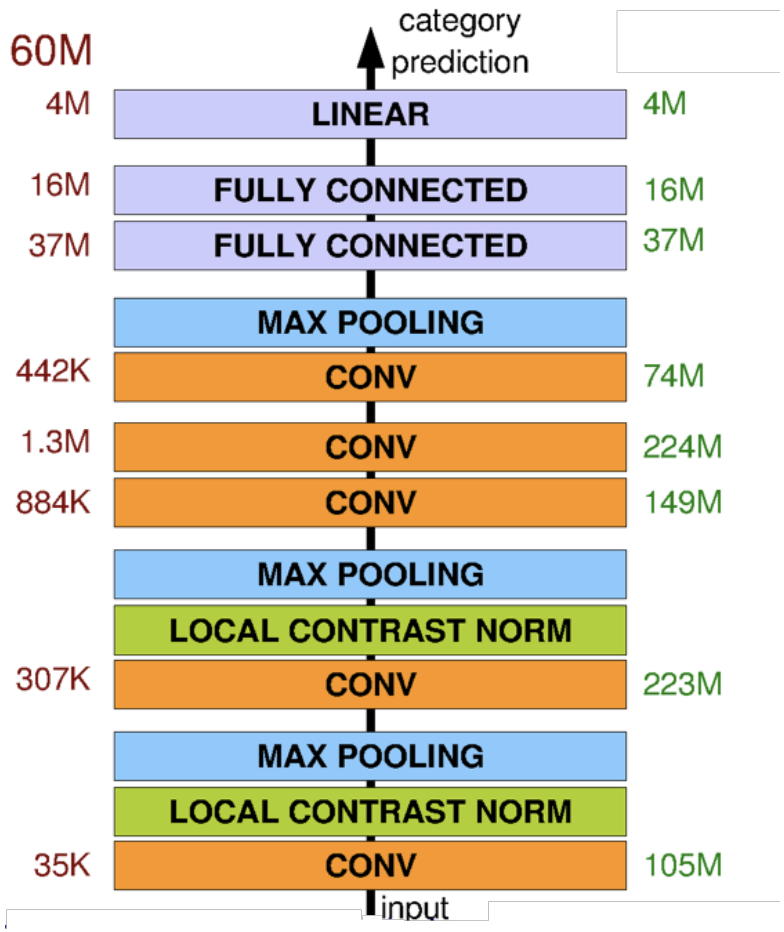
Architecture of a Convolutional network



- A convolutional network comprises of three kinds of layers that are combined in a network.
  - Convolutional Layer

Figure: Alexnet CNN architecture, fig. credit. Ranzato

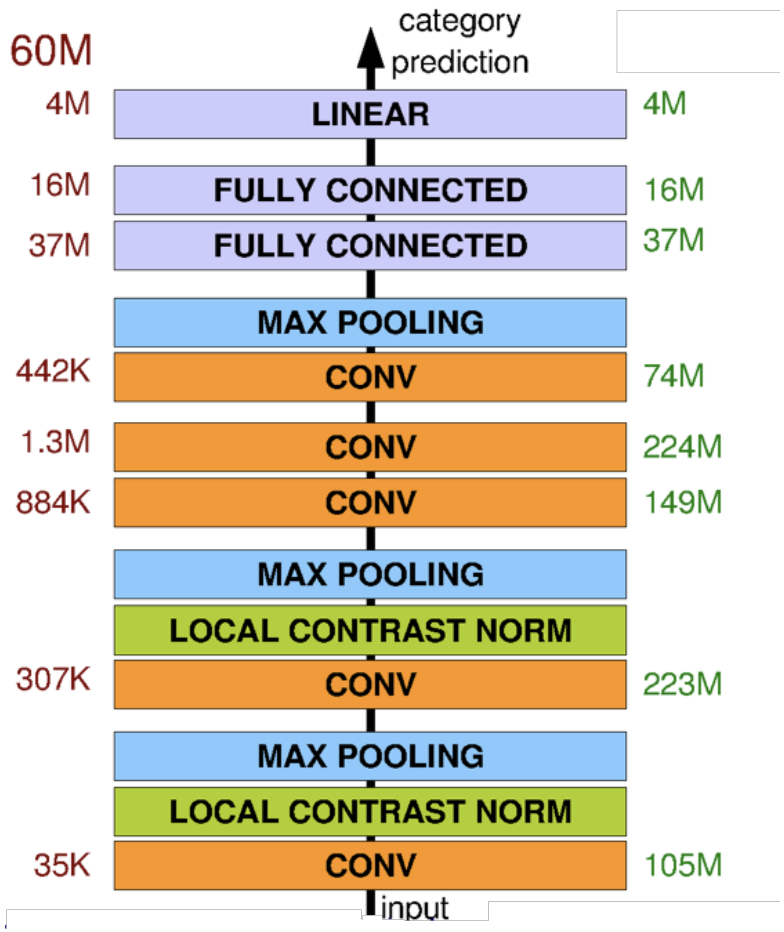
Architecture of a Convolutional network



- A convolutional network comprises of three kinds of layers that are combined in a network.
  - Convolutional Layer
  - Pooling Layer

Figure: Alexnet CNN architecture, fig. credit. Ranzato

Architecture of a Convolutional network



- A convolutional network comprises of three kinds of layers that are combined in a network.
  - Convolutional Layer
  - Pooling Layer
  - Normalization Layer

Figure: Alexnet CNN architecture, fig. credit. Ranzato

Architecture of a Convolutional network

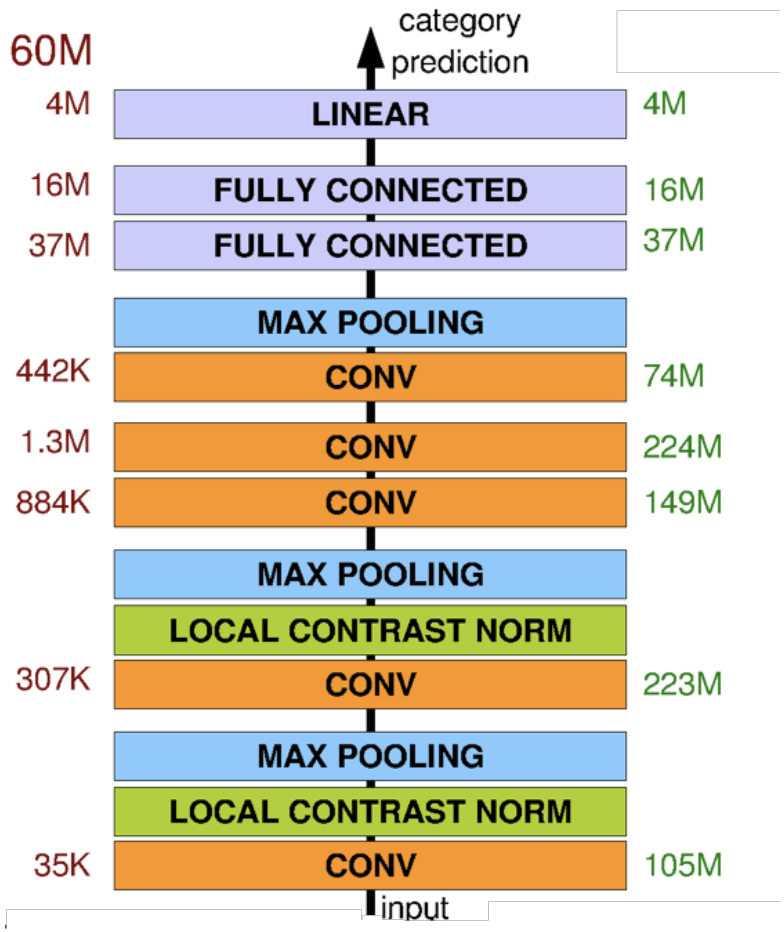


Figure: Alexnet CNN architecture, fig. credit. Ranzato

- A convolutional network comprises of three kinds of layers that are combined in a network.
  - Convolutional Layer
  - Pooling Layer
  - Normalization Layer
- We now consider the architectural variants for various network before considering other aspects for convolutional networks.

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96
  11x11 filters at stride 4, pad
  0

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96
  11x11 filters at stride 4, pad
  0
- (27x27x96) MAX POOL1:
  3x3 filters at stride 2

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96
  11x11 filters at stride 4, pad
  0
- (27x27x96) MAX POOL1:
  3x3 filters at stride 2
- (27x27x96) NORM1:
  Normalization layer

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96
  11x11 filters at stride 4, pad
  0
- (27x27x96) MAX POOL1:
  3x3 filters at stride 2
- (27x27x96) NORM1:
  Normalization layer
- (27x27x256) CONV2: 256
  5x5 filters at stride 1, pad 2

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96
  11x11 filters at stride 4, pad
  0
- (27x27x96) MAX POOL1:
  3x3 filters at stride 2
- (27x27x96) NORM1:
  Normalization layer
- (27x27x256) CONV2: 256
  5x5 filters at stride 1, pad 2
- (13x13x256) MAX POOL2:
  3x3 filters at stride 2

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96
  11x11 filters at stride 4, pad
  0
- (27x27x96) MAX POOL1:
  3x3 filters at stride 2
- (27x27x96) NORM1:
  Normalization layer
- (27x27x256) CONV2: 256
  5x5 filters at stride 1, pad 2
- (13x13x256) MAX POOL2:
  3x3 filters at stride 2
- (13x13x256)] NORM2:

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96
  11x11 filters at stride 4, pad 0
- (27x27x96) MAX POOL1:
  3x3 filters at stride 2
- (27x27x96) NORM1:
  Normalization layer
- (27x27x256) CONV2: 256
  5x5 filters at stride 1, pad 2
- (13x13x256) MAX POOL2:
  3x3 filters at stride 2
- (13x13x256)] NORM2:
- (13x13x384) CONV3: 384
  3x3 filters at stride 1, pad 1

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96 11x11 filters at stride 4, pad 0
- (27x27x96) MAX POOL1: 3x3 filters at stride 2
- (27x27x96) NORM1: Normalization layer
- (27x27x256) CONV2: 256 5x5 filters at stride 1, pad 2
- (13x13x256) MAX POOL2: 3x3 filters at stride 2
- (13x13x256)] NORM2:
- (13x13x384) CONV3: 384 3x3 filters at stride 1, pad 1
- (13x13x384) CONV4: 384 3x3 filters at stride 1, pad 1

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96 11x11 filters at stride 4, pad 0
- (27x27x96) MAX POOL1: 3x3 filters at stride 2
- (27x27x96) NORM1: Normalization layer
- (27x27x256) CONV2: 256 5x5 filters at stride 1, pad 2
- (13x13x256) MAX POOL2: 3x3 filters at stride 2
- (13x13x256)] NORM2:
- (13x13x384) CONV3: 384 3x3 filters at stride 1, pad 1
- (13x13x384) CONV4: 384 3x3 filters at stride 1, pad 1
- (13x13x256) CONV5: 256 3x3 filters at stride 1, pad 1

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96 11x11 filters at stride 4, pad 0
- (27x27x96) MAX POOL1: 3x3 filters at stride 2
- (27x27x96) NORM1: Normalization layer
- (27x27x256) CONV2: 256 5x5 filters at stride 1, pad 2
- (13x13x256) MAX POOL2: 3x3 filters at stride 2
- (13x13x256)] NORM2:
- (13x13x384) CONV3: 384 3x3 filters at stride 1, pad 1
- (13x13x384) CONV4: 384 3x3 filters at stride 1, pad 1
- (13x13x256) CONV5: 256 3x3 filters at stride 1, pad 1
- (6x6x256) MAX POOL3: 3x3 filters at stride 2

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96 11x11 filters at stride 4, pad 0
- (27x27x96) MAX POOL1: 3x3 filters at stride 2
- (27x27x96) NORM1: Normalization layer
- (27x27x256) CONV2: 256 5x5 filters at stride 1, pad 2
- (13x13x256) MAX POOL2: 3x3 filters at stride 2
- (13x13x256)] NORM2:
- (13x13x384) CONV3: 384 3x3 filters at stride 1, pad 1
- (13x13x384) CONV4: 384 3x3 filters at stride 1, pad 1
- (13x13x256) CONV5: 256 3x3 filters at stride 1, pad 1
- (6x6x256) MAX POOL3: 3x3 filters at stride 2
- (4096) FC6: 4096 neurons

Alex Net Architecture

The Alex Net Architecture in detail is as follows

- (227x227x3) INPUT
- (55x55x96) CONV1: 96 11x11 filters at stride 4, pad 0
- (27x27x96) MAX POOL1: 3x3 filters at stride 2
- (27x27x96) NORM1: Normalization layer
- (27x27x256) CONV2: 256 5x5 filters at stride 1, pad 2
- (13x13x256) MAX POOL2: 3x3 filters at stride 2
- (13x13x256)] NORM2:
- (13x13x384) CONV3: 384 3x3 filters at stride 1, pad 1
- (13x13x384) CONV4: 384 3x3 filters at stride 1, pad 1
- (13x13x256) CONV5: 256 3x3 filters at stride 1, pad 1
- (6x6x256) MAX POOL3: 3x3 filters at stride 2
- (4096) FC6: 4096 neurons
- (4096) FC7: 4096 neurons

Alex Net Architecture

The Alex Net Architecture in detail is as follows

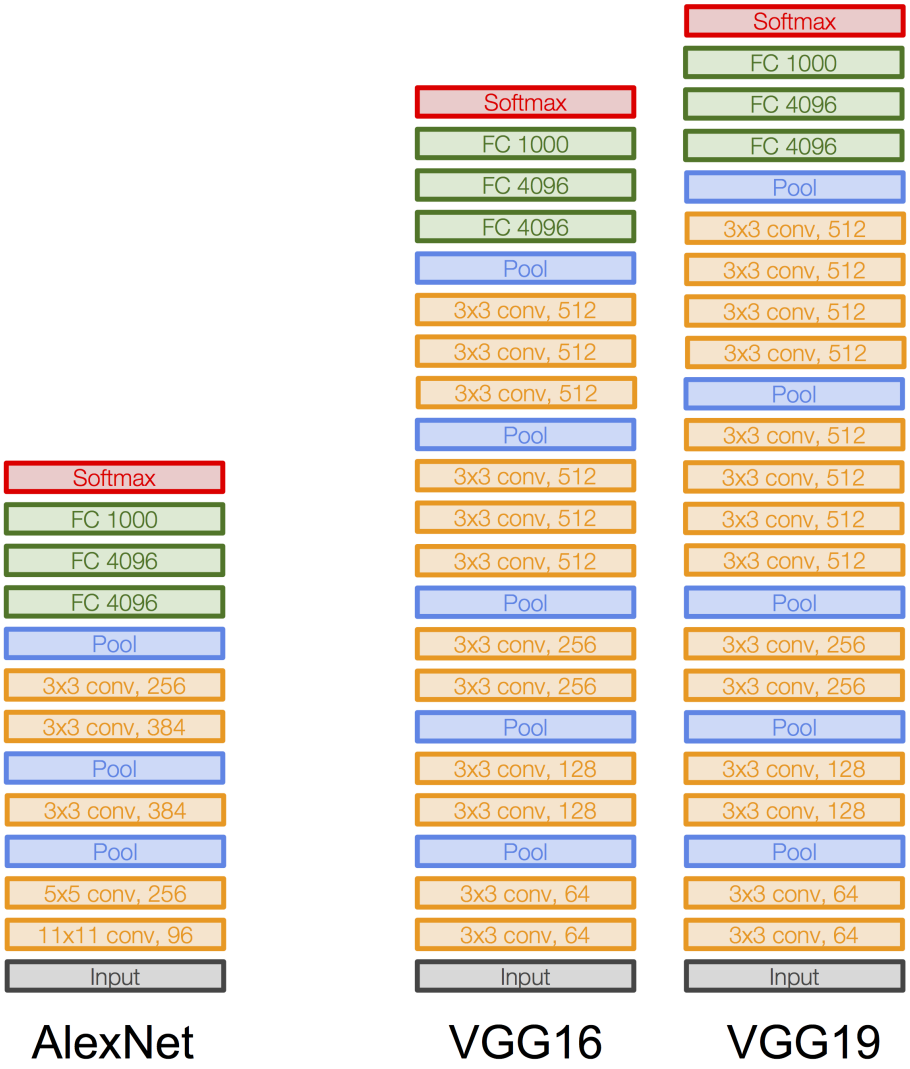
- (227x227x3) INPUT
- (55x55x96) CONV1: 96 11x11 filters at stride 4, pad 0
- (27x27x96) MAX POOL1: 3x3 filters at stride 2
- (27x27x96) NORM1: Normalization layer
- (27x27x256) CONV2: 256 5x5 filters at stride 1, pad 2
- (13x13x256) MAX POOL2: 3x3 filters at stride 2
- (13x13x256)] NORM2:
- (13x13x384) CONV3: 384 3x3 filters at stride 1, pad 1
- (13x13x384) CONV4: 384 3x3 filters at stride 1, pad 1
- (13x13x256) CONV5: 256 3x3 filters at stride 1, pad 1
- (6x6x256) MAX POOL3: 3x3 filters at stride 2
- (4096) FC6: 4096 neurons
- (4096) FC7: 4096 neurons
- (1000) FC8: 1000 neurons (class scores)

Other Architectures

There has been since Alexnet a number of architectures with increased depth

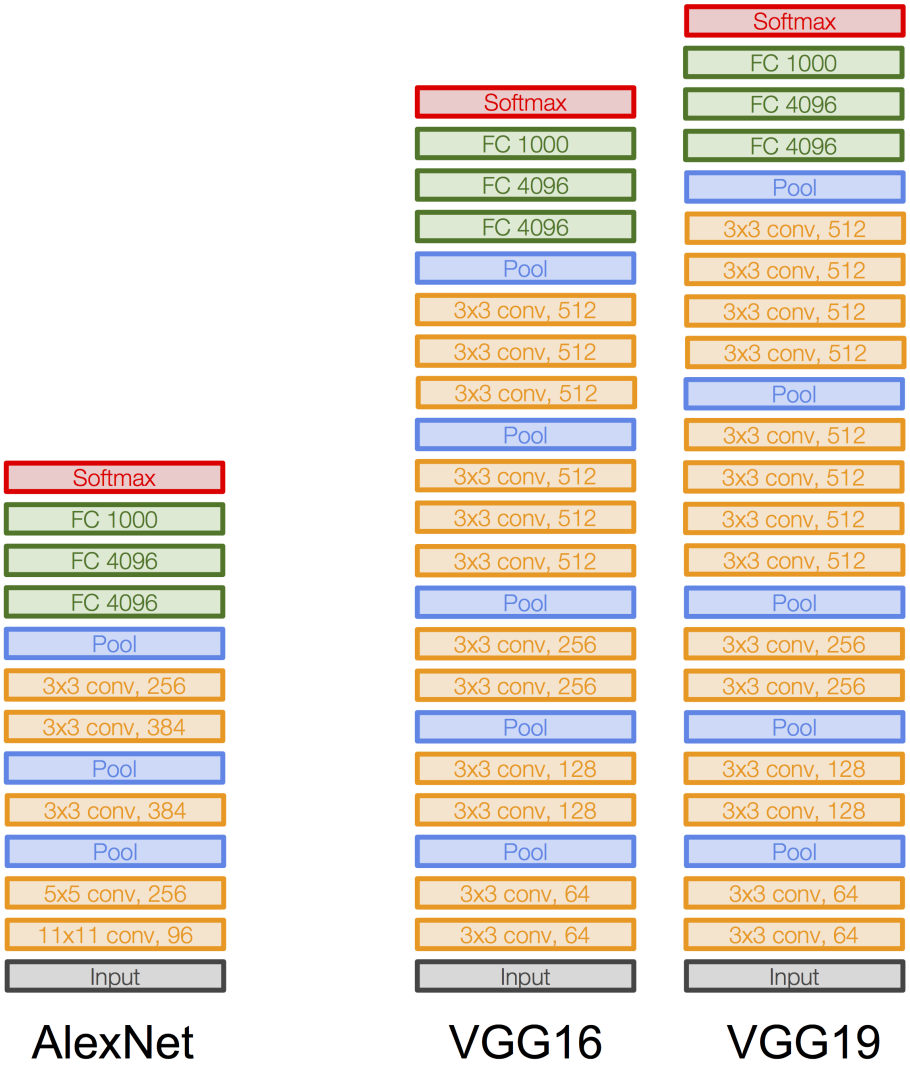
ILSVRC Year	Architecture	Depth	Top 5% Error
ILSVRC10	Shallow	-	28.2
ILSVRC11	Shallow	-	25.8
ILSVRC12	AlexNet	8 layers	16.4
ILSVRC 13	- Clarifai	8 layers	11.7
ILSVRC14	VGG	19 layers	7.3
ILSVRC14	GoogleNet	22 layers	6.7
ILSVRC15	ResNet	152 layers	3.57

VGG Deep Net



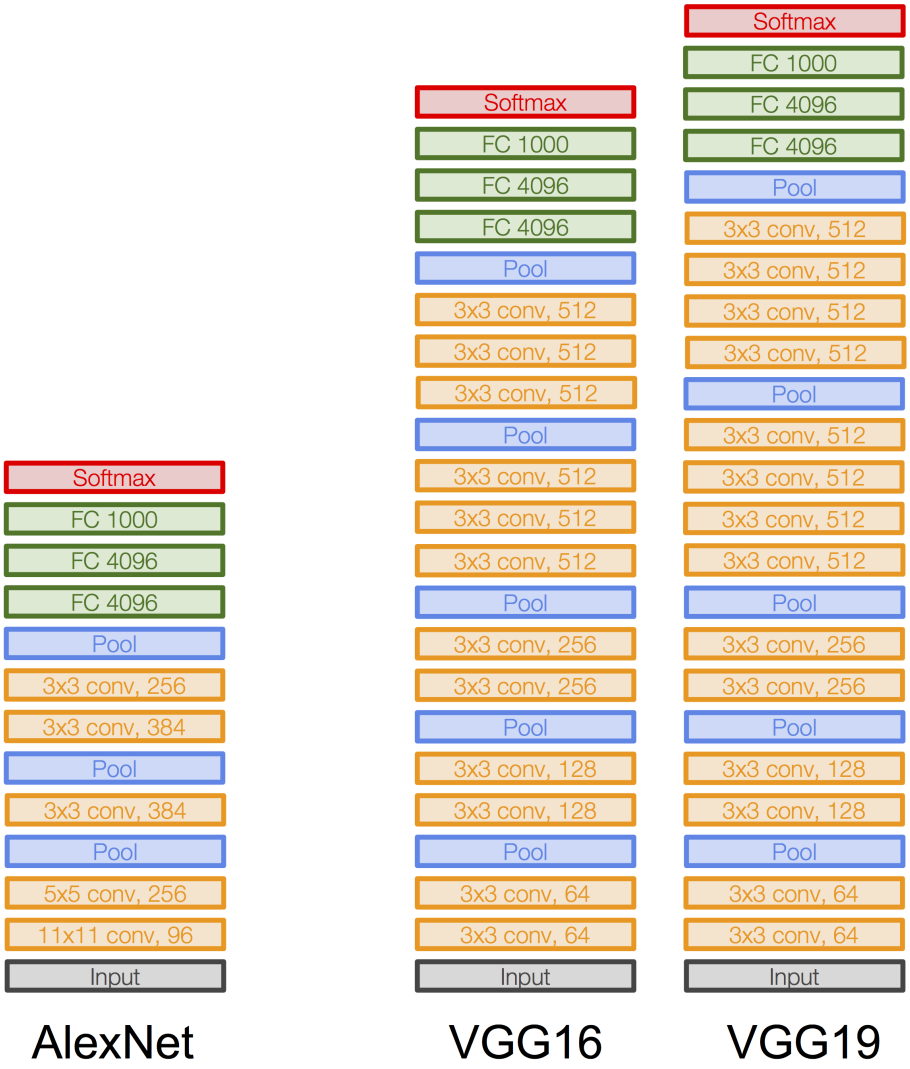
- VGG Net (by Simonyan and Zisserman) had few changes from the original AlexNet.

VGG Deep Net



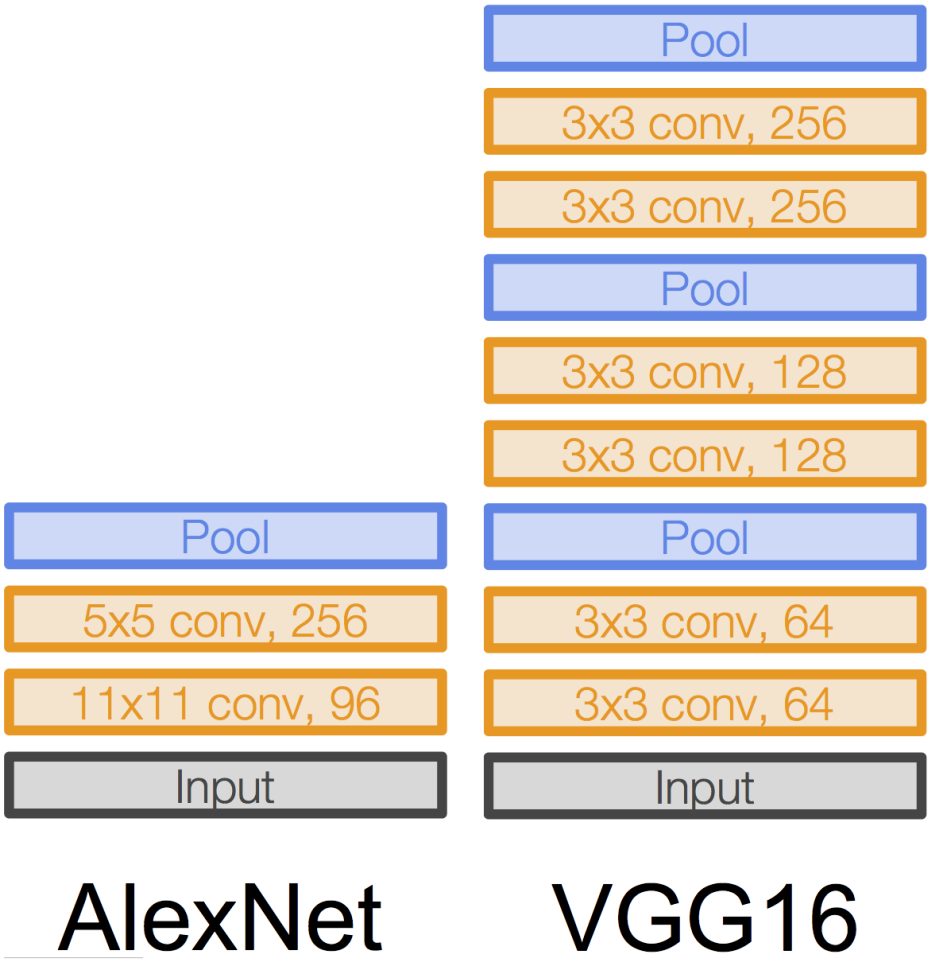
- VGG Net (by Simonyan and Zisserman) had few changes from the original AlexNet.
- The main change was that instead of using broad filters (for instance 11x11) used by AlexNet, they used smaller convolutional filters of (3x3)

VGG Deep Net



- VGG Net (by Simonyan and Zisserman) had few changes from the original AlexNet.
- The main change was that instead of using broad filters (for instance 11x11) used by AlexNet, they used smaller convolutional filters of (3x3)
- They obtained a deeper net using this strategy

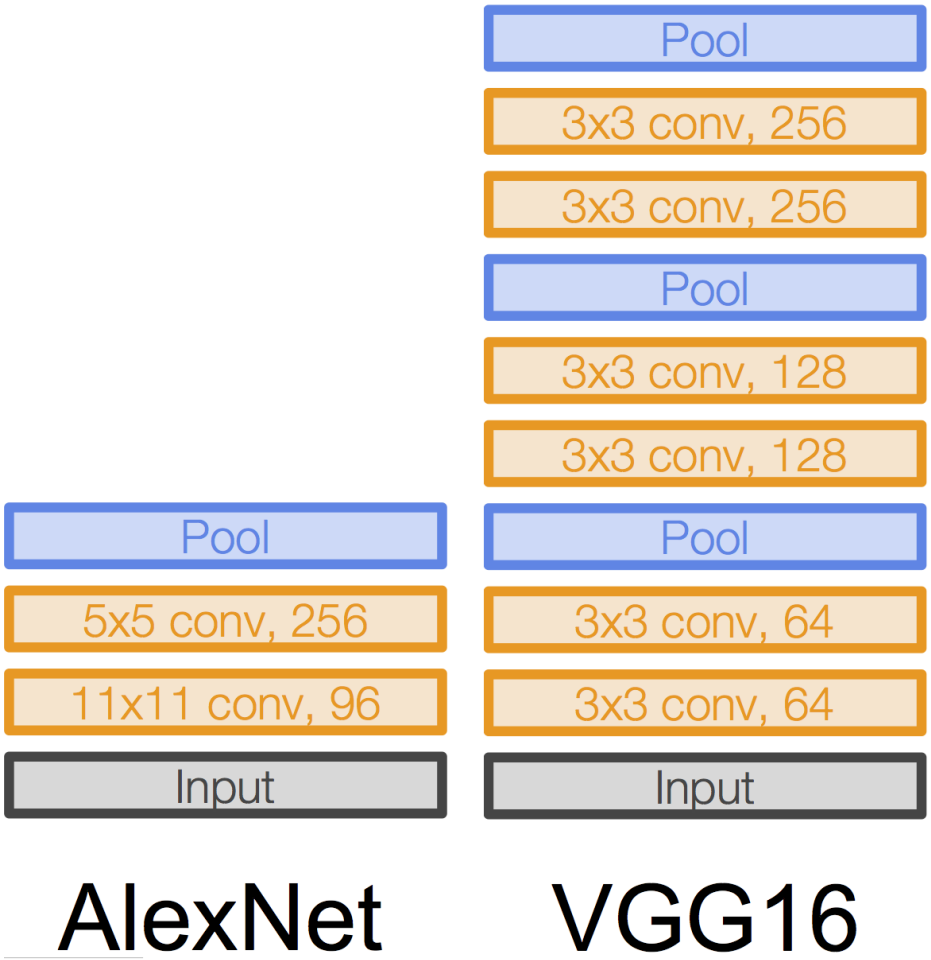
VGG Deep Net



- The reason for this change is, the effective receptive field of a 7x7 convolutional filter can be obtained by stacking multiple convolutional filters with receptive field size of 3x3

Figure: VGG Net- a closer look at the

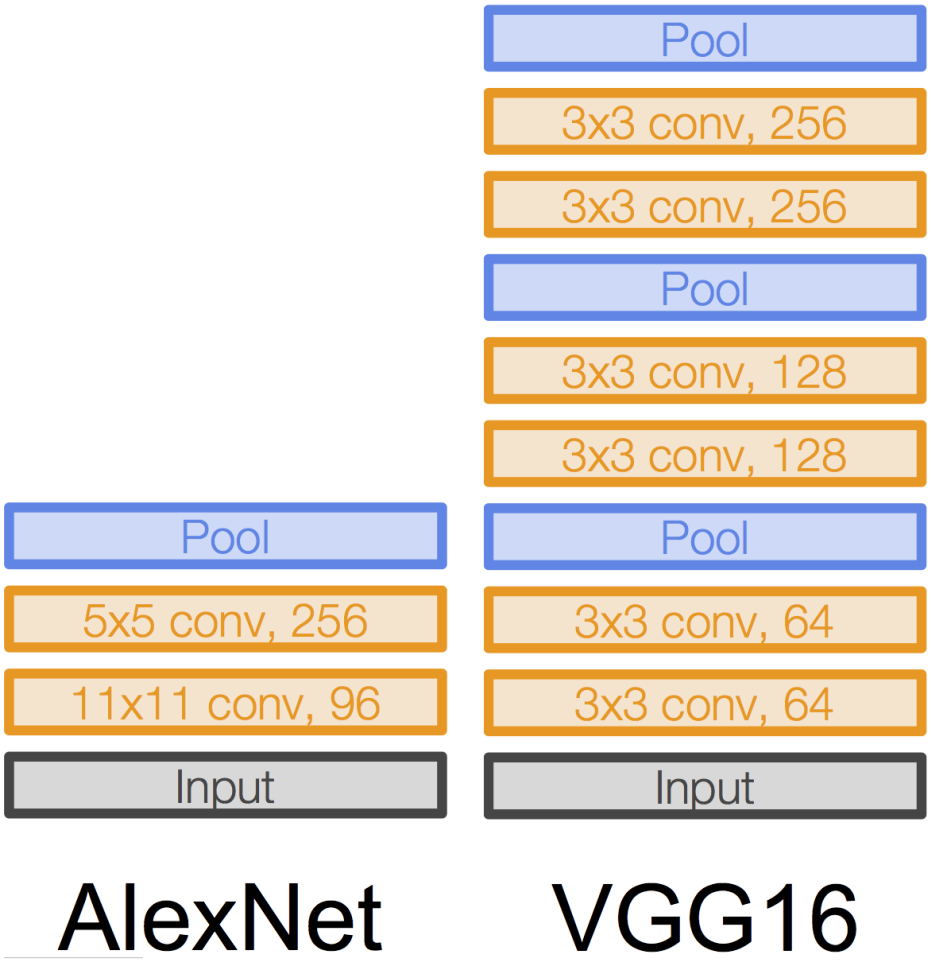
VGG Deep Net



- The reason for this change is, the effective receptive field of a 7x7 convolutional filter can be obtained by stacking multiple convolutional filters with receptive field size of 3x3
- The deeper network obtained by stacking multiple convolutional filters also has a smaller number of parameters

Figure: VGG Net- a closer look at the

VGG Deep Net



- The reason for this change is, the effective receptive field of a 7x7 convolutional filter can be obtained by stacking multiple convolutional filters with receptive field size of 3x3
- The deeper network obtained by stacking multiple convolutional filters also has a smaller number of parameters
- The deeper network however has more non-linearities

Figure: VGG Net- a closer look at the

GoogleNet Inception Network

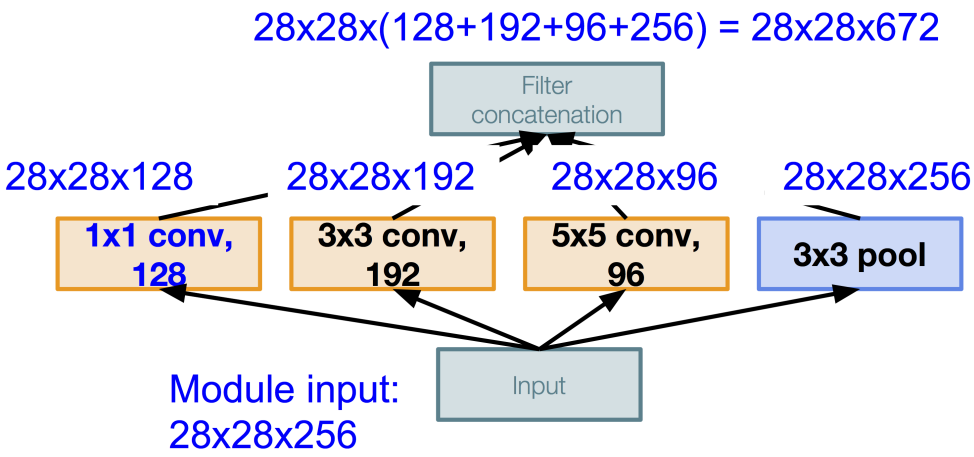


Figure: GoogleNet- a closer look at the initial layers, figure courtesy cs231n Stanford

- The main idea of the inception network architecture is to obtain the optimal local sparse structure by using multiple dense components.

GoogleNet Inception Network

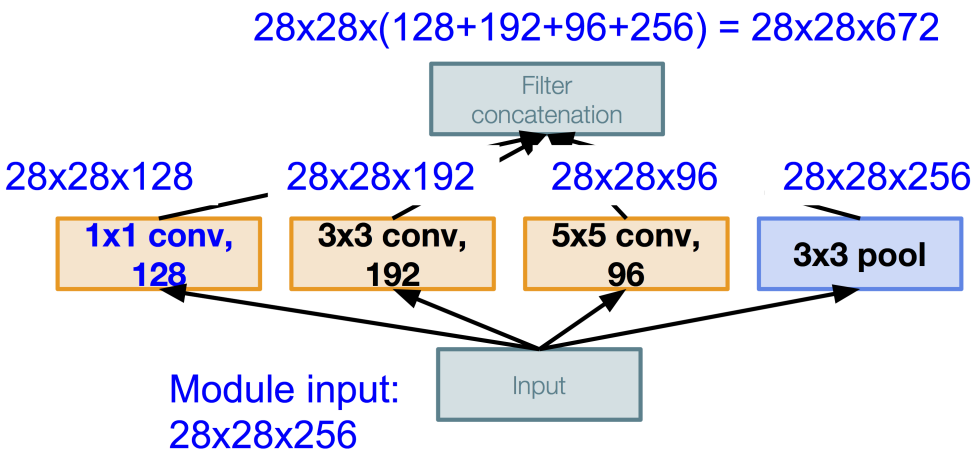


Figure: GoogleNet- a closer look at the initial layers, figure courtesy cs231n Stanford

- The main idea of the inception network architecture is to obtain the optimal local sparse structure by using multiple dense components.
- This is obtained by examining the network using multiple scales locally at a block

GoogleNet Inception Network

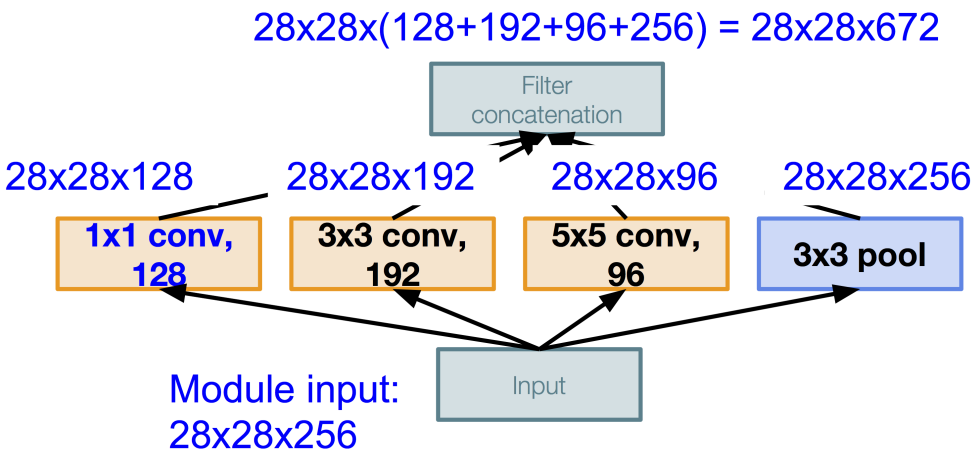


Figure: GoogleNet- a closer look at the initial layers, figure courtesy cs231n Stanford

- The main idea of the inception network architecture is to obtain the optimal local sparse structure by using multiple dense components.
- This is obtained by examining the network using multiple scales locally at a block
- The deep network is then obtained by stacking these blocks

GoogleNet Inception Network

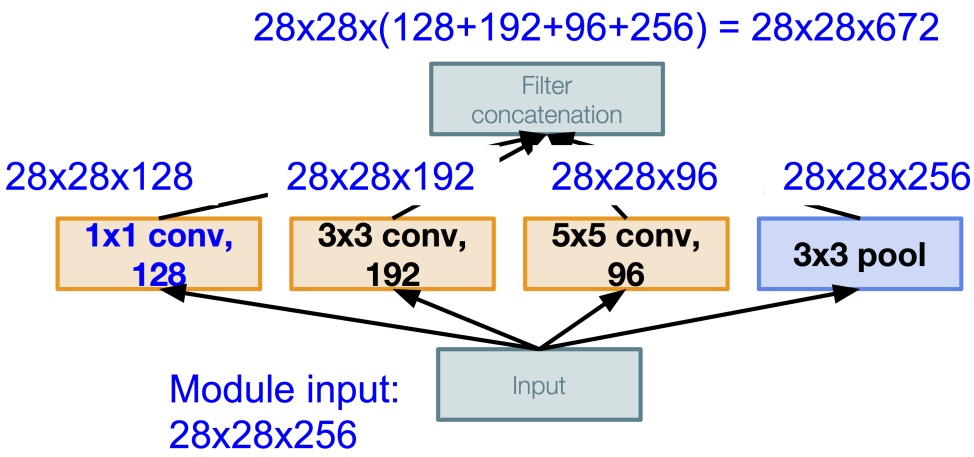


Figure: GoogleNet- inception block, figure courtesy cs231n Stanford

- The main idea of the inception network architecture is to obtain the optimal local sparse structure by using multiple dense components.

GoogleNet Inception Network

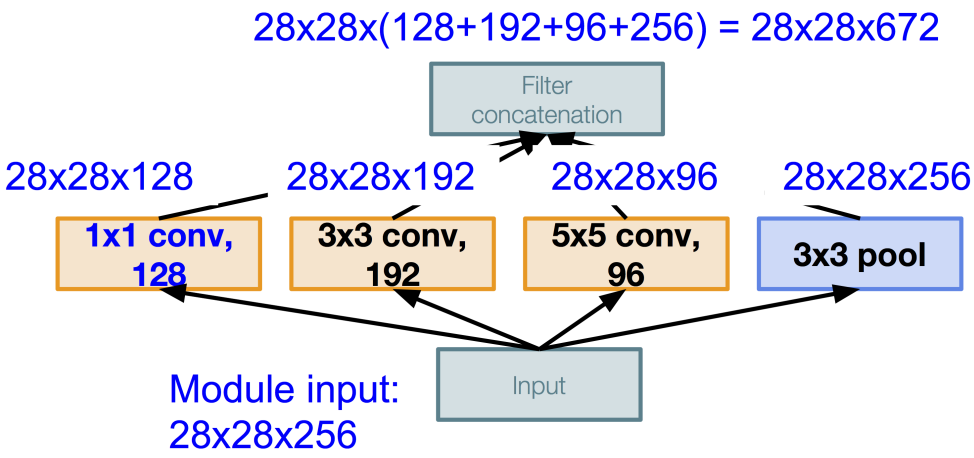


Figure: GoogleNet- inception block, figure courtesy cs231n Stanford

- The main idea of the inception network architecture is to obtain the optimal local sparse structure by using multiple dense components.
- This is obtained by examining the network using multiple scales locally at a block

GoogleNet Inception Network

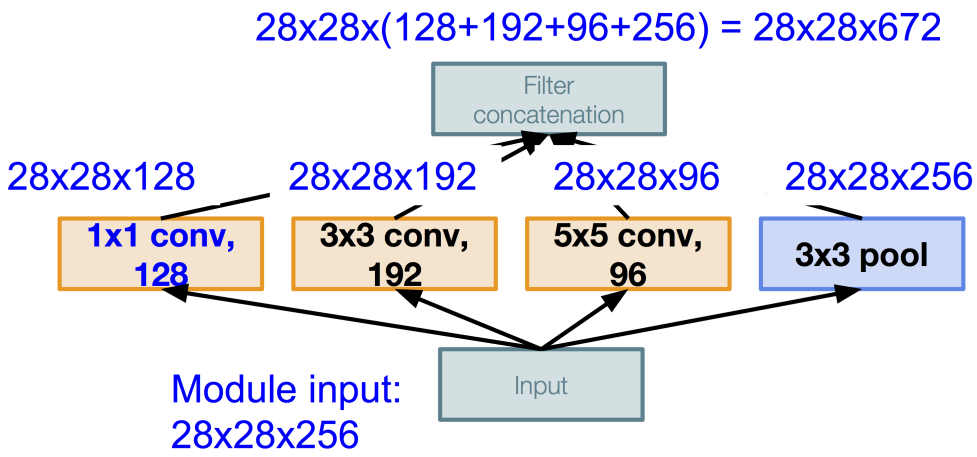


Figure: GoogleNet- inception block, figure courtesy cs231n Stanford

- The main idea of the inception network architecture is to obtain the optimal local sparse structure by using multiple dense components.
- This is obtained by examining the network using multiple scales locally at a block
- The deep network is then obtained by stacking these blocks

GoogleNet Inception Network

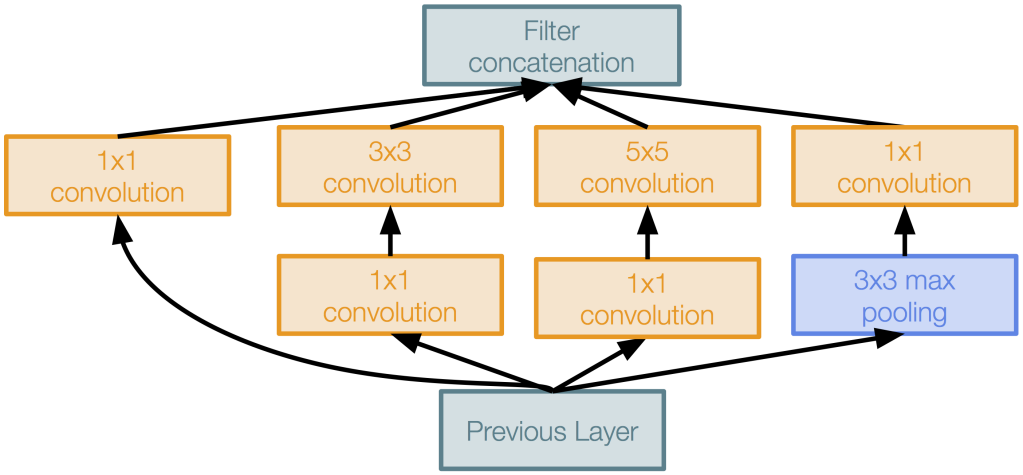


Figure: GoogleNet- inception block with dimensionality reduction, figure courtesy cs231n Stanford

- The resulting filters obtained by concatenating the output is quite large

GoogleNet Inception Network

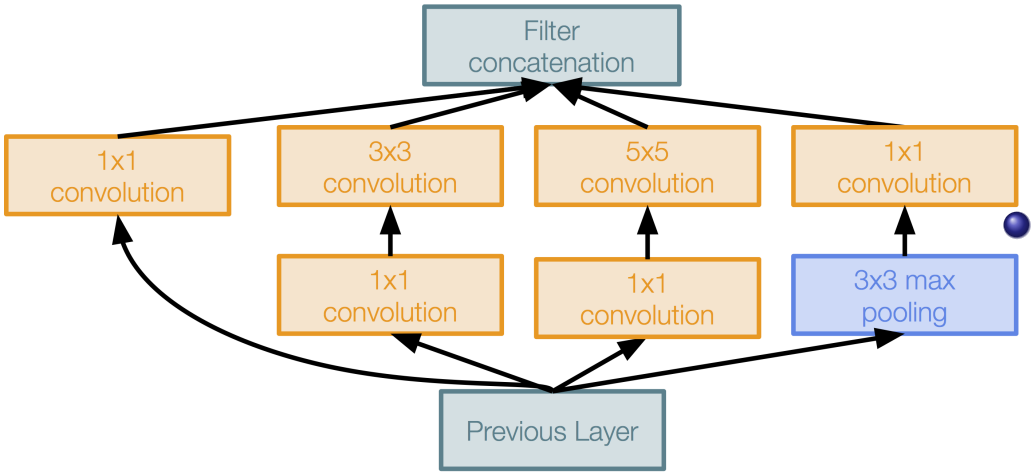


Figure: GoogleNet- inception block with dimensionality reduction, figure courtesy cs231n Stanford

- The resulting filters obtained by concatenating the output is quite large
- The design adopted in the inception architecture uses 1x1 convolutional blocks to reduce the size of the network

GoogleNet Inception Network

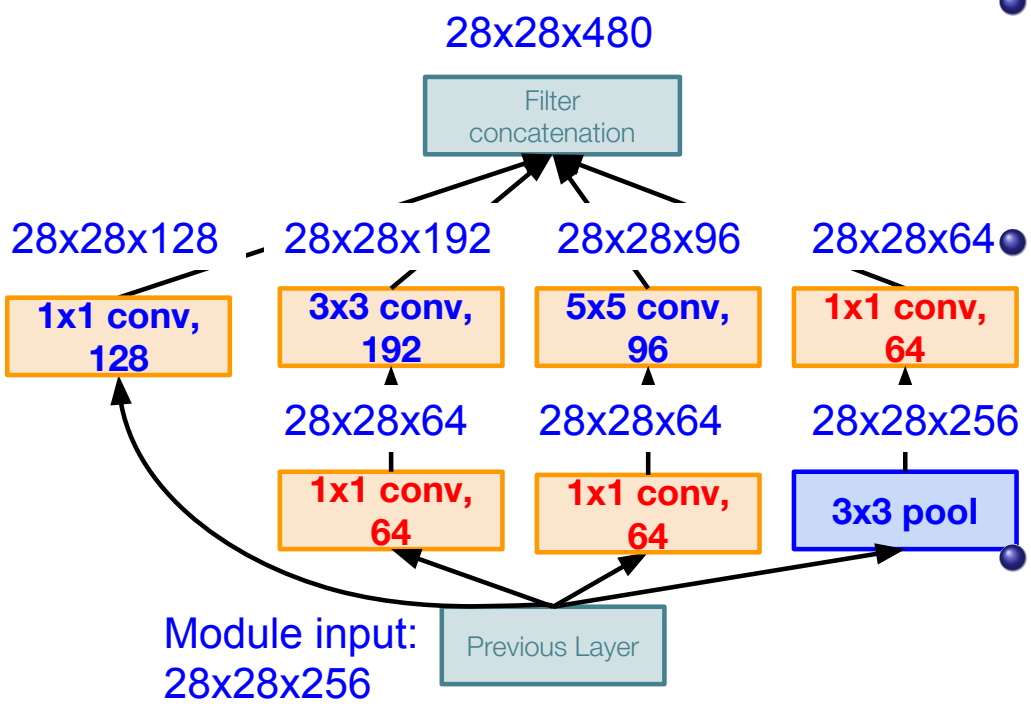


Figure: GoogleNet- inception block with dimensionality reduction, figure courtesy cs231n Stanford

- The resulting filters obtained by concatenating the output is quite large
- The design adopted in the inception architecture uses 1x1 convolutional blocks to reduce the size of the network
- The resulting network has a smaller output dimension as can be observed by looking at the detailed resultant numbers

ResNet Architecture - He et al

He et al investigated the question as to what would happen in a 'plain' convolutional network if we continued stacking layers and made it deeper.

Figure: Evaluation of deeper plain networks in CIFAR dataset by He et al

ResNet Architecture - He et al

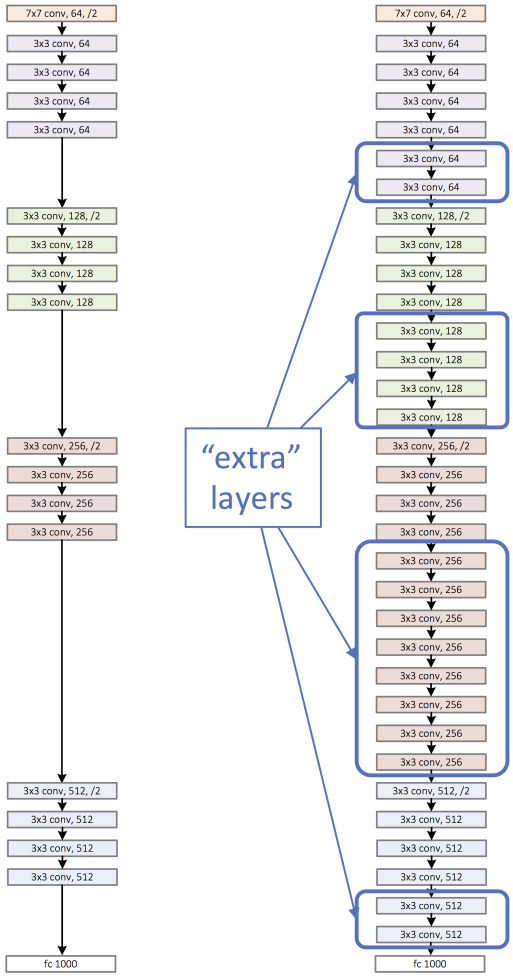
He et al investigated the question as to what would happen in a 'plain' convolutional network if we continued stacking layers and made it deeper.

Figure: Evaluation of deeper plain networks in CIFAR dataset by He et al

- They observed that both the training and test error increased.

ResNet Architecture - He et al

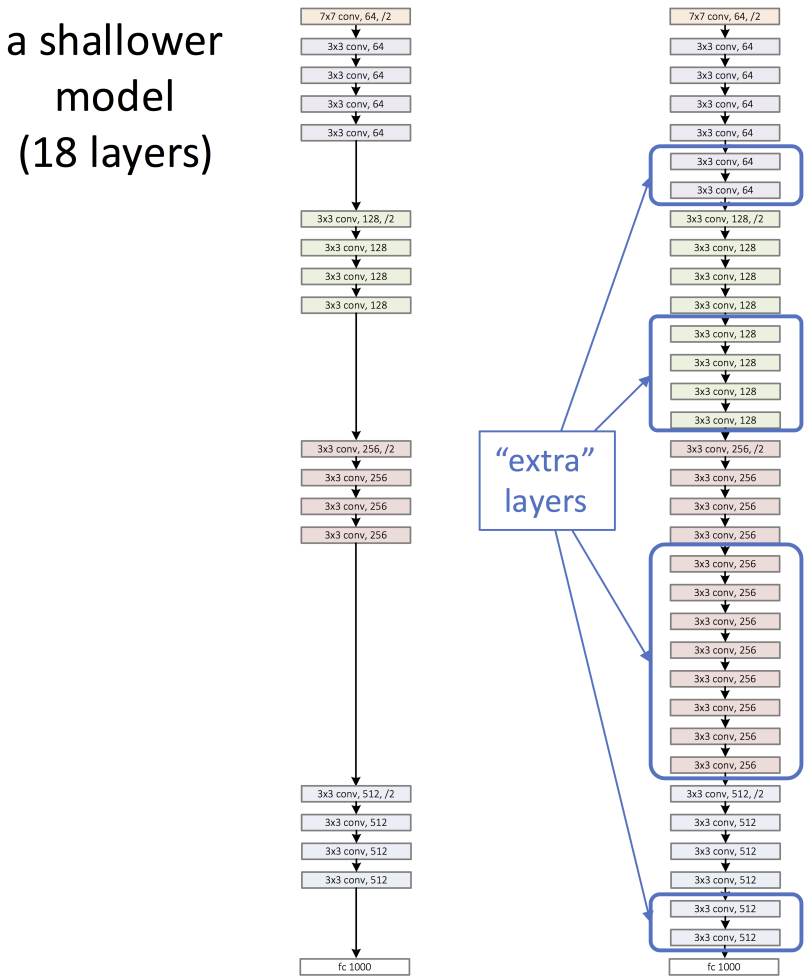
a shallower model (18 layers)



- This result was counter intuitive to the authors as they expected that a deeper model should be able to model the problem better.

Figure: Fig. by Kaiming He

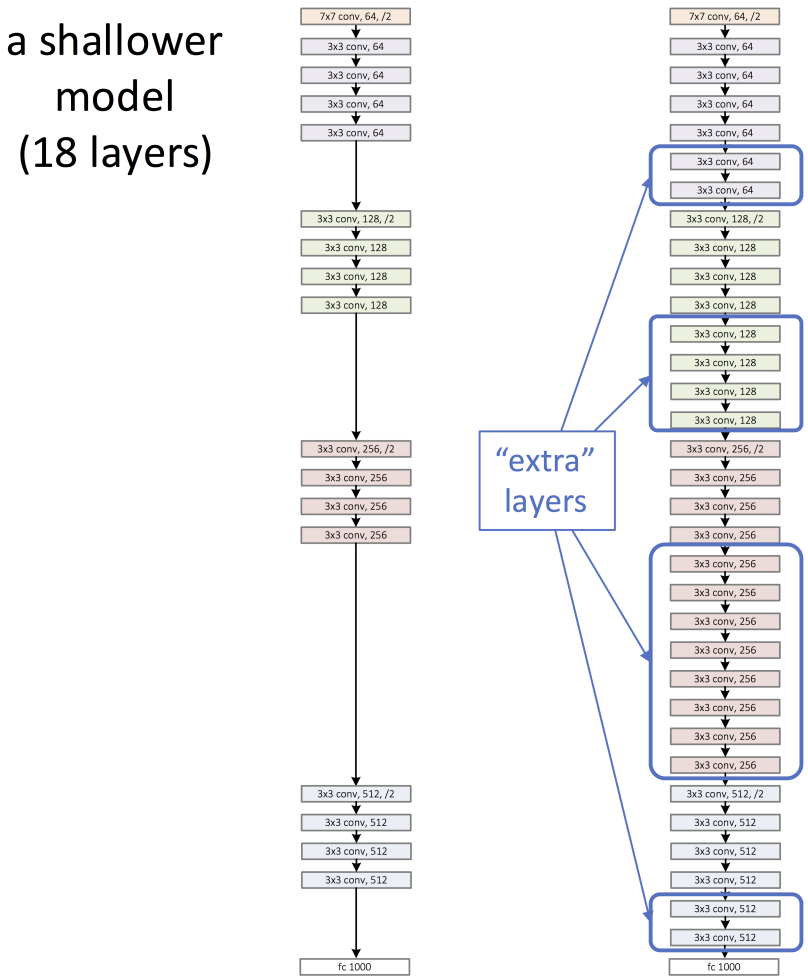
ResNet Architecture - He et al



- This result was counter intuitive to the authors as they expected that a deeper model should be able to model the problem better.
- For the deeper network, one can obtain a better output just by setting the additional layers to identity

Figure: Fig. by Kaiming He

ResNet Architecture - He et al



- This result was counter intuitive to the authors as they expected that a deeper model should be able to model the problem better.
- For the deeper network, one can obtain a better output just by setting the additional layers to identity
- This gave them the idea that the network that is being learnt should learn to model just the additional residue

Figure: Fig. by Kaiming He

ResNet Architecture - He et al

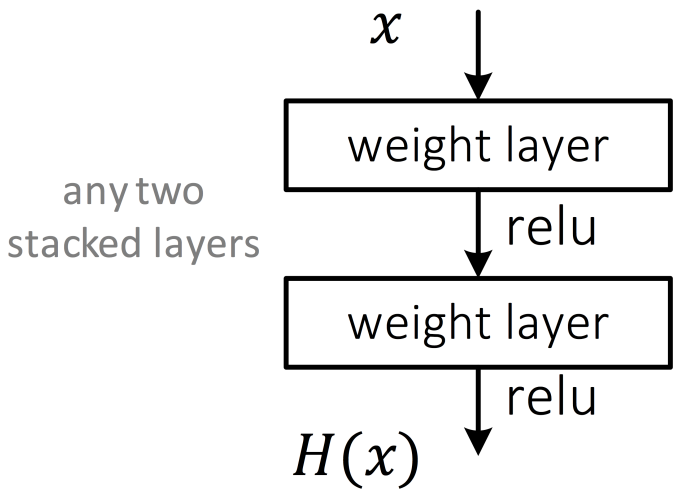


Figure: Illustration of idea by Kaiming He

- The main idea is as follows, normally given a stack of two layers that is learning a function $H(x)$ given an input x

ResNet Architecture - He et al

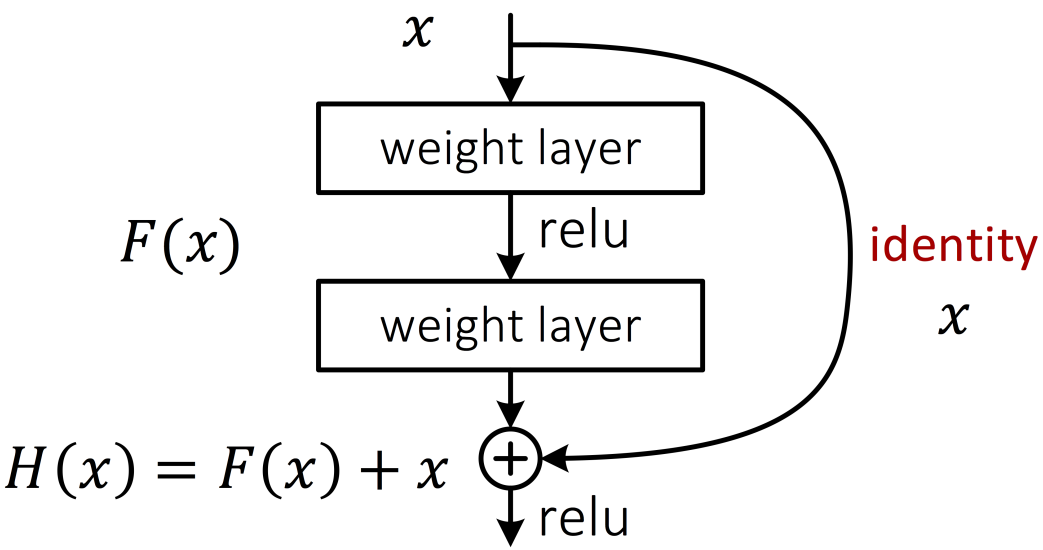


Figure: Illustration of idea by Kaiming He

- The main idea is as follows, normally given a stack of two layers that is learning a function $H(x)$ given an input x
- The idea is that the stack of layers could be used to model the residue function $F(x)$ where $F(x)$ could learn the identity function. The resultant function would be $H(x) = F(x) + x$ and the stack of layers learns the residue function $H(x) - x$.

Results of ResNet Architecture - He et al,

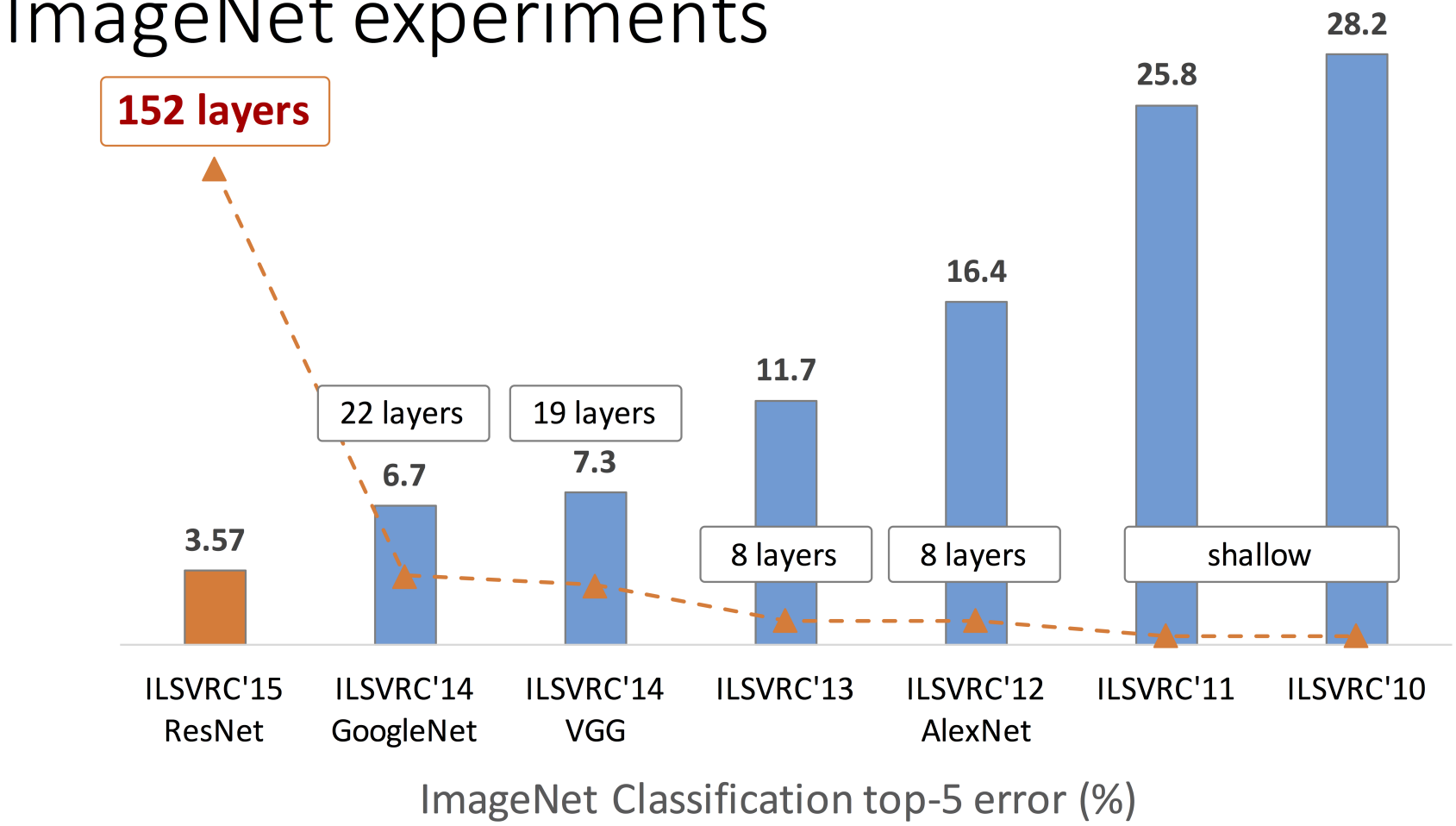


Figure: Results of Resnet 152 layer network as against other networks on ImageNet challenge. Figure by Kaiming He

Results of ResNet Architecture - He et al,

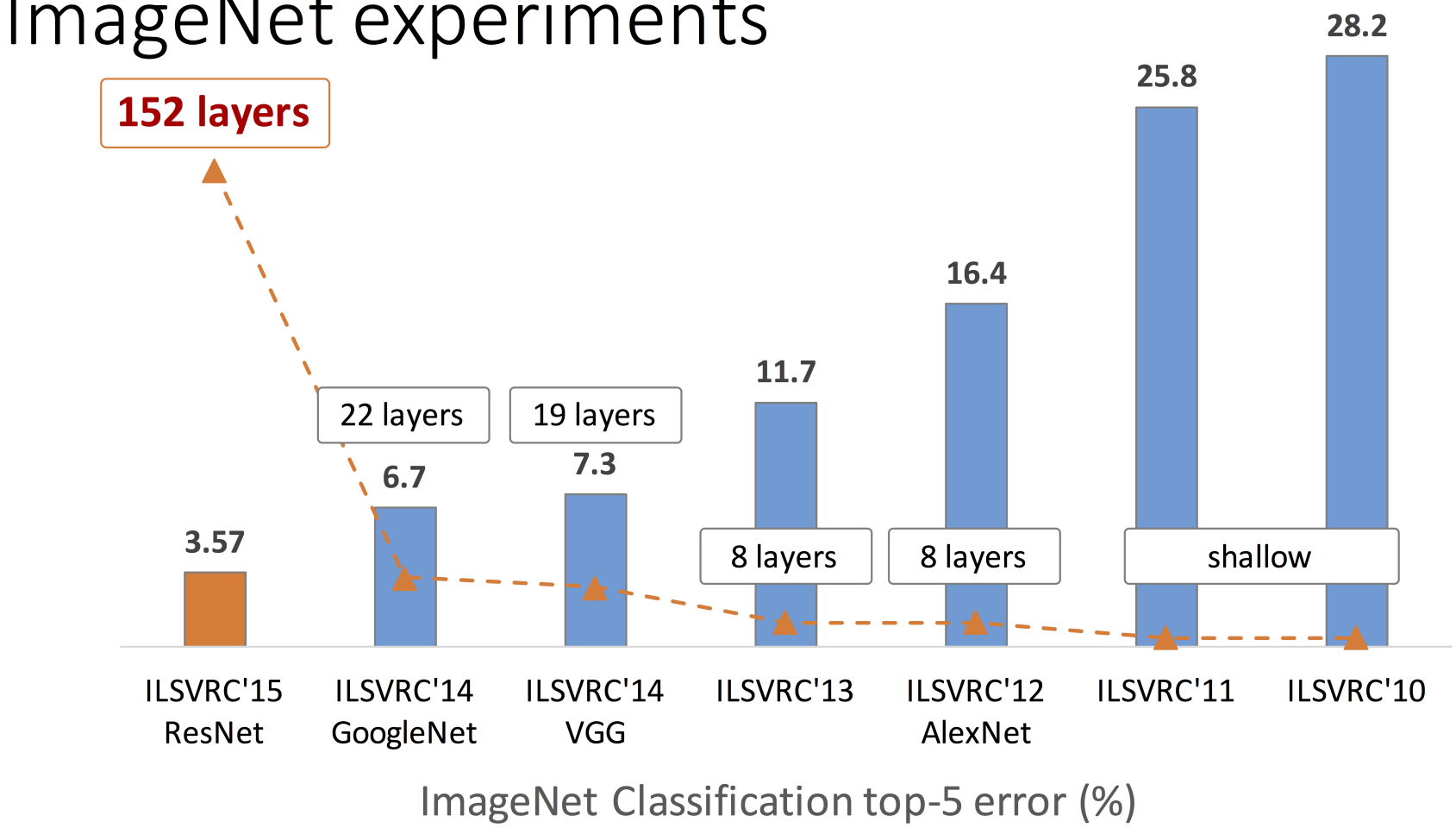


Figure: Results of Resnet 152 layer network as against other networks on ImageNet challenge. Figure by Kaiming He

Results of ResNet Architecture - He et al,

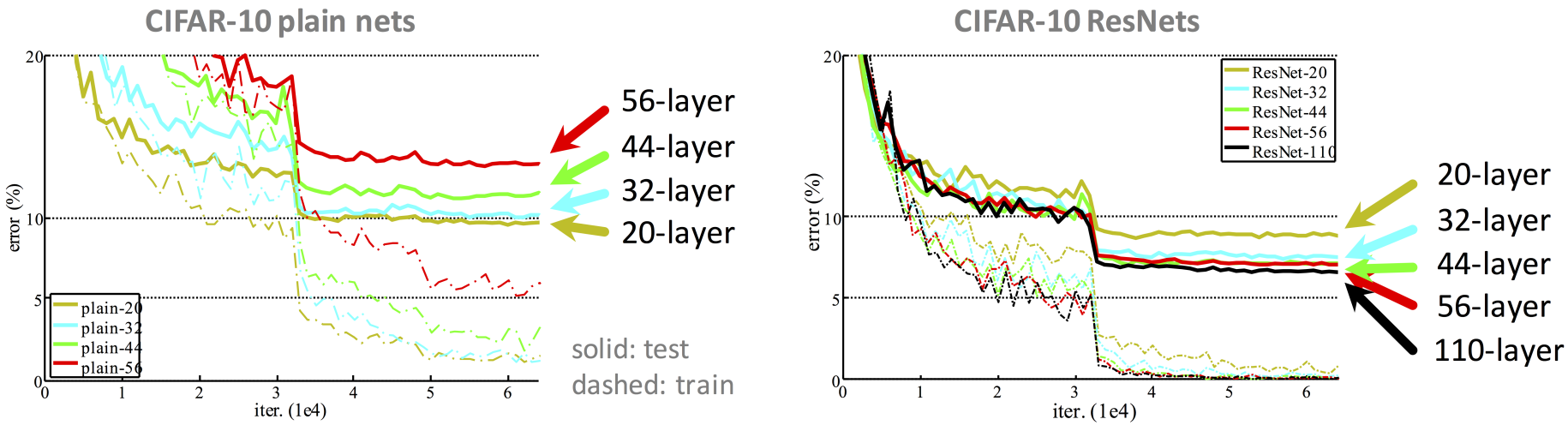


Figure: Results of Resnet as against a plain network on Cifar 10 dataset. Figure by Kaiming He

The End