DS 5220

Supervised Machine Learning and Learning Theory

Alina Oprea
Associate Professor, CCIS
Northeastern University

November 18 2019

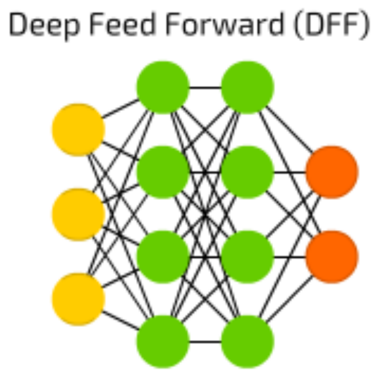
Outline

- Convolutional Neural Networks
  - Convolution layer
  - Max pooling layer
  - Examples of famous architectures
- Lab CNN
- Regularization of neural networks
  - Weight decay
  - Dropout regularization

Neural Network Architectures

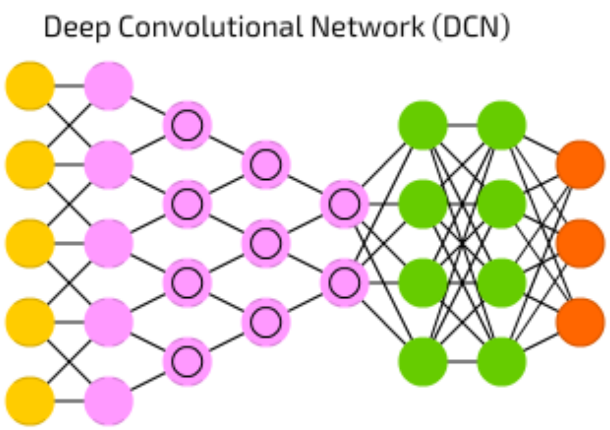
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



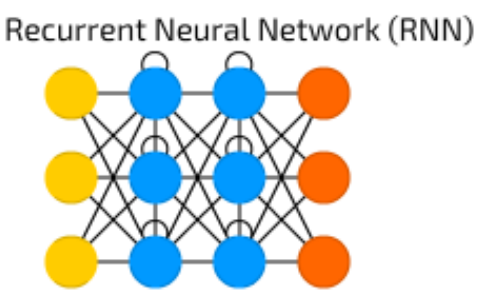
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations



Recurrent Networks

- Keep hidden state
- Have cycles in computational graph

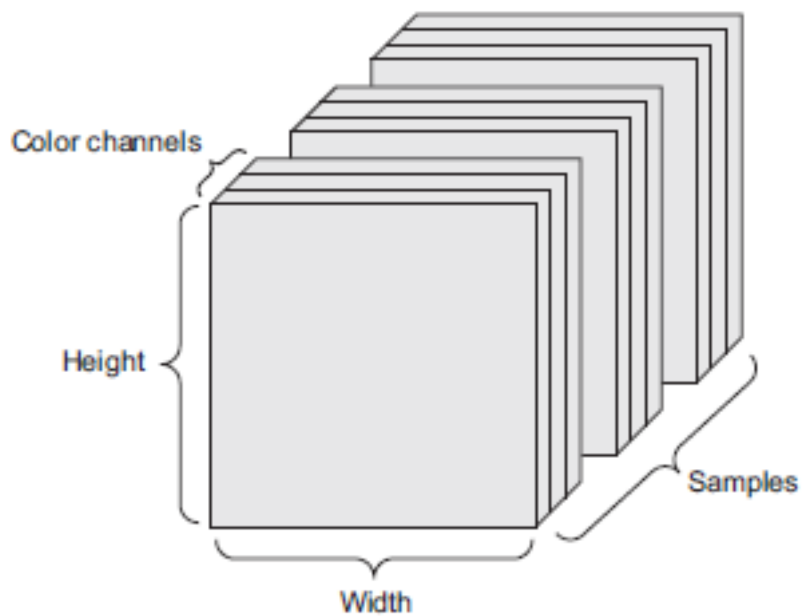


Convolutional Nets

- 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

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

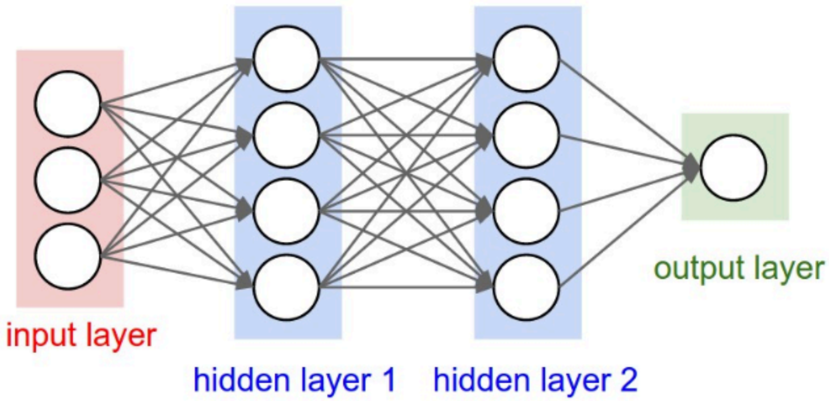


Computer vision principles

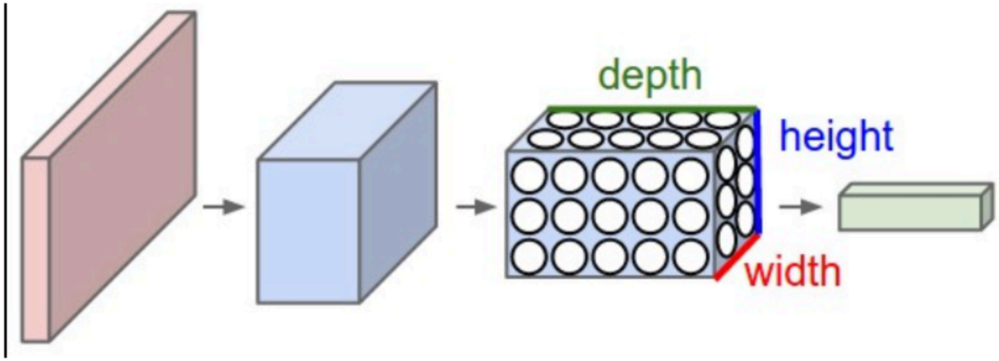
- Task: image classification (object identification)
- Translation invariance
  - Classification should work if object appears in different locations in the image => All image regions are treated the same
- Locality
  - Focus on local regions for object detection => computation should be local
- Mathematical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

Convolutional Neural Networks

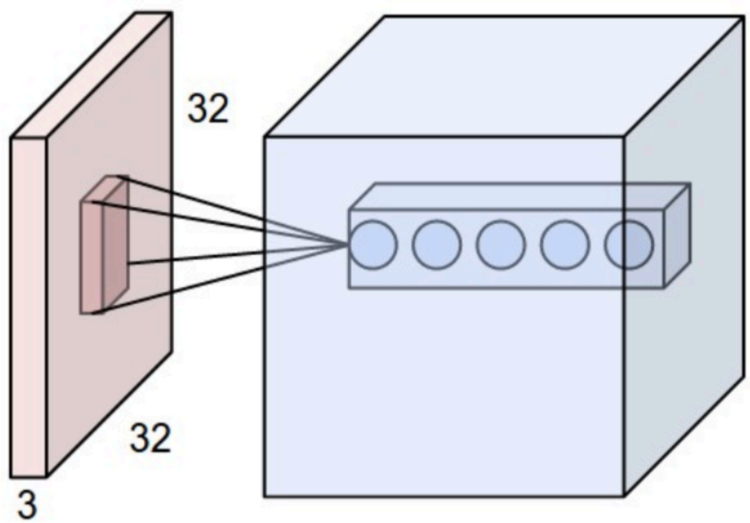
Feed-forward network



Convolutional network

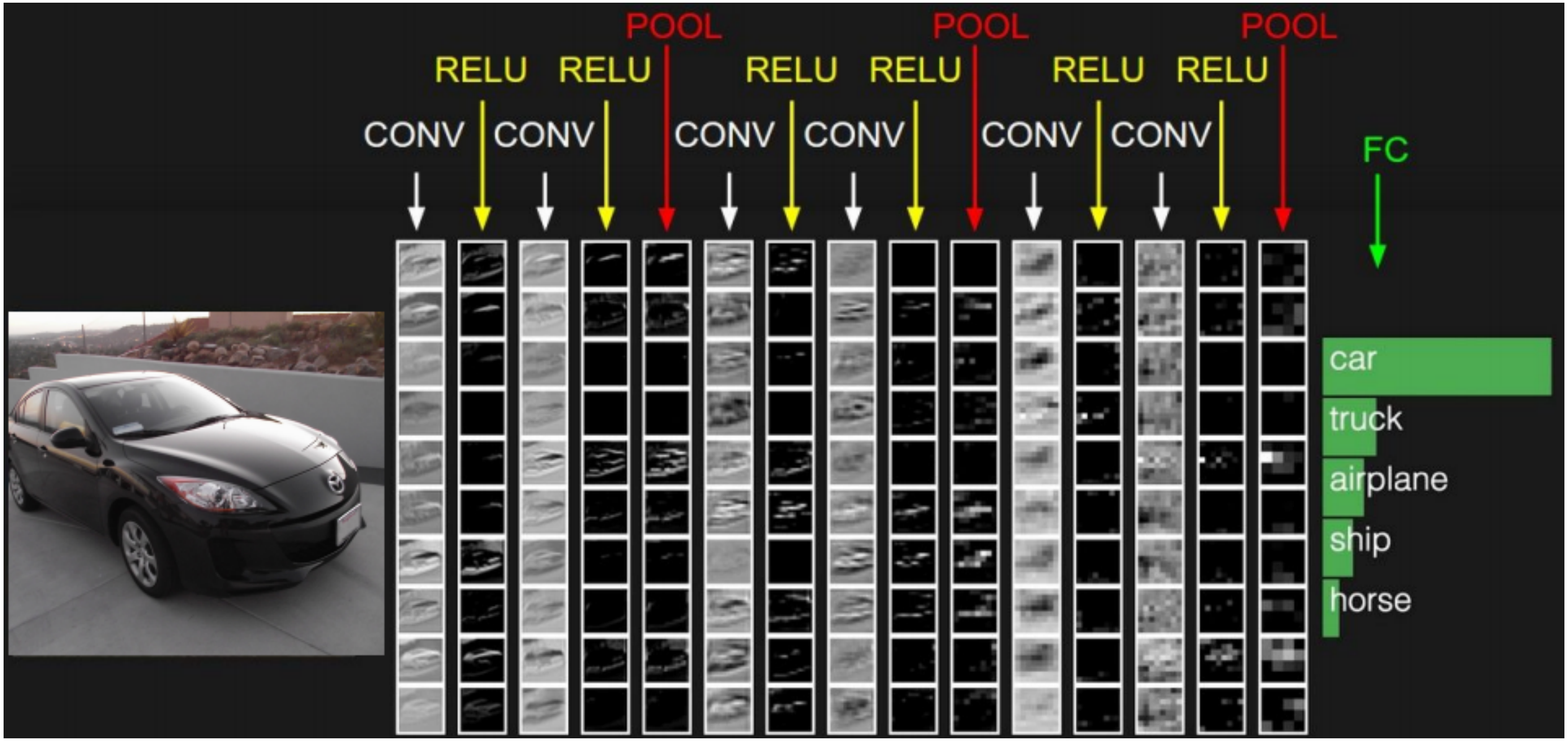


Filter



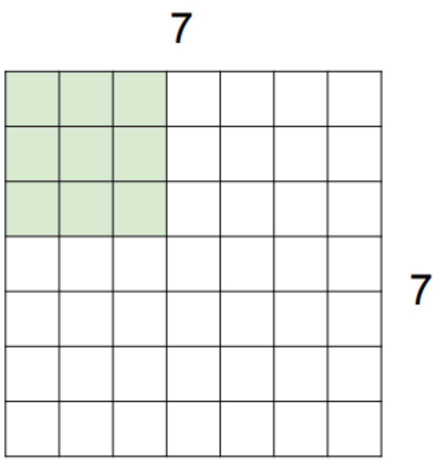
https://www.youtube.com/watch?v=f0t-OCG79-U

Convolutional Nets

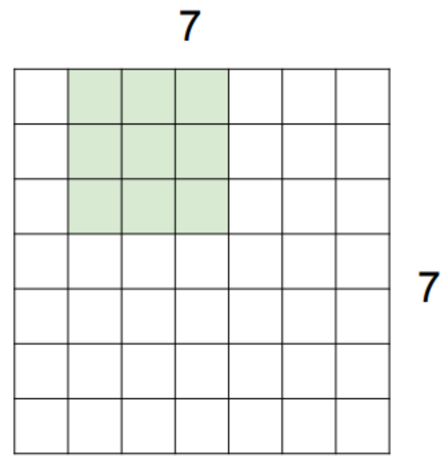


Convolutions

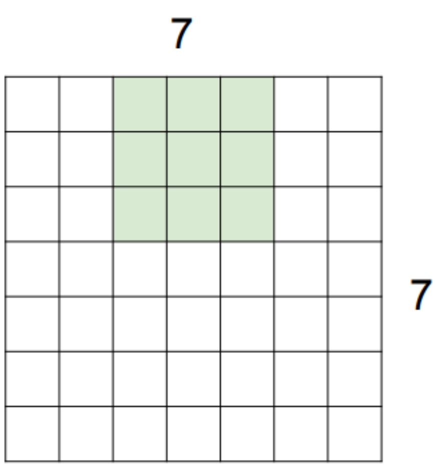
A closer look at spatial dimensions:

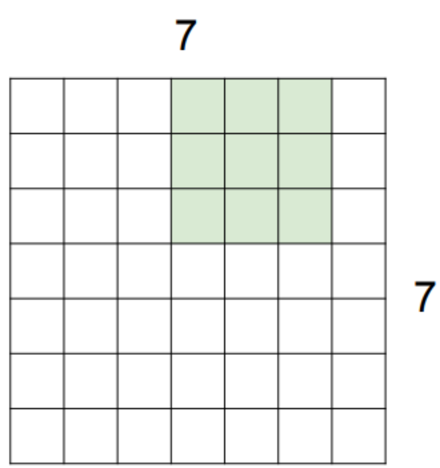


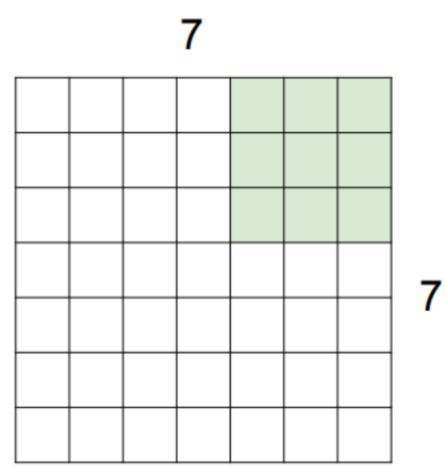
7x7 input (spatially)
assume 3x3 filter



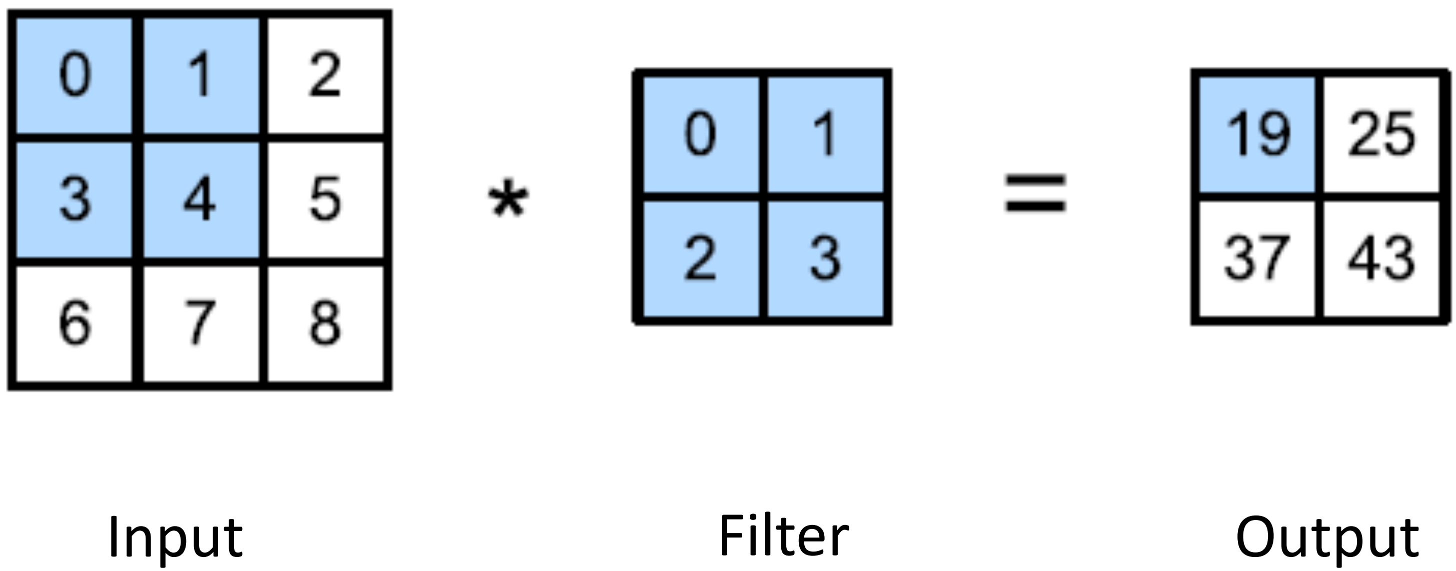
=> 5x5 output





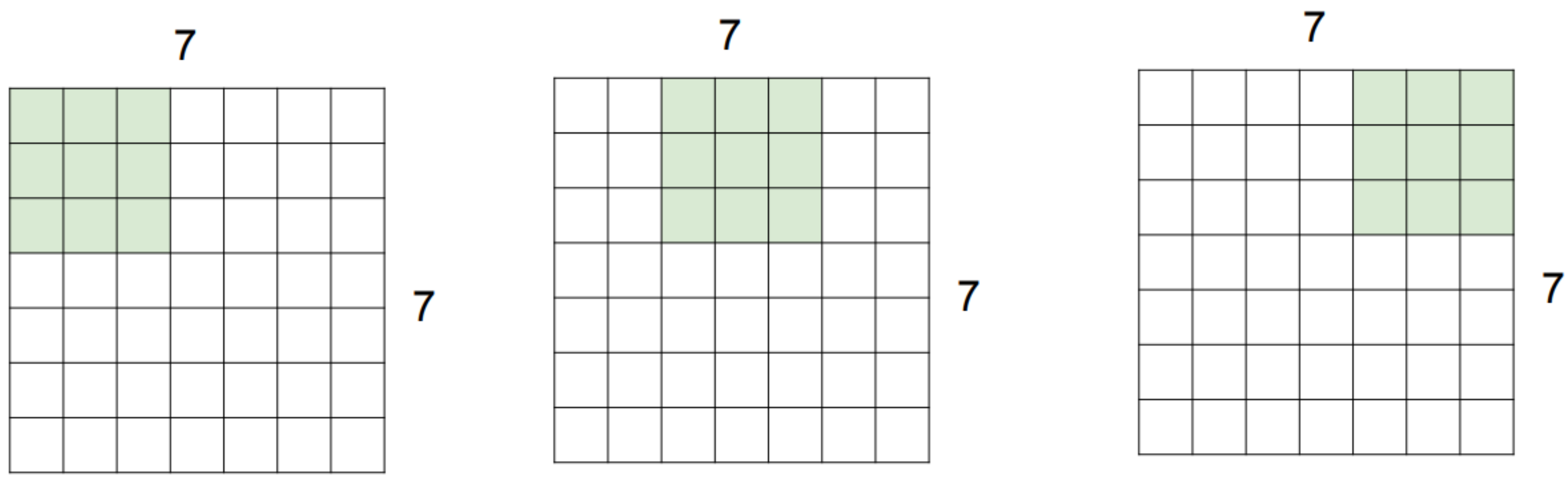


Example



Convolutions with stride

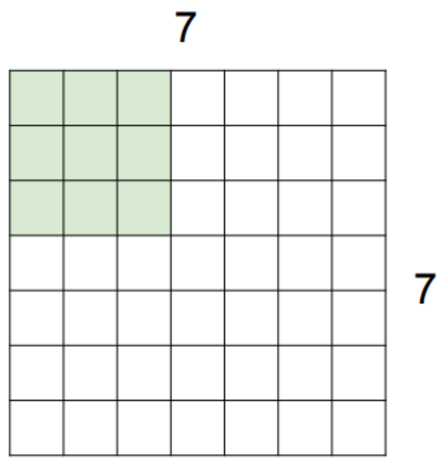
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



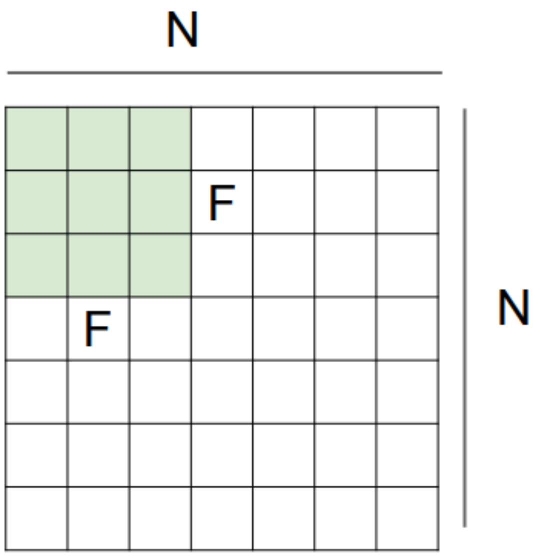
=> 3x3 output!

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.

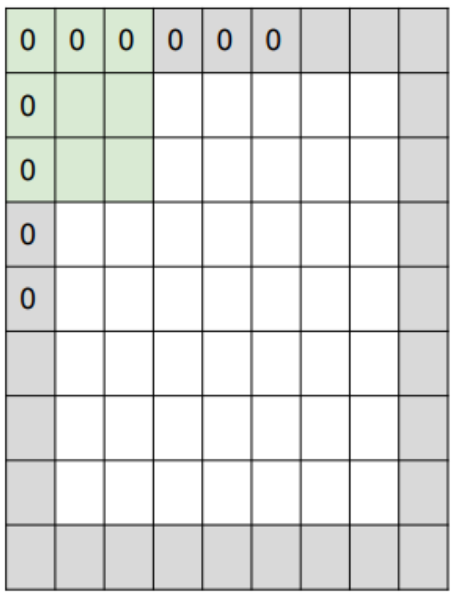


Output size:
(N - F) / stride + 1

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

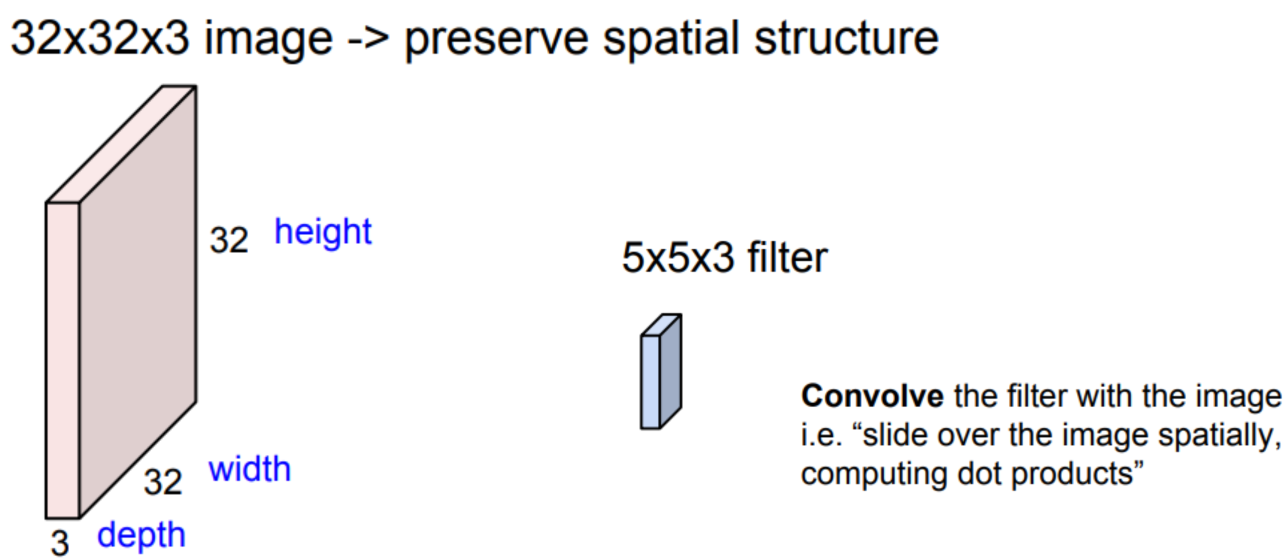


e.g. input 7x7
3x3 filter, applied with **stride 3**
pad with 1 pixel border => what is the output?

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

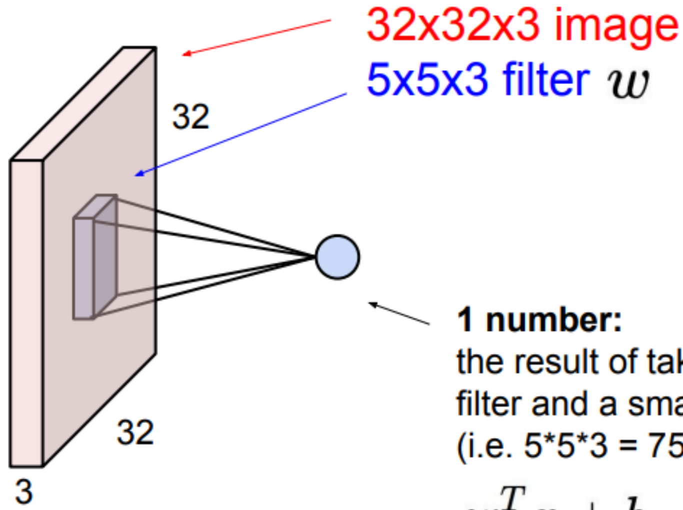
=> 3x3 output!

Convolution Layer



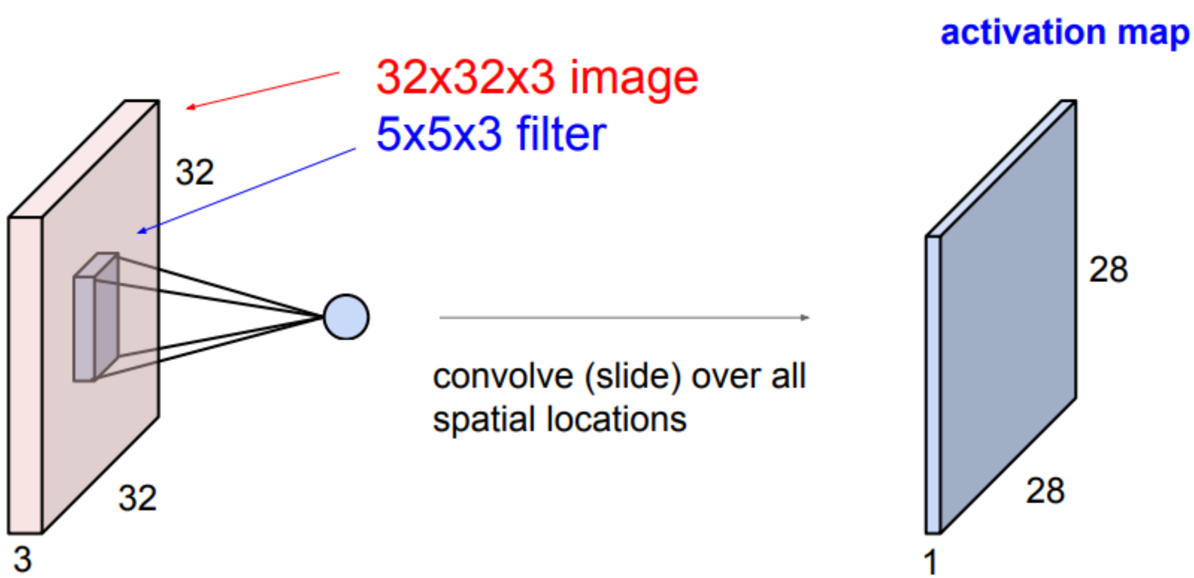
- Depth of filter always depth of input
- Computation is based only on local information

Convolution Layer

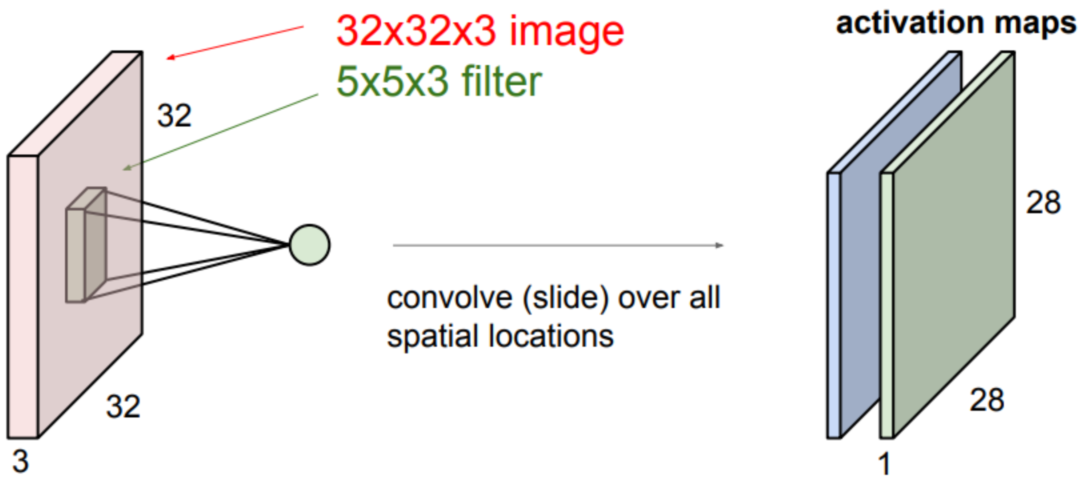


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

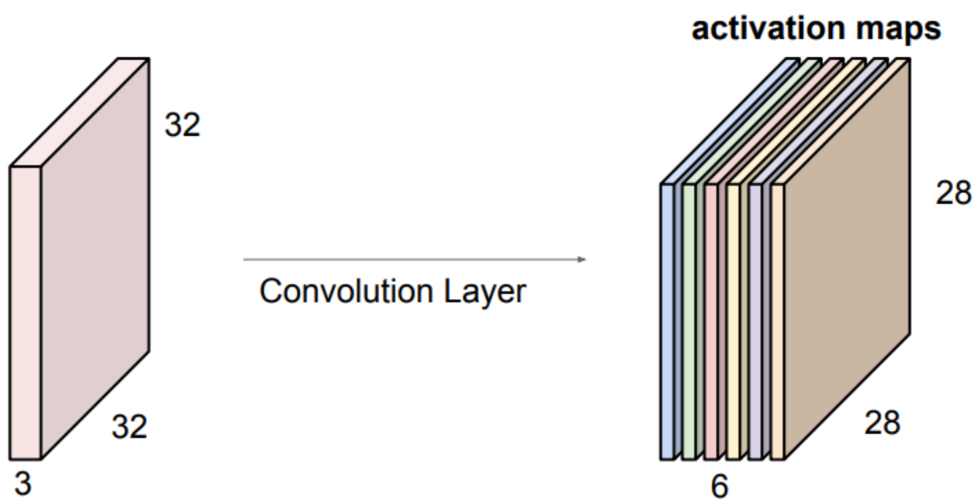
$$w^T x + b$$



Convolution Layer



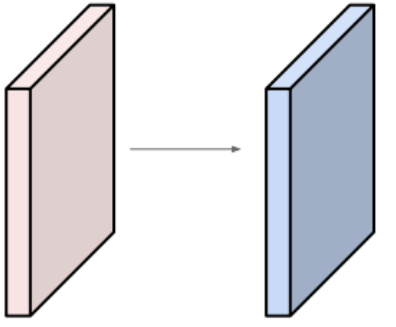
Second, green filter



6 filters

Examples

Examples time:



Input volume: **32x32x3**

10 5x5x3 filters with stride 1, pad 2

Output volume size: ?

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

32x32x10

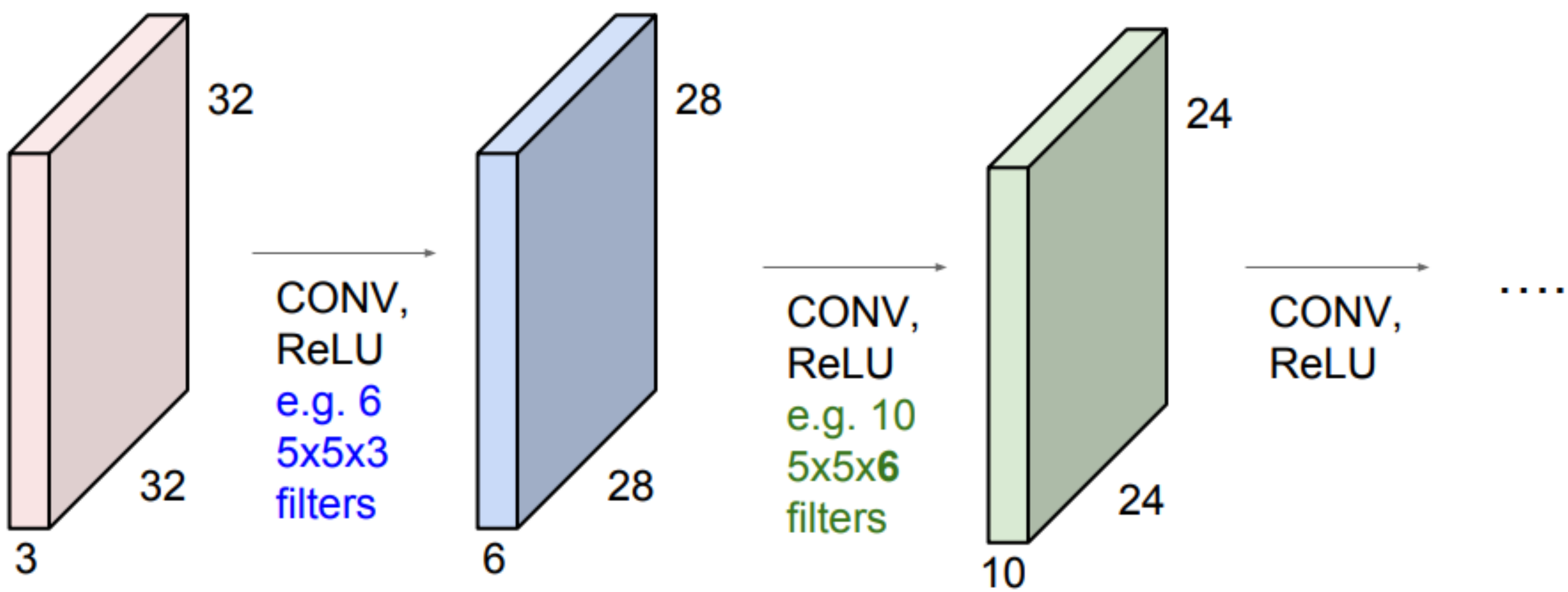
Number of parameters in this layer?

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



Summary: Convolution Layer

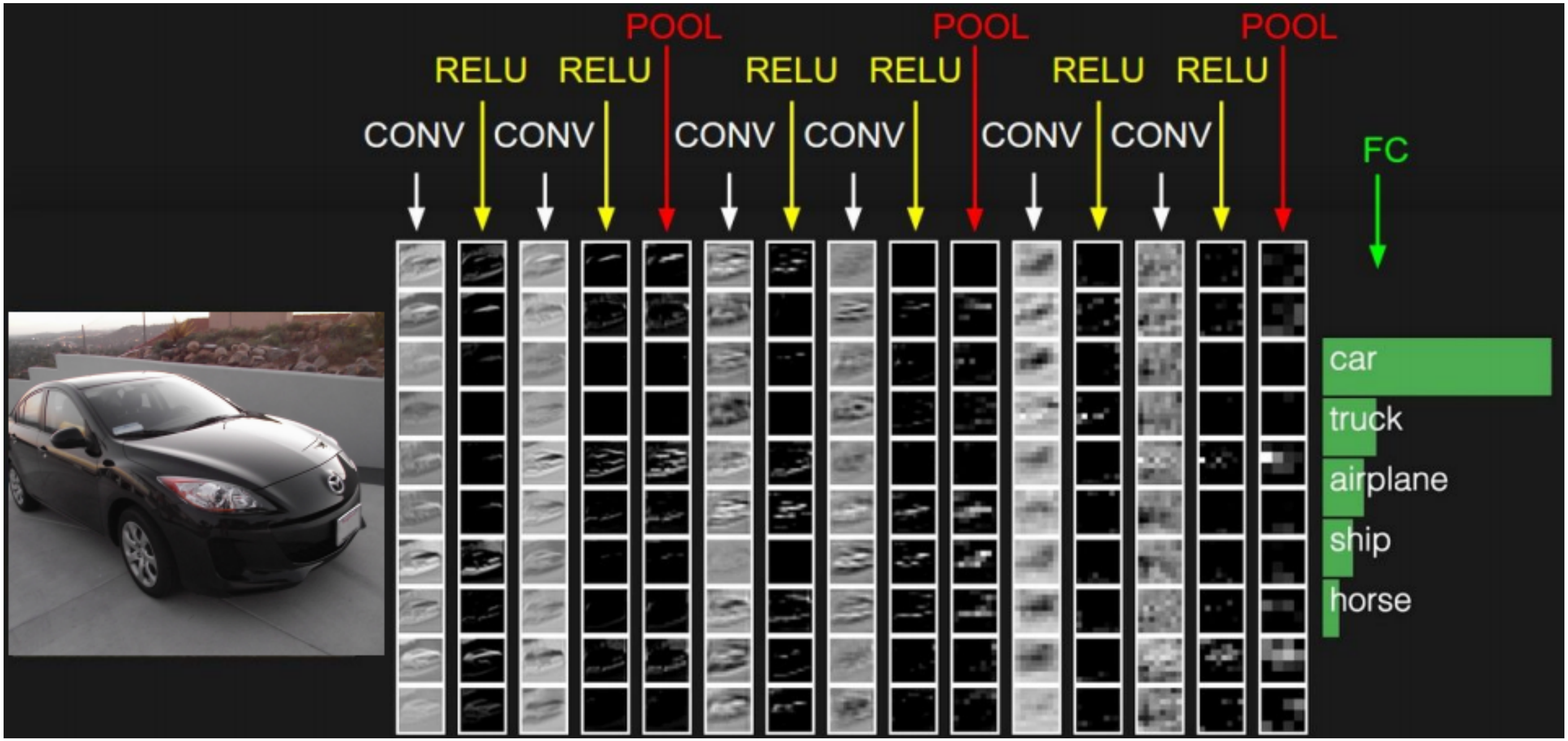
Summary. To summarize, the Conv Layer:

- 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

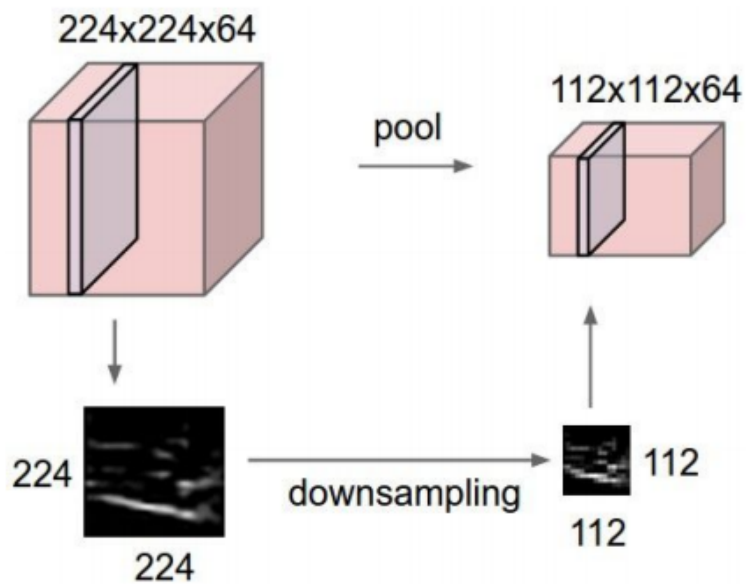
Convolutional Nets



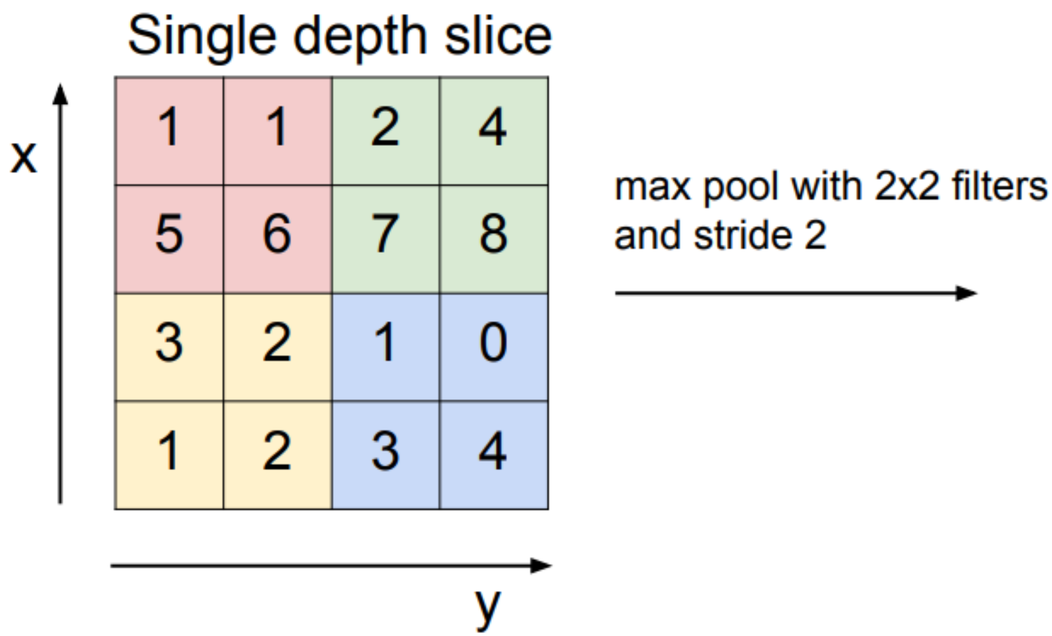
Pooling layer

Pooling layer

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



Max Pooling

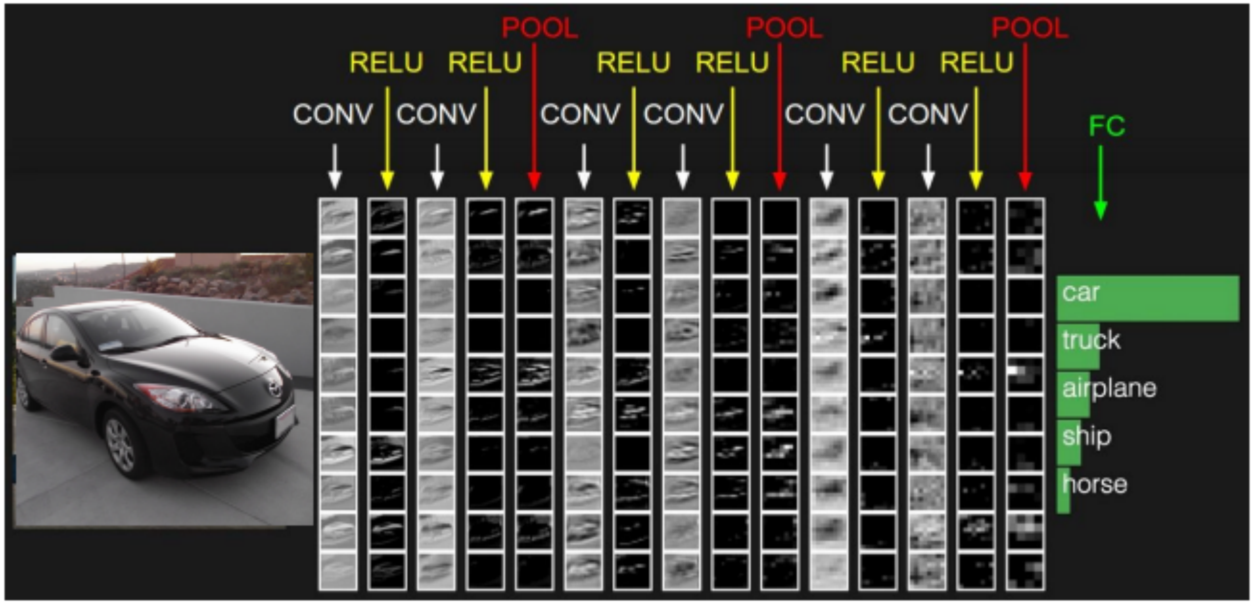


- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since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

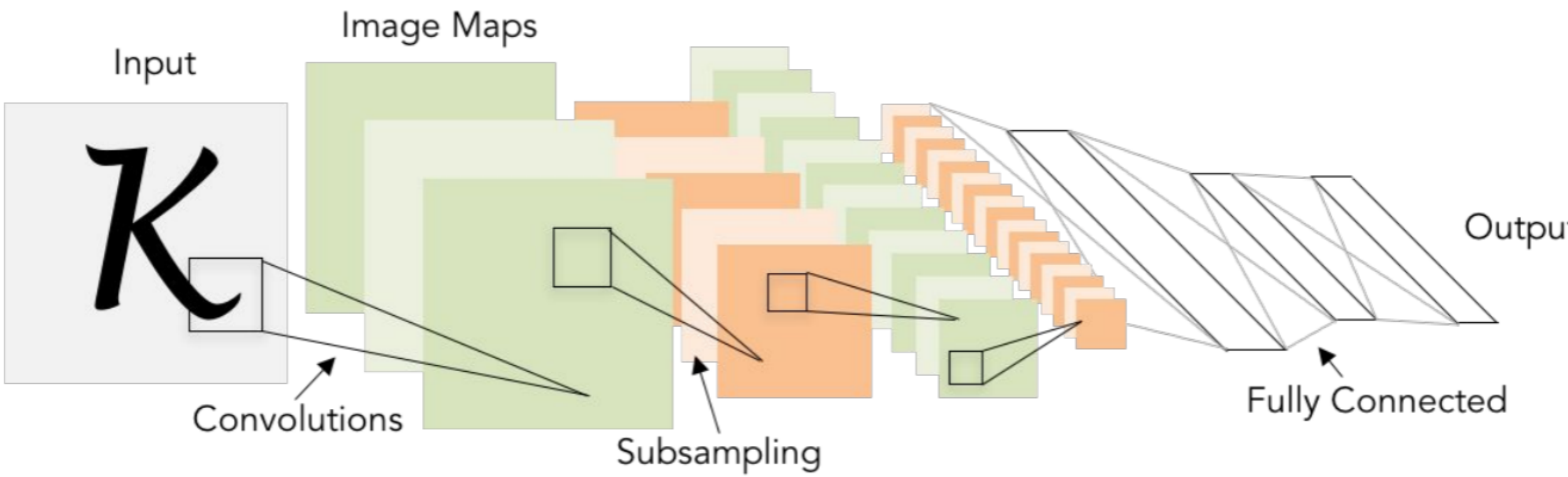
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



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

LeNet 5

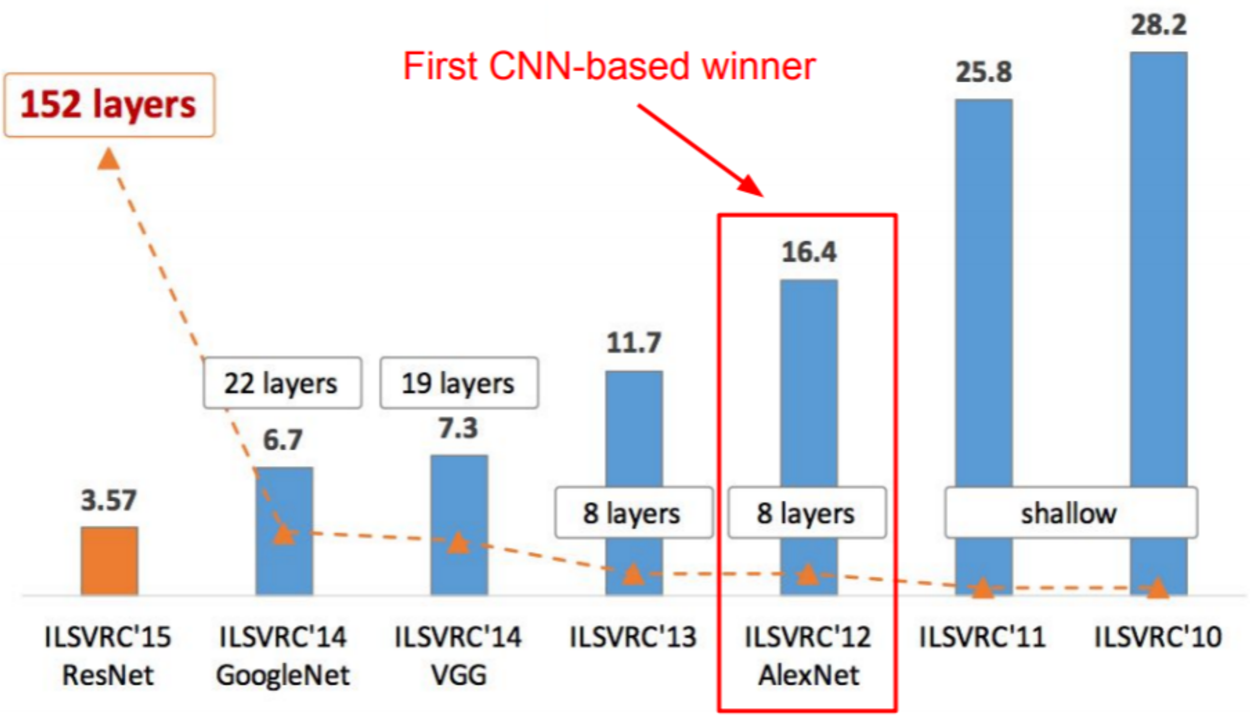
[LeCun et al., 1998]



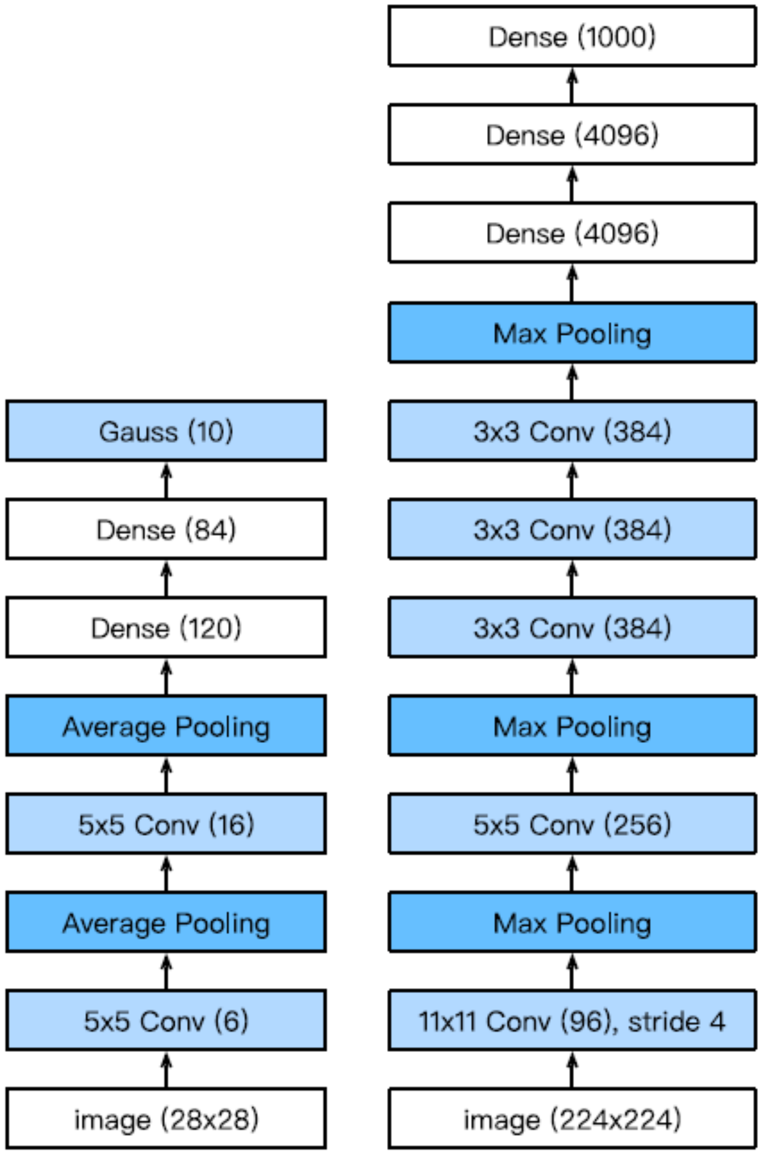
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

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

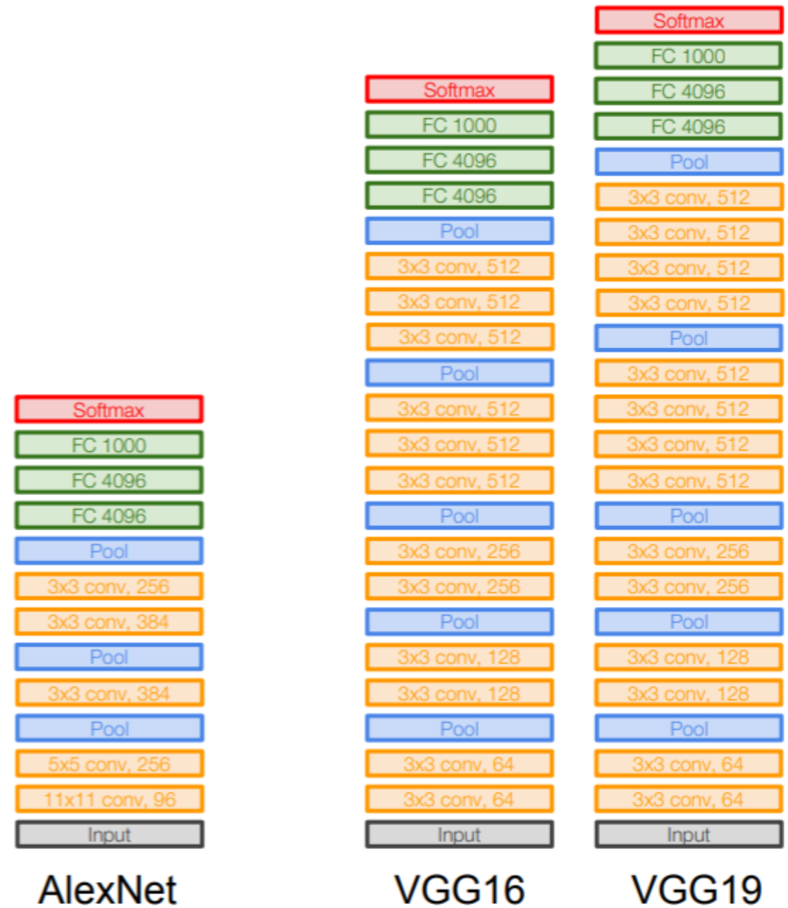
[Simonyan and Zisserman, 2014]

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

```
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
```

10 filters, size 3x3x1

5 filters, size 3x3x10

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
```

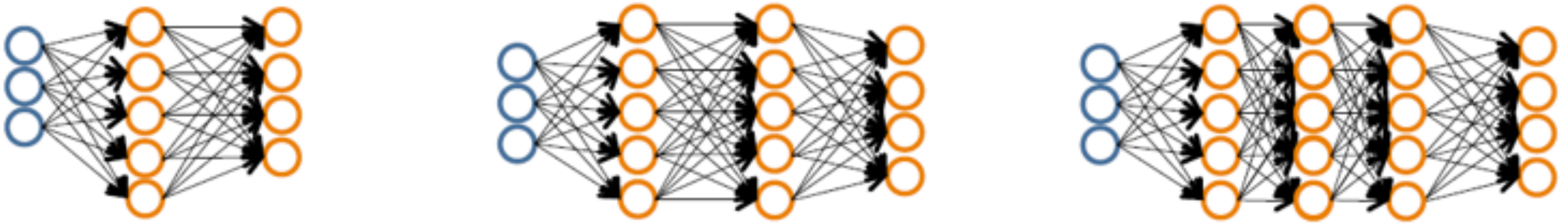
Results

```
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:[0.06516754976463853 8, 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, 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

Overfitting



- The larger the network, the higher the capacity (more model parameters)
- But also more prone to overfitting!

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing too well on training data}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

λ = regularization strength (hyperparameter)

<u>L2 regularization:</u> $R(W) = \sum_k \sum_l W_{k,l}^2$ → Weight decay

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

- When computing gradients of loss function, regularization term needs to be taken into account

Dropout

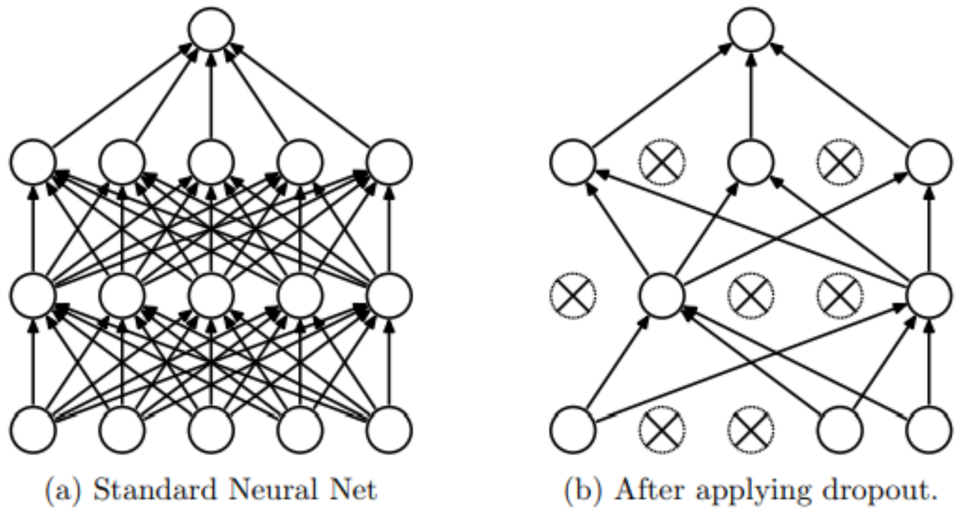


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- Regularization technique that has proven very effective for deep learning
- Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 2014

Dropout

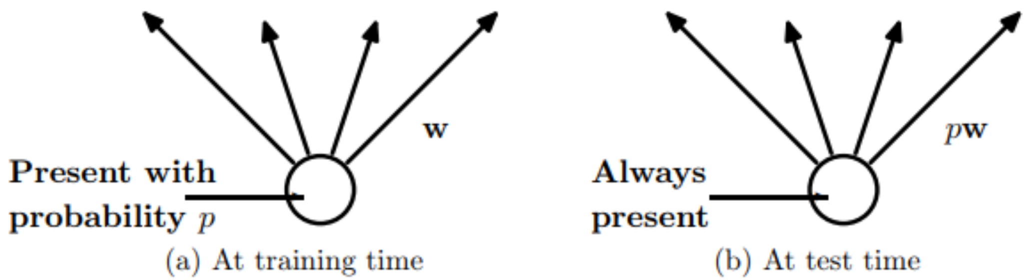


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

- At training time, sample a sub-network and learn weights
  - Keep each neuron with probability p
- At testing time, all neurons are there, but reduce weight by a factor of p

Dropout Implementation

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

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

    # Hidden Layer 1
    model.add(Dense(500, input_dim=784))
    model.add(Dropout(0.3))
    model.add(Activation('relu'))

    # Hidden Layer 2
    model.add(Dense(300))
    model.add(Dropout(0.3))
    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
```

Dropout layers

Results on MNIST

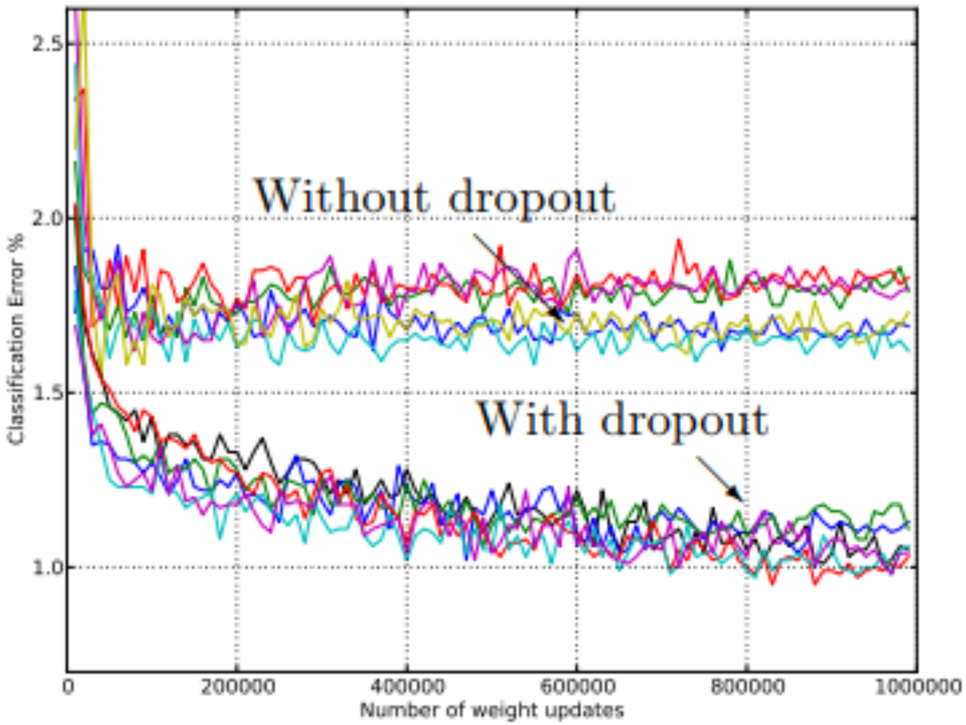


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Acknowledgements

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