Convolutional neural nets: TensorFlow

Glenn Bruns
CSUMB

Much material in this deck from Géron, Hands-on Machine Learning with Scikit-Learn and TensorFlow

Learning outcomes

After this lecture you should be able to:

- implement convolutional neural nets in TensorFlow
- understand and use pooling
- describe the structure of typical large CNNs

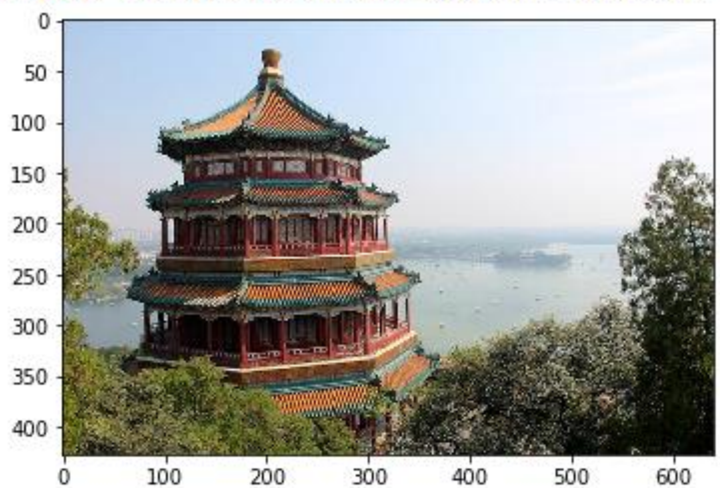
Build a convolutional layer in TF

```
# Load sample images
china = load_sample_image("china.jpg")
flower = load_sample_image("flower.jpg")
dataset = np.array([china, flower], dtype=np.float32)
batch_size, height, width, channels = dataset.shape

# 7x7 sized filters, 2 channels, and 2 filters
filters = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters[:, 3, :, 0] = 1  # vertical line
filters[3, :, :, 1] = 1  # horizontal line

# Create a graph with one convolutional layer applying the 2 filters
tf.reset_default_graph()
X = tf.placeholder(tf.float32, shape=(None, height, width, channels))
convolution = tf.nn.conv2d(X, filters, strides=[1,2,2,1],
                           padding="SAME")

# execute
with tf.Session() as sess:
    output = sess.run(convolution, feed_dict={X: dataset})
```

Loading the images

```
# Load sample images
china = load_sample_image("china.jpg")
flower = load_sample_image("flower.jpg")
dataset = np.array([china, flower], dtype=np.float32)
batch_size, height, width, channels = dataset.shape

# display the image
plt.imshow(china)
```

```
In [86]: plt.imshow(china)
Out[86]: <matplotlib.image.AxesImage at 0x15ab7748>
```




```
china.shape
Out[87]: (427, 640, 3)
```

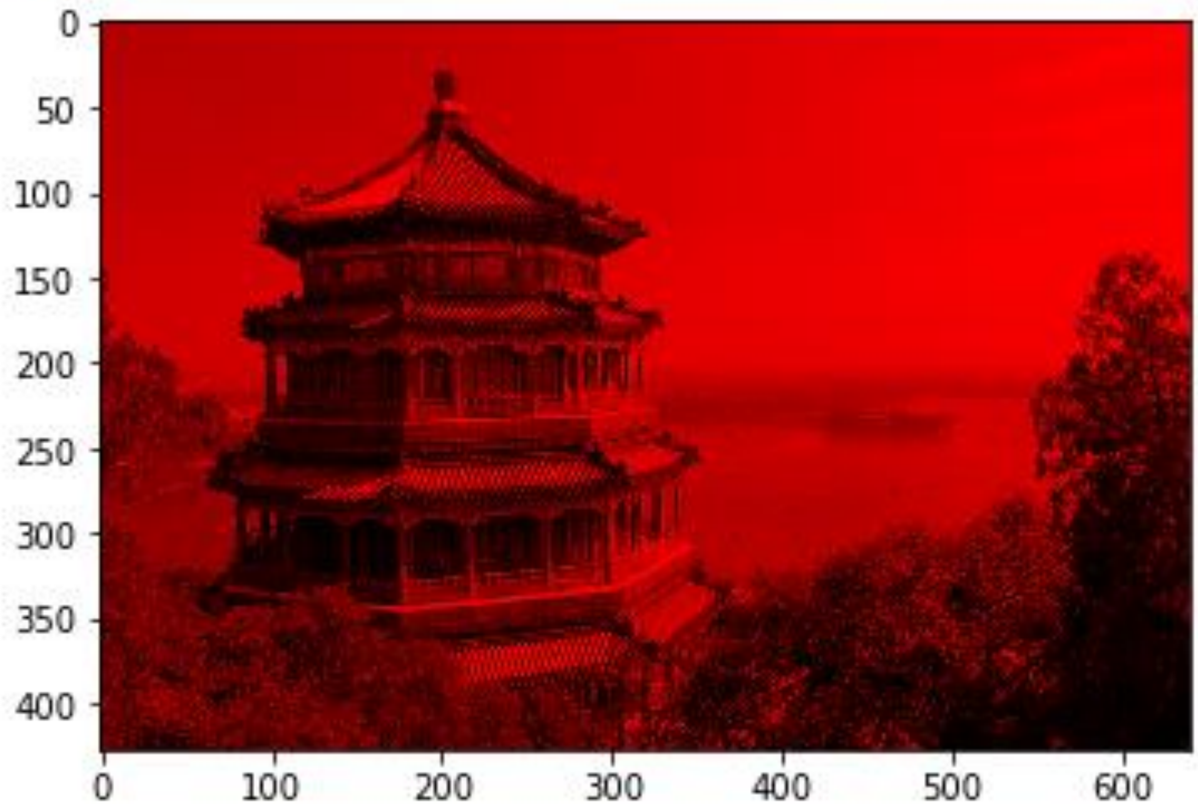
height

width

number of channels
(red, green, blue)

Detail on channels

```
In [92]: plt.imshow(china)
    ...: temp = np.zeros(china.shape, dtype='uint8')
    ...: temp[:,:,0] = china[:,:,0]
    ...: plt.imshow(temp)
Out[92]: <matplotlib.image.AxesImage at 0x12e1b8d0>
```



Channel 0 component of the image

Defining the filters

```
# 7x7 sized filters, 2 channels, and 2 filters
filters = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters[:, 3, :, 0] = 1  # vertical line
filters[3, :, :, 1] = 1  # horizontal line
```

```
# channel 0, filter 0 – a vertical line
temp = filters[:, :, 0, 0]
```

temp - NumPy array

	0	1	2	3	4	5	6
0	0.000	0.000	0.000	1.000	0.000	0.000	0.000
1	0.000	0.000	0.000	1.000	0.000	0.000	0.000
2	0.000	0.000	0.000	1.000	0.000	0.000	0.000
3	0.000	0.000	0.000	1.000	0.000	0.000	0.000
4	0.000	0.000	0.000	1.000	0.000	0.000	0.000
5	0.000	0.000	0.000	1.000	0.000	0.000	0.000
6	0.000	0.000	0.000	1.000	0.000	0.000	0.000

Creating the convolutional layer

```
# Create a graph with one convolutional layer applying the 2 filters
tf.reset_default_graph()
X = tf.placeholder(tf.float32, shape=(None, height, width, channels))
convolution = tf.nn.conv2d(X, filters, strides=[1,2,2,1],
                           padding="SAME")
```

Strides:

- The two middle elements are the vertical and horizontal strides (s_h, s_w)
- The other elements must currently be 1

Padding:

- "SAME" means use zero padding as needed
- "VALID" means no padding

Putting the pieces back together

```
# Load sample images
china = load_sample_image("china.jpg")
flower = load_sample_image("flower.jpg")
dataset = np.array([china, flower], dtype=np.float32)
batch_size, height, width, channels = dataset.shape

# 7x7 sized filters, 2 channels, and 2 filters
filters = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters[:, 3, :, 0] = 1  # vertical line
filters[3, :, :, 1] = 1  # horizontal line

# Create a graph with one convolutional layer applying the 2 filters
tf.reset_default_graph()
X = tf.placeholder(tf.float32, shape=(None, height, width, channels))
convolution = tf.nn.conv2d(X, filters, strides=[1,2,2,1],
                           padding="SAME")

# execute
with tf.Session() as sess:
    output = sess.run(convolution, feed_dict={X: dataset})
```

Memory requirements

Consider a CNN layer with:

- 5x5 filters, 200 feature maps (not unusual)
- stride 1, SAME padding
- image size 150 x 100 (tiny), 3 channels

You get:

- 15,200 neurons, 225 million floating point ops
- 11.4 MB for one training instance
- 1 GB for 100 instance mini-batch

During training you need memory for all layers

Pooling

Basically: aggregate over receptive field

As with convolution, you must specify:

- size of the rectangle
- stride
- padding type

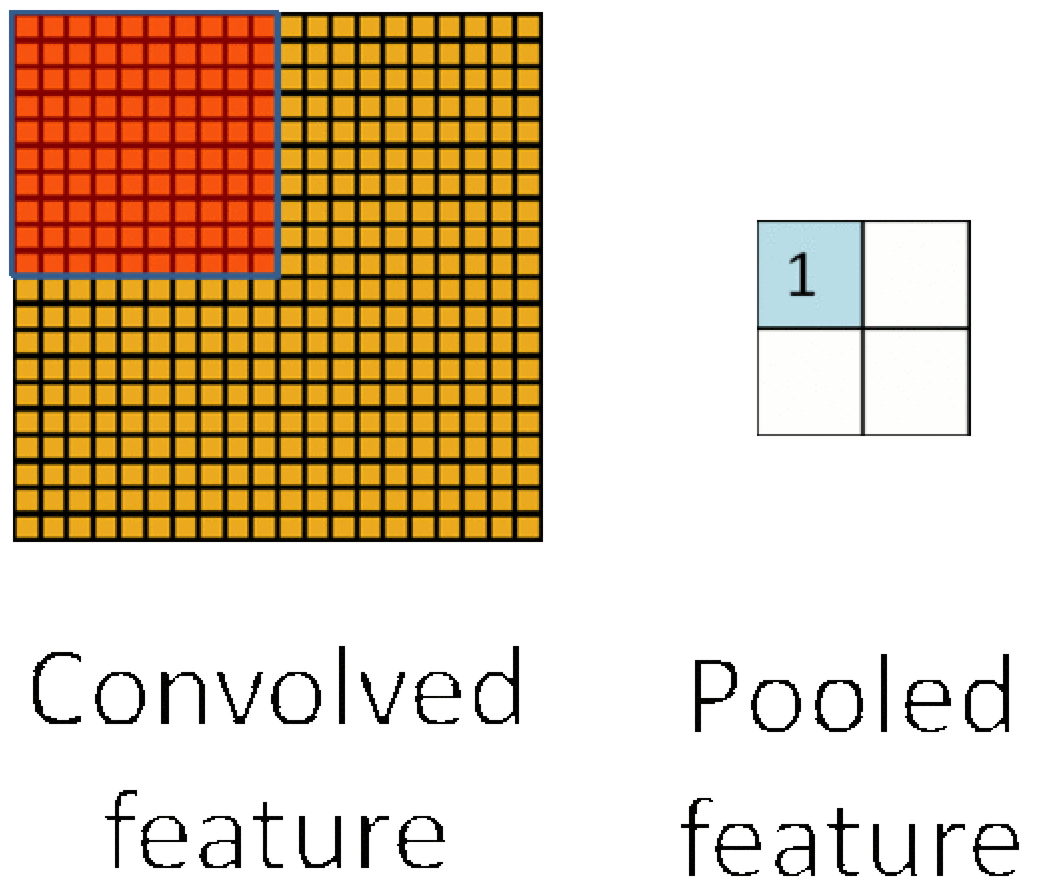
But now, instead of weights, also specify:

- aggregation function (max, mean, etc.)

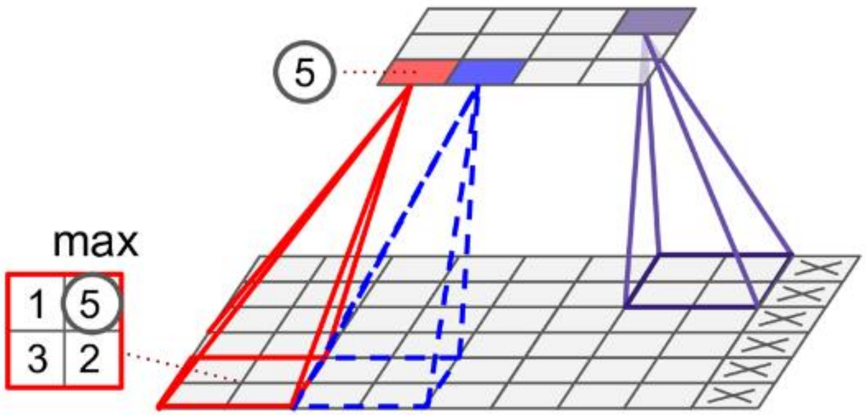
Reduces computational load, memory usage, and number of parameters

Question: how is overfitting affected?

Pooling



Max pooling example



Notes:
- pooling is usually done on each channel independently
- but, pooling can also be done across channels

Pooling in TensorFlow

```
# Create a graph with just one max pooling layer
tf.reset_default_graph()
X = tf.placeholder(tf.float32, shape=(None, height, width, channels))
max_pool = tf.nn.max_pool(X, ksize=[1,2,2,1], strides=[1,2,2,1],
                          padding="VALID")
```

ksize:

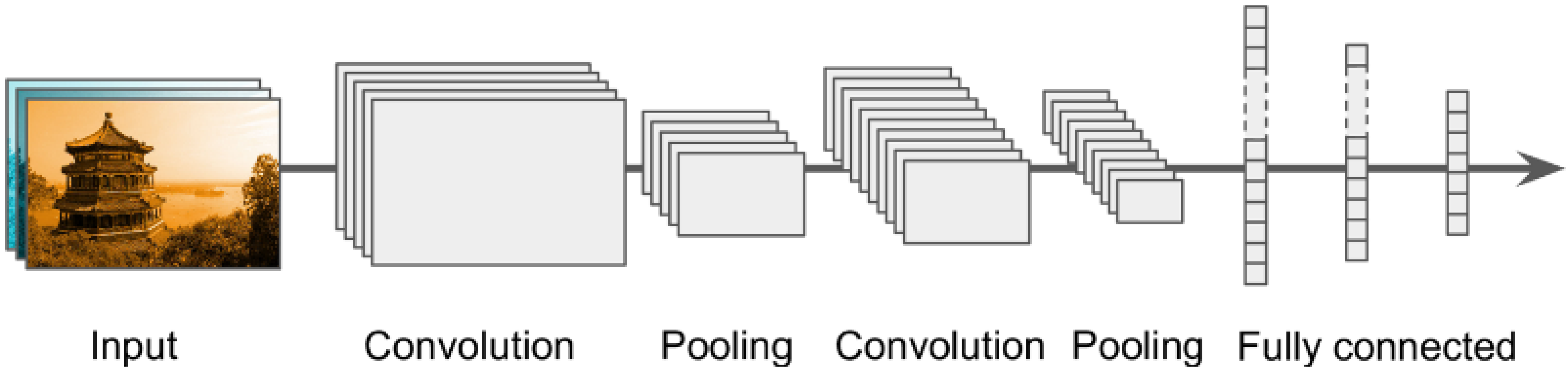
- kernel shape along the four dimensions of X:

  [instance, height, width, channels]

- no pooling over instances, so first value must be 1
- no pooling across both spatial dimensions and channels, so ksize[1]=ksize[2]=1, or ksize[3] = 1.

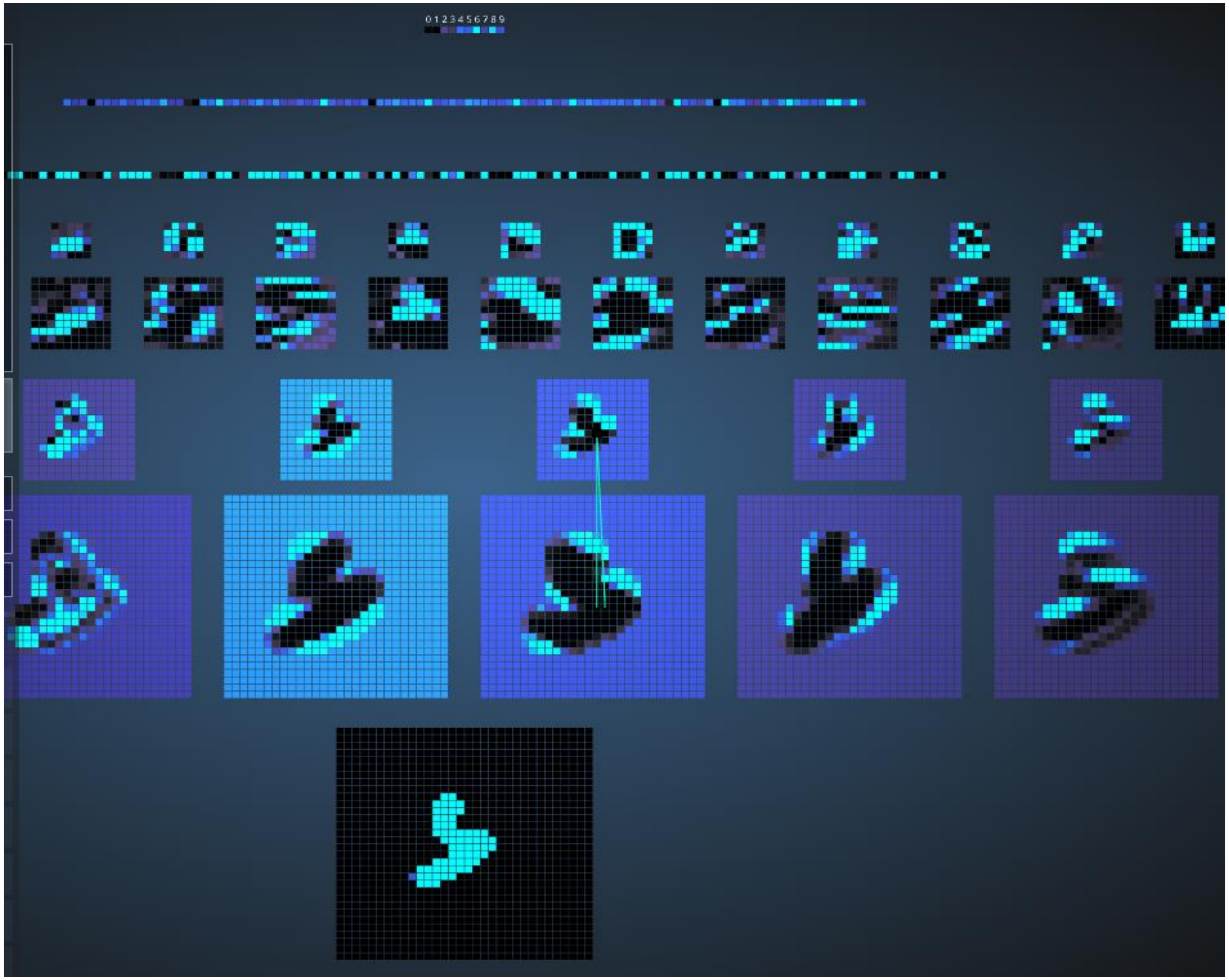
CNN architecture

How are convolution and pooling combined in common CNNs?



- The convolutional layers usually have ReLU activation
- Image gets smaller but also deeper (more feature maps) as it goes through network
- Final layer outputs prediction (e.g. softmax layer)

Online CNN demo



Example: LeNet-5

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

Yann LeCun is Director of AI Research, Facebook

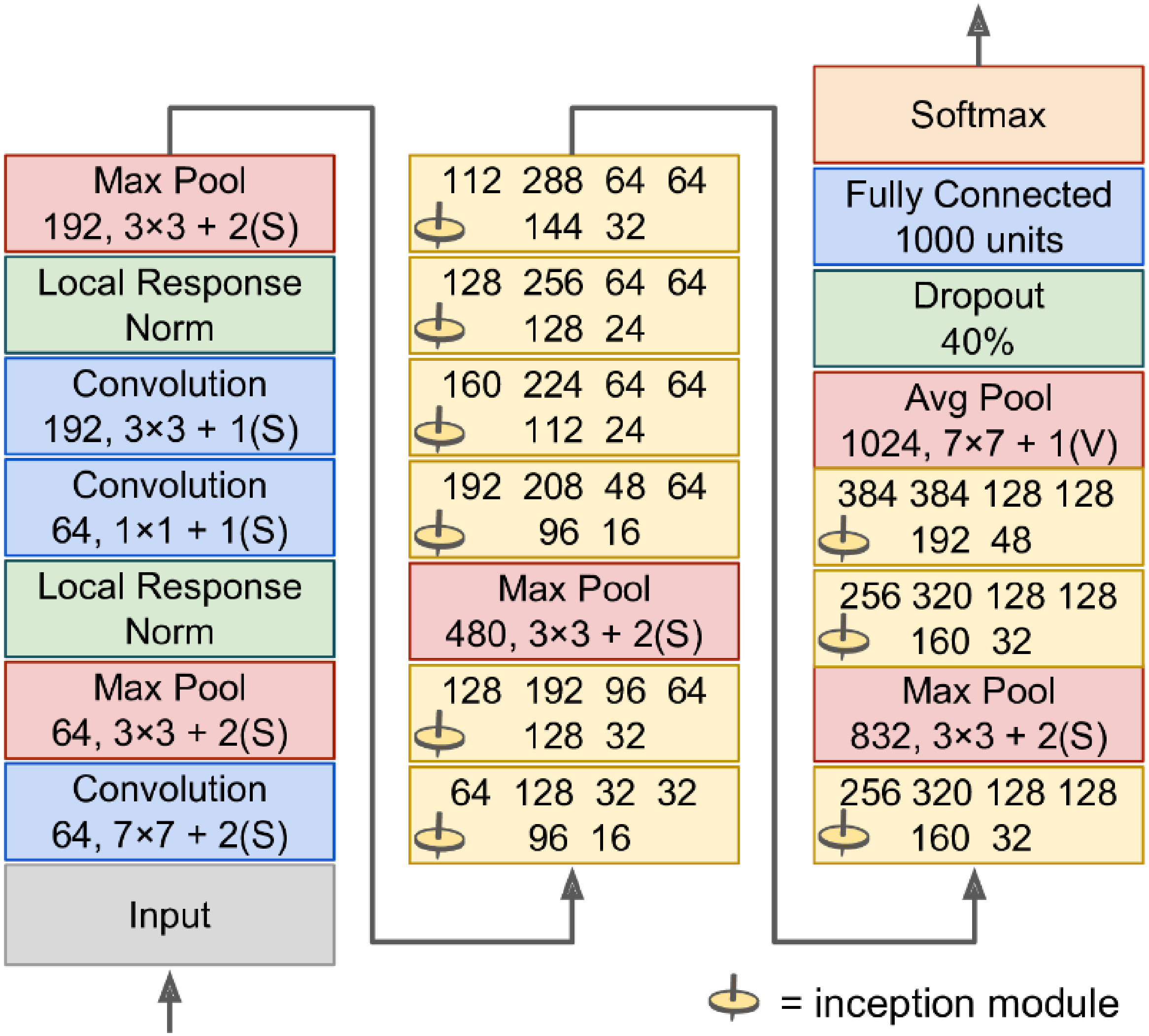
- very famous; created by Yann LeCun in 1998
- widely used for MNIS
- output layer is unusual

Example: AlexNet

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

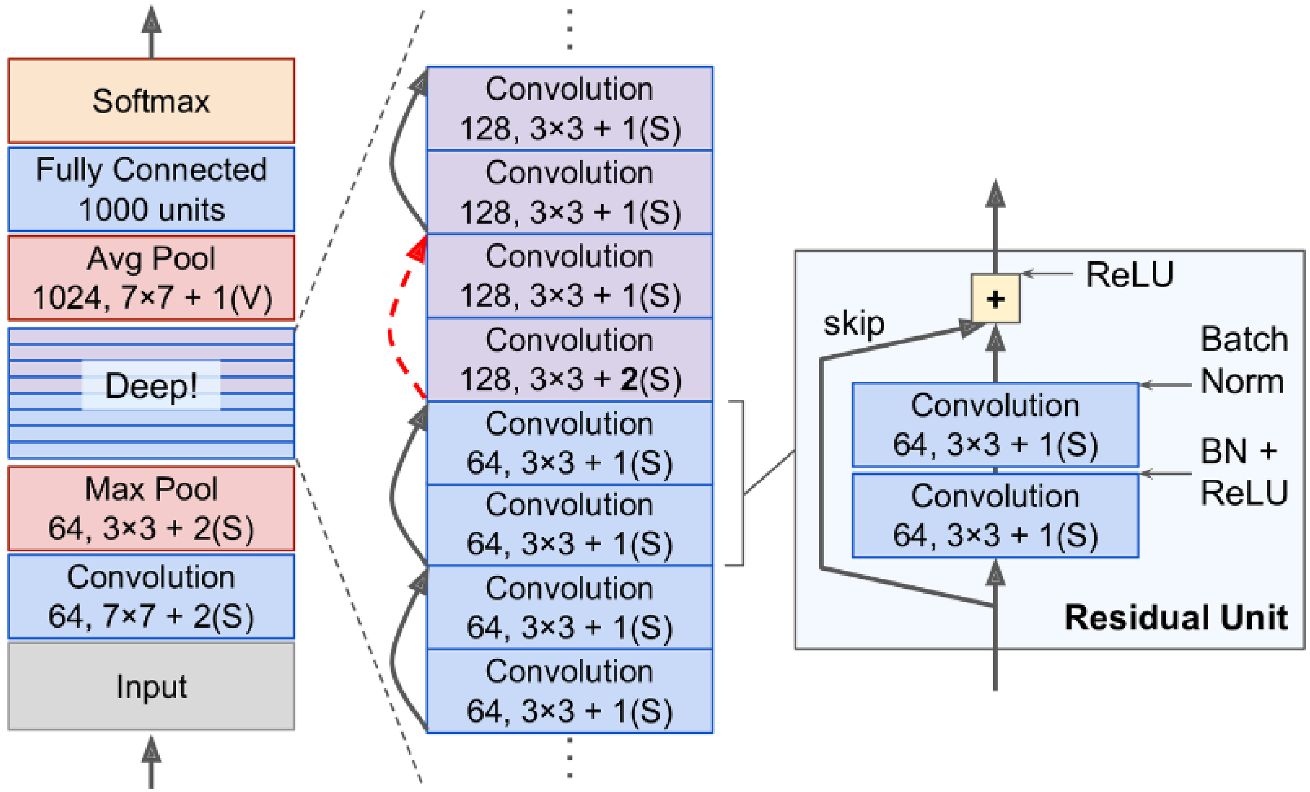
- Krizhevsky et al, 2012
- like LeNet, but larger and deeper
- first to stack convolutional layers on top of each other
- won the 2012 ImageNet ILSVRC challenge

Example: GoogLeNet



- Szegedy et al, 2014
- much deeper than previous CNNs
- uses sub-networks called "inception modules"
- won the 2014 ILSVRC challenge

Example: ResNet



- Kaiming He et al, 2015 (won the ILSVRC 2015 challenge)
- uses "skip connections": signal feeding into a layer is added to the output of a higher level
- structure of network: starts and ends like GoogLeNet; in between is a deep stack of simple residual units

Summary

- creating convolutional layers in TF
- pooling
- pooling in TF
- typical CNN architecture
- state-of-the-art CNN architectures