Deep learning for Images

ImageNet Project

ImageNet project: large visual database for recognition research

Ongoing research effort to provide easily accessible image database

Over 10 million images

Hand-annotated on what objects are pictured

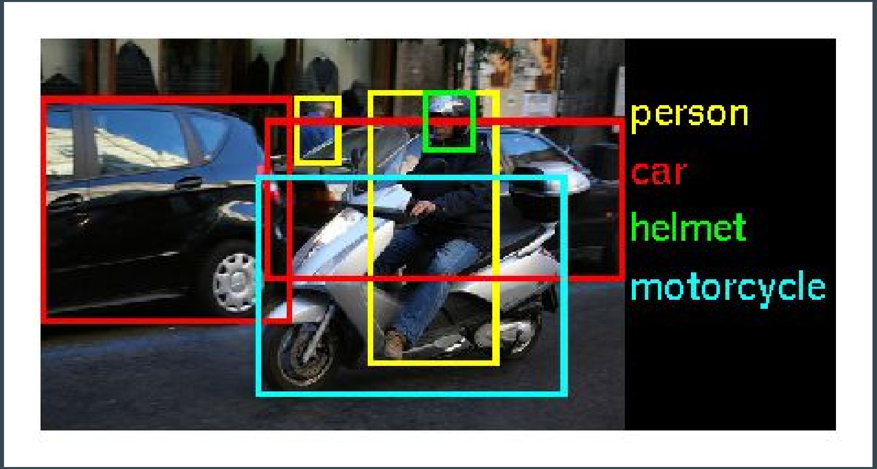
ImageNet Project

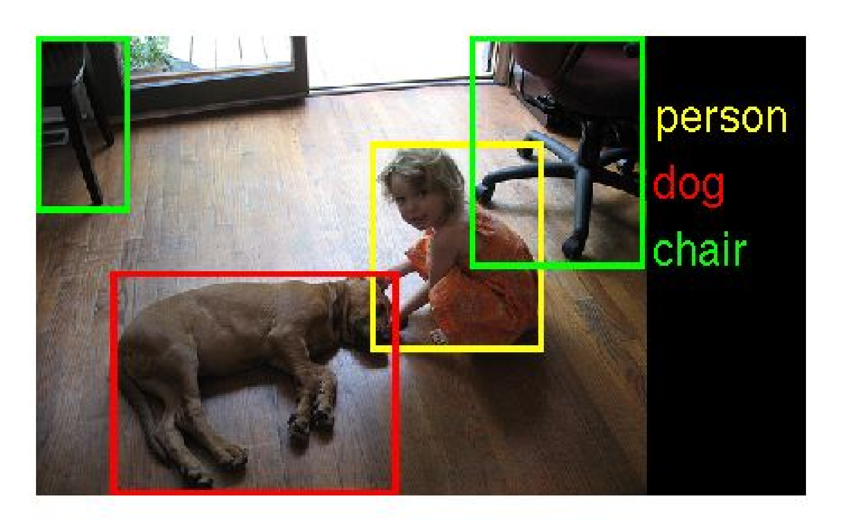
ImageNet Categories: synsets

police van, police wagon, paddy wagon, patrol wagon, wagon, black Maria

Komodo dragon, Komodo lizard, dragon lizard, giant lizard, Varanus komodoensis

ImageNet Categories:





ILSVRC

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

since 2010, annual software contest

Software programs compete to classify and detect objects and scenes

Model performance:

2011: error rate 25%

2012: CNN AlexNet 16%, runner up 26%

2015: error rate a few %, exceeds human ability at object recognition 1000 class

Convolutional Neural Nets (CNN)

Basic Units:

Input Layer

Convolutional layer

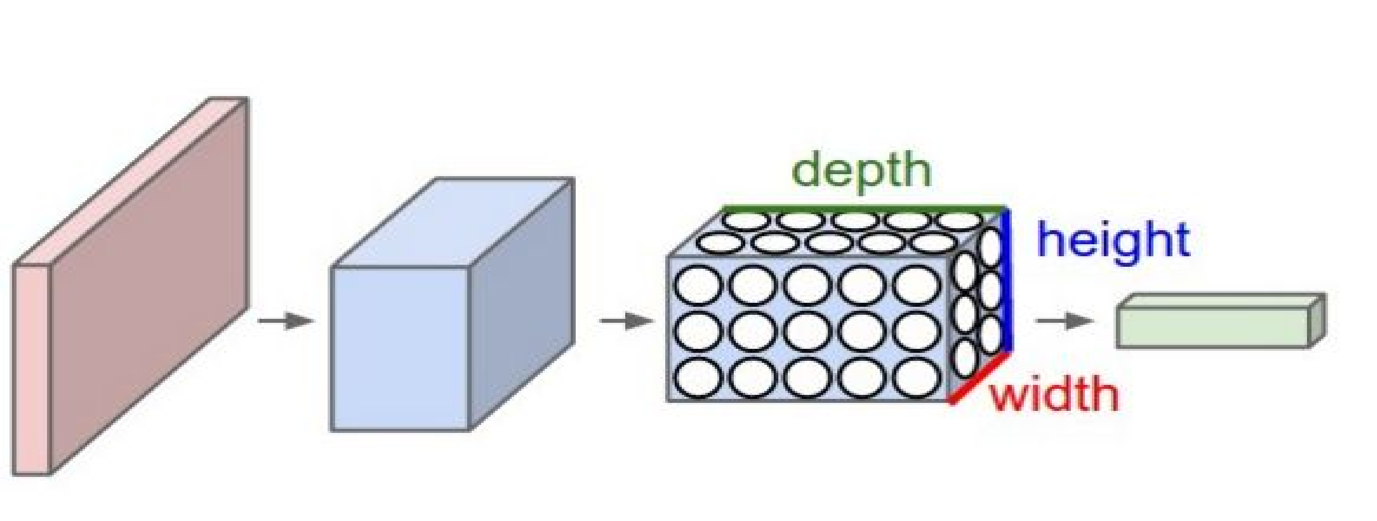
Relu layer

Pooling layer

FC layer

Input Layer

Input [224 x 224 x 3] : will hold the raw pixel values of the image, in this case an image of width 224, height 224, and with three color channels R,G,B.



Conv Layer

CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.

This will result in an output volume of [224 x 224 x 64] if we decided to use 64 filters, with each filter’s receptive field size being 3 x 3, using a stride of 1, and no zero-paddings.

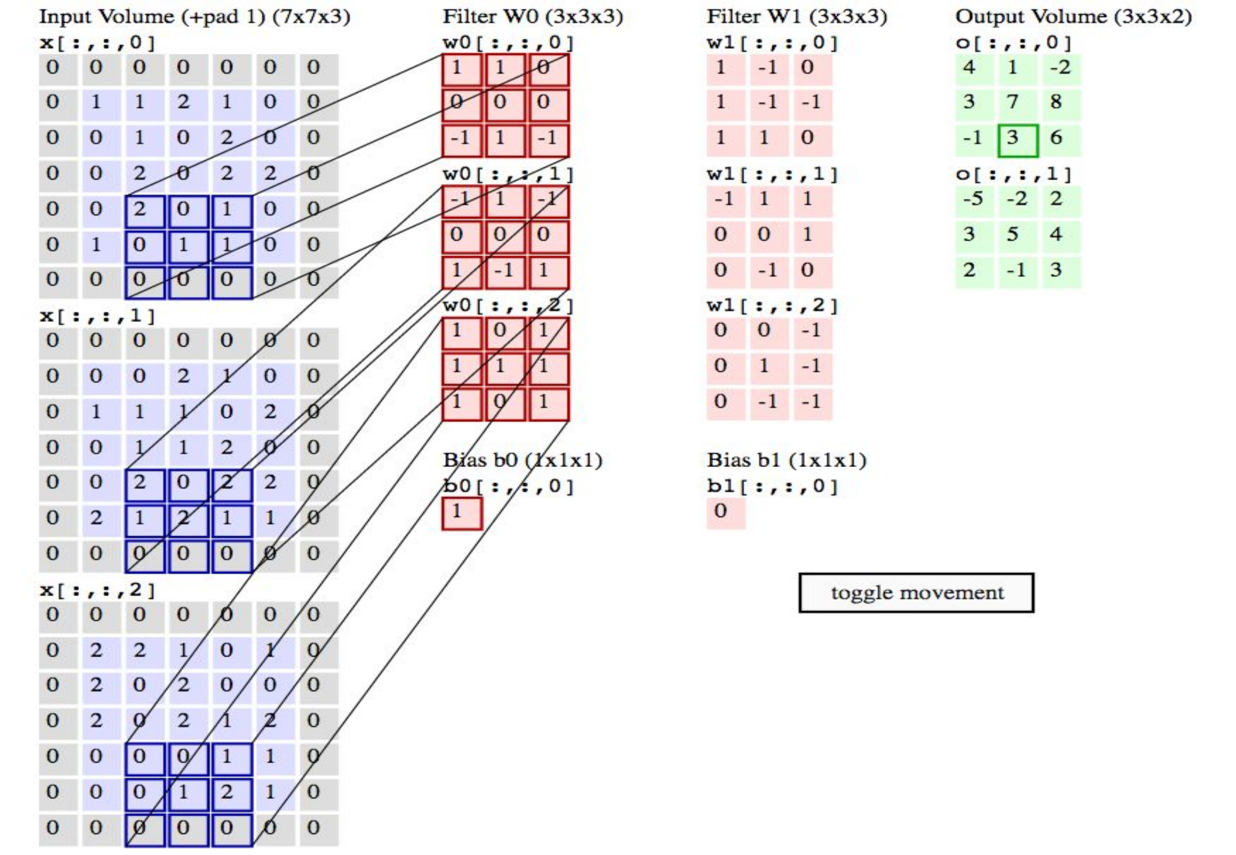
Conv Layer

CONV layer spatial arrangement: 3 hyperparameters control the size of output

- W: Input volume size
- F: Receptive Field size of the conv layer neurons
- S: Stride
- P: zero-padding on the border

Output size: $(W - F + 2P) / S + 1$

Conv Layer



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	2	1	0	0
0	0	1	0	2	0	0
0	0	2	0	2	2	0
0	0	2	0	1	0	0
0	1	0	1	1	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	0	2	1	0	0
0	1	1	1	0	2	0
0	0	1	1	2	0	0
0	0	2	0	2	2	0
0	2	1	2	1	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	1	0	1	0
0	2	0	2	0	0	0
0	2	0	2	1	2	0
0	0	0	0	1	1	0
0	0	0	1	2	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	0
0	0	0
-1	1	-1

w0[:,:,1]

-1	1	-1
0	0	0
1	-1	1

w0[:,:,2]

1	0	1
1	1	1
1	0	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

1	-1	0
1	-1	-1
1	1	0

w1[:,:,1]

-1	1	1
0	0	1
0	-1	0

w1[:,:,2]

0	0	-1
0	1	-1
0	-1	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

toggle movement

Output Volume (3x3x2)

o[:,:,0]

4	1	-2
3	7	8
-1	3	6

o[:,:,1]

-5	-2	2
3	5	4
2	-1	3

W = 5, P = 1

F = 3, S = 2

Output size

= (5 - 3 + 2*1) / 2 + 1

= 3

Output dimension:

3 x 3 x 2

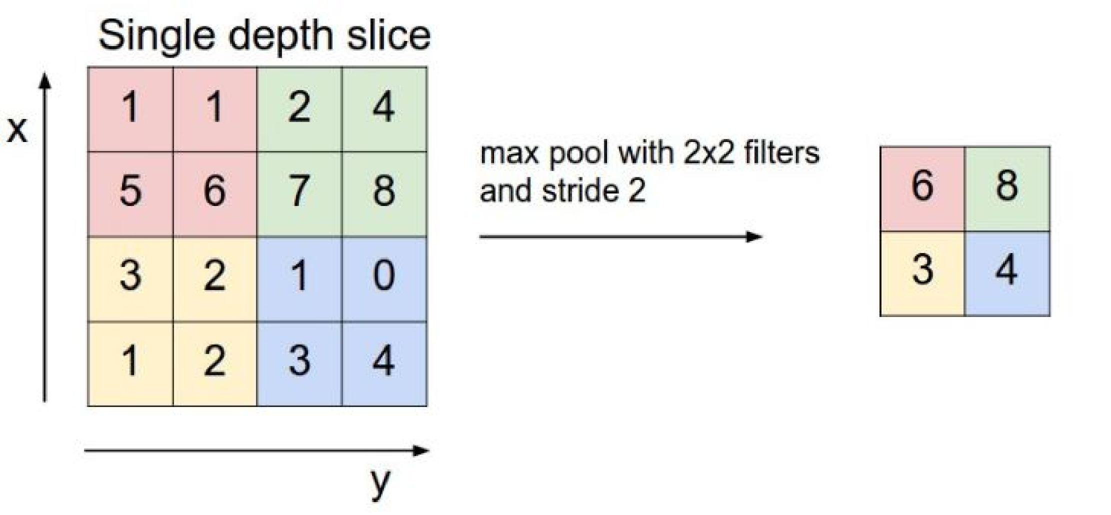
Relu Layer

Rectified linear unit layer will apply an elementwise activation function, such as the max(0, x) thresholding at zero.

This leaves the size of the output volume unchanged ([224 x 224 x 64]).

Max Pooling

POOL layer will perform a downsampling operation along the spatial dimensions (width, height). If we use a max pooling of 2 x 2 filters with a stride of 2, the resulting output volume is [112 x 112 x 64].



Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2

6	8
3	4

Fully Connected Layer

FC (i.e. fully-connected) layer: each neuron will be connected to all activations of the previous volume.

The output layer will compute class scores, resulting in volume size [1 x 1 x 1000]

ConvNets architecture summary

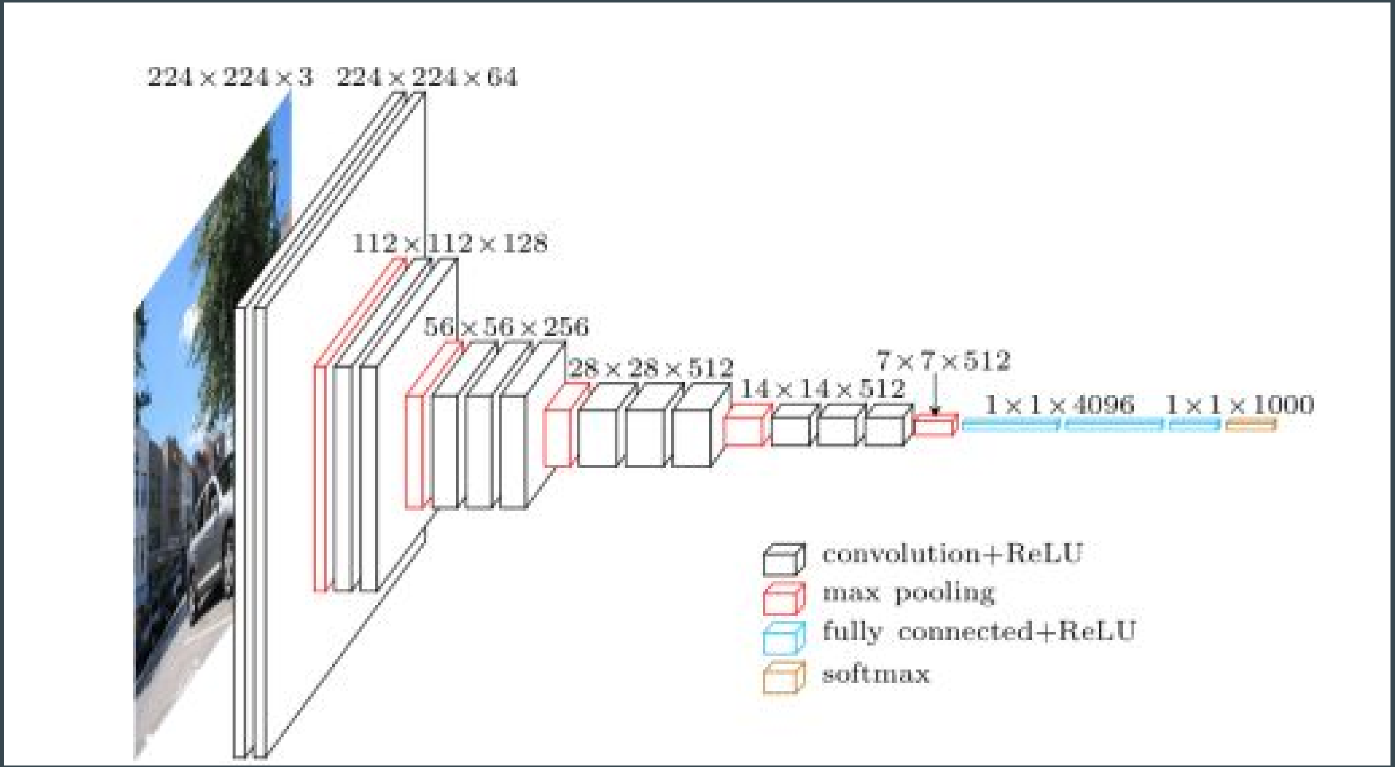
ConvNets transform the original image layer by layer from the original pixel values to the final class scores

Conv/FC layers perform transformations: weights and biases

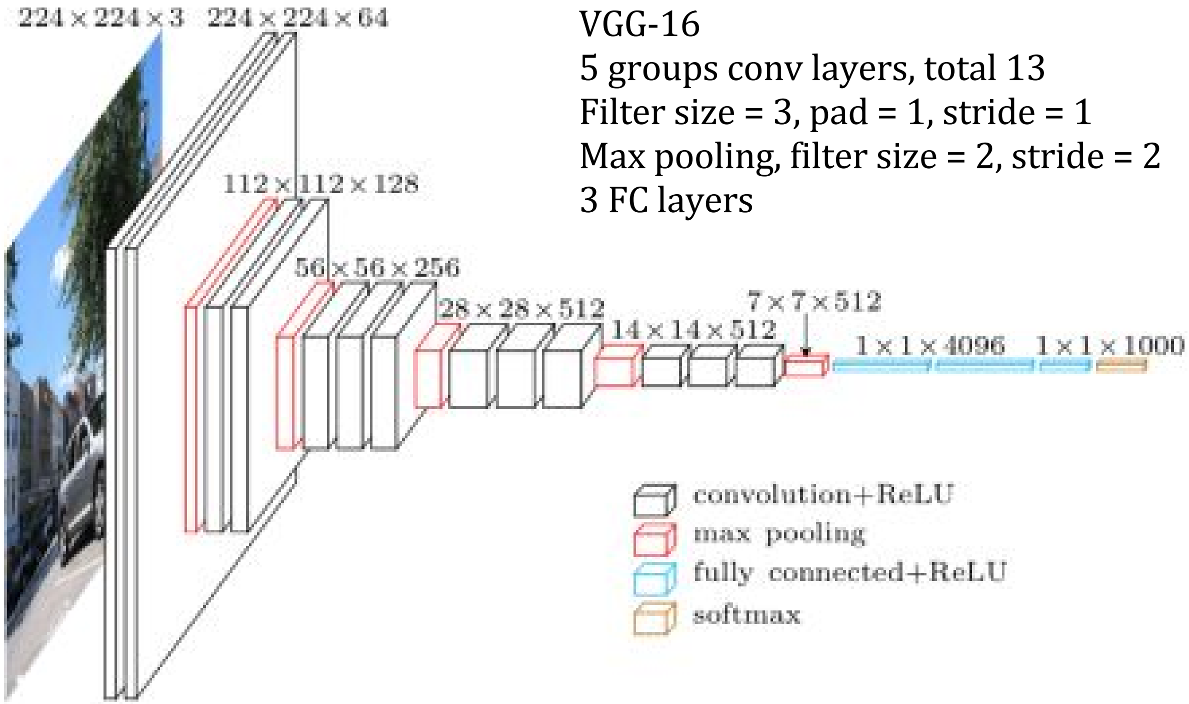
RELU/POOL layers will implement a fixed function

CONV/FC/POOL layers have additional hyperparameters (F, P, S), RELU don’t

ConvNet Example in Keras !

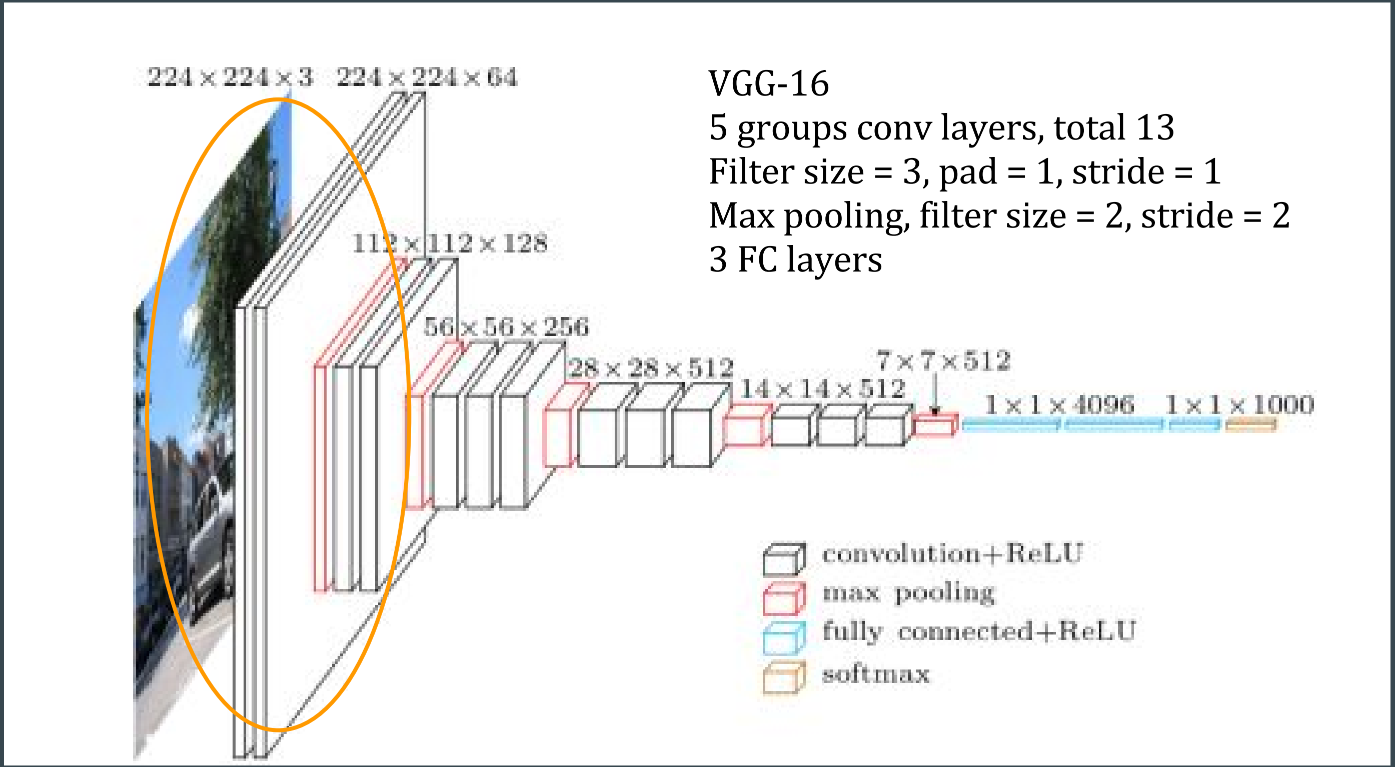


ConvNet Example in Keras !



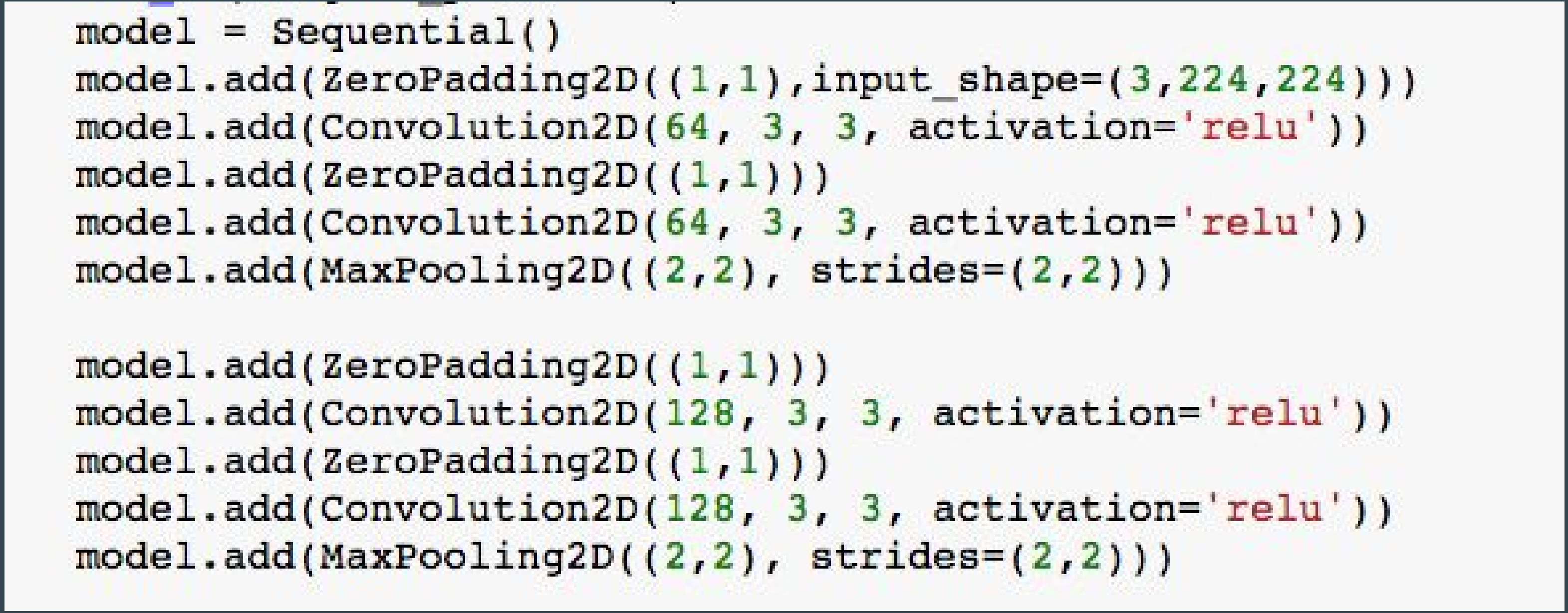
VGG-16
5 groups conv layers, total 13
Filter size = 3, pad = 1, stride = 1
Max pooling, filter size = 2, stride = 2
3 FC layers

ConvNet Example in Keras !



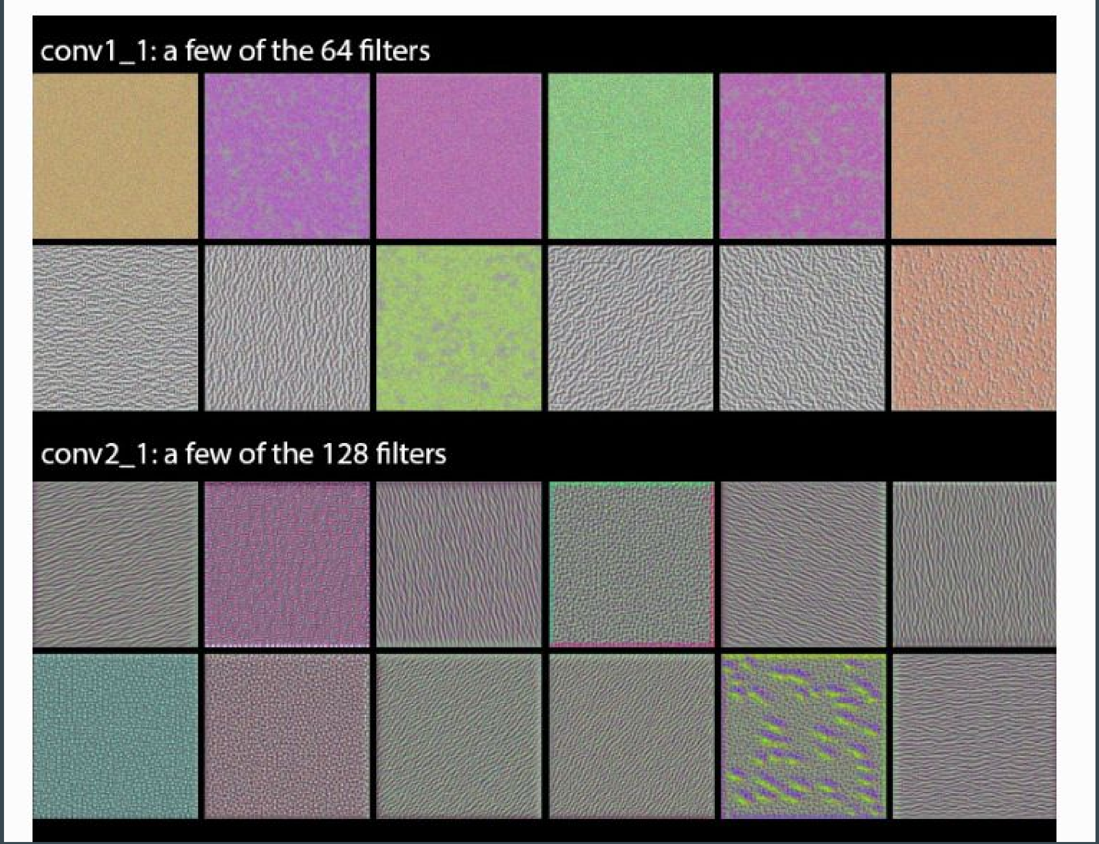
VGG-16
5 groups conv layers, total 13
Filter size = 3, pad = 1, stride = 1
Max pooling, filter size = 2, stride = 2
3 FC layers

ConvNet Example in Keras !

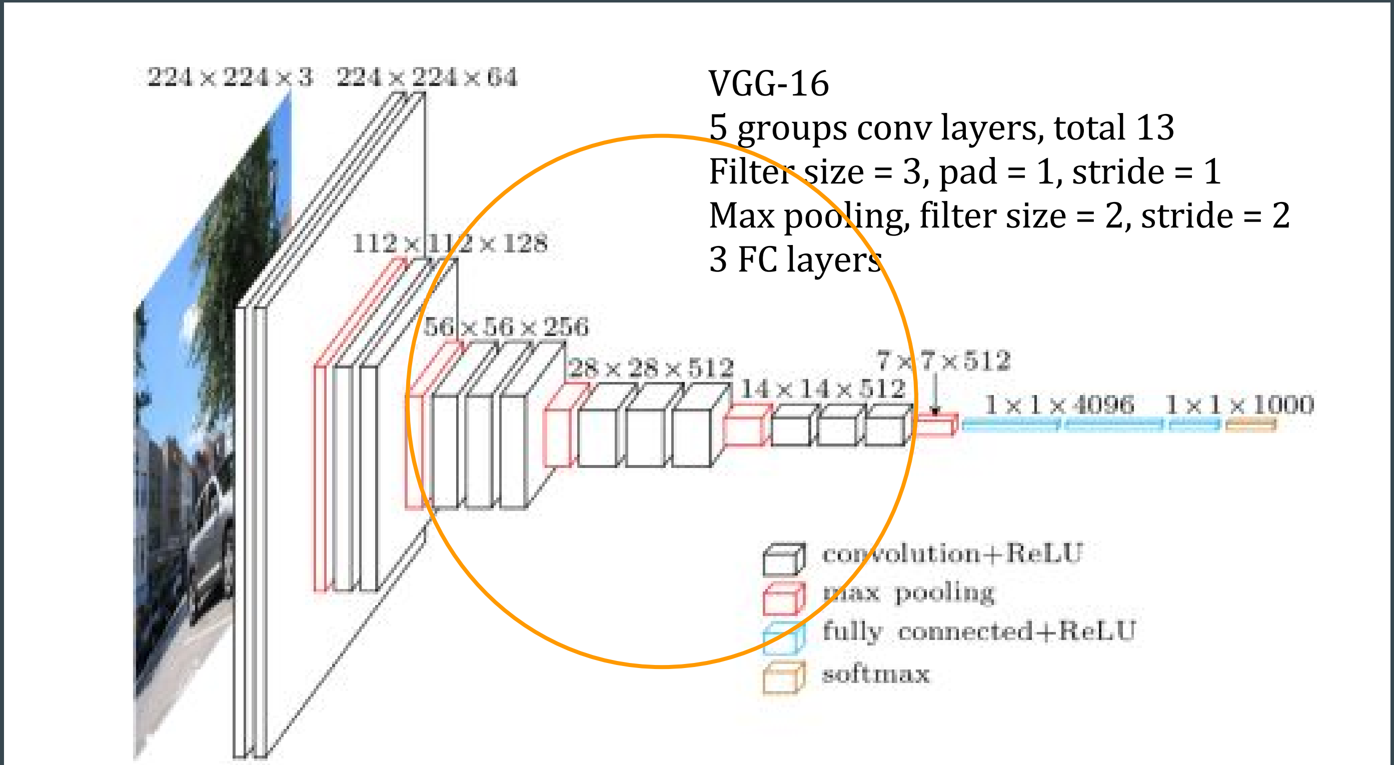


```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(3,224,224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
```

ConvNet Example in Keras !

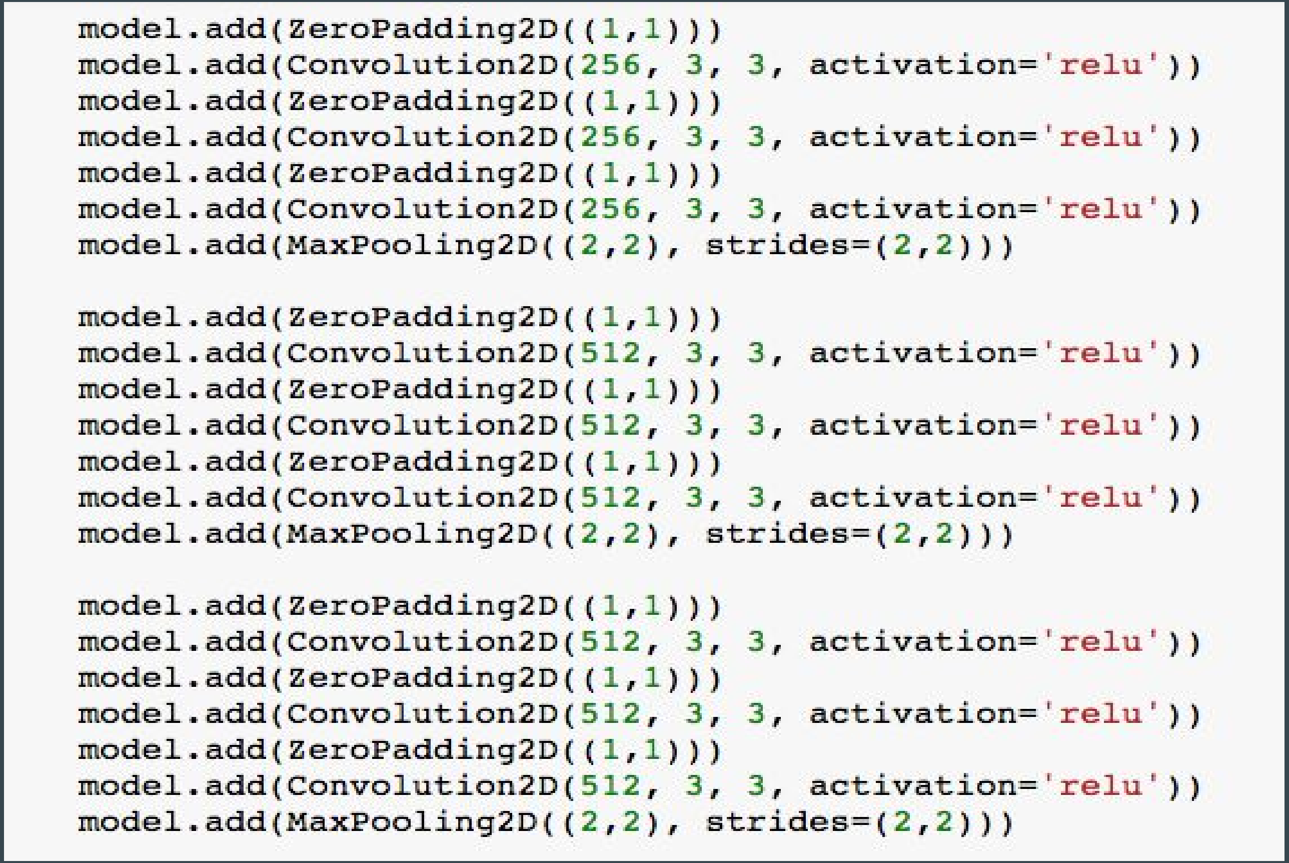

conv1_1: a few of the 64 filters
conv2_1: a few of the 128 filters

ConvNet Example in Keras !



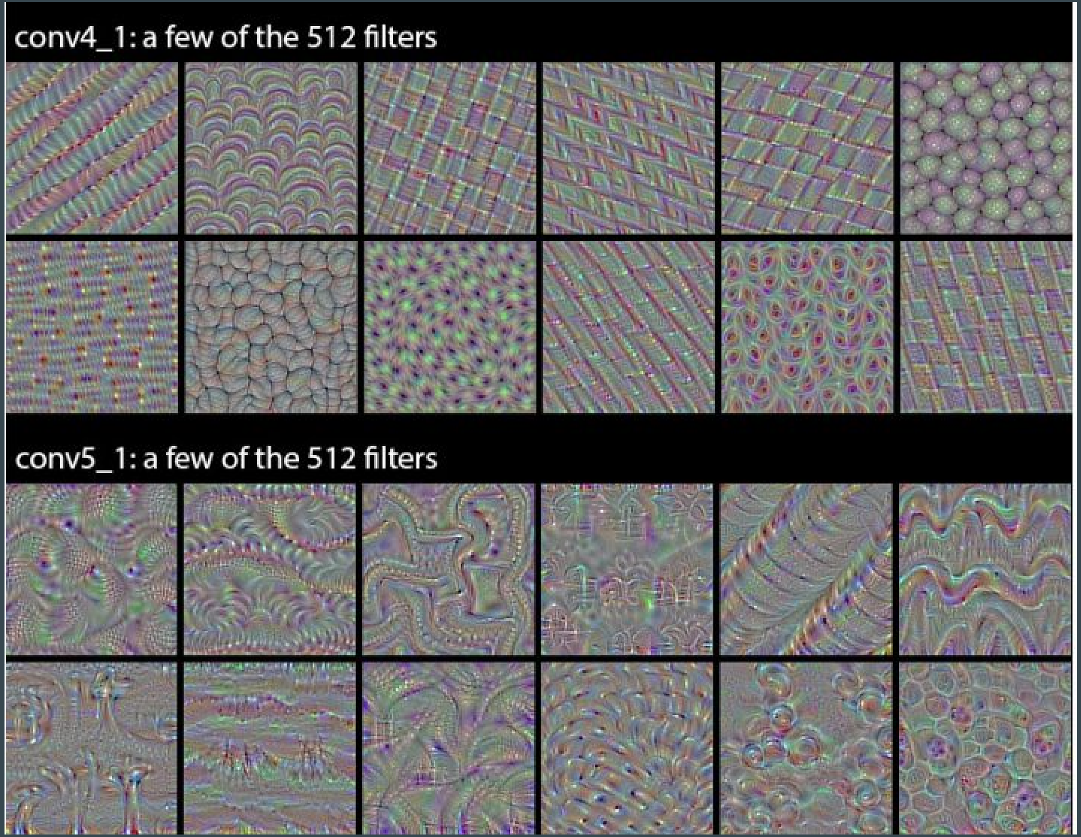
VGG-16
5 groups conv layers, total 13
Filter size = 3, pad = 1, stride = 1
Max pooling, filter size = 2, stride = 2
3 FC layers

ConvNet Example in Keras !

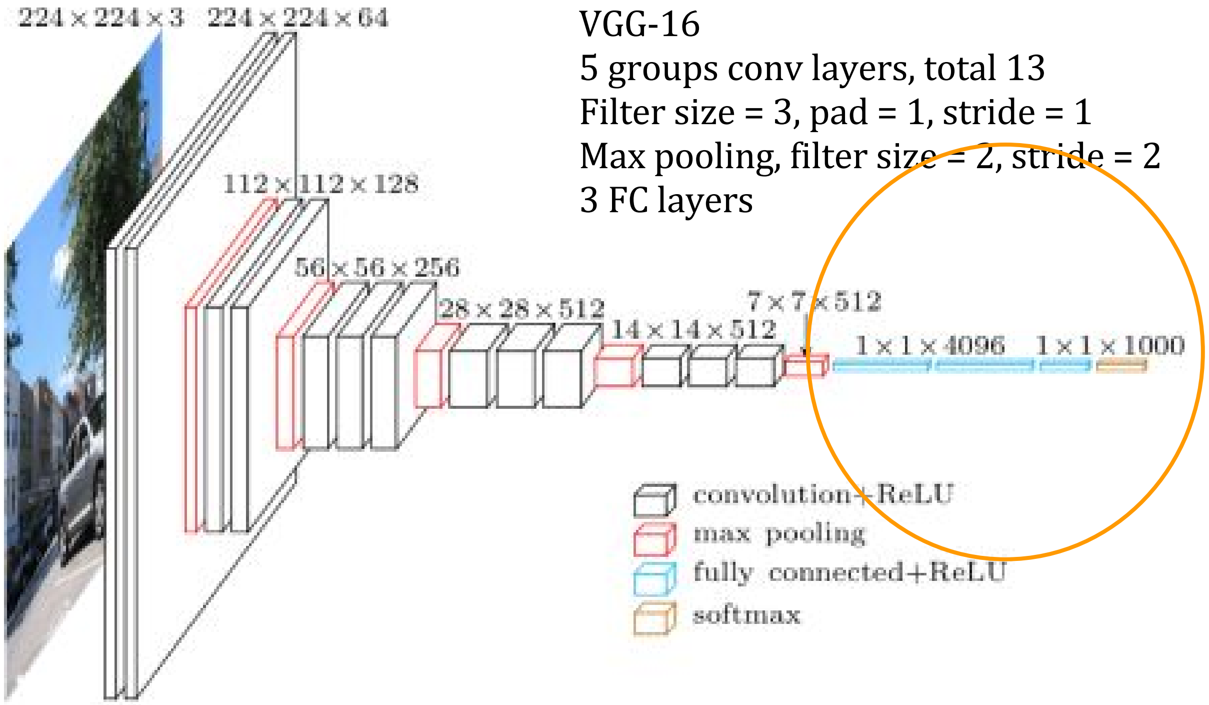


```
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
```

ConvNet Example in Keras !

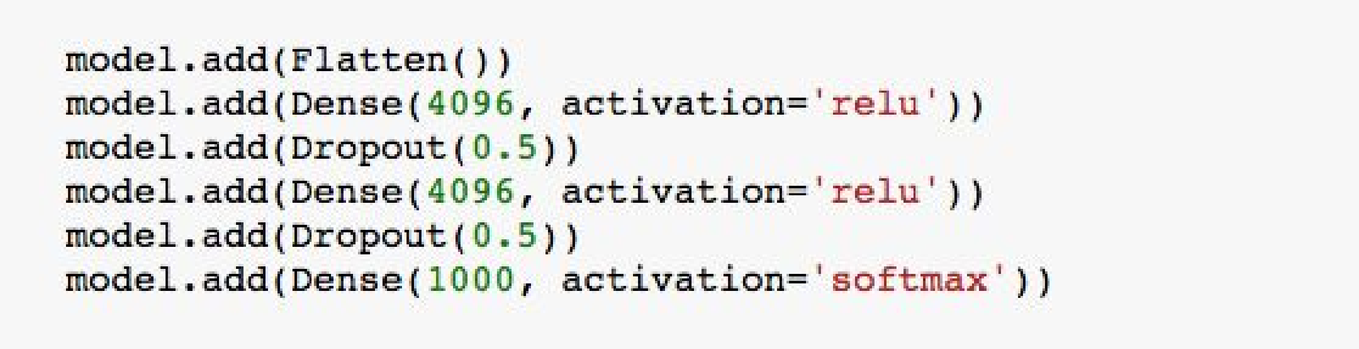

conv4_1: a few of the 512 filters
conv5_1: a few of the 512 filters

ConvNet Example in Keras !



VGG-16
5 groups conv layers, total 13
Filter size = 3, pad = 1, stride = 1
Max pooling, filter size = 2, stride = 2
3 FC layers

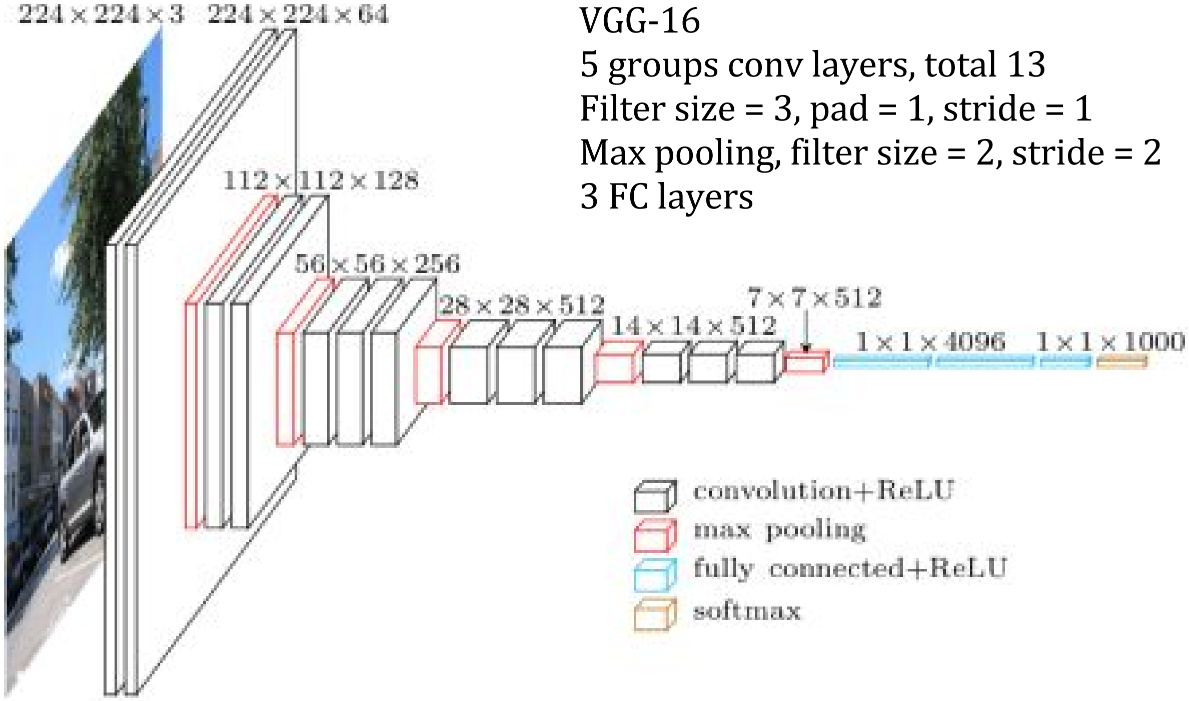
ConvNet Example in Keras !



```
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))
```

ConvNet Example in Keras !

VGG-16
5 groups conv layers, total 13
Filter size = 3, pad = 1, stride = 1
Max pooling, filter size = 2, stride = 2
3 FC layers


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax