

رَبِّ الْعَالَمِينَ



یادگیری عمیق

مدرس: محمدرضا محمدی

زمستان ۱۴۰۱

شبکه‌های عصبی کانولوشنی

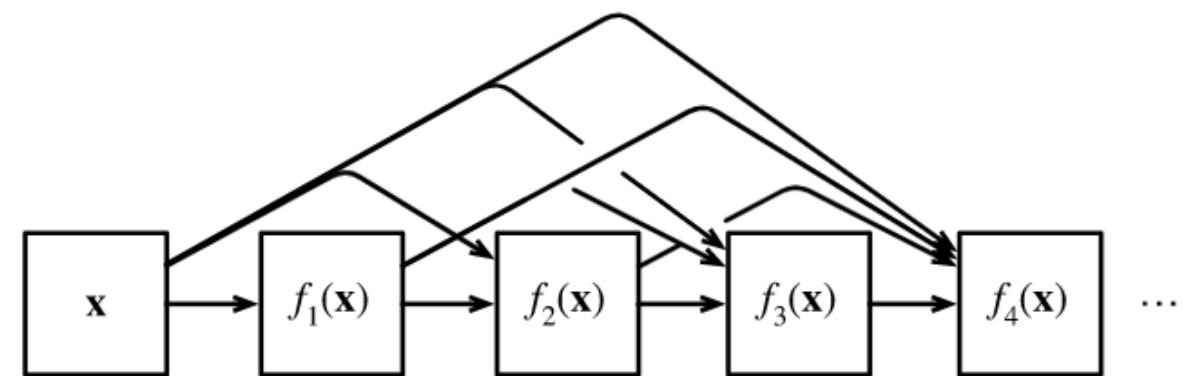
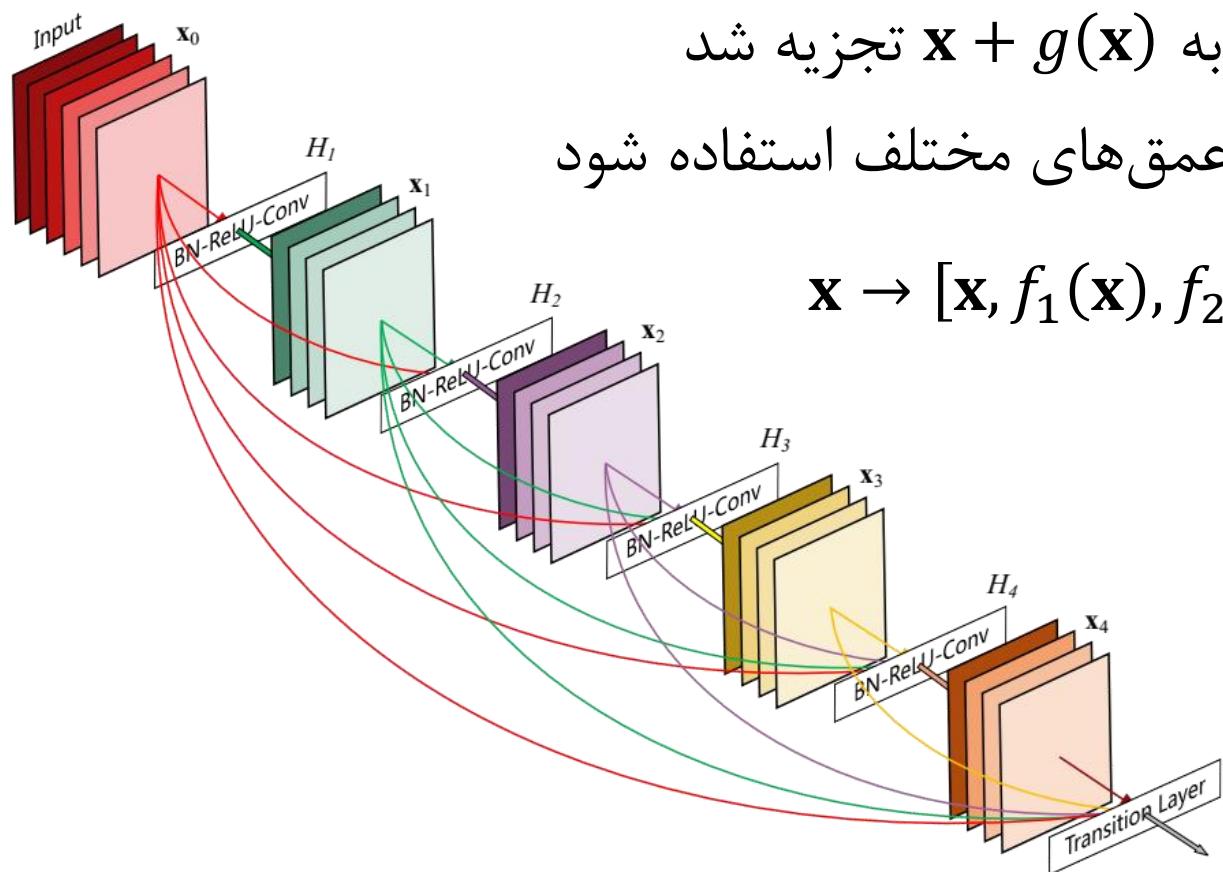
Convolutional Neural Networks

DenseNet

- در Inception از ایده فیلترهای موازی چند مقایسه استفاده شد
- در ResNet از ایده اتصال باقیمانده و یادگیری $f(\mathbf{x}) + g(\mathbf{x})$ به \mathbf{x} تجزیه شد
- در DenseNet پیشنهاد شده است تا از ویژگی‌های با عمق‌های مختلف استفاده شود

$$\mathbf{x} \rightarrow [\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), f_3([\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})])]), \dots]$$

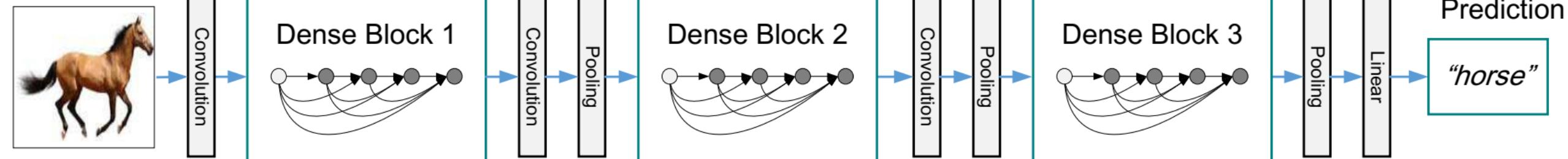
- نماد الحق (Concatenate) [,] است



DenseNet

Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112×112		7×7 conv, stride 2		
Pooling	56×56		3×3 max pool, stride 2		
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56			1×1 conv	
	28×28			2×2 average pool, stride 2	
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28			1×1 conv	
	14×14			2×2 average pool, stride 2	
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14			1×1 conv	
	7×7			2×2 average pool, stride 2	
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1			7×7 global average pool	
				1000D fully-connected, softmax	

Input



DenseNet

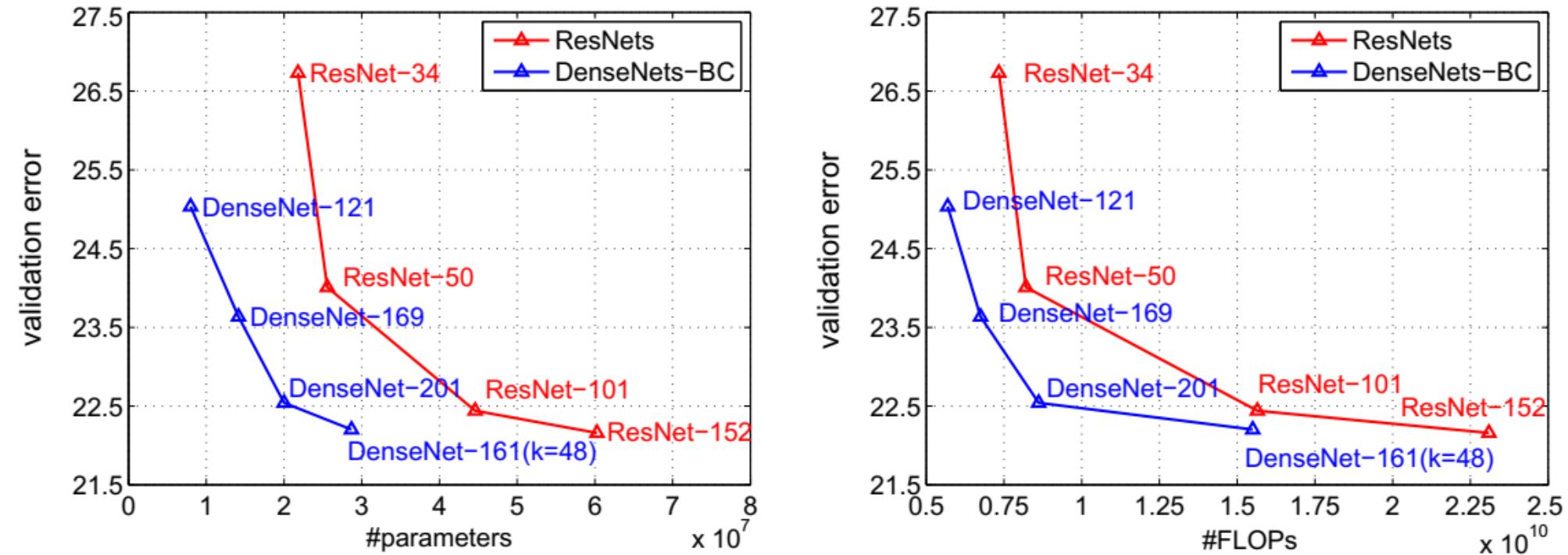


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

Models and pre-trained weights - x +

pytorch.org/vision/stable/models.html#classification

email Calendar Bale واتساب mali MVIP مجله فرمالو بانک قرض الحسن ر... جلسات ارتباط با صنعت face-analysis face-datasets ذرهبین EasyTrader تجارت پیگیر-چهره

main (0.14.7af+deed2f) ▾

Docs > Models and pre-trained weights

Shortcuts

Classification

The following classification models are available, with or without pre-trained weights:

- [AlexNet](#)
- [ConvNeXt](#)
- [DenseNet](#)
- [EfficientNet](#)
- [EfficientNetV2](#)
- [GoogLeNet](#)
- [Inception V3](#)
- [MaxVit](#)
- [MNASNet](#)
- [MobileNet V2](#)
- [MobileNet V3](#)
- [RegNet](#)
- [ResNet](#)
- [ResNeXt](#)
- [ShuffleNet V2](#)
- [SqueezeNet](#)
- [SwinTransformer](#)
- [VGG](#)
- [VisionTransformer](#)
- [Wide ResNet](#)

Models and pre-trained weights

- + General information on pre-trained we
- + Classification
- + Semantic Segmentation
- + Object Detection, Instance Segmentat
- Keypoint Detection
- + Video Classification
- Optical Flow

Search Docs

Package Reference

Transforming and augmenting images

Models and pre-trained weights

Datasets

Utils

Operators

Reading/Writing images and videos

Feature extraction for model inspection

Examples and training references

Example gallery

Training references

PyTorch Libraries

PyTorch

torchaudio

torchtext

torchvision

TorchElastic

TorchServe

PyTorch on XLA Devices

Here is an example of how to use the pre-trained image classification models:

```
from torchvision.io import read_image
```

https://pytorch.org/vision/stable/training_references.html

تصویرسازی

Visualization

شبکه چه آموخته است؟

- اغلب گفته می‌شود که مدل‌های یادگیری عمیق «جعبه سیاه» هستند
 - قطعاً برای شبکه‌های کانولوشنی درست نیست
- بازنمایی‌هایی که توسط شبکه‌های کانولوشنی آموخته می‌شوند قابلیت مصورسازی بالایی دارند
 - تا حد زیادی به این دلیل که بازنمایی مفاهیم بصری هستند



شبکه چه آموخته است؟

- روش‌های مختلفی برای مصورسازی یا تفسیر بازنمایی‌های آموخته شده توسط شبکه وجود دارد که سه روش رایج عبارتند از:
 - نمایش خروجی‌های لایه‌های میانی (فعال شدن لایه‌های میانی)
 - برای درک اینکه لایه‌های متوالی ورودی خود را چگونه تبدیل می‌کنند
 - نمایش فیلترهای آموخته شده
- برای درک دقیق الگوی بصری یا مفهومی که هر فیلتر آموخته است



شبکه چه آموخته است؟

- روش‌های مختلفی برای مصورسازی یا تفسیر بازنمایی‌های آموخته شده توسط شبکه وجود دارد که سه روش رایج عبارتند از:
 - نمایش خروجی‌های لایه‌های میانی (فعال شدن لایه‌های میانی)
 - برای درک اینکه لایه‌های متوالی ورودی خود را چگونه تبدیل می‌کنند
 - نمایش فیلترهای آموخته شده
- برای درک دقیق الگوی بصری یا مفهومی که هر فیلتر آموخته است
- نمایش نقشه‌های حرارتی (heatmaps) مربوط به فعالیت هر کلاس در یک تصویر
- برای درک اینکه کدام بخش از یک تصویر باعث شده است شبکه کلاس مربوطه را تشخیص دهد
- این امکان را می‌دهد که مکان اشیاء در تصاویر را تخمین زد

خروجی‌های لایه‌های میانی

- نمایش نقشه‌های ویژگی که توسط لایه‌های کانولوشنی و تجمعی مختلف در یک شبکه با توجه به ورودی مشخصی محاسبه می‌شوند
- نقشه‌های ویژگی دارای سه بعد عرض، ارتفاع و عمق (کanal) هستند



```
>>> from keras.models import load_model  
>>> model = load_model('cats_and_dogs_small_2.h5')  
>>> model.summary()
```

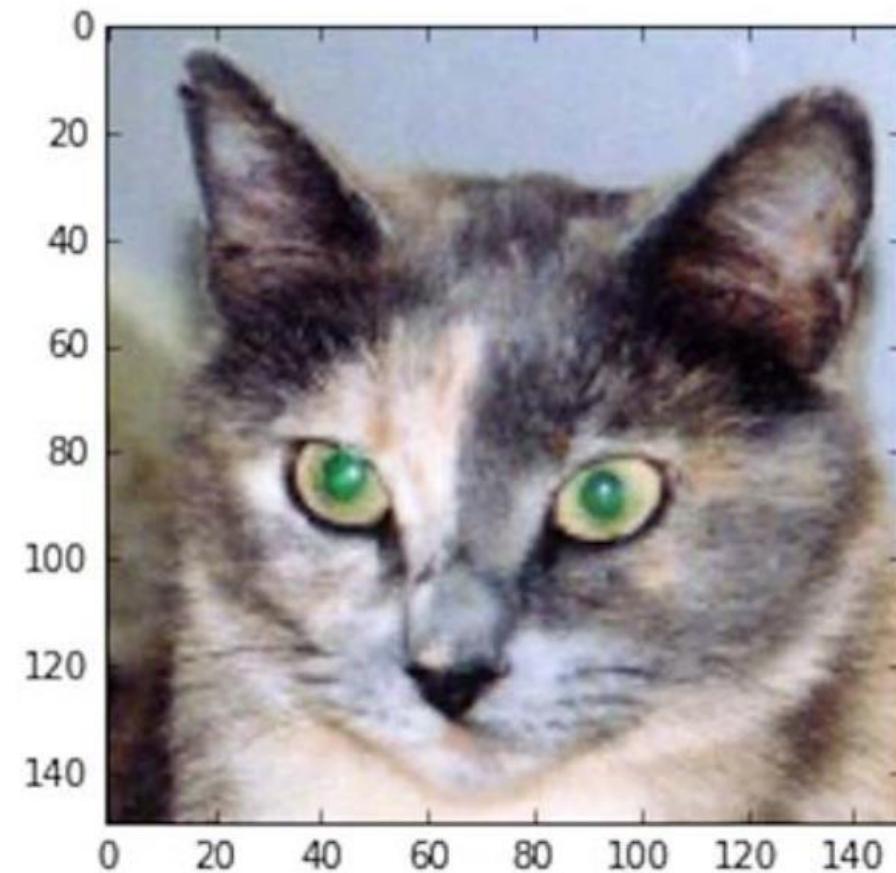
Layer (type)	Output Shape	Param #
<hr/>		
conv2d_5 (Conv2D)	(None, 148, 148, 32)	896
maxpooling2d_5 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_6 (Conv2D)	(None, 72, 72, 64)	18496
maxpooling2d_6 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_7 (Conv2D)	(None, 34, 34, 128)	73856
maxpooling2d_7 (MaxPooling2D)	(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
<hr/>		
Total params: 3,453,121		
Trainable params: 3,453,121		
Non-trainable params: 0		

Listing 5.25 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)  
  
import matplotlib.pyplot as plt  
  
plt.imshow(img_tensor[0])  
plt.show()
```

Preprocesses the image
into a 4D tensor

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



Listing 5.27 Instantiating a model from an input tensor and a list of output tensors

```
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

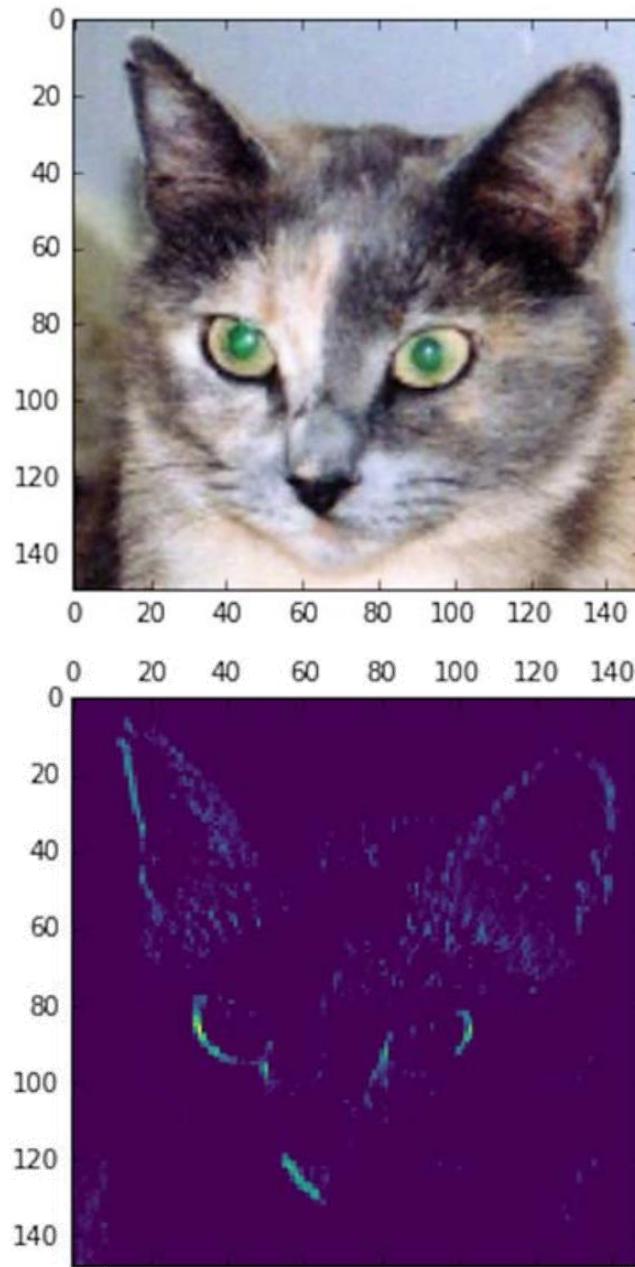
Listing 5.28 Running the model in predict mode

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

Reeight a list of five eight Numpy arrays: one array per layer activation

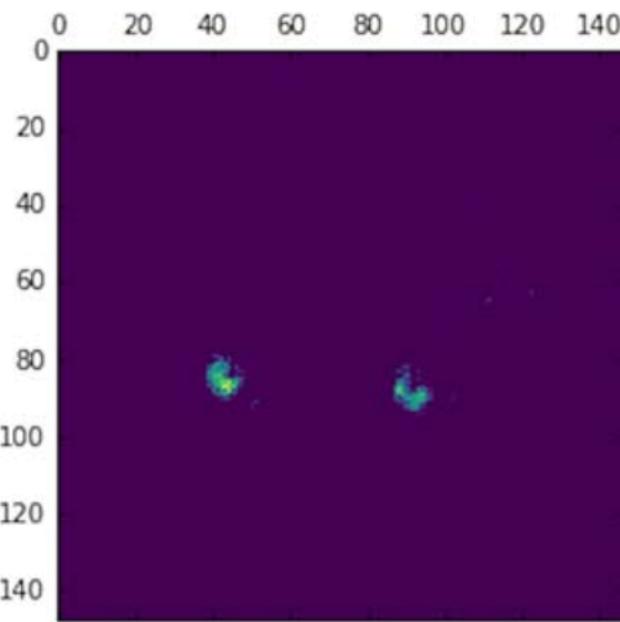
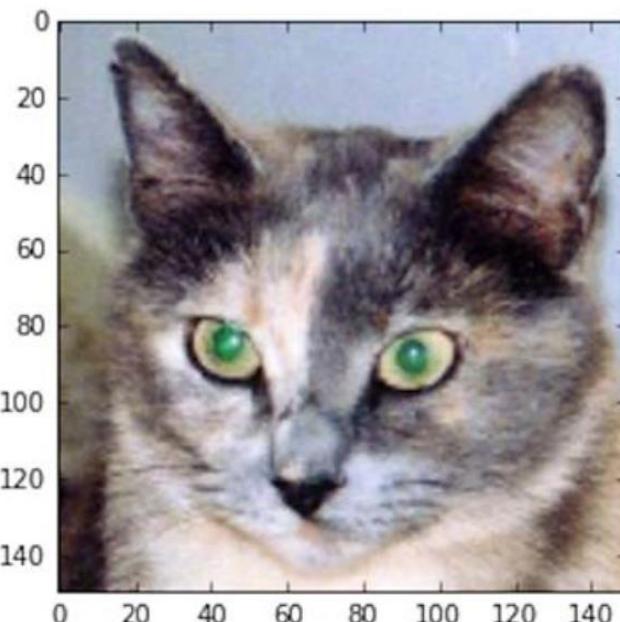
Listing 5.29 Visualizing the fourth channel

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



Listing 5.30 Visualizing the seventh channel

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



Listing 5.31 Visualizing every channel in every intermediate activation

```
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

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

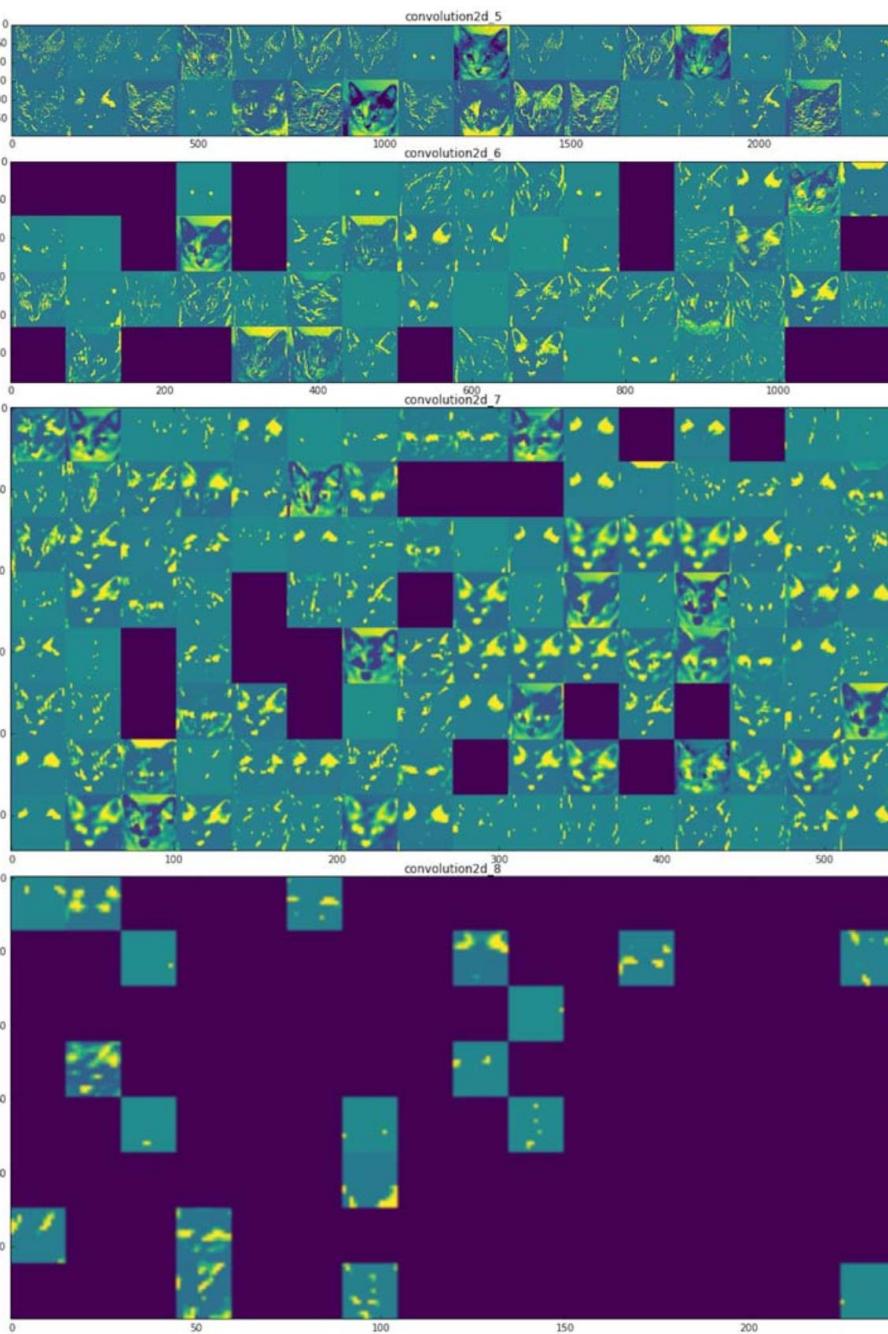
Tiles each filter into a big horizontal grid

Displays the grid

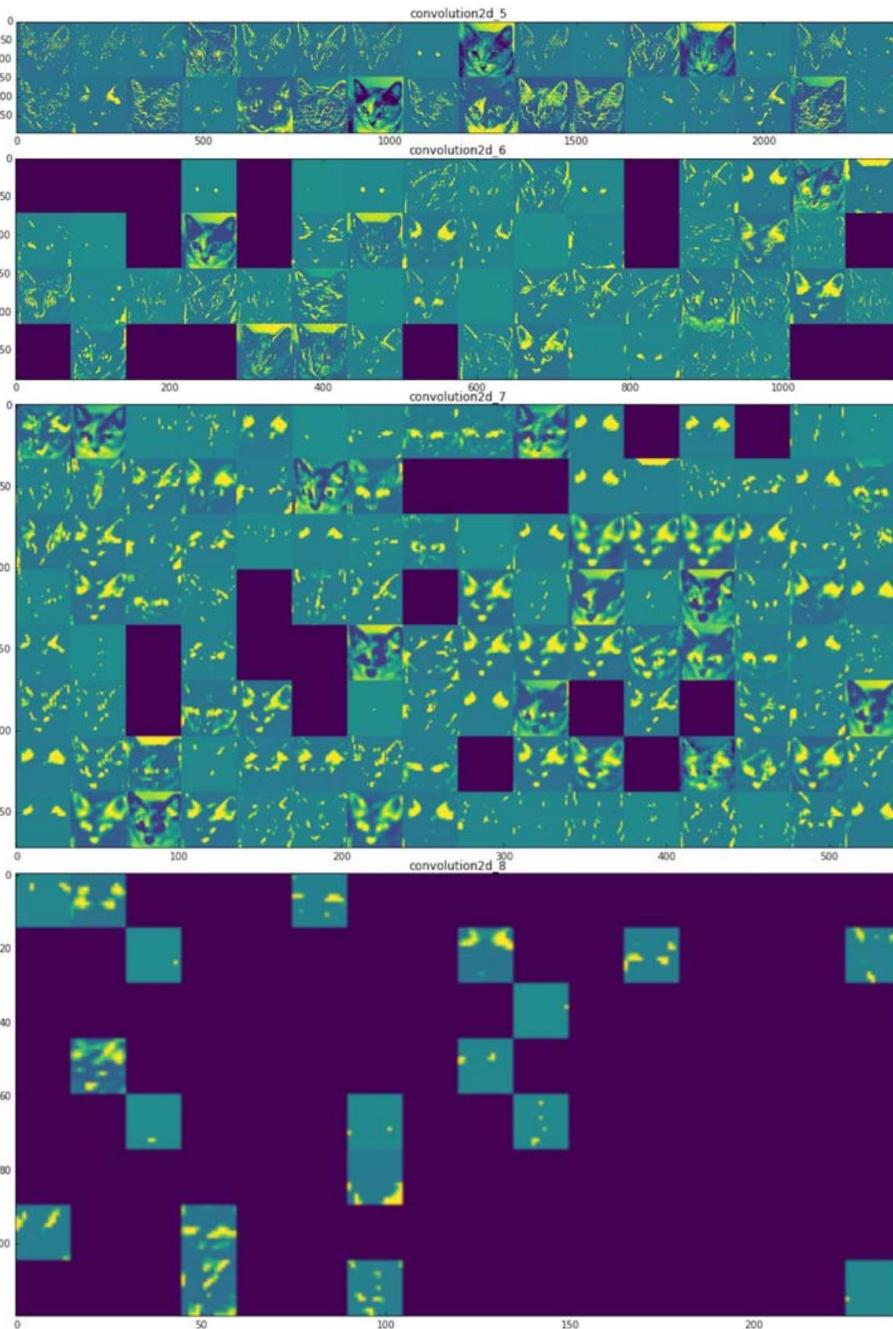
Number of features in the feature map

Tiles the activation channels in this matrix

Post-processes the feature to make it visually palatable



- لایه اول به عنوان مجموعه‌ای از آشکارسازهای مختلف لبه و رنگ عمل می‌کند
 - در این مرحله، فعال‌سازی‌ها تقریباً تمام اطلاعات موجود در تصویر اولیه را حفظ می‌کنند
- هرچه جلوتر برویم، فعال‌سازی‌ها به طور فزاینده‌ای انتزاعی‌تر و کمتر قابل تفسیر بصری می‌شوند
 - شروع به تشخیص مفاهیم سطح بالاتری مانند "گوش سگ" و "چشم گربه" می‌کنند
 - بازنمایی‌های بالاتر اطلاعات کمتری را در مورد محتوای بصری تصویر و اطلاعات بیشتر مربوط به کلاس تصویر را در خود دارند
- تنک (sparse) شدن فعال‌سازی‌ها با عمق لایه افزایش می‌یابد
 - این بدان معناست که الگوی کدگذاری شده توسط فیلتر مورد نظر در تصویر ورودی کمتر یافت می‌شود



نمایش فیلترها

- یکی راه ساده دیگر برای بررسی فیلترهای آموخته شده توسط شبکه‌های کانولوشنی، نمایش الگوی بصری است که هر فیلتر قرار است به آن پاسخ دهد
- این کار را می‌توان با گرادیان افزایشی در فضای ورودی انجام داد
 - بهینه‌سازی تصویر ورودی به منظور به حداقل رساندن پاسخ یک فیلتر خاص
- تصویر ورودی به دست آمده تصویری خواهد بود که فیلتر انتخاب شده حداقل پاسخ را به آن می‌دهد
- برای این منظور، یکتابع ضرر تعريف می‌کنیم که مقدار خروجی یک فیلتر معین در یک لایه کانولوشنی مشخص را محاسبه کند

Listing 5.32 Defining the loss tensor for filter visualization

```
from keras.applications import VGG16
from keras import backend as K

model = VGG16(weights='imagenet',
                include_top=False)

layer_name = 'block3_conv1'
filter_index = 0

layer_output = model.get_layer(layer_name).output
loss = K.mean(layer_output[:, :, :, filter_index])
```

Listing 5.38 Function to generate filter visualizations

Builds a loss function that maximizes the activation of the nth filter of the layer under consideration

```
def generate_pattern(layer_name, filter_index, size=150):
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])

    grads = K.gradients(loss, model.input)[0] ←

    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5) ←

    iterate = K.function([model.input], [loss, grads]) ←

    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128. ←

    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)
```

Computes the gradient of the input picture with regard to this loss

Normalization trick: normalizes the gradient

Returns the loss and grads given the input picture

Starts from a gray image with some noise

Runs gradient ascent for 40 steps

Listing 5.37 Utility function to convert a tensor into a valid image

```
def deprocess_image(x):
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1

    x += 0.5
    x = np.clip(x, 0, 1)

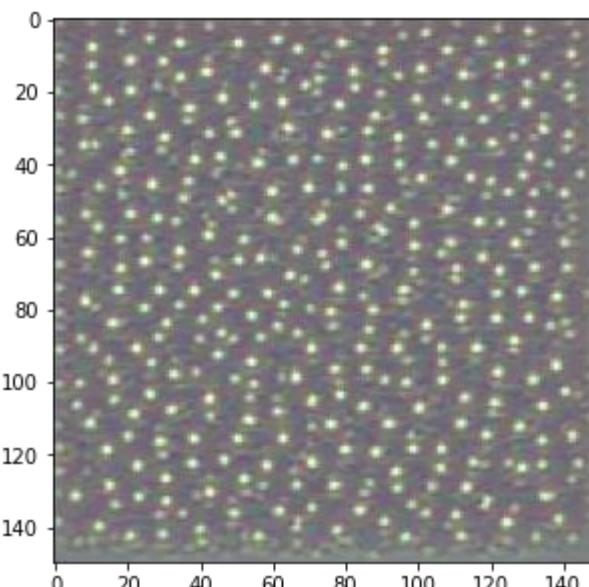
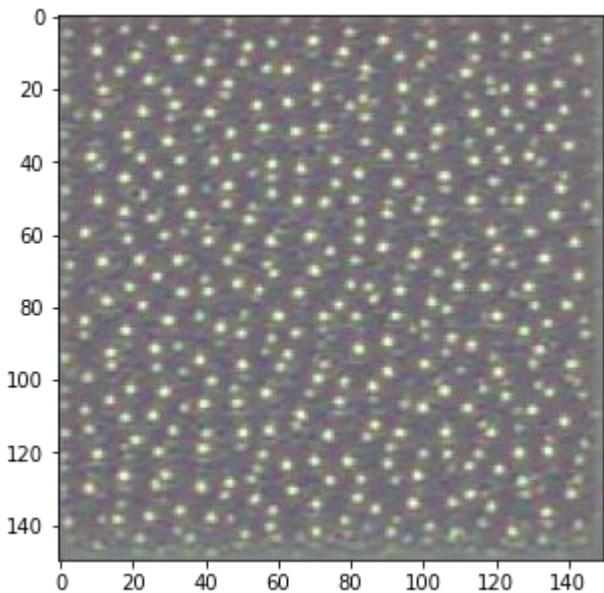
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

**Normalizes the tensor:
centers on 0, ensures
that std is 0.1**

Clips to [0, 1]

Converts to an RGB array

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



Listing 5.39 Generating a grid of all filter response patterns in a layer

```
layer_name = 'block1_conv1'  
size = 64  
margin = 5  
  
results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3)) ← Empty (black) image  
to store results  
  
for i in range(8): ← Iterates over the rows of the results grid  
    for j in range(8): ← Iterates over the columns of the results grid  
        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)
```

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

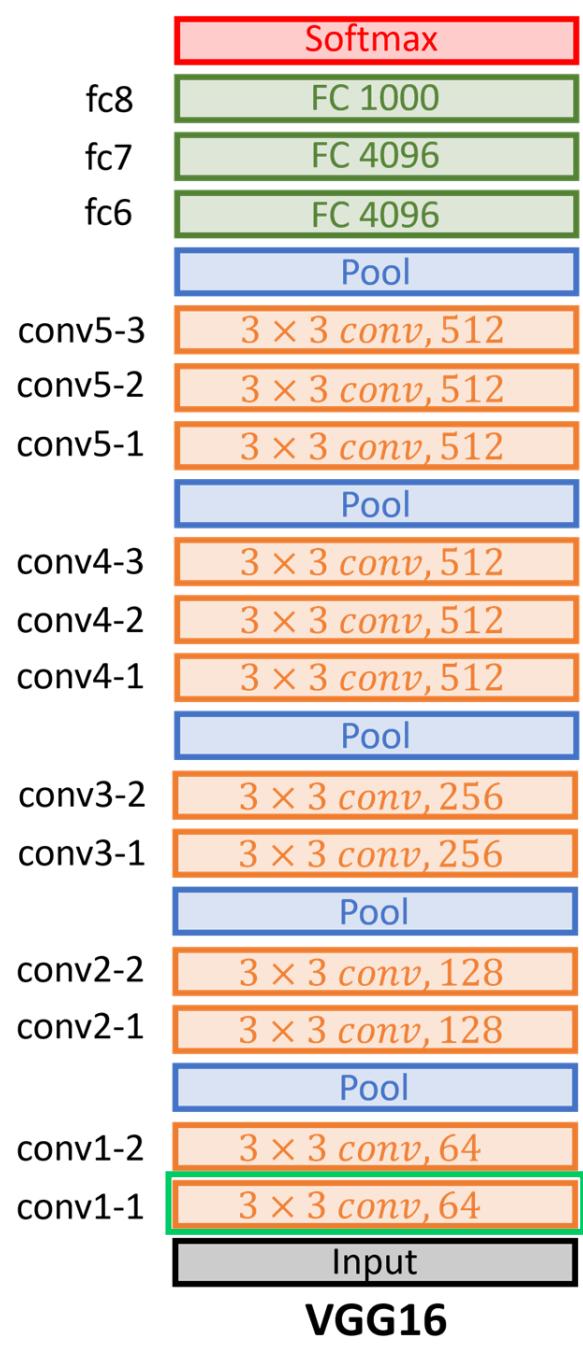
Empty (black) image to store results

Iterates over the rows of the results grid

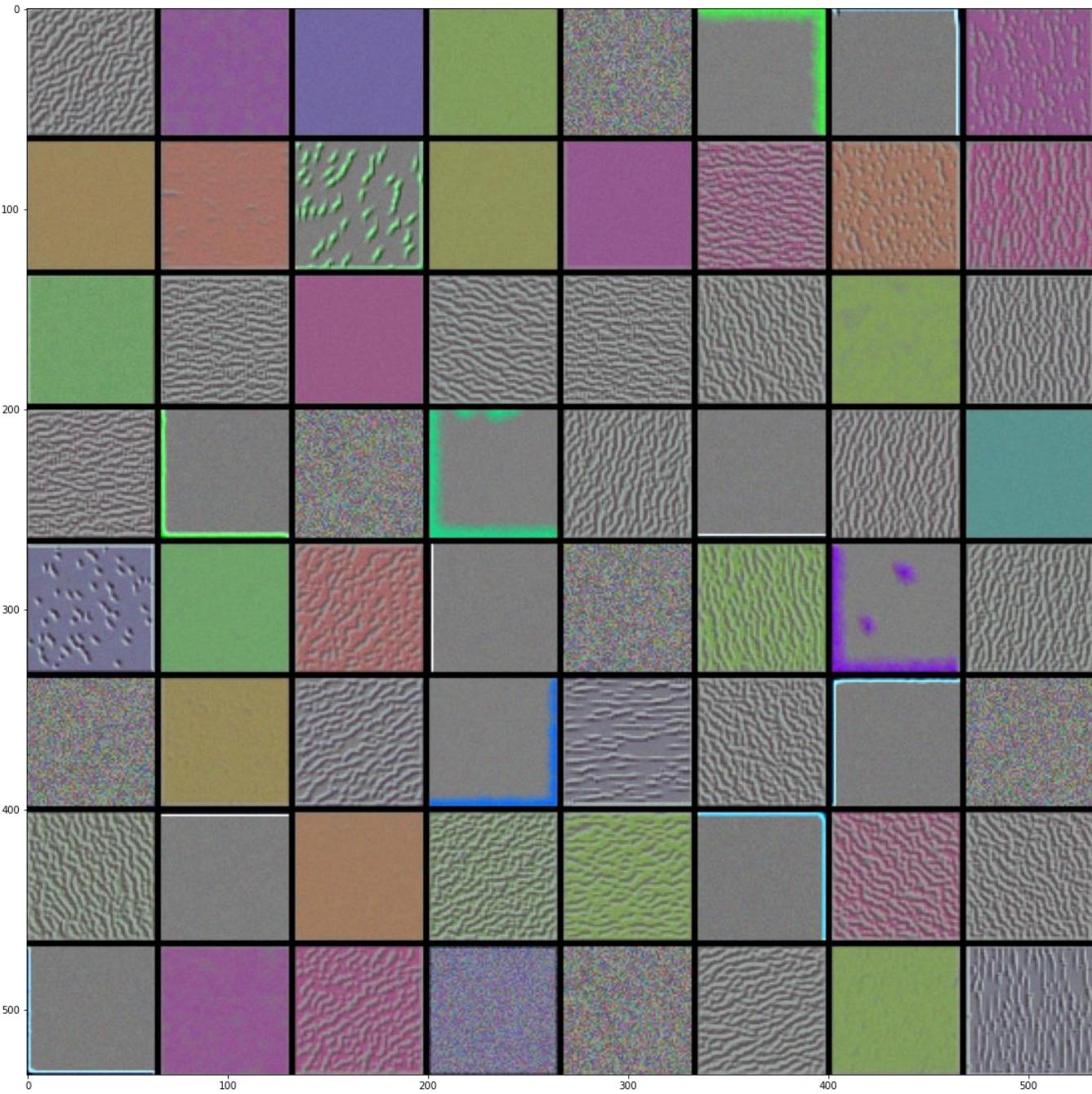
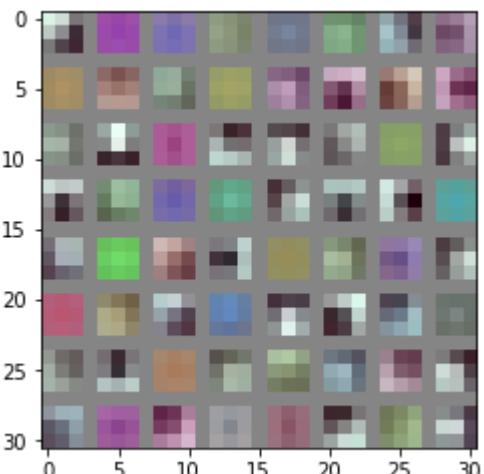
Iterates over the columns of the results grid

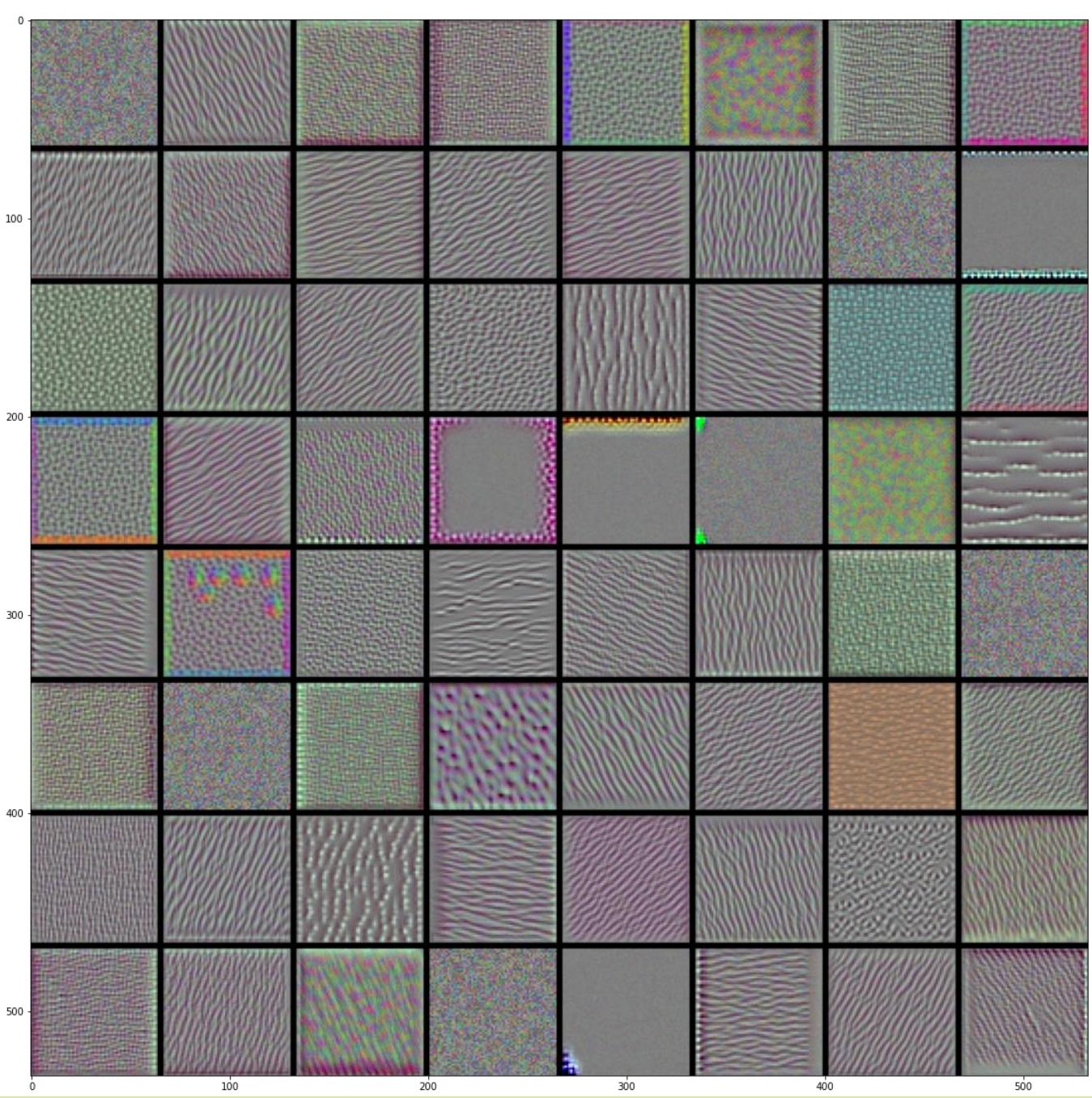
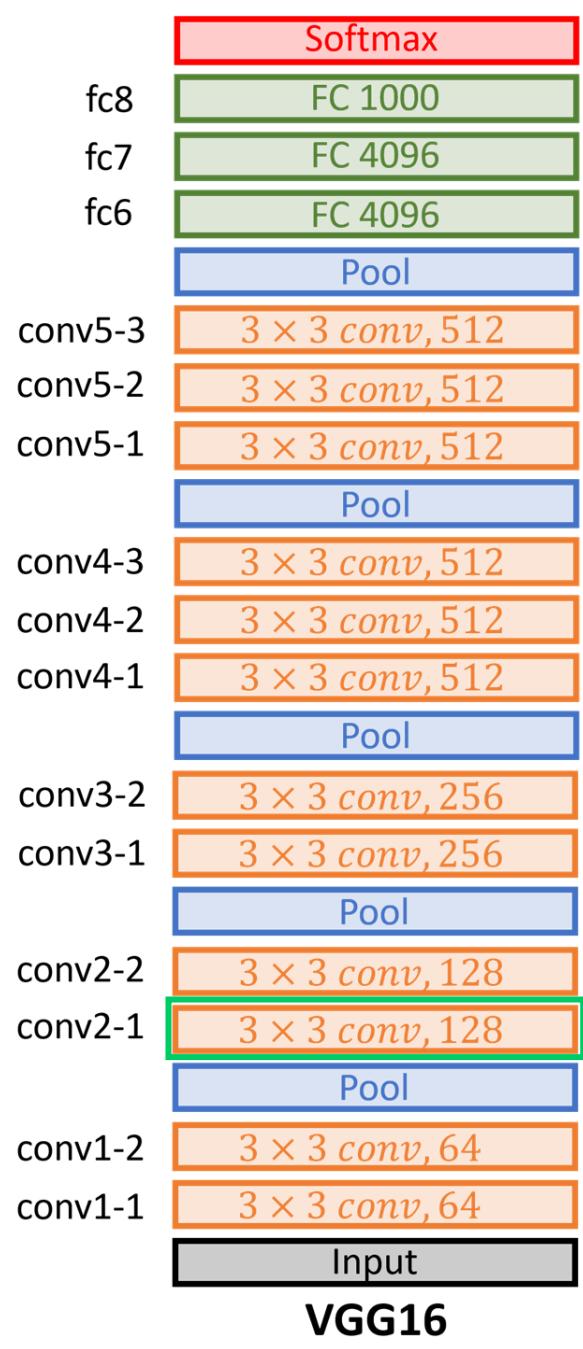
Puts the result in the square (i, j) of the results grid

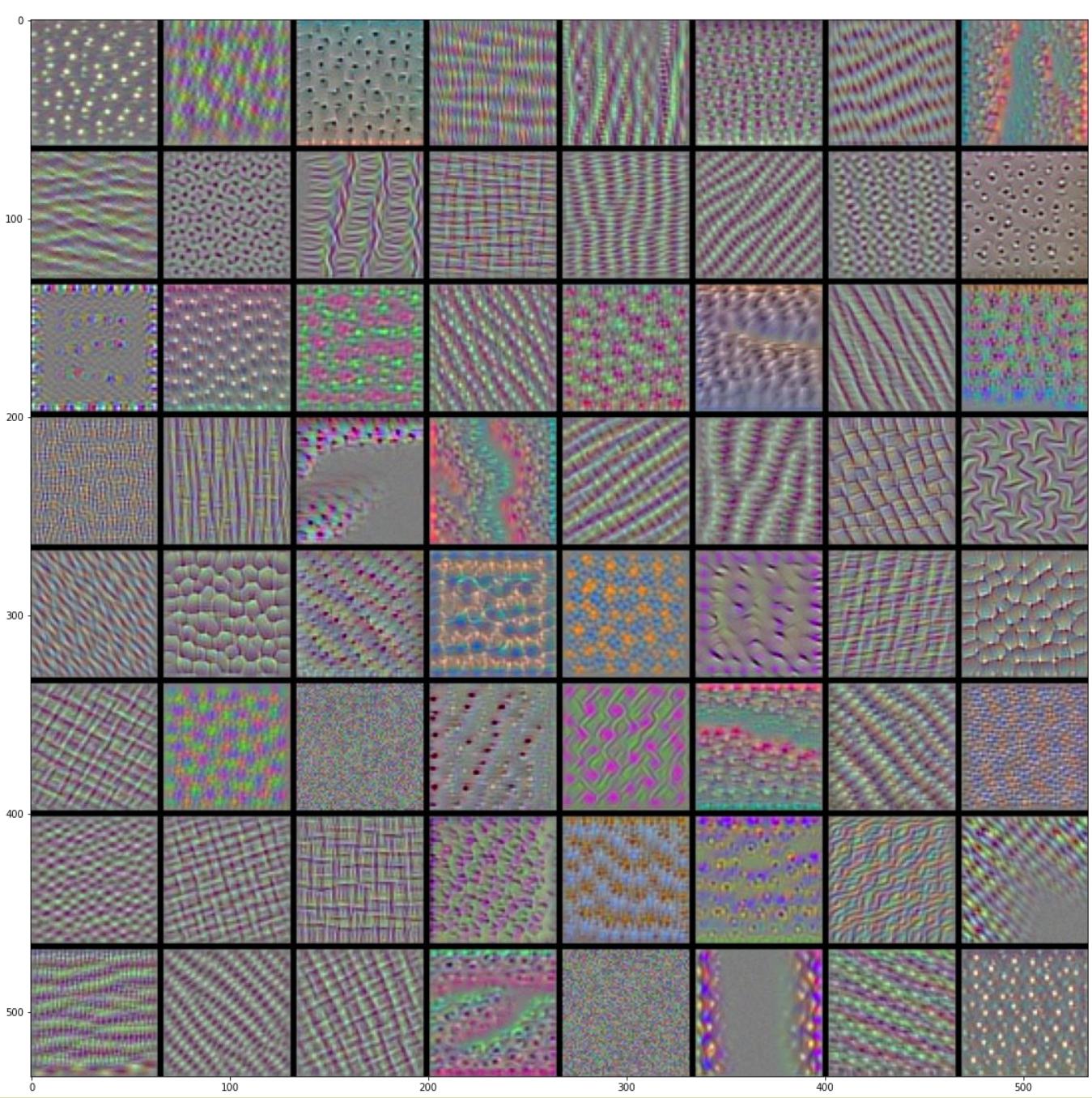
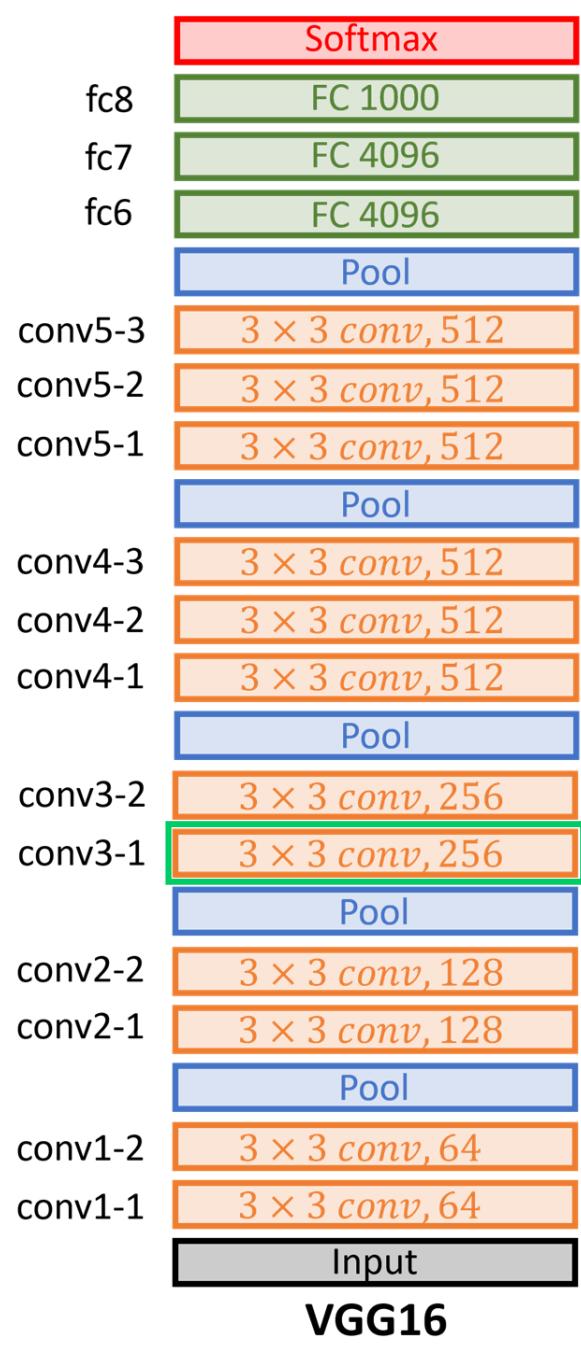
Displays the results grid

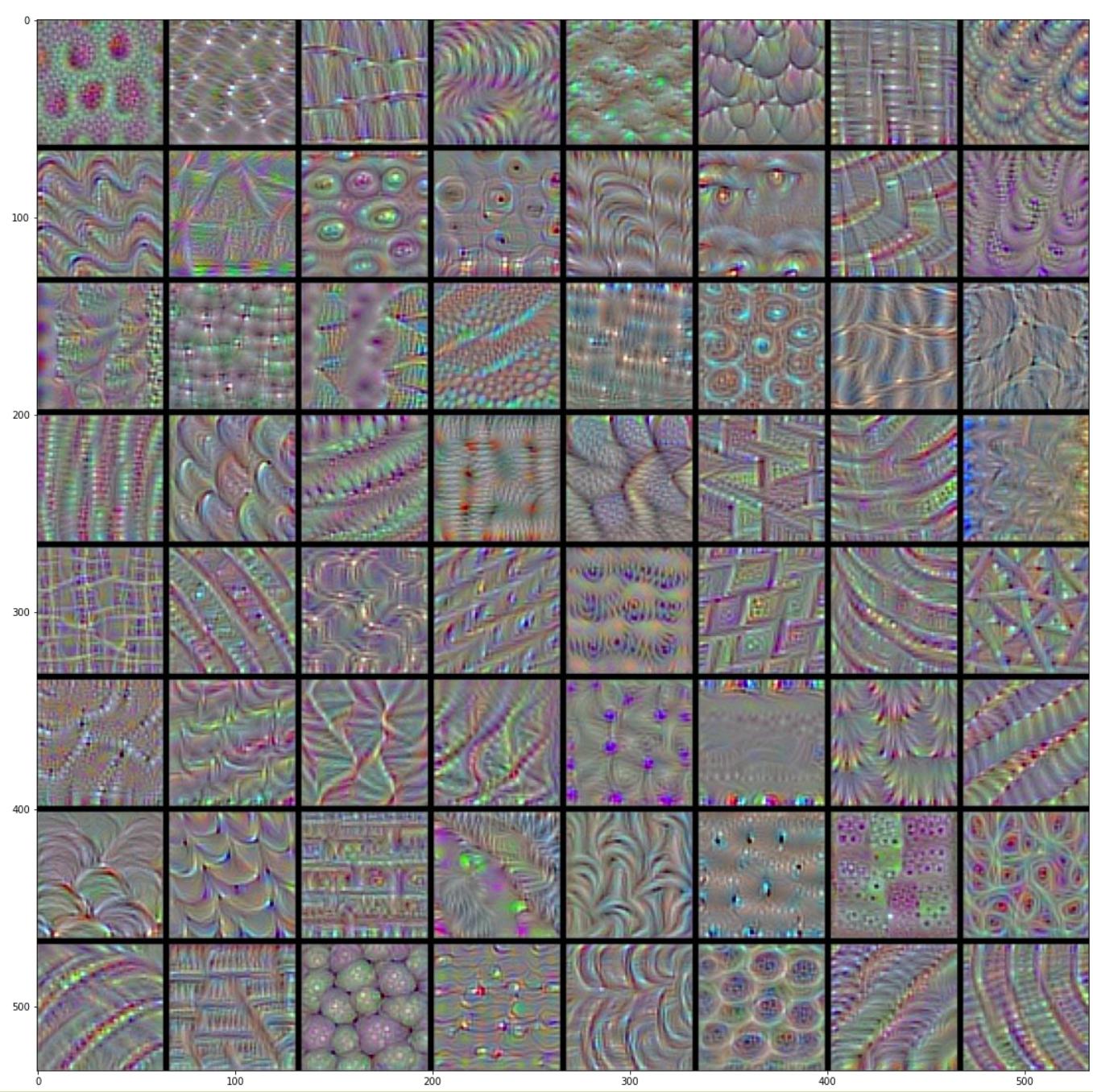
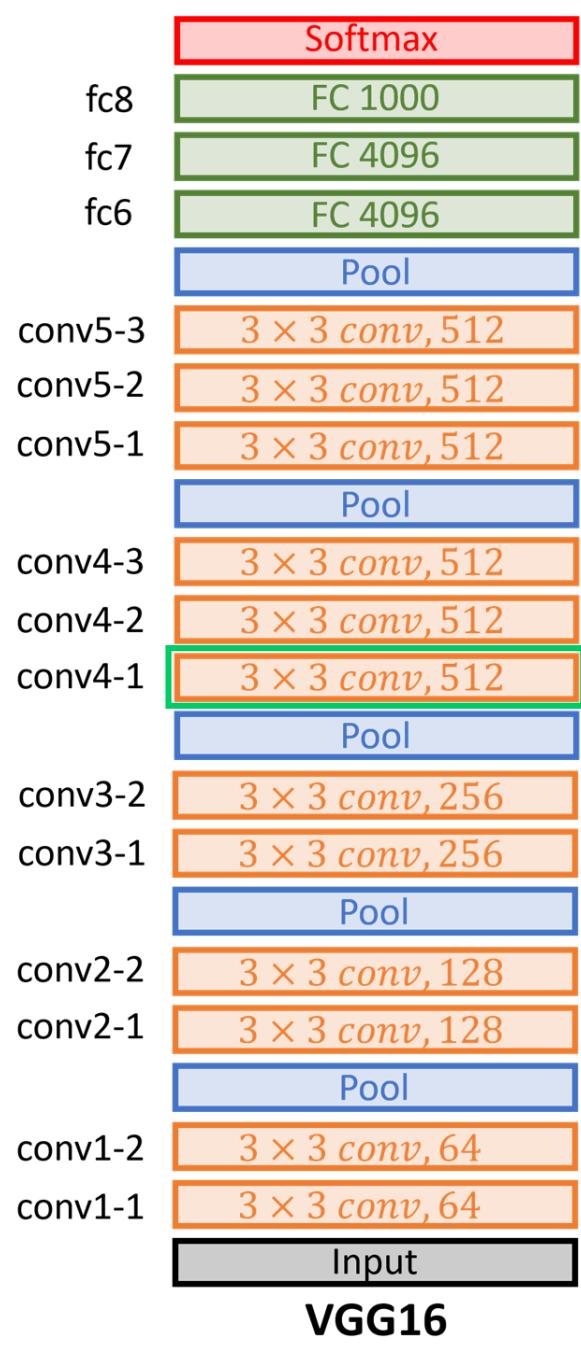


64, $3 \times 3 \times 3$ filters



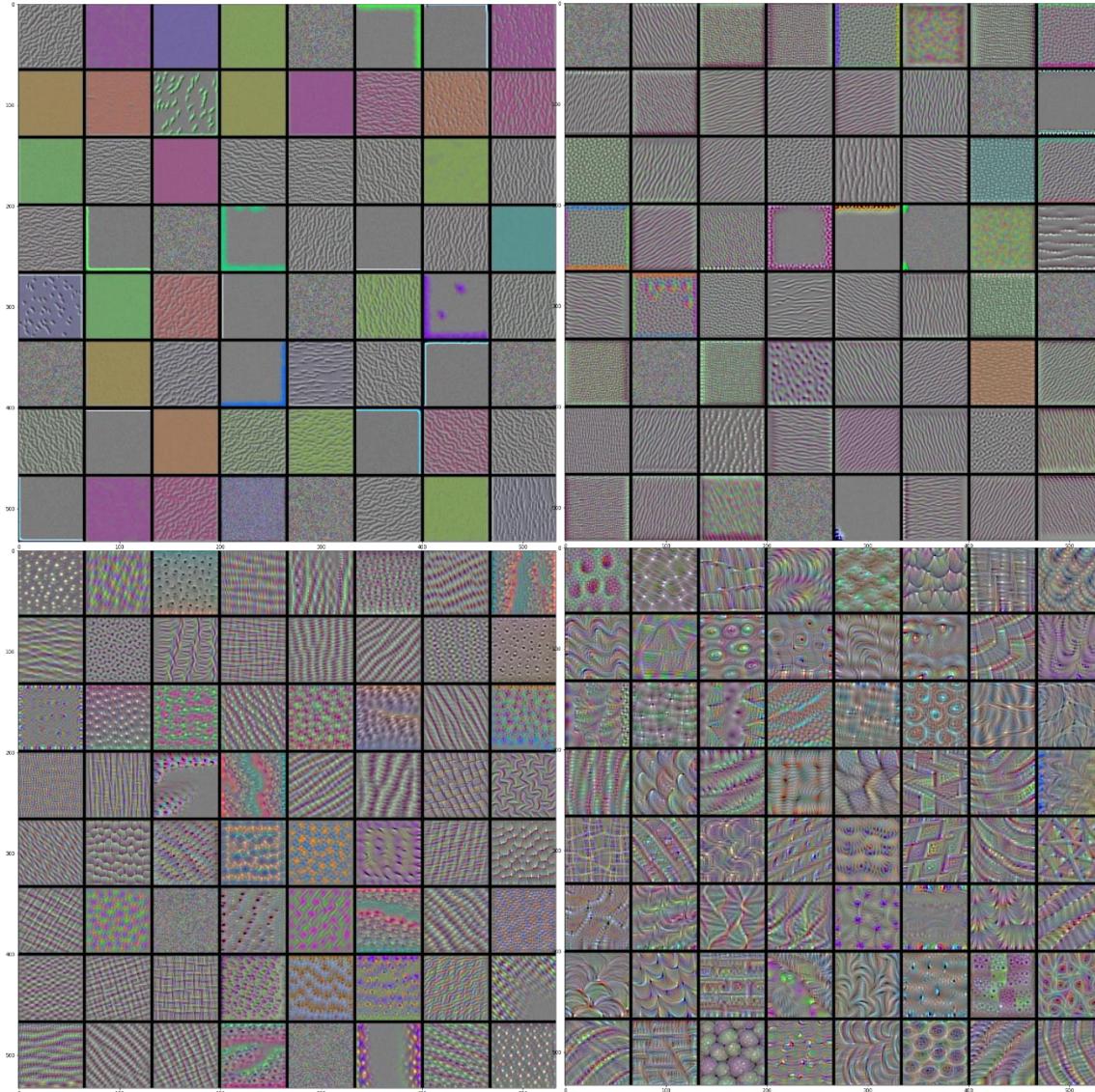






نمایش فیلترها

- هرچه در مدل جلو برویم، فیلترها پیچیده‌تر می‌شوند
 - فیلترهای لایه اول مدل لبه‌ها و رنگ‌های جهت‌دار ساده (یا لبه‌های رنگی، در برخی موارد) را آشکار می‌کنند
 - فیلترهای `block2_conv1` بافت‌های ساده ساخته شده از ترکیب لبه‌ها و رنگ‌ها را آشکار می‌کنند
 - فیلترها در لایه‌های بالاتر شبیه بافت‌های موجود در تصویر طبیعی می‌شوند، مانند پر، چشم، برگ و غیره.



DeepVis

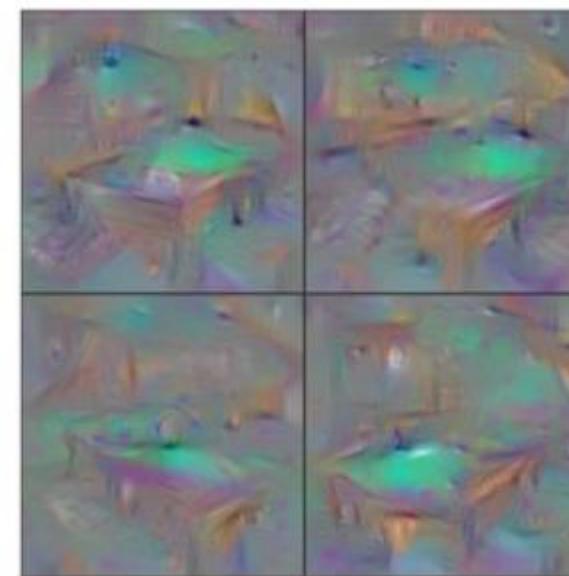
- مصورسازی نورون‌های خروجی

- تصویر ورودی مصنوعی که به بهترین نحو آن نورون را فعال می‌کند

	Softmax
fc8	FC 1000
fc7	FC 4096
fc6	FC 4096
	Pool
conv5-3	$3 \times 3 conv, 512$
conv5-2	$3 \times 3 conv, 512$
conv5-1	$3 \times 3 conv, 512$
	Pool
conv4-3	$3 \times 3 conv, 512$
conv4-2	$3 \times 3 conv, 512$
conv4-1	$3 \times 3 conv, 512$
	Pool
conv3-2	$3 \times 3 conv, 256$
conv3-1	$3 \times 3 conv, 256$
	Pool
conv2-2	$3 \times 3 conv, 128$
conv2-1	$3 \times 3 conv, 128$
	Pool
conv1-2	$3 \times 3 conv, 64$
conv1-1	$3 \times 3 conv, 64$
	Input
	VGG16



Flamingo



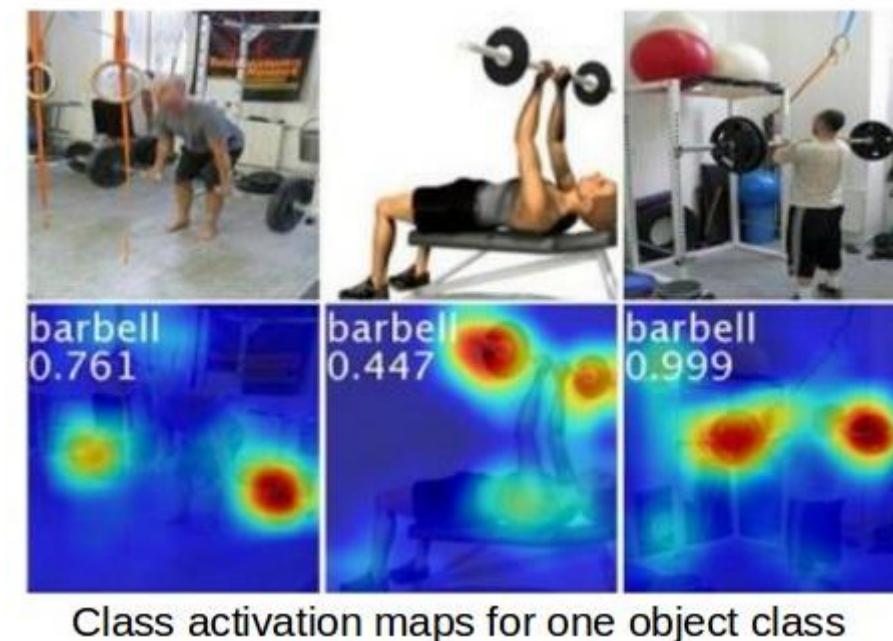
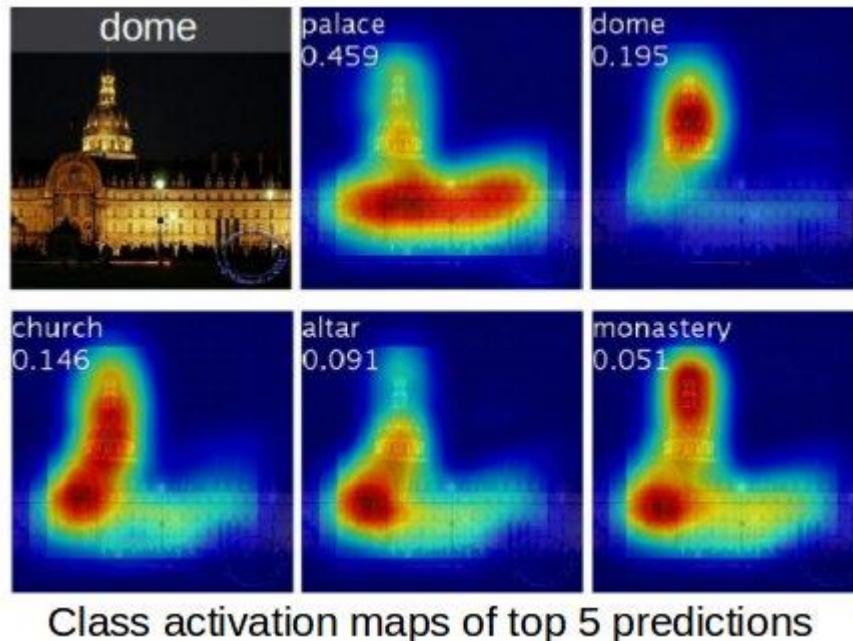
Billiard Table



School Bus

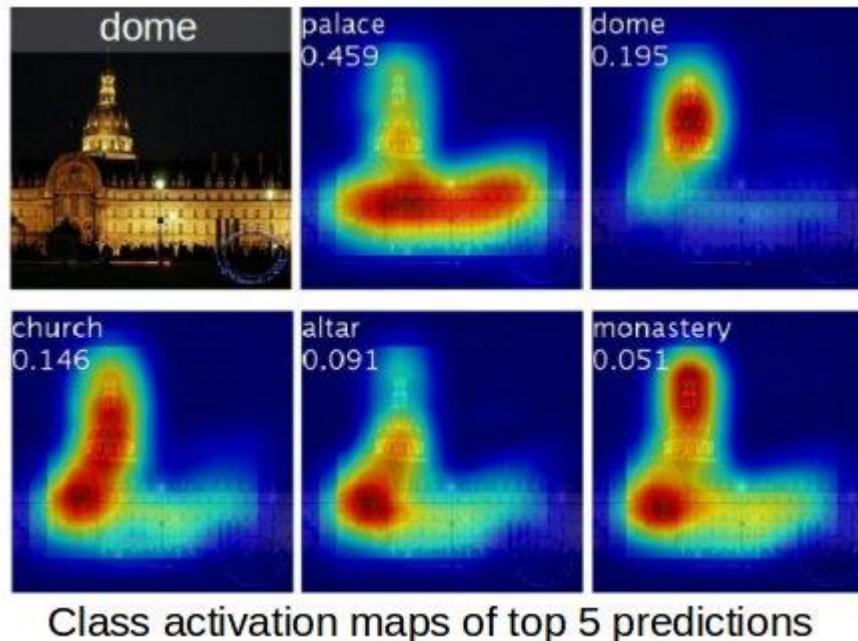
نمایش نقشه‌های حرارتی (heatmaps)

- نقشه فعالیت کلاس (Class Activation Map)
- نشان می‌دهد که هر مکان با توجه به کلاس مورد بررسی چقدر اهمیت دارد
- برای درک اینکه یک شبکه بر اساس کدام بخش از یک تصویر به تصمیم نهایی رسیده است مفید است



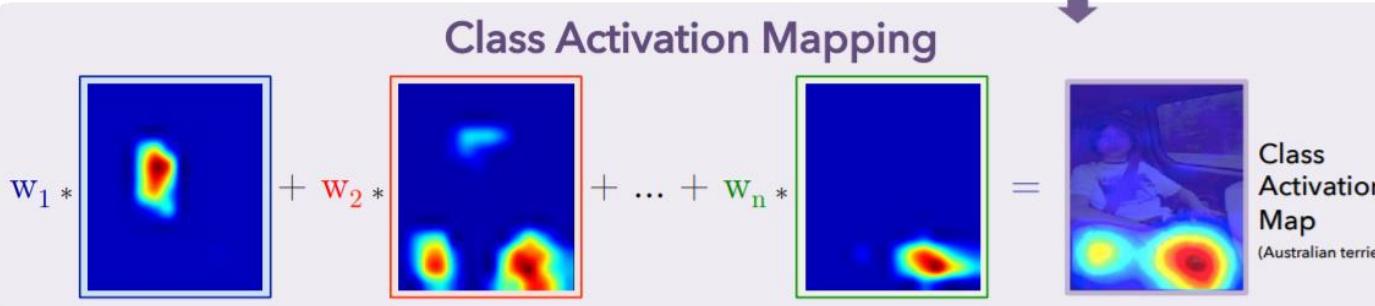
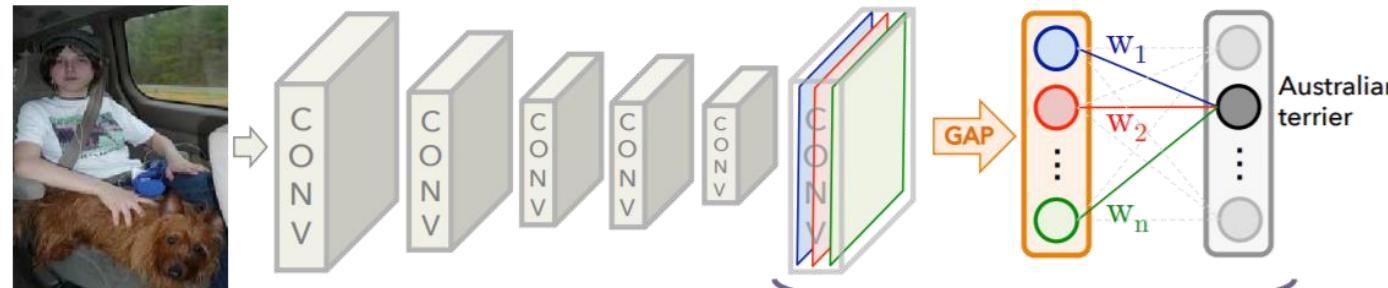
نمایش نقشه‌های حرارتی (heatmaps)

- نقشه فعالیت کلاس (Class Activation Map)
- نشان می‌دهد که هر مکان با توجه به کلاس مورد بررسی چقدر اهمیت دارد
- برای درک اینکه یک شبکه بر اساس کدام بخش از یک تصویر به تصمیم نهایی رسیده است مفید است
- برای اشکال‌زدایی فرآیند تصمیم‌گیری یک شبکه، به ویژه در مواردی که اشتباه کرده است کمک‌کننده است
- همچنین می‌دهد مکان اشیاء مورد نظر را در یک تصویر تخمین بزنیم



Class activation maps of top 5 predictions

CAM



$$F_k = \sum_{x,y} f_k(x, y)$$

$$S_c = \sum_k w_k^c F_k = \sum_k w_k^c \sum_{x,y} f_k(x, y) \sum_k \sum_{x,y} w_k^c f_k(x, y) \sum_{x,y} M_c(x, y)$$

- خروجی آخرین لایه کانولوشنی $f_k(x, y)$ قبل از GAP است

- احتمال (S_c غیرنرمالیزه) پیش‌بینی شده برای کلاس C است (قبل از Softmax)

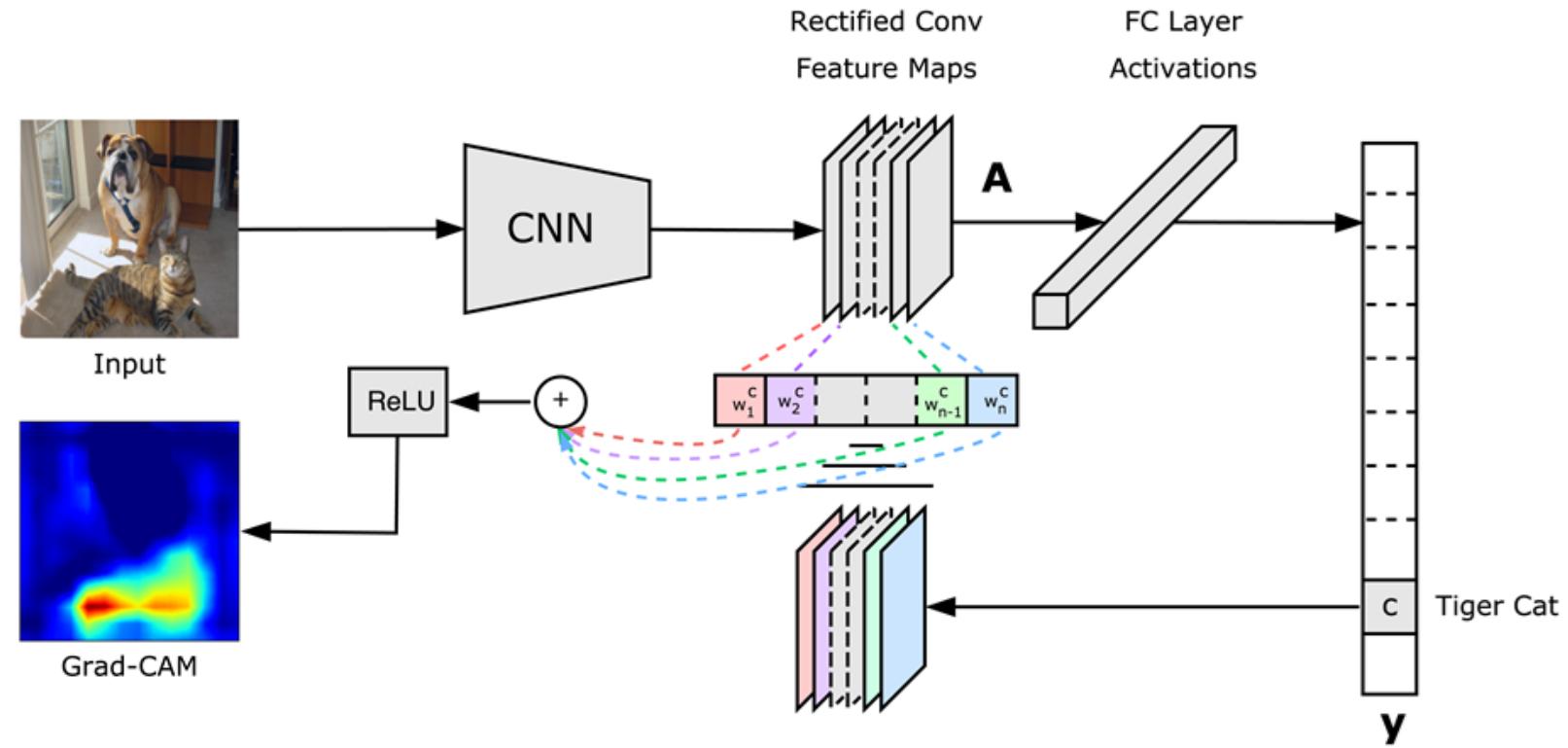
$$M_c(x, y) = \sum_k w_k^c f_k(x, y)$$

Grad-CAM

- برای هر لایه کانولوشنی، وزن هر نقشه فعالیت را بر اساس گرادیان خروجی کلاس مورد نظر نسبت به آن محاسبه می‌کند

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

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



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)
```

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

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

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

```
x = preprocess_input(x)
```

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

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

Local path to the target image

```
>>> 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

```
► 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]])
```

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

```
► 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

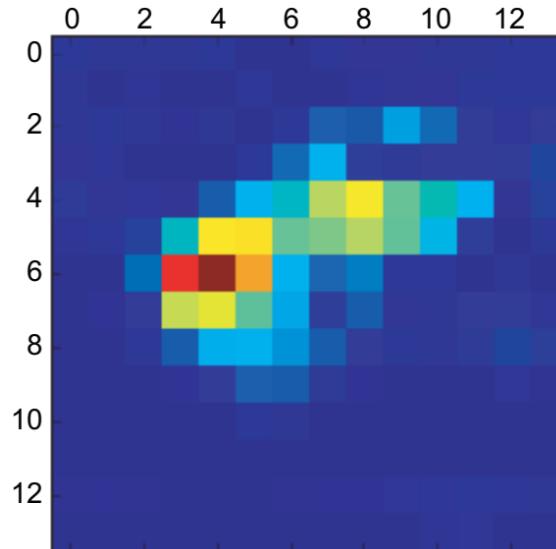
The channel-wise mean of the resulting feature map is the heatmap of the class activation.

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

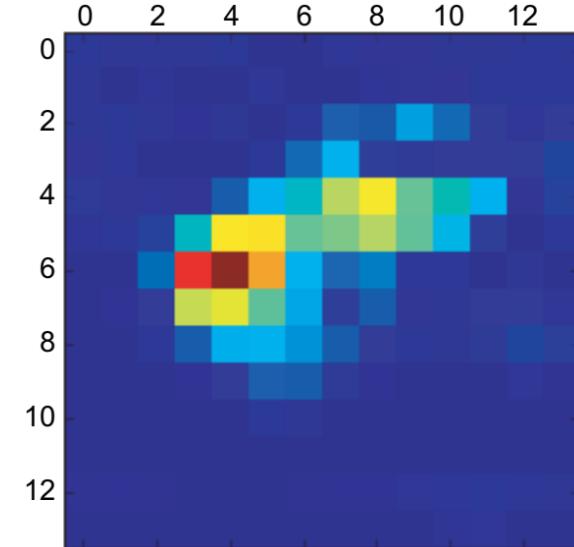
```
import cv2
img = cv2.imread(img_path) ↳ Uses cv2 to load the original image
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0])) ↳ Resizes the heatmap to be the same size as the original image
heatmap = np.uint8(255 * heatmap) ↳ Converts the heatmap to RGB
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
superimposed_img = heatmap * 0.4 + img
cv2.imwrite('/Users/fchollet/Downloads/elephant_cam.jpg', superimposed_img) ↳ 0.4 here is a heatmap intensity factor.
```

Applies the heatmap to the original image



Listing 5.43 Heatmap post-processing

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



نمایش نقشه‌های حرارتی (heatmaps)

- این تکنیک نمایش به دو سوال مهم پاسخ می‌دهد:
 - چرا این شبکه تصمیم گرفت که این تصویر حاوی یک فیل آفریقایی است؟
 - فیل آفریقایی در کجا تصویر قرار دارد؟
- در این مثال، گوش‌های بچه فیل به شدت فعال شده‌اند
- احتمالاً به این دلیل که این شبکه اینگونه می‌تواند تفاوت بین فیل‌های آفریقایی و هندی را تشخیص دهد

