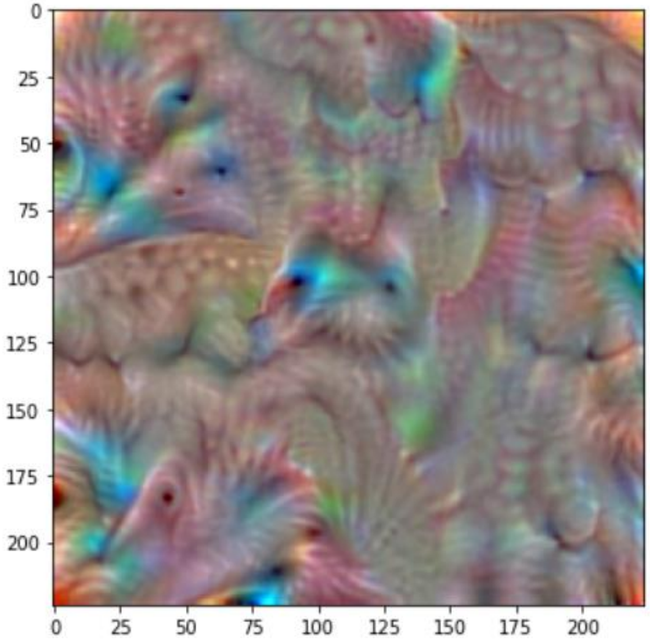
Convolutional Neural Networks: Visualization

Dr. Mongkol Ekpanyapong

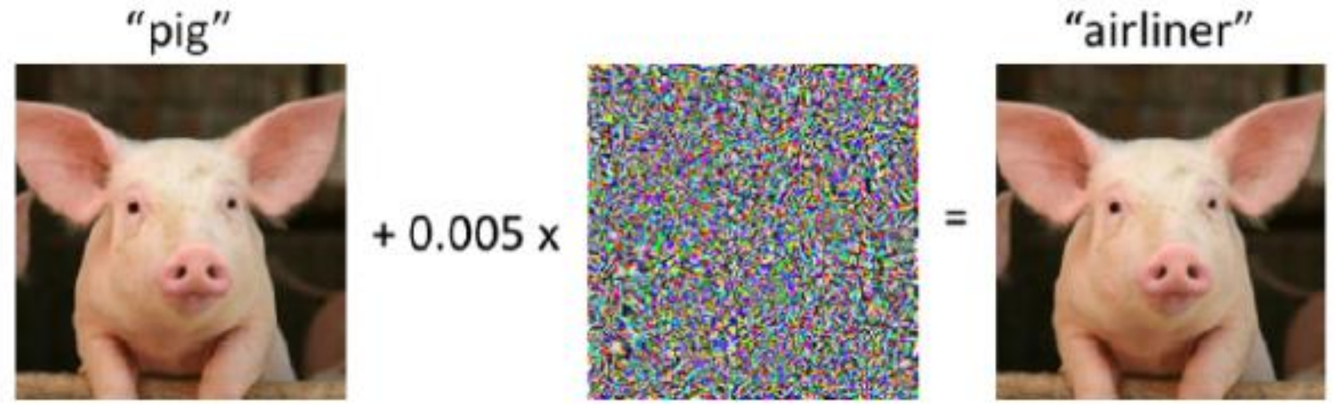
Class Maximization

- The right image is the one that maximize Bald Eagle on VGG

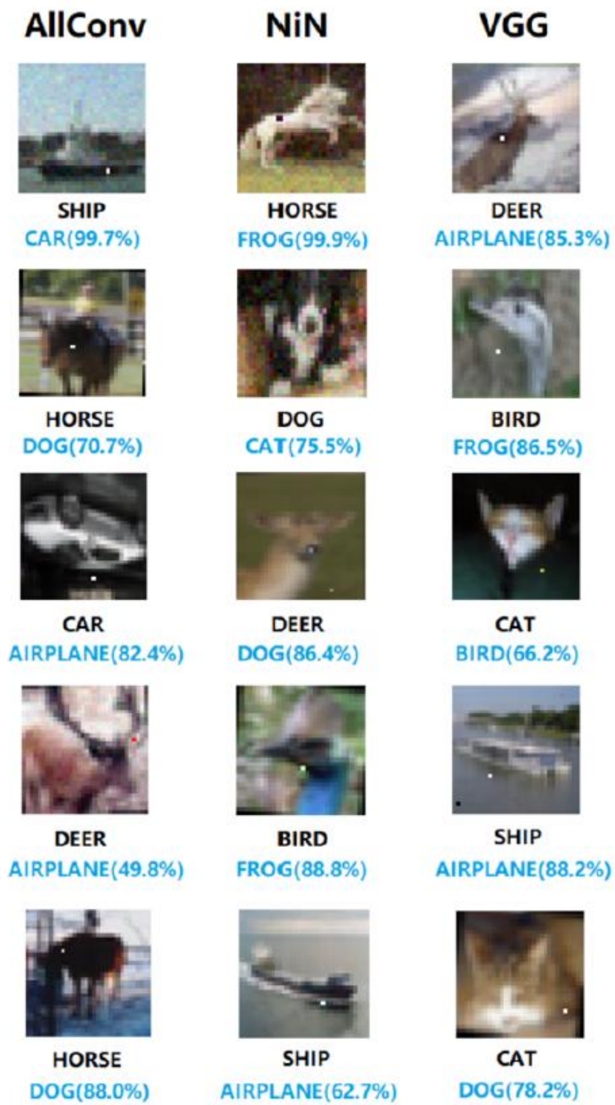


Fooling CNN

- Sometimes small changes in image can fool CNN Network



One
Pixel Attack



https://arxiv.org/pdf/1710.08864.pdf

Visualization

There are many tools available to help understand CNN models

- Visualizing intermediate CNN outputs
- Visualizing CNN filters
- Visualizing heatmaps of class activation

Visualizing intermediate activations

- Display the Network Architecture
  - model.summary can print architecture info.

- Display feature maps that are output by convolution and pooling layers (aka. activation layers)

Model Information

```
>>> from keras.models import load_model
>>> model = load_model('cats_and_dogs_small_2.h5')
>>> model.summary()  <1> As a reminder.
________________________________________________________________
Layer (type)                     Output Shape          Param #
================================================================
________________________________________________________________
conv2d_7 (Conv2D)                (None, 34, 34, 128)   73856
________________________________________________________________
maxpooling2d_7 (MaxPooling2D)    (None, 17, 17, 128)   0
________________________________________________________________
conv2d_8 (Conv2D)                (None, 15, 15, 128)   147584
________________________________________________________________
maxpooling2d_8 (MaxPooling2D)    (None, 7, 7, 128)     0
________________________________________________________________
flatten_2 (Flatten)              (None, 6272)          0
________________________________________________________________
dropout_1 (Dropout)              (None, 6272)          0
________________________________________________________________
dense_3 (Dense)                  (None, 512)           3211776
________________________________________________________________
dense_4 (Dense)                  (None, 1)             513
================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

Preprocessing a single image

```
img_path = '/Users/fchollet/Downloads/cats_and_dogs_small/test/cats/cat.1700.jpg'

from keras.preprocessing import image
import numpy as np

img = image.load_img(img_path, target_size=(150, 150))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
img_tensor /= 255.

<1> Its shape is (1, 150, 150, 3)
print(img_tensor.shape)
```

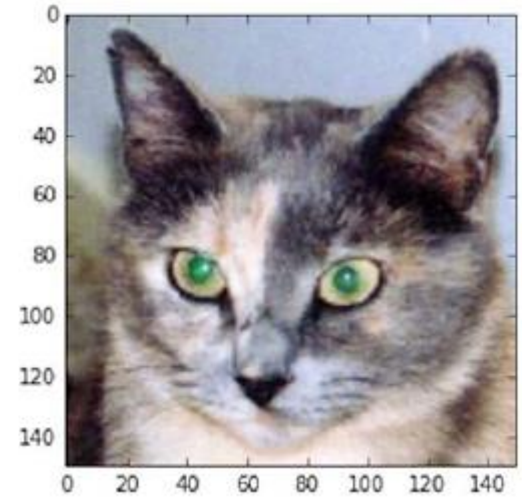
Preprocesses the image into a 4D tensor

Remember that the model was trained on inputs that were preprocessed this way.

```
import matplotlib.pyplot as plt

plt.imshow(img_tensor[0])
plt.show()
```

Image Example



Keras Class Model

- We use Keras Class Model instead of Sequential (single input single output) for single input, multiple outputs

```
from keras import models

layer_outputs = [layer.output for layer in model.layers[:8]]
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

Extracts the outputs of the top eight layers

Creates a model that will return these outputs, given the model input

```
activations = activation_model.predict(img_tensor)
```

Implementation

- Create a new model that takes an image as input and output feature maps
- Load our image and normalize it
- Extract the features map that we want and plot it

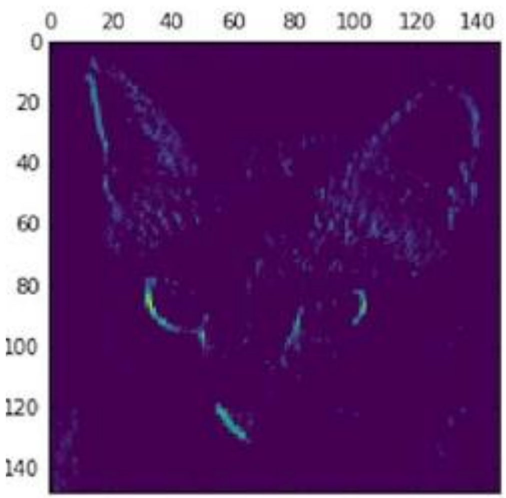
Example of activation layer of the fourth channel of the first layer

cmap='viridis' is one of the color style

```
>>> first_layer_activation = activations[0]
>>> print(first_layer_activation.shape)
(1, 148, 148, 32)

import matplotlib.pyplot as plt

plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')
```

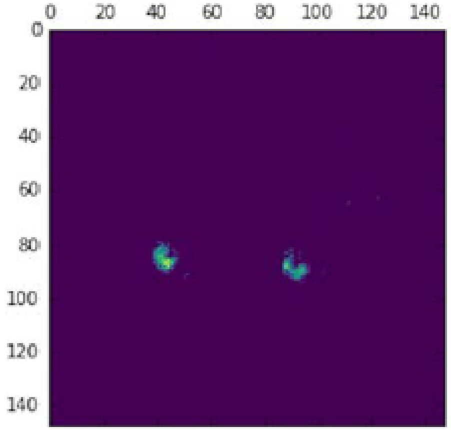


this channel filter diagonal edge detection

Another example

This layer encodes bright dot detctor

```
plt.matshow(first_layer_activation[0, :, :, 7], cmap='viridis')
```



```
import keras
keras.__version__

from keras.models import load_model
model = load_model('cats_and_dogs_small_2.h5')
model.summary()  # As a reminder.
img_path = './images/cat.1700.jpg'
from keras.preprocessing import image
import numpy as np
img = image.load_img(img_path, target_size=(150, 150))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
img_tensor /= 255.
print(img_tensor.shape)

import matplotlib.pyplot as plt
plt.imshow(img_tensor[0])
plt.show()
```

```
from keras import models

# Extracts the outputs of the top 8 layers:
layer_outputs = [layer.output for layer in model.layers[:8]]
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
activations = activation_model.predict(img_tensor)

first_layer_activation = activations[0]
print(first_layer_activation.shape)

import matplotlib.pyplot as plt

 plt.matshow(first_layer_activation[0, :, :, 3], cmap='viridis')
plt.show()
plt.matshow(first_layer_activation[0, :, :, 30], cmap='viridis')
plt.show()
```

Complete code

```
layer_names = []
for layer in model.layers[:8]:
    layer_names.append(layer.name)

images_per_row = 16

for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1]

    size = layer_activation.shape[1]

    n_cols = n_features // images_per_row
    display_grid = np.zeros((size * n_cols, images_per_row * size))

    for col in range(n_cols):
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                             :, :,
                                             col * images_per_row + row]
            channel_image -= channel_image.mean()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
            display_grid[col * size : (col + 1) * size,
                         row * size : (row + 1) * size] = channel_image

    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

Names of the layers, so you can have them as part of your plot

Displays the feature maps

Number of features in the feature map

The feature map has shape (l, size, size, n_features).

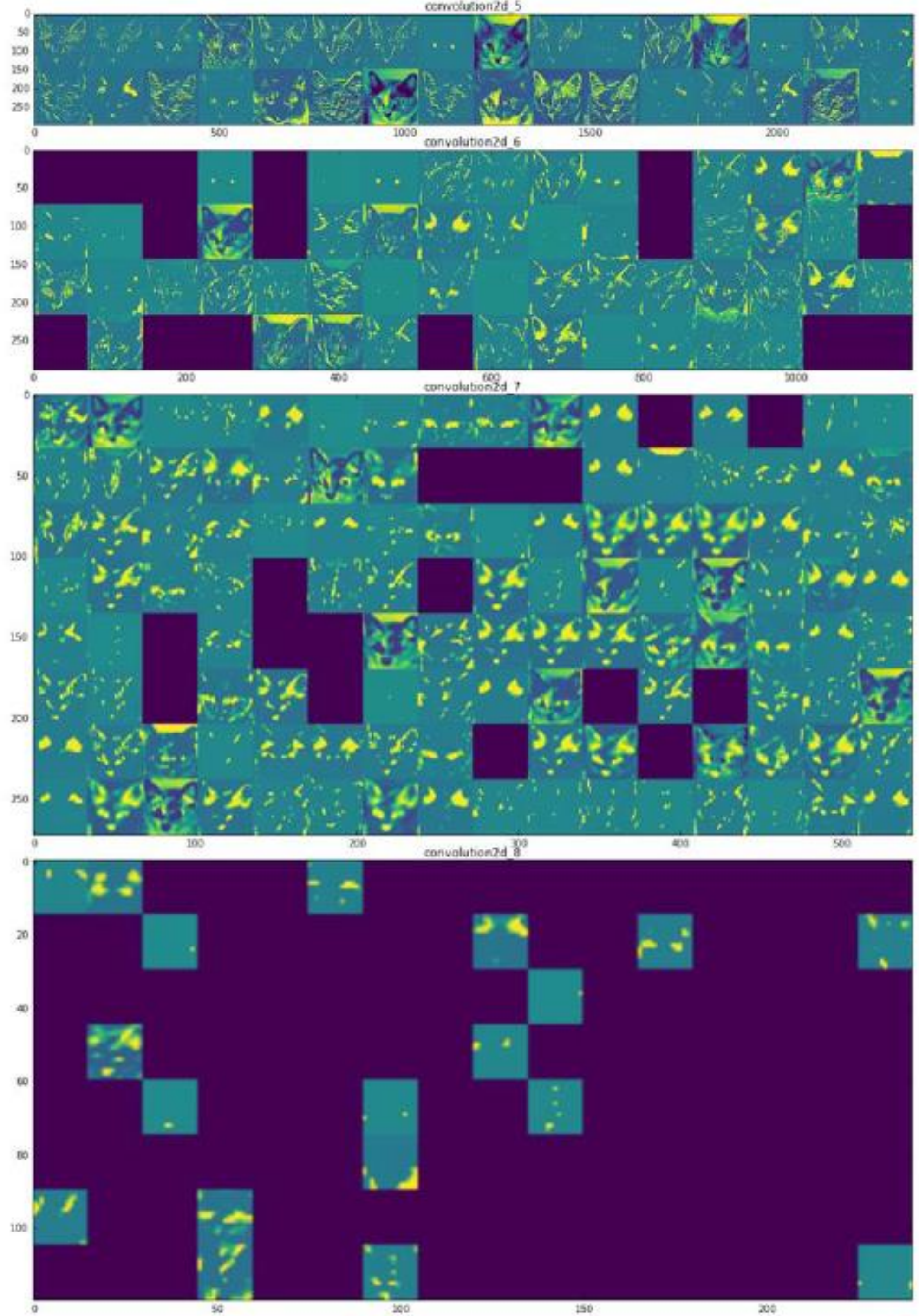
Tiles the activation channels in this matrix

Tiles each filter into a big horizontal grid

Post-processes the feature to make it visually palatable

Displays the grid

Output


convolution2d_5
convolution2d_6
convolution2d_7
convolution2d_8

Things to note

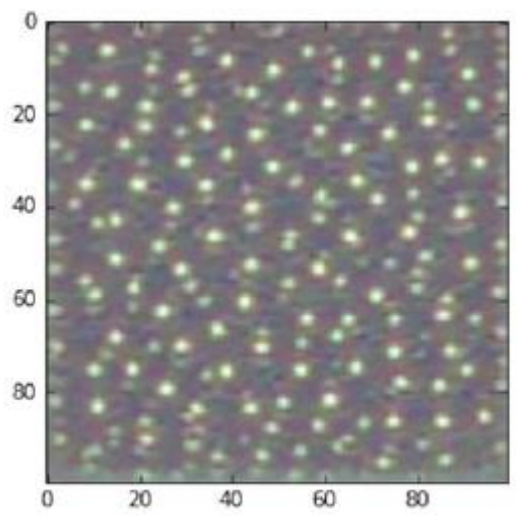
- The first layer act as a collection of various edge detectors
- As we go higher, the activations become increasing abstract and less visually interpretable
- The sparsity of the activations increases with the depth of the layer

Visualizing CNN layers

- To see the CNN filter, first we obtain the weights and bias of a filter (use Keras get_weights() function)
- Normalize weights between 0 and 1
- Plot the weight values

Output

```
>>> plt.imshow(generate_pattern('block3_conv1', 0))
```



Generate a grid of all filters

```
layer_name = 'block1_conv1'
size = 64
margin = 5

results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3))

for i in range(8):
    for j in range(8):
        filter_img = generate_pattern(layer_name, i + (j * 8), size=size)

        horizontal_start = i * size + i * margin
        horizontal_end = horizontal_start + size
        vertical_start = j * size + j * margin
        vertical_end = vertical_start + size
        results[horizontal_start: horizontal_end,
                vertical_start: vertical_end, :] = filter_img

plt.figure(figsize=(20, 20))
plt.imshow(results)
```

Empty (black) image to store results

Iterates over the rows of the results grid

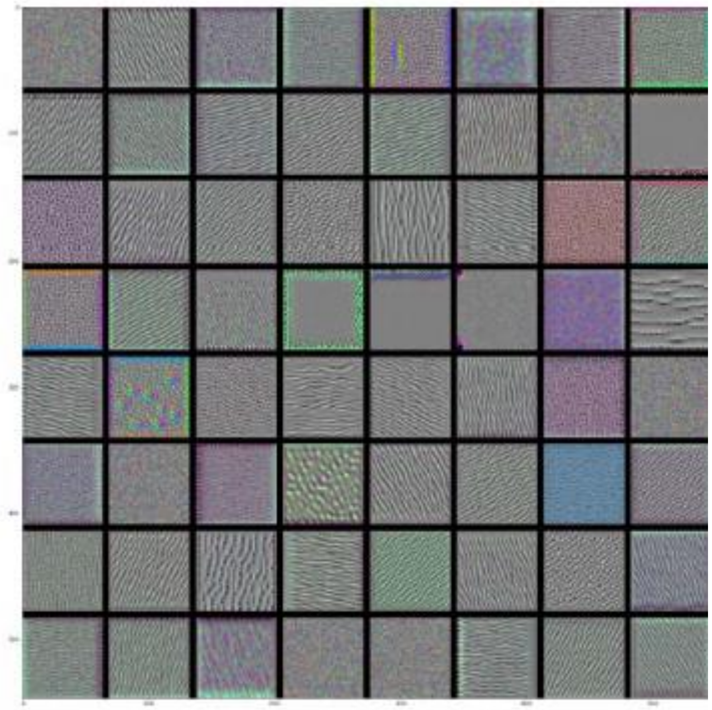
Iterates over the columns of the results grid

Generates the pattern for filter i + (j * 8) in layer_name

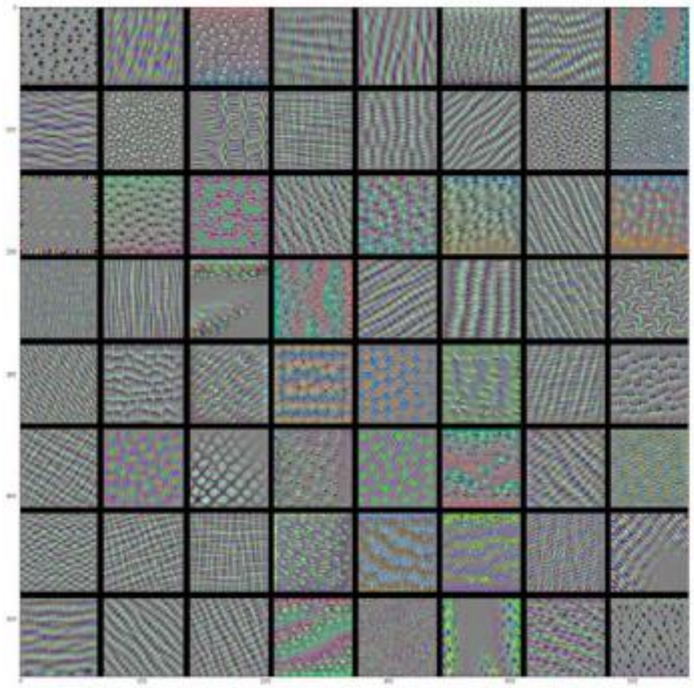
Puts the result in the square (i, j) of the results grid

Displays the results grid

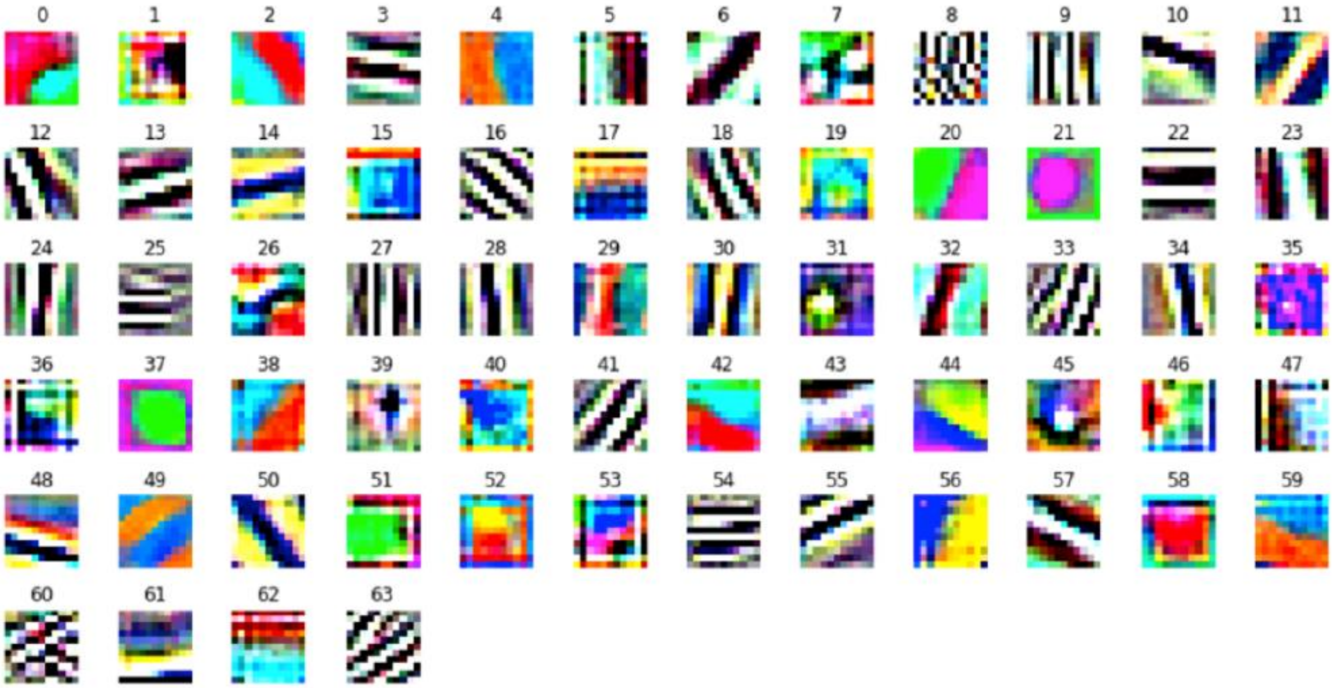
Filter output: block1_CNN1



Filter Output



More Examples



Filters from first convolution layer in AlexNet

https://towardsdatascience.com/visualizing-convolution-neural-networks-using-pytorch-3dfa8443e74e

Notes

- Filters from the first layer encode simple directional edge and colors
- The filters from second layer encode simple textures made from combinations of edges and colors
- Filter in higher layer begin to resemble textures frond in natural images, feathers, eyes, leaves, and so on

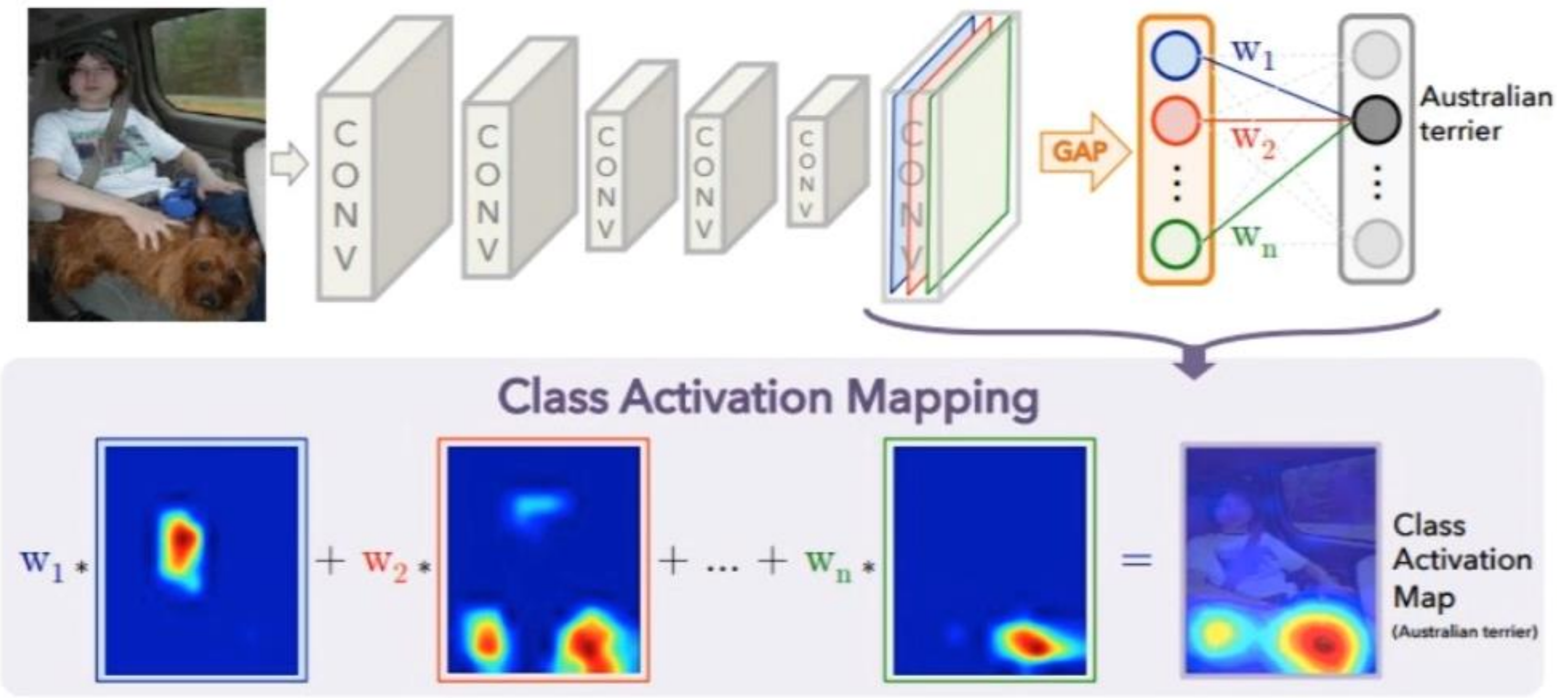
Identify the critical regions

- We can use heatmap to visualize the image for classification
- This technique is called class activation map

(CAM)

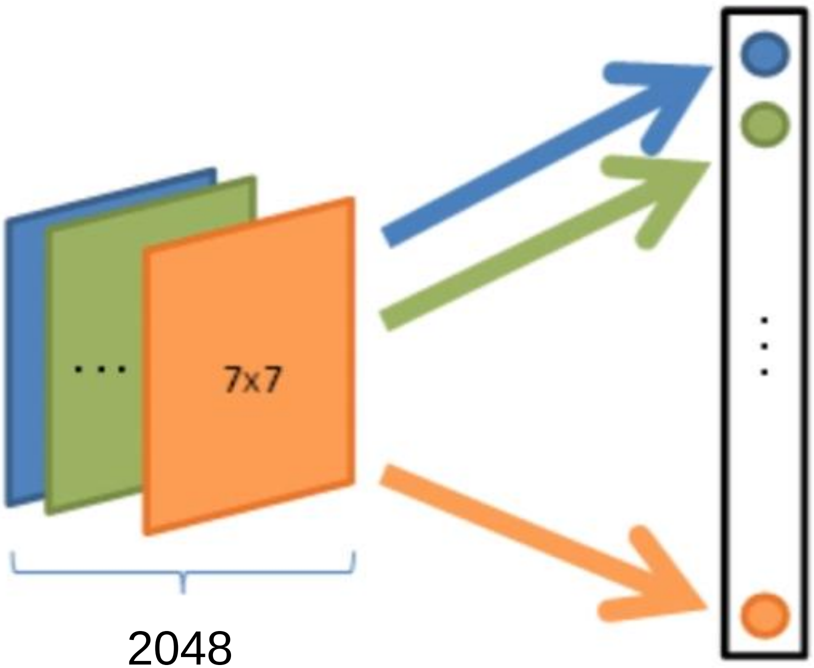
Image from Deep Learning with Python book

ResNet Model



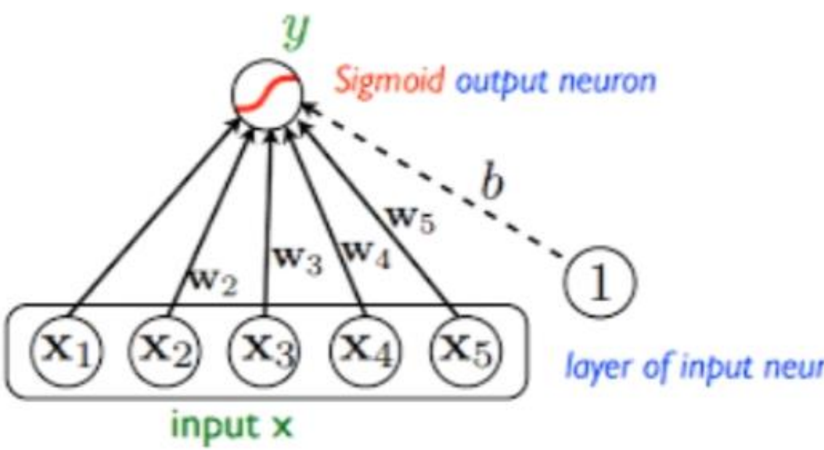
GAP = global average pooling

Last layer



2048

2048



if the input x is high, then it is likely that image belong to this class

ResNet

activation_49 (Activation)	(None, 7, 7, 2048)	0	add_16[0][0]
avg_pool (AveragePooling2D)	(None, 1, 1, 2048)	0	activation_49[0][0]
flatten_1 (Flatten)	(None, 2048)	0	avg_pool[0][0]
fc1000 (Dense)	(None, 1000)	2049000	flatten_1[0][0]

Class Activation Map

For final predicted class,

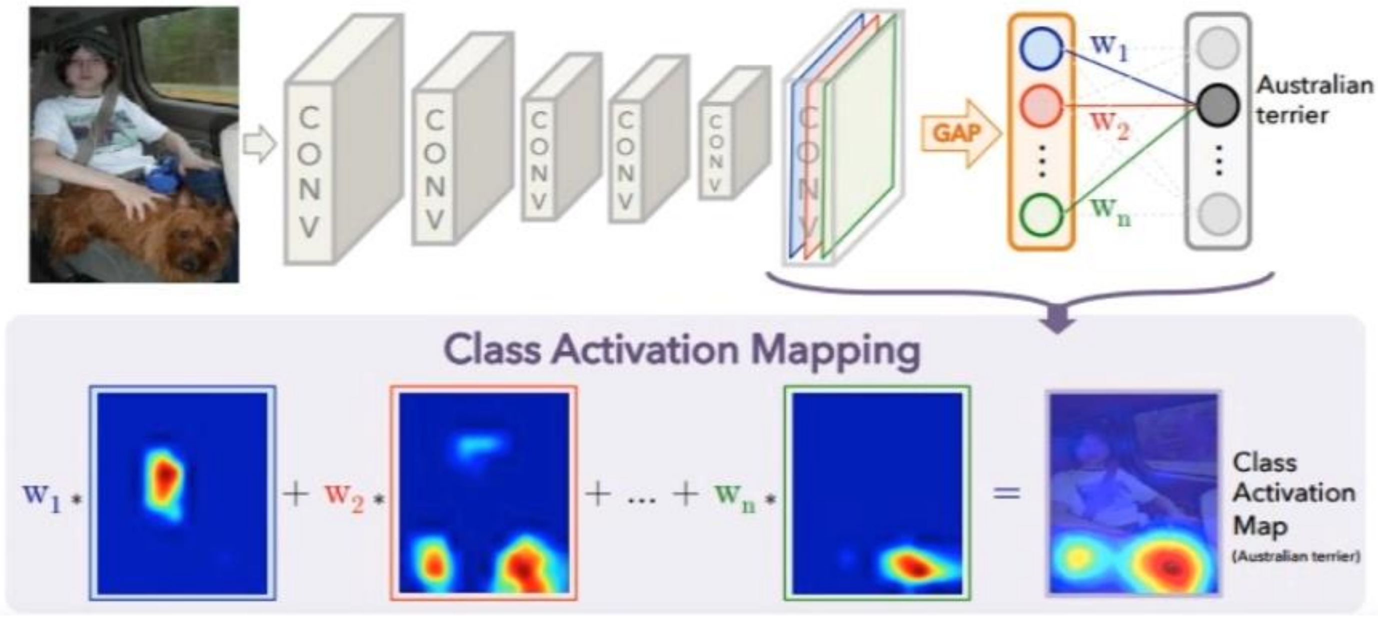
w = W[:; Australian terrier] # size 2048

Before the last layer:

F = 2048 channels 7 x 7 images

Class Activation Map =
F[0]*w[0]+F[1]*w[1]+…+F[2047]w[2047]

Class Activation Map Output



Example

Assume we have a feature map with a shape of [4x4x3]

Channel depth = 3

Gradient

Channel1 :

```csharp
[0.1, 0.2, 0.3, 0.4]
[0.2, 0.3, 0.4, 0.5]
[0.3, 0.4, 0.5, 0.6]
[0.4, 0.5, 0.6, 0.7]
```

Channel2:

```csharp
[0.5, 0.6, 0.7, 0.8]
[0.6, 0.7, 0.8, 0.9]
[0.7, 0.8, 0.9, 1.0]
[0.8, 0.9, 1.0, 1.1]
```

Channel3:

```csharp
[1.0, 1.1, 1.2, 1.3]
[1.1, 1.2, 1.3, 1.4]
[1.2, 1.3, 1.4, 1.5]
[1.3, 1.4, 1.5, 1.6]
```

Pool Gradient per Layer

Pooled Gradient per layer

= [0.45   0.8   1.3]

Load 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.

Test Picture

Preprocessing Inputs

```
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)
```

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)

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

Local path to the target image

Run the pretrained network

```
>>> 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)]
```

The top three classes predicted for this image are as follows:

- African elephant (with 92.5% probability)
- Tusker (with 7% probability)
- Indian elephant (with 0.4% probability)

Find maximally activated class

```
>>> np.argmax(preds[0])
386
```

Setting up the CAM algorithm

"African elephant" entry in the prediction vector

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

```
african_e66lephant_output = model.output[:, 386]

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

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

Heatmap post-processing

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

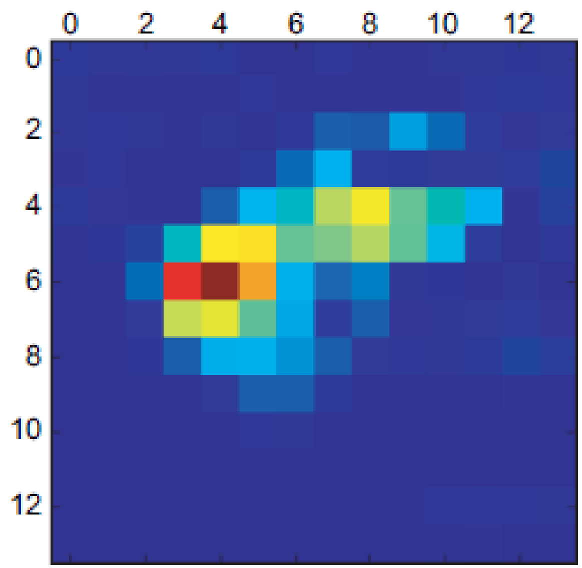


Figure 5.35 African elephant class activation heatmap over the test picture

Superimposing with original

```
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

Applies the heatmap to the original image

0.4 here is a heatmap intensity factor.

Saves the image to disk

Complete code

```
from keras.applications.vgg16 import VGG16
import matplotlib.pyplot as plt
from keras import backend as K

K.clear_session()
model = VGG16(weights='imagenet')
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

img_path = './images/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)

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

```
# This is the "african elephant" entry in the prediction vector
african_elephant_output = model.output[:, 386]
last_conv_layer = model.get_layer('block5_conv3')

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

```
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)

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

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

Questions?