DS 4400

Machine Learning and Data Mining I

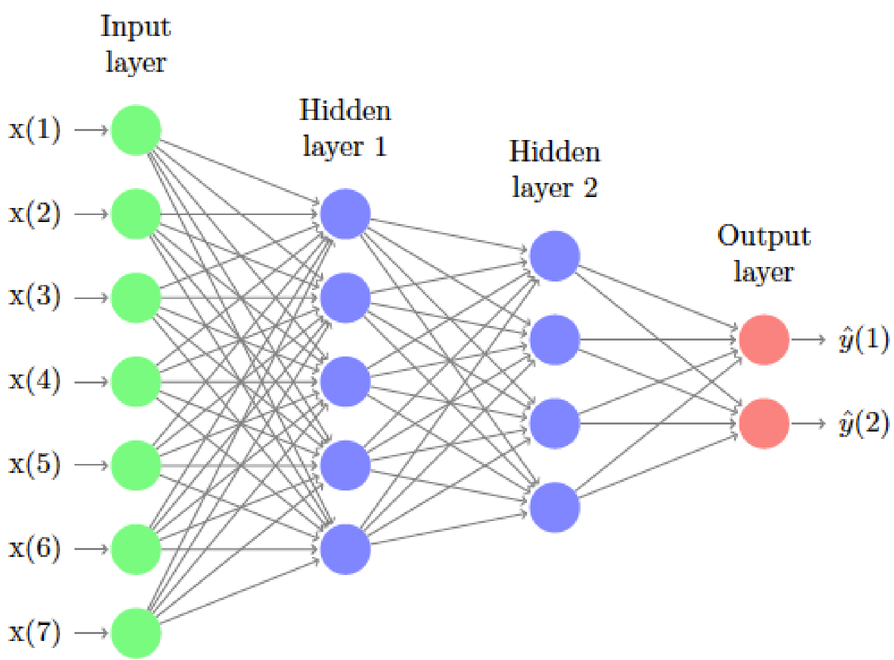
Alina Oprea
Associate Professor, CCIS
Northeastern University

March 21 2019

Review Feed-Forward Neural Networks

- Simplest architecture of NN
- Neurons from one layer are connected to neurons from next layer
  - Input layer has feature space dimension
  - Output layer has number of classes
  - Hidden layers use linear operations, followed by non-linear activation function
  - Multi-Layer Perceptron (MLP): fully connected layers
- Activation functions
  - Sigmoid for binary classification in last layer
  - Softmax for multi-class classification in last layer
  - ReLU for hidden layers
- Forward propagation is the computation of the network output given an input
- Back propagation is the training of a network
  - Determine weights and biases at every 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 always smaller than number of neurons in Layer i

Two Layers

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

    print("Compiling Model")
    model = Sequential()

    # Hidden Layer 1
    model.add(Dense(500, input_dim=784))
    model.add(Activation('relu'))

    # Hidden Layer 2
    model.add(Dense(300))
    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
```

Layer 1

Layer 2

Output Softmax Layer

Model Parameters

```
model.summary()
```

```
Using TensorFlow backend.
Loading data
Data loaded
Compiling Model

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 500)               392500
_________________________________________________________________
activation_1 (Activation)    (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 300)               150300
_________________________________________________________________
activation_2 (Activation)    (None, 300)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                3010
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 545,810
Trainable params: 545,810
Non-trainable params: 0
```

Results

```
2s - loss: 0.2800 - acc: 0.9132 - val_loss: 0.1821 - val_acc: 0.9409
Epoch 2/10
1s - loss: 0.0974 - acc: 0.9699 - val_loss: 0.0951 - val_acc: 0.9703
Epoch 3/10
0s - loss: 0.0616 - acc: 0.9803 - val_loss: 0.0843 - val_acc: 0.9754
Epoch 4/10
0s - loss: 0.0429 - acc: 0.9862 - val_loss: 0.0670 - val_acc: 0.9809
Epoch 5/10
0s - loss: 0.0303 - acc: 0.9904 - val_loss: 0.0820 - val_acc: 0.9777
Epoch 6/10
0s - loss: 0.0233 - acc: 0.9922 - val_loss: 0.0794 - val_acc: 0.9783
Epoch 7/10
0s - loss: 0.0180 - acc: 0.9941 - val_loss: 0.0866 - val_acc: 0.9792
Epoch 8/10
0s - loss: 0.0137 - acc: 0.9956 - val_loss: 0.0788 - val_acc: 0.9814
Epoch 9/10
0s - loss: 0.0116 - acc: 0.9963 - val_loss: 0.0901 - val_acc: 0.9795
Epoch 10/10
1s - loss: 0.0100 - acc: 0.9966 - val_loss: 0.0812 - val_acc: 0.9827
Training duration:11.816290140151978
 9744/10000 [============================>.] - ETA: 0s
```

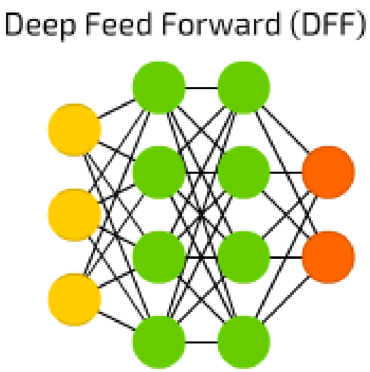
Outline

- Convolutional Neural Networks
  - Convolution layer
  - Max pooling layer
  - Strides, padding
  - Parameter counting
- Examples of famous architectures
  - LeNet 5, AlexNet, VGG16
- Lab

Neural Network Architectures

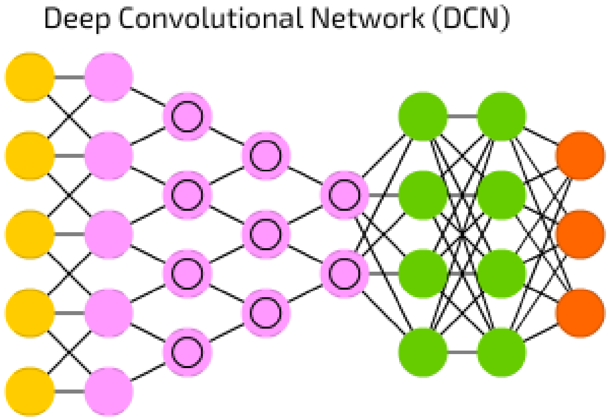
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



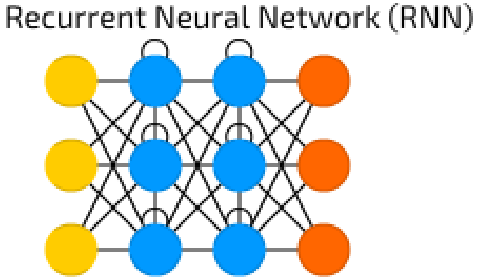
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations



Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



Convolutional Neural Networks

First strong results

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

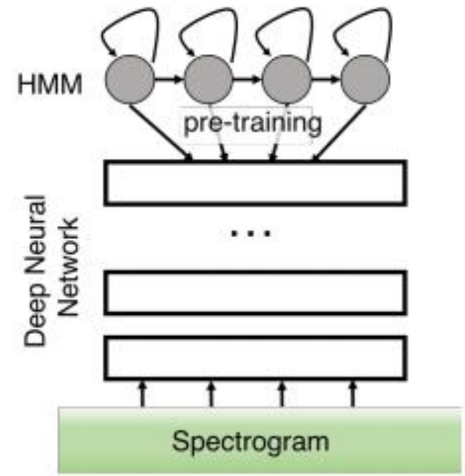
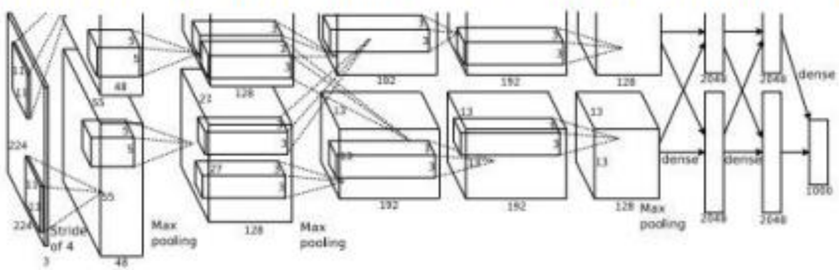
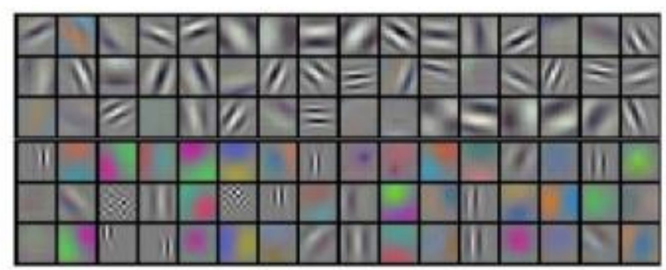


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

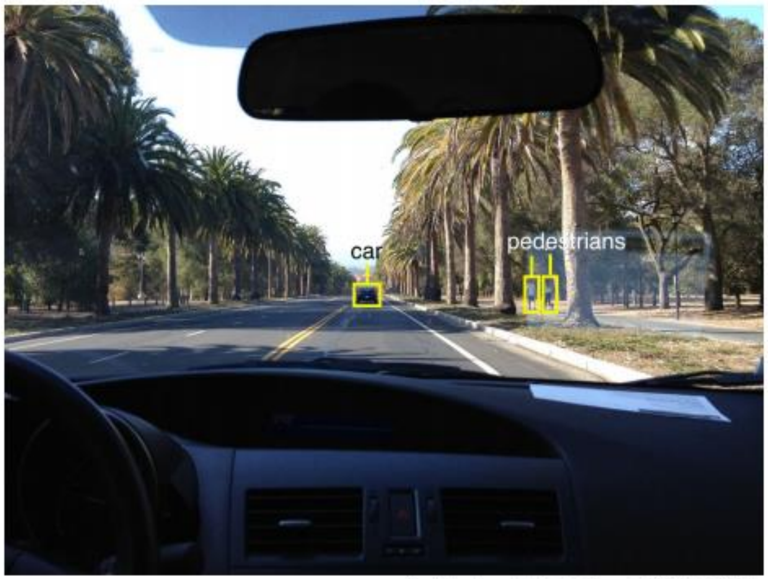
Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Convolutional Nets



self-driving cars

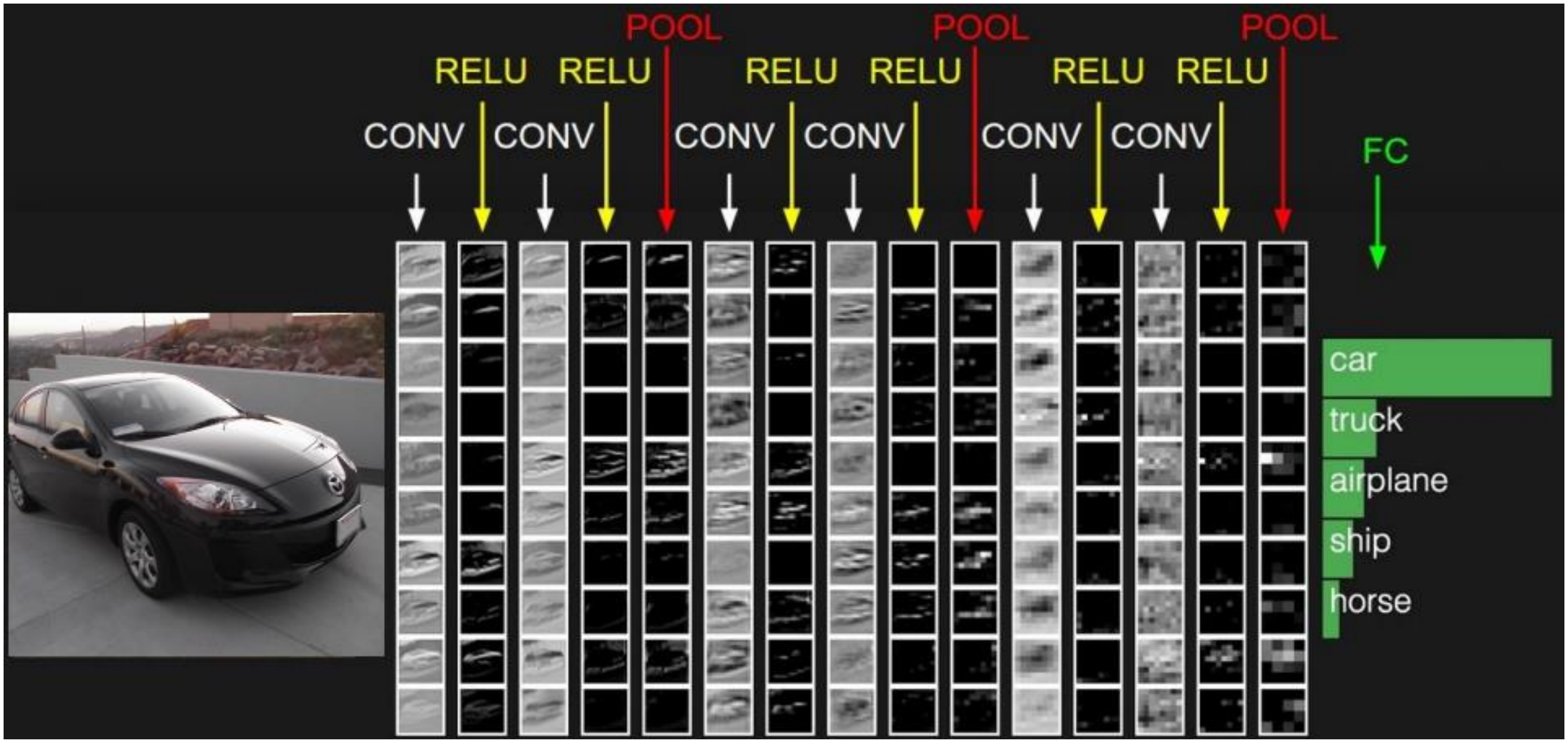
- Object recognition
- Steering angle prediction
- Assist drivers in making decisions

- Image captioning

Convolutional Nets

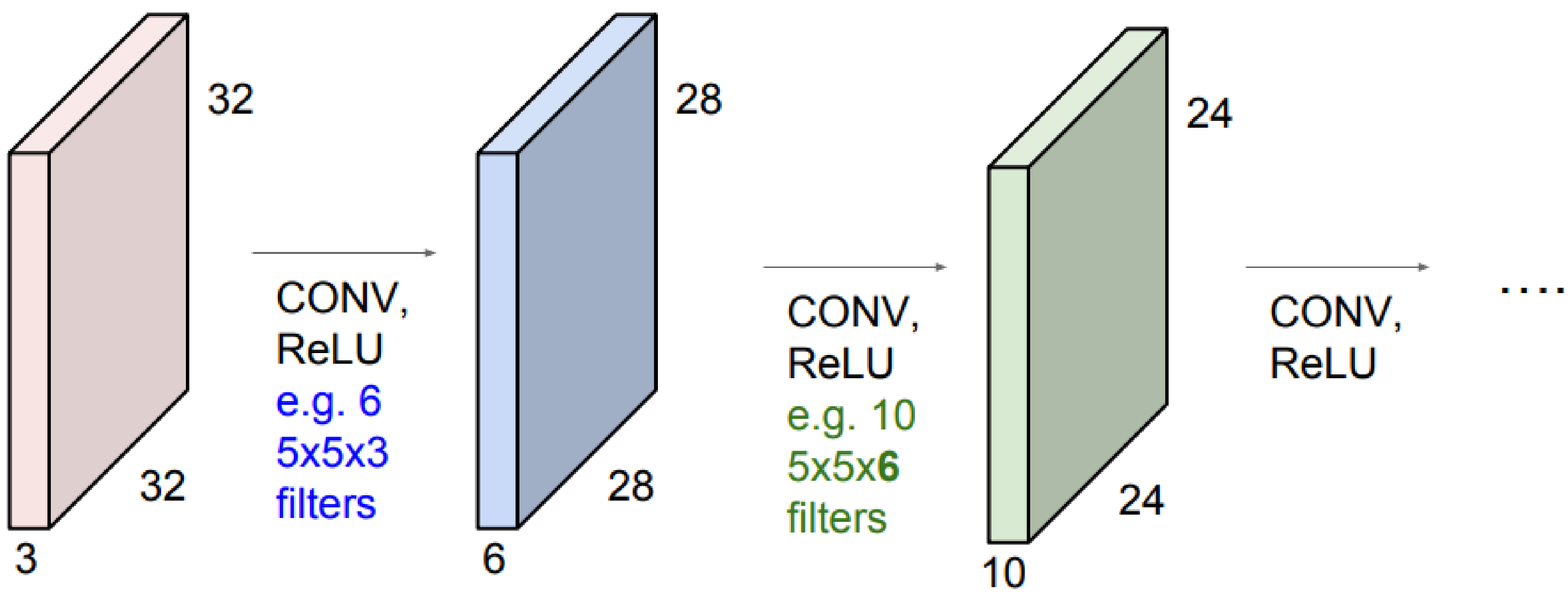
- Particular type of Feed-Forward Neural Nets
  - Invented by [LeCun 89]
- Applicable to data with natural grid topology
  - Time series
  - Images
- Use convolutions on at least one layer
  - Convolution is a linear operation that uses local information
  - Also use pooling operation
  - Used for dimensionality reduction and learning hierarchical feature representations

Convolutional Nets



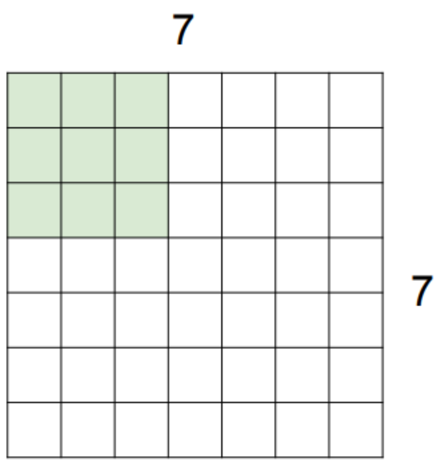
Convolutional Nets

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

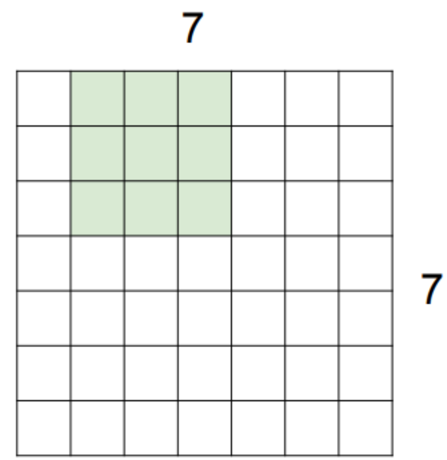


Convolutions

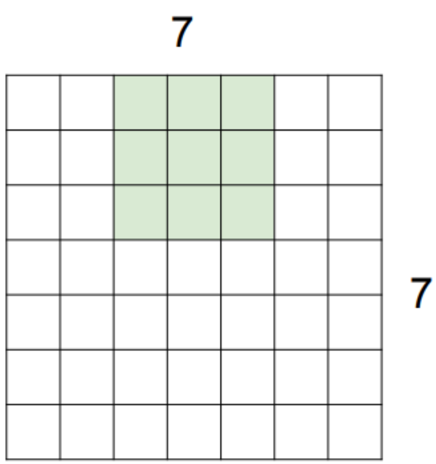
A closer look at spatial dimensions:

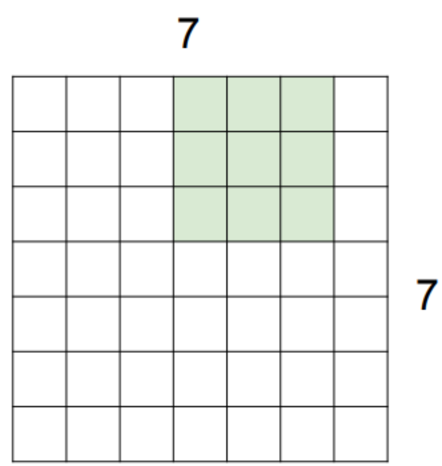


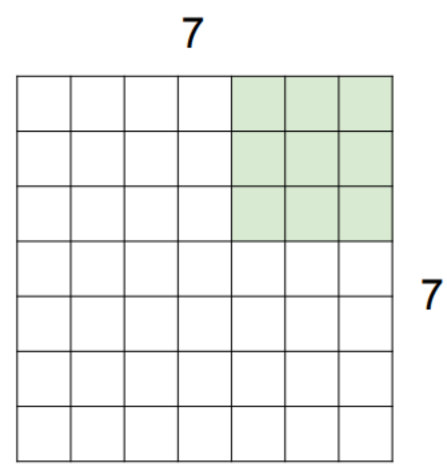
7x7 input (spatially)
assume 3x3 filter



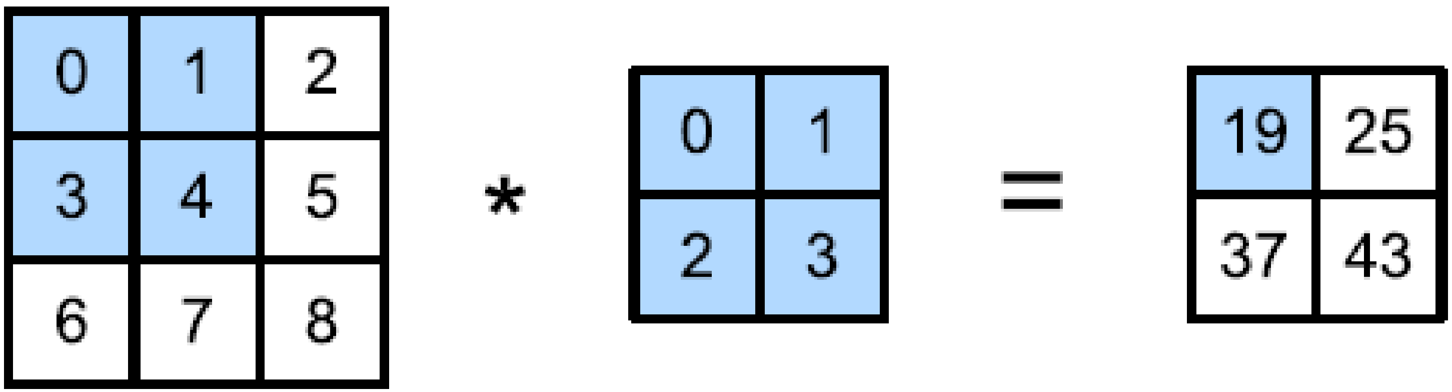
=> 5x5 output







Example



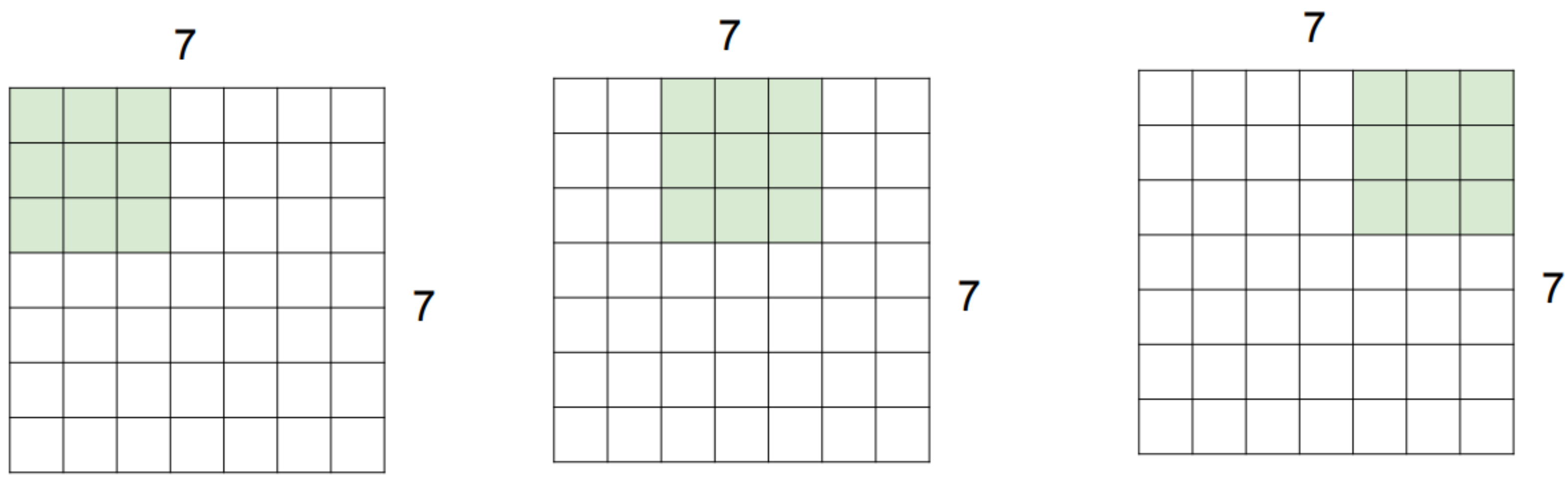
Input

Filter

Output

Convolutions with stride

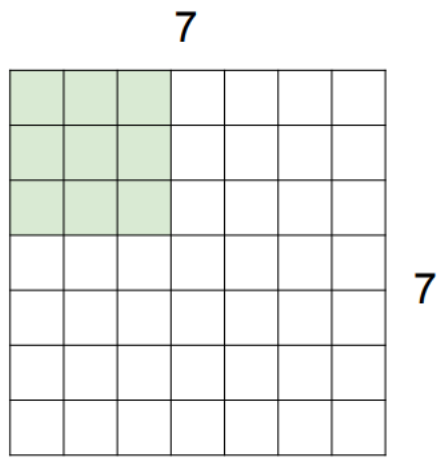
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



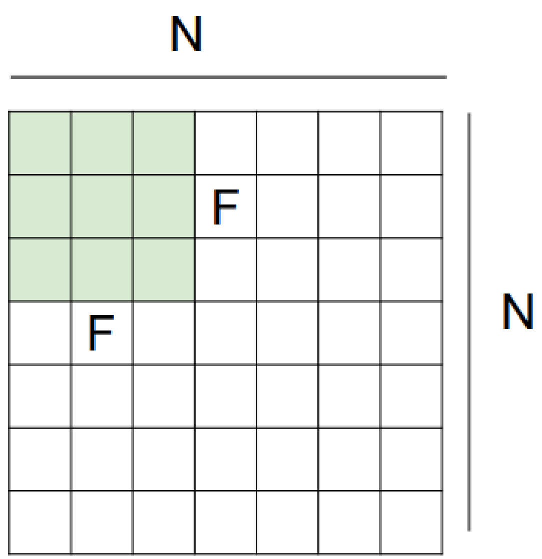
=> 3x3 output!

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.

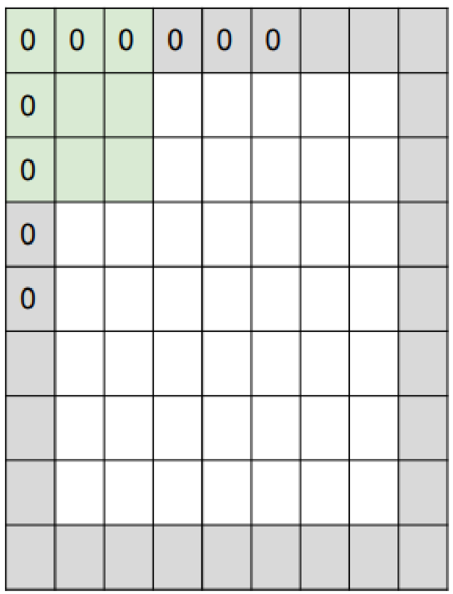


Output size:
(N - F) / stride + 1

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

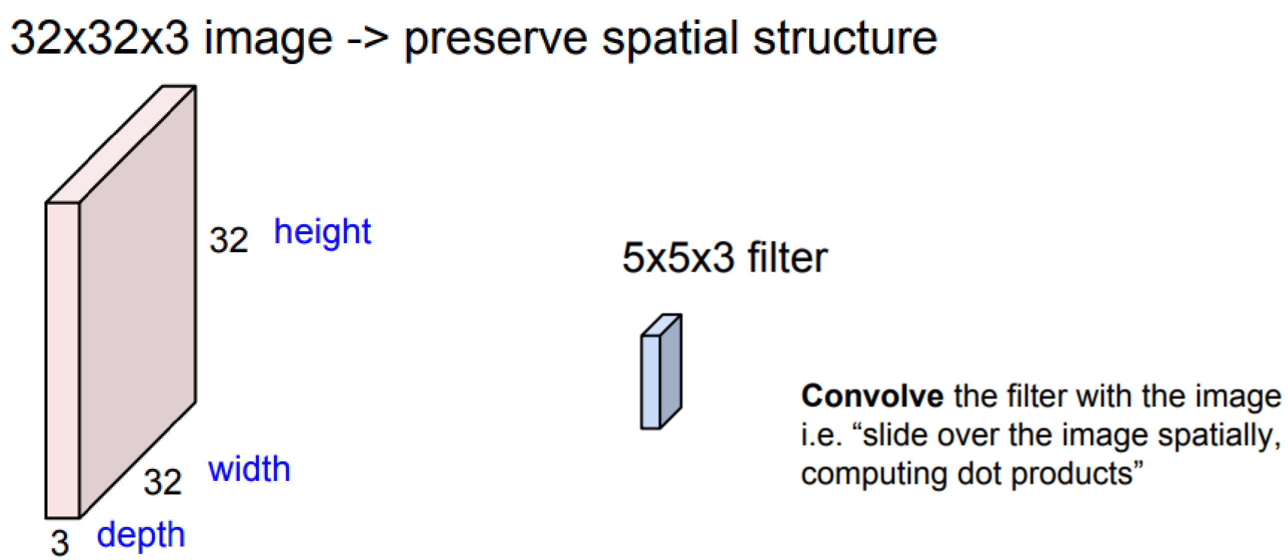


e.g. input 7x7
3x3 filter, applied with **stride 3**
pad with 1 pixel border => what is the output?

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

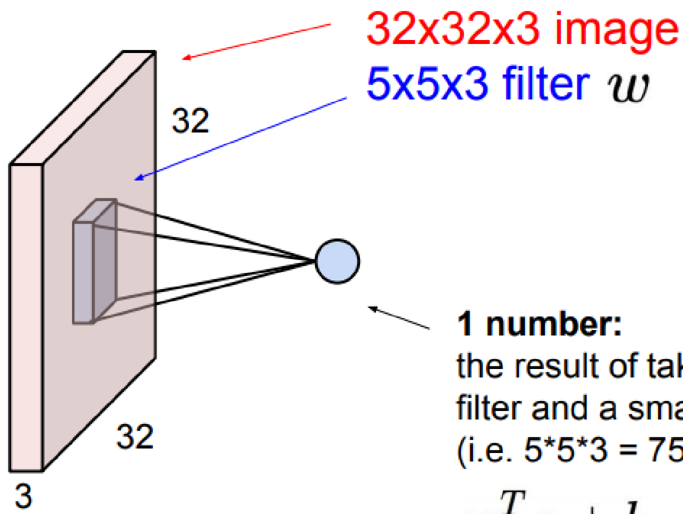
=> 3x3 output!

Convolution Layer



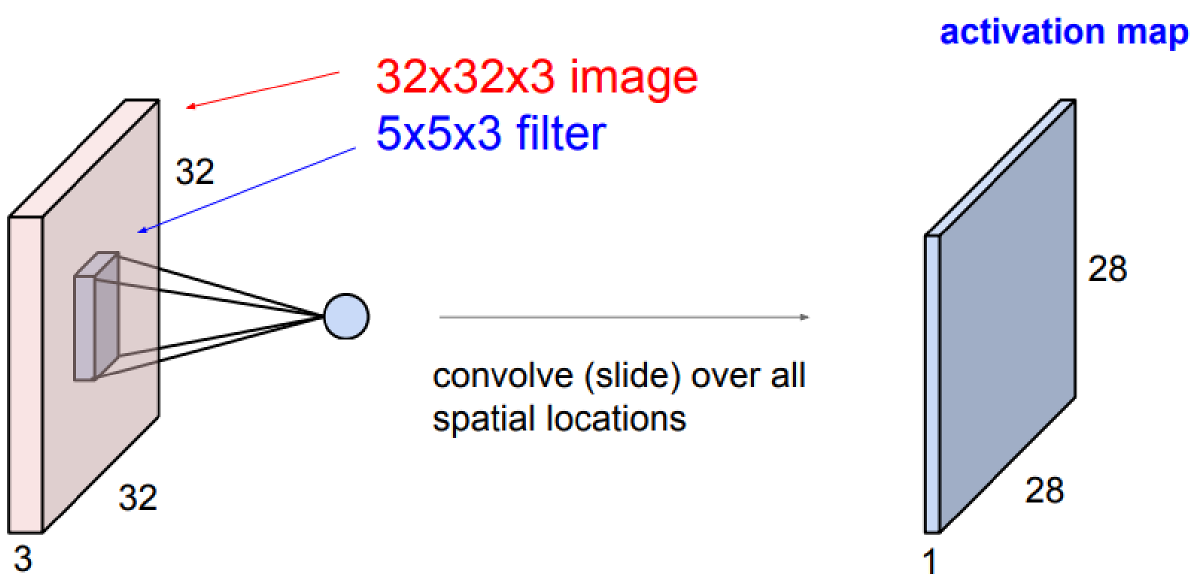
- Depth of filter always depth of input
- Computation is based only on local information

Convolution Layer

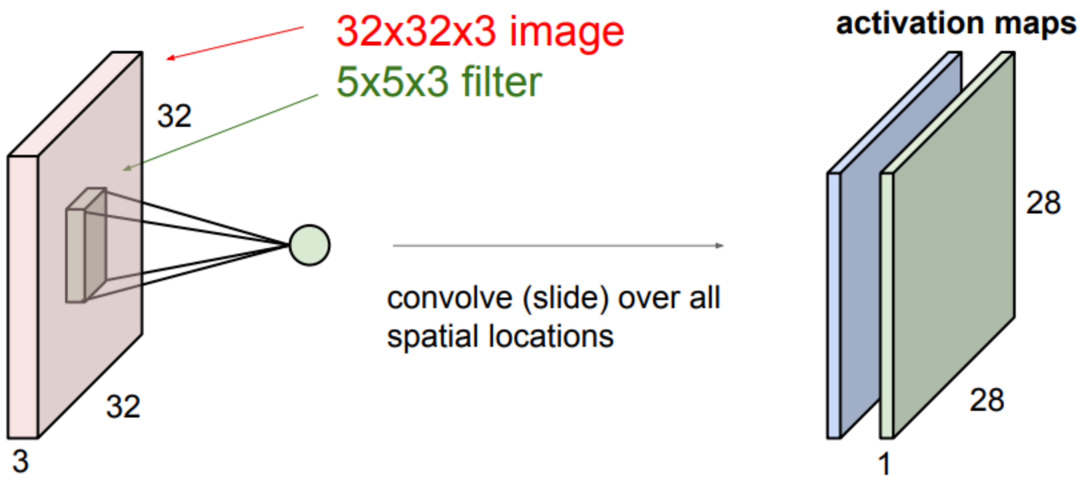


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

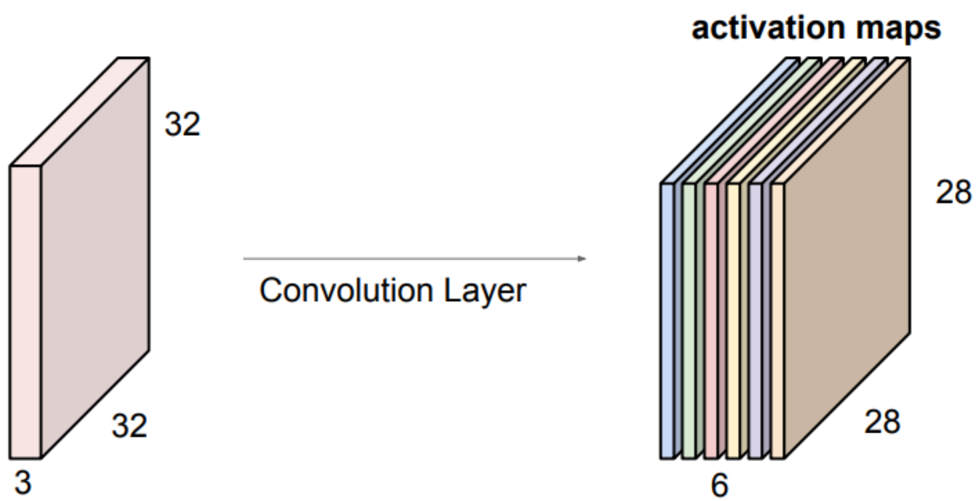
$$w^T x + b$$



Convolution Layer



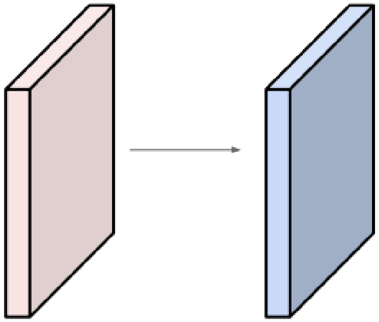
Second, green filter



6 filters

Examples

Examples time:



Input volume: **32x32x3**
10 5x5 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**

Summary: Convolution Layer

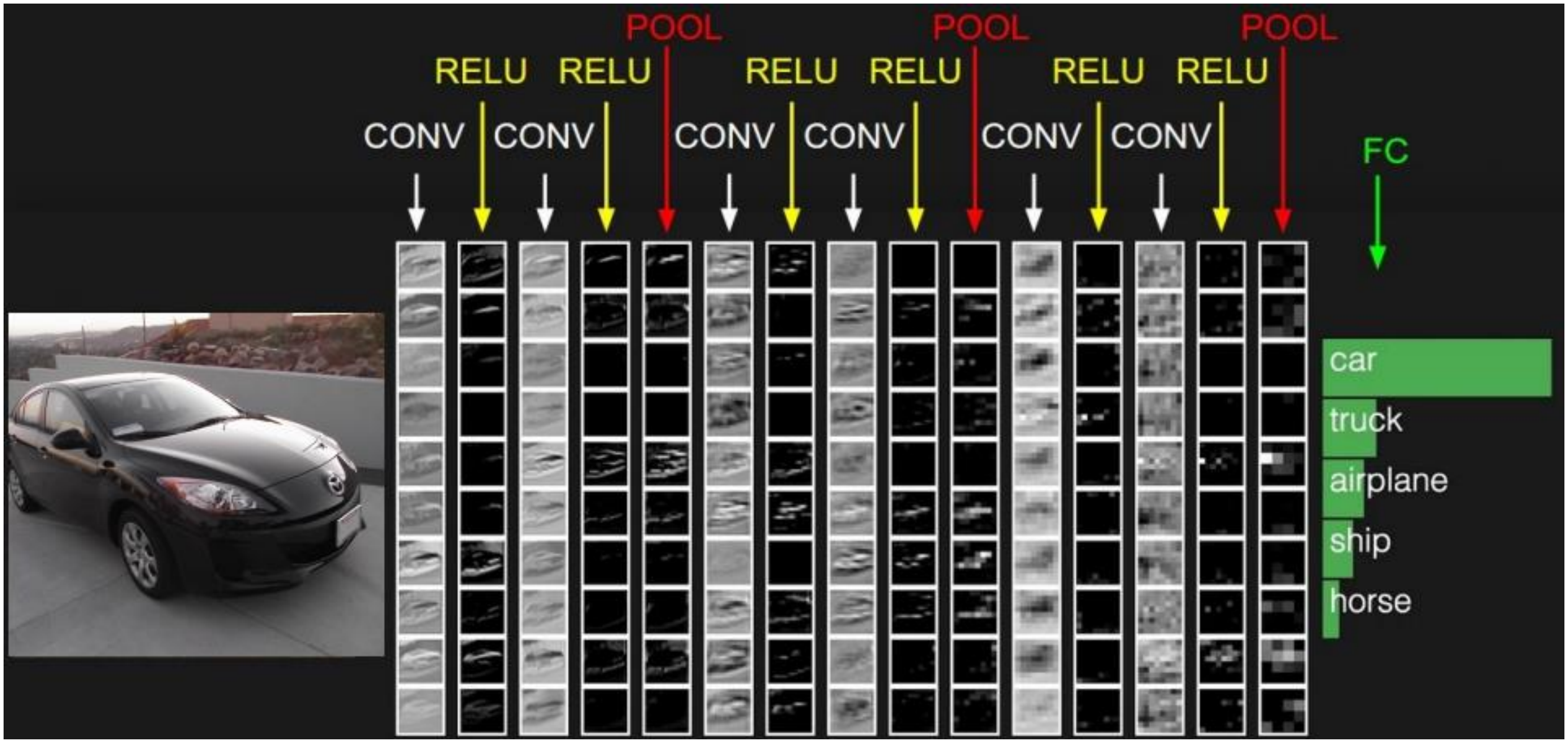
Summary. To summarize, the Conv Layer:

- 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

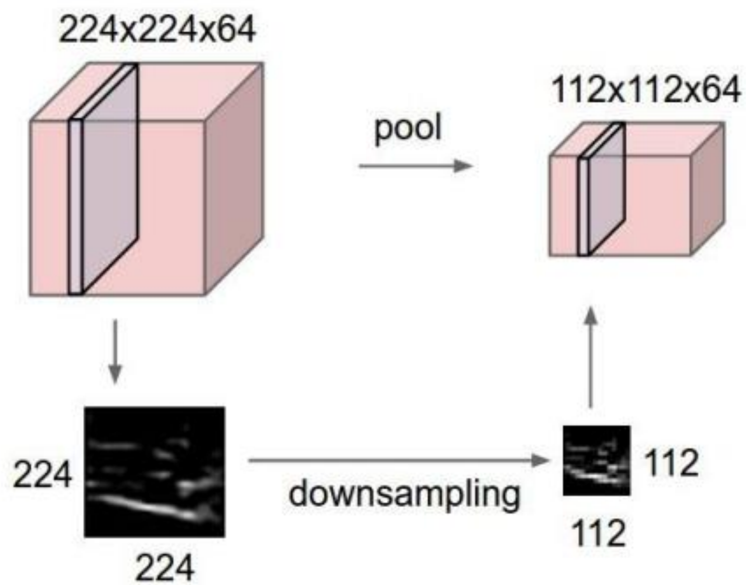
Convolutional Nets



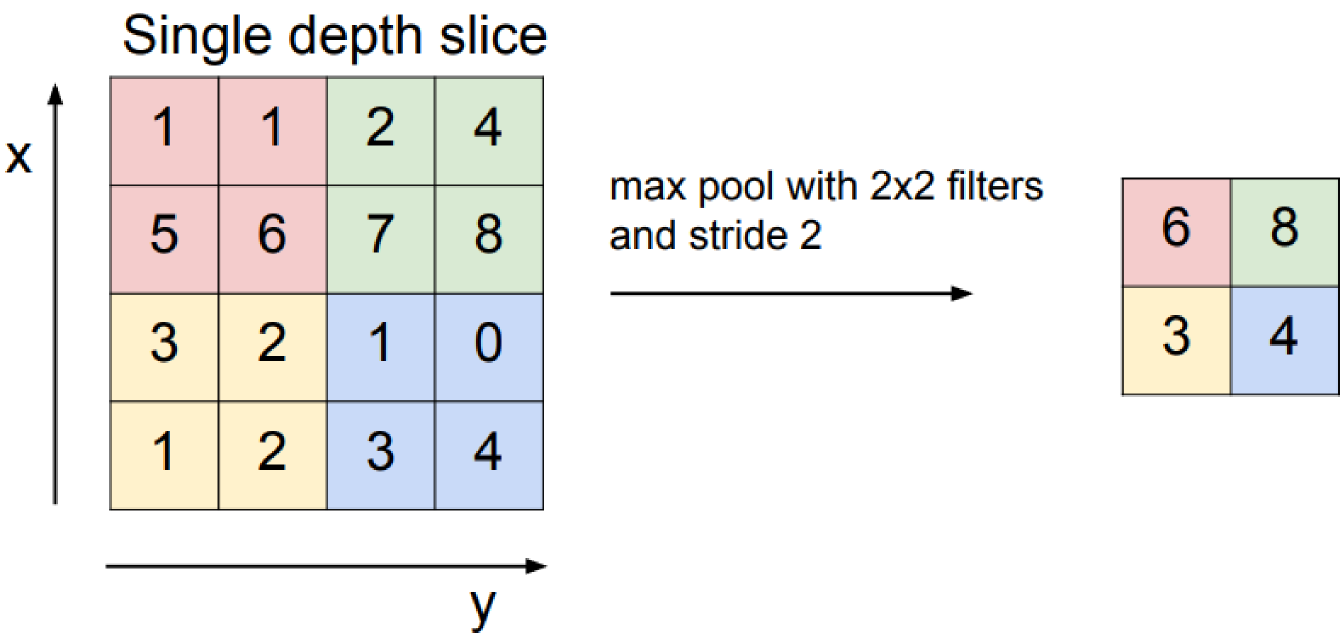
Pooling layer

Pooling layer

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



Max Pooling

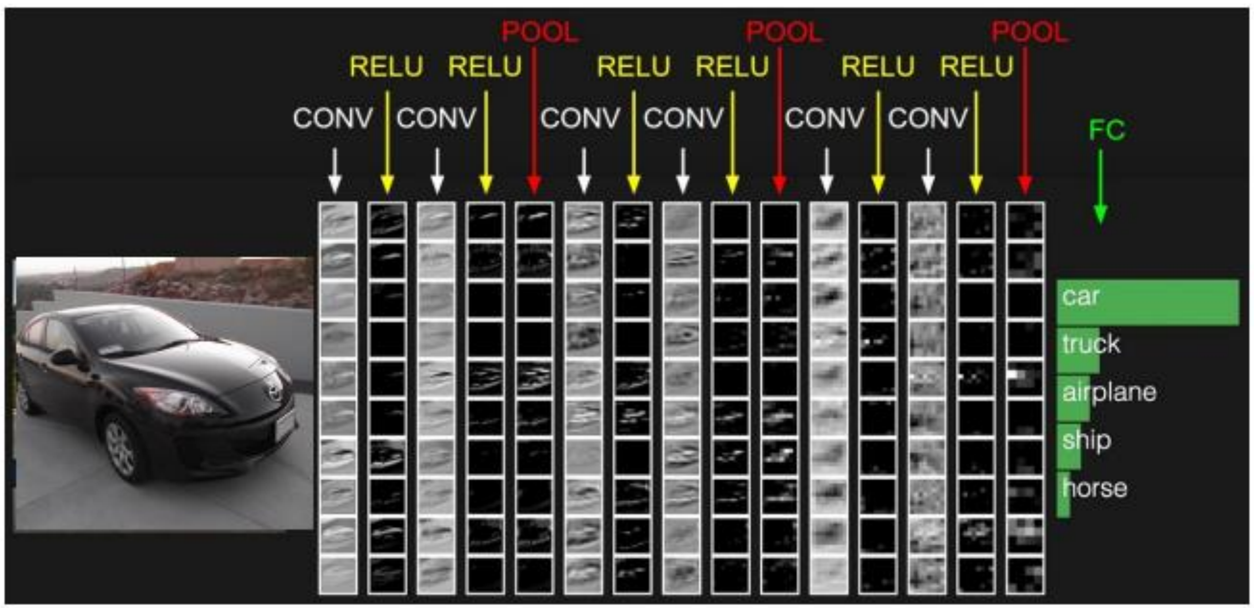


- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since 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)

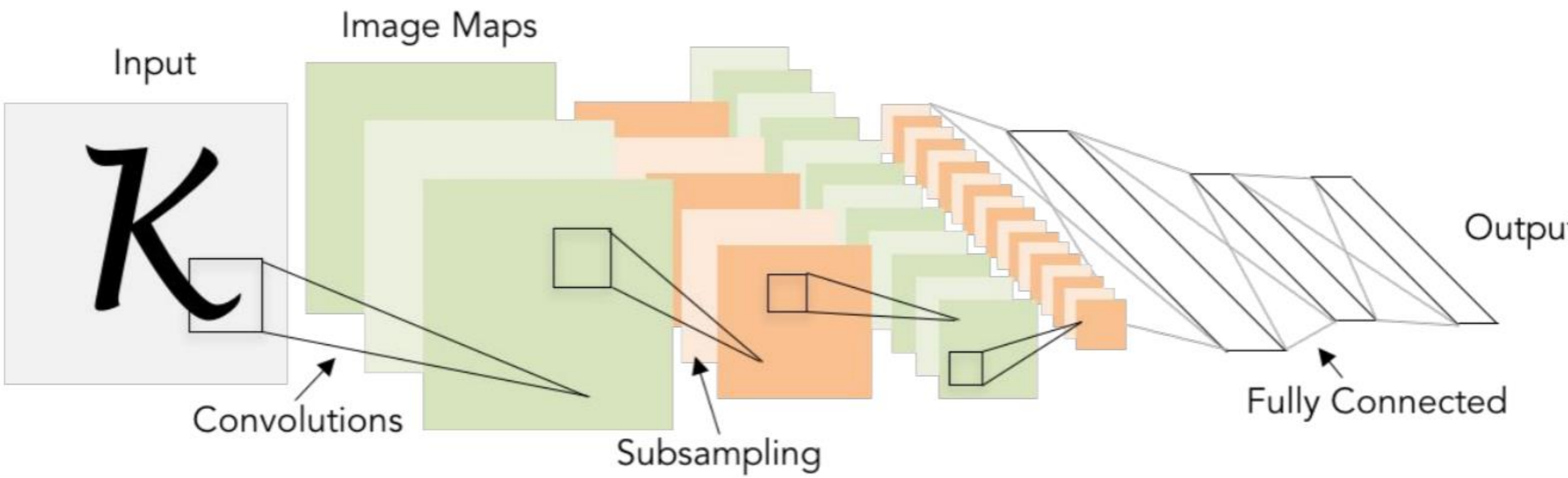
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



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

LeNet 5

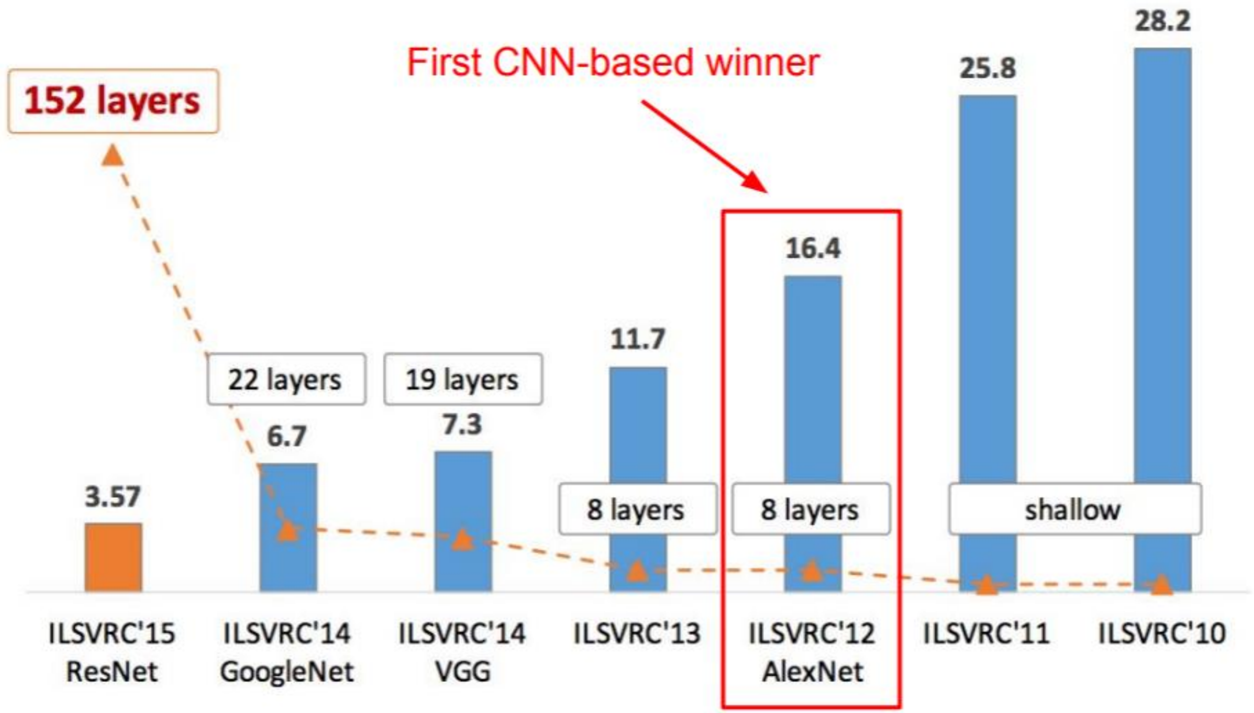
[LeCun et al., 1998]



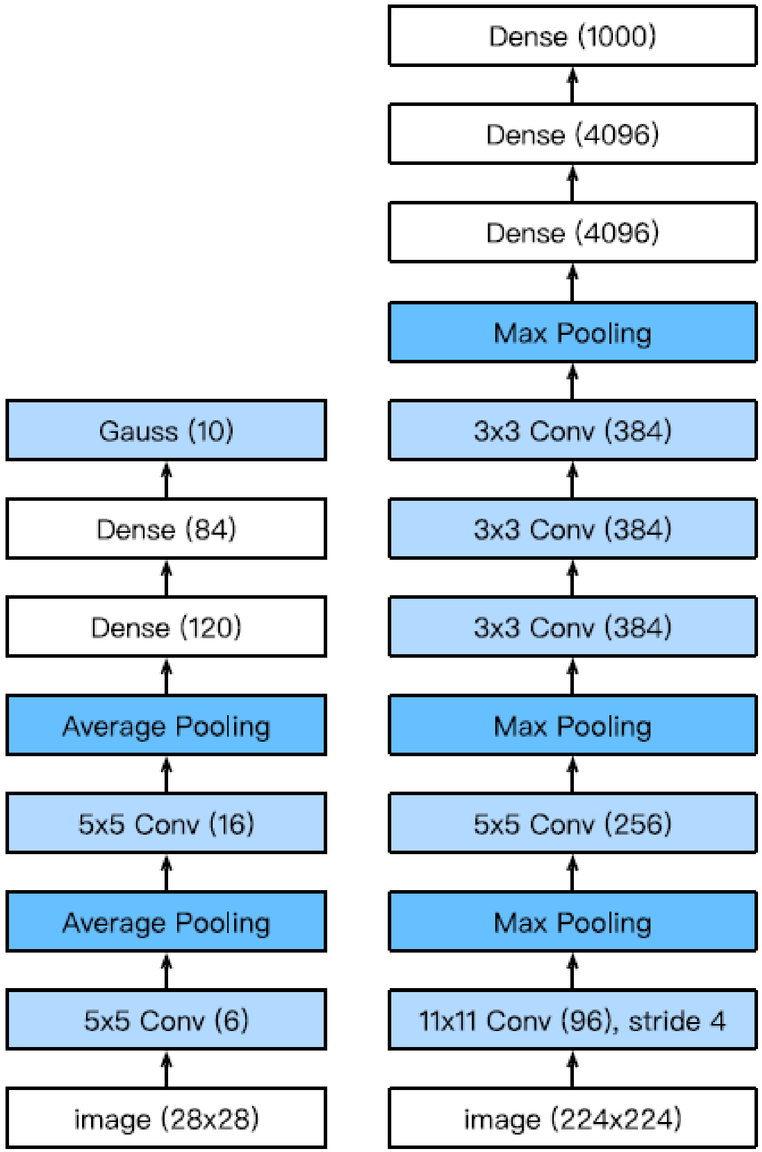
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

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

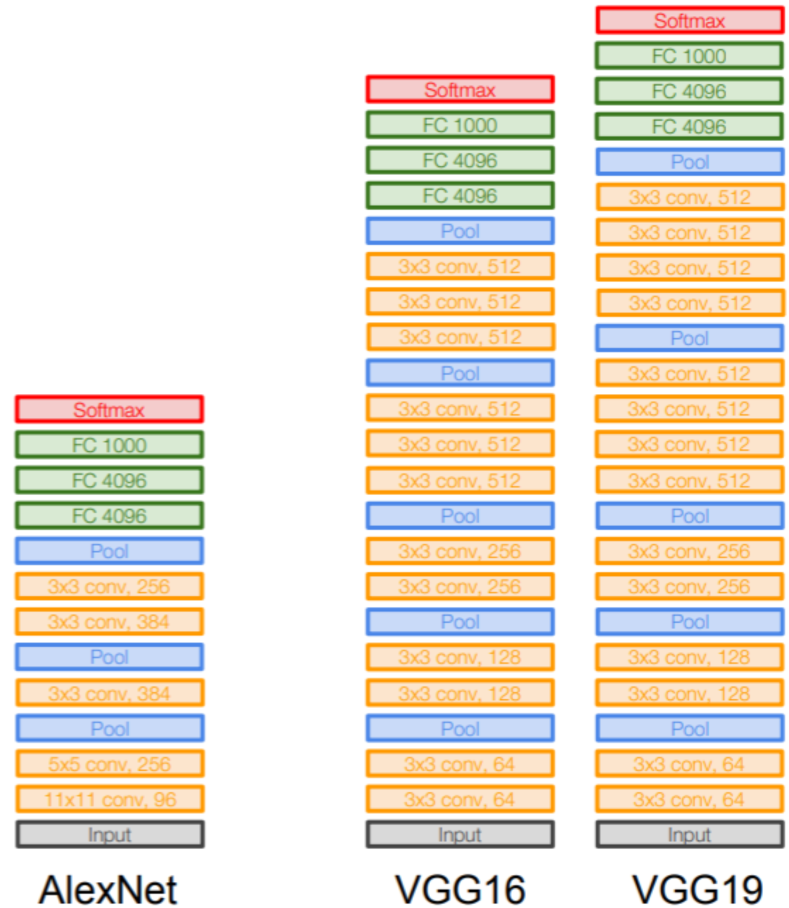
[Simonyan and Zisserman, 2014]

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

Lab: Load Data

```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')

    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, 28, 28, 1))
    X_test = np.reshape(X_test, (10000, 28, 28, 1))

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

Matrix form

Model Architecture

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

    print("Compiling Model")
    model = Sequential()
    model.add(layers.Conv2D(10, (3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(5, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))

    model.summary()

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

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

Vector form

Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 10)        100
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 10)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 5)         455
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 5)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 125)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8064
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 9,269
Trainable params: 9,269
Non-trainable params: 0
```

Results

```
totalMemory: 11.90GiB freeMemory: 11.74GiB
2019-03-20 15:23:18.838024: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1308]
2019-03-20 15:23:19.083693: I tensorflow/core/common_runtime/gpu/gpu_device.cc:989]
with 11374 MB memory) -> physical GPU (device: 0, name: TITAN X (Pascal), pci bus id
3s - loss: 0.6465 - acc: 0.8064 - val_loss: 0.3107 - val_acc: 0.9080
Epoch 2/10
1s - loss: 0.2527 - acc: 0.9233 - val_loss: 0.2123 - val_acc: 0.9326
Epoch 3/10
1s - loss: 0.1777 - acc: 0.9466 - val_loss: 0.1556 - val_acc: 0.9550
Epoch 4/10
1s - loss: 0.1386 - acc: 0.9578 - val_loss: 0.1303 - val_acc: 0.9615
Epoch 5/10
1s - loss: 0.1164 - acc: 0.9649 - val_loss: 0.1062 - val_acc: 0.9692
Epoch 6/10
1s - loss: 0.0996 - acc: 0.9697 - val_loss: 0.1032 - val_acc: 0.9677
Epoch 7/10
1s - loss: 0.0882 - acc: 0.9732 - val_loss: 0.0798 - val_acc: 0.9749
Epoch 8/10
1s - loss: 0.0787 - acc: 0.9758 - val_loss: 0.0676 - val_acc: 0.9799
Epoch 9/10
1s - loss: 0.0711 - acc: 0.9783 - val_loss: 0.0680 - val_acc: 0.9804
Epoch 10/10
1s - loss: 0.0664 - acc: 0.9802 - val_loss: 0.0652 - val_acc: 0.9789
Training duration:15.190229892730713
 9760/10000 [============================>.] - ETA: 0s
Network's test loss and accuracy:[0.065167549764638538, 0.97889999999999999]
[alina@dome MNIST]$
```

Summary CNNs

- Convolutional Nets are Feed-Forward Networks with at least one convolution layer and optionally max pooling layers
- Convolutions enable dimensionality reduction
- 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

Acknowledgements

- Slides made using resources from:
  - Yann LeCun
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Andrew Moore
- Thanks!