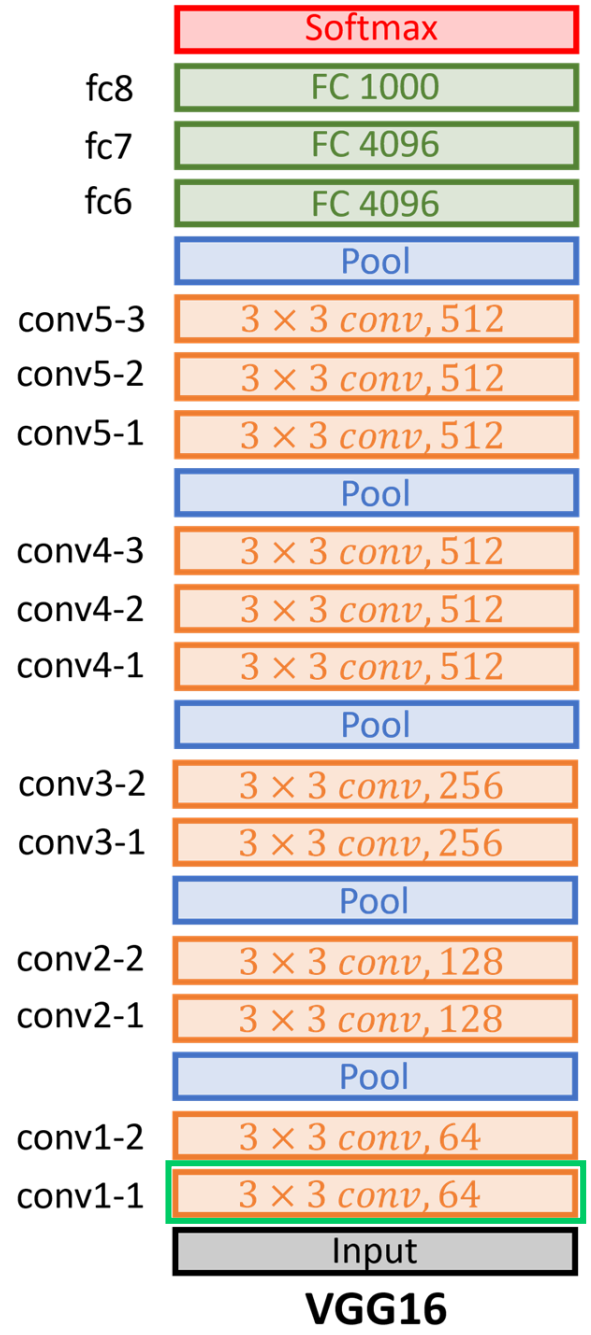
# Deep Learning

Mohammad Reza Mohammadi

2021

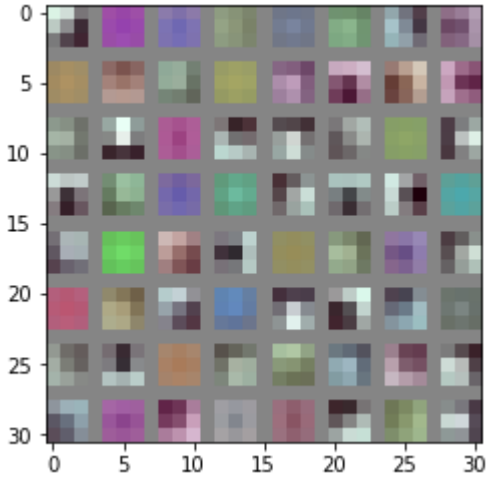
# Visualizing what convnets learn

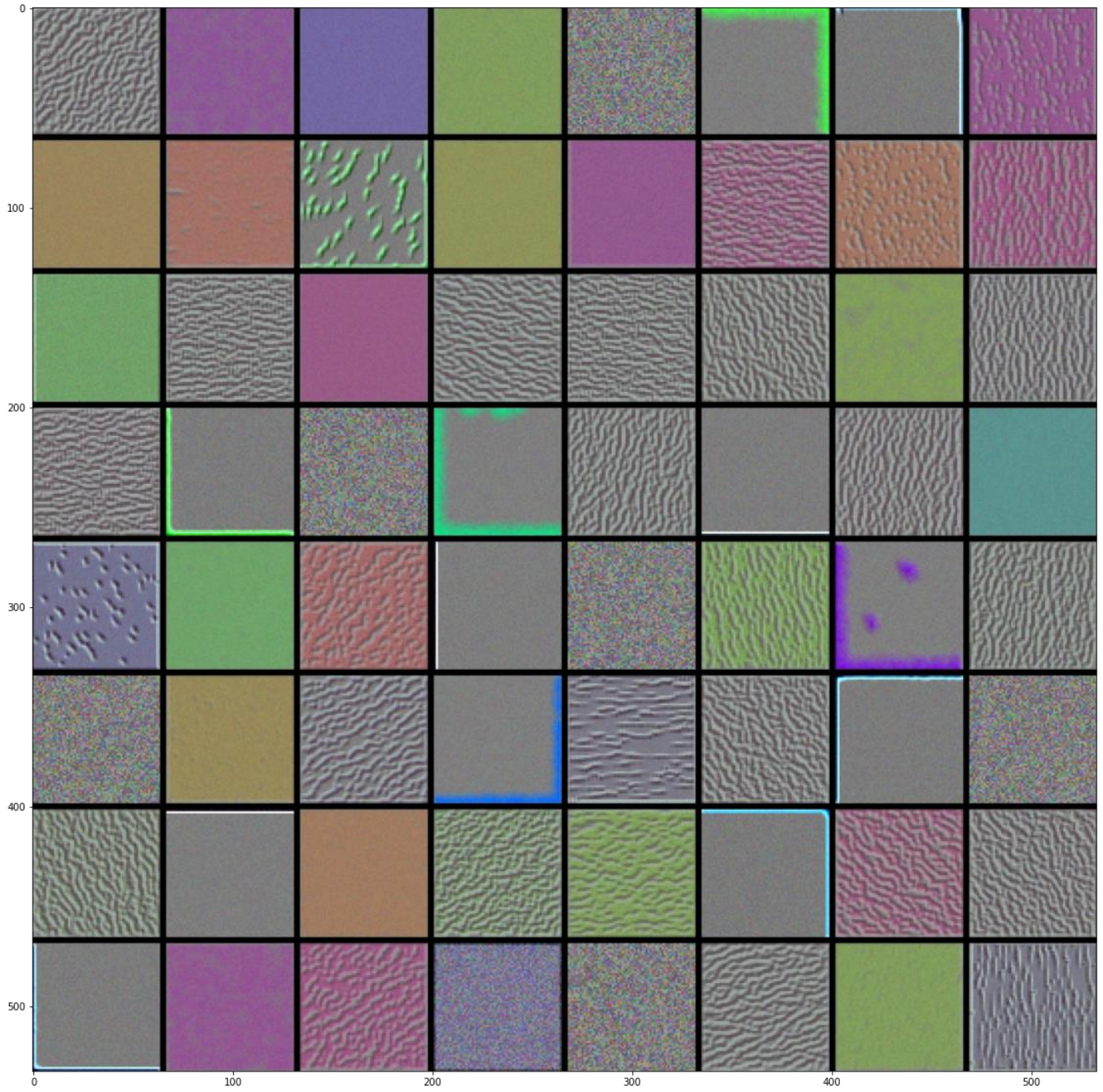
- Different ways to visualize or interpret NN representations, three common ones:
  - Visualizing intermediate convnet outputs (intermediate activations)
  - Visualizing convnets filters
  - Visualizing heatmaps of class activation in an image

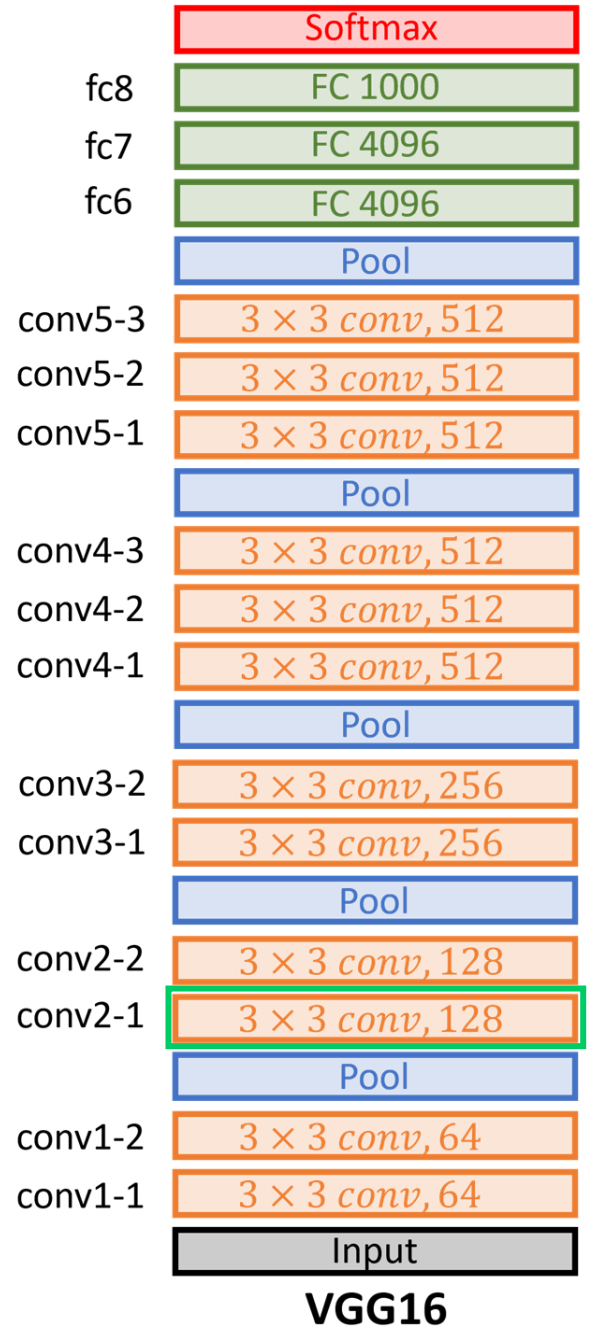



**VGG16**

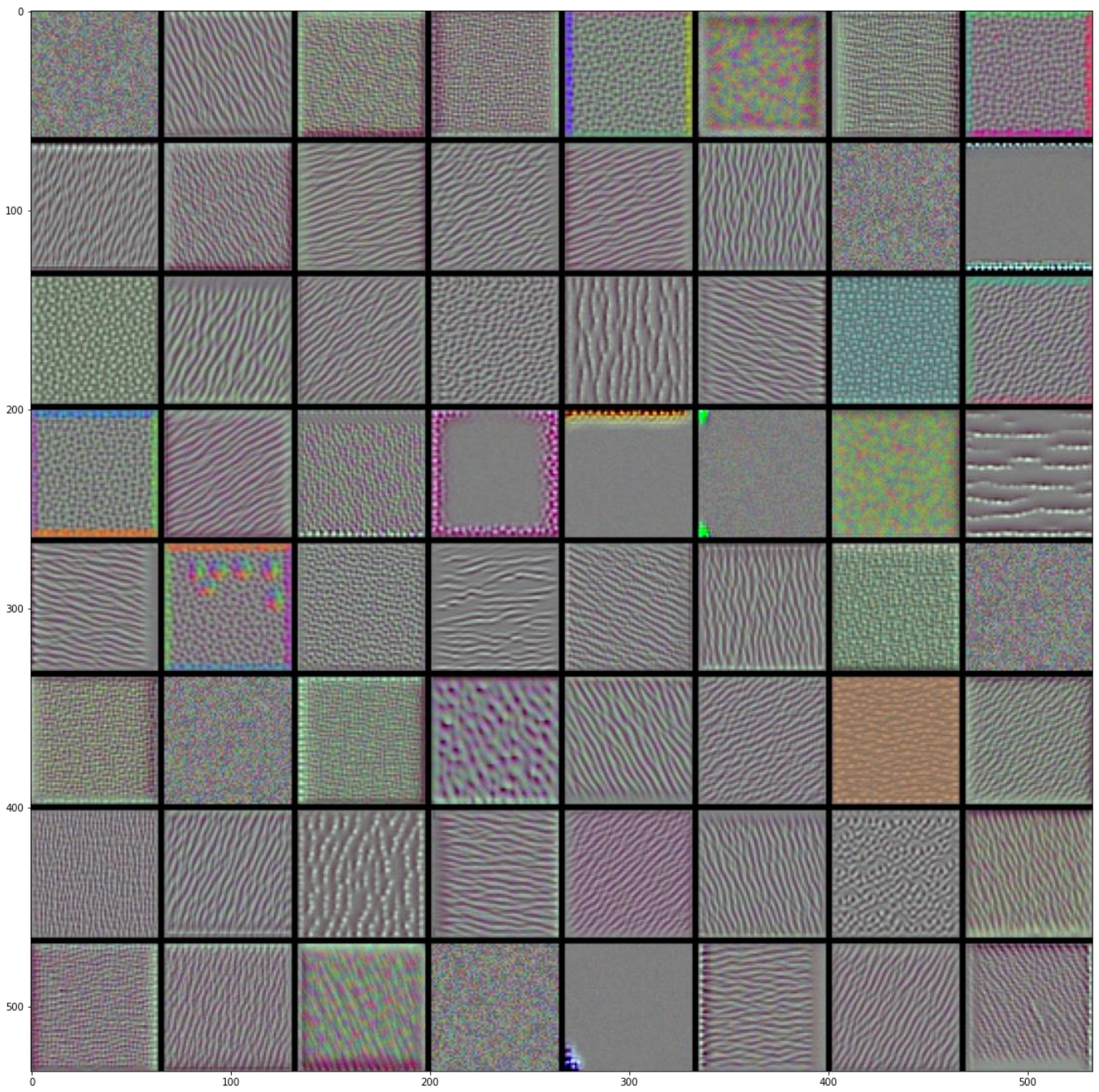
64, 3×3×3 filters

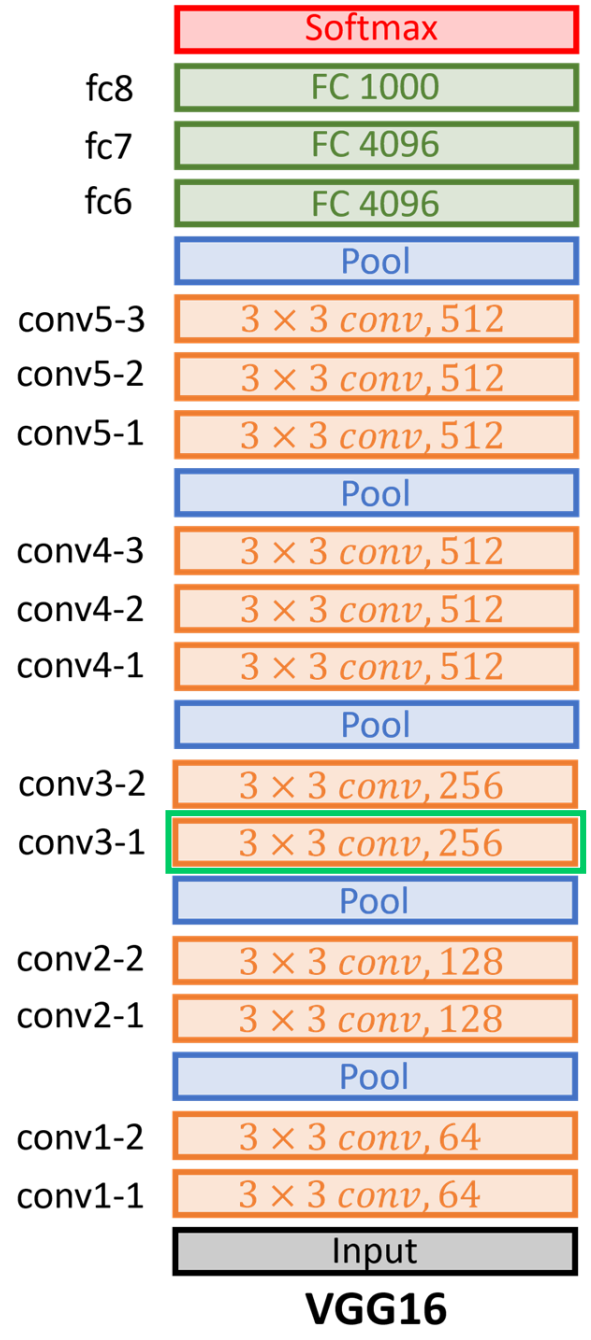


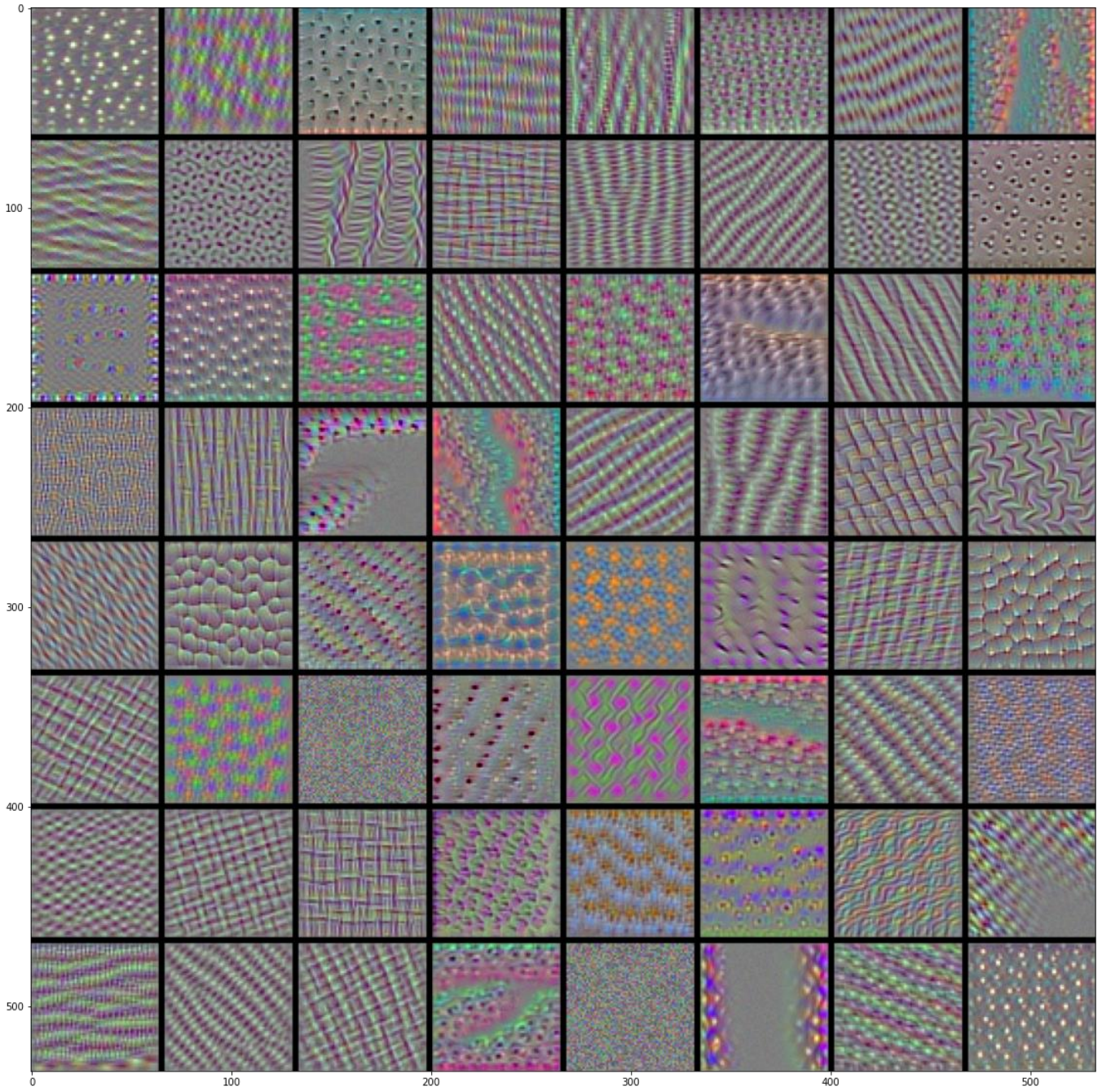

Softmax
fc8
FC 1000
fc7
FC 4096
fc6
FC 4096
Pool
conv5-3
3 × 3 conv, 512
conv5-2
3 × 3 conv, 512
conv5-1
3 × 3 conv, 512
Pool
conv4-3
3 × 3 conv, 512
conv4-2
3 × 3 conv, 512
conv4-1
3 × 3 conv, 512
Pool
conv3-2
3 × 3 conv, 256
conv3-1
3 × 3 conv, 256
Pool
conv2-2
3 × 3 conv, 128
conv2-1
3 × 3 conv, 128
Pool
conv1-2
3 × 3 conv, 64
conv1-1
3 × 3 conv, 64
Input

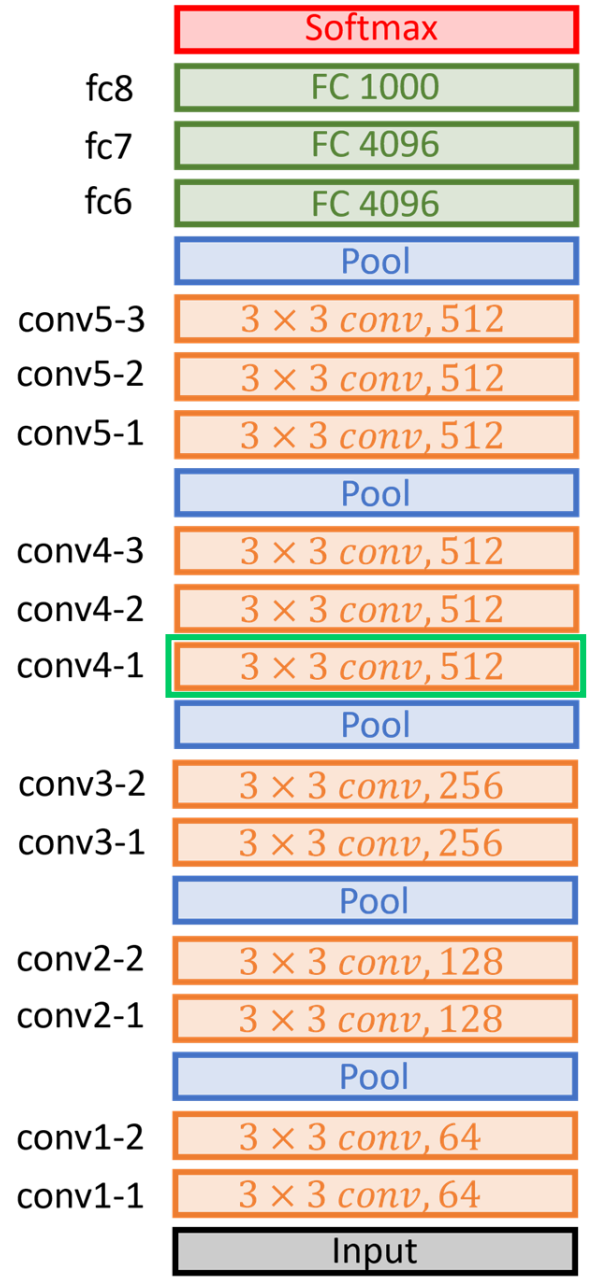
**VGG16**


0
100
200
300
400
500
0
100
200
300
400
500

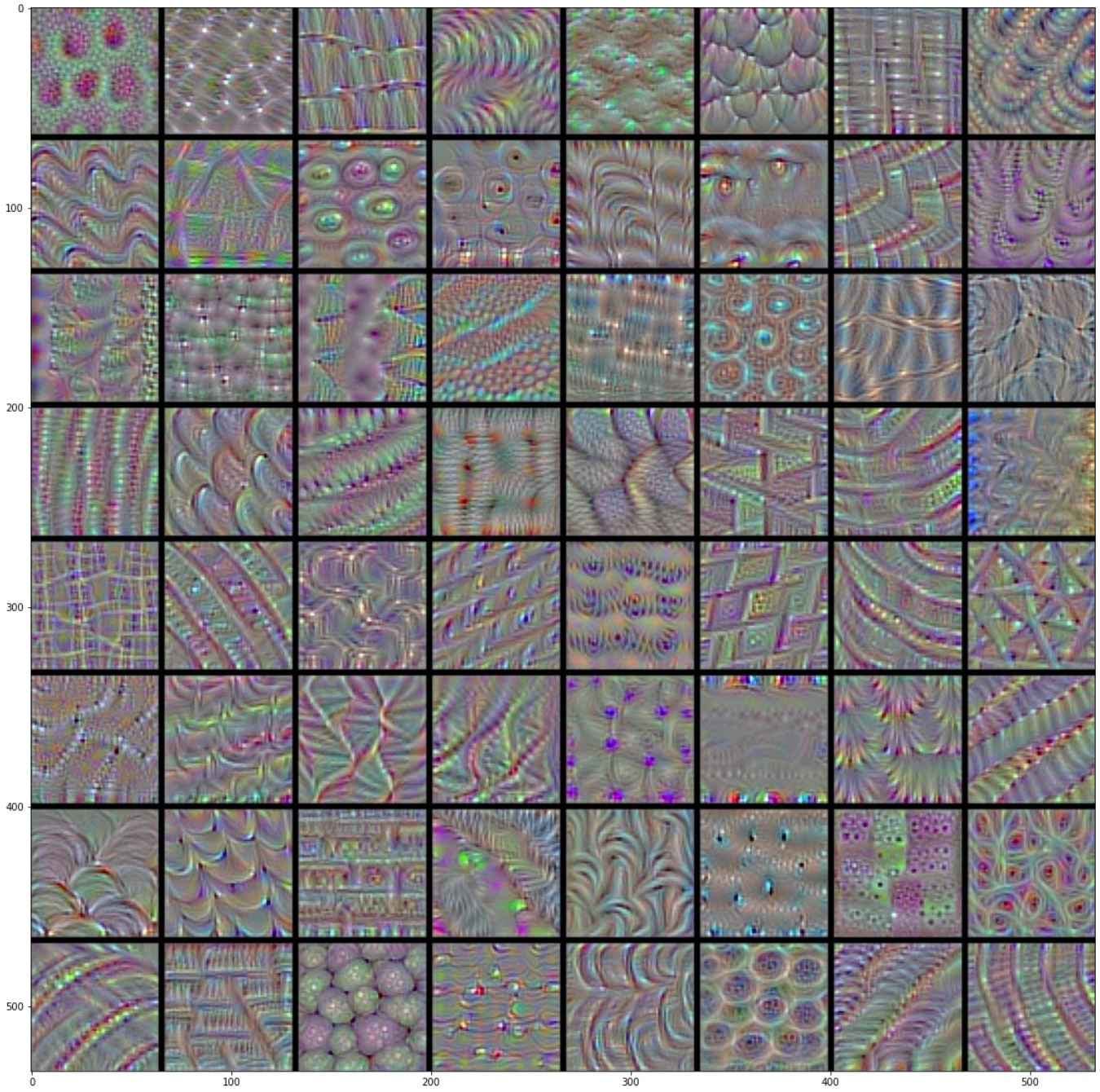

Softmax
fc8
FC 1000
fc7
FC 4096
fc6
FC 4096
Pool
conv5-3
3 × 3 conv, 512
conv5-2
3 × 3 conv, 512
conv5-1
3 × 3 conv, 512
Pool
conv4-3
3 × 3 conv, 512
conv4-2
3 × 3 conv, 512
conv4-1
3 × 3 conv, 512
Pool
conv3-2
3 × 3 conv, 256
conv3-1
3 × 3 conv, 256
Pool
conv2-2
3 × 3 conv, 128
conv2-1
3 × 3 conv, 128
Pool
conv1-2
3 × 3 conv, 64
conv1-1
3 × 3 conv, 64
Input

**VGG16**


0
100
200
300
400
500
0
100
200
300
400
500

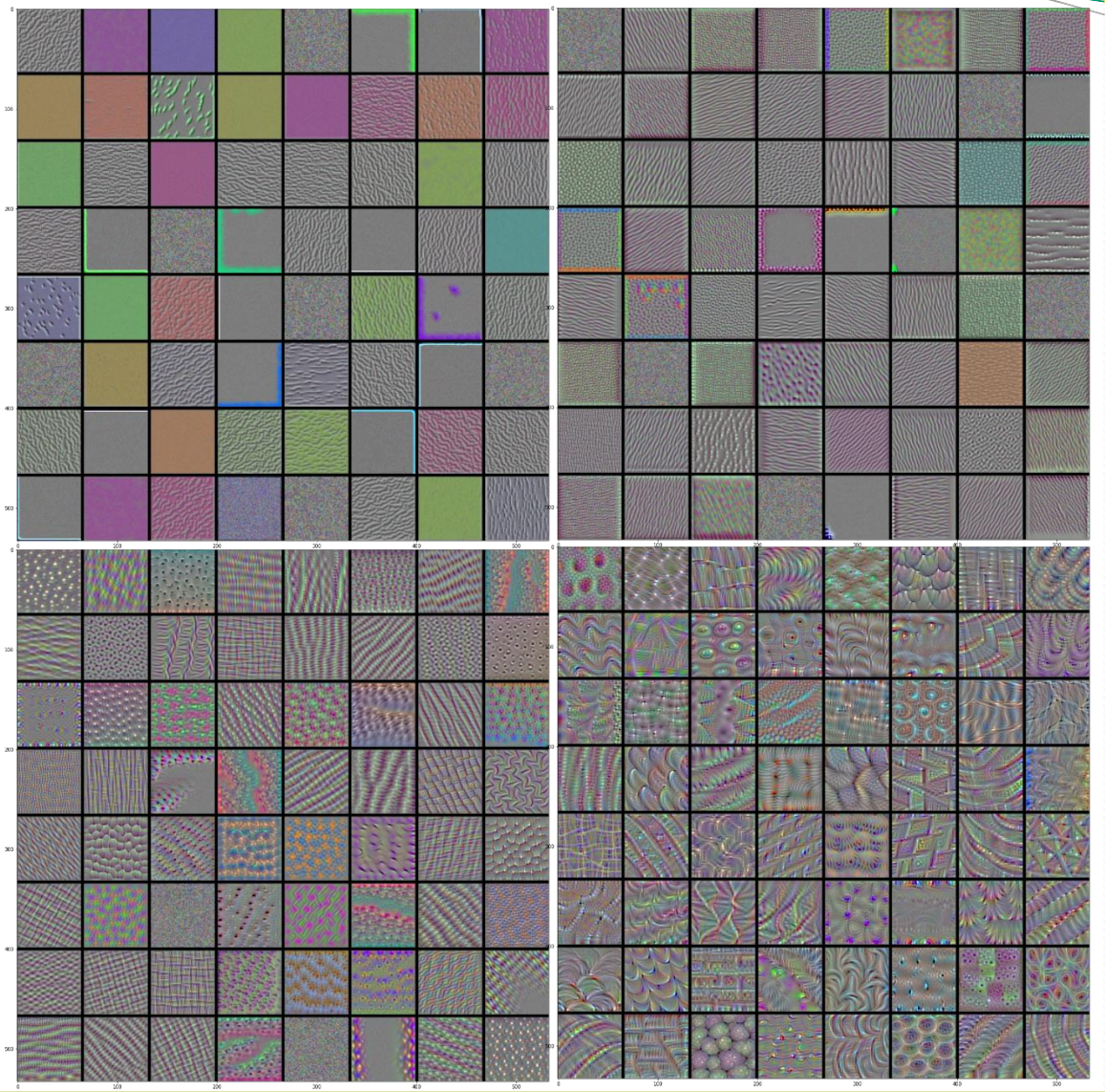

Softmax
fc8
FC 1000
fc7
FC 4096
fc6
FC 4096
Pool
conv5-3
3 × 3 conv, 512
conv5-2
3 × 3 conv, 512
conv5-1
3 × 3 conv, 512
Pool
conv4-3
3 × 3 conv, 512
conv4-2
3 × 3 conv, 512
conv4-1
3 × 3 conv, 512
Pool
conv3-2
3 × 3 conv, 256
conv3-1
3 × 3 conv, 256
Pool
conv2-2
3 × 3 conv, 128
conv2-1
3 × 3 conv, 128
Pool
conv1-2
3 × 3 conv, 64
conv1-1
3 × 3 conv, 64
Input

**VGG16**


0
100
200
300
400
500
0
100
200
300
400
500

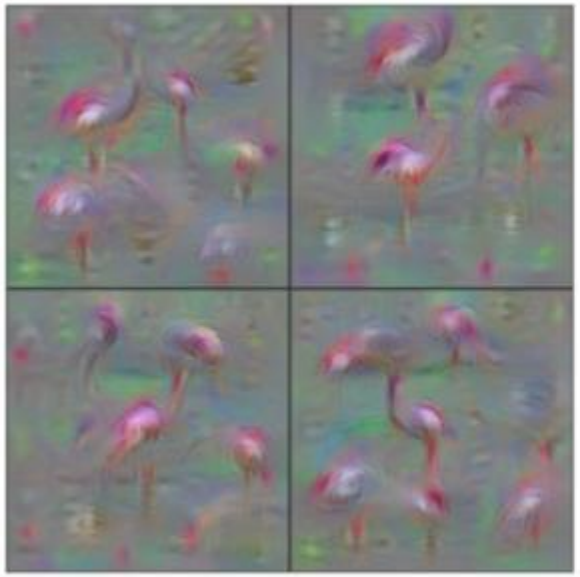
# Visualizing convnet filters

- The filters get increasingly complex and refined as you go higher in the model:
  - The filters from the first layer in the model encode simple directional edges and colors (or colored edges, in some cases)
  - The filters from block2_conv1 encode simple textures made from combinations of edges and colors
  - The filters in higher layers begin to resemble textures found in natural image: feathers, eyes, leaves, and so on

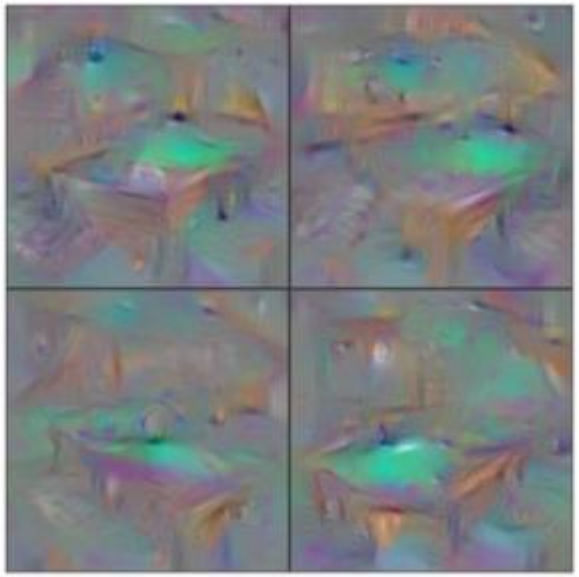


# DeepVis

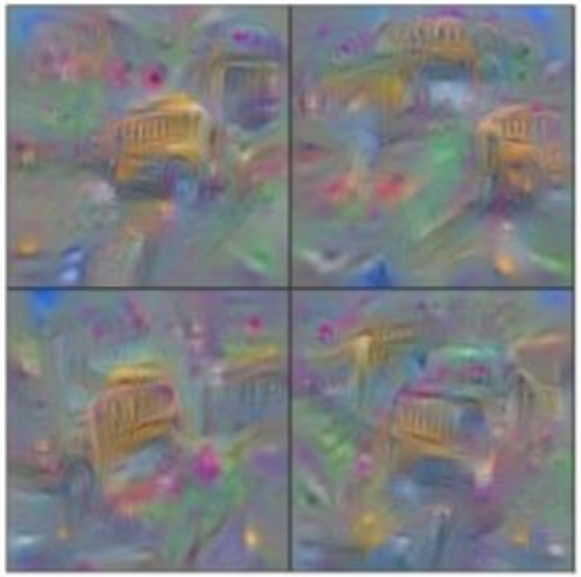
- Visualizing output neurons
  - The synthetic input image that best fires that neuron



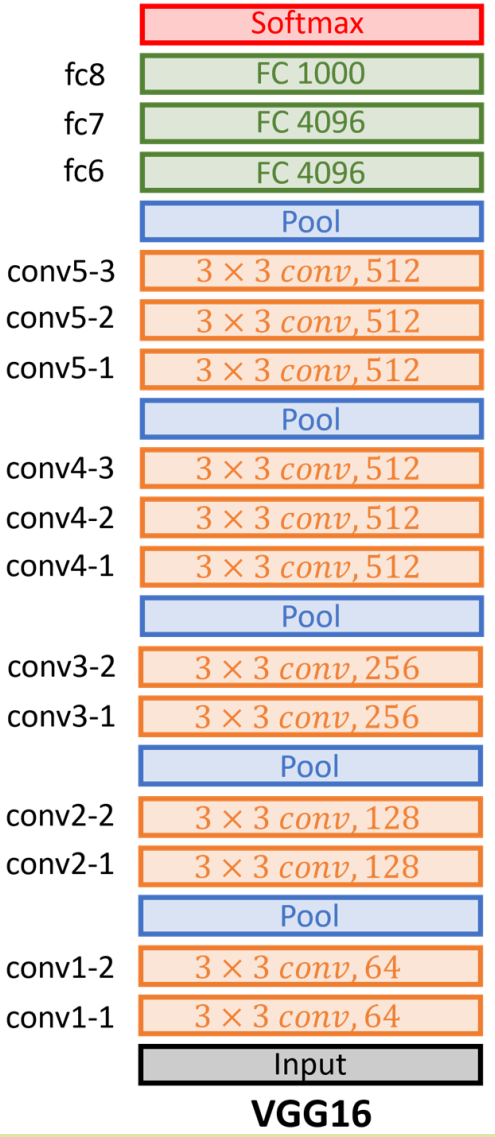
Flamingo



Billiard Table



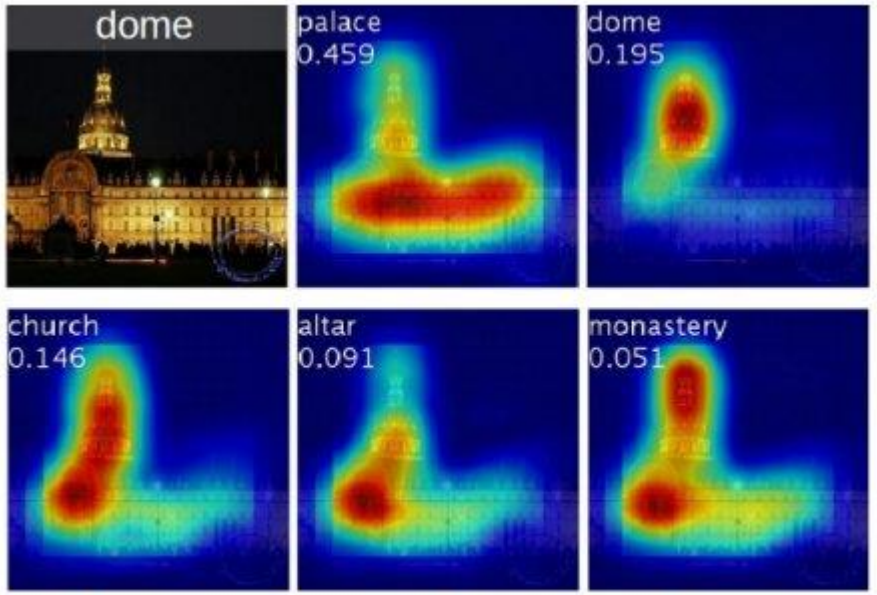
School Bus




**VGG16**

# Visualizing heatmaps of class activation

- Class activation map (CAM) visualization
  - Heatmaps of class activation over input images
- Indicates how important each location is with respect to the class under consideration



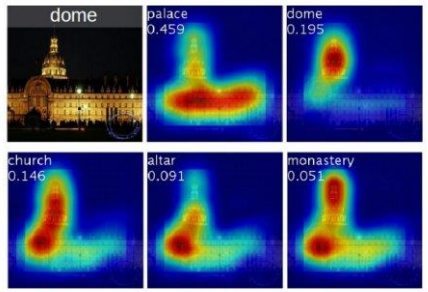
Class activation maps of top 5 predictions

Class activation maps for one object class

https://github.com/zhoubolei/CAM

# Visualizing heatmaps of class activation

- Class activation map (CAM) visualization
  - Producing heatmaps of class activation over input images
- Useful for understanding which parts of a given image led a convnet to its final classification decision
  - Helpful for debugging the decision process of a convnet, particularly in the case of a classification mistake
- It also allows you to locate specific objects in an image



Class activation maps of top 5 predictions

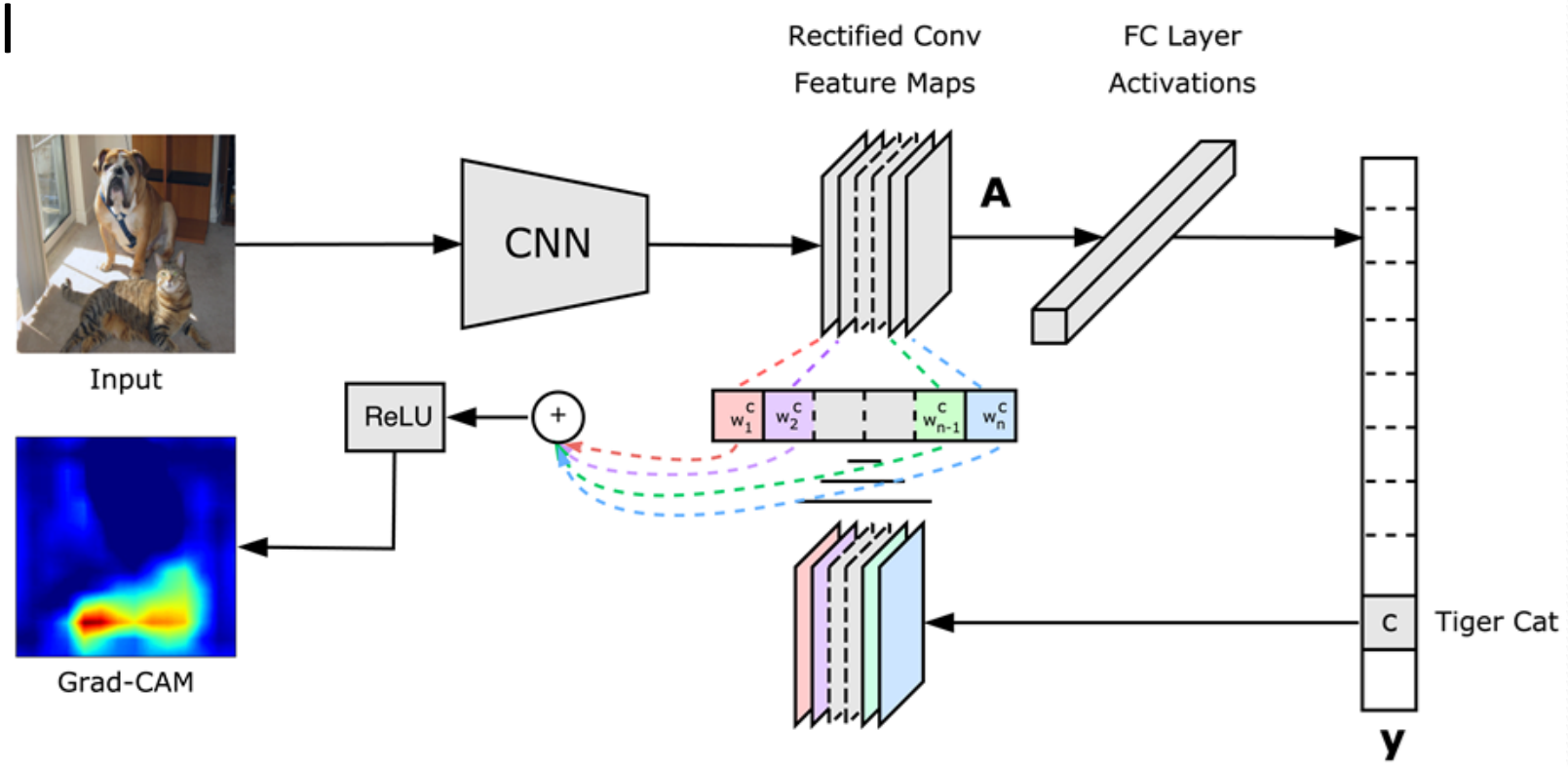
Class activation maps for one object class

# Grad-CAM

- Take the output feature map of a convolution layer, given an input image, and weight every channel in that feature map by the gradient of the class with respect to the channel

$$w_k^c = \frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

$$L^c_{\text{Grad-CAM}} = ReLU\left(\sum_k w_k^c\, A^k\right)$$



## Listing 5.40 Loading the VGG16 network with pretrained weights

```
from keras.applications.vgg16 import VGG16

model = VGG16(weights='imagenet')
```

Note that you include the densely connected classifier on top; in all previous cases, you discarded it.

## Listing 5.41 Preprocessing an input image for VGG16

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

img_path = '/Users/fchollet/Downloads/creative_commons_elephant.jpg'

img = image.load_img(img_path, target_size=(224, 224))

x = image.img_to_array(img)

x = np.expand_dims(x, axis=0)

x = preprocess_input(x)
```

Local path to the target image

Python Imaging Library (PIL) image of size 224 × 224

float32 Numpy array of shape (224, 224, 3)

Adds a dimension to transform the array into a batch of size (1, 224, 224, 3)

Preprocesses the batch (this does channel-wise color normalization)

```
>>> preds = model.predict(x)
>>> print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
>>> preds = model.predict(x)
>>> print('Predicted:', decode_predictions(preds, top=3)[0])

Predicted:', [(u'n02504458', u'African_elephant', 0.92546833),
(u'n01871265', u'tusker', 0.070257246),
(u'n02504013', u'Indian_elephant', 0.0042589349)]

>>> np.argmax(preds[0])

386
```

## Listing 5.42 Setting up the Grad-CAM algorithm

"African elephant" entry in the prediction vector

```
african_elephant_output = model.output[:, 386]

last_conv_layer = model.get_layer('block5_conv3')
```

Output feature map of the block5_conv3 layer, the last convolutional layer in VGG16

Gradient of the "African elephant" class with regard to the output feature map of block5_conv3

Vector of shape (512,), where each entry is the mean intensity of the gradient over a specific feature-map channel

```
grads = K.gradients(african_elephant_output, last_conv_layer.output)[0]

pooled_grads = K.mean(grads, axis=(0, 1, 2))

iterate = K.function([model.input],
                     [pooled_grads, last_conv_layer.output[0]])

pooled_grads_value, conv_layer_output_value = iterate([x])

for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]

heatmap = np.mean(conv_layer_output_value, axis=-1)
```

Values of these two quantities, as Numpy arrays, given the sample image of two elephants

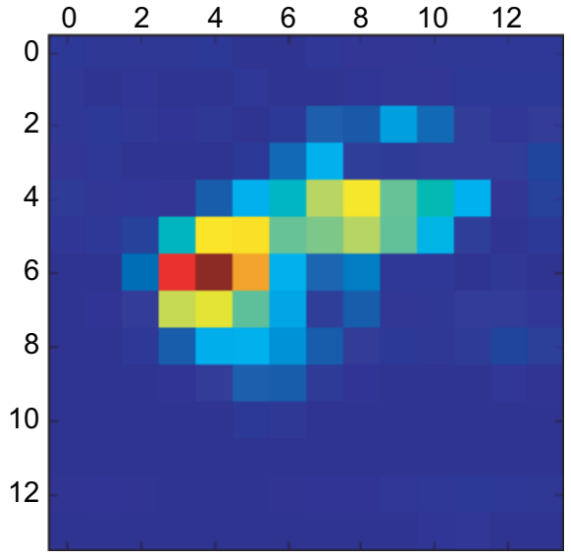
The channel-wise mean of the resulting feature map is the heatmap of the class activation.

Multiplies each channel in the feature-map array by "how important this channel is" with regard to the "elephant" class

Lets you access the values of the quantities you just defined: pooled_grads and the output feature map of block5_conv3, given a sample image

## Listing 5.43 Heatmap post-processing

```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
```




**Listing 5.44 Superimposing the heatmap with the original picture**

```
import cv2
img = cv2.imread(img_path)
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
heatmap = np.uint8(255 * heatmap)
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
superimposed_img = heatmap * 0.4 + img
cv2.imwrite('/Users/fchollet/Downloads/elephant_cam.jpg', superimposed_img)
```

Uses cv2 to load the original image

Resizes the heatmap to be the same size as the original image

Converts the heatmap to RGB

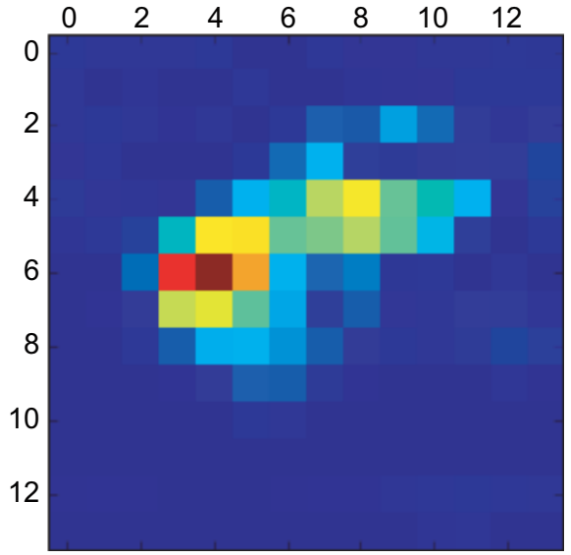
0.4 here is a heatmap intensity factor.

Applies the heatmap to the original image

Saves the image to disk

**Listing 5.43 Heatmap post-processing**

```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
```



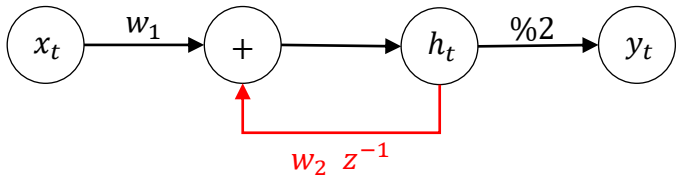

# Visualizing heatmaps of class activation

- This visualization technique answers two important questions:
  - Why did the network think this image contained an African elephant?
  - Where is the African elephant located in the picture?
- The ears of the elephant calf are strongly activated:
  - This is probably how the network can tell the difference between African and Indian elephants

# Sequence Modeling: Recurrent Neural Networks

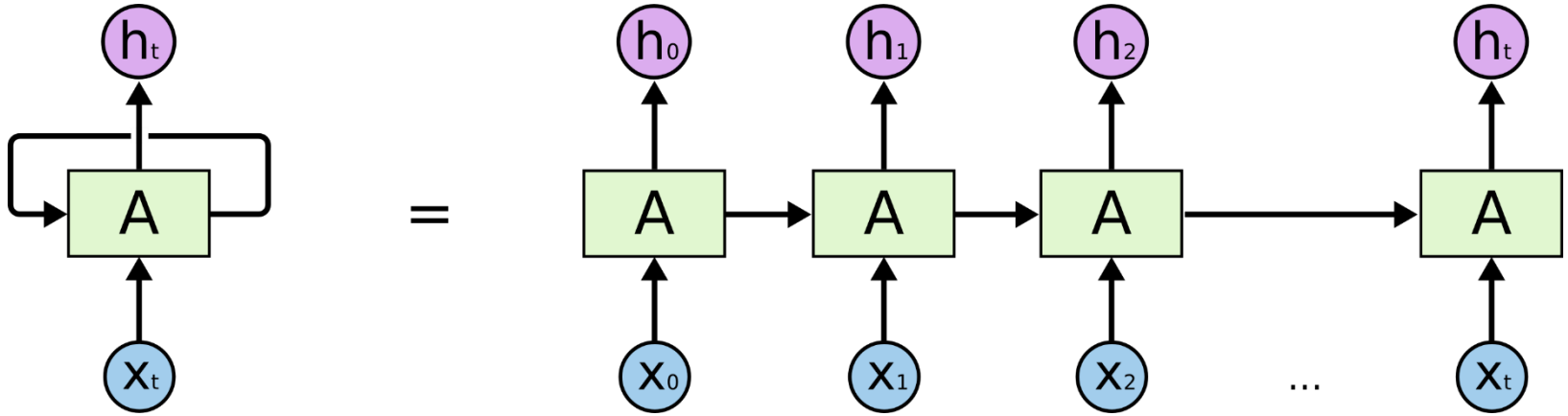
# Motivation

- Not all problems can be converted into one with fixed-length inputs and outputs
- Problems such as Speech Recognition or Time-series Prediction require a system to store and use context information
- Simple case: Output YES if the number of 1s is odd, else NO
  - 1000010101 – No, 10001100000000000000 – Yes, …
- Hard/Impossible to choose a fixed context window

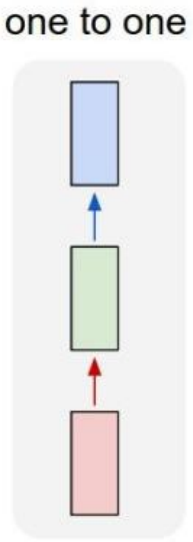


# Recurrent neural networks

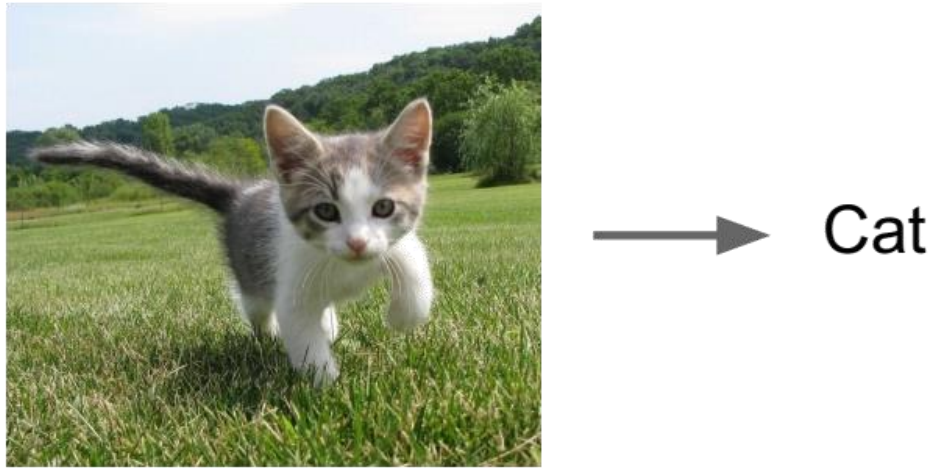
- Recurrent neural networks or RNNs are a family of neural networks for processing sequential data
- RNNs are specialized for processing a sequence of values $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}$
- Most recurrent networks can also process sequences of variable length
- An RNN shares the same weights across several time steps
- Computational graphs that include cycles



# Neural Networks

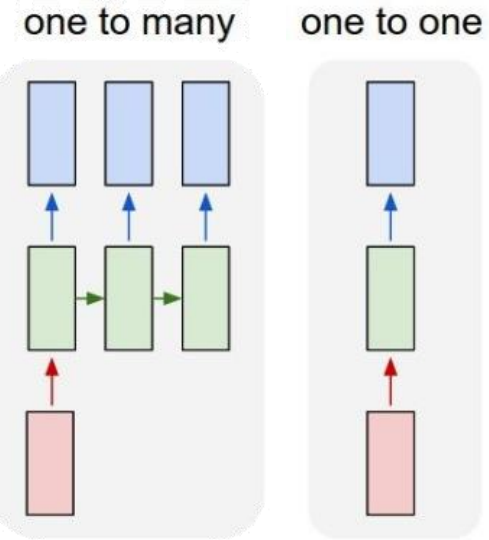



- Dense and ConvNet have no memory!
  - They process the input shown to them independently (with no state in between them)
- These are called feedforward networks



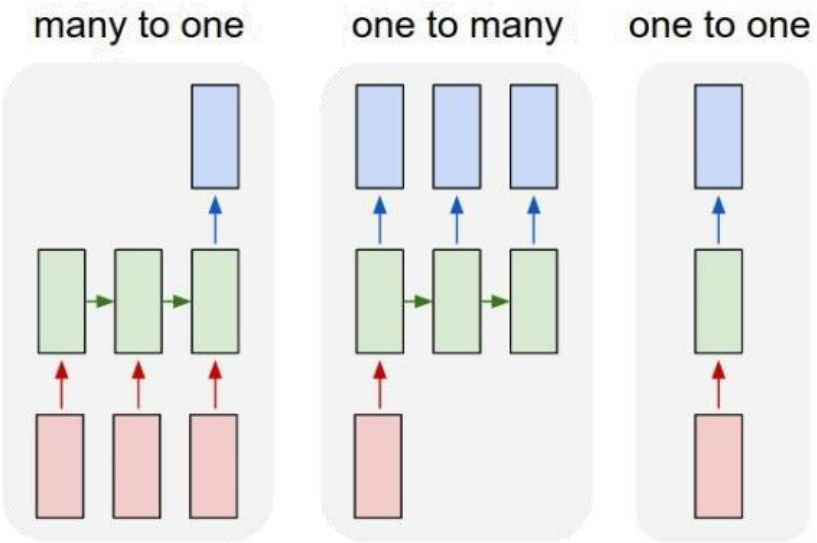
Cat

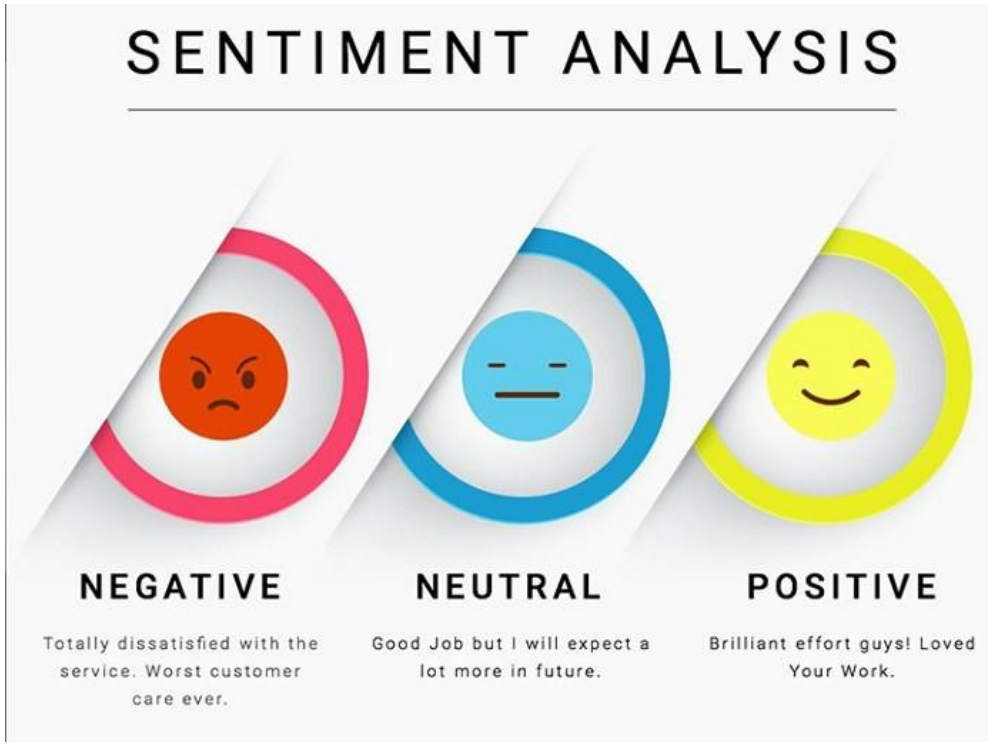
# Neural Networks



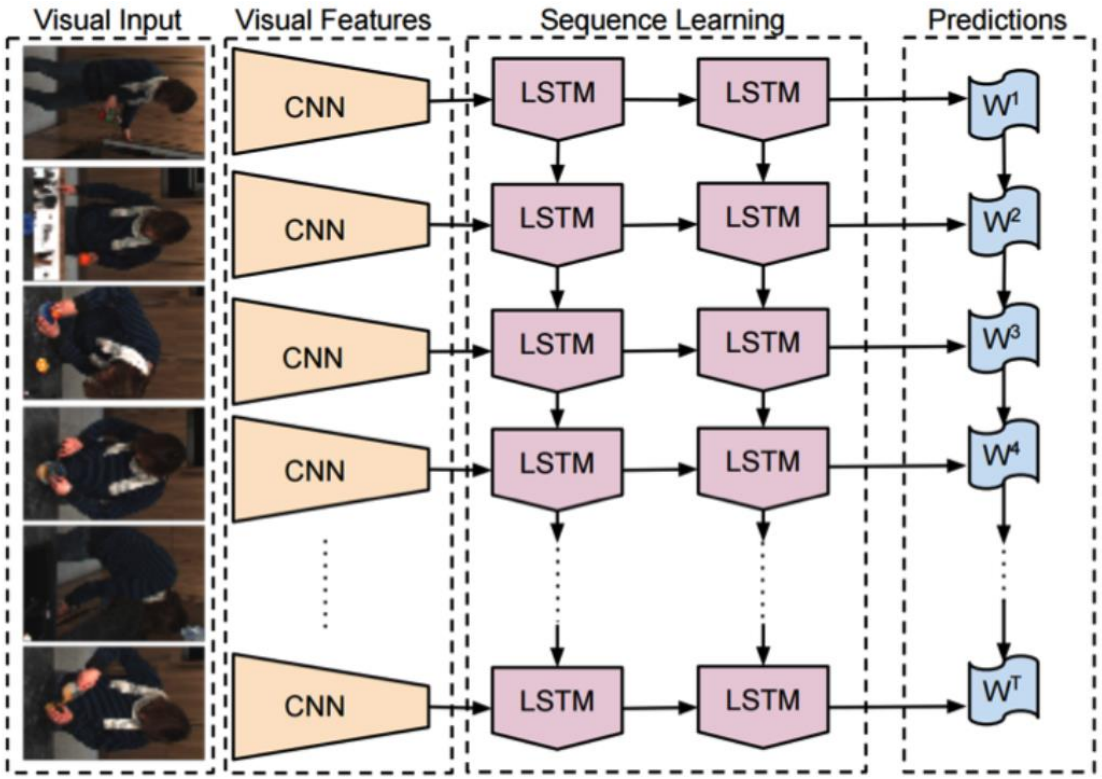

*A cat is sitting on a tree branch*

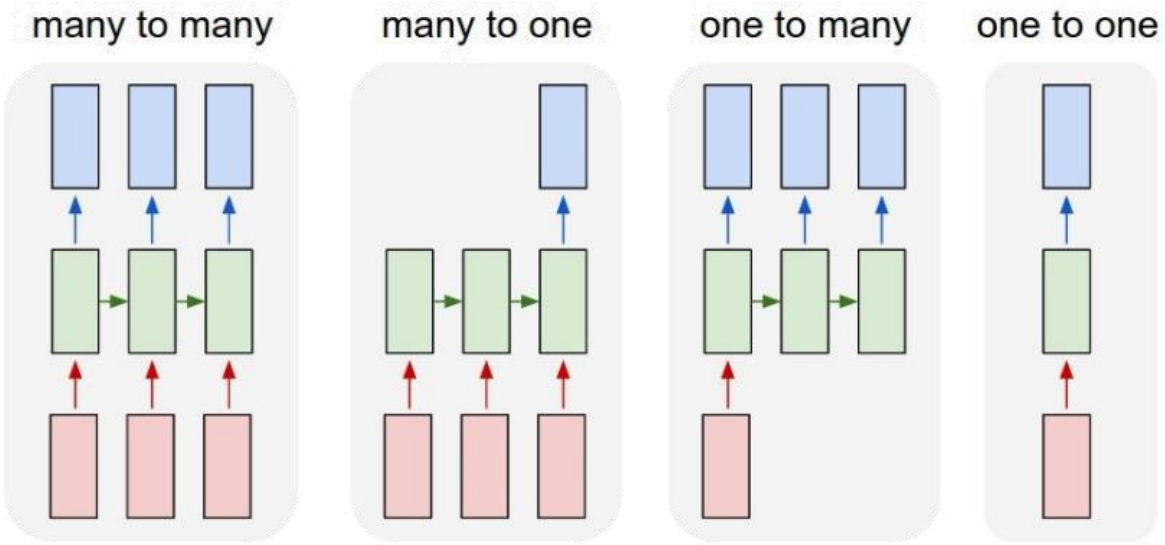
# Neural Networks



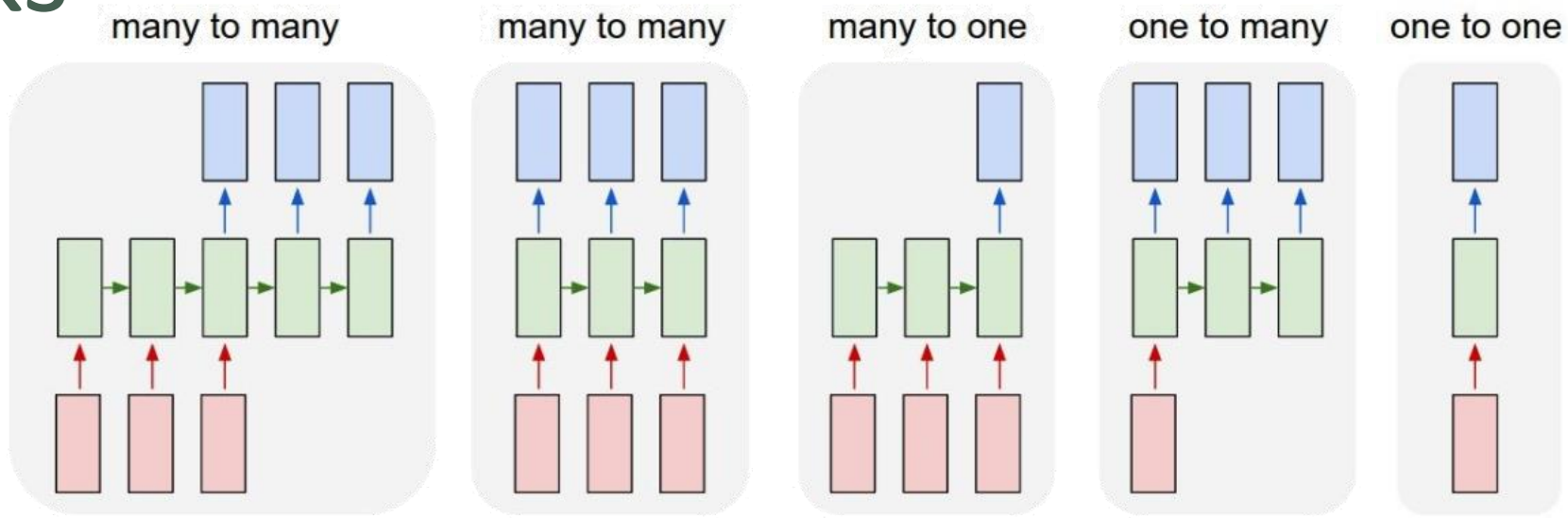



# Neural Networks



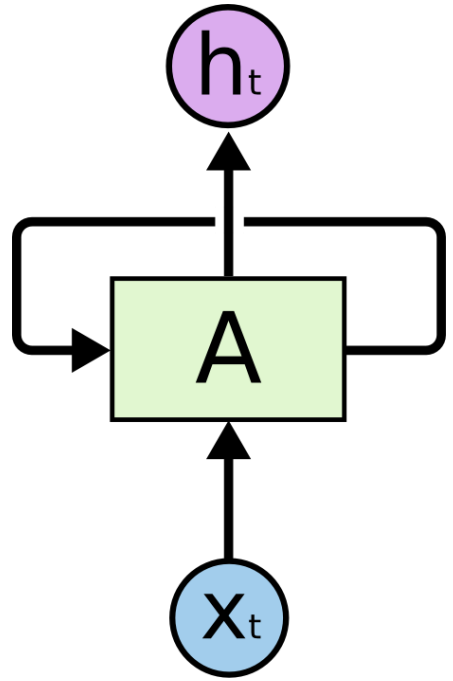
# Neural Networks




| | |
|---|---|
| Input: | If you face a problem try to find the solution not the reason |
| Output: | مشکل که به وجود اومد بگرد راه حلش را پیدا کن نگرد دنبال این که چرا به وجود اومد |

# Recurrent Neural Networks

- RNNs take the previous output or hidden states as inputs
- The composite input at time $t$ has some historical information about the happenings at time $T < t$
- RNNs are useful as their intermediate values (state) can store information about past inputs



# Recurrent Neural Networks

$$\boldsymbol{h}_3 = f_W(\boldsymbol{h}_2, \boldsymbol{x}_3)$$
$$= f_W(f_W(\boldsymbol{h}_1, \boldsymbol{x}_2), \boldsymbol{x}_3)$$
$$= f_W(f_W(f_W(\boldsymbol{h}_0, \boldsymbol{x}_1), \boldsymbol{x}_2), \boldsymbol{x}_3)$$
$$= g^{(3)}(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$$