DS 4400

Machine Learning and Data Mining I
Spring 2024

David Liu
Khoury College of Computer Science
Northeastern University

March 22 2024

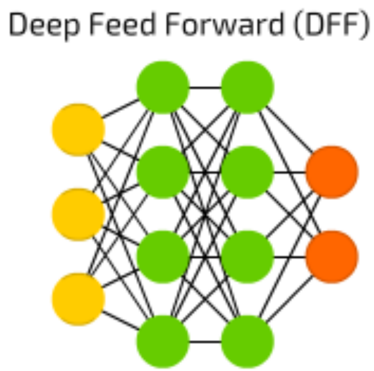
Outline

- Feed-forward neural networks
  - Activations
  - Softmax classifier
  - Architectures and parameters
- Convolutional neural networks
  - Convolution layer
  - Max pooling
  - Well-known convolutional networks architectures

Neural Network Architectures

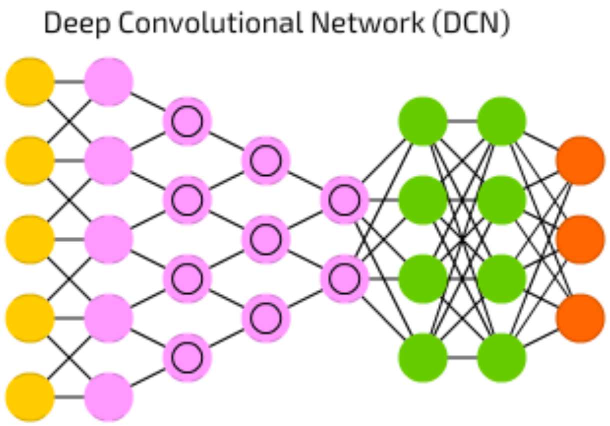
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



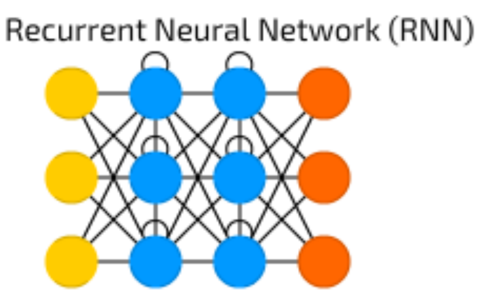
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

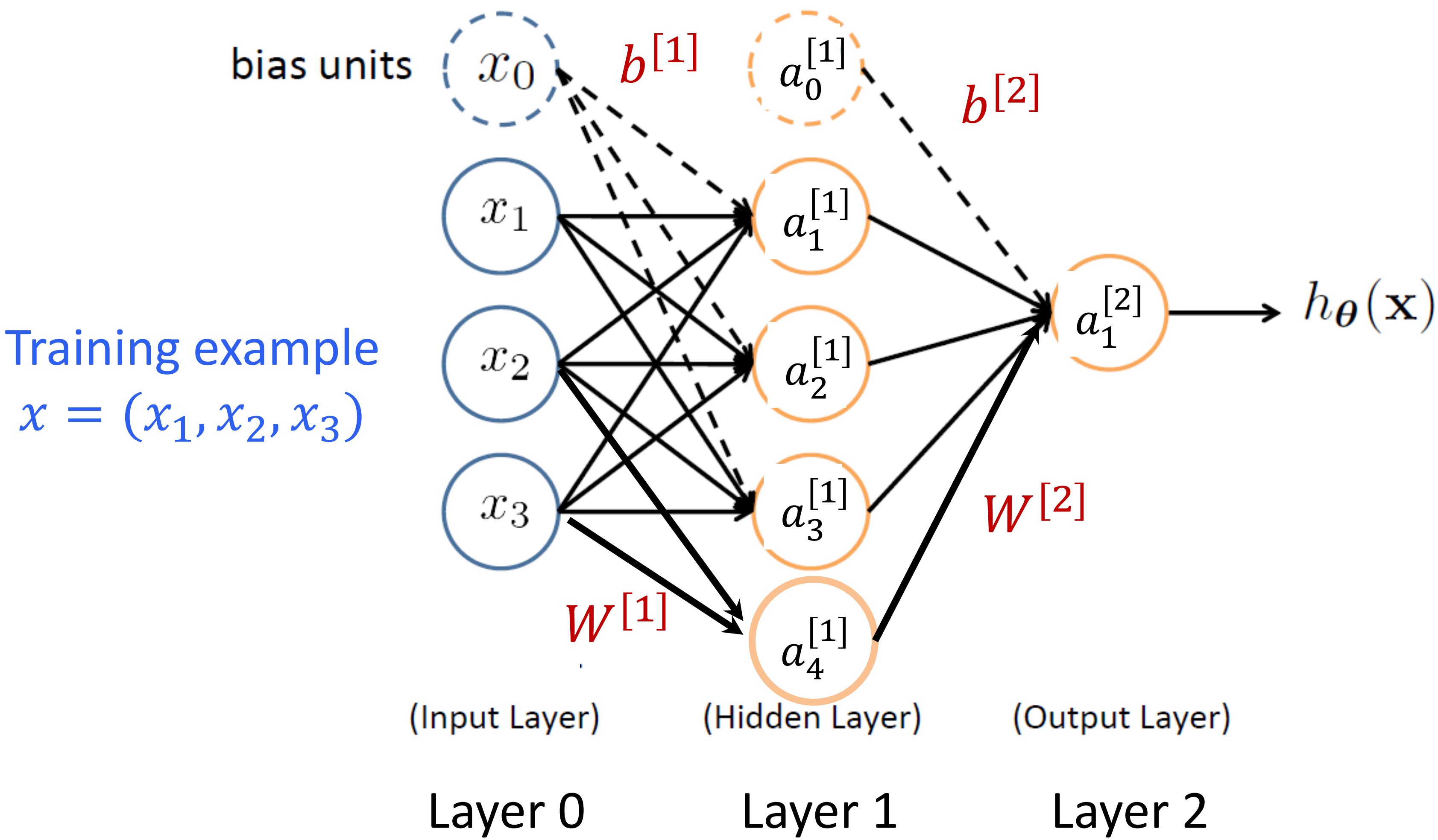


Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



Feed-Forward Neural Network



Feed-Forward NN

- Hyper-parameters
  – Number of layers
  – Architecture (how layers are connected)
  – Number of hidden units per layer
  – Number of units in output layer
  – Activation functions
- Other
  – Initialization
  – Regularization

Vectorization

$$z_1^{[1]} = W_1^{[1]}\ x + b_1^{[1]} \quad \text{and} \quad a_1^{[1]} = g(z_1^{[1]})$$
$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$
$$z_4^{[1]} = W_4^{[1]}\ x + b_4^{[1]} \quad \text{and} \quad a_4^{[1]} = g(z_4^{[1]})$$

$$\boxed{\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} — & W_1^{[1]} & — \\ — & W_2^{[1]} & — \\ & \vdots & \\ — & W_4^{[1]} & — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}}$$

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

Linear

$$a^{[1]} = g(z^{[1]})$$

Non-Linear

Vectorization

Output layer

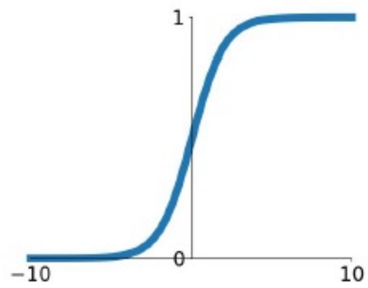
$$z_1^{[2]} = W_1^{[2]^T} a^{[1]} + b_1^{[2]} \quad \text{and} \quad a_1^{[2]} = g(z_1^{[2]})$$

$$\boxed{\underbrace{z^{[2]}}_{1\times 1} = \underbrace{W^{[2]}}_{1\times 4} \underbrace{a^{[1]}}_{4\times 1} + \underbrace{b^{[2]}}_{1\times 1} \quad \text{and} \quad \underbrace{a^{[2]}}_{1\times 1} = g(\underbrace{z^{[2]}}_{1\times 1})}$$

Non-Linear Activation Functions

Sigmoid

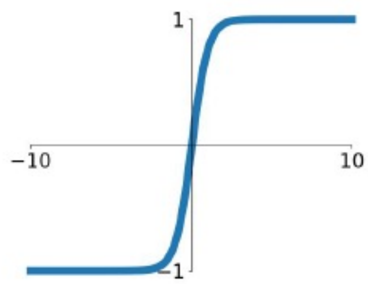
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Binary Classification

tanh

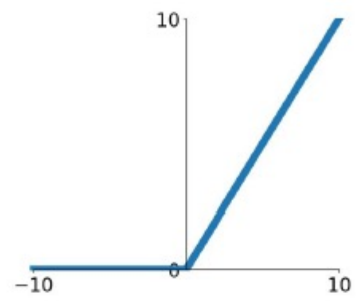
$$\tanh(x)$$



Regression

ReLU

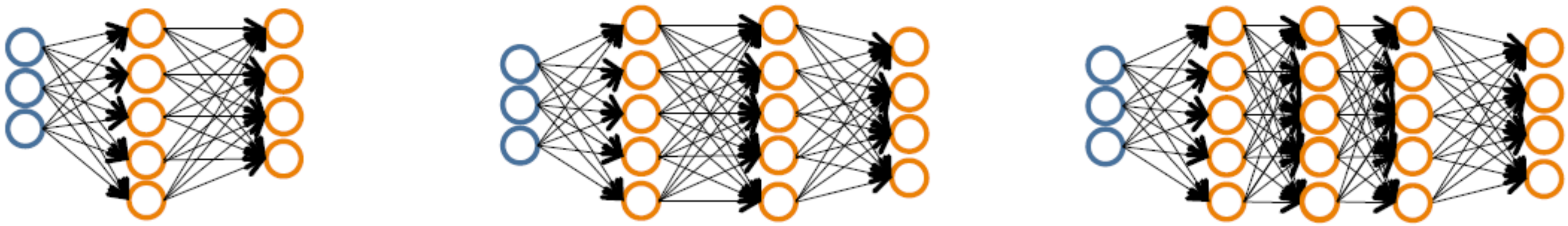
$$\max(0, x)$$



Intermediary layers

How to pick architecture?

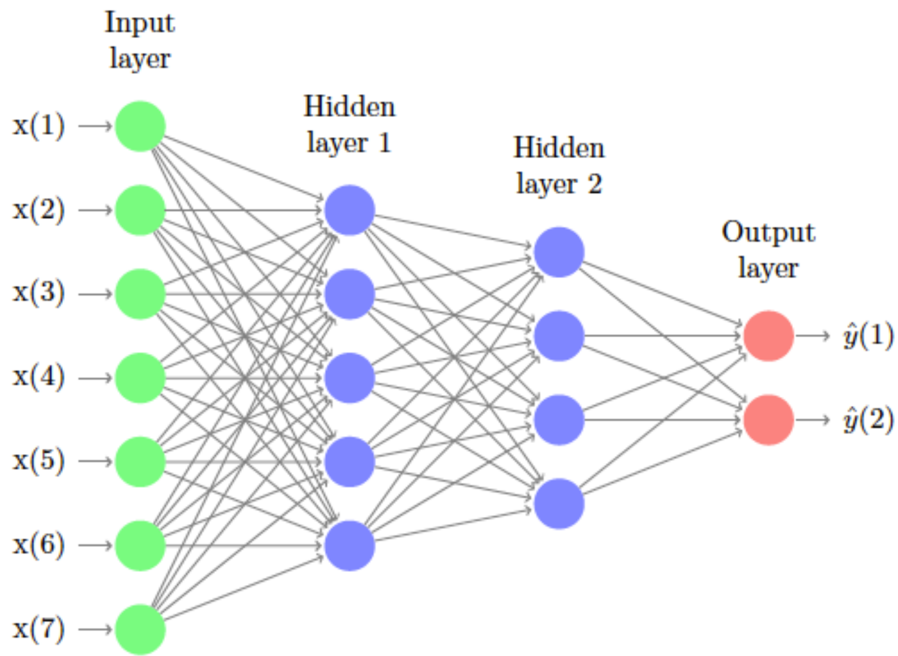
Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

Reasonable default: 1 hidden layer

FFNN Architectures



- Input and Output Layers are completely specified by the problem domain
- In the Hidden Layers, number of neurons in Layer i+1 is usually smaller or equal to the number of neurons in Layer i

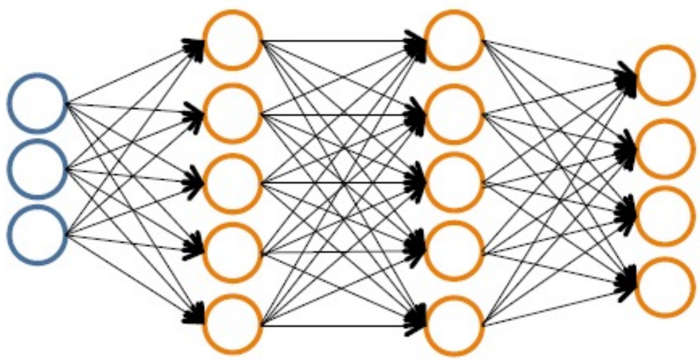
Multi-Class Classsification

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

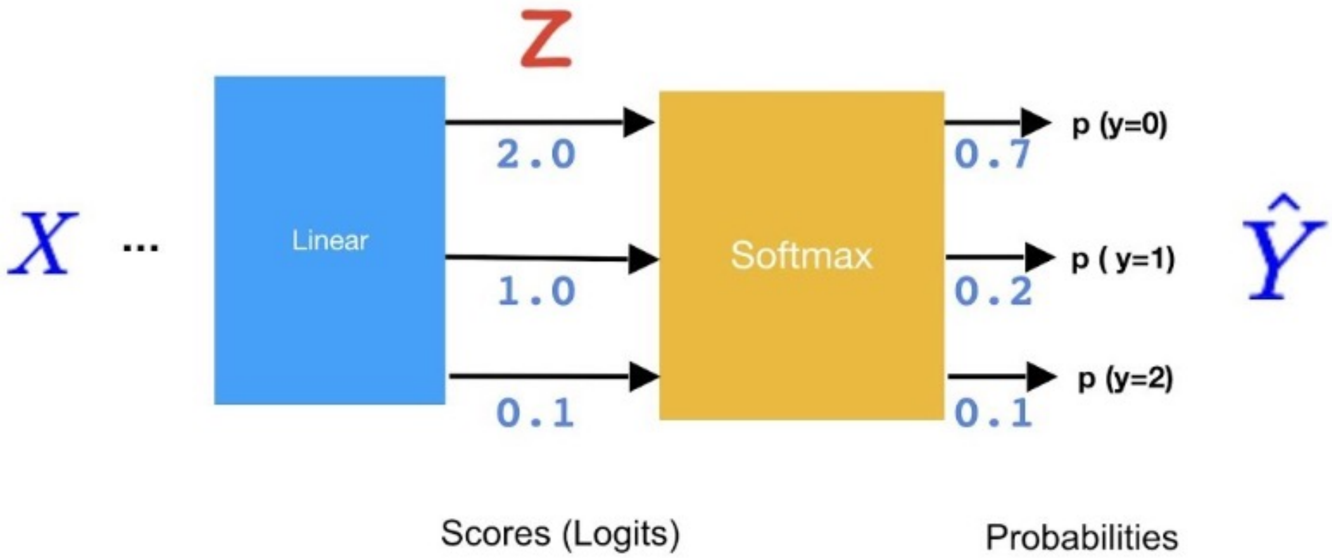
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$
when car

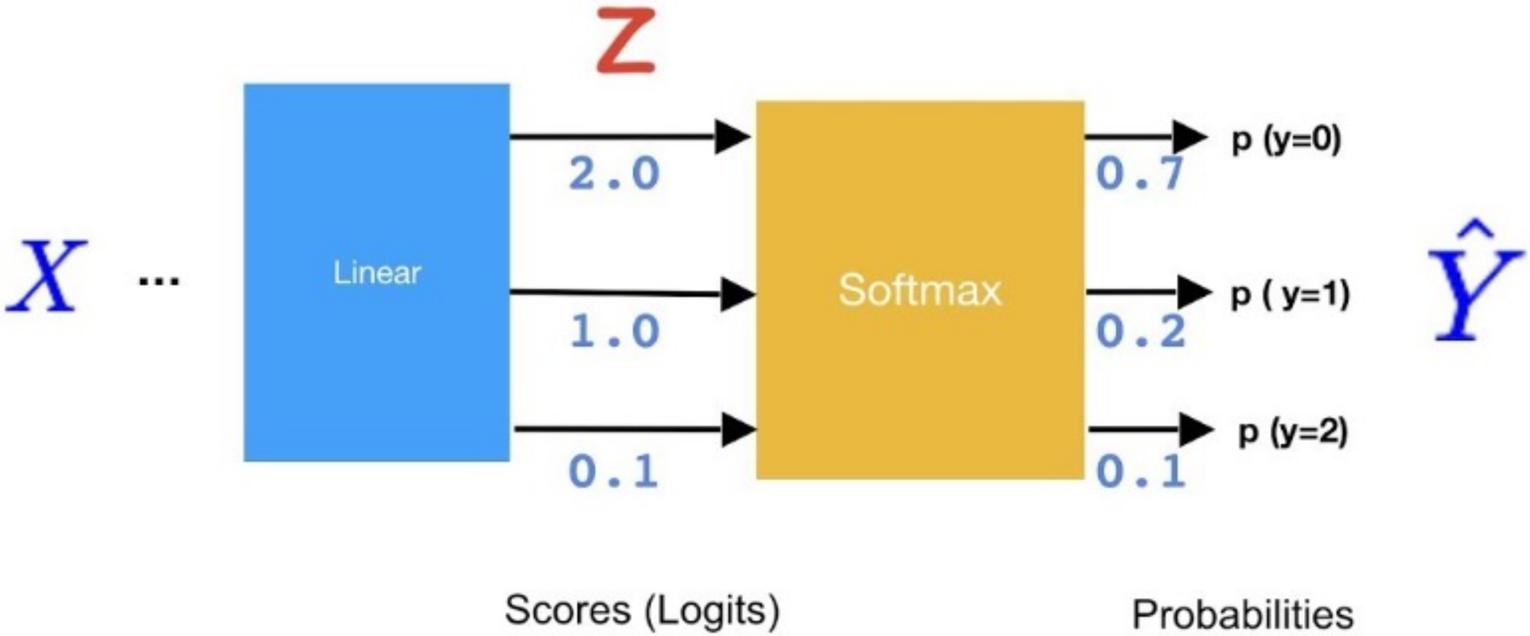
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$
when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$
when truck

Softmax classifier



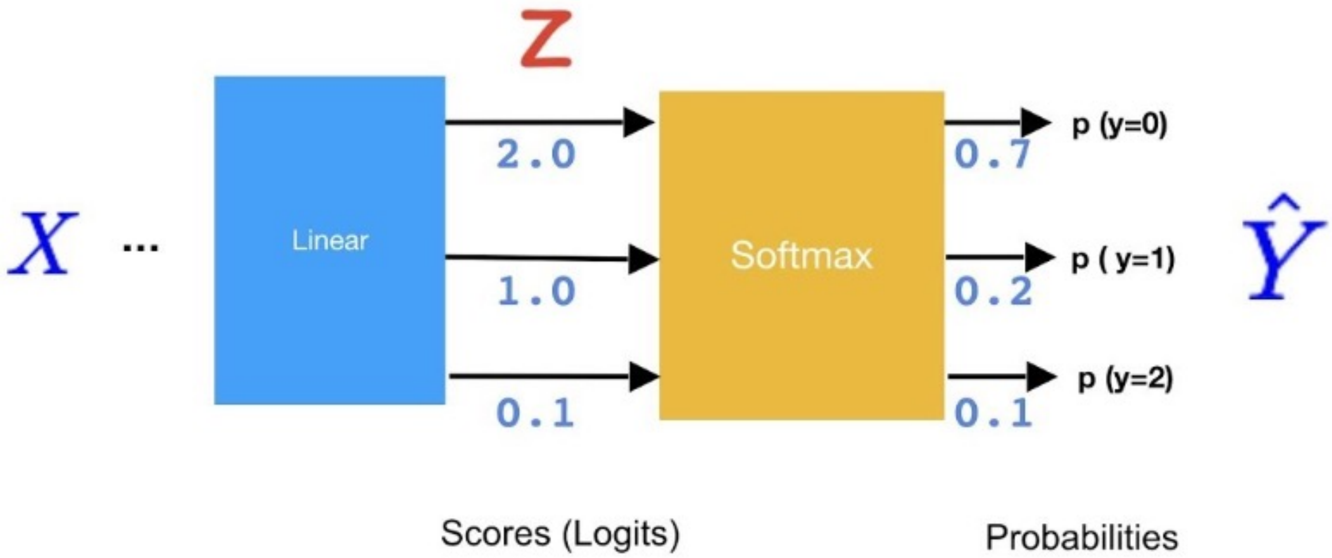
Softmax classifier



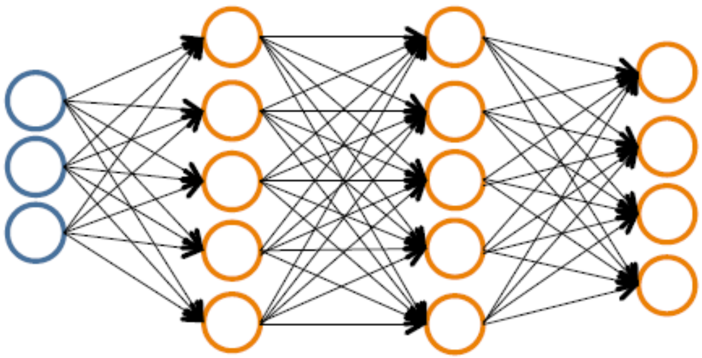
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Predict the class with highest probability
- Generalization of sigmoid/logistic regression to multi-class

Cross-entropy loss



Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$ contains # nodes at each layer

- $s_0 = d$ (# features)

Binary classification

y = 0 or 1

1 output unit $(s_{L-1} = 1)$

Sigmoid

Multi-class classification (K classes)

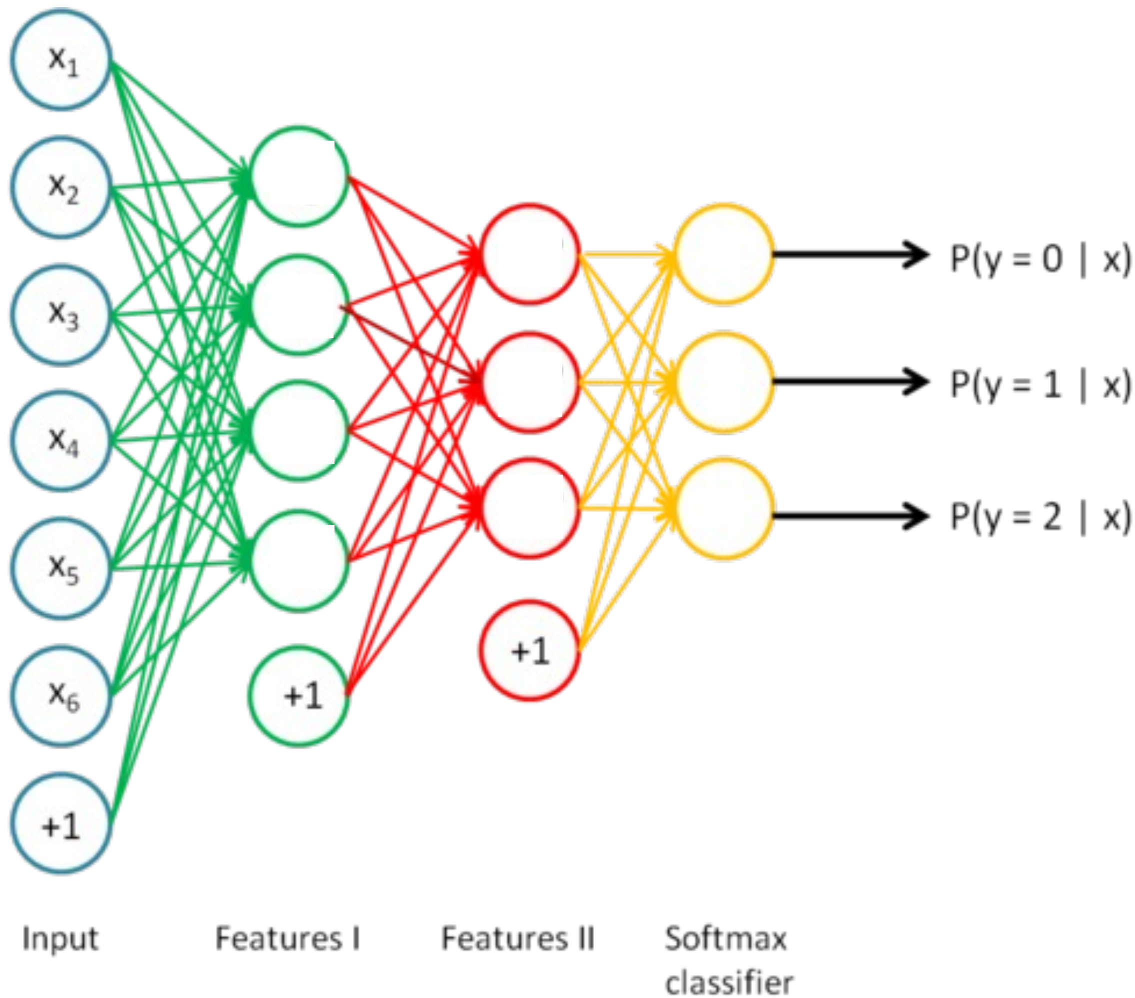
$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

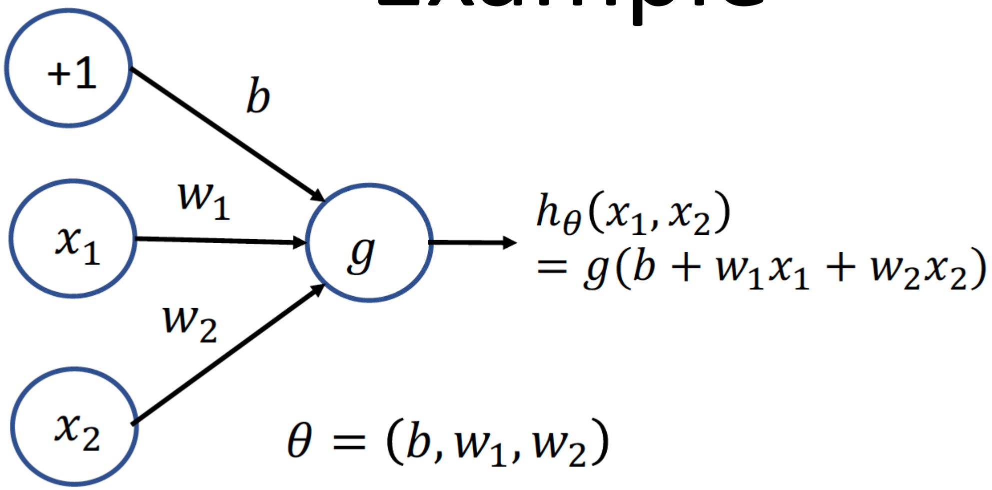
K output units $(s_{L-1} = K)$

Softmax

Multi-class classification



Example



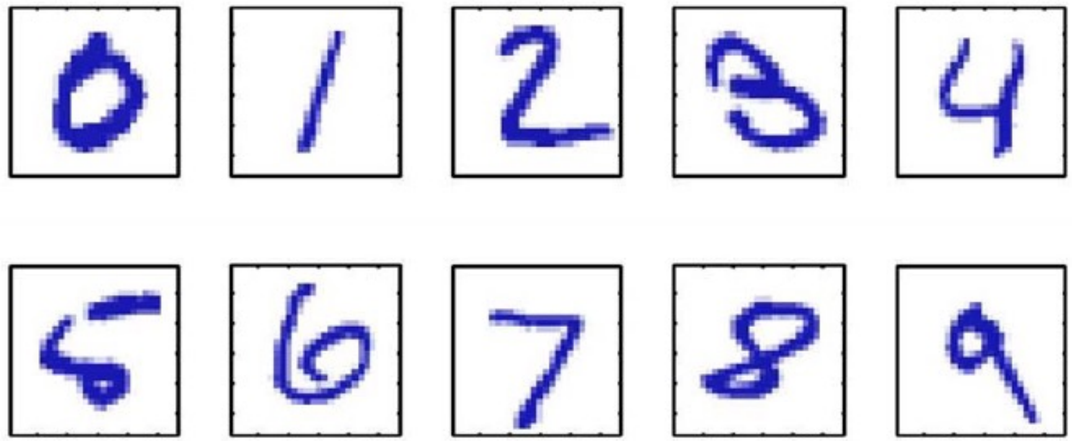
1. Given $b = -10, w_1 = 12, w_2 = 5$
Activation $g(z) = sign(z)$
Compute the output:

x_1	x_2	$h(x_1, x_2)$
0	0	
0	1	
1	0	
1	1	

2. Find out the weights $b,\ w_1, w_2$ and activation function to get the following output:

x_1	x_2	$h(x_1, x_2)$
0	0	1
0	1	1
1	0	1
1	1	0

MNIST: Handwritten digit recognition



Images are 28 x 28 pixels

Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$

Learn a classifier $f(\mathbf{x})$ such that,

$$f : \mathbf{x} \to \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Predict the digit

Multi-class classifier

Parameter Counting

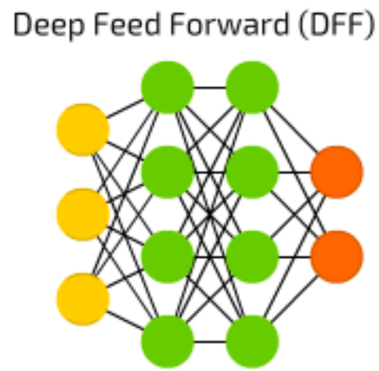
Review FFNN

- Feed-Forward Neural Networks are common neural networks architectures
  - Fully connected networks are called Multi-Layer Perceptron
  - Usually use 1 or 2 hidden layers
- Input, output, and hidden layers
  - Linear matrix operations followed by non-linear activations at every layer
- Activations:
  - ReLU, tanh, etc., for hidden layers
  - Sigmoid (binary classification) and softmax (for multi-class classification) at last layer
- Forward propagation: process of evaluating input through the network

Neural Network Architectures

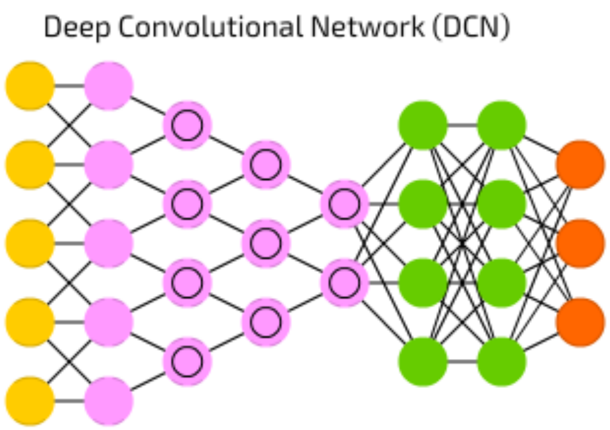
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



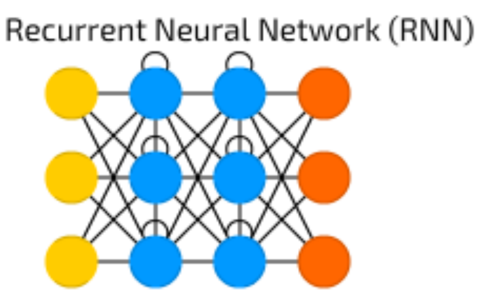
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations



Recurrent Networks

- Keep hidden state
- Have cycles in computational graph

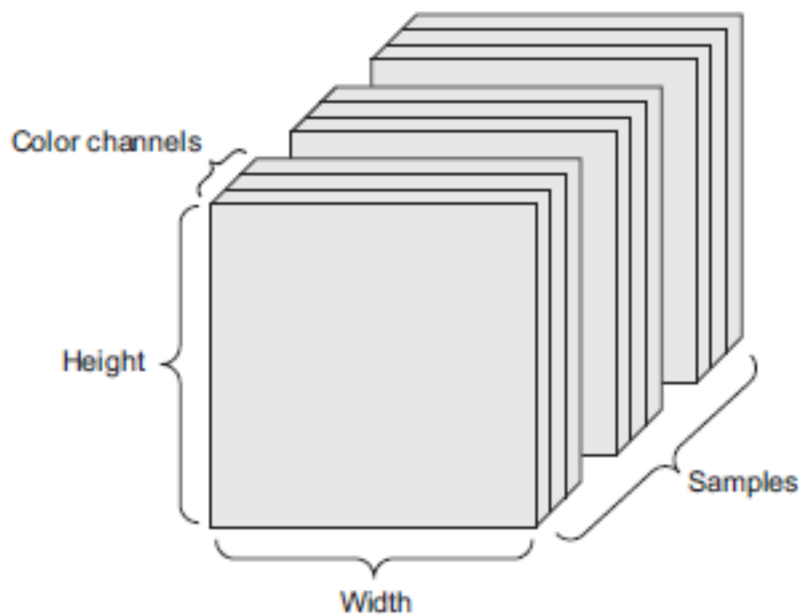


Convolutional Nets

- Neurons are connected from layer to the next
  - Invented by [LeCun 89]
- Applicable to data with natural grid topology
  - Time series
  - Images
- Use convolutions on at least one layer
  - Convolution is a linear operation that uses local information
  - Also use pooling operation
  - Used for dimensionality reduction and learning hierarchical feature representations

Image Representation

- Image is 3D "tensor": height, width, color channel (RGB)
- Black-and-white images are 2D matrices: height, width
  - Each value is pixel intensity

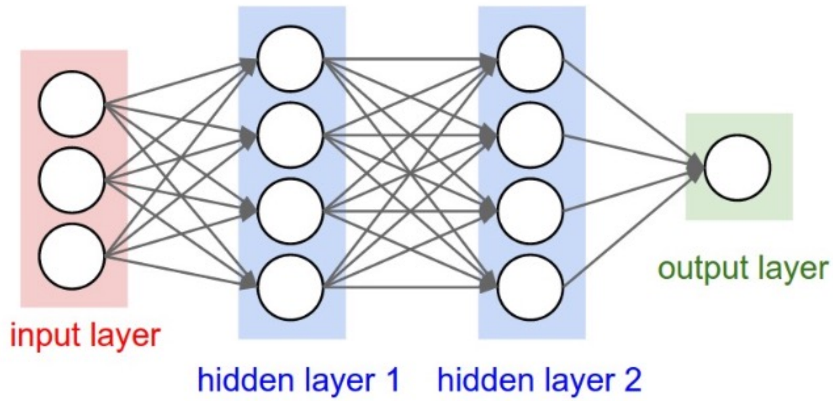


Computer vision principles

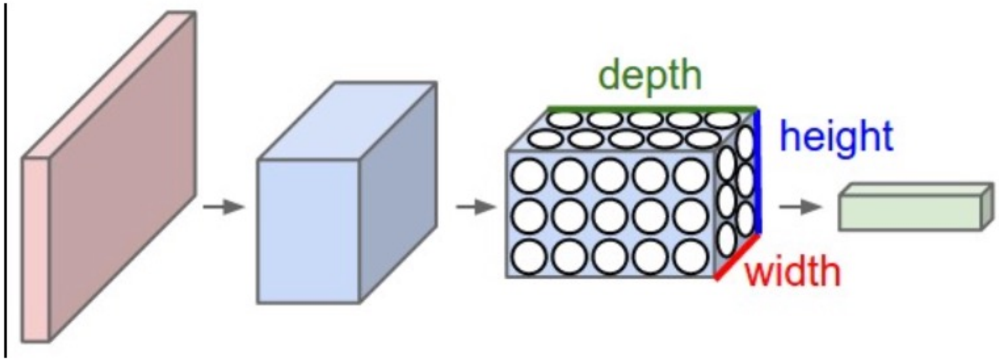
- Task: image classification (object identification)
- Translation invariance
  - Classification should work if object appears in different locations in the image => All image regions are treated the same
- Locality
  - Focus on local regions for object detection => computation should be local
- Mathematical operation: Convolution

Convolutional Neural Networks

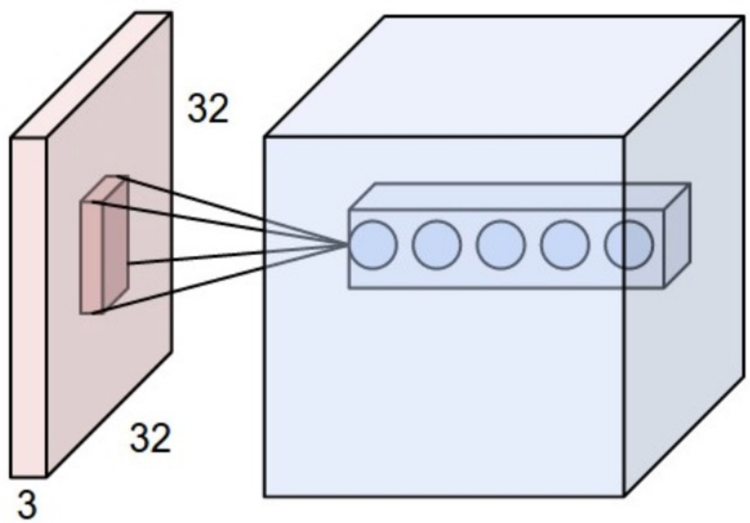
Feed-forward network



Convolutional network

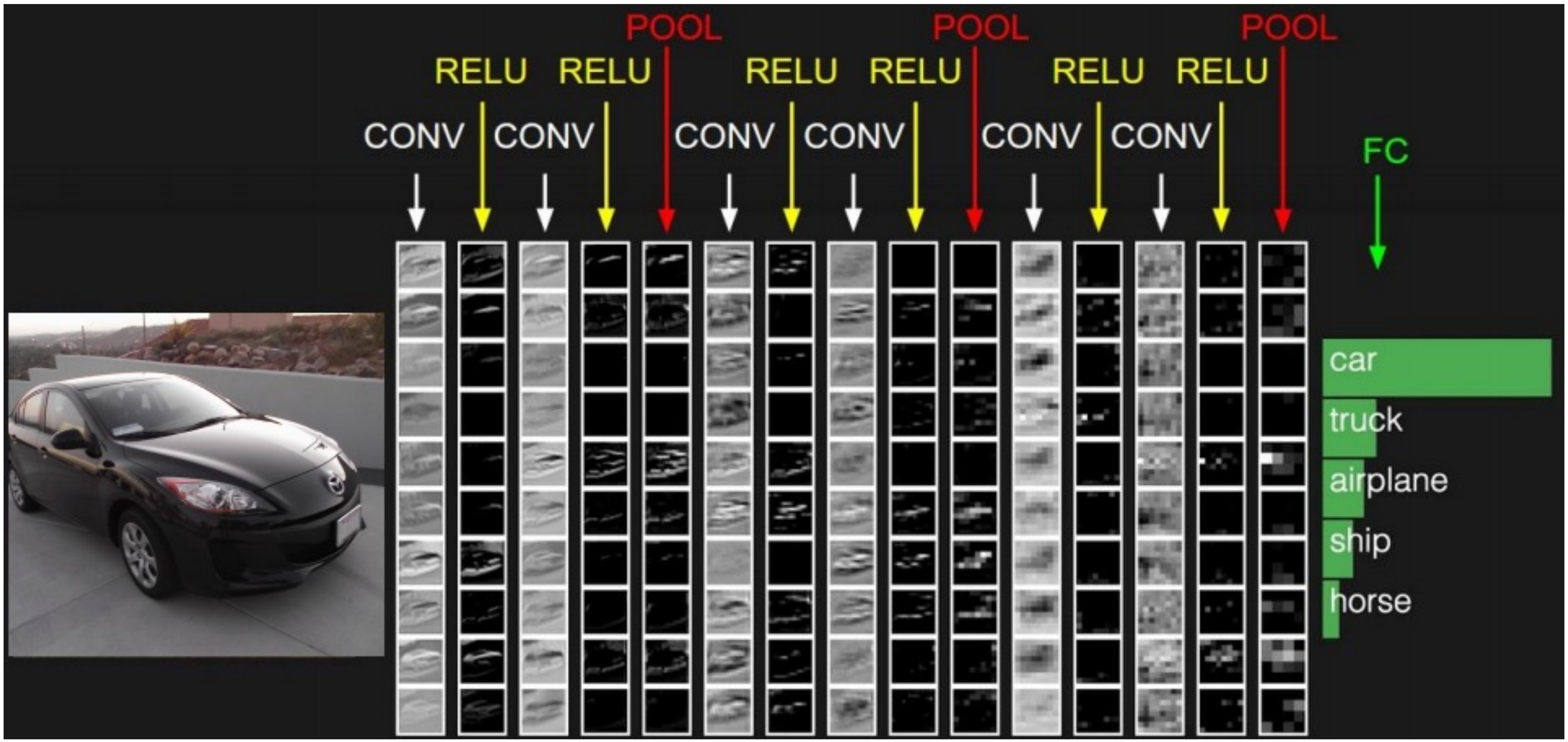


Filter



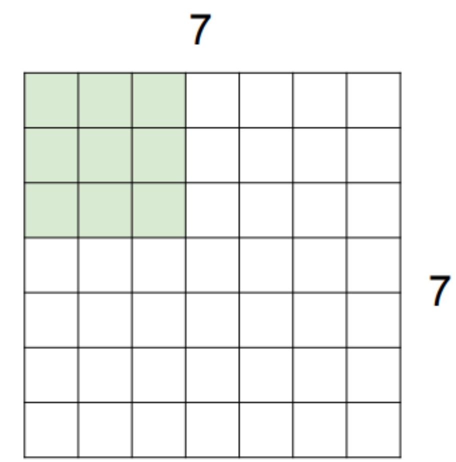
https://www.youtube.com/watch?v=f0t-OCG79-U

Convolutional Nets

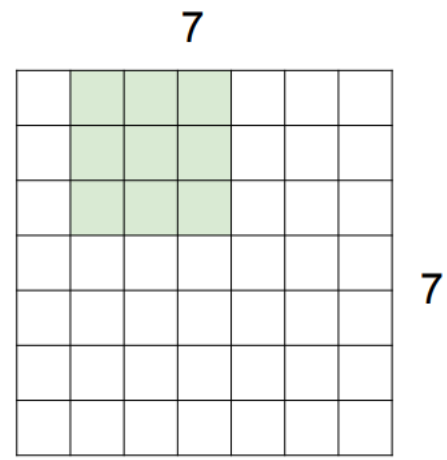


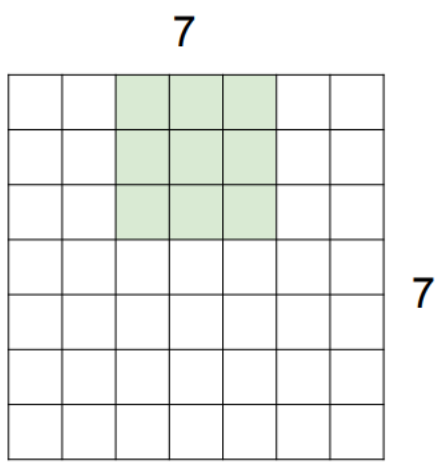
Convolutions

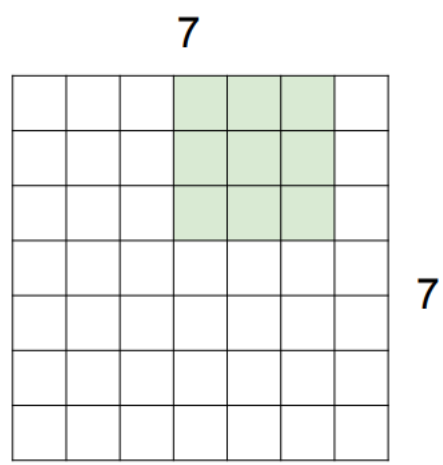
A closer look at spatial dimensions:

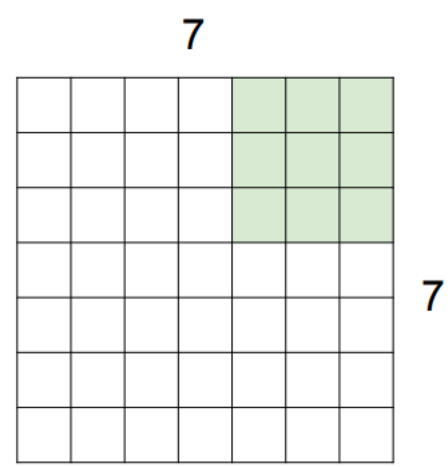


7x7 input (spatially)
assume 3x3 filter

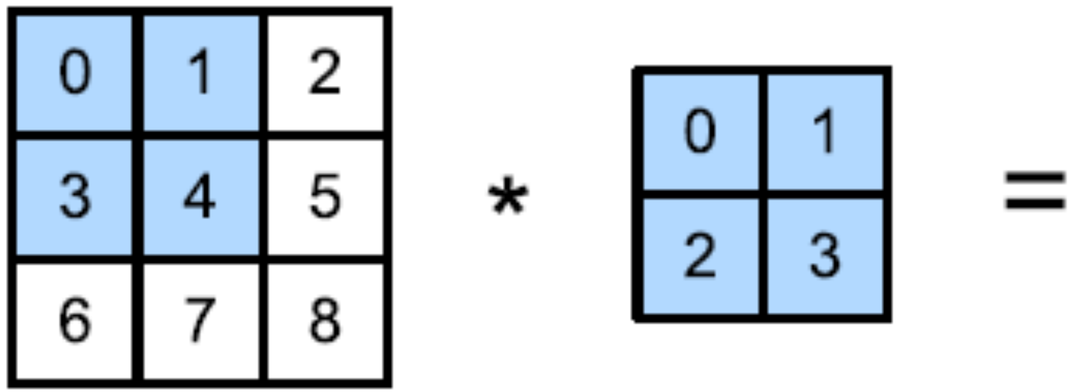






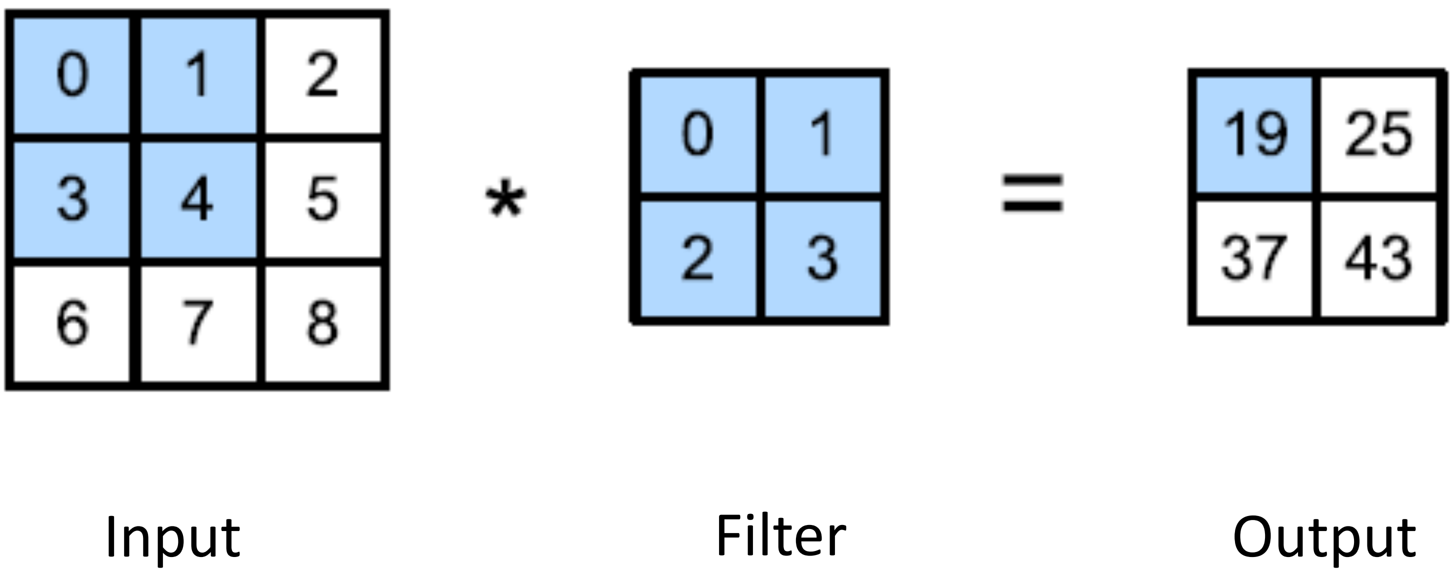


Example



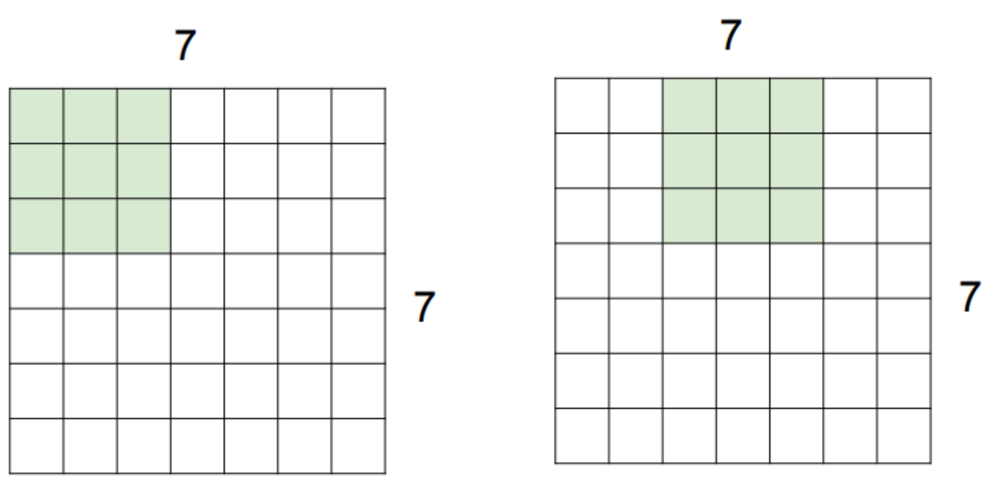
Input Filter Output

Example



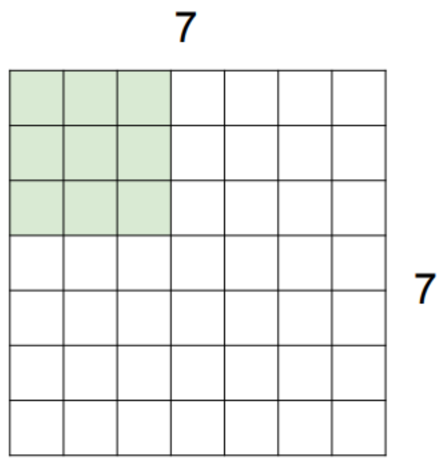
Convolutions with stride

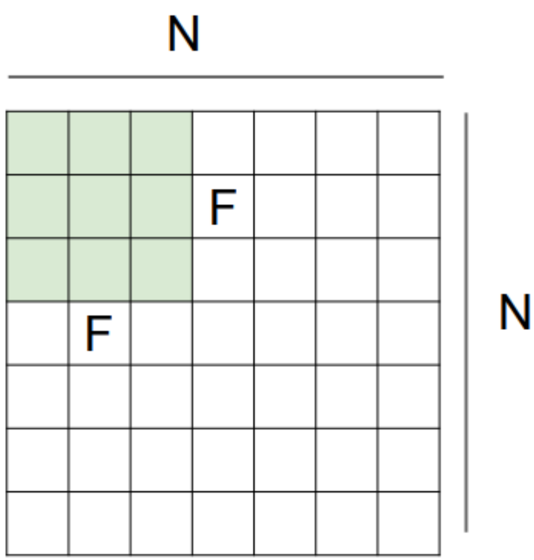
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



Convolutions with stride

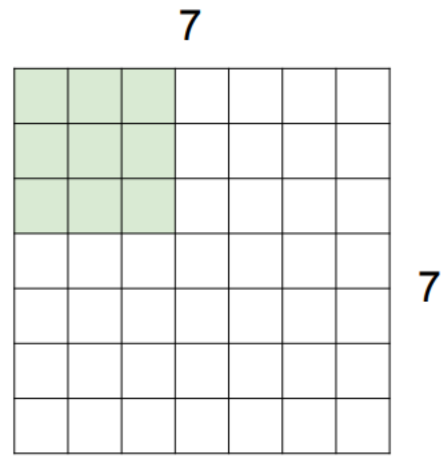
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**



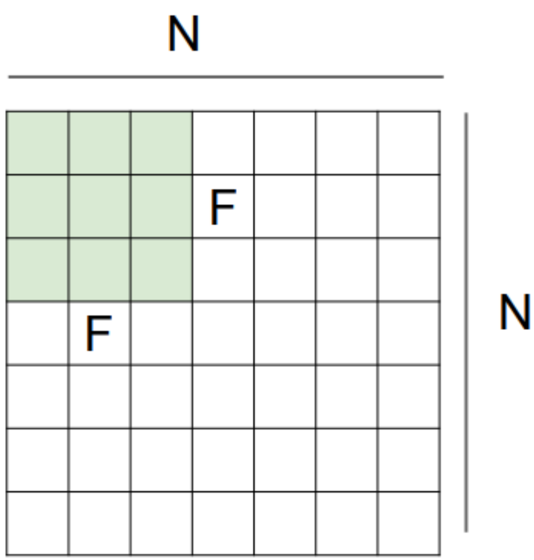


Convolutions with stride

7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**



doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

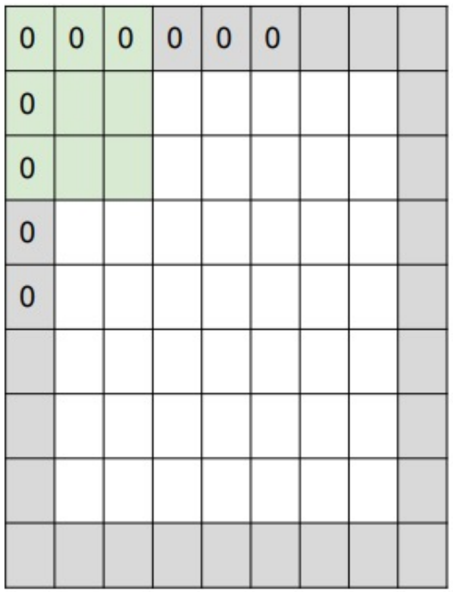


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Padding

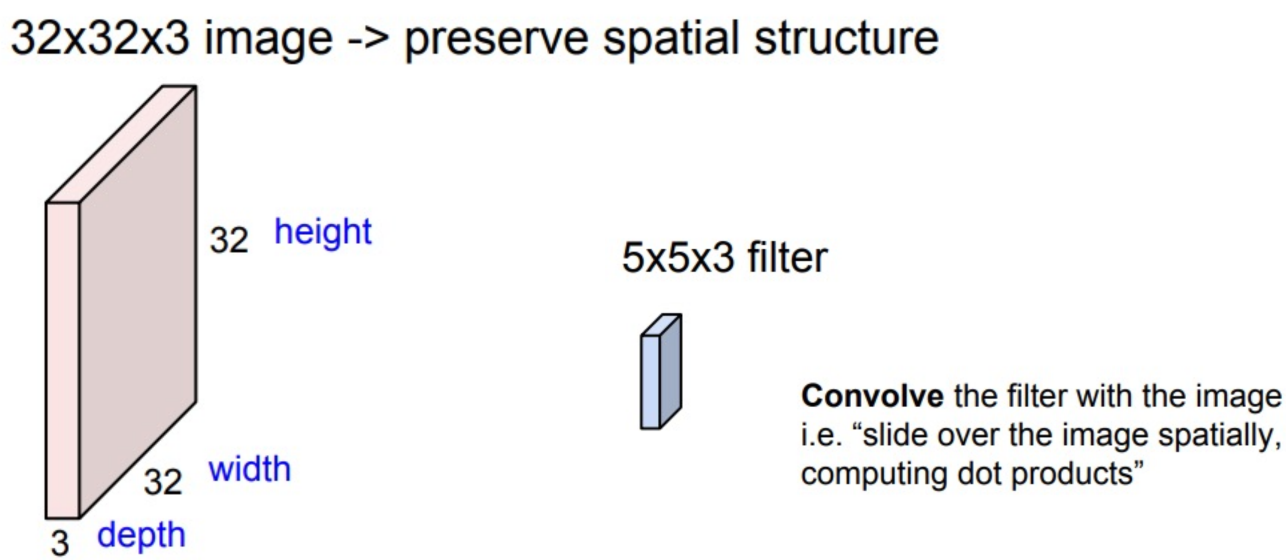
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 3**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

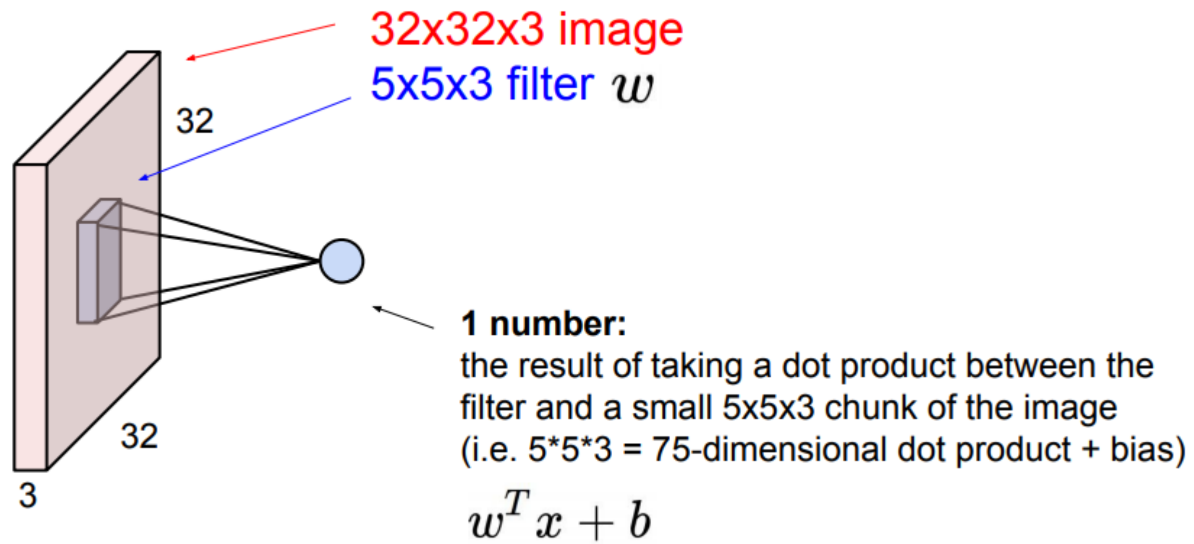
Convolution Layer

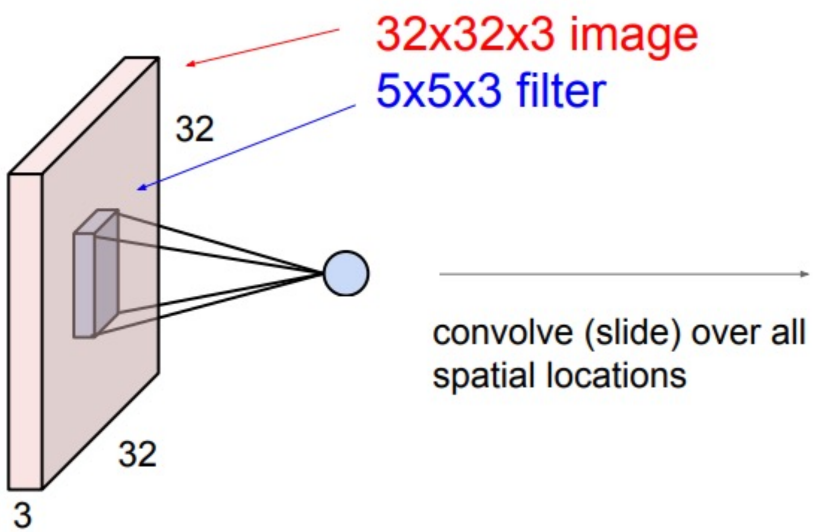


- Depth of filter always depth of input
- Computation is based only on local information

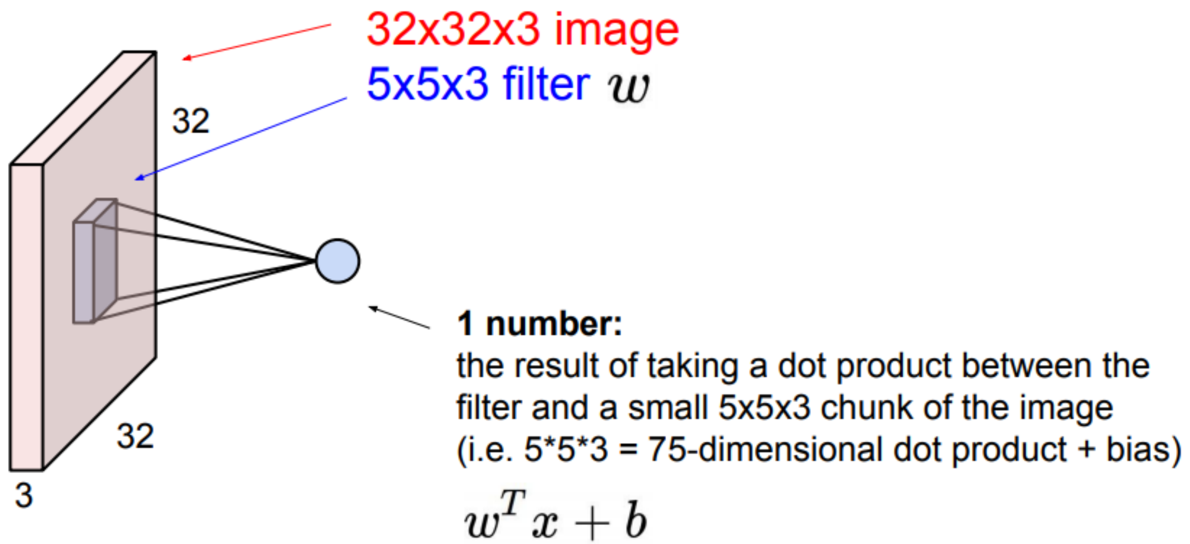
Convolution Operation

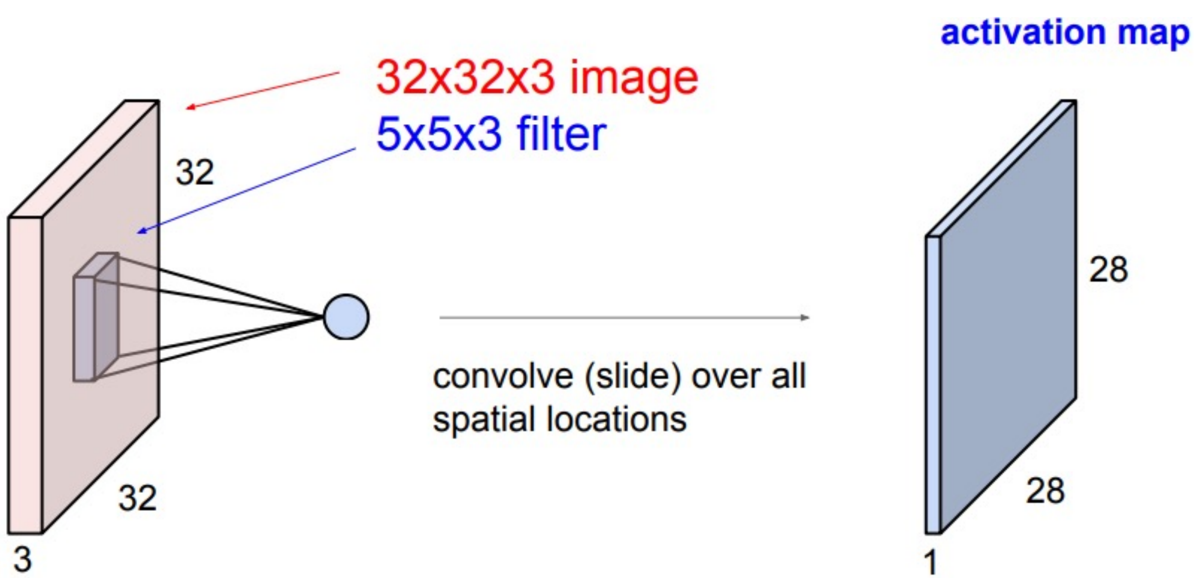
Convolution Layer



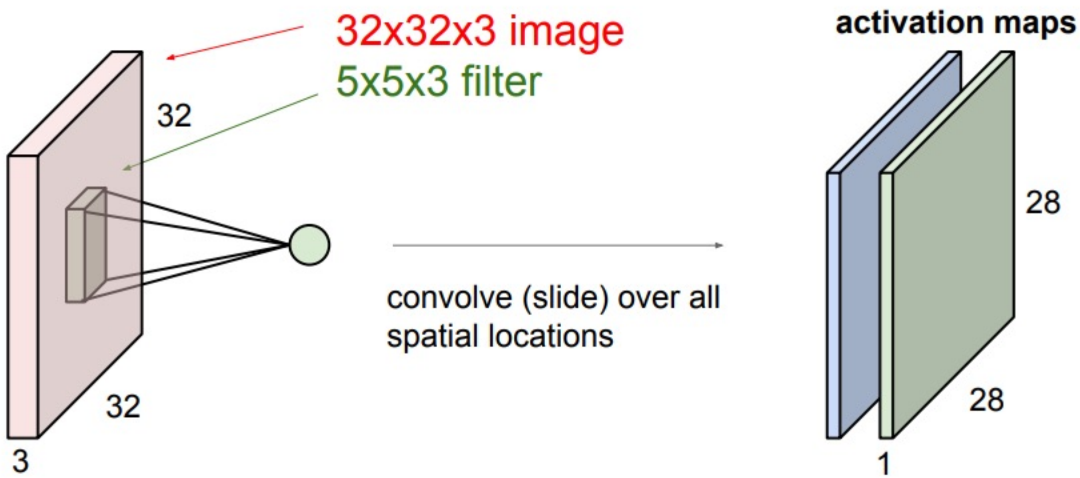


Convolution Layer

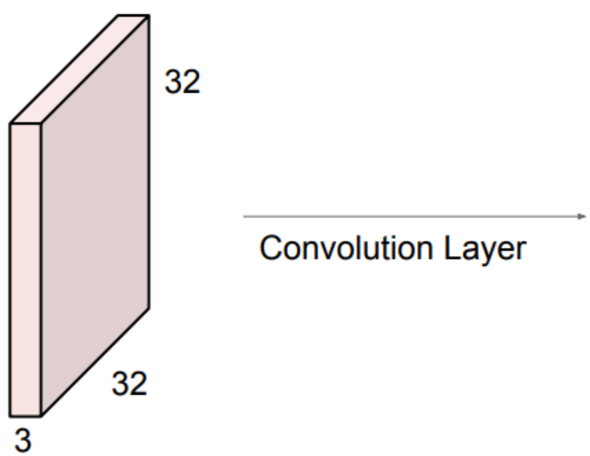




Convolution Layer

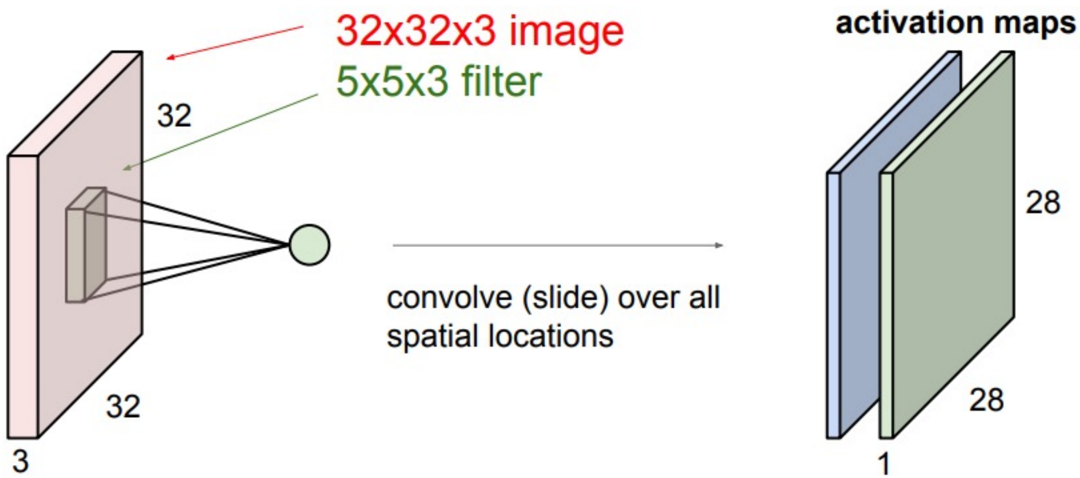


Second, green filter

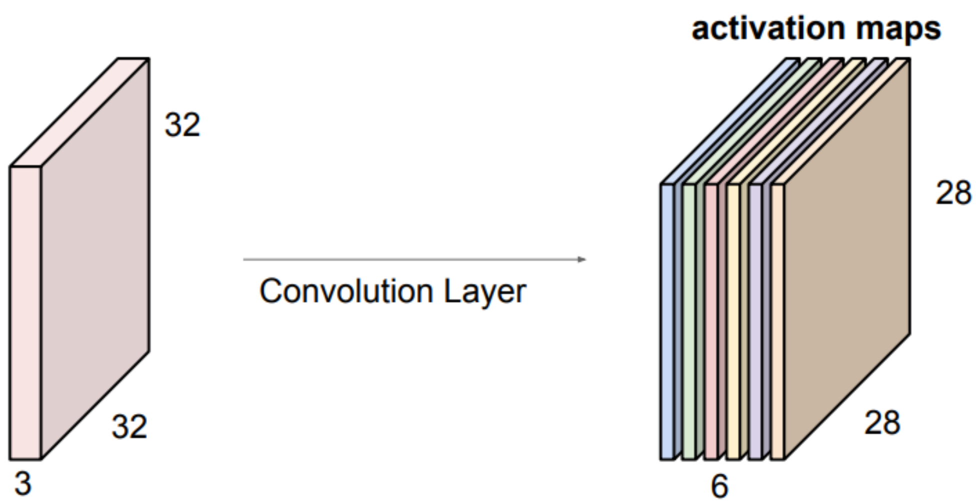


6 filters

Convolution Layer



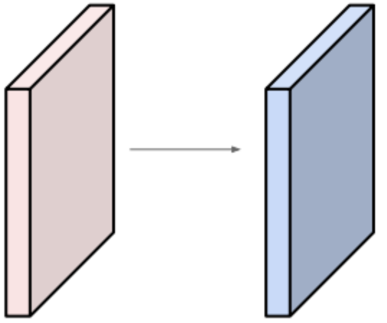
Second, green filter



6 filters

Examples

Examples time:



Input volume 36x36x3

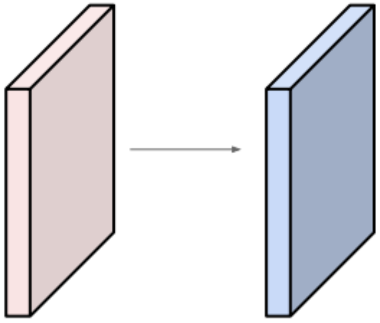
10 5x5x3 filters with stride 1

Output volume size: ?

Number of parameters in this layer?

Examples

Examples time:



Input volume: **32x32x3**

10 5x5x3 filters with stride 1, pad 2

Output volume size: ?

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

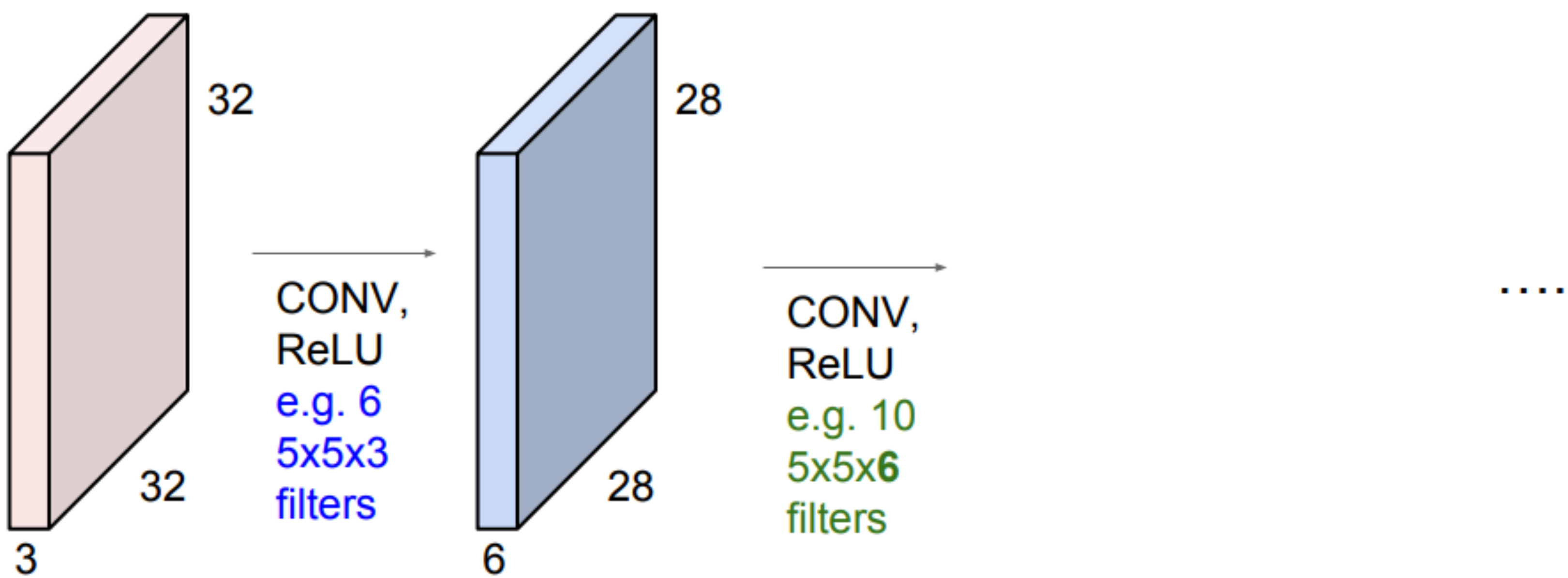
=> 76*10 = **760**

Convolutional Nets

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

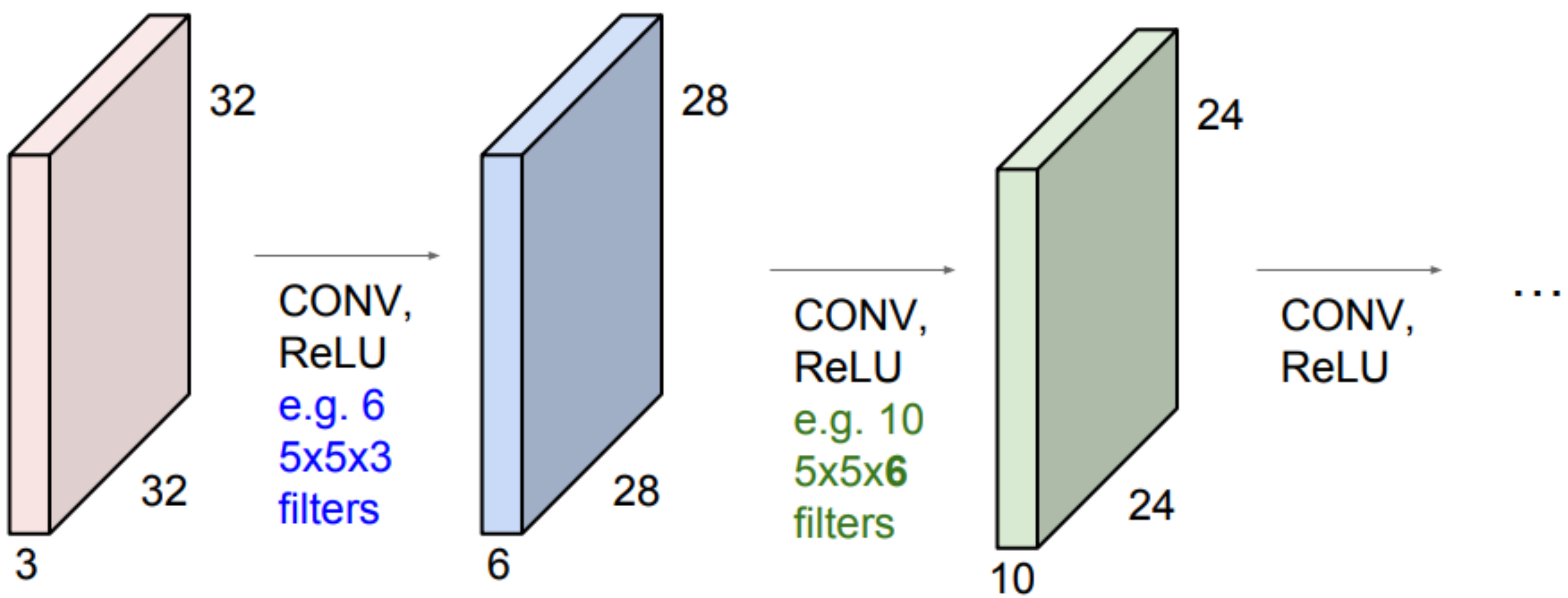
Convolutional Nets

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Convolutional Nets

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Summary: Convolution Layer

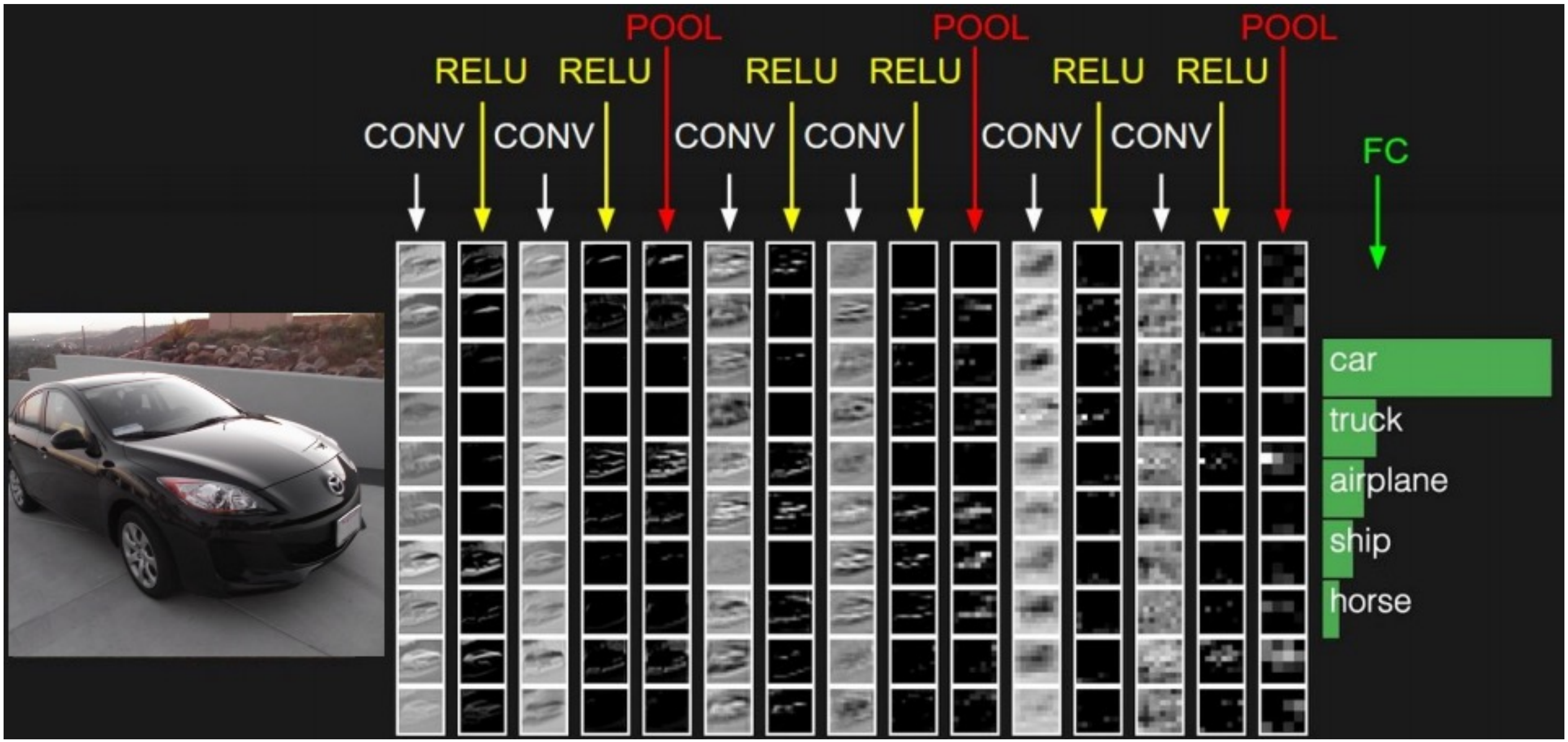
Summary. To summarize, the Conv Layer:

- 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:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Convolution layer: Takeaways

- Convolution is a linear operation
  - Reduces parameter space of Feed-Forward Neural Network considerably
  - Capture locality of pixels in images
  - Smaller filters need less parameters
  - Multiple filters in each layer (computation can be done in parallel)
- Convolutions are followed by activation functions
  - Typically ReLU

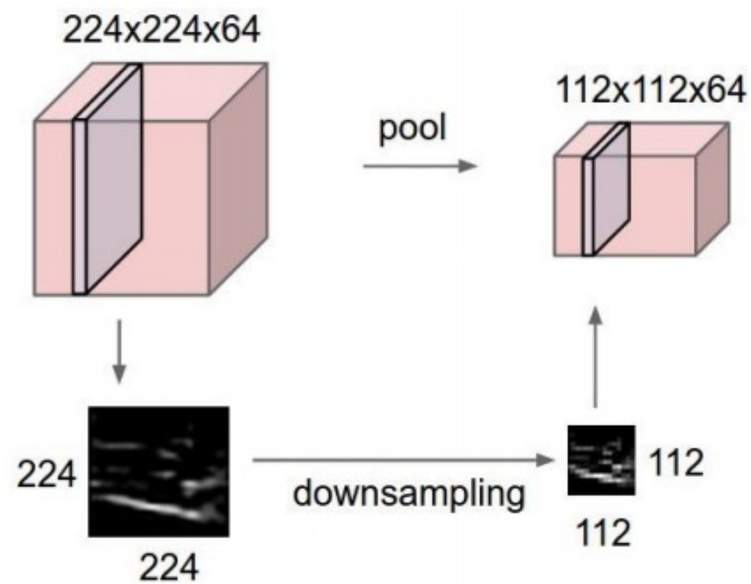
Convolutional Nets



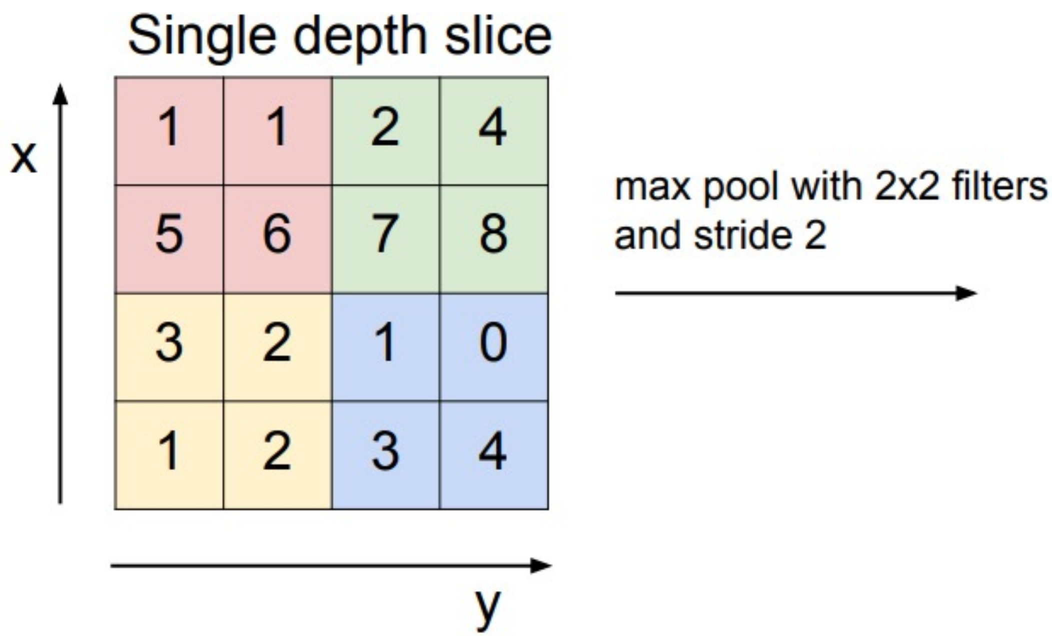
Pooling layer

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

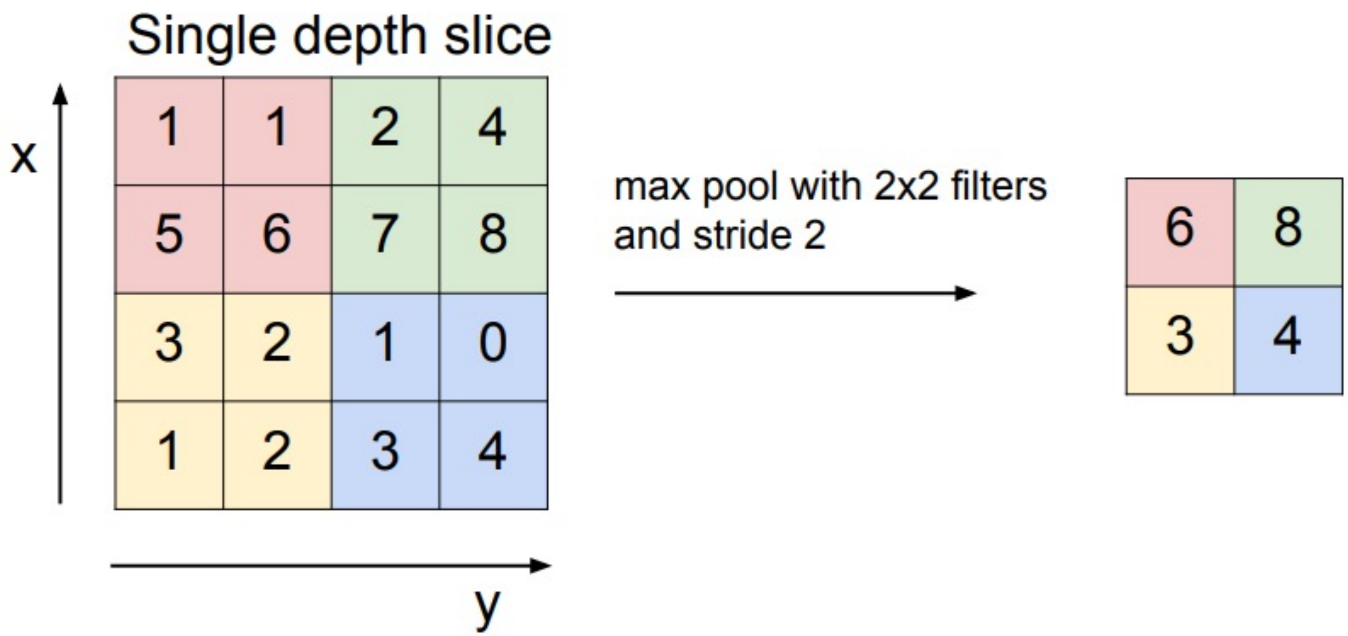


Max Pooling



Single depth slice
x
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
y
max pool with 2x2 filters
and stride 2

Max Pooling

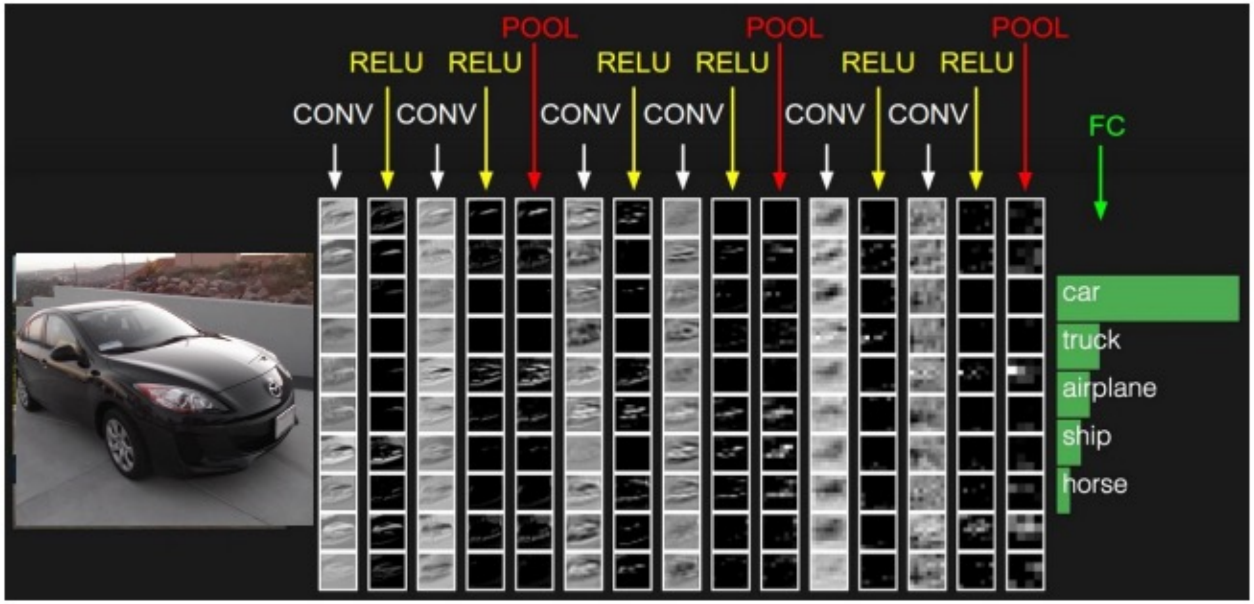


- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
    - their spatial extent F,
    - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
    - $W_2 = (W_1 - F)/S + 1$
    - $H_2 = (H_1 - F)/S + 1$
    - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Convolutional Nets

Fully Connected Layer (FC layer)

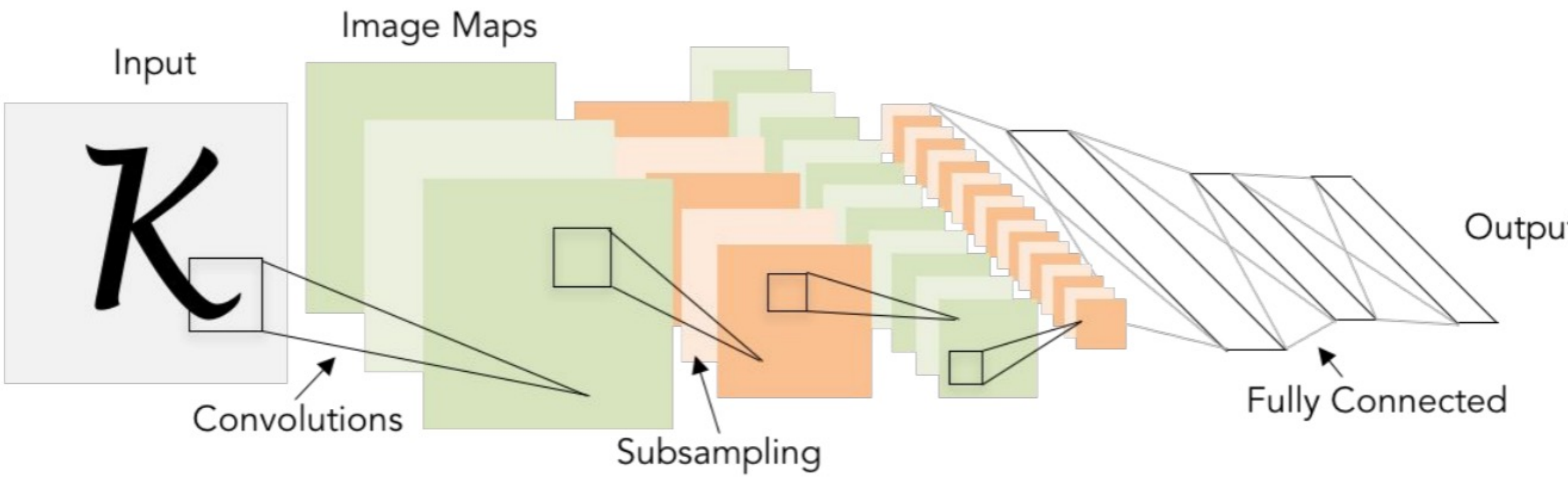
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



- FC layers are usually at the end, after several Convolutions and Pooling layers

LeNet 5

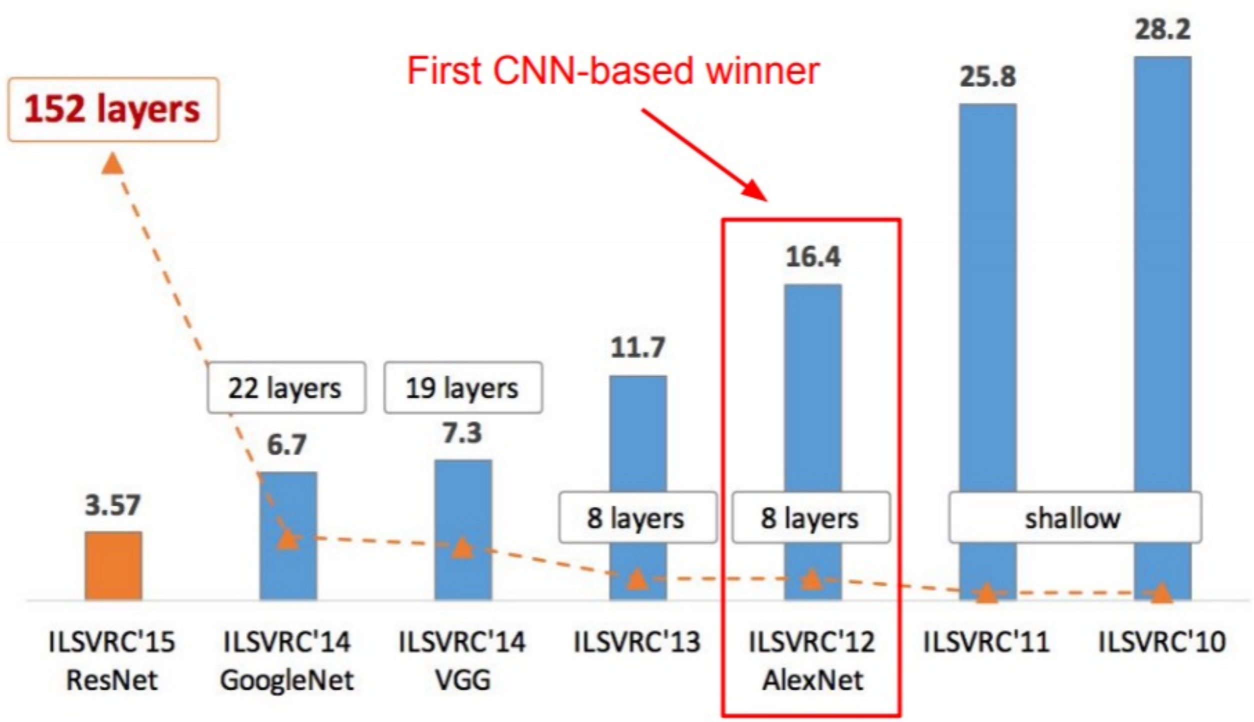
[LeCun et al., 1998]



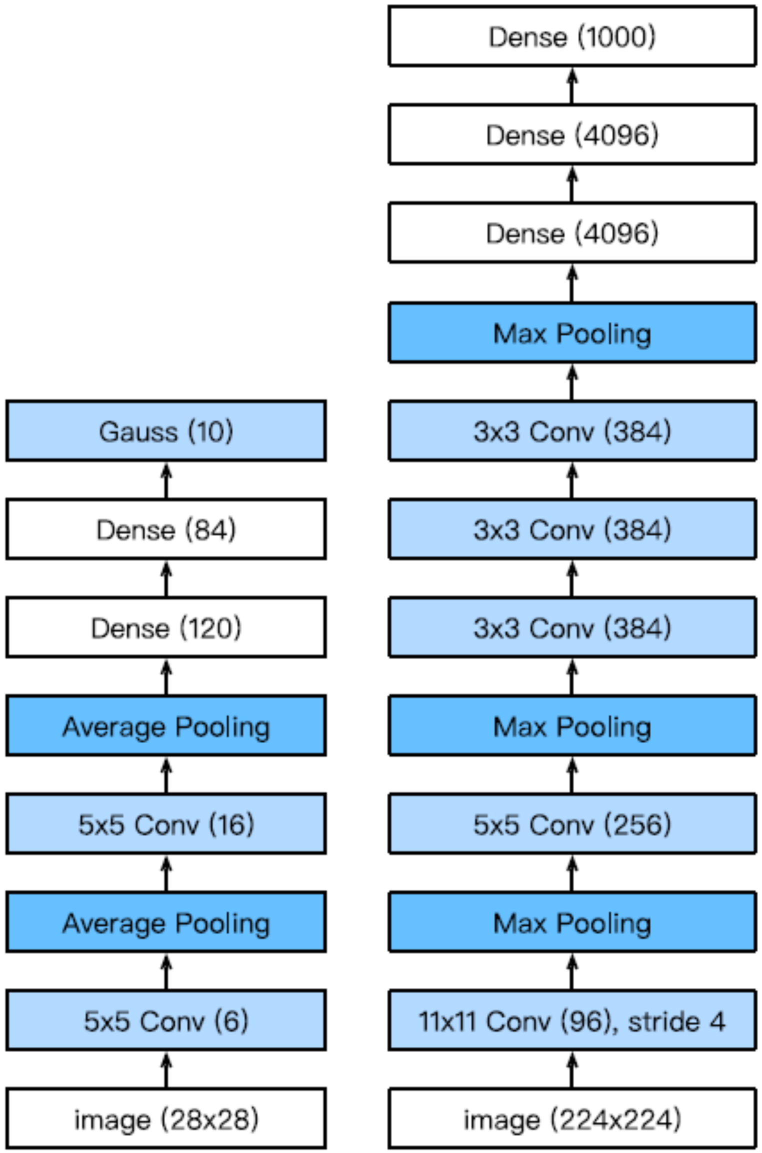
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

History

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



LeNet (left) and AlexNet (right)



Main differences

- Deeper
- Wider layers
- ReLU activation
- More classes in output layer
- Max Pooling instead of Avg Pooling

VGGNet

Case Study: VGGNet

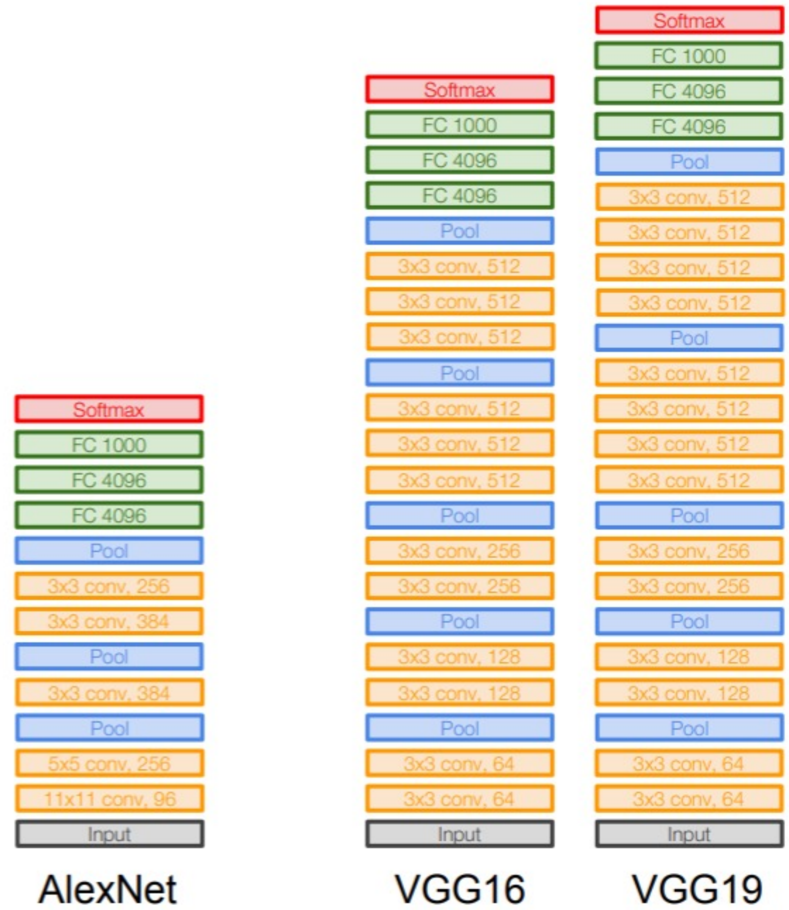
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14

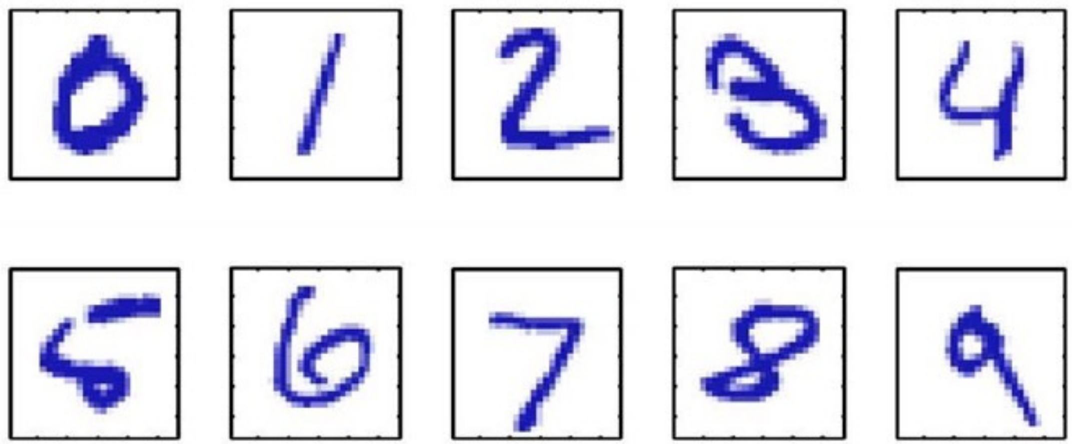


138 million parameters

Summary CNNs

- Convolutional Nets have at least one convolution layer and optionally max pooling layers
- Convolutions enable dimensionality reduction, are translation invariant and exploit locality
- Much fewer parameters relative to Feed-Forward Neural Networks
  - Deeper networks with multiple small filters at each layer is a trend
- Fully connected layer at the end (fewer parameters)
- Learn hierarchical feature representations
  - Data with natural grid topology (images, maps)
- Reached human-level performance in ImageNet in 2014

MNIST: Handwritten digit recognition



Images are 28 x 28 pixels

Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$

Learn a classifier $f(\mathbf{x})$ such that,

$$f : \mathbf{x} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Predict the digit

Multi-class classifier

Lab – Feed Forward NN

```python
import time
import numpy as np
#!pip install tensorflow
#!pip install keras

from keras.utils import np_utils
import keras.callbacks as cb
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop
from keras.datasets import mnist

import matplotlib
import matplotlib.pyplot as plt
```

```python
def load_data():
    print("Loading data")
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')

    # Normalize
    X_train /= 255
    X_test /= 255

    y_train = np_utils.to_categorical(y_train, 10)
    y_test = np_utils.to_categorical(y_test, 10)

    X_train = np.reshape(X_train, (60000, 784))
    X_test = np.reshape(X_test, (10000, 784))

    print("Data loaded")
    return [X_train, X_test, y_train, y_test]
```

Lab – Feed Forward NN

```
import time
import numpy as np
#!pip install tensorflow
#!pip install keras

from keras.utils import np_utils
import keras.callbacks as cb
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop
from keras.datasets import mnist

import matplotlib
import matplotlib.pyplot as plt
```

Import modules

```
def load_data():
    print("Loading data")
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')

    # Normalize
    X_train /= 255
    X_test /= 255

    y_train = np_utils.to_categorical(y_train, 10)
    y_test = np_utils.to_categorical(y_test, 10)

    X_train = np.reshape(X_train, (60000, 784))
    X_test = np.reshape(X_test, (10000, 784))

    print("Data loaded")
    return [X_train, X_test, y_train, y_test]
```

Load MNIST data Processing

Vector representation

Neural Network Architecture

```python
def init_model1():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(10, input_dim=784))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished "+format(time.time() - start_time))
    return model
```

Feed-Forward Neural Network Architecture

- 1 Hidden Layer ("Dense" or Fully Connected)
- 10 neurons
- Output layer uses softmax activation

Neural Network Architecture

```python
def init_model():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(10, input_dim=784))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished"+format(time.time() - start_time))
    return model
```

10 hidden units
ReLU activation

Output Layer
Softmax activation

Loss function

Optimizer

Feed-Forward Neural Network Architecture

- 1 Hidden Layer ("Dense" or Fully Connected)
- 10 neurons
- Output layer uses softmax activation

Number of Parameters

Number of Parameters

```
model1.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 10)                7850
_________________________________________________________________
activation_16 (Activation)   (None, 10)                0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                110
_________________________________________________________________
activation_17 (Activation)   (None, 10)                0
=================================================================
Total params: 7,960
Trainable params: 7,960
Non-trainable params: 0
_________________________________________________________________
```

Train and evaluate

```
def run_network(data=None, model=None, epochs=20, batch=256):
    try:
        start_time = time.time()
        if data is None:
            X_train, X_test, y_train, y_test = load_data()
        else:
            X_train, X_test, y_train, y_test = data

        print("Training model")
        history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch,
                  validation_data=(X_test, y_test), verbose=2)

        print("Training duration:"+format(time.time() - start_time))
        score = model.evaluate(X_test, y_test, batch_size=16)

        print("\nNetwork's test loss and accuracy:"+format(score))
        return history
    except KeyboardInterrupt:
        print("KeyboardInterrupt")
        return history
```

Training/testing results

```
Compiling Model
Model finished 0.040144205093383 79
Loading data
Data loaded
Training model
Epoch 1/10
235/235 - 1s - loss: 0.9142 - accuracy: 0.7501 - val_loss: 0.4398 - val_accuracy: 0.8833
Epoch 2/10
235/235 - 0s - loss: 0.3856 - accuracy: 0.8959 - val_loss: 0.3392 - val_accuracy: 0.9050
Epoch 3/10
235/235 - 0s - loss: 0.3245 - accuracy: 0.9093 - val_loss: 0.3043 - val_accuracy: 0.9141
Epoch 4/10
235/235 - 0s - loss: 0.2992 - accuracy: 0.9165 - val_loss: 0.2890 - val_accuracy: 0.9178
Epoch 5/10
235/235 - 0s - loss: 0.2853 - accuracy: 0.9202 - val_loss: 0.2797 - val_accuracy: 0.9214
Epoch 6/10
235/235 - 0s - loss: 0.2755 - accuracy: 0.9234 - val_loss: 0.2735 - val_accuracy: 0.9217
Epoch 7/10
235/235 - 0s - loss: 0.2690 - accuracy: 0.9251 - val_loss: 0.2689 - val_accuracy: 0.9252
Epoch 8/10
235/235 - 0s - loss: 0.2634 - accuracy: 0.9263 - val_loss: 0.2658 - val_accuracy: 0.9271
Epoch 9/10
235/235 - 0s - loss: 0.2590 - accuracy: 0.9276 - val_loss: 0.2666 - val_accuracy: 0.9257
Epoch 10/10
235/235 - 0s - loss: 0.2554 - accuracy: 0.9284 - val_loss: 0.2616 - val_accuracy: 0.9284
Training duration:3.1347107887268066
625/625 [==============================] - 1s 707us/step - loss: 0.2616 - accuracy: 0.9284

Network's test loss and accuracy:[0.2615792751312256, 0.9283999800682068]
```

Training/testing results

```
Epoch 98/100
235/235 - 0s - loss: 0.1611 - accuracy: 0.9552 - val_loss: 0.2329 - val_accuracy: 0.9411
Epoch 99/100
235/235 - 0s - loss: 0.1609 - accuracy: 0.9550 - val_loss: 0.2334 - val_accuracy: 0.9392
Epoch 100/100
235/235 - 0s - loss: 0.1603 - accuracy: 0.9550 - val_loss: 0.2323 - val_accuracy: 0.9401
Training duration:22.163272857666016
625/625 [==============================] - 1s 711us/step - loss: 0.2323 - accuracy: 0.9401

Network's test loss and accuracy:[0.23233847320079803, 0.9401000142097473]
```

Changing Number of Neurons

```python
def init_model2():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(500, input_dim=784))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished "+format(time.time() - start_time))
    return model
```

```
Epoch 98/100
235/235 - 1s - loss: 1.1261e-08 - accuracy: 1.0000 - val_loss: 0.1432 - val_accuracy: 0.9835
Epoch 99/100
235/235 - 1s - loss: 1.1088e-08 - accuracy: 1.0000 - val_loss: 0.1432 - val_accuracy: 0.9834
Epoch 100/100
235/235 - 1s - loss: 1.0918e-08 - accuracy: 1.0000 - val_loss: 0.1435 - val_accuracy: 0.9835
Training duration:82.20909094810486
625/625 [==============================] - 1s 932us/step - loss: 0.1435 - accuracy: 0.9835

Network's test loss and accuracy:[0.1434623748064041, 0.9835000038146973]
```

Number of Parameters

Number of Parameters

```
model2.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 500)               392500
_________________________________________________________________
activation_22 (Activation)   (None, 500)               0
_________________________________________________________________
dense_23 (Dense)             (None, 10)                5010
_________________________________________________________________
activation_23 (Activation)   (None, 10)                0
=================================================================
Total params: 397,510
Trainable params: 397,510
Non-trainable params: 0
_________________________________________________________________
```

Two Layers

```python
def init_model4():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(500, input_dim=784))
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    model.add(Dense(300))
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished"+format(time.time() - start_time))
    return model
```

```
Epoch 98/100
235/235 - 1s - loss: 0.0164 - accuracy: 0.9961 - val_loss: 0.1677 - val_accuracy: 0.9844
Epoch 99/100
235/235 - 1s - loss: 0.0146 - accuracy: 0.9966 - val_loss: 0.1701 - val_accuracy: 0.9841
Epoch 100/100
235/235 - 1s - loss: 0.0134 - accuracy: 0.9968 - val_loss: 0.1666 - val_accuracy: 0.9849
Training duration:131.49207520484924
625/625 [==============================] - 1s 1ms/step - loss: 0.1666 - accuracy: 0.9849

Network's test loss and accuracy:[0.16656503081321716, 0.9848999977111816]
```

Number of Parameters

Model Parameters

```
model4.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 500)               392500
_________________________________________________________________
activation_26 (Activation)   (None, 500)               0
_________________________________________________________________
dropout_9 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_27 (Dense)             (None, 300)               150300
_________________________________________________________________
activation_27 (Activation)   (None, 300)               0
_________________________________________________________________
dropout_10 (Dropout)         (None, 300)               0
_________________________________________________________________
dense_28 (Dense)             (None, 10)                3010
_________________________________________________________________
activation_28 (Activation)   (None, 10)                0
=================================================================
Total params: 545,810
Trainable params: 545,810
Non-trainable params: 0
_________________________________________________________________
```

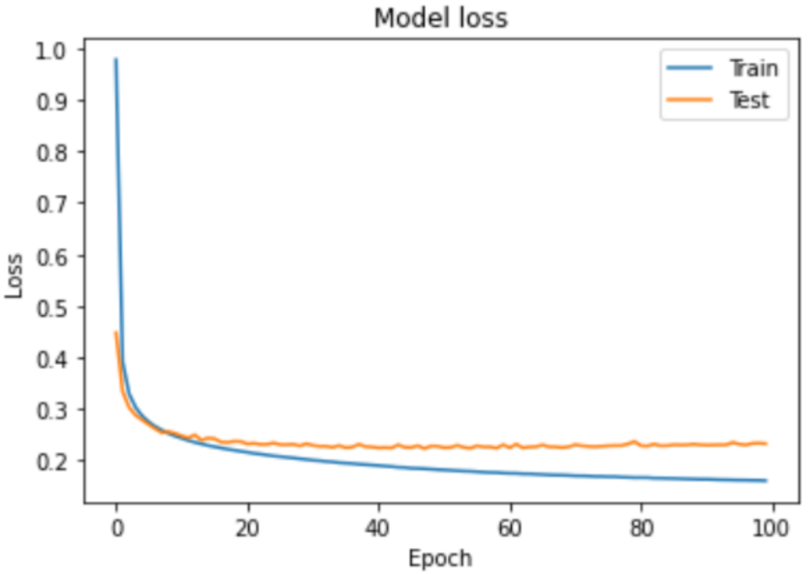
Monitor Loss

```
def plot_losses(hist):
    plt.plot(hist.history['loss'])
    plt.plot(hist.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper right')
    plt.show()
```

```
model1 = init_model1()

history1 = run_network(model = model1, epochs=100)

plot_losses(history1)
```



Loss

```
model2 = init_model2()

history2 = run_network(model = model2, epochs=100)

plot_losses(history2)
```

```
model4 = init_model4()

history4 = run_network(model = model4, epochs=100)

plot_losses(history4)
```