Convolutional Neural Networks

Arjun Krishna

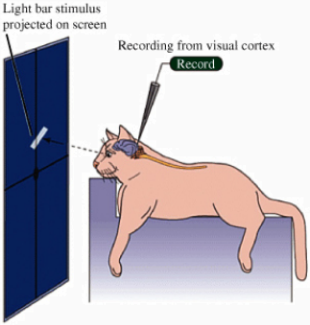
Indian Institute of Technology, Madras

Table of contents

Biological Motivation

Biological Motivation

In the early 1950s, David Hubel and Torsten Wiesel performed several interesting experiments on mammalian visual cortex.



Their experiments suggested the following:

- Nearby cells process information from nearby visual fields
- Cells with similar functions are organized into columns, tiny computational machines that relay information to a higher region of the brain

The Computational Model

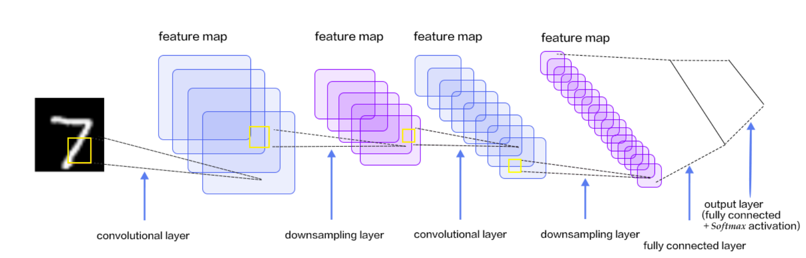


Figure 1: LeNet Architecture for digit recognition proposed LeCun et al.

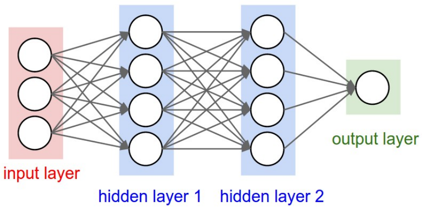
See `http://yann.lecun.com/exdb/lenet/`

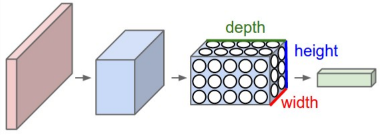
What are CNNs?

What are CNNs?

- Convolutional Neural Networks (CNNs) is very similar to the ordinary feed-forward networks.
- The have learn-able weights and biases and the neurons perform a dot product optionally followed by a non-linear activation.
- What changes is the fact that, the model makes an assumption that the input is an image or a grid shaped data.
- This helps encode some properties which tremendously reduce the number of parameters making the feed-forward efficient.

Regular Neural Networks vs ConvNet





- Neurons of a ConvNet is arranged in 3 dimension (width, hight, depth).
- Each layer transforms a 3D input volume to a 3D output volume.

The building blocks of ConvNets

Compnents of ConvNets

The three main type of layers used to build ConvNets are:

- Convolution Layer
- Pooling Layer
- Fully-Connected Layer

Convolution Layer

Input : 3D volume of dimension (W_i, H_i, C_i)

Parameters :

- **number of filters** = K
- **Size of filter** = (F, F, C_i)
- **padding** = P
- **stride** = (S, S)

Overview

- We have learn-able filters/kernels that is small spatially along width and height but extends through the depth of the input volume.

Convolution Layer

Input : 3D volume of dimension (W_i, H_i, C_i)

Parameters :

- **number of filters** = K
- **Size of filter** = (F, F, C_i)
- **padding** = P
- **stride** = (S, S)

Overview

- We have learn-able filters/kernels that is small spatially along width and height but extends through the depth of the input volume.
- Then the each filter is convolved with the input which outputs a 2D activation map.

Convolution Layer

Input : 3D volume of dimension (W_i, H_i, C_i)

Parameters :

- **number of filters** = K
- **Size of filter** = (F, F, C_i)
- **padding** = P
- **stride** = (S, S)

Overview

- We have learn-able filters/kernels that is small spatially along width and height but extends through the depth of the input volume.
- Then the each filter is convolved with the input which outputs a 2D activation map.
- Intuitively, each filter tries to look for features in the image. In the first few layer's it might look for edges in orientation or some blotch of colors. The higher layers look for complex patterns like a wheel etc., (recollect the visual cortex analogy).

Convolution Operation Example

$$
\begin{bmatrix} -1 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}
$$

Convolution Operation Example

$$
\begin{bmatrix} -1 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 & -1 \end{bmatrix}
*
\begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix}
=
\begin{bmatrix} -1 & & \\ & & \\ & & \end{bmatrix}
$$

Convolution Operation Example

$$
\begin{bmatrix}
-1 & 0 & 1 & 1 & 1 \\
1 & -1 & 0 & -1 & 0 \\
0 & 0 & 1 & 1 & 0 \\
-1 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & -1 & -1
\end{bmatrix}
*
\begin{bmatrix}
1 & 0 & -1 \\
1 & 1 & 1 \\
-1 & 0 & 1
\end{bmatrix}
=
\begin{bmatrix}
-1 & -2 & \\
 & & \\
 & &
\end{bmatrix}
$$

Convolution Operation Example

$$\begin{bmatrix} -1 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -2 \\ & & \\ & & \end{bmatrix}$$

Convolution Operation Example

$$
\begin{bmatrix} -1 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -2 \\ 3 & & \\ & & \end{bmatrix}
$$

Convolution Operation Example

$$
\begin{bmatrix} -1 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 & -1 \end{bmatrix}
*
\begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix}
=
\begin{bmatrix} -1 & -2 & -2 \\ 3 & 2 & \\ & & \end{bmatrix}
$$

Convolution Operation Example

$$
\begin{bmatrix} -1 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 & -1 \end{bmatrix}
*
\begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix}
=
\begin{bmatrix} -1 & -2 & -2 \\ 3 & 2 & 3 \\ & & \end{bmatrix}
$$

Convolution Operation Example

$$
\begin{bmatrix} -1 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ \boxed{0} & \boxed{0} & \boxed{1} & 1 & 0 \\ \boxed{-1} & \boxed{1} & \boxed{0} & 1 & 1 \\ \boxed{1} & \boxed{0} & \boxed{0} & -1 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -2 \\ 3 & 2 & 3 \\ -2 & & \end{bmatrix}
$$

Convolution Operation Example

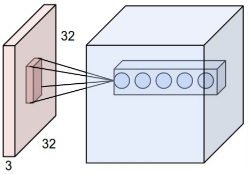
$$\begin{bmatrix} -1 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -2 \\ 3 & 2 & 3 \\ -2 & 0 & \end{bmatrix}$$

Convolution Operation Example

$$\begin{bmatrix} -1 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -2 \\ 3 & 2 & 3 \\ -2 & 0 & 2 \end{bmatrix}$$

All the parameters of CNN control the output volume of neuron activation maps.

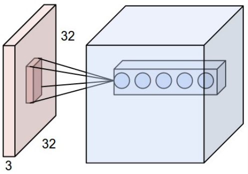
- The depth of the output layer is equal to the number of filters/kernels. Each filter/kernel is looking for something specific in the input image.



Spatial Arrangement of Neurons

All the parameters of CNN control the output volume of neuron activation maps.

- The depth of the output layer is equal to the number of filters/kernels. Each filter/kernel is looking for something specific in the input image.

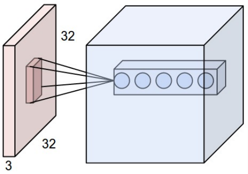


- The **stride** hyper-parameter specifies the number of pixels to move by when sliding along the image during convolution. Higher the stride lower is the output's spatial dimension.

Spatial Arrangement of Neurons

All the parameters of CNN control the output volume of neuron activation maps.

- The depth of the output layer is equal to the number of filters/kernels. Each filter/kernel is looking for something specific in the input image.



- The **stride** hyper-parameter specifies the number of pixels to move by when sliding along the image during convolution. Higher the stride lower is the output's spatial dimension.
- The **zero-padding** hyper-parameter P, helps control the output dimension by zero-padding the image along the borders.

Calculating the Output Dimension

Input Volume Size : W
Filter Size/ Receptive Field Size : F
Stride : S
Zero-Padding : P
Number of Filters: K

Output spatial dimension $M = \frac{(W+2P-F)}{S} + 1$
Output Volume Size = $(M \times M \times K)$

Note that M must be an integer. This imposes some constraint on the choice of hyper-parameters

Another Example

Padding $= 1$, Stride $= 2$, Filter Size $= 3$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$$

Another Example

Padding $= 1$, Stride $= 2$, Filter Size $= 3$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & \\ & \end{bmatrix}$$

Another Example

Padding $= 1$, Stride $= 2$, Filter Size $= 3$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 \\ & \end{bmatrix}$$

Another Example

Padding $= 1$, Stride $= 2$, Filter Size $= 3$

$$
\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 \\ 0 & \end{bmatrix}
$$

Another Example

Padding $= 1$, Stride $= 2$, Filter Size $= 3$

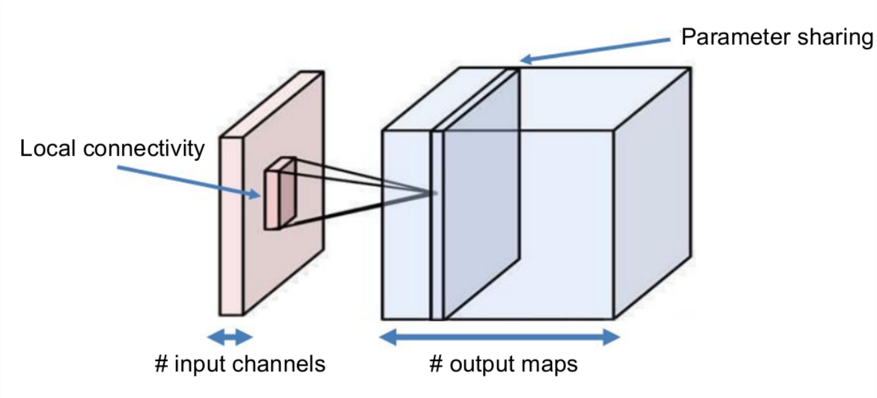
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 \\ 0 & 2 \end{bmatrix}$$

The key take-aways of ConvNets

- **Local Connectivity**: When dealing with high dimensional data like images it becomes impractical to use a fully connected neural network. ConvNets are modeled such that the neuron sees only a part of the image known as the receptive field (equivalent to the filter size). This reduces the number of parameters of the network significantly.
- **Parameter Sharing**: To further reduce the number of parameter, the parameter sharing scheme is employed. Along a depth slice of the output, the parameters for the weight is the same which is the filter that produced the activation map.

The total number of parameters of a Convolution Layer is $(F \cdot F \cdot C) \cdot K$

The key take-aways of ConvNets



Pooling Layer

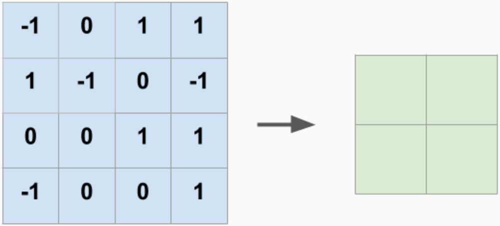
- It is very common to insert a Pooling Layer in between successive Convolution Layers.
- The main objective of the pooling layer is to reduce the spatial extent of the input, thereby reducing the parameters required in successive layers.
- The most common pooling layer is the MAX-pool, which outputs the max of values in a receptive field.

Input volume size: (W, W, D)
Parameters:

- **Spatial Extent** = F
- **Stride** = S

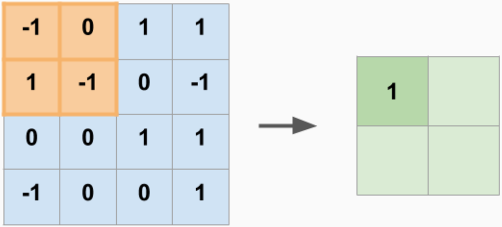
Output spatial dimension $M = \frac{W-F}{S} + 1$
Output volume size: (M, M, D)

Stride $= 2$, Filter Size $= 2$



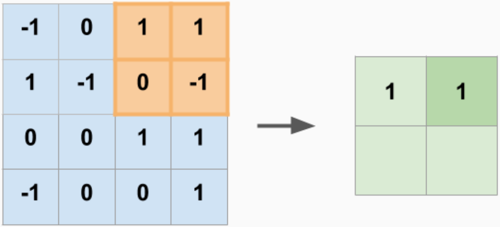
MAX-pooling example

Stride $= 2$, Filter Size $= 2$

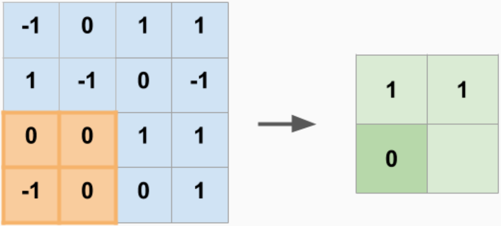


MAX-pooling example

Stride $= 2$, Filter Size $= 2$

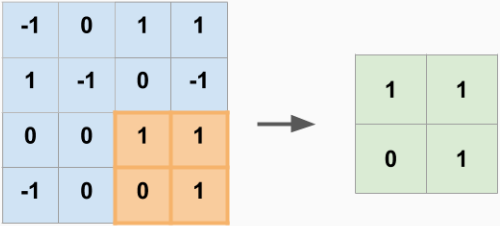


Stride $= 2$, Filter Size $= 2$



MAX-pooling example

Stride $= 2$, Filter Size $= 2$



ConvNet Architectures

ConvNet Architectures

- Typically ConvNet Architectures have the following layer pattern.

```
INPUT -> [[CONV -> RELU]*N -> POOL]*M -> [FC -> RELU]*L
```

- It is usually the case that for Conv Layers small filter size 3×3 or 5×5 are preferred over very large filter sizes. This helps us express more important features.
- In practice, people choose architectures that have worked very well on the ImageNet data. And rarely do people train very big ConvNet's from scratch.

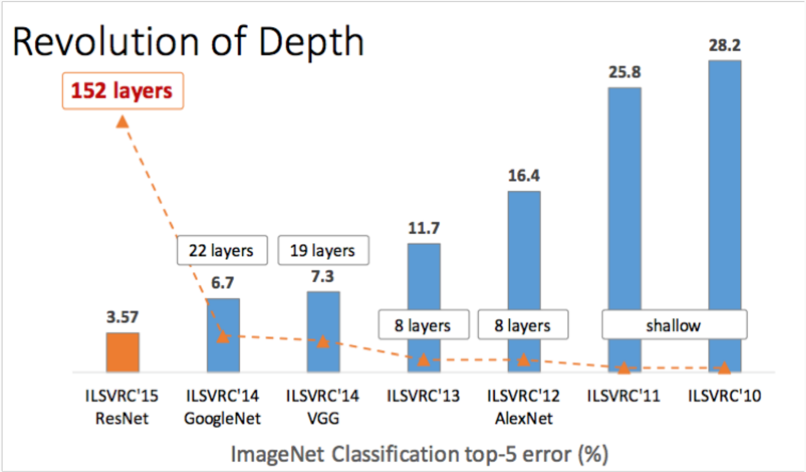
ImageNet dataset

IMAGENET

- Has about 14 million image URLs that are hand-annotated.
- Has over 20,000 object categories.
- **ImageNet Challenge**
    - 1,000 object categories
    - Images: 1.2 M train and 100k test.

Performance of Deep Architectures in ILSVRC

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Case Study: VGGNet

Name	Size	Memory	Parameters
INPUT	$(224, 224, 3)$	150K	0
CONV3-64	$(224, 224, 64)$	3.2M	(3*3*3)*64 = 1,728
CONV3-64	$(224, 224, 64)$	3.2M	(3*3*64)*64 = 36,864
POOL2	$(112, 112, 64)$	800K	0
CONV3-128	$(112, 112, 128)$	1.6M	(3*3*64)*128 = 73,728
CONV3-128	$(112, 112, 128)$	1.6M	(3*3*128)*128 = 147,456
POOL2	$(56, 56, 128)$	800K	0
CONV3-256	$(56, 56, 256)$	800K	(3*3*128)*256 = 294,912
CONV3-256	$(56, 56, 256)$	800K	(3*3*256)*256 = 589,824
CONV3-256	$(56, 56, 256)$	800K	(3*3*256)*256 = 589,824
POOL2	$(28, 28, 256)$	200K	0
CONV3-512	$(28, 28, 512)$	400K	(3*3*256)*512 = 1,179,648
CONV3-512	$(28, 28, 512)$	400K	(3*3*512)*512 = 2,359,296
CONV3-512	$(28, 28, 512)$	400K	(3*3*512)*512 = 2,359,296
POOL2	$(14, 14, 512)$	100K	0

Case Study: VGGNet

CONV3-512	$(14, 14, 512)$	100K	(3*3*512)*512 = 2,359,296
CONV3-512	$(14, 14, 512)$	100K	(3*3*512)*512 = 2,359,296
CONV3-512	$(14, 14, 512)$	100K	(3*3*512)*512 = 2,359,296
POOL2	$(7, 7, 512)$	25K	0
FC	4096	4096	7*7*512*4096 = 102,760,448
FC	4096	4096	4096*4096 = 16,777,216
FC	1000	1000	4096*1000 = 4,096,000

Total Memory = 24M * 4 bytes = 93MB / image [feed-forward]

Total Number of Parameters = 138M

ConvNet Visualization

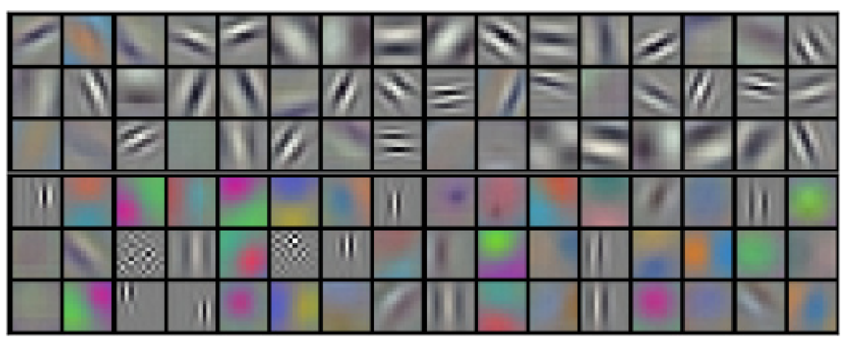
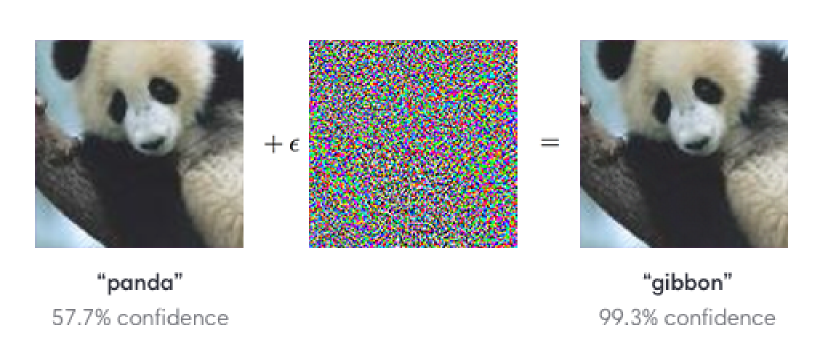


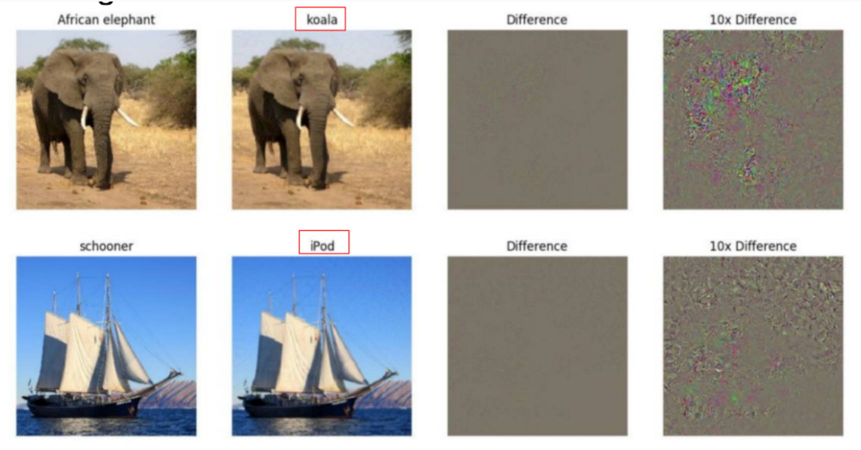
Figure 2: Filters learned by the first convolutional layer, AlexNet. From Krizehvsky et al. (2012)

Fooling ConvNets

ConvNets can be easily fooled by adversarial examples.



Fooling ConvNets



TensorFlow implementation

We will use the following Tensorflow-v1.7 API's for implementing a simple ConvNet Architecture for MNIST Digit Recognition Task

`Convolution : tf.layers.conv2d`

`Max-Pool: tf.nn.max_pool`

`Fully-Connected: tf.layers.dense`

TASK

Let's implement the following ConvNet Architecture in TensorFlow!

```
INPUT          : [28, 28, 1 ]
CONV3-32 (P=1) : [28, 28, 32]
POOL2          : [14, 14, 32]
CONV3-32 (P=1) : [14, 14, 32]
POOL2          : [ 7,  7, 32]
FC             : 500
FC             : 10
```

References i

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei.
Imagenet: A large-scale hierarchical image database.
In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio.
***Deep learning*, volume 1.**
MIT press Cambridge, 2016.

I. J. Goodfellow, J. Shlens, and C. Szegedy.
Explaining and harnessing adversarial examples.
arXiv preprint arXiv:1412.6572, 2014.

A. Karpathy.
Cs231n convolutional neural networks for visual recognition.
Neural networks, 1, 2016.

A. Krizhevsky, I. Sutskever, and G. E. Hinton.
Imagenet classification with deep convolutional neural networks.
In *Advances in neural information processing systems*, pages 1097–1105, 2012.