Machine Learning

Lecture 07: Convolutional Neural Networks

Nevin L. Zhang
lzhang@cse.ust.hk

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

This set of notes is based on various sources on the internet and Stanford CS course CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.stanford.edu/
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press. http://www.deeplearningbook.org

Outline

1 What are Convolutional Neural Networks?

2 Convolutional Layer

3 Technical Issues with Convolutional Layer

4 Pooling Layer

5 Batch Normalization

6 Example CNN Architectures

Convolutional Neural Networks

- **Convolutional Neural Networks (CNNs, ConvNets)** are specialized neural networks for processing data that has a known grid-like topology, such as images.

- The input is a 3D tensor, where spatial relationships are important.

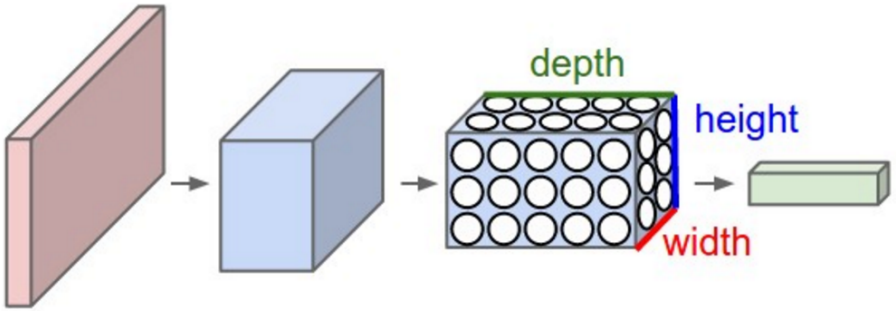
Convolutional Neural Networks

- **Convolutional Neural Networks (CNNs, ConvNets)** are specialized neural networks for processing data that has a known grid-like topology, such as images.

- The input is a 3D tensor, where spatial relationships are important.
- In contrast, the input to an FNN is vector, whose components can be permuted (prior to training) without loosing any information.

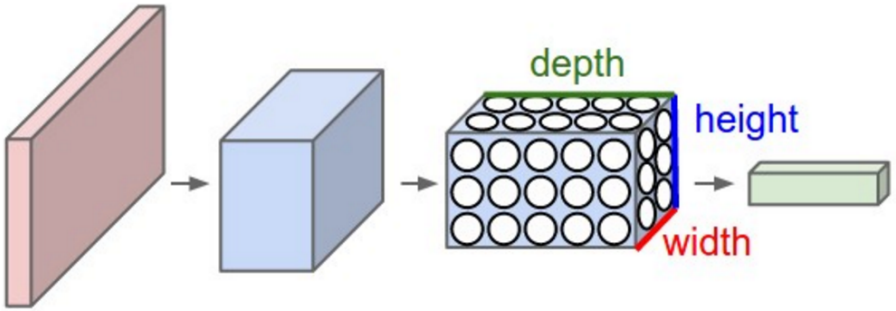
Convolutional Neural Networks

- The hidden layers are also organized into tensors.



Convolutional Neural Networks

- The hidden layers are also organized into tensors.



- A basic CNN consists of:
  - **Convolutional Layer**: Sparse connections, shared weights.
  - **Pooling Layer**: No weights.
  - **Normalization layers**: Special purpose.
  - **Fully-Connected Layer**: As in FNN.

Outline

1 What are Convolutional Neural Networks?

2 Convolutional Layer

3 Technical Issues with Convolutional Layer

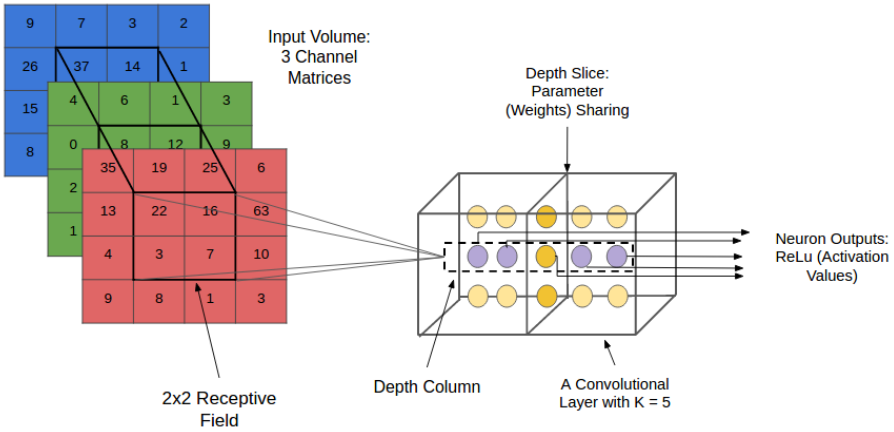
4 Pooling Layer

5 Batch Normalization

6 Example CNN Architectures

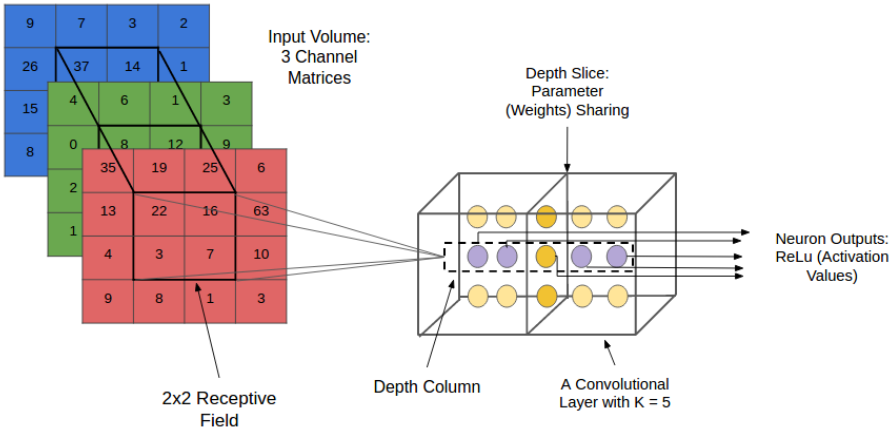
Convolutional Layer

- Each **convolutional unit** is connected only to units in a small patch (which is called the **receptive field** of the unit) of the previous layer.
- The receptive field is local in space (width and height), but always full along the entire depth of the input volume.



Convolutional Layer

- The parameters of a convolutional unit are the connection weights and the bias. They are to be learned.
- Intuitively, the task is to learn weights so that the unit will activate when some type of visual feature such as an edge is present.



Convolutional Layer

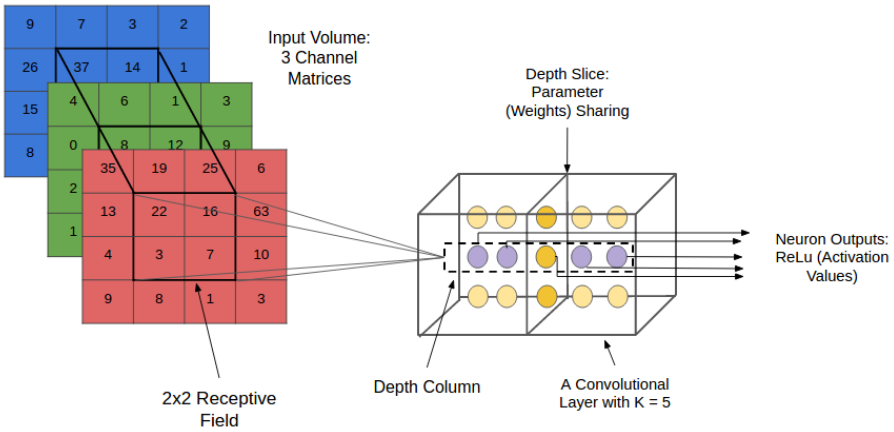
- A **convolutional layer** consists of a volume of convolutional units.
- All units on a given depth slice **share the same parameters**,

Convolutional Layer

- A **convolutional layer** consists of a volume of convolutional units.
- All units on a given depth slice **share the same parameters**,so that we can detect the same feature at different locations. (Edges can be at multiple locations)

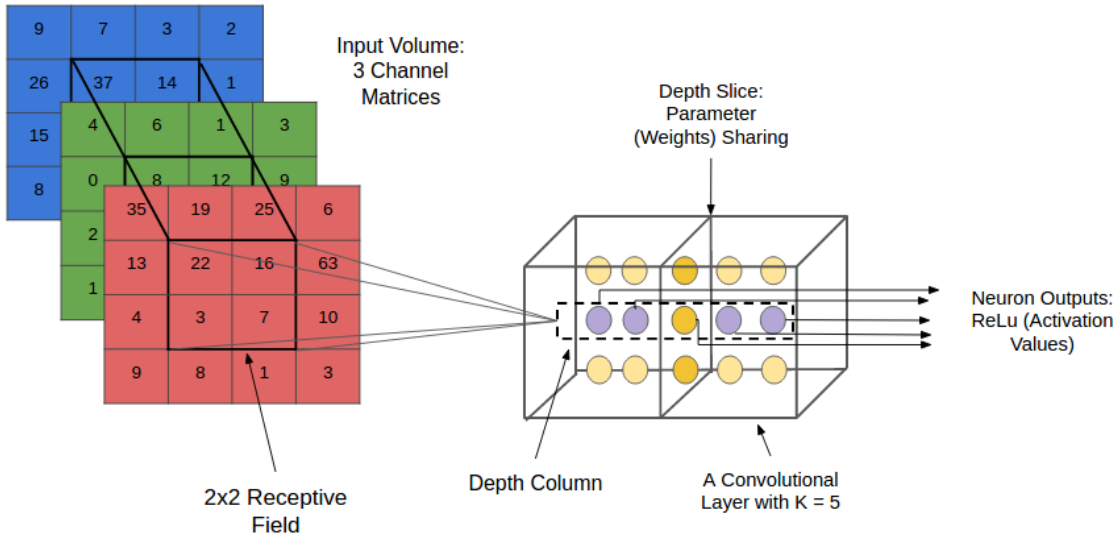
Convolutional Layer

- A **convolutional layer** consists of a volume of convolutional units.
- All units on a given depth slice **share the same parameters**,so that we can detect the same feature at different locations. (Edges can be at multiple locations)
  - Hence, the set of weights is called a **filter** or **kernel**.
  - Different units on the depth slices are obtained by sliding the filter.



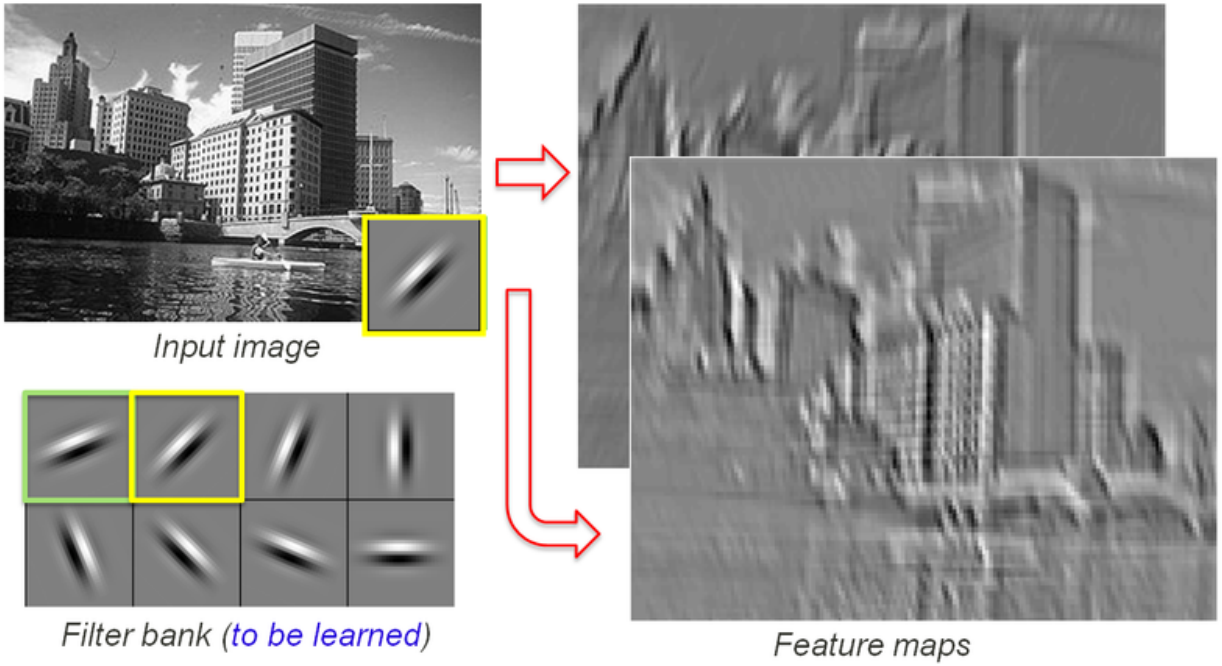
Convolutional Layer

- There are multiple depth slices (i.e., multiple filters) so that different features can be detected.



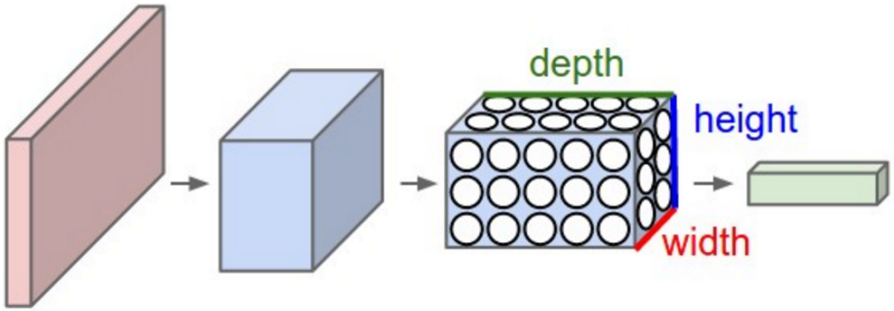
Convolutional Layer

- The output of each depth slice is called a **feature map**.



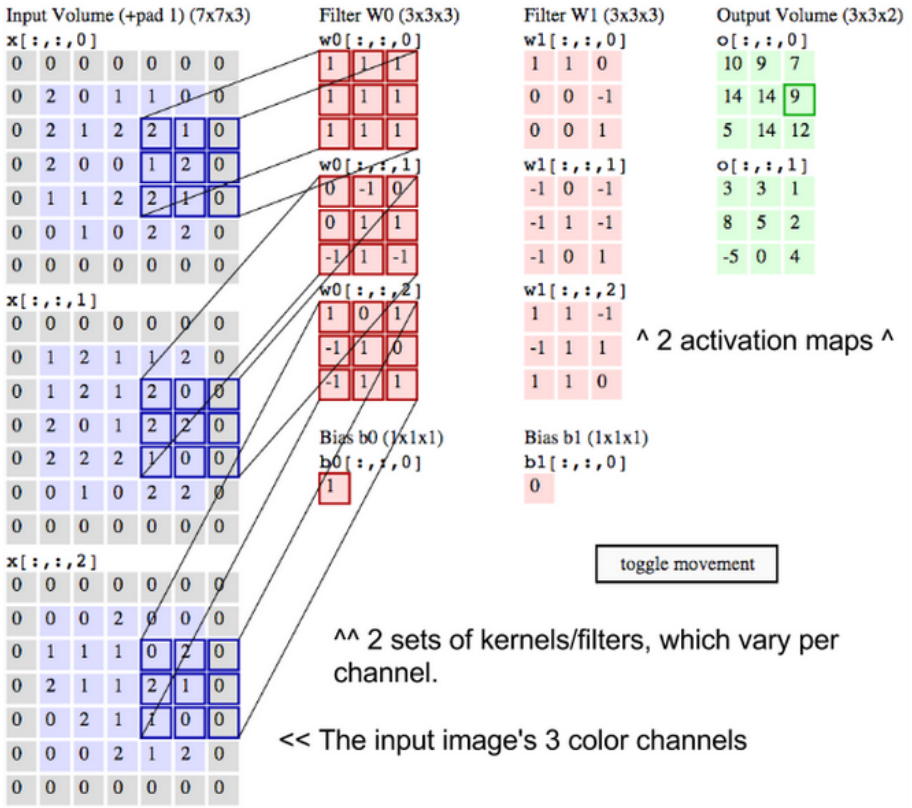
Convolutional Layer

- The output a conv layer is a collection of stacked feature maps.
- Mathematically, a conv layer maps a 3D tensor to another 3D tensor.

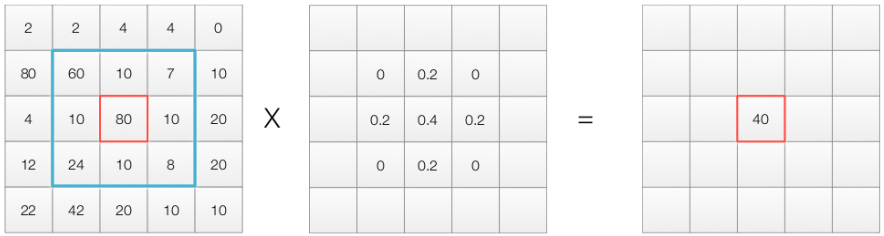


Convolutional Layer

Convolutional demo: `http://cs231n.github.io/convolutional-networks/`



Computation of a convolutional unit



- Let I be a 2D image (one of the three channels), and K be the filter.
  - $K(m,n) = 0$ when $|m| > r$ or $|n| > r$ where $2r$ is the weight and height of the receptive field.
- Computation carried by the convolutional unit at coordinates (i,j) is

$$S(i,j) = \sum_{m,n} I(i+m, j+n)K(m,n)$$

This is **cross-correlation**, although it is referred to as **convolution** in deep learning (`en.wikipedia.org/wiki/Cross-correlation`).

NOTE: Cross-correlation vs convolution

- **Cross-correlation**: $\sum_{m,n} I(i+m, j+n)K(m,n)$

NOTE: Cross-correlation vs convolution

- **Cross-correlation**: $\sum_{m,n} I(i+m, j+n)K(m,n)$
- **Convolution**: $\sum_{m,n} I(i-m, j-n)K(m,n)$

NOTE: Cross-correlation vs convolution

- **Cross-correlation**: $\sum_{m,n} I(i+m, j+n)K(m,n)$
- **Convolution**: $\sum_{m,n} I(i-m, j-n)K(m,n)$

NOTE: Cross-correlation vs convolution

- **Cross-correlation**: $\sum_{m,n} I(i+m, j+n)K(m,n)$
- **Convolution**: $\sum_{m,n} I(i-m, j-n)K(m,n)$

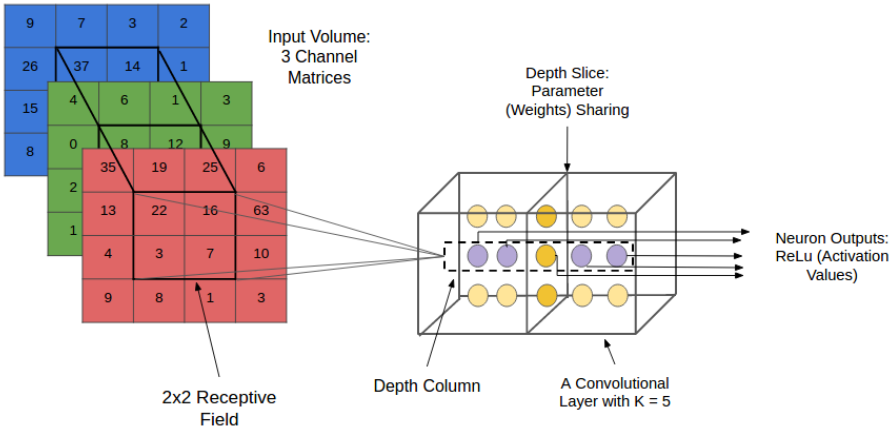
- Let us flip the kernel K to get $K'(m,n) = K(-m,-n)$. Then

$$\begin{aligned}\sum_{m,n} I(i-m, j-n)K(m,n) &= \sum_{m,n} I(i-m, j-n)K'(-m,-n) \\ &= \sum_{m,n} I(i+m, j+n)K'(m,n)\end{aligned}$$

- So, convolution with kernel K is the same as cross-correlation with the flipped kernel K'.
- And because the kernel is to be learned., it is not necessary to distinguish between the two (and hence cross-correlation and convolution) in deep learning.

Computation of a convolutional unit

- The result of convolution is passed through a nonlinear activation function (ReLU) to get the output of the unit.



Outline

1 What are Convolutional Neural Networks?

2 Convolutional Layer

3 Technical Issues with Convolutional Layer

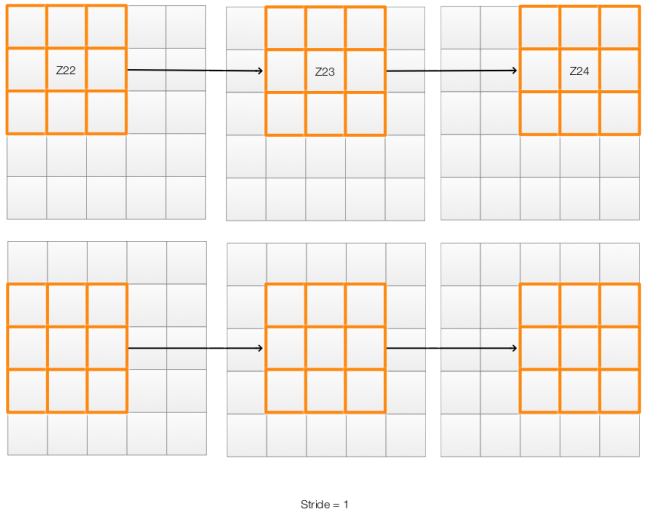
4 Pooling Layer

5 Batch Normalization

6 Example CNN Architectures

Reduction in Spatial Size

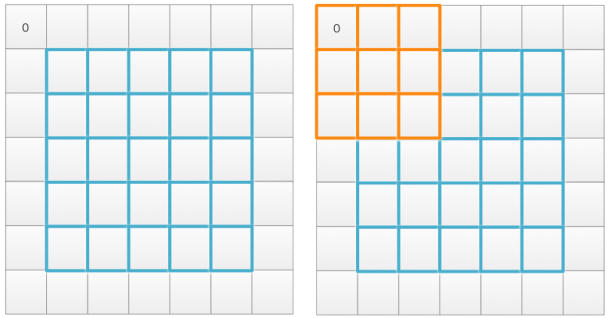
- To apply the filter to an image, we can move the filter 1 pixel at a time from left to right and top to bottom until we process every pixel.
- And we have a convolutional unit for each location.



- The width and height of the array convolutional units are reduced by 2 from the width and height of the array of input units.

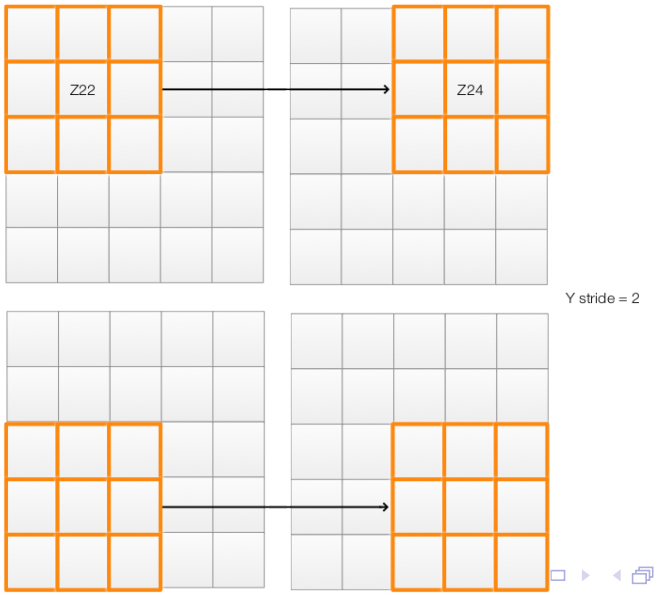
Zero Padding

- If we want to maintain the spatial dimensions, we can pack extra 0 or replicate the edge of the original image.
- **Zero padding** helps to better preserve information on the edge.



Stride

- Sometimes we do not move the filter only by 1 pixel each time. Instead we might want to move the filter 2 pixels each time. In this case, we use **stride** 2.
- It is uncommon to use stride 3 or more.



Summary of Convolutional Layer

Summary of Convolutional Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K, their spatial extent F, the stride S, the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:

$$\begin{aligned} W_2 &= (W_1 - F + 2P)/S + 1 \\ H_2 &= (H_1 - F + 2P)/S + 1 \\ D_2 &= K \end{aligned}$$

Summary of Convolutional Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K, their spatial extent F, the stride S, the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:

$$\begin{aligned} W_2 &= (W_1 - F + 2P)/S + 1 \\ H_2 &= (H_1 - F + 2P)/S + 1 \\ D_2 &= K \end{aligned}$$

- Need to make sure that $(W_1 - F + 2P)/S$ and $(W_2 - F + 2P)/S$ are integers.
  - For example, we cannot have $W = 10$, $P = 0$, $F = 3$ and $S = 2$.

Summary of Convolutional Layer

Input: $W_1 \times H_1 \times D_1$

K FXF filters,

stride: S

Zero padding: P

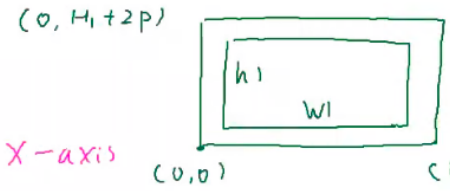
Output: $W_2 \times H_2 \times D_2$

$D_2 = K$

$W_2 = \frac{1}{S}(W_1 + 2P - F) + 1$

$H_2 = \frac{1}{S}(H_1 + 2P - F) + 1$

integer



1st possible location of filter?

$\frac{F}{2}$

last possible location of filter?

$W_1 + 2P - \frac{F}{2}$

# of possible location

$$\frac{W_1 + 2P - \frac{F}{2} - \frac{F}{2}}{S} + 1$$

$$= \frac{1}{S}(W_1 + 2P - F) + 1 \Rightarrow W_2$$

Summary of Convolutional Layer

- **Number of parameters** of a convolutional layer
  - $(FFD_1 + 1)K$ (K filters each with $FFD_1 + 1$ parameters.)

Summary of Convolutional Layer

- **Number of parameters** of a convolutional layer
  - $(FFD_1 + 1)K$ (K filters each with $FFD_1 + 1$ parameters.)
- **Number of FLOPs (floating point operations)** of a convolutional layer
  - $(FFD_1 + (FFD_1 - 1) + 1)W_2H_2D_2 = 2FFD_1W_2H_2D_2$

Summary of Convolutional Layer

- **Number of parameters** of a convolutional layer
  - $(FFD_1 + 1)K$ (K filters each with $FFD_1 + 1$ parameters.)
- **Number of FLOPs (floating point operations)** of a convolutional layer
  - $(FFD_1 + (FFD_1 - 1) + 1)W_2H_2D_2 = 2FFD_1W_2H_2D_2$
- A fully connected layer with the same number of units and no parameter sharing would require $(W_1H_1D_1 + 1)W_2H_2D_2$, which can be prohibitively large

Summary of Convolutional Layer

- **Number of parameters** of a convolutional layer
  - $(FFD_1 + 1)K$ (K filters each with $FFD_1 + 1$ parameters.)
- **Number of FLOPs (floating point operations)** of a convolutional layer
  - $(FFD_1 + (FFD_1 - 1) + 1)W_2H_2D_2 = 2FFD_1W_2H_2D_2$
- A fully connected layer with the same number of units and no parameter sharing would require $(W_1H_1D_1 + 1)W_2H_2D_2$, which can be prohibitively large

Outline

1 What are Convolutional Neural Networks?

2 Convolutional Layer

3 Technical Issues with Convolutional Layer

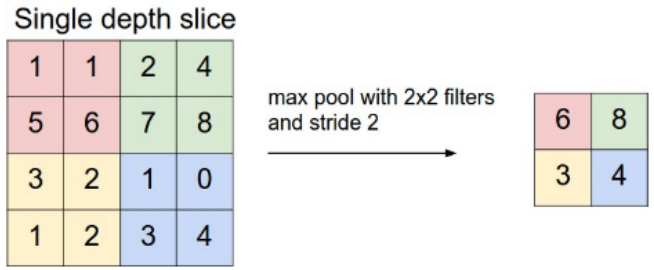
4 Pooling Layer

5 Batch Normalization

6 Example CNN Architectures

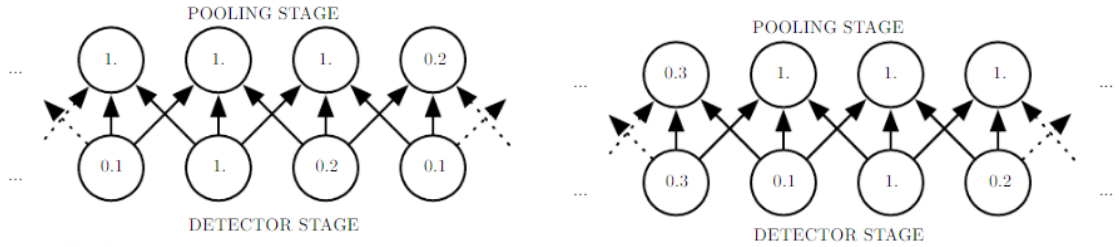
Pooling

- One objective of a pooling layer is to reduce the spatial size of the feature map.
- It aggregates a patch of units into one unit.
- There are different ways to do this, and **MAX pooling** is found the work the best.



Pooling

- Pooling also helps to make the representation approximately invariant to small translations of the input.
- Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.



Every value in the bottom row has changed, but only half of the values in the top row have changed.

- Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.

Outline

1 What are Convolutional Neural Networks?

2 Convolutional Layer

3 Technical Issues with Convolutional Layer

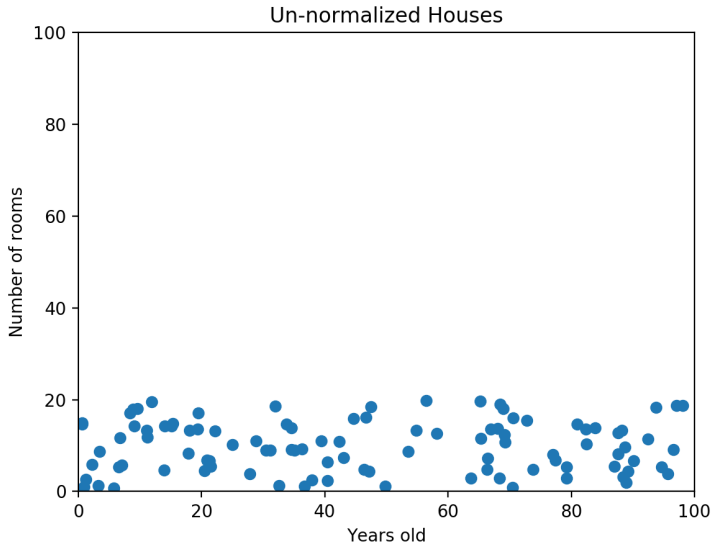
4 Pooling Layer

5 Batch Normalization

6 Example CNN Architectures

Data Normalization

- Sometimes different features in data have different scales:



- This can cause slow training and poor regularization effect.

Data Normalization

- Imagine two features: $x_1 \in [0, 1]$ and $x_2 \in [10, 1000]$.
- Ridge regression:

$$\begin{aligned} \hat{y} &= w_0 + w_1 x_1 + w_2 x_2 \\ J(\mathbf{w}) &= E[L(y, \hat{y})] + \lambda(w_1^2 + w_2^2) \end{aligned}$$

Data Normalization

- Imagine two features: $x_1 \in [0, 1]$ and $x_2 \in [10, 1000]$.
- Ridge regression:

$$\begin{aligned} \hat{y} &= w_0 + w_1 x_1 + w_2 x_2 \\ J(\mathbf{w}) &= E[L(y, \hat{y})] + \lambda(w_1^2 + w_2^2) \end{aligned}$$

- Problem 1: Regularization affects w_1 more than w_2:

Data Normalization

- Imagine two features: $x_1 \in [0, 1]$ and $x_2 \in [10, 1000]$.
- Ridge regression:

$$\begin{aligned} \hat{y} &= w_0 + w_1 x_1 + w_2 x_2 \\ J(\mathbf{w}) &= E[L(y, \hat{y})] + \lambda(w_1^2 + w_2^2) \end{aligned}$$

- Problem 1: Regularization affects w_1 more than w_2:
  - In comparison with w_2, changing w_1 has less impact on $\hat{y}$ and hence $E[L(y, \hat{(y)})]$.
  - As such, the value of w_1 will be affected more by the regularizer, and be pushed toward 0

Data Normalization

- Imagine two features: $x_1 \in [0, 1]$ and $x_2 \in [10, 1000]$.
- Ridge regression:

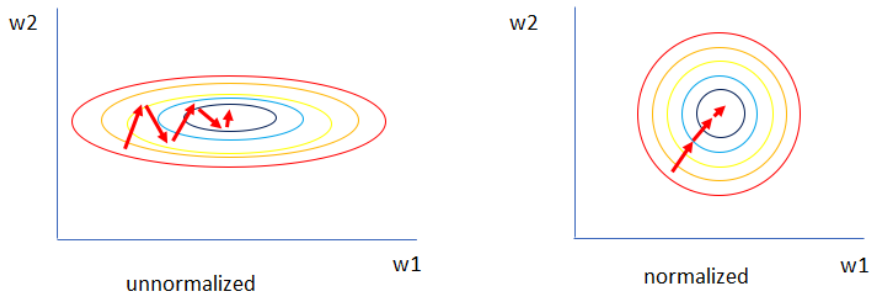
$$\begin{aligned} \hat{y} &= w_0 + w_1 x_1 + w_2 x_2 \\ J(\mathbf{w}) &= E[L(y, \hat{y})] + \lambda(w_1^2 + w_2^2) \end{aligned}$$

- Problem 1: Regularization affects w_1 more than w_2:
  - In comparison with w_2, changing w_1 has less impact on $\hat{y}$ and hence $E[L(y, \hat{(y)})]$.
  - As such, the value of w_1 will be affected more by the regularizer, and be pushed toward 0
- Consequently, bias would be increased (making x_1 too small than should be) .

Data Normalization

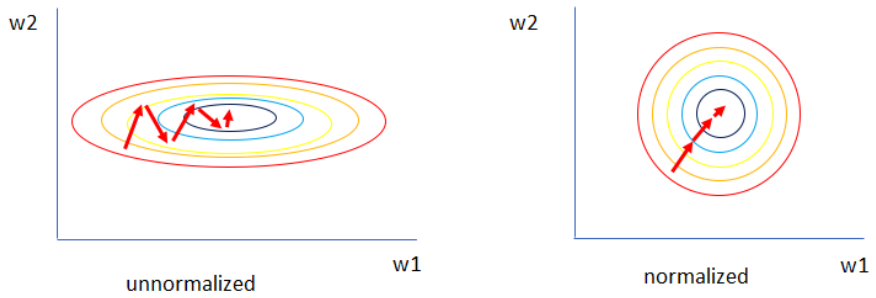
Data Normalization

- Problem 2: Contours of the loss function would be elongated along w_1.



Data Normalization

- Problem 2: Contours of the loss function would be elongated along w_1.



- This would lead to slow training (left).

$$\frac{\partial J}{\partial w_1} = \frac{\partial L}{\partial w_1} x_1 + 2\lambda w_1 \quad \textit{small}$$
$$\frac{\partial J}{\partial w_2} = \frac{\partial L}{\partial w_2} x_2 + 2\lambda w_2 \quad \textit{large}$$

Data Normalization

- Let $\mathbf{X} = [x_{ij}]_{N \times D}$. One way to normalize the data:

Data Normalization

- Let $\mathbf{X} = [x_{ij}]_{N \times D}$. One way to normalize the data:

$$\mu_j = \sum_{i=1}^{N} x_{ij} \quad \text{mean of column}$$

Data Normalization

- Let $\mathbf{X} = [x_{ij}]_{N \times D}$. One way to normalize the data:

$$\mu_j = \sum_{i=1}^{N} x_{ij} \quad \text{mean of column}$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{ij} - \mu_j)^2 \quad \text{variance of column}$$

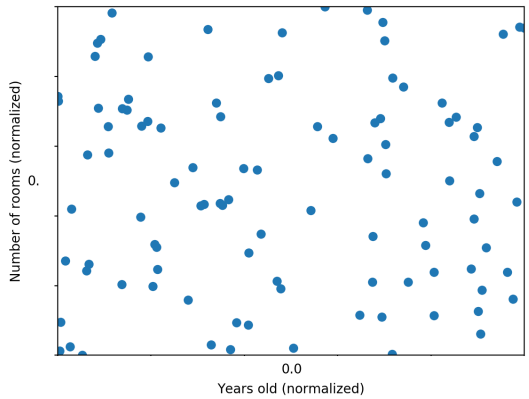
Data Normalization

- Let $\mathbf{X} = [x_{ij}]_{N \times D}$. One way to normalize the data:

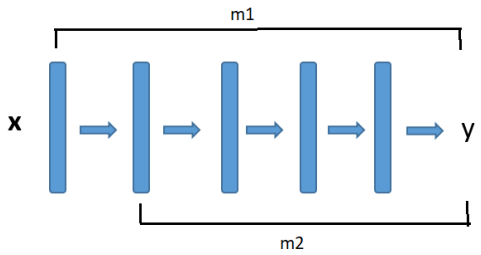
$$\mu_j = \sum_{i=1}^{N} x_{ij} \quad \text{mean of column}$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{ij} - \mu_j)^2 \quad \text{variance of column}$$

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sigma_i}$$

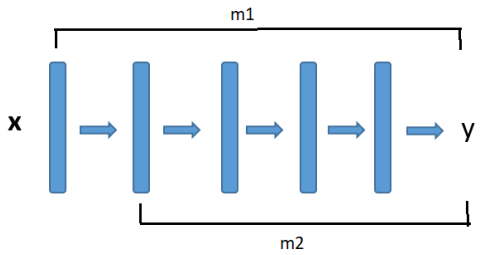


Normalization in Deep Models



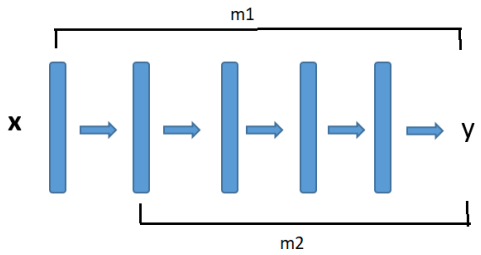
- A deep model can be viewed as a sequence of models
  - m_1 takes the original input $\mathbf{x}$,
  - m_2 takes the output of the first hidden layer as input.
  - ...

Normalization in Deep Models



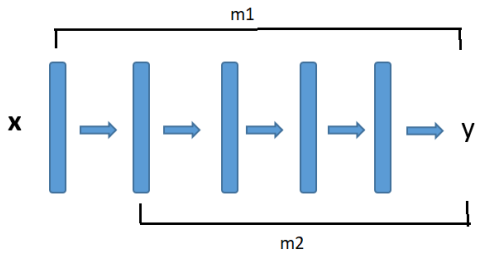
- A deep model can be viewed as a sequence of models
  - m_1 takes the original input $\mathbf{x}$,
  - m_2 takes the output of the first hidden layer as input.
  - ...
- Naturally, we want to normalize the input to each of those models, i.e., normalize at each layer.

Normalization in Deep Models



- A deep model can be viewed as a sequence of models
  - m_1 takes the original input $\mathbf{x}$,
  - m_2 takes the output of the first hidden layer as input.
  - ...
- Naturally, we want to normalize the input to each of those models, i.e., normalize at each layer.
- This makes different layers more independent, and avoids covariate shift,

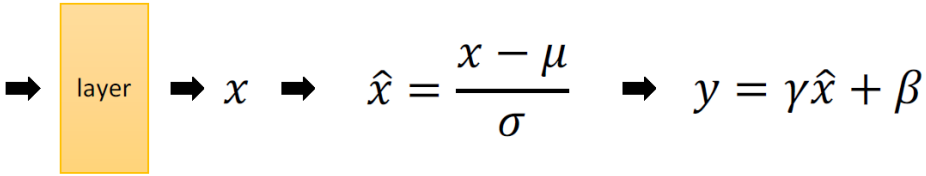
Normalization in Deep Models



- A deep model can be viewed as a sequence of models
  - m_1 takes the original input $\mathbf{x}$,
  - m_2 takes the output of the first hidden layer as input.
  - ...
- Naturally, we want to normalize the input to each of those models, i.e., normalize at each layer.
- This makes different layers more independent, and avoids covariate shift, (i.e., changes in parameters of earlier layers affect input distribution for later layers).

Batch normalization

- Normalizing **each layer** for **each mini-batch**.

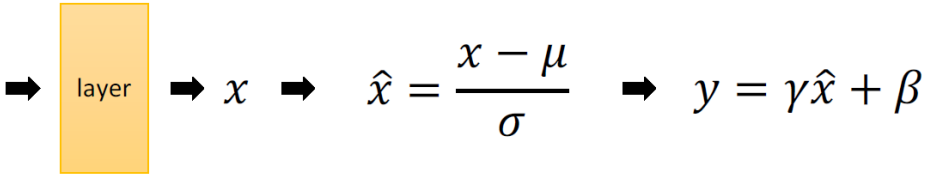


- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

Batch normalization

- Normalizing **each layer** for **each mini-batch**.

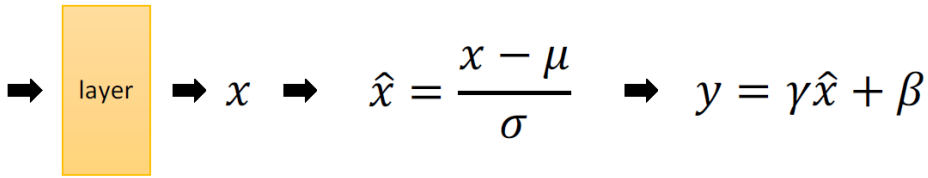


- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

Batch normalization

- Normalizing **each layer** for **each mini-batch**.



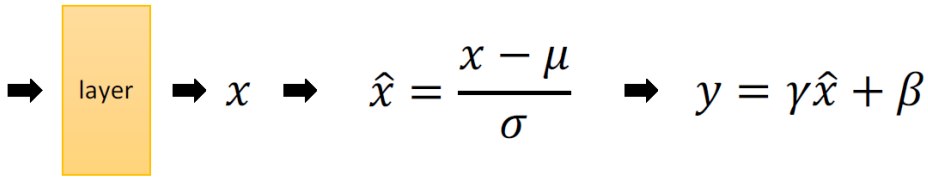
- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

- Accelerates training,

Batch normalization

- Normalizing **each layer** for **each mini-batch**.



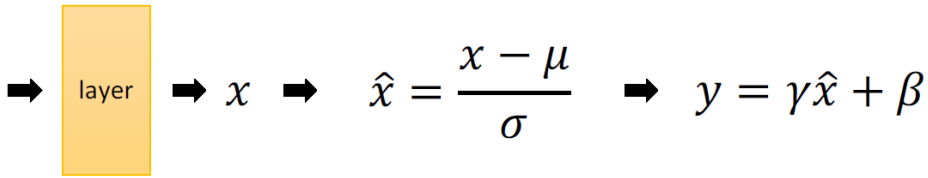
- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

- Accelerates training, decreases sensitivity to initialization,

Batch normalization

- Normalizing **each layer** for **each mini-batch**.



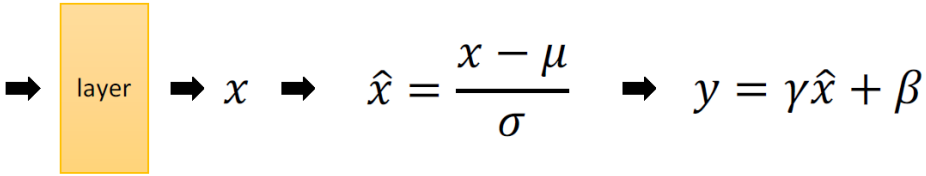
- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

- Accelerates training, decreases sensitivity to initialization, improves regularization

Batch normalization

- Normalizing **each layer** for **each mini-batch**.



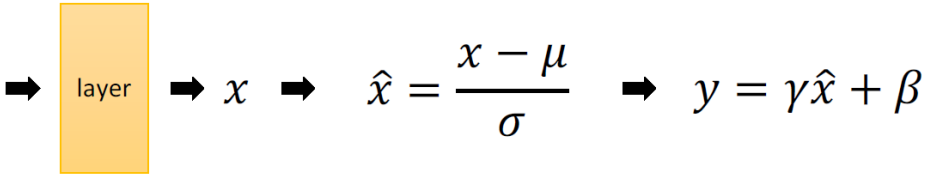
- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

- Accelerates training, decreases sensitivity to initialization, improves regularization
- In test mode, use μ and σ

Batch normalization

- Normalizing **each layer** for **each mini-batch**.



- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

- Accelerates training, decreases sensitivity to initialization, improves regularization
- In test mode, use μ and σ computed on training set.

How do the different layers fit together?

```
def __init__(self):
    super(LeNet, self).__init__()
    self.conv1 = nn.Conv2d(1, 20, 5, 1)
    self.conv2 = nn.Conv2d(20, 50, 5, 1)
    self.fc1 = nn.Linear(4*4*50, 500)
    self.fc2 = nn.Linear(500, 10)
    self.drop_out = nn.Dropout(p=0.5)

def forward(self, x):
    x = F.relu(self.conv1(x))
    x = F.max_pool2d(x, 2, 2)
    x = F.relu(self.conv2(x))
    x = F.max_pool2d(x, 2, 2)
    x = x.view(-1, 4*4*50)
    x = self.drop_out(x)
    x = F.relu(self.fc1(x))
    x = self.drop_out(x)
    x = self.fc2(x)
    return x
```

How do the different layers fit together?

```
def __init__(self):
    super(LeNet, self).__init__()
    self.conv1 = nn.Conv2d(1, 20, 5, 1)
    self.conv2 = nn.Conv2d(20, 50, 5, 1)
    self.fc1 = nn.Linear(4*4*50, 500)
    self.fc2 = nn.Linear(500, 10)
    self.drop_out = nn.Dropout(p=0.5)

def forward(self, x):
    x = F.relu(self.conv1(x))
    x = F.max_pool2d(x, 2, 2)
    x = F.relu(self.conv2(x))
    x = F.max_pool2d(x, 2, 2)
    x = x.view(-1, 4*4*50)
    x = self.drop_out(x)
    x = F.relu(self.fc1(x))
    x = self.drop_out(x)
    x = self.fc2(x)
    return x
```

Outline

1 What are Convolutional Neural Networks?

2 Convolutional Layer

3 Technical Issues with Convolutional Layer

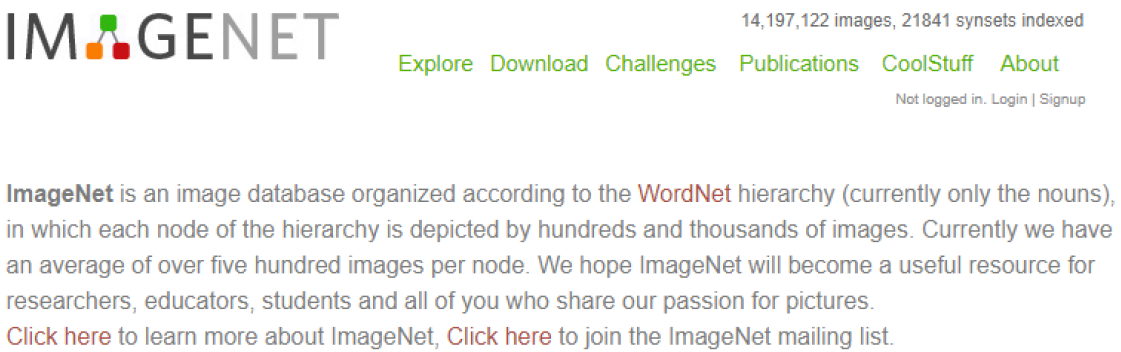
4 Pooling Layer

5 Batch Normalization

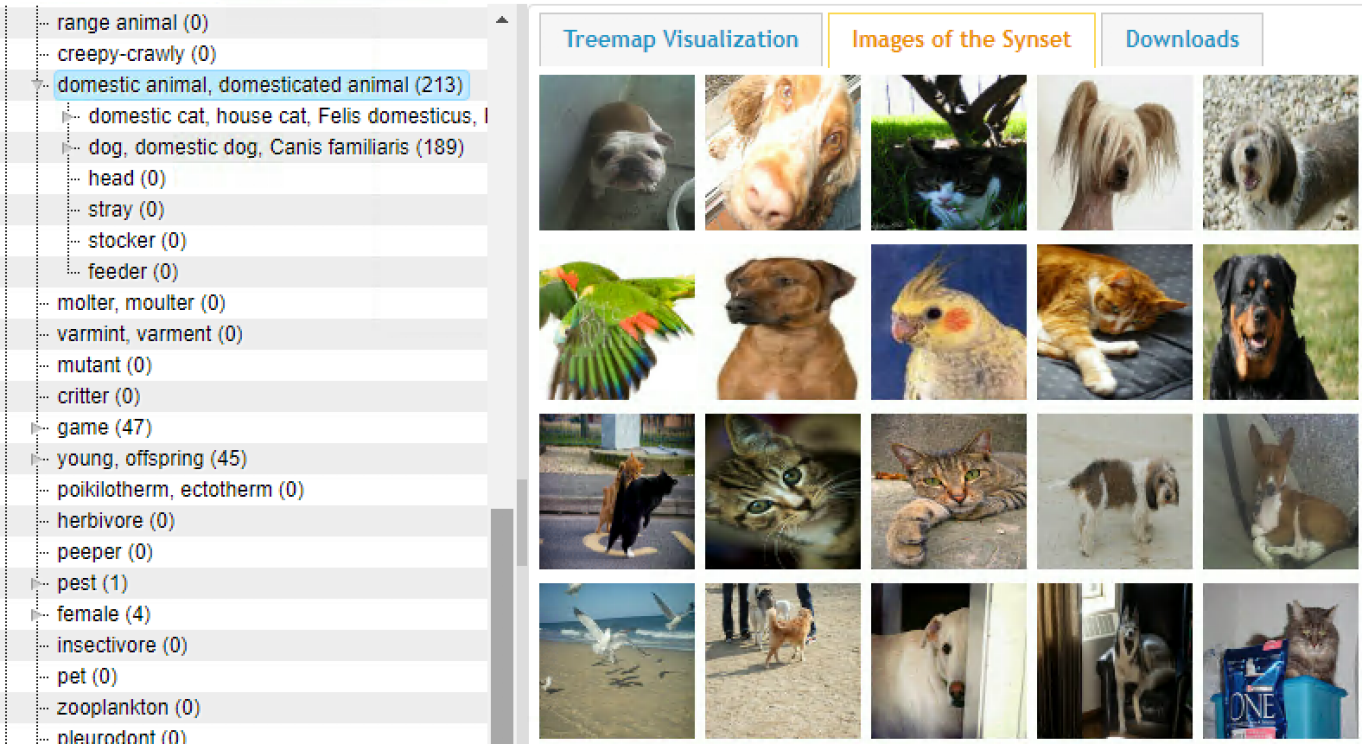
6 Example CNN Architectures

ImageNet

- The ImageNet project is a large visual database designed for use in visual object recognition software research.
- 14 million images have been hand-annotated by ImageNet to indicate what objects are pictured

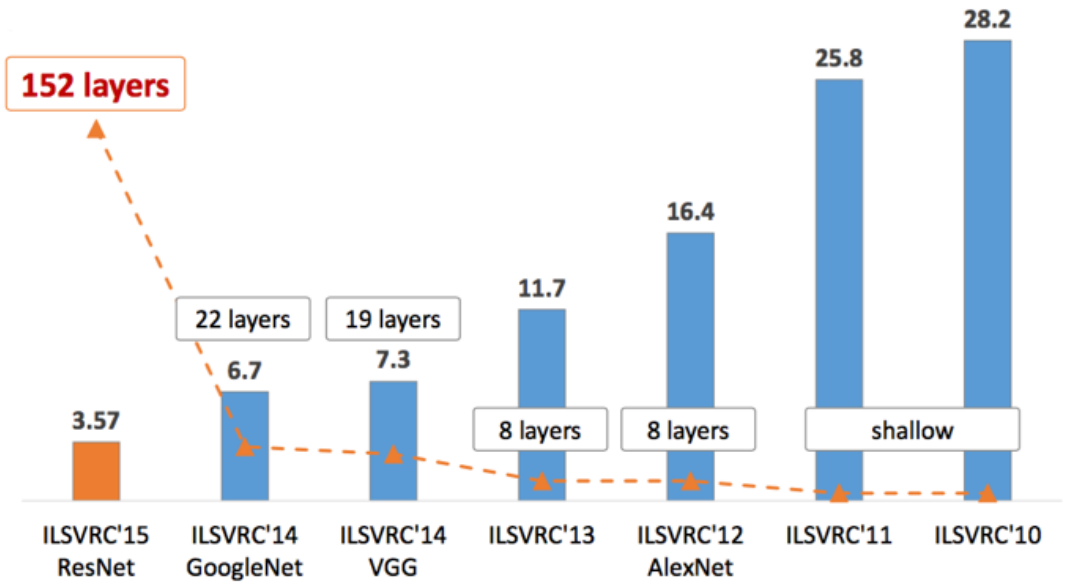


Examples from ImageNet



Large Scale Visual Recognition Challenge

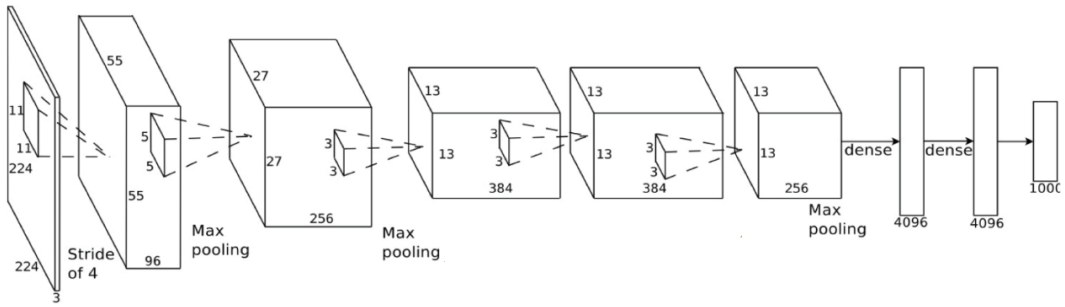
- Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Related Literature

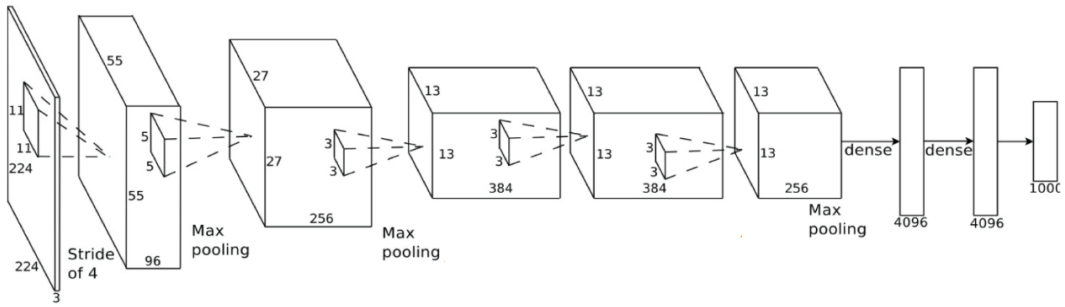
Series	Year	Title	Cited
NIPS	2012	ImageNet Classification with Deep Convolutional Neural Networks	56236
CVPR	2016	Deep Residual Learning for Image Recognition	38731
ICLR	2015	Adam: A Method for Stochastic Optimization	37276
ICLR	2015	Very Deep Convolutional Networks for Large-Scale Image Recognition	33834
CVPR	2015	Going deeper with convolutions	19159
JMLR	2014	Dropout: A Simple Way to Prevent Neural Networks from Overfitting	17715
NIPS	2015	Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks	15873
NIPS	2014	Generative Adversarial Nets	15802
ICML	2015	Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift	15717
CVPR	2015	Fully convolutional networks for semantic segmentation	14010

AlexNet (in terms of tensor shapes)



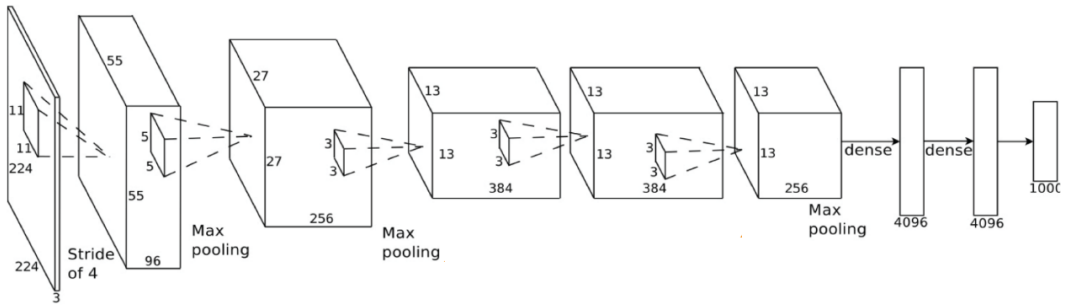
- Alexnet won the ImageNet ILSVRC challenge in 2012, beating all previous non-DL based method.

AlexNet (in terms of tensor shapes)



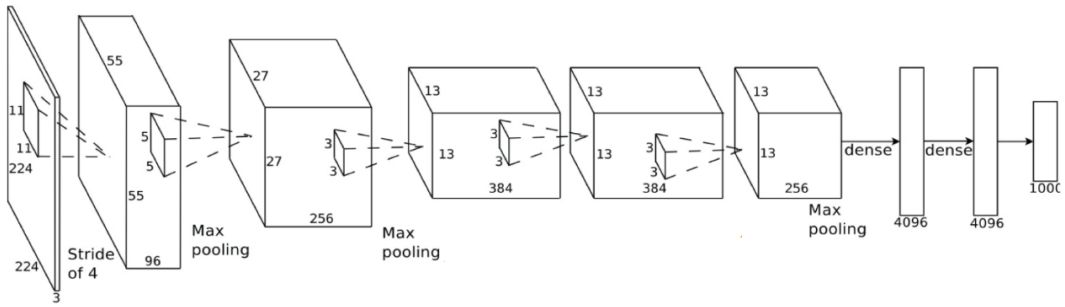
- Alexnet won the ImageNet ILSVRC challenge in 2012, beating all previous non-DL based method. Starting the surge of research on CNN.

AlexNet (in terms of tensor shapes)



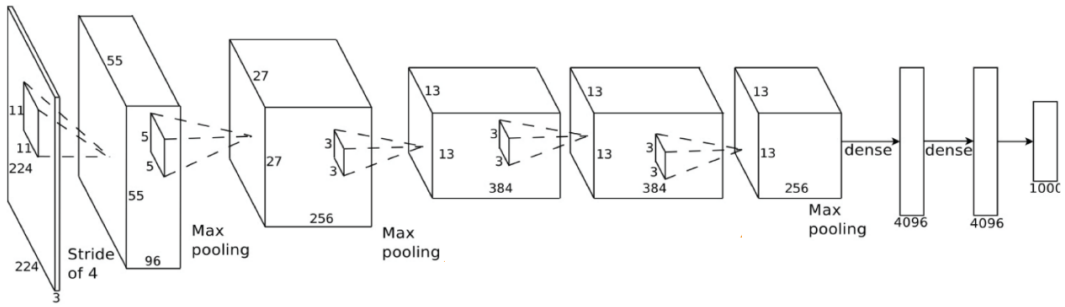
- Alexnet won the ImageNet ILSVRC challenge in 2012, beating all previous non-DL based method. Starting the surge of research on CNN.
- It has 5 conv layers which are followed by 3 fully connected layers.

AlexNet (in terms of tensor shapes)



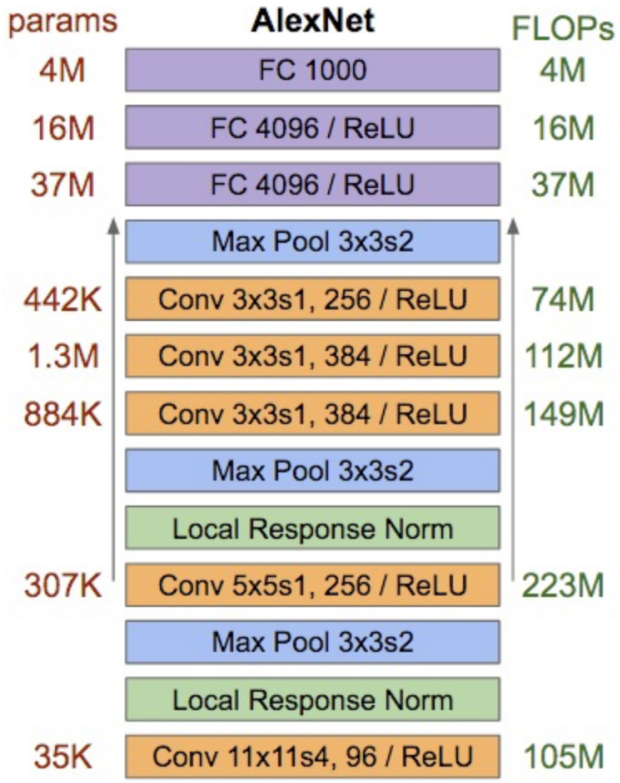
- Alexnet won the ImageNet ILSVRC challenge in 2012, beating all previous non-DL based method. Starting the surge of research on CNN.
- It has 5 conv layers which are followed by 3 fully connected layers.
- Totally, around 60M parameters, most of which are in the FC layers.

AlexNet (in terms of tensor shapes)

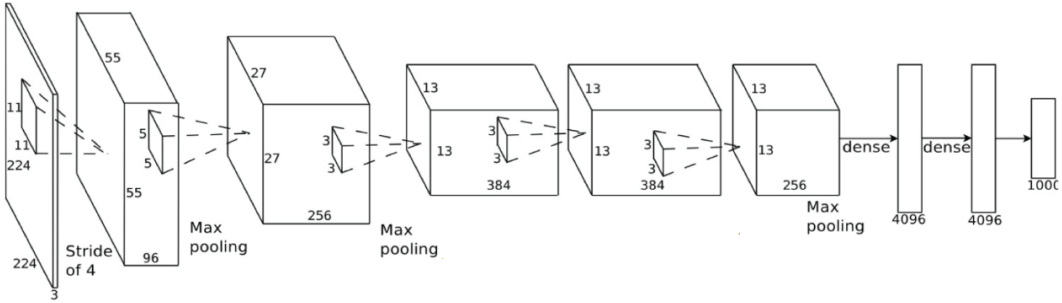


- Alexnet won the ImageNet ILSVRC challenge in 2012, beating all previous non-DL based method. Starting the surge of research on CNN.
- It has 5 conv layers which are followed by 3 fully connected layers.
- Totally, around 60M parameters, most of which are in the FC layers.
- Most operations (FLOPs) in forward propagation takes place in the Conv layers.

AlexNet (in terms of connections)

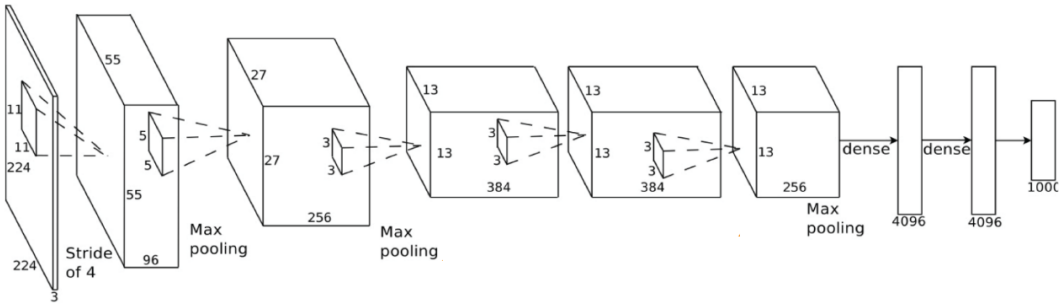


AlexNet



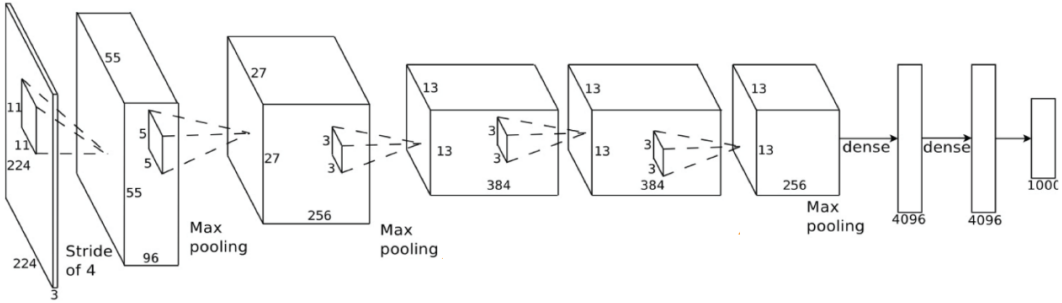
- First use of ReLU

AlexNet



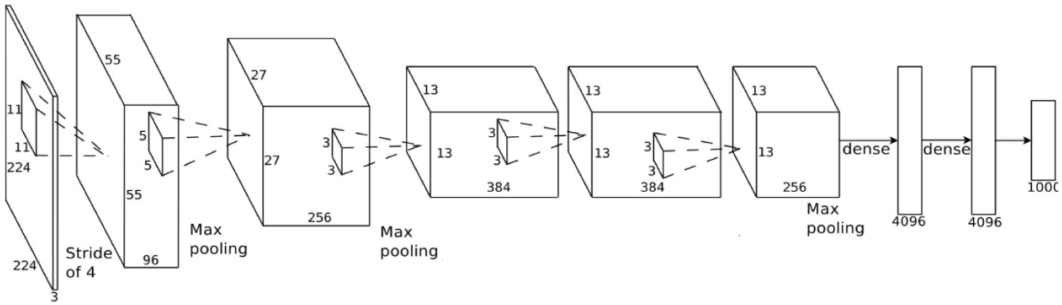
- First use of ReLU
- Heavy data augmentation

AlexNet



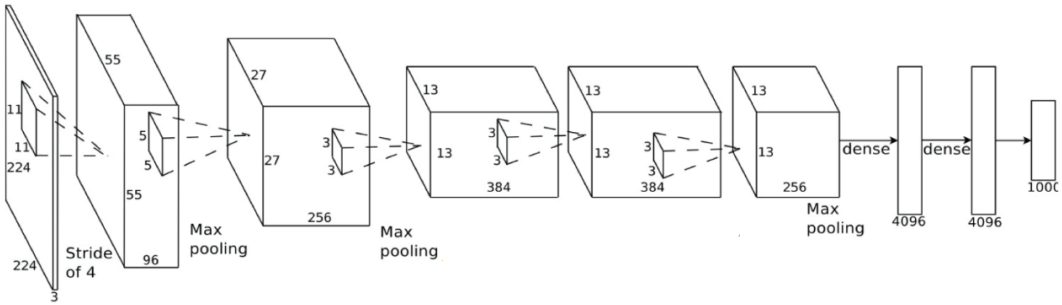
- First use of ReLU
- Heavy data augmentation
- Dropout: 0.5

AlexNet



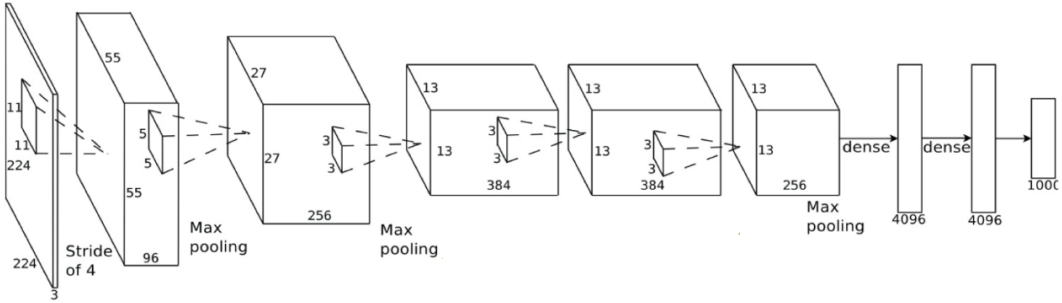
- First use of ReLU
- Heavy data augmentation
- Dropout: 0.5 ; Batch size: 128;

AlexNet



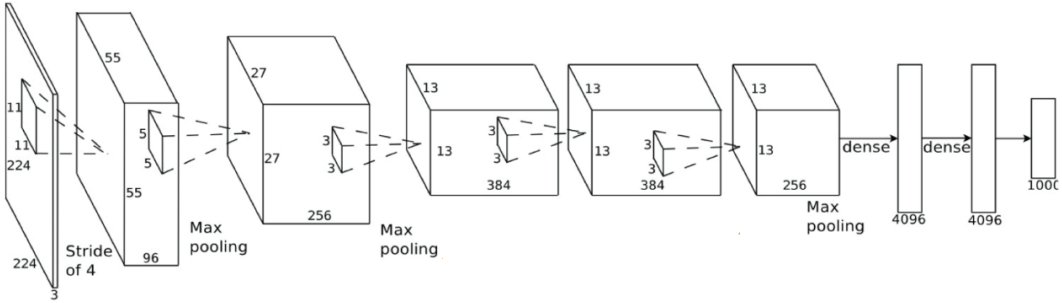
- First use of ReLU
- Heavy data augmentation
- Dropout: 0.5 ; Batch size: 128; SGD momentum 0.9

AlexNet



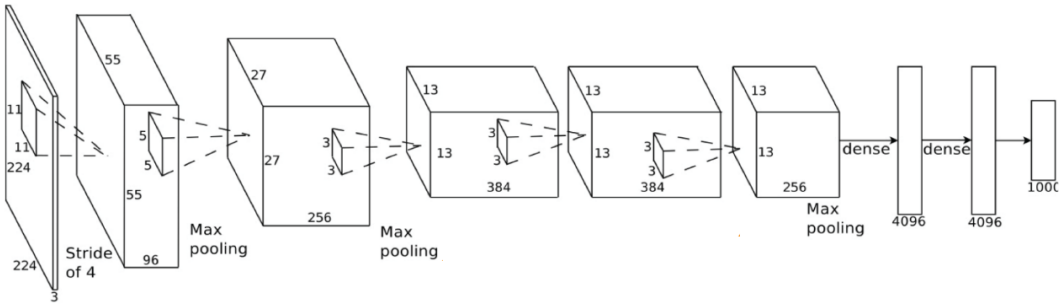
- First use of ReLU
- Heavy data augmentation
- Dropout: 0.5 ; Batch size: 128; SGD momentum 0.9
- Learning rate: 0.01, reduced by 10 manually when validation accuracy plateaus.

AlexNet



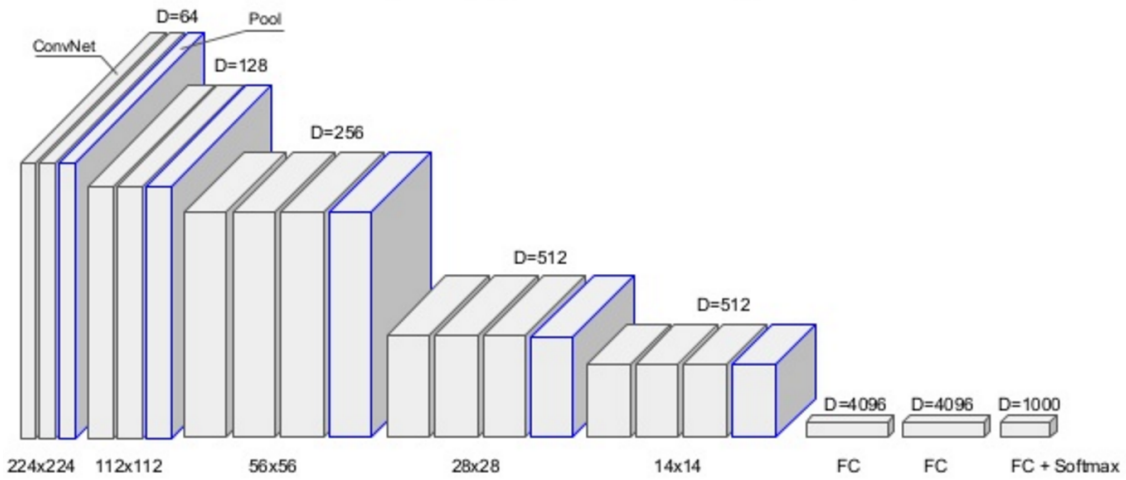
- First use of ReLU
- Heavy data augmentation
- Dropout: 0.5 ; Batch size: 128; SGD momentum 0.9
- Learning rate: 0.01, reduced by 10 manually when validation accuracy plateaus.
- L2 weight decay;

AlexNet

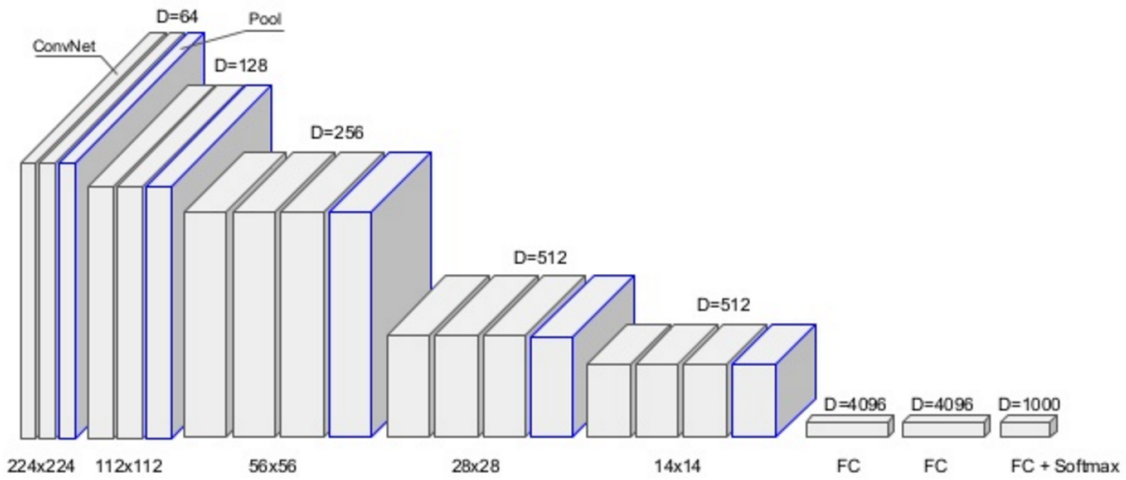


- First use of ReLU
- Heavy data augmentation
- Dropout: 0.5 ; Batch size: 128; SGD momentum 0.9
- Learning rate: 0.01, reduced by 10 manually when validation accuracy plateaus.
- L2 weight decay; 7 CNN ensemble.

VGGNet (in terms of tensor shapes)

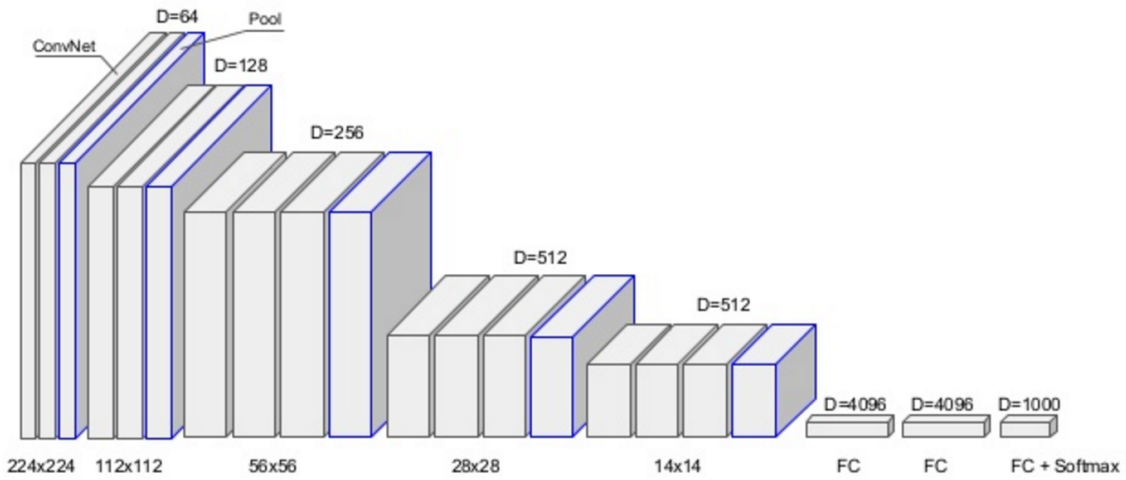


VGGNet (in terms of tensor shapes)



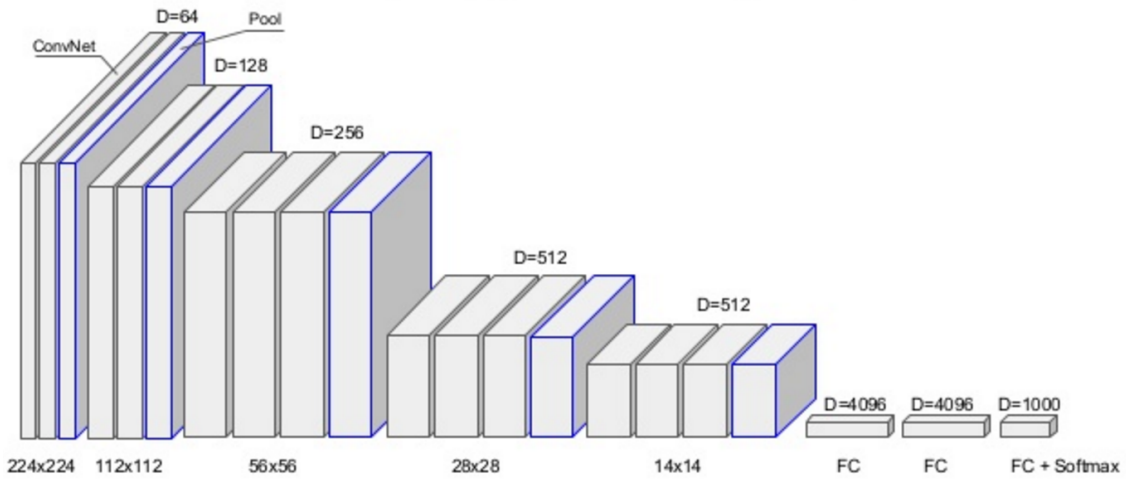
- Runner-up in the ImageNet ILSVRC challenge 2014,

VGGNet (in terms of tensor shapes)



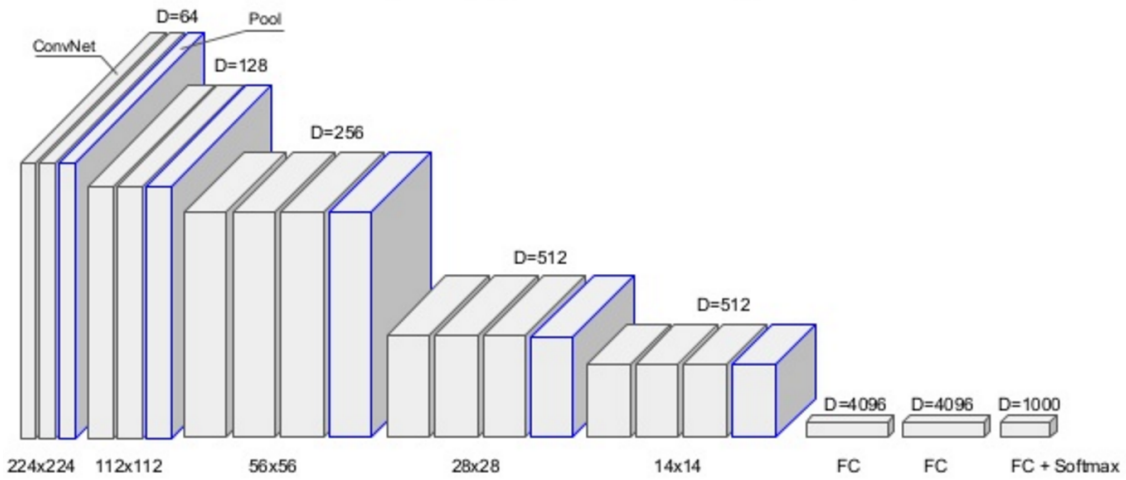
- Runner-up in the ImageNet ILSVRC challenge 2014, beating Alexnet .

VGGNet (in terms of tensor shapes)



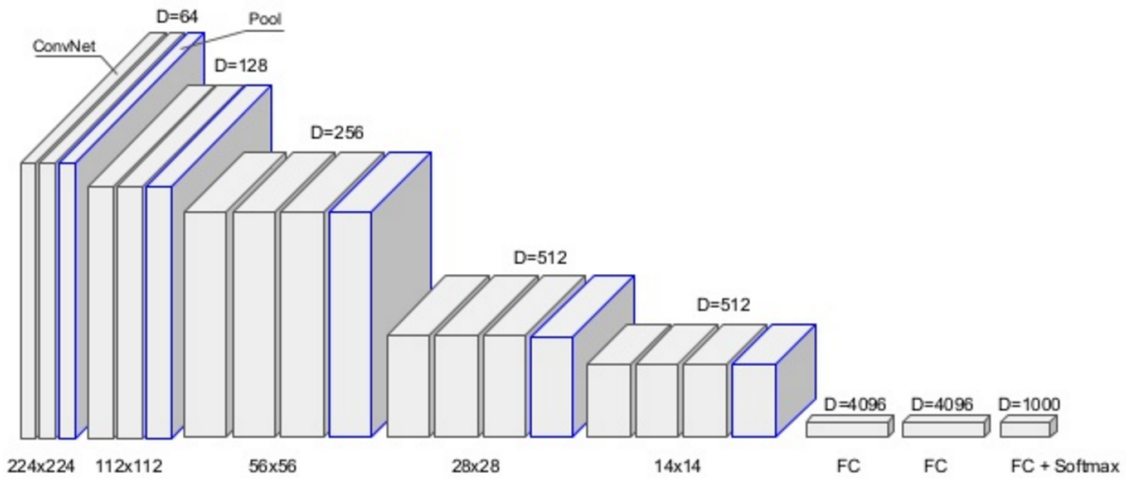
- Runner-up in the ImageNet ILSVRC challenge 2014, beating Alexnet .
- Key point: Depth of network is a critical component for good performance.

VGGNet (in terms of tensor shapes)



- Runner-up in the ImageNet ILSVRC challenge 2014, beating Alexnet .
- Key point: Depth of network is a critical component for good performance.
- It has 16 layers and always use 3 x 3 kernels.

VGGNet (in terms of tensor shapes)



- Runner-up in the ImageNet ILSVRC challenge 2014, beating Alexnet .
- Key point: Depth of network is a critical component for good performance.
- It has 16 layers and always use 3 x 3 kernels.
- Totally, 138M parameters.

VGGNet (in terms of connections)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

GoogleNet

- Winner the ImageNet ILSVRC challenge 2014, 22 layers.

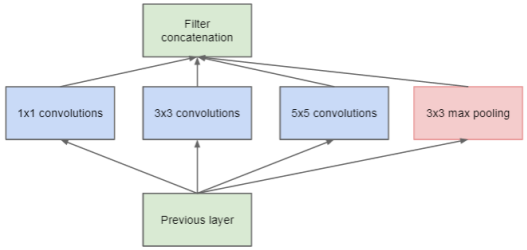
GoogleNet

- Winner the ImageNet ILSVRC challenge 2014, 22 layers.
- Main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (5M, compared to AlexNet with 60M).

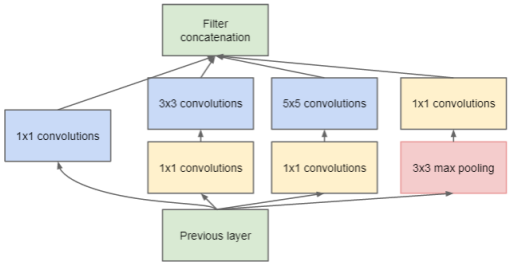
GoogleNet

- Winner the ImageNet ILSVRC challenge 2014, 22 layers.
- Main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (5M, compared to AlexNet with 60M).
- No fully connected layers.

Inception Module

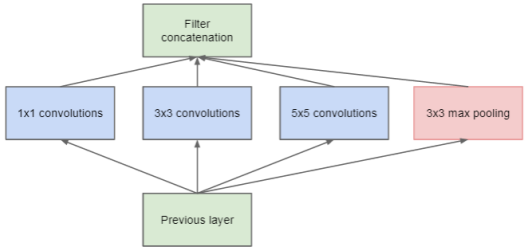


(a) Inception module, naïve version

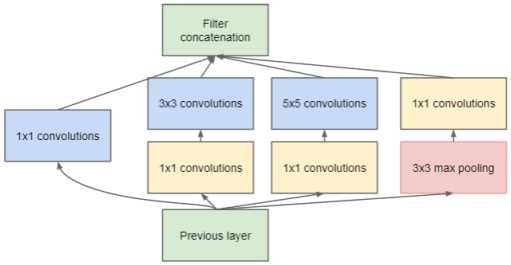


(b) Inception module with dimension reductions

Inception Module



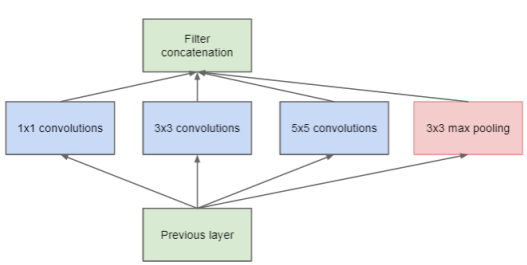
(a) Inception module, naïve version



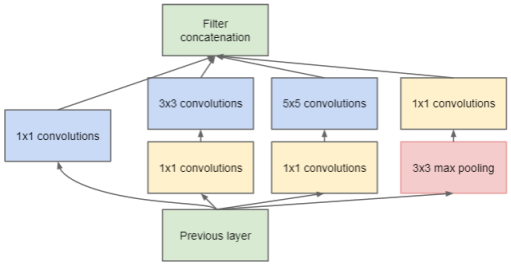
(b) Inception module with dimension reductions

- The idea of the inception layer is to cover a bigger area, but also keep a fine resolution for small information on the images.

Inception Module



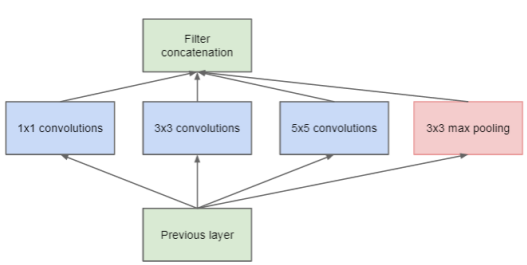
(a) Inception module, naïve version



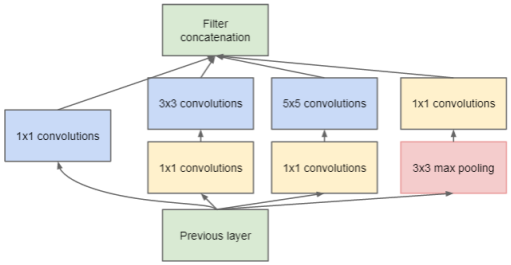
(b) Inception module with dimension reductions

- The idea of the inception layer is to cover a bigger area, but also keep a fine resolution for small information on the images.
- So the idea is to convolve in parallel different sizes from the most accurate detailing (1x1) to a bigger one (5x5).

Inception Module



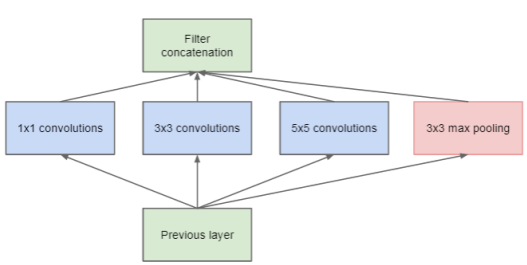
(a) Inception module, naïve version



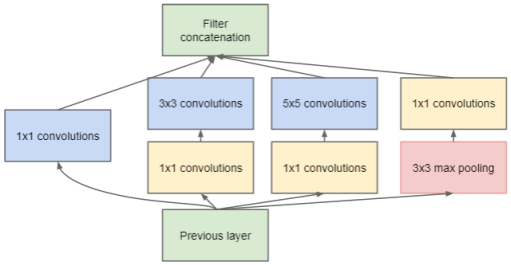
(b) Inception module with dimension reductions

- The idea of the inception layer is to cover a bigger area, but also keep a fine resolution for small information on the images.
- So the idea is to convolve in parallel different sizes from the most accurate detailing (1x1) to a bigger one (5x5).
- The naive version has too many features (parameters), more than in VGGNet.

Inception Module



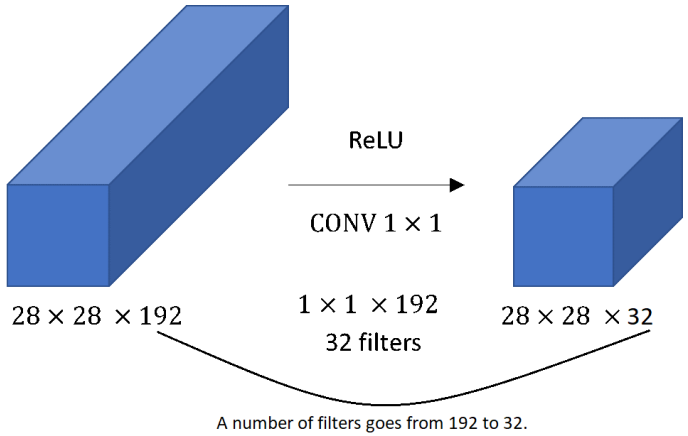
(a) Inception module, naïve version



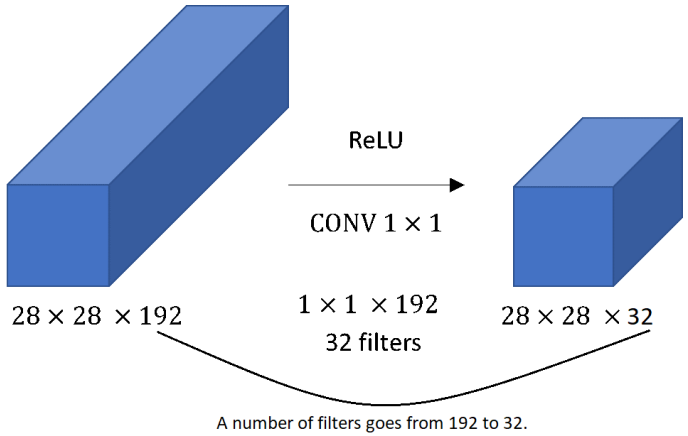
(b) Inception module with dimension reductions

- The idea of the inception layer is to cover a bigger area, but also keep a fine resolution for small information on the images.
- So the idea is to convolve in parallel different sizes from the most accurate detailing (1x1) to a bigger one (5x5).
- The naive version has too many features (parameters), more than in VGGNet.
- So, bottleneck layers (1 x 1 conv layers) are used to reduce the number of the features (parameters).

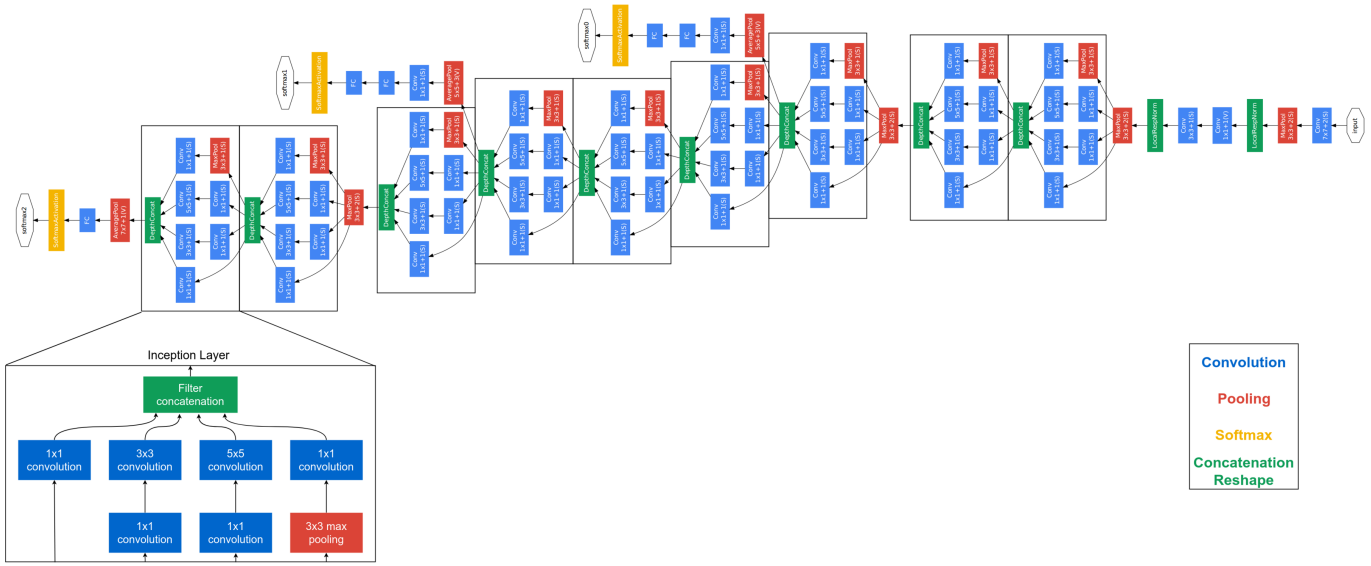
1 x 1 Conv Layer Reduces Depth of Tensor



1 x 1 Conv Layer Reduces Depth of Tensor



Full GoogleNet



Full GoogleNet

- Totally, 22 layers;

Full GoogleNet

- Totally, 22 layers;
- Starts with some vanilla layers (called stem network),

Full GoogleNet

- Totally, 22 layers;
- Starts with some vanilla layers (called stem network),
- Followed by 9 stacked inception layers,

Full GoogleNet

- Totally, 22 layers;
- Starts with some vanilla layers (called stem network),
- Followed by 9 stacked inception layers,
- Auxiliary classification outputs to inject additional gradient at lower layers.

Full GoogleNet

- Totally, 22 layers;
- Starts with some vanilla layers (called stem network),
- Followed by 9 stacked inception layers,
- Auxiliary classification outputs to inject additional gradient at lower layers.
- No fully connected layers.

Full GoogleNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

ResNet

- Winner the ImageNet ILSVRC challenge 2015, beating all other methods by large margins.
- It goes much deeper than previous methods with 152 layers.

ResNet

- Winner the ImageNet ILSVRC challenge 2015, beating all other methods by large margins.
- It goes much deeper than previous methods with 152 layers.
- It brings about the revolution of depth.

ResNet

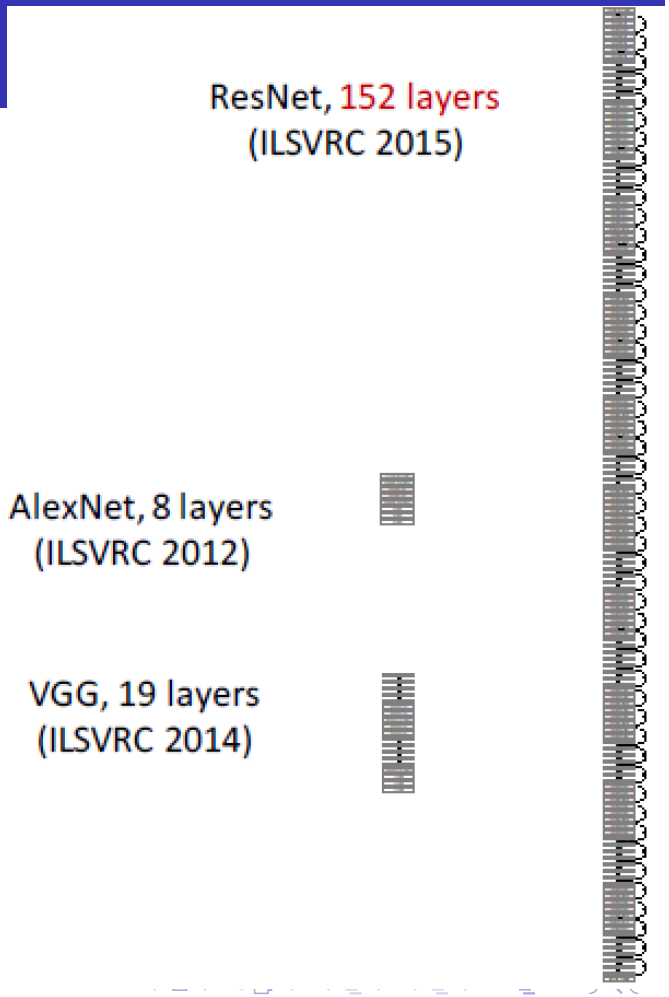
- Winner the ImageNet ILSVRC challenge 2015, beating all other methods by large margins.
- It goes much deeper than previous methods with 152 layers.
- It brings about the revolution of depth.
- But, how to go deeper?

ResNet

- Winner the ImageNet ILSVRC challenge 2015, beating all other methods by large margins.
- It goes much deeper than previous methods with 152 layers.
- It brings about the revolution of depth.
- But, how to go deeper? Simply stacking more layers does not work.

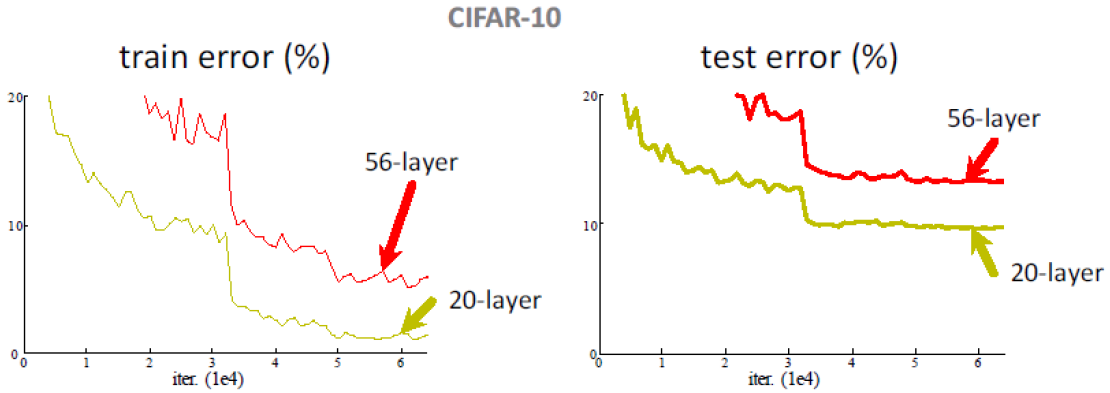
ResNet

- Winner the ImageNet ILSVRC challenge 2015, beating all other methods by large margins.
- It goes much deeper than previous methods with 152 layers.
- It brings about the revolution of depth.
- But, how to go deeper? Simply stacking more layers does not work.

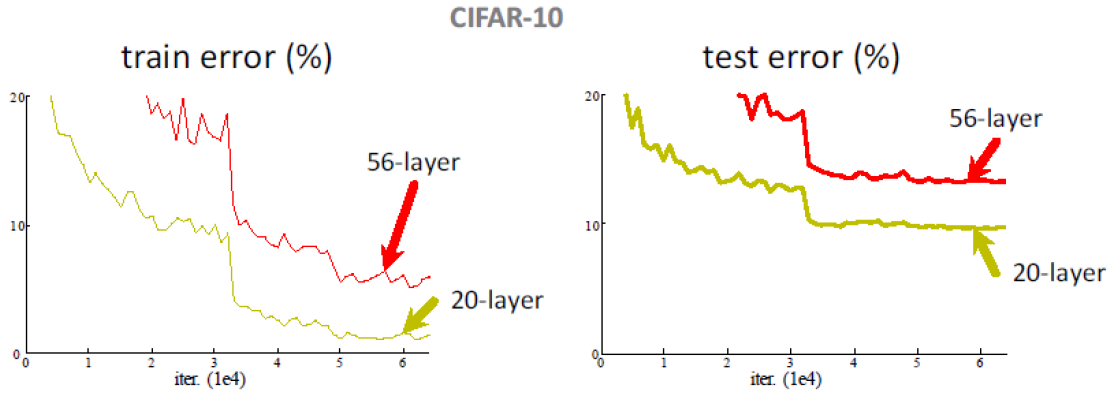


Simply Stacking Many Layers?

Simply Stacking Many Layers?

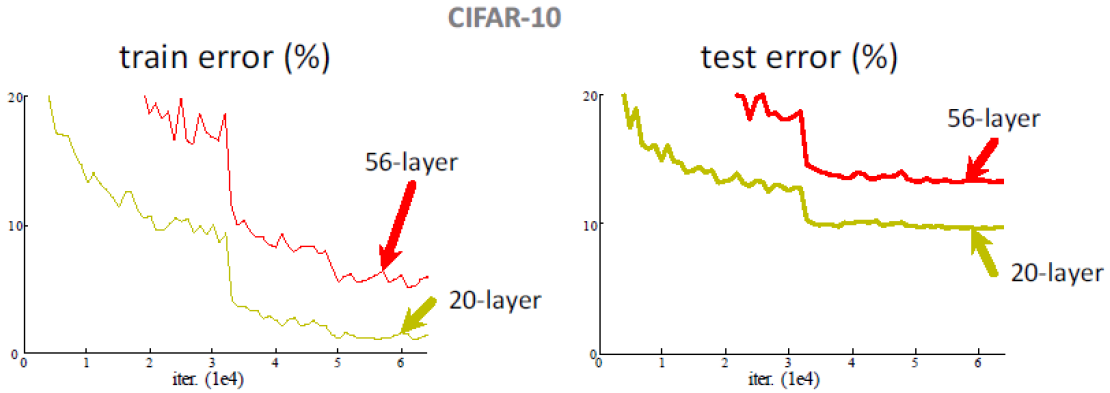


Simply Stacking Many Layers?



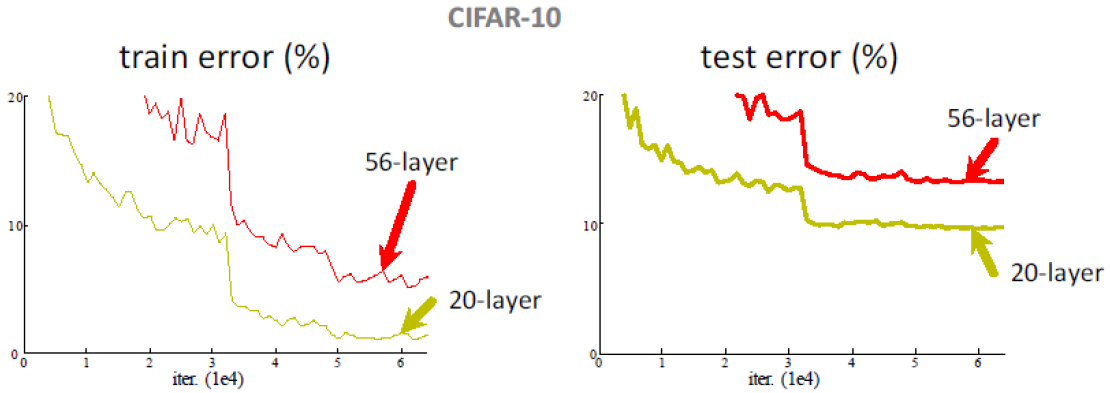
- Stacking 3x3 conv layers.

Simply Stacking Many Layers?



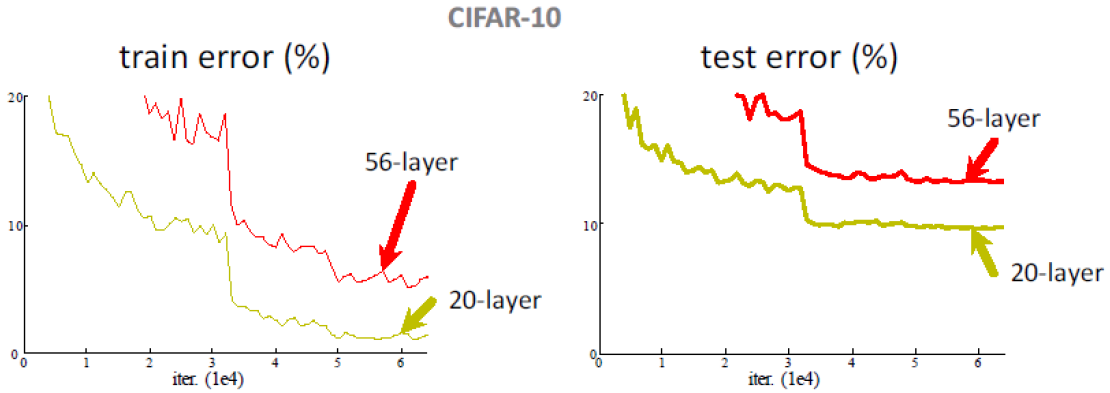
- Stacking 3x3 conv layers.
- 56-layer net has **higher training error** as well as test error.

Simply Stacking Many Layers?



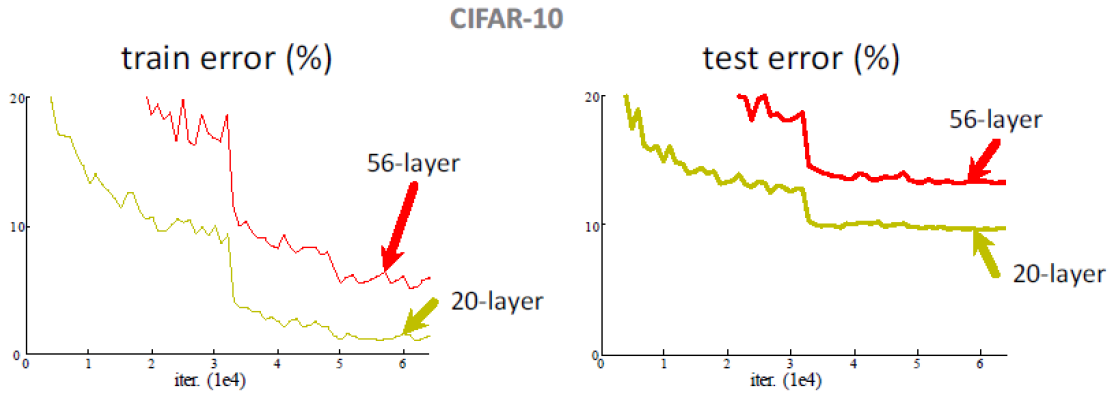
- Stacking 3x3 conv layers.
- 56-layer net has **higher training error** as well as test error.
- Usually, models with higher capacity should have lower training errors.

Simply Stacking Many Layers?



- Stacking 3x3 conv layers.
- 56-layer net has **higher training error** as well as test error.
- Usually, models with higher capacity should have lower training errors.
- It is not the case here. This suggests difficulties in training very deep models.

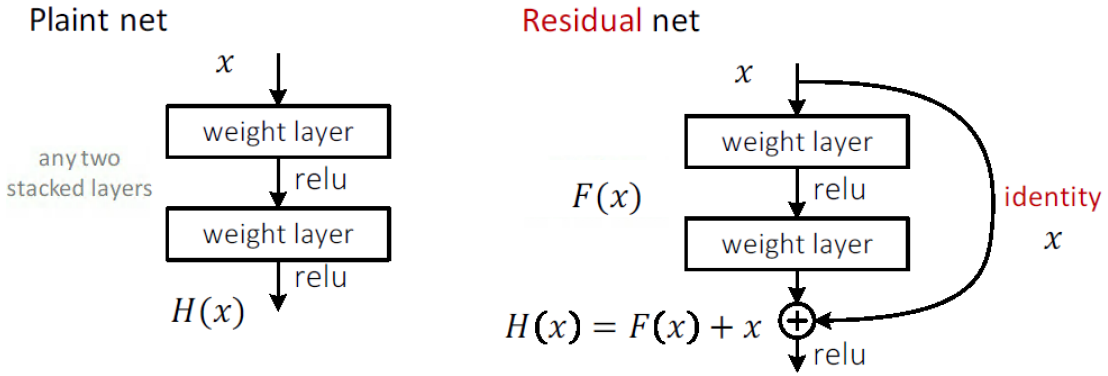
Simply Stacking Many Layers?



- Stacking 3x3 conv layers.
- 56-layer net has **higher training error** as well as test error.
- Usually, models with higher capacity should have lower training errors.
- It is not the case here. This suggests difficulties in training very deep models.
- It is difficult for gradients to propagate back to lower layers.

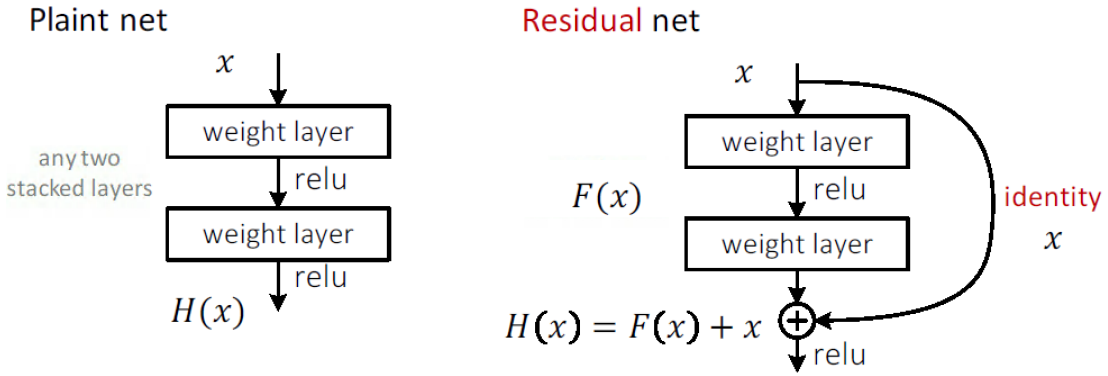
Plain Modules vs Residual Modules

ResNet stacks residual modules instead of plain modules



Plain Modules vs Residual Modules

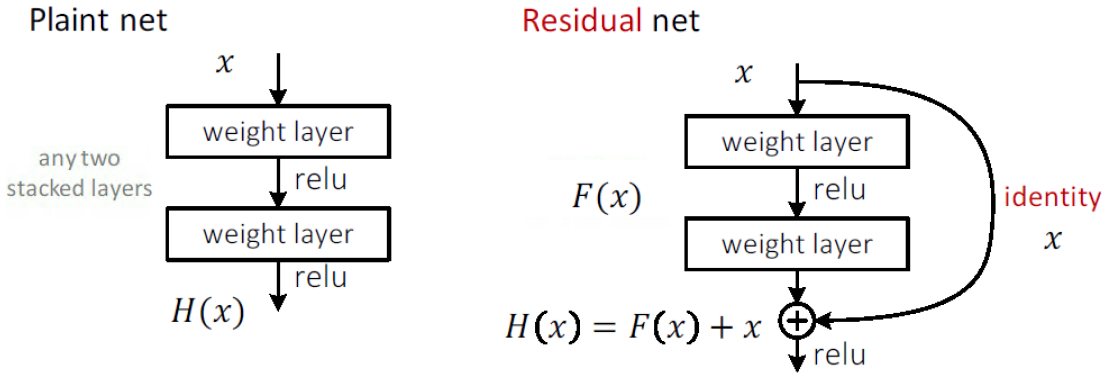
ResNet stacks residual modules instead of plain modules



- In a **plain module**, we try to represent a target function $H(x)$ using plain layers

Plain Modules vs Residual Modules

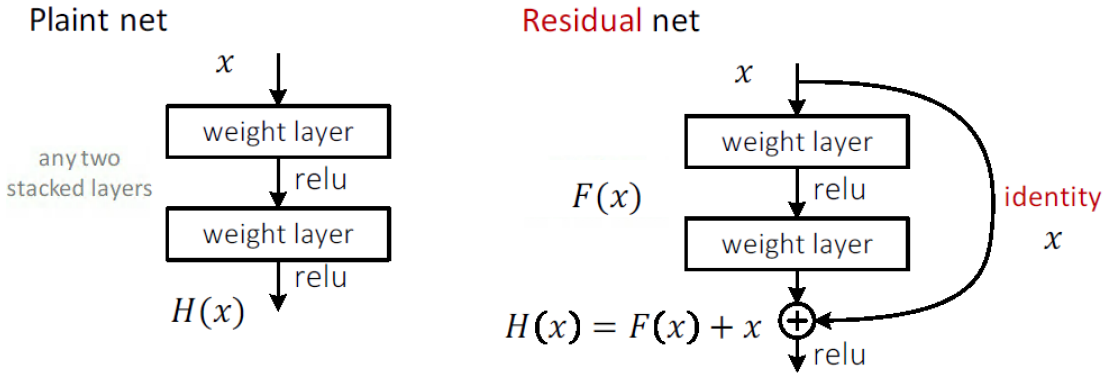
ResNet stacks residual modules instead of plain modules



- In a **plain module**, we try to represent a target function $H(x)$ using plain layers
- In a **residual module**, we have a identity connection from input to output, and use plain layers to represent the difference $F(x) = H(x) - x$,

Plain Modules vs Residual Modules

ResNet stacks residual modules instead of plain modules

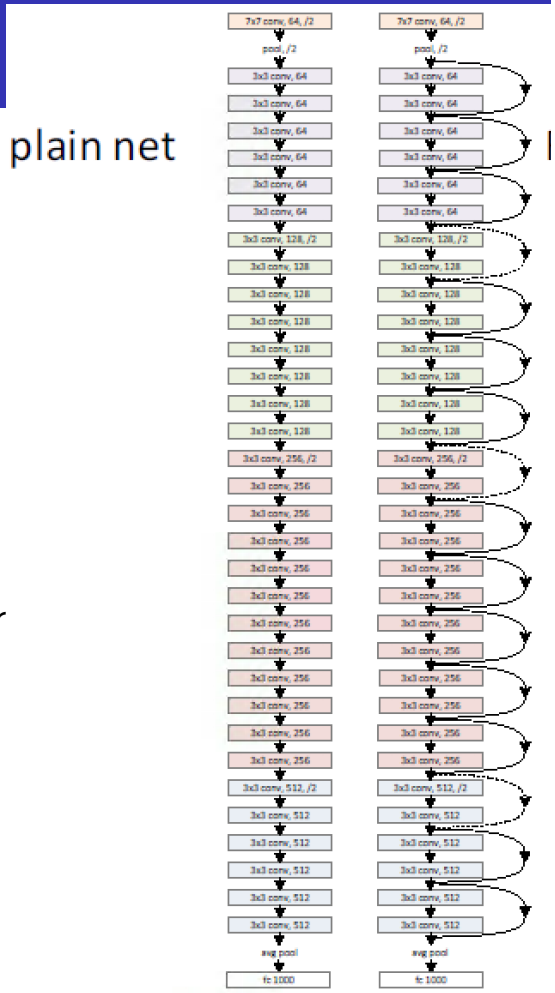


- In a **plain module**, we try to represent a target function $H(x)$ using plain layers
- In a **residual module**, we have a identity connection from input to output, and use plain layers to represent the difference $F(x) = H(x) - x$, which is called **residual**.

ResNet

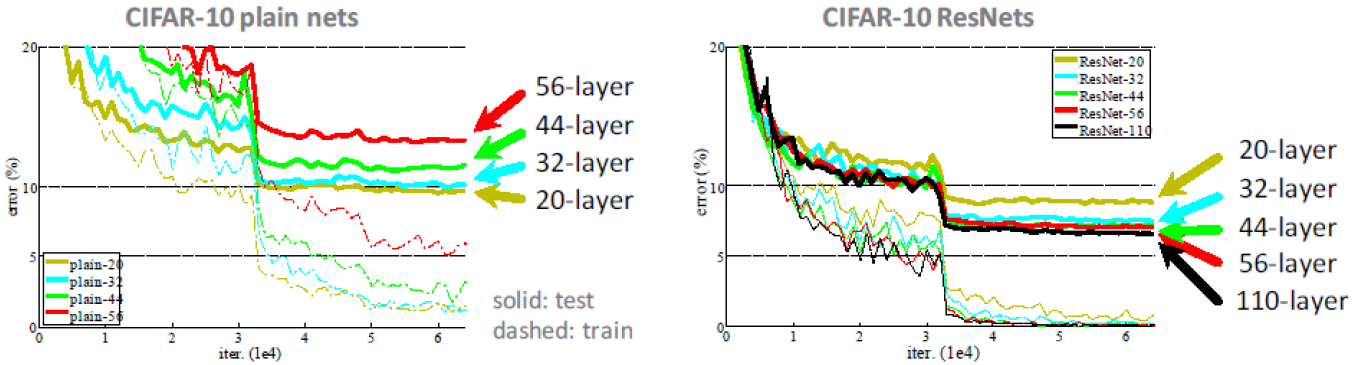
- In plain net, it is difficult for gradients to back propagate to lower layers.

ResNet



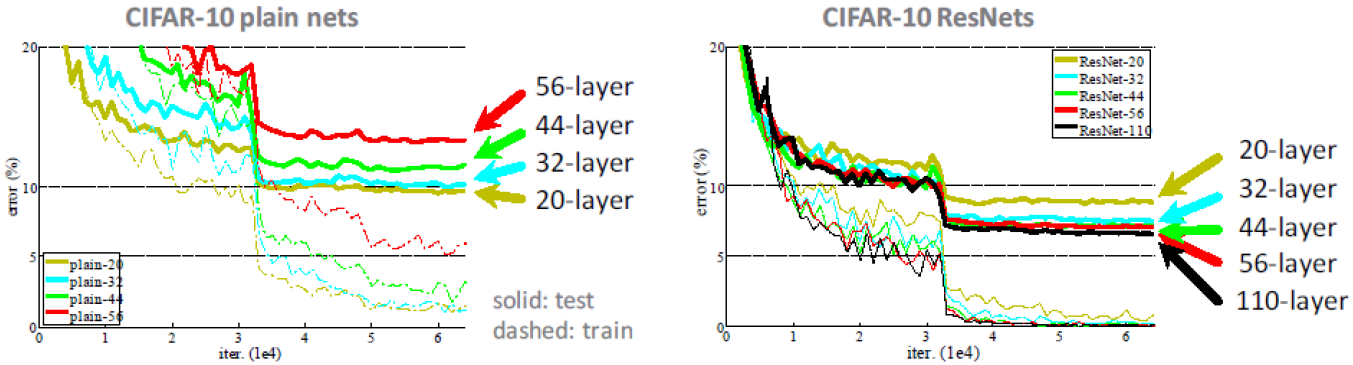
- In plain net, it is difficult for gradients to back propagate to lower layers.
- In ResNet, gradients can back propagate to lower layers because of the identity connections.

Simply Stacking Many Layers?



- With plain net, higher training and test errors observed with more layers (for deep architectures)

Simply Stacking Many Layers?



- With plain net, higher training and test errors observed with more layers (for deep architectures)
- With ResNet, lower training and test errors observed with more layers (for deep architectures).

Training of CNNs

- CNNs are trained the same ways as feedforward networks.

Training of CNNs

- CNNs are trained the same ways as feedforward networks. There are some practical tips to improve training and the results.

Training of CNNs

- CNNs are trained the same ways as feedforward networks. There are some practical tips to improve training and the results.
- **Data augmentation** is usually used to improve generalization.
  - **Data augmentation** means to create fake data and add them to the training set.
  - Fake images can be created by translating, rotating or scaling real images.

Training of CNNs

- CNNs are trained the same ways as feedforward networks. There are some practical tips to improve training and the results.
- **Data augmentation** is usually used to improve generalization.
  - **Data augmentation** means to create fake data and add them to the training set.
  - Fake images can be created by translating, rotating or scaling real images.

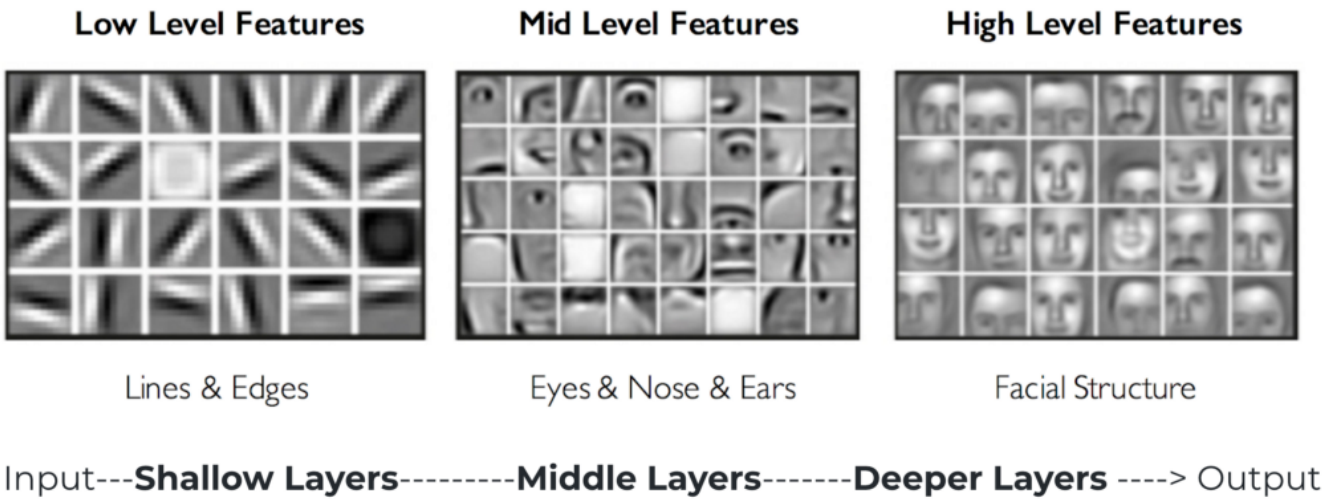
Popular CNN Architectures

Structure	# Parameters (M)	Storage Space (MB)
VGG16	138	500
ResNet50	25	98
ResNet101	44	171
InceptionV3	24	92
Xception	23	88
DenseNet121	8	33
DenseNet169	14	57
DenseNet201	20	80
EfficientNetB0	5.3	20
EfficientNetB2	9	36

Pre-trained version of the models can be downloaded from:
`https://keras.io/api/applications/`

Hierarchical Representation Learning

It is observed in multiple studies that neurons at lower layers detect simple features like edge and color, while neurons at higher layers can detect complex features like objects and body parts.



CNNs do NOT Think Like Humans

https://www.youtube.com/watch?v=YFL-MI5xzgg&t=105s