An Introduction to Deep Learning With Python

[5.5] Visualizing convnet filters

Prof. Yuzo Iano

pgs: 167 - 172

Defining the loss tensor for filter visualization

In [1]:
```
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])
```

```
Using TensorFlow backend.

WARNING:tensorflow:From C:\Users\pablo\AppData\Roaming\Python\Python36\site-packages\tensorflow\python\framework\o
p_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Colocations handled automatically by placer.
```

Obtaining the gradient of the loss with regard to the input

In [2]:
```
grads = K.gradients(loss, model.input)[0]
```

Gradient-normalization trick

In [3]:
```
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
```

Fetching Numpy output values given Numpy input values

In [4]:
```
iterate = K.function([model.input], [loss, grads])

import numpy as np
loss_value, grads_value = iterate([np.zeros((1, 150, 150, 3))])
```

Loss maximization via stochastic gradient descent

In [5]:
```
input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.
step = 1.

for i in range(40):
    loss_value, grads_value = iterate([input_img_data])
    input_img_data += grads_value * step
```

Utility function to convert a tensor into a valid image

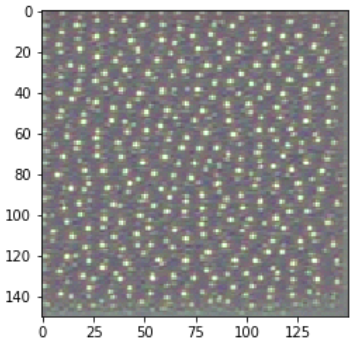
In [6]:
```
def deprocess_image(x):
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1

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

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

Function to generate filter visualizations

In [7]:
```
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)
```

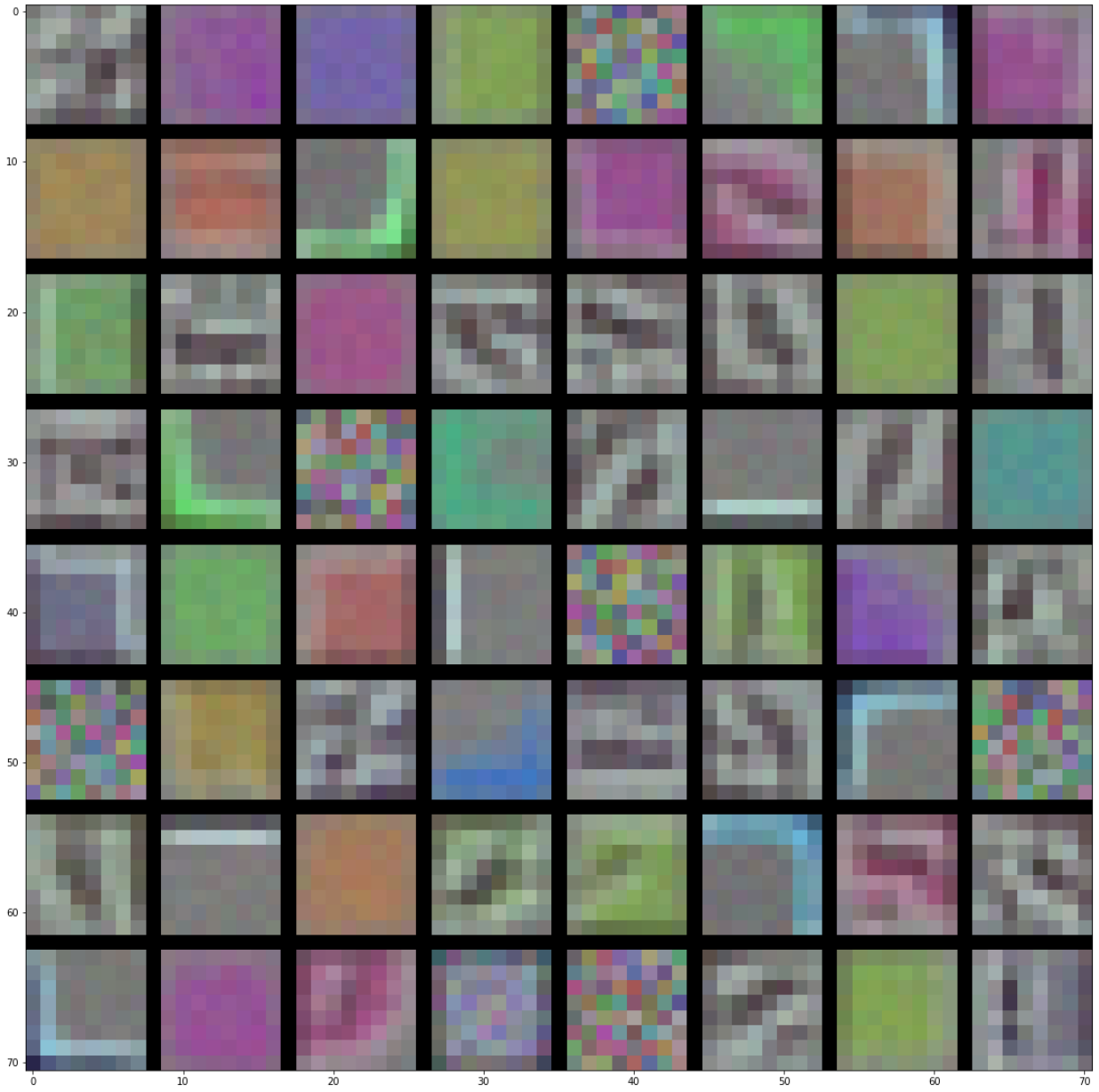
In [10]:
```
import matplotlib.pyplot as plt

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

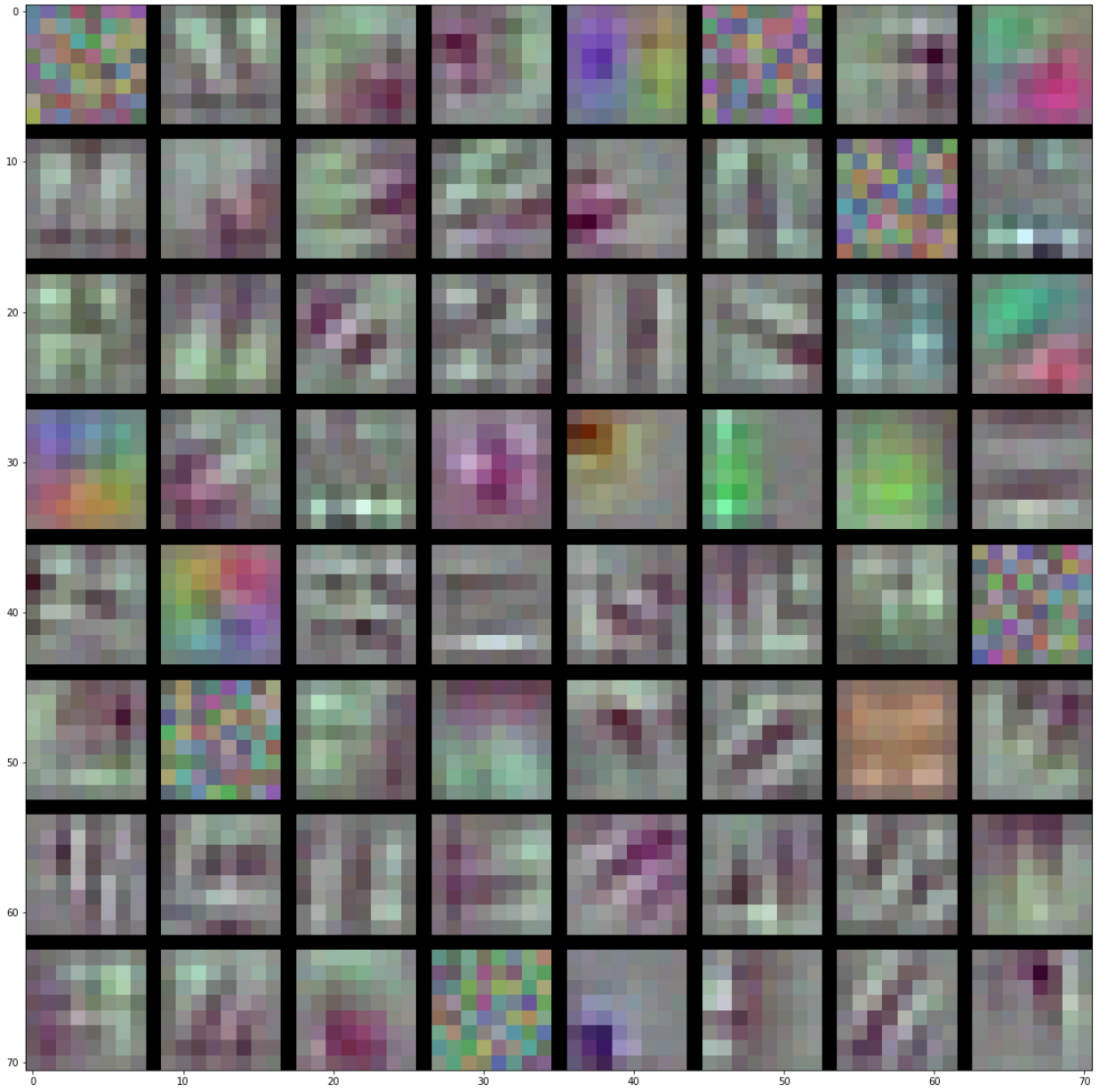
Out[10]:
```
<matplotlib.image.AxesImage at 0x1ff727e2b38>
```

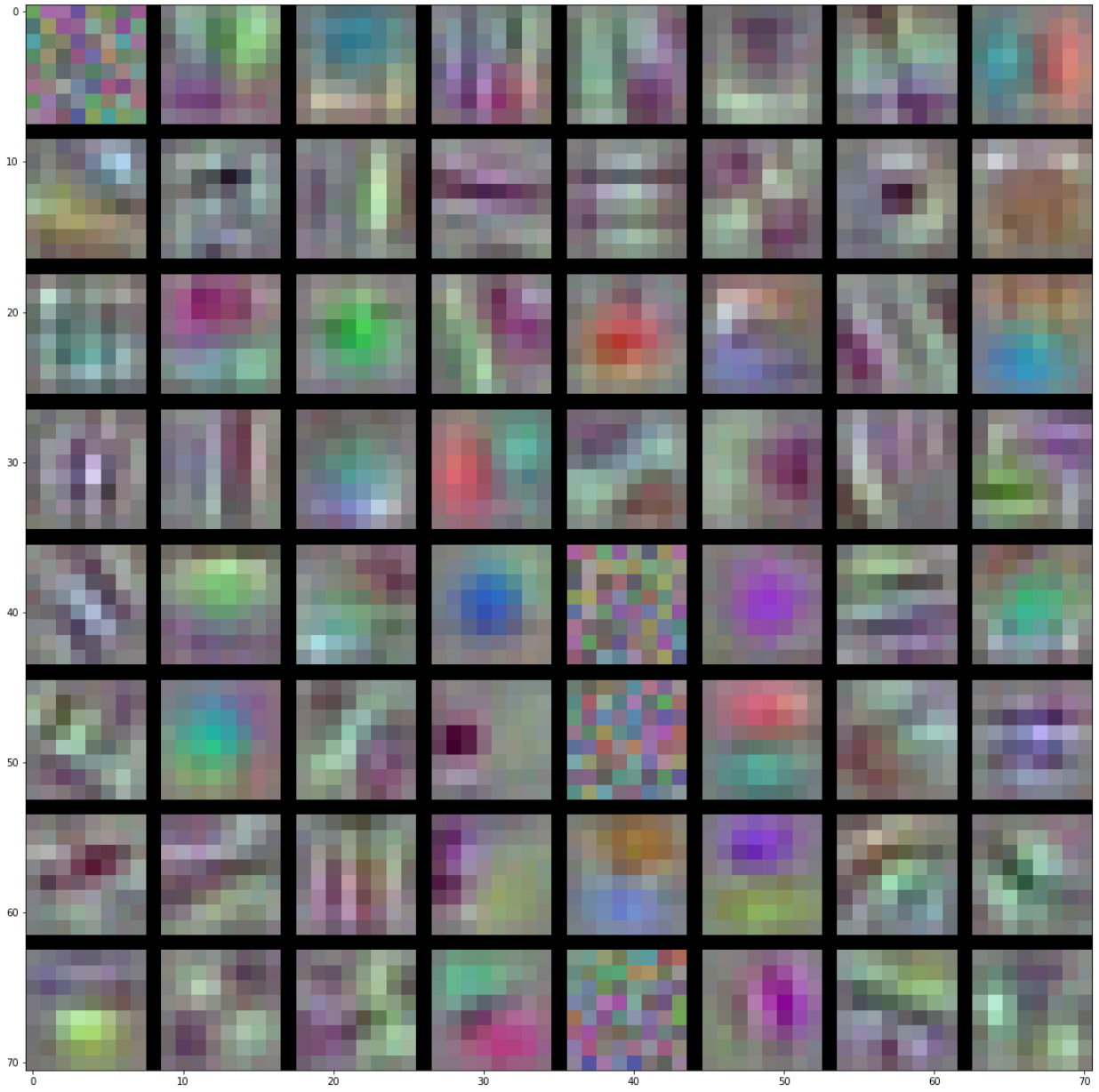


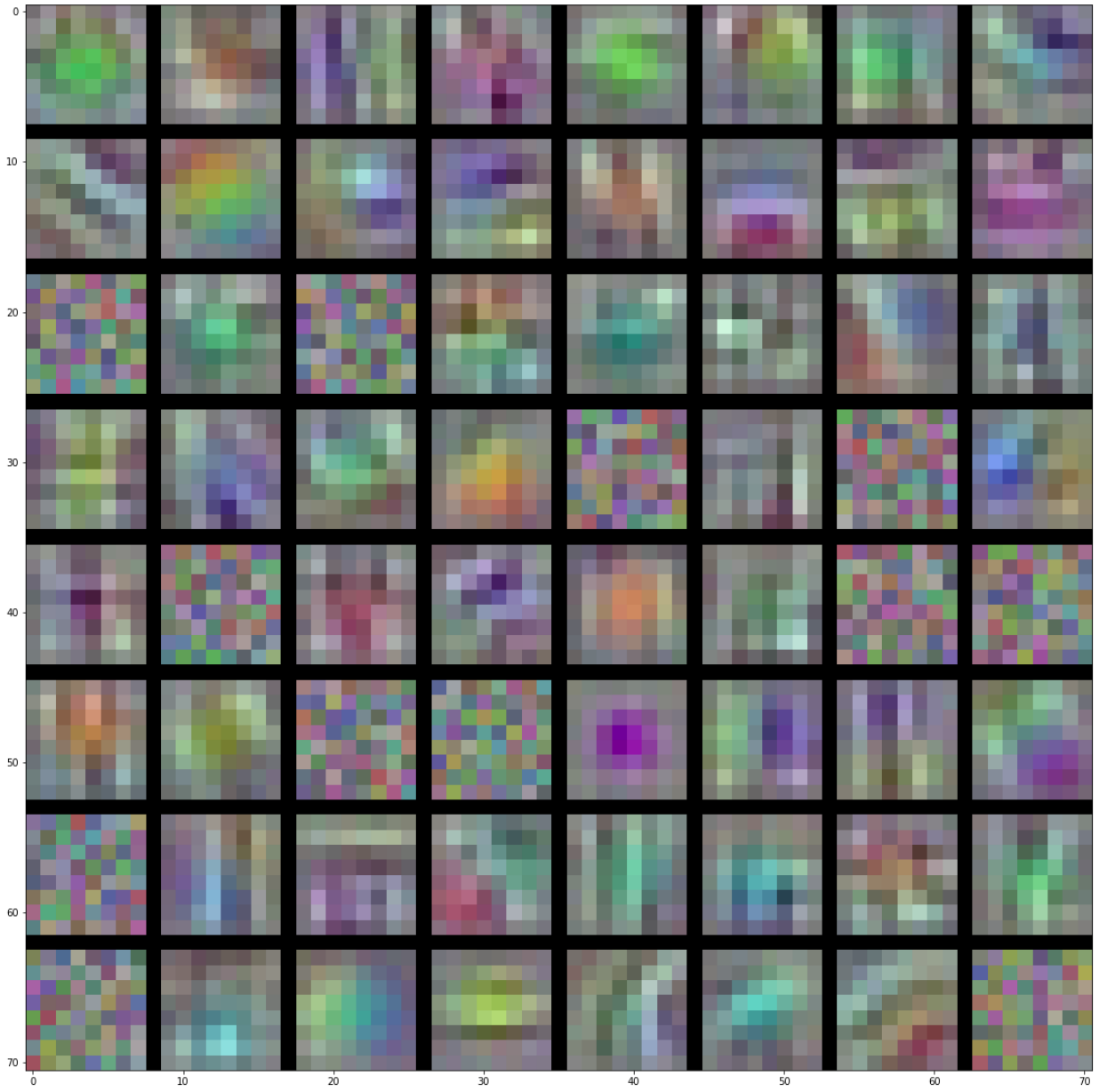
Generating a grid of all filter response patterns in a layer

In [9]:
```
for layer_name in ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1']:
    size = 8
    margin = 1

    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.astype('uint8'))
    plt.show()
```


0
10
20
30
40
50
60
70
0
10
20
30
40
50
60
70


0
10
20
30
40
50
60
70
0
10
20
30
40
50
60
70


0
10
20
30
40
50
60
70
0
10
20
30
40
50
60
70



Pablo Minango

- pablodavid218@gmail.com