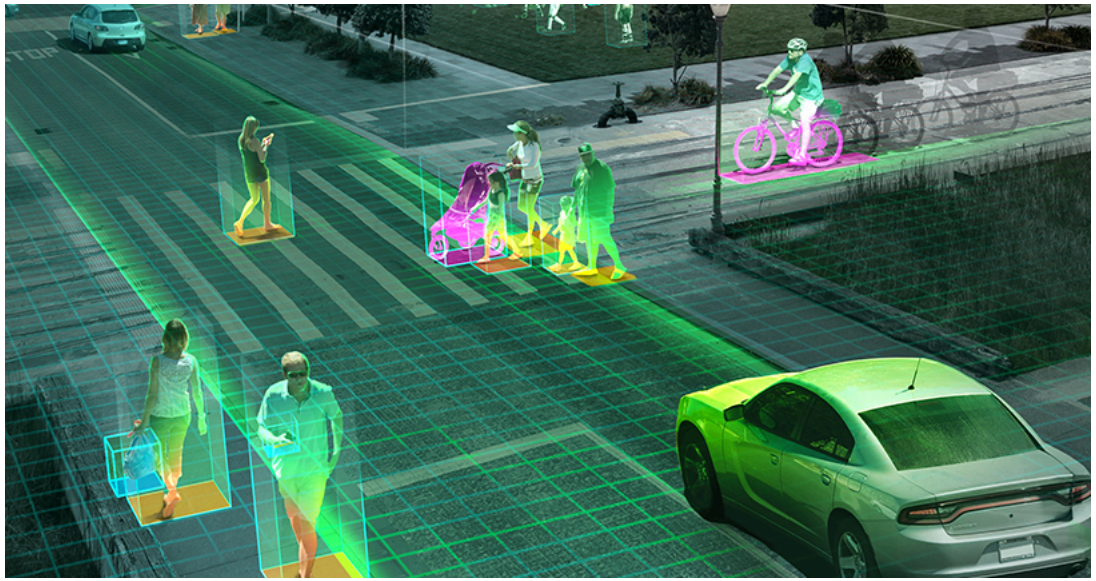


Deep Learning in Practice:
Computer Vision

Mohammad Taher Pilehvar

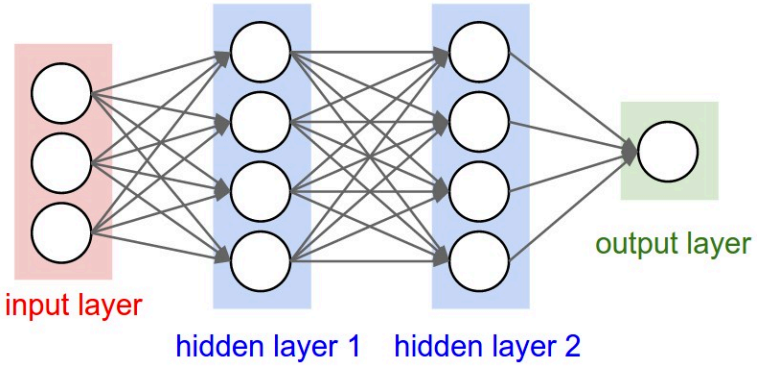
Introduction to Artificial Intelligence, 97-98

http://iust-courses.github.io/ai97/

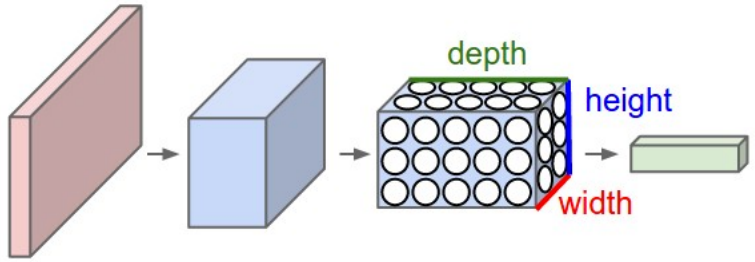
Image courtesy Nvidia

Dense for images?

- They don't scale well to large images!
  - Image size 32x32x3 (32 wide, 32 high, 3 color channels)
    - 32*32*3 = 3072 weights
    - A more realistic image:
      - 200*200*3 = 120,000 weights



- Not easy to have *channels* for them
  - Instead, ConvNets have neurons arranged in 3 dimensions: **width, height, depth**

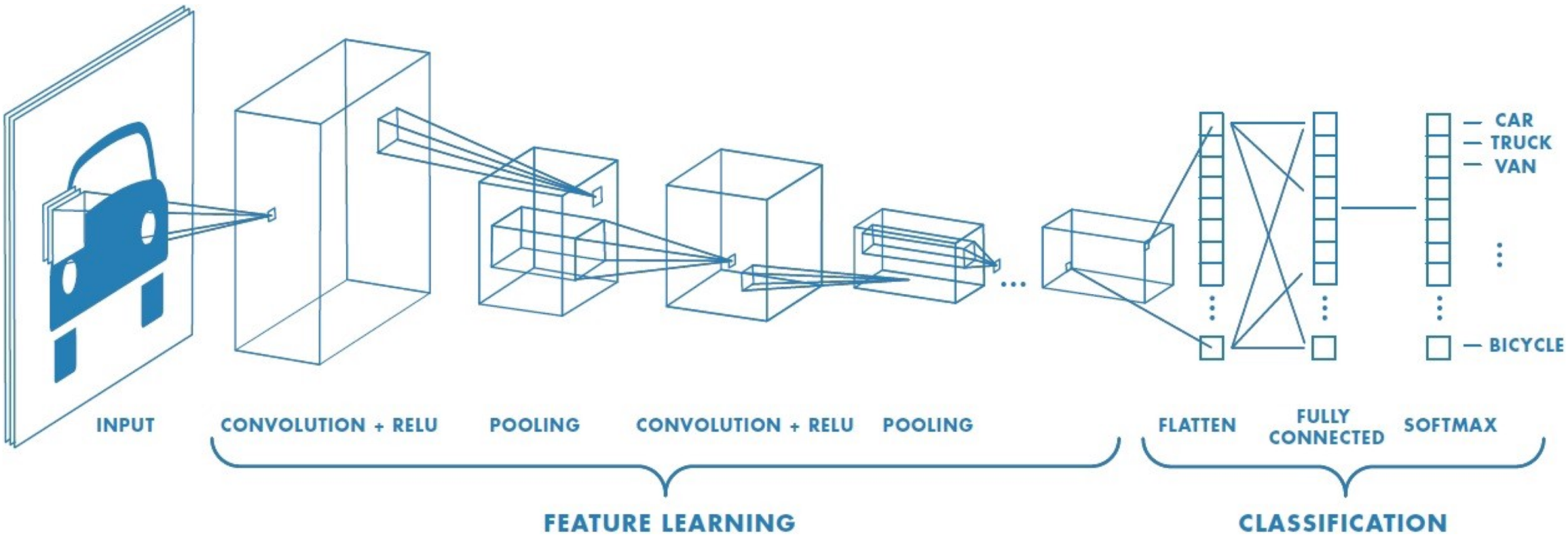


CNN: Convolutional Neural Network

- Designed specifically for images
- Still, a sequence of layers, each with many neurons and learnable weights
- Still, we have a loss function, gradient descent optimization, etc.

But: a different architecture!

CNN: Convolutional Neural Network

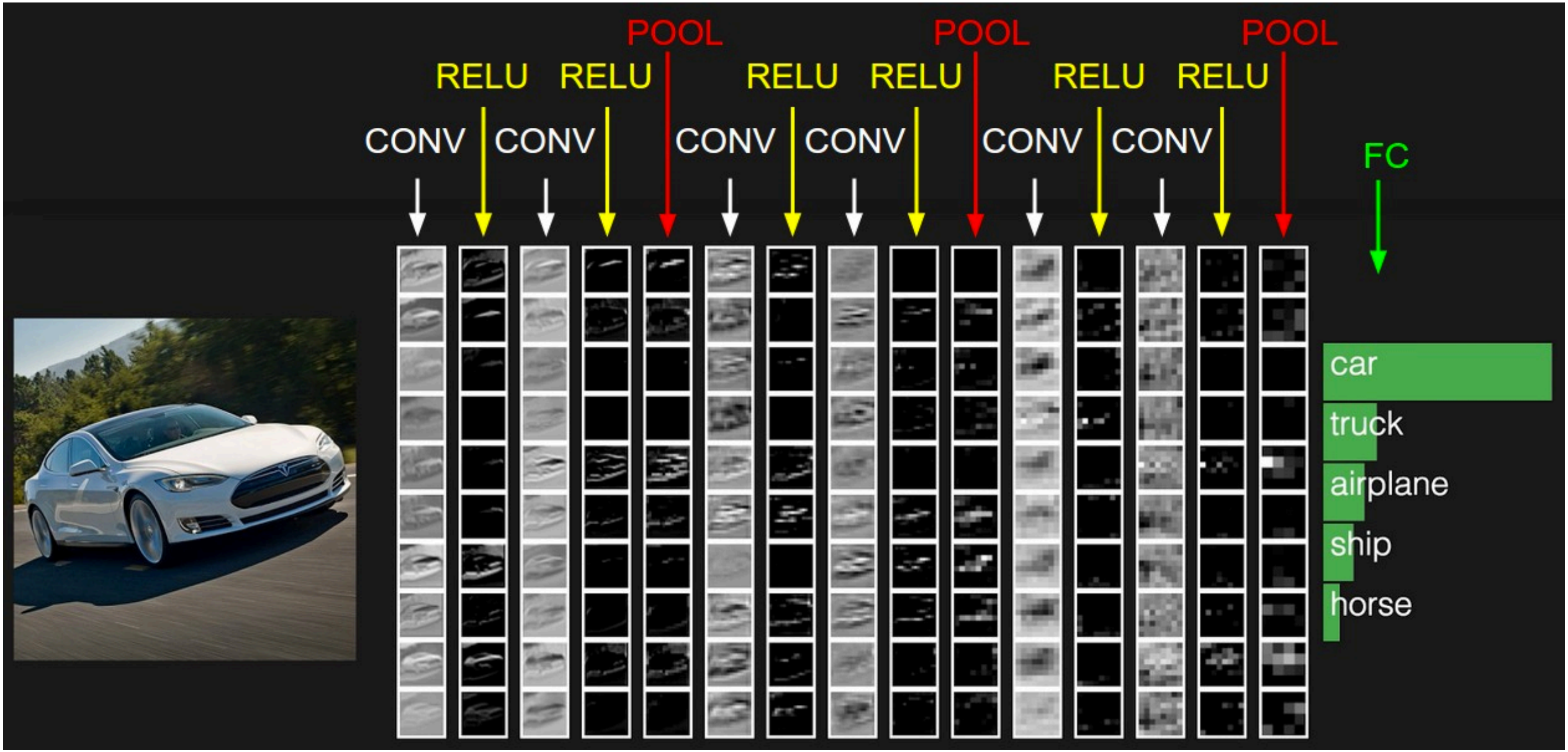


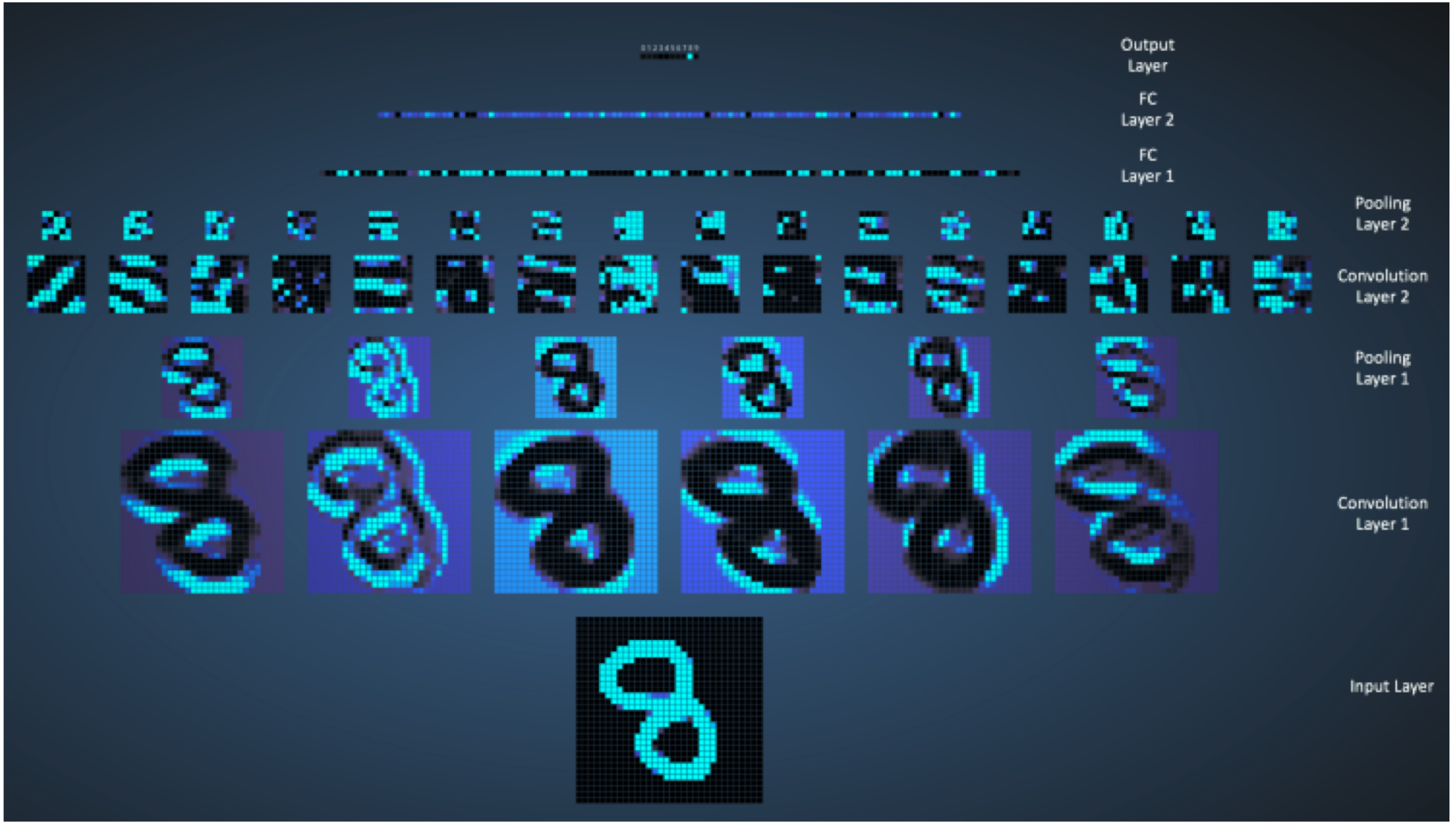
Shyamal Patel and Johanna Pingel, MathWorks

CNNs

Three types of layer:

- Convolutional layer
- Pooling layer
- Fully-connected layer


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse



http://scs.ryerson.ca/~aharley/vis/conv/flat.html

Dive into CNNs: MNIST

- A basic convnet is a stack of `Conv2D` and `MaxPooling2D` layers.

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

CNNs for MNIST

- A convnet takes takes input tensors of shape
  - `(image_height, image_width, image_channels)`
  - MNIST images: `(28, 28, 1)`

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

CNNs for MNIST: model summary

```
>>> model.summary()
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d_1 (Conv2D) (None, 26, 26, 32) 320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D) (None, 13, 13, 32) 0
_________________________________________________________________
conv2d_2 (Conv2D) (None, 11, 11, 64) 18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D) (None, 5, 5, 64) 0
_________________________________________________________________
conv2d_3 (Conv2D) (None, 3, 3, 64) 36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

CNNs for MNIST: add a classifier

The next step is to feed the last output tensor (of shape `(3, 3, 64)`) into a densely connected classifier network like those you're already familiar with: a stack of `Dense` layers.

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

CNNs for MNIST: so far

```
>>> model.summary()
Layer (type) Output Shape Param #
=================================================================
conv2d_1 (Conv2D) (None, 26, 26, 32) 320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D) (None, 13, 13, 32) 0
_________________________________________________________________
conv2d_2 (Conv2D) (None, 11, 11, 64) 18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D) (None, 5, 5, 64) 0
_________________________________________________________________
conv2d_3 (Conv2D) (None, 3, 3, 64) 36928
_________________________________________________________________
flatten_1 (Flatten) (None, 576) 0
_________________________________________________________________
dense_1 (Dense) (None, 64) 36928
_________________________________________________________________
dense_2 (Dense) (None, 10) 650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

CNNs for MNIST: train the network

```
from keras.datasets import mnist
from keras.utils import to_categorical
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

CNNs for MNIST: evaluate on test

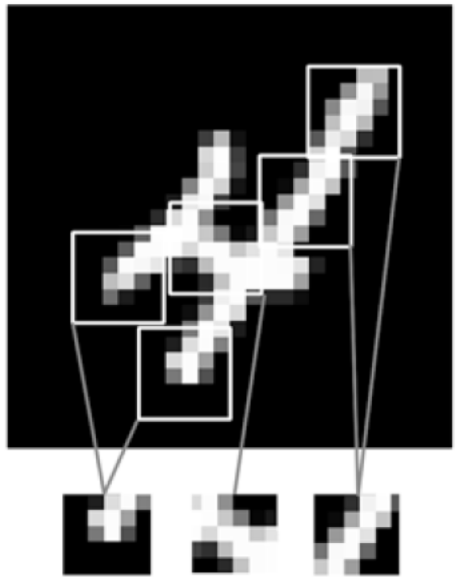
```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> test_acc
0.9908000000000001
```

Convolution operation

The fundamental difference between Dense and CNN:

- Dense layers learn global patterns in their input feature space
  - For example, for a MNIST digit, patterns involving all pixels),

- CNNs learn local patterns: in the case of images, patterns found in small 2D windows of the inputs.

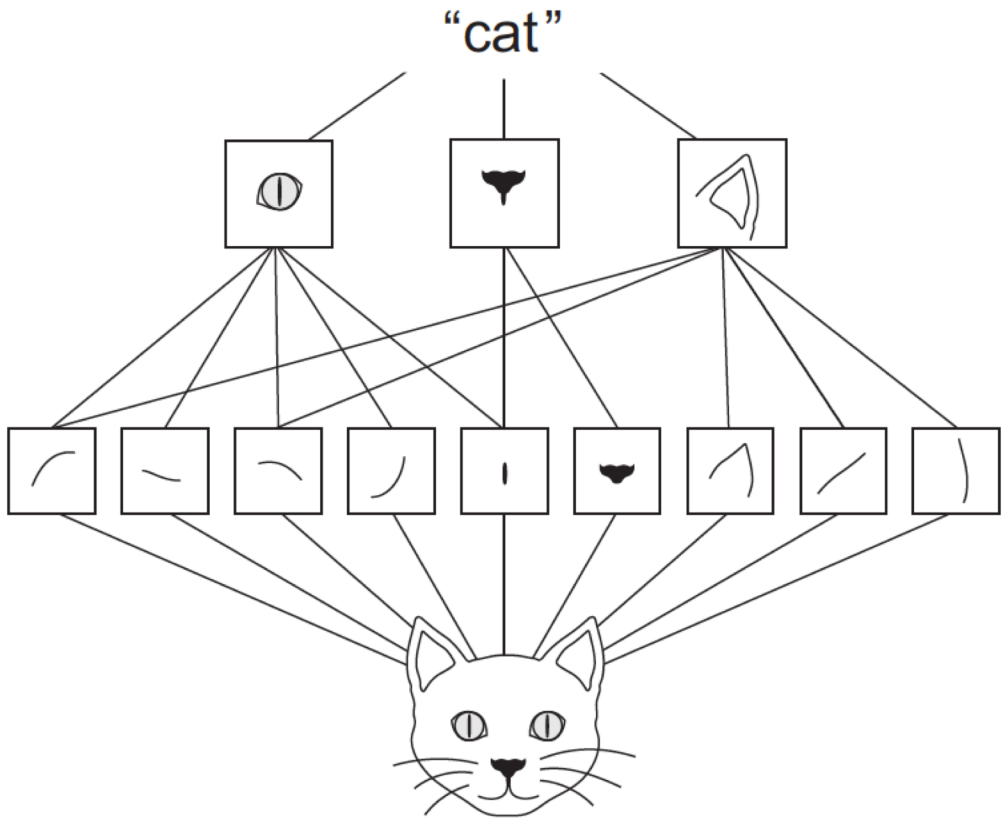


CNNS: characteristics

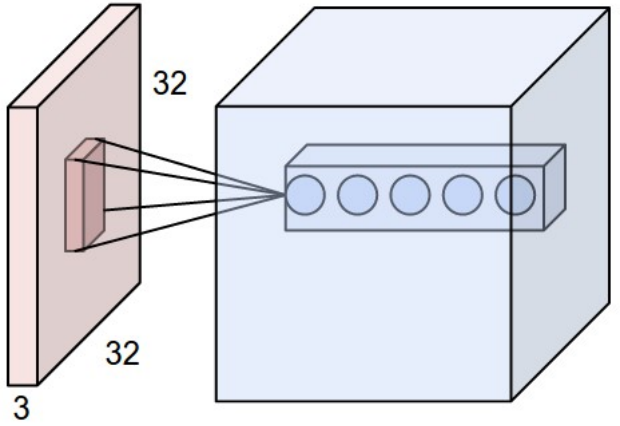
1. The patterns they learn are translation invariant
   - Unlike Dense, a convnet can recognize a pattern anywhere in the image
   - *The visual world is fundamentally translation invariant*

CNNS: characteristics

2. They can learn spatial hierarchies of patterns
   - A first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on



Example 1

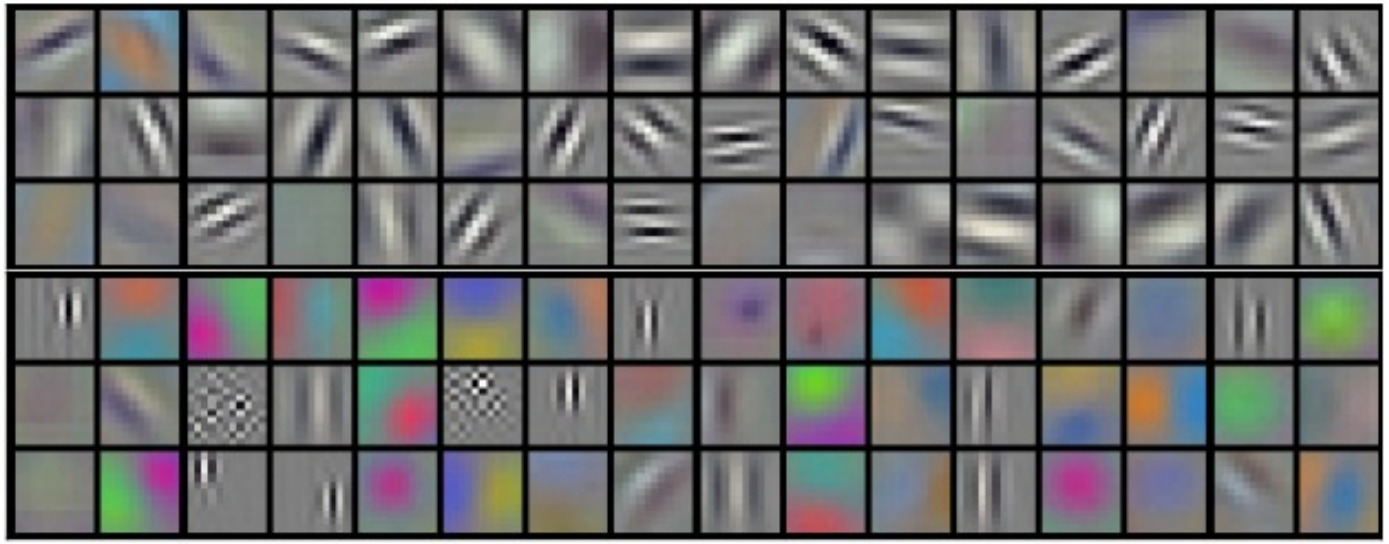


- **Depth**: a hyperparamter, the number of filters, each looking at a different feature in input, 5 in the example

- **Stride**: the step size for sliding the filter.

- **Zero-padding**: pad the input volume across borders with zeros.
  - We can control the spatial size of the output volumes

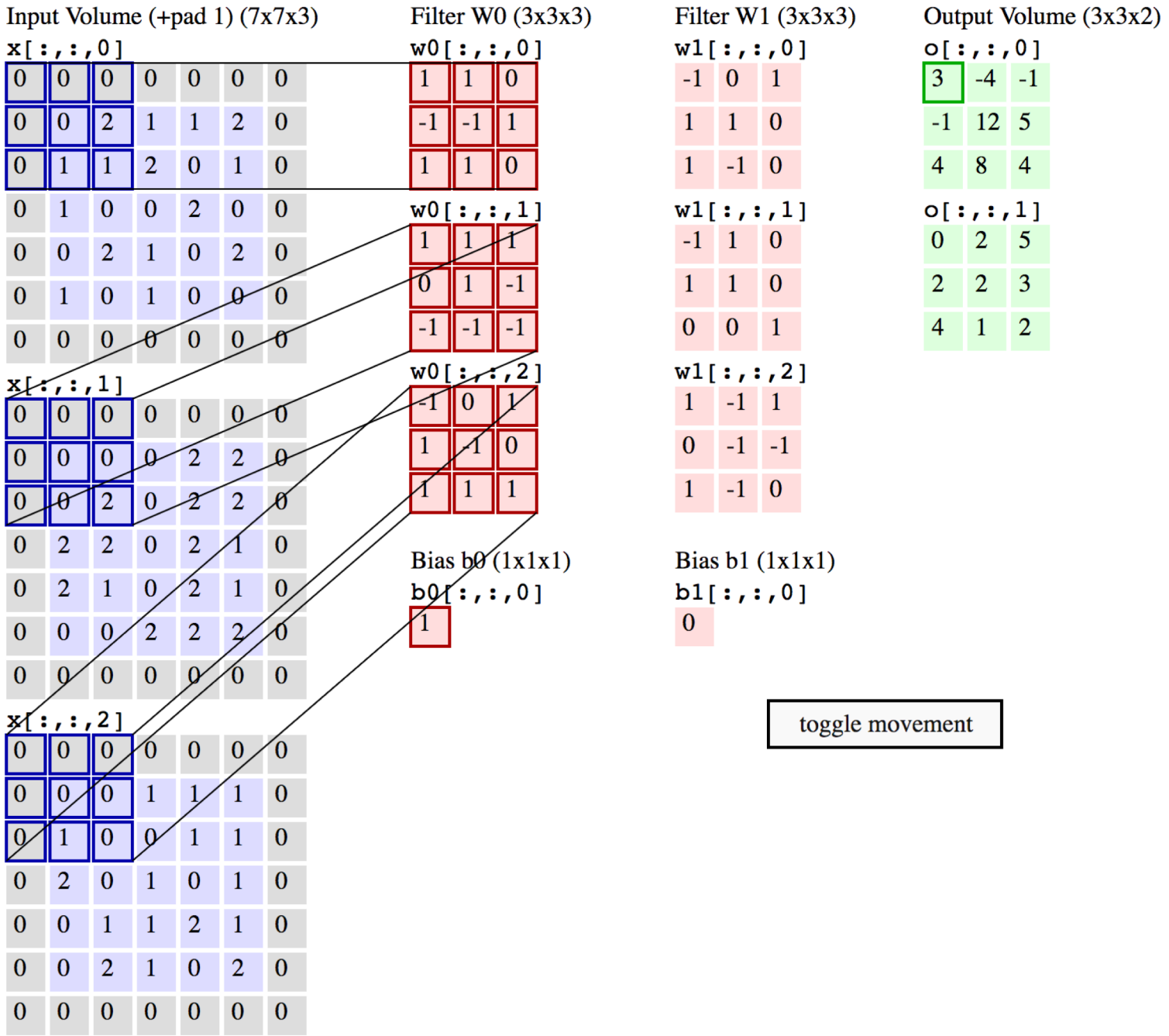
Filters



Demo

W_1=5
H_1=5
D_1=3

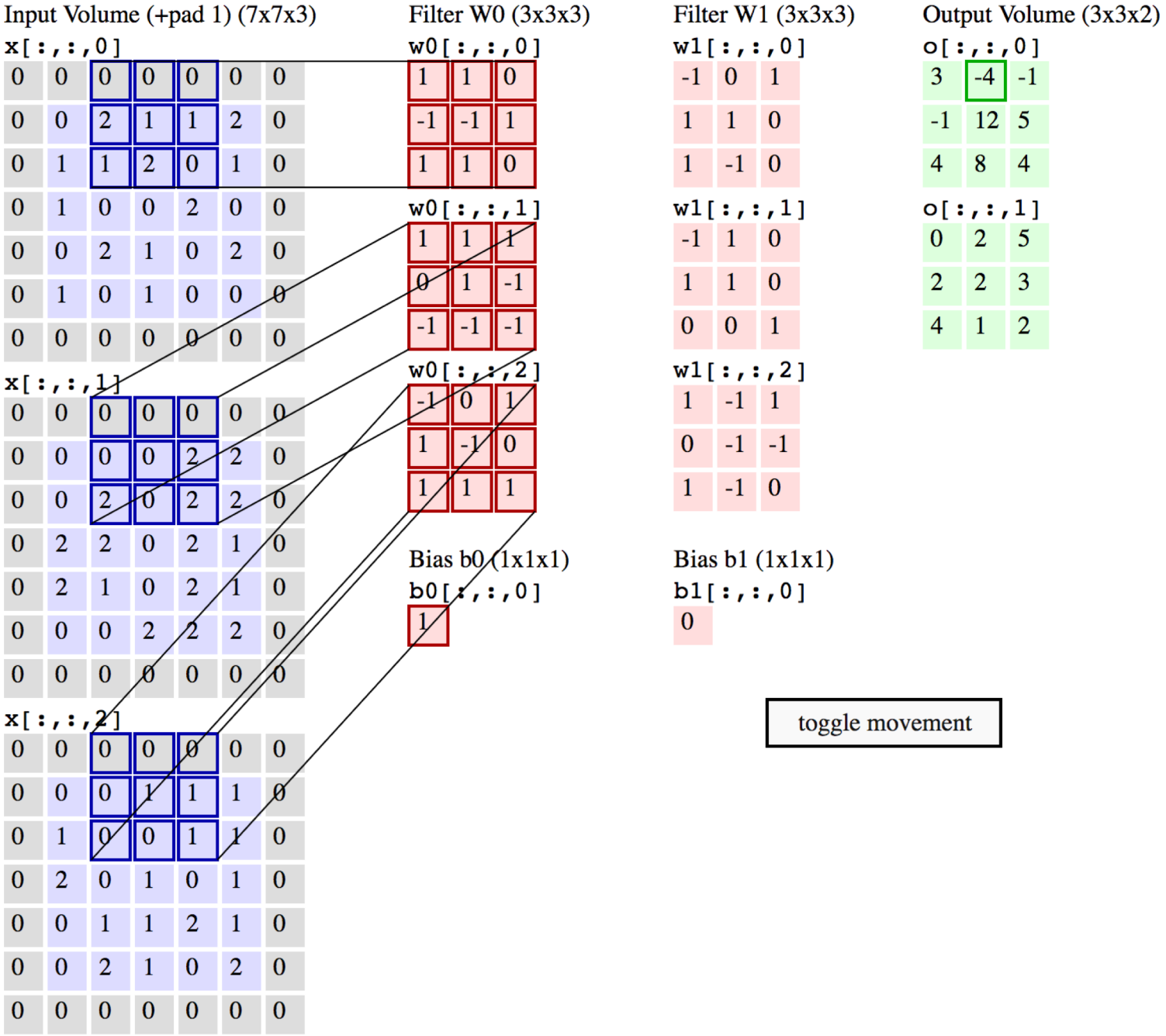
K=2
F=3
S=2
P=1



Demo

W_1=5
H_1=5
D_1=3

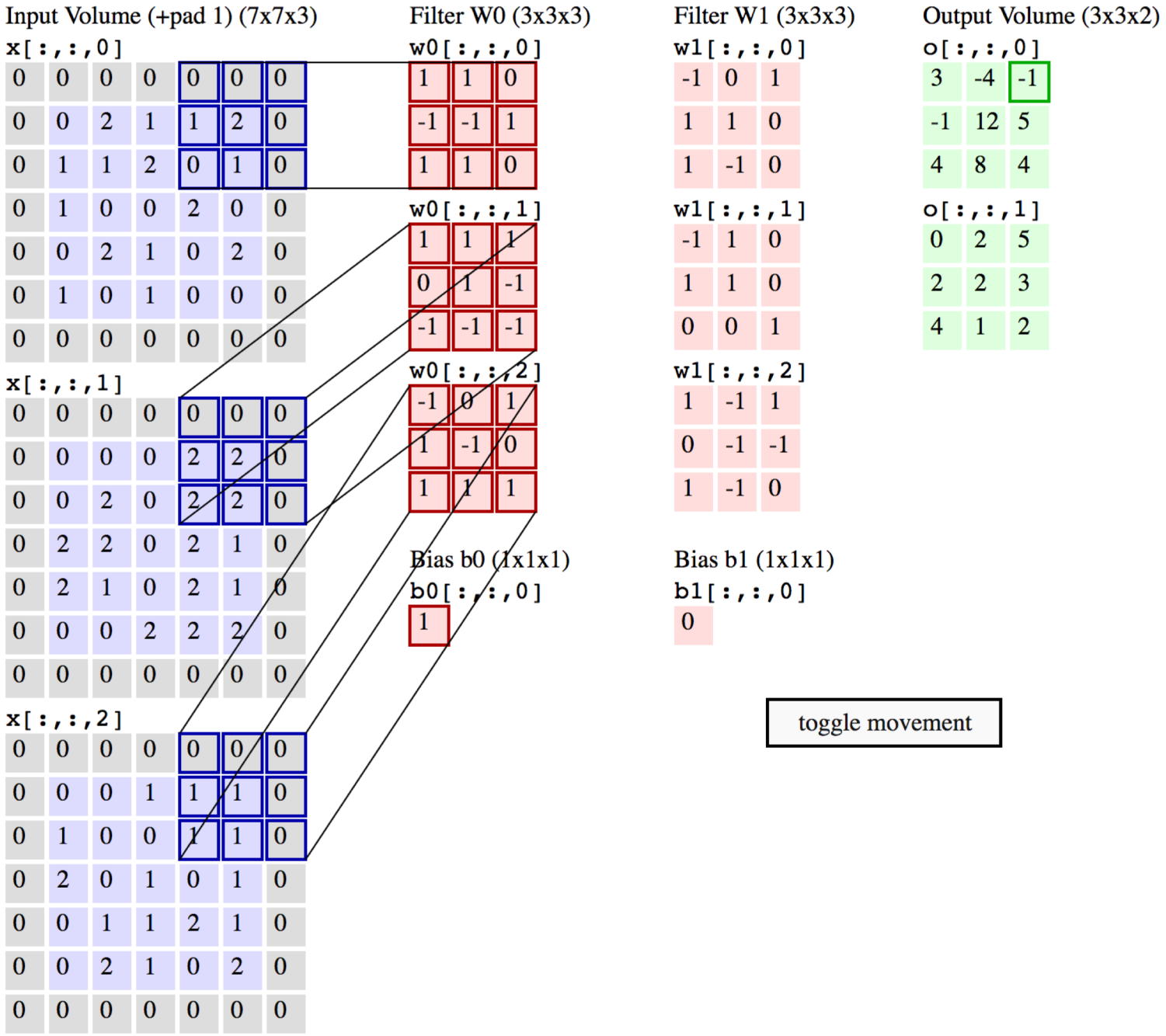
K=2
F=3
S=2
P=1



Demo

W_1=5
H_1=5
D_1=3

K=2
F=3
S=2
P=1

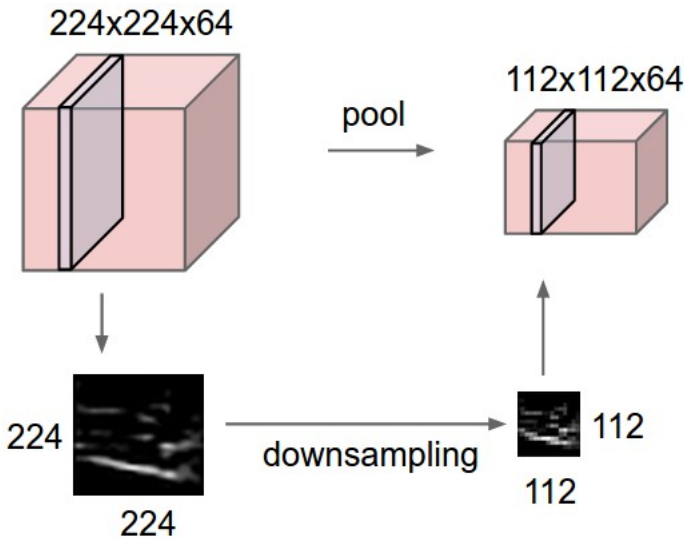


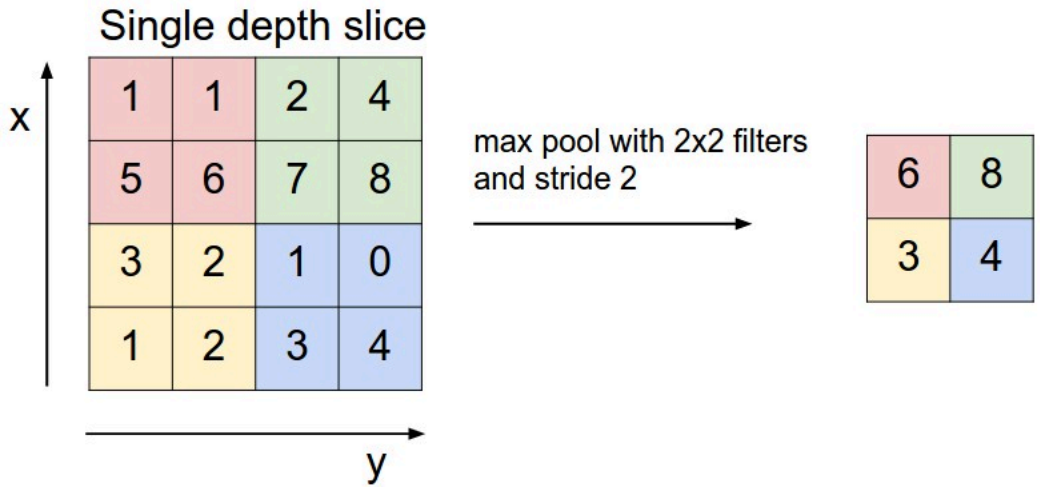
Pooling layer

- Inserted in between Conv layers
- Aim: reduce the spatial size of the representation
  - Hence, reduce the amount of parameters
    - Hence, to control overfitting

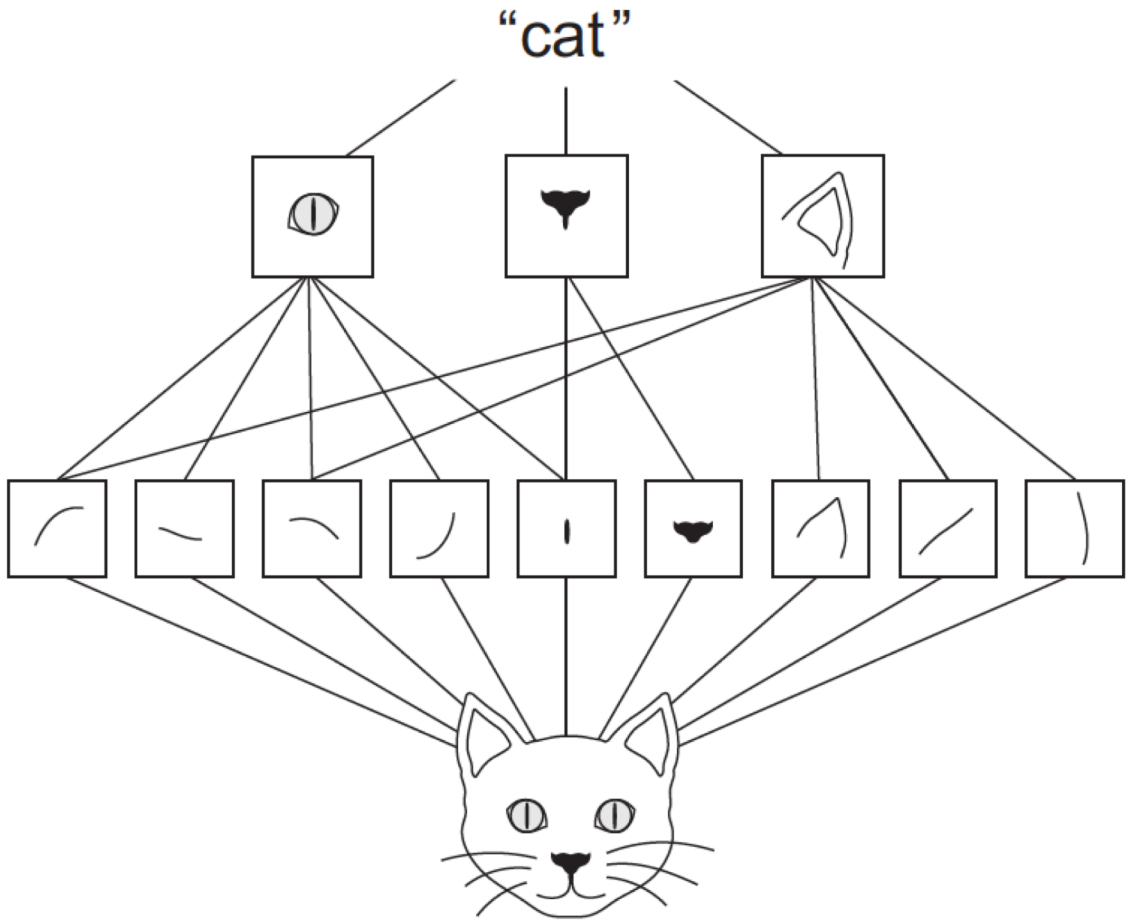
- Usually, filters of 2x2, applied with a stride of 2, downsamples every depth in both height and weight, discarding 75% of the activations
  - MAXing over four numbers

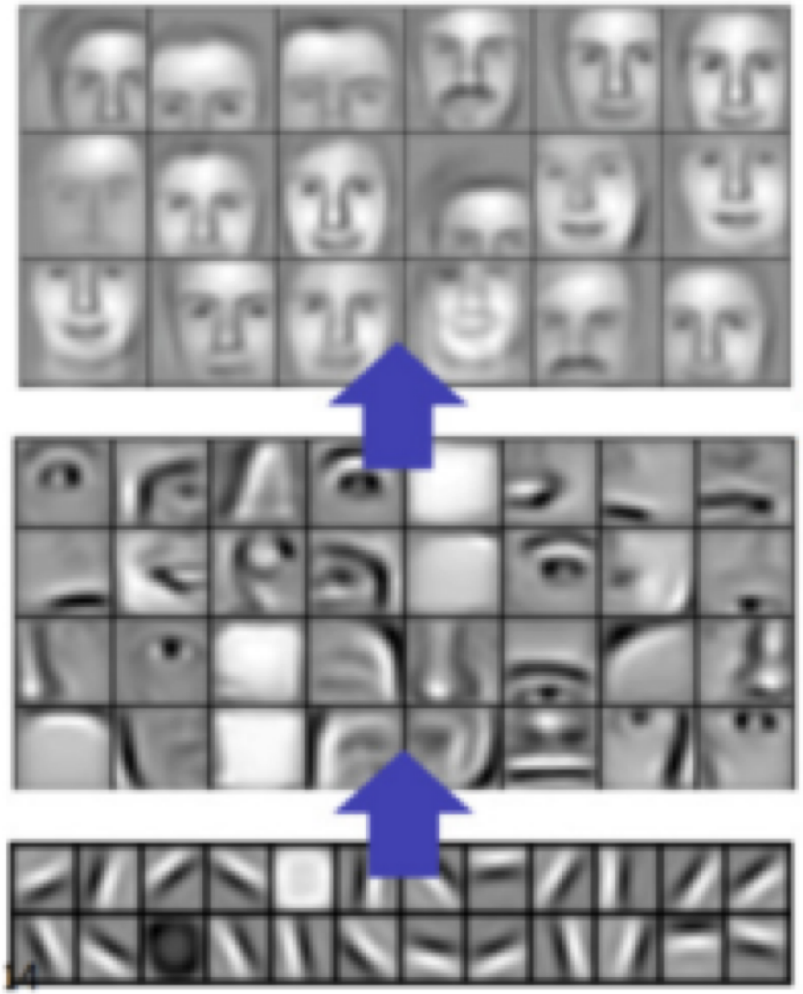
Pooling layer





Why pooling?

- 

- 

Questions?