BST 261: Data Science II

Lecture 9

Heather Mattie

Department of Biostatistics
Harvard T.H. Chan School of Public Health
Harvard University

April 16, 2018

Visualizing What CNNs Learn

Types of Visualization

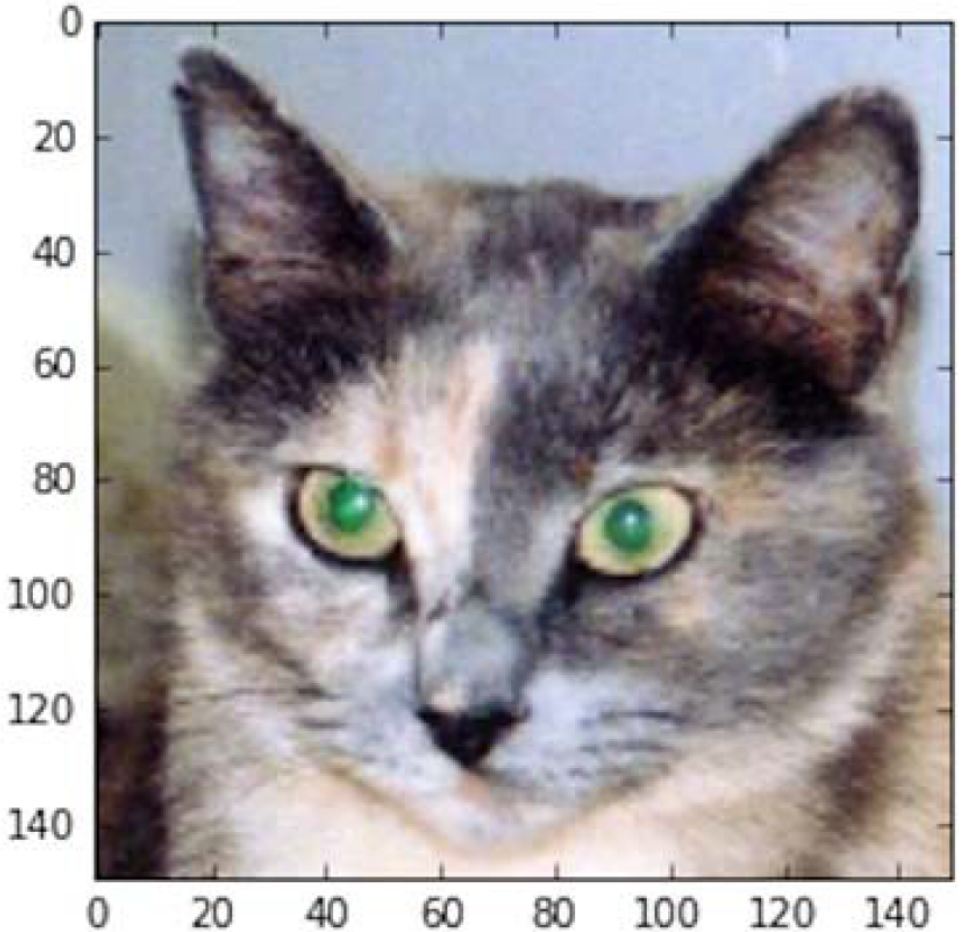
- Deep learning models are typically referred to as 'black boxes' because it is difficult to extract and present the learned representations in a way that humans can understand
- Because CNNs are used to learn visual representations, it is possible for us to visualize the learned representations
- The 3 most common techniques are
  - **Visualizing intermediate outputs** (intermediate activations)
    - For understanding how successive layers transform their input, and for getting an idea of the meaning of individual filters
  - **Visualizing filters**
    - For understanding exactly what visual pattern or concept each filter is receptive to
  - **Visualizing heatmaps of class activation**
    - For understanding which parts of an image were identified as belonging to a given class

Visualizing Intermediate Outputs

- Display the feature maps that are output by various convolution and pooling layers
- You should look at each channel separately

```
1  from keras.models import load_model
2  from keras import models
3  from keras.preprocessing import image
4  import numpy as np
5
6  model = load_model('cats_and_dogs_small_2.h5')
7  img_path = '/Users/fchollet/Downloads/cats_and_dogs_small/test/cats/cat.1700.jpg
       '
8
9  img = image.load_img(img_path, target_size=(150, 150))
10 img_tensor = image.img_to_array(img)
11 img_tensor = np.expand_dims(img_tensor, axis=0)
12 # Remember that the model was trained on inputs that were preprocessed in the
       following way:
13 img_tensor /= 255.
14
15 # Its shape is (1, 150, 150, 3)
16 print(img_tensor.shape)
```

Visualizing Intermediate Outputs

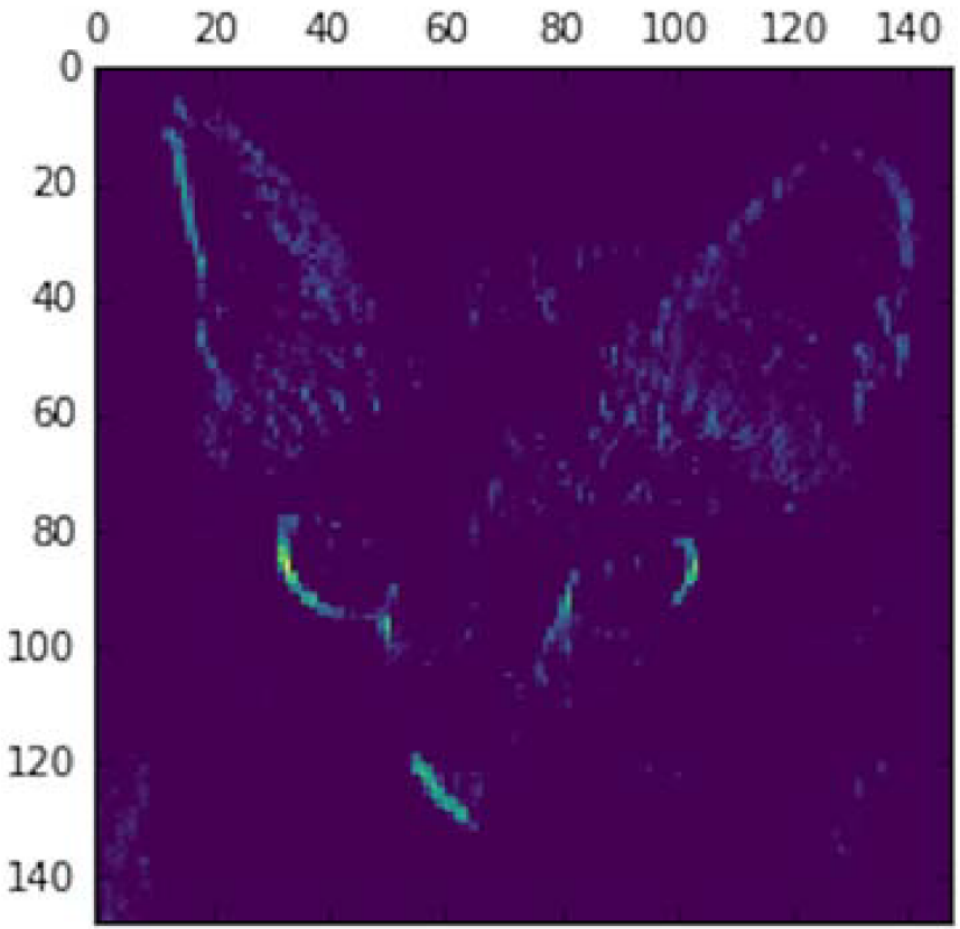


Visualizing Intermediate Outputs

```
# Extracts the outputs of the top 8 layers:
layer_outputs = [layer.output for layer in model.layers[:8]]
# Creates a model that will return these outputs, given the model input:
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)


# This will return a list of 5 Numpy arrays:
# one array per layer activation
activations = activation_model.predict(img_tensor)

first_layer_activation = activations[0]

# Visualize the 3rd channel
plt.matshow(first_layer_activation[0, :, :, 3], cmap='viridis')
plt.show()
```

Visualizing Intermediate Outputs

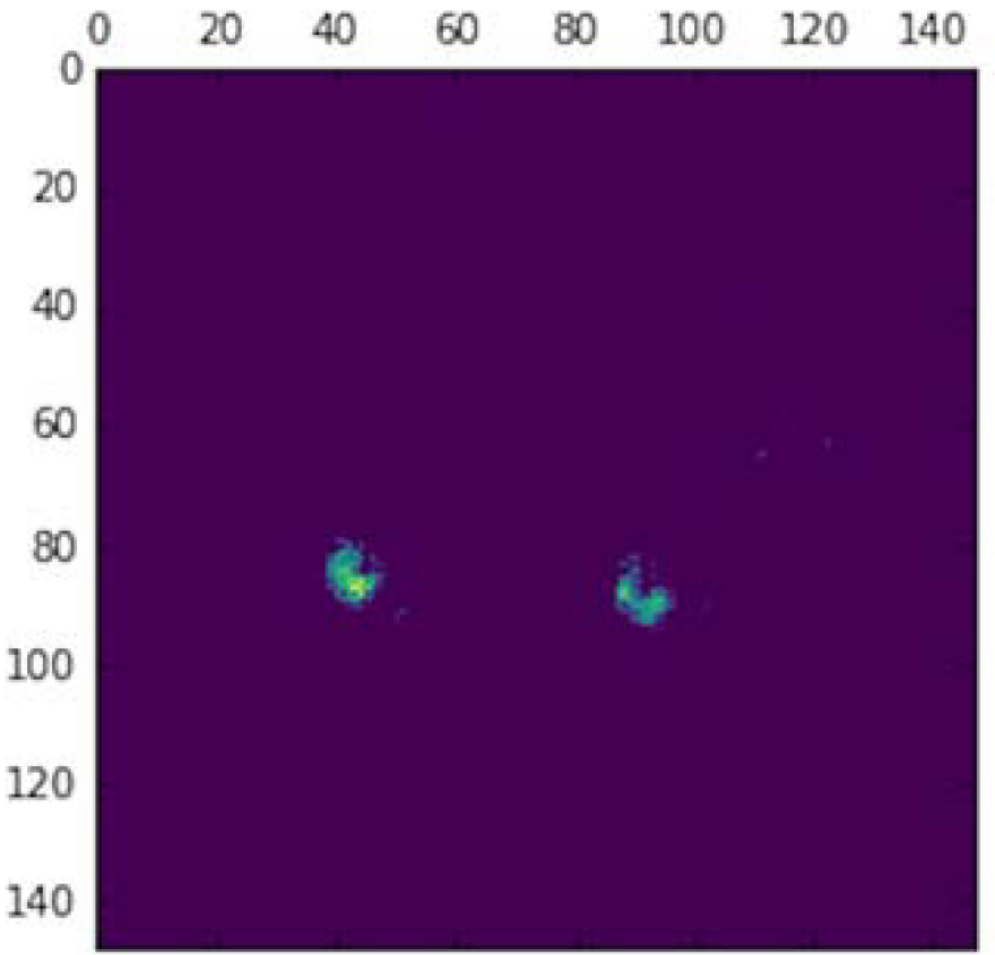
- This channel appears to encode a diagonal edge detector



Visualizing Intermediate Outputs

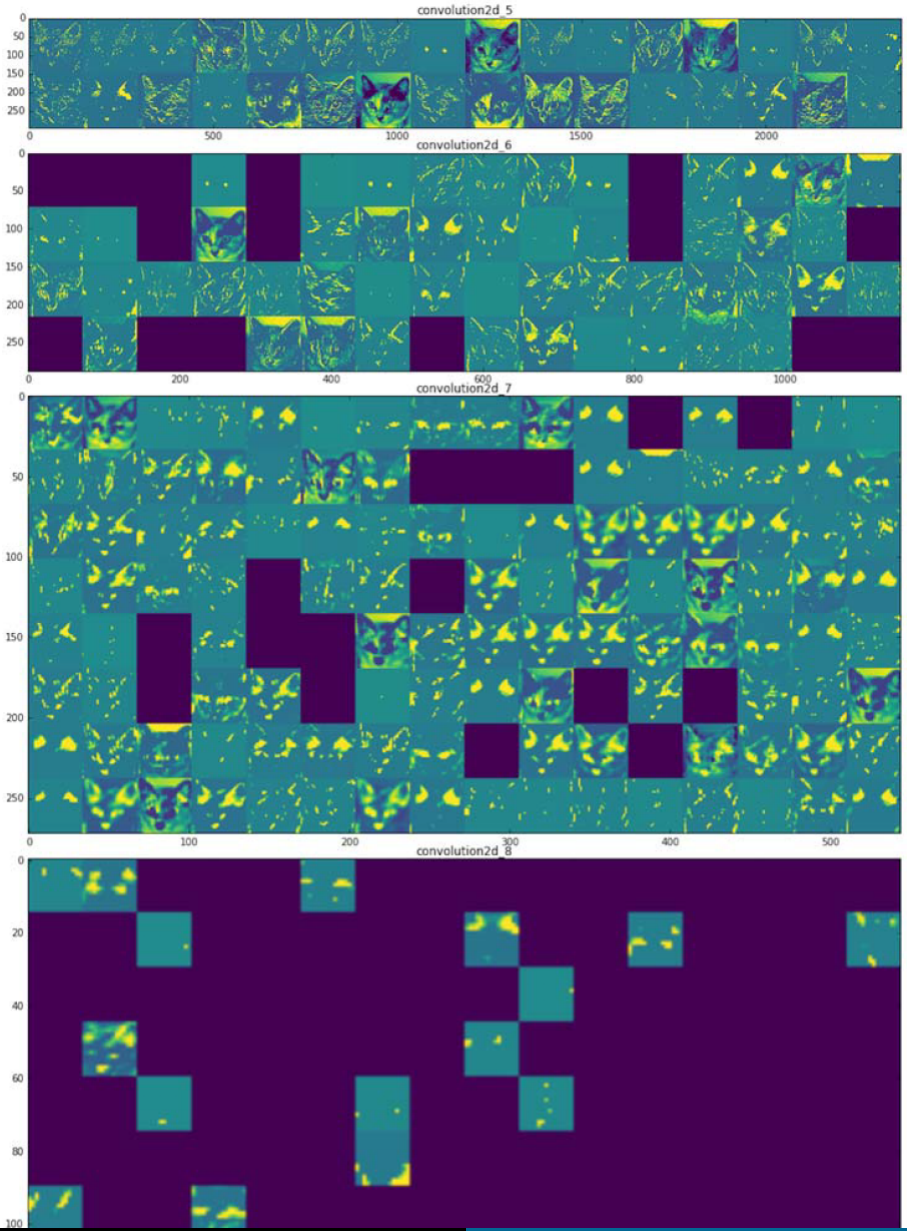
- It seems like this channels picks up on bright green dots

```
1 plt.matshow(first_layer_activation[0, :, :, 30], cmap='viridis')
2 plt.show()
```



Visualizing Intermediate Outputs

- You can also visualize all of the channels for every layer of the network



Visualizing Intermediate Outputs

- The first layer acts as a collection of edge detectors
- The later layers contain more abstract activations that are less visually interpretable
- Deeper layers carry less information about visual contents of the image, and more information related to the class of the image
- The sparsity of the activations increases with the depth of the layer
    - Blank activations mean the pattern encoded by that filter isn't found in the input image

Visualizing Filters

- Shows the visual pattern that each filter is meant to respond to
- This is done with gradient ascent in input space: applying gradient descent to the value of the input image to maximize the response of a specific filter, starting with a blank input image
- The resulting image will be one that the chosen filter is maximally responsive to
- Steps:
  - Build a loss function that maximizes the value of a given filter in a given convolution layer
  - Use stochastic gradient descent to adjust the values of the input image in order to maximize the activation value

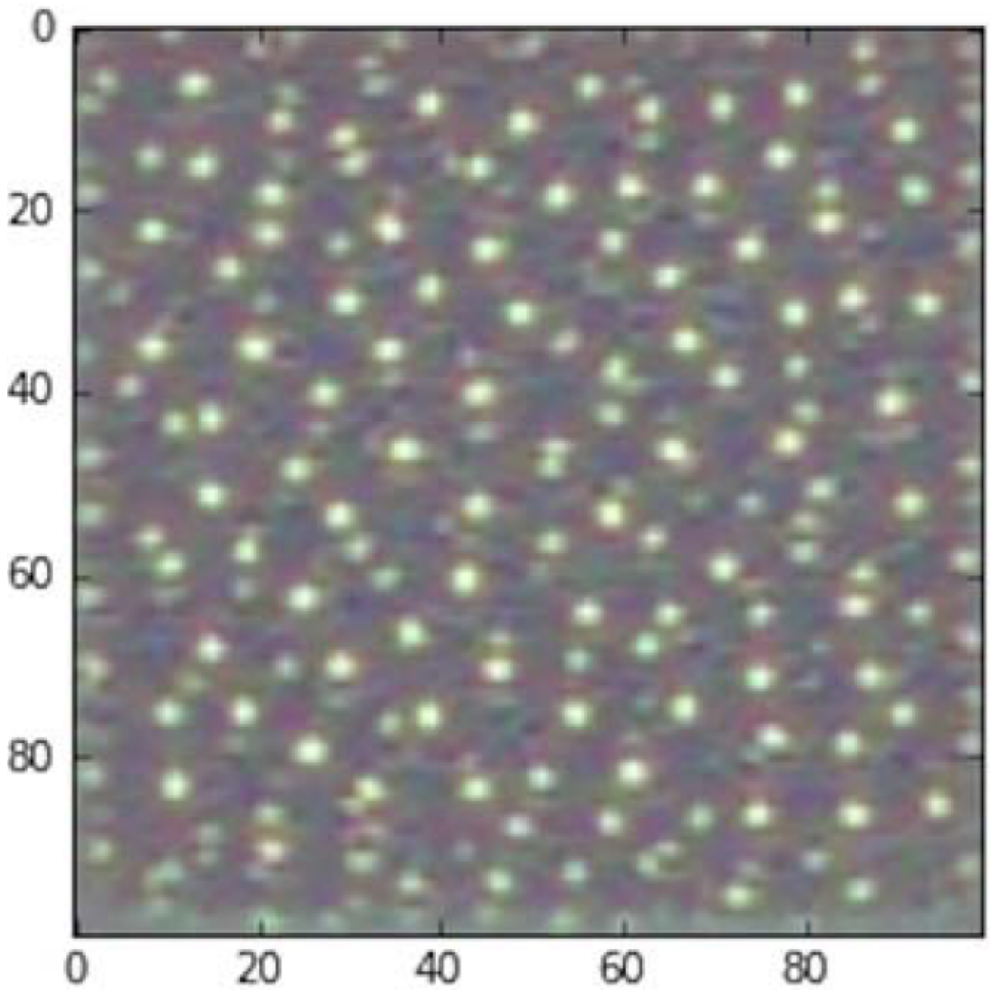
Visualizing Filters

```python
def generate_pattern(layer_name, filter_index, size=150):
    # Build a loss function that maximizes the activation
    # of the nth filter of the layer considered.
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])

    # Compute the gradient of the input picture wrt this loss
    grads = K.gradients(loss, model.input)[0]

    # Normalization trick: we normalize the gradient
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

    # This function returns the loss and grads given the input picture
    iterate = K.function([model.input], [loss, grads])

    # We start from a gray image with some noise
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.

    # Run gradient ascent for 40 steps
    step = 1.
    for i in range(40):
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step

    img = input_img_data[0]
    return deprocess_image(img)
```

Visualizing Filters

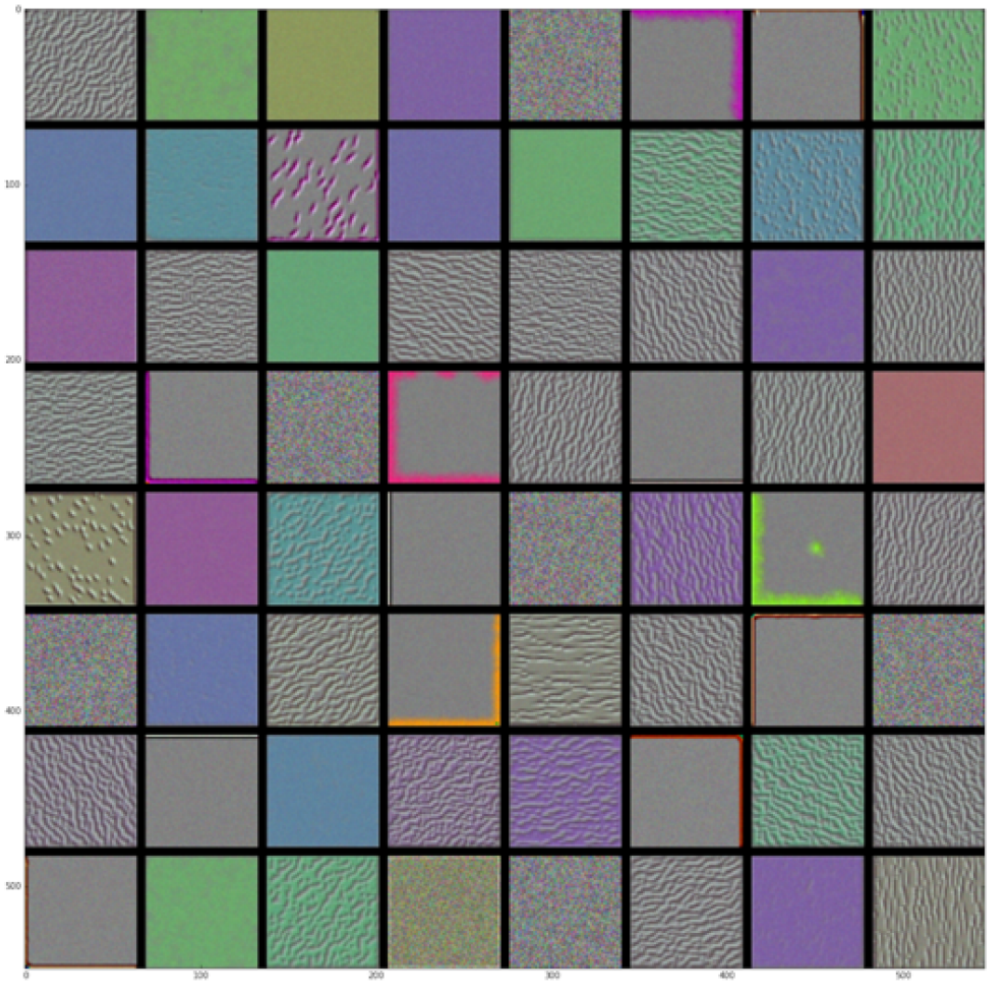
- It seems that filter 0 in layer `block3_conv1` is responsive to a polka-dot pattern

```
1 plt.imshow(generate_pattern('block3_conv1', 0))
2 plt.show()
```



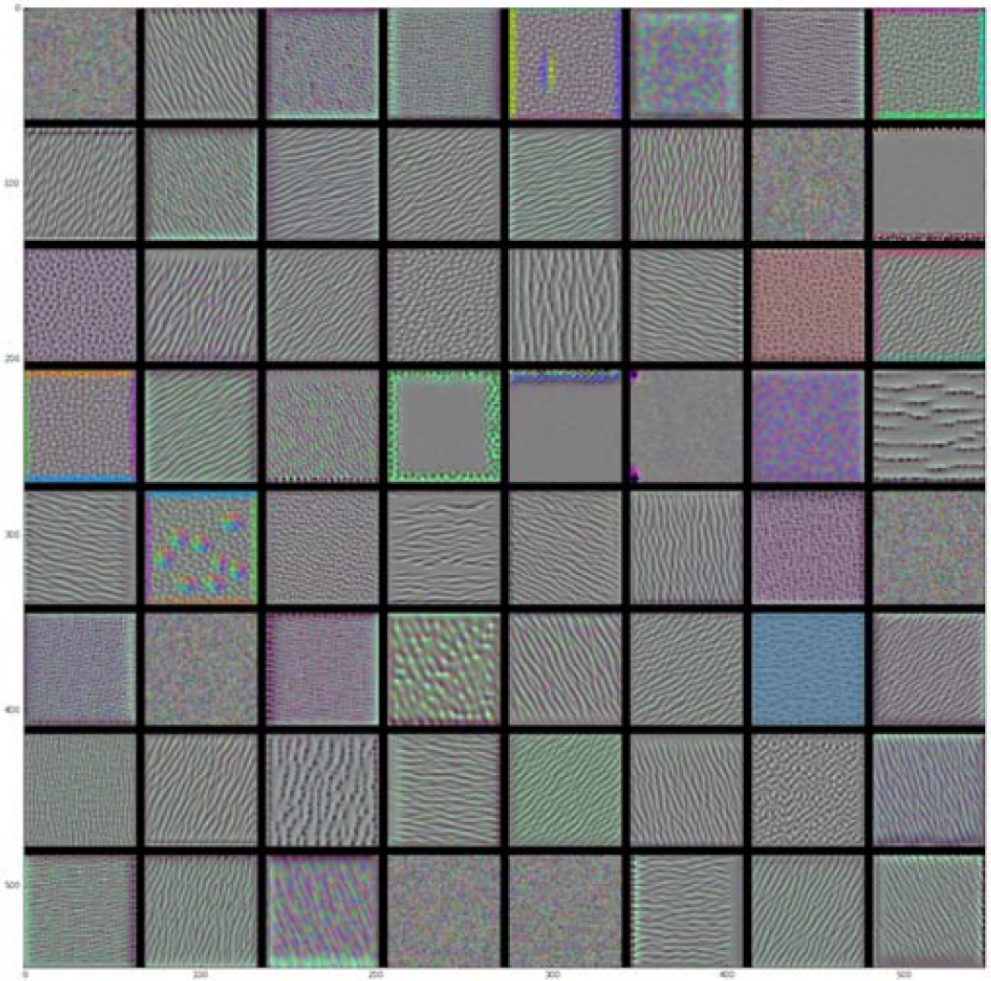
Visualizing Filters

- You can also visualize every every filter in every layer
- Below are the filter patterns for layer `block1_conv1`



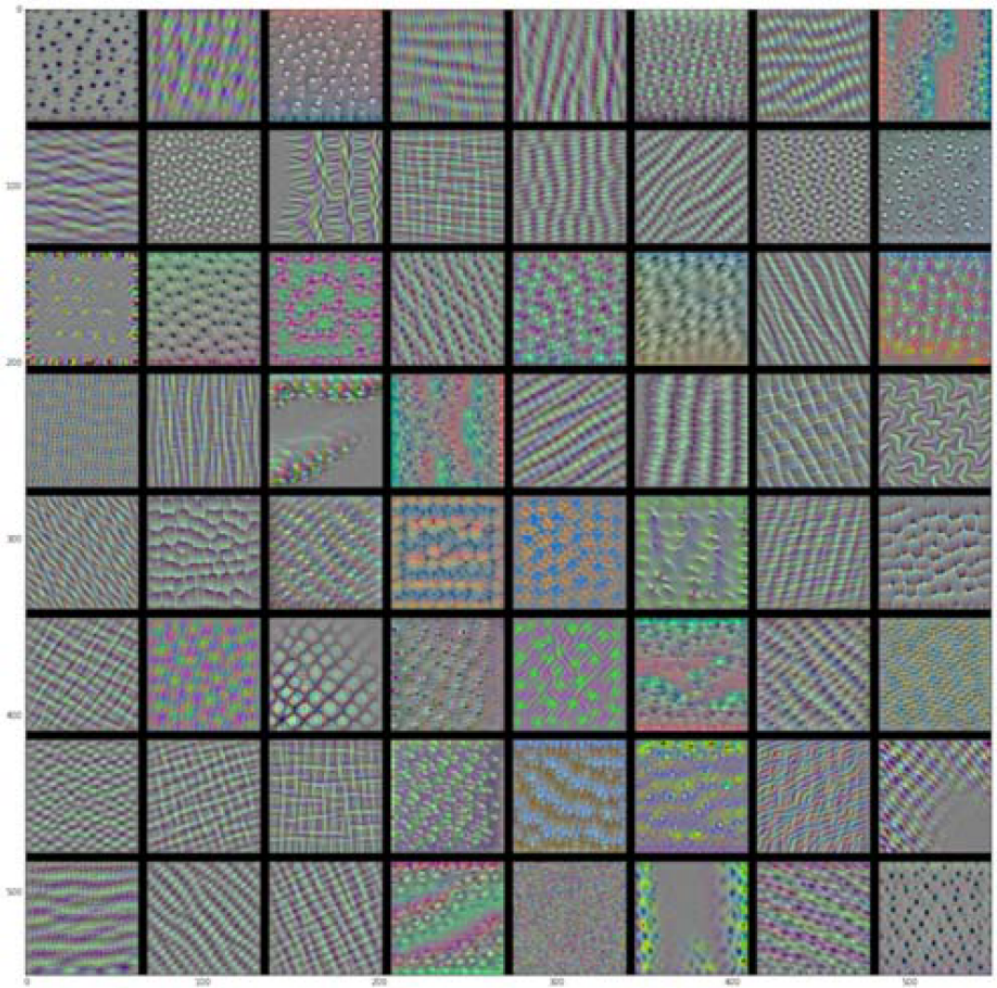
Visualizing Filters

- Below are the filter patterns for layer `block2_conv1`



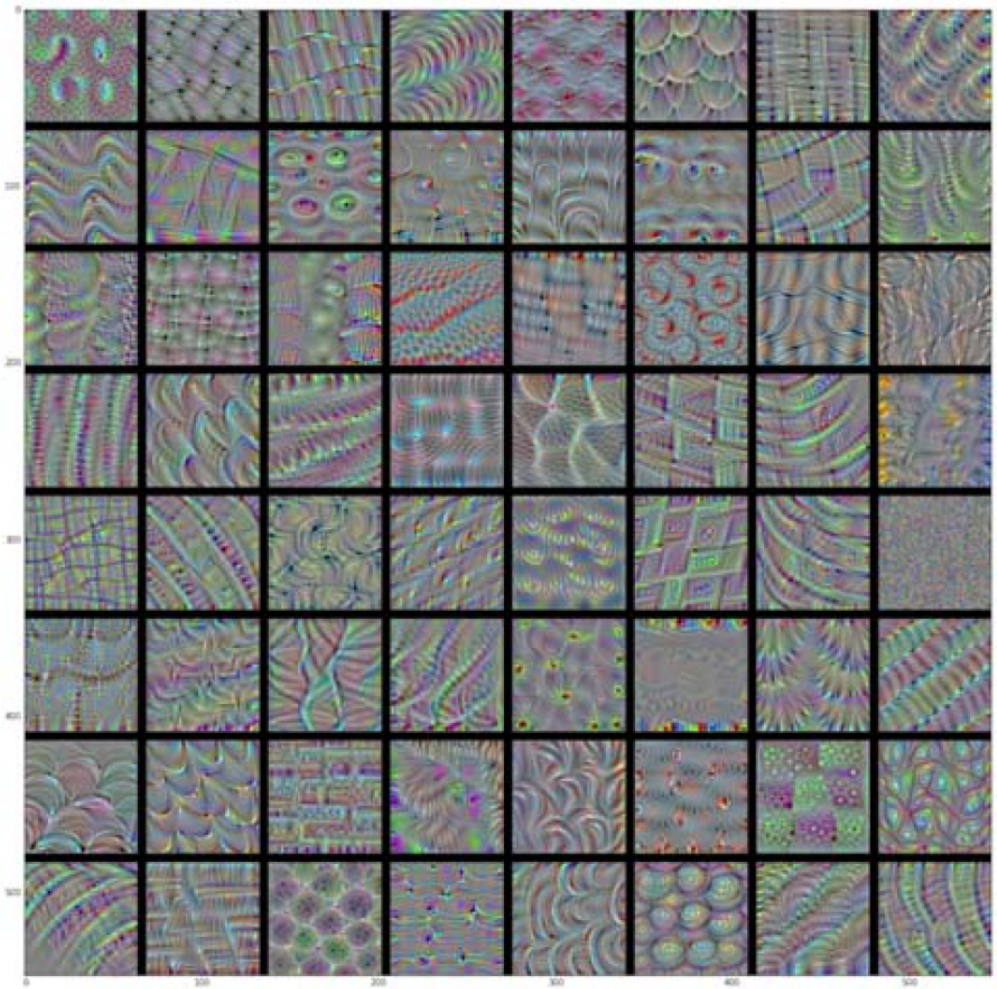
Visualizing Filters

- Below are the filter patterns for layer `block3_conv1`



Visualizing Filters

- Below are the filter patterns for layer `block4_conv1`



Visualizing Filters

- The filters get increasingly complex and refined as you go deeper in the model
- The filters from the first layer encode single directional edges and colors
- The next set of filters encode simple textures made from combinations of edges and colors-
- The filters in later layers resemble textures found in natural images - eyes, feathers, leaves, etc.

Visualizing Heatmaps of Class Activation

- Great for understanding which parts of an image led the network to its final classification
- Helpful for debugging the decision process
- This also allows you to locate specific objects in an image
- Called class activation map (CAM) visualization
- A class activation heatmap is a 2D grid of scores associated with a specific output class, computed for every location in an input image, indicating how important each location is with respect to the class under consideration

Visualizing Heatmaps of Class Activation

Visualizing Heatmaps of Class Activation

```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

# Note that we are including the densely-connected classifier on top;
# all previous times, we were discarding it.
model = VGG16(weights='imagenet')

# The local path to our target image
img_path = '/Users/fchollet/Downloads/creative_commons_elephant.jpg'

# `img` is a PIL image of size 224x224
img = image.load_img(img_path, target_size=(224, 224))

# `x` is a float32 Numpy array of shape (224, 224, 3)
x = image.img_to_array(img)

# We add a dimension to transform our array into a "batch"
# of size (1, 224, 224, 3)
x = np.expand_dims(x, axis=0)

# Finally we preprocess the batch
# (this does channel-wise color normalization)
x = preprocess_input(x)
```

Visualizing Heatmaps of Class Activation

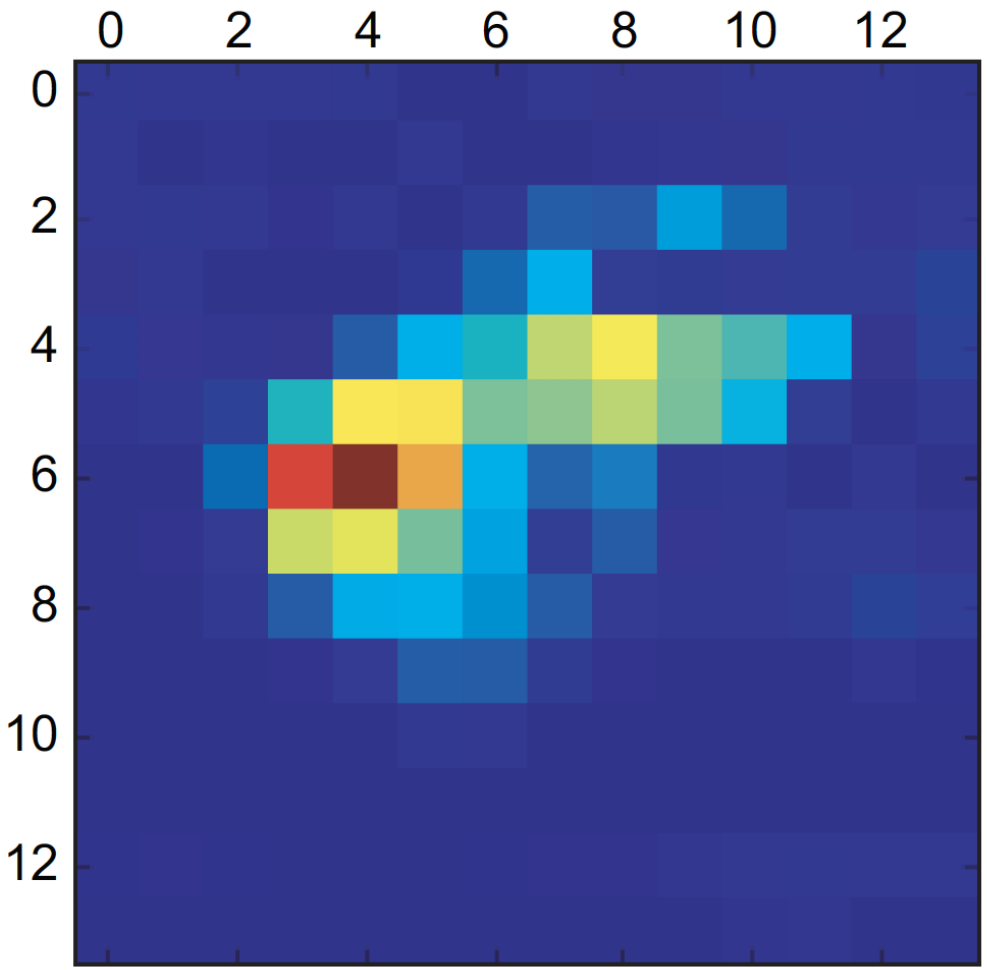
```
preds = model.predict(x)
print('Predicted:', decode_predictions(preds, top=3)[0])

# The top-3 classes predicted for this image are: African elephant, Tusker,
    Indian elephant

np.argmax(preds[0])
# This is the "african elephant" entry in the prediction vector
african_elephant_output = model.output[:, 386]

# The is the output feature map of the `block5_conv3` layer,
# the last convolutional layer in VGG16
last_conv_layer = model.get_layer('block5_conv3')

# This is the gradient of the "african elephant" class with regard to
# the output feature map of `block5_conv3`
grads = K.gradients(african_elephant_output, last_conv_layer.output)[0]

# This is a vector of shape (512,), where each entry
# is the mean intensity of the gradient over a specific feature map channel
pooled_grads = K.mean(grads, axis=(0, 1, 2))
```

Visualizing Heatmaps of Class Activation

```
# This function allows us to access the values of the quantities we just defined
    :
# `pooled_grads` and the output feature map of `block5_conv3`,
# given a sample image
iterate = K.function([model.input], [pooled_grads, last_conv_layer.output[0]])

# These are the values of these two quantities, as Numpy arrays,
# given our sample image of two elephants
pooled_grads_value, conv_layer_output_value = iterate([x])

# We multiply each channel in the feature map array
# by "how important this channel is" with regard to the elephant class
for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]

# The channel-wise mean of the resulting feature map
# is our heatmap of class activation
heatmap = np.mean(conv_layer_output_value, axis=-1)
```

Visualizing Heatmaps of Class Activation

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



Visualizing Heatmaps of Class Activation

```
import cv2

# We use cv2 to load the original image
img = cv2.imread(img_path)

# We resize the heatmap to have the same size as the original image
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

# We convert the heatmap to RGB
heatmap = np.uint8(255 * heatmap)

# We apply the heatmap to the original image
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

# 0.4 here is a heatmap intensity factor
superimposed_img = heatmap * 0.4 + img

# Save the image to disk
cv2.imwrite('/Users/fchollet/Downloads/elephant_cam.jpg', superimposed_img)
```

Visualizing Heatmaps of Class Activation

Object Localization and Detection

Localization and Detection

Image Classification

"Car"

Classification with Localization

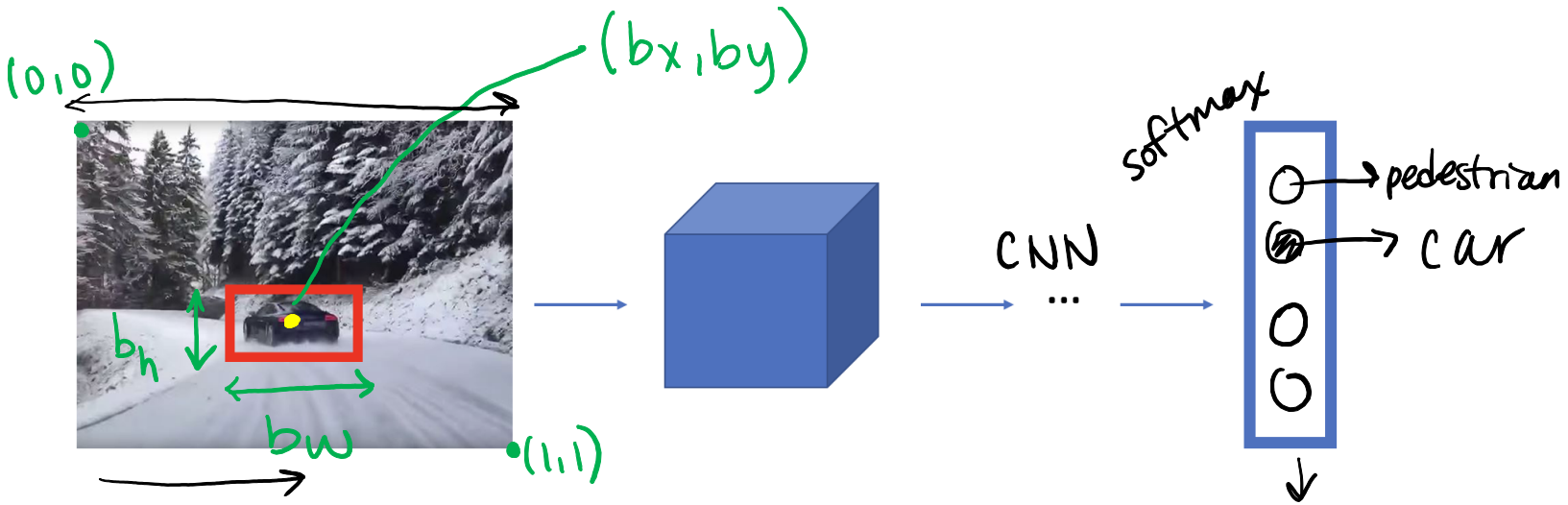
"Car" but where?

1 object

Detection

multiple objects - can be from different classes

Localization and Detection



Classes

1. Motorcycle
2. Car
3. Pedestrian
4. Background (no object)

Bounding Box

$b_x \approx 0.5$
$b_y \approx 0.5$
$b_h \approx$ ~~0.3~~ 0.2
$b_w \approx$ ~~0.1~~ 0.3

output: $\begin{bmatrix} \text{class} \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

Classes

1. Motorcycle c_1
2. Car c_2
3. Pedestrian c_3
4. Background (no object)

label $y =$ $\begin{bmatrix} p_d \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ ← indicator object is present

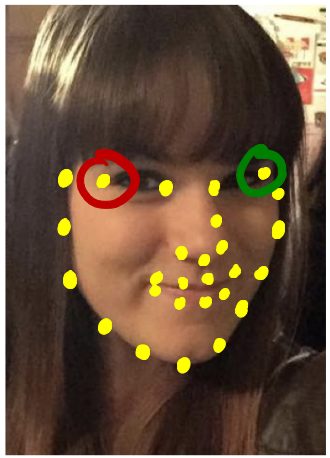
$y = \begin{bmatrix} 1 \\ 0.5 \\ 0.5 \\ 0.2 \\ 0.3 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

$y = \begin{bmatrix} 0 \\ ? \\ \vdots \\ ? \\ \vdots \\ ? \\ \vdots \\ ? \end{bmatrix}$ → numbers to be "ignored" → not relevant

$$\mathcal{L}(\hat{y}, y) = \begin{cases} [(\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \cdots + (\hat{y}_8 - y_8)^2] & \text{if } y_1 = 1, p_d = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0, p_d = 0 \end{cases}$$

"Landmark detection"

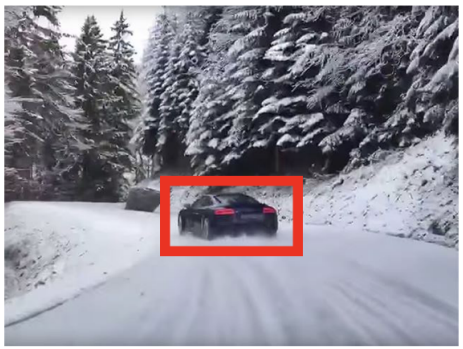
b_x, b_y, b_h, b_w



(l_{1x}, l_{1y})

(l_{2x}, l_{2y})

Localization and Detection



Localization and Detection

"pose detection"