In [1]:

```
import keras
keras.__version__
```

```
Using TensorFlow backend.
```

Out[1]:

```
'2.2.4'
```

Visualizing what convnets learn

This notebook contains the code sample found in Chapter 5, Section 4 of [Deep Learning with Python](). Note that the original text features far more content, in particular further explanations and figures: in this notebook, you will only find source code and related comments.

It is often said that deep learning models are "black boxes", learning representations that are difficult to extract and present in a human-readable form. While this is partially true for certain types of deep learning models, it is definitely not true for convnets. The representations learned by convnets are highly amenable to visualization, in large part because they are *representations of visual concepts*. Since 2013, a wide array of techniques have been developed for visualizing and interpreting these representations. We won't survey all of them, but we will cover three of the most accessible and useful ones:

- Visualizing intermediate convnet outputs ("intermediate activations"). This is useful to understand how successive convnet layers transform their input, and to get a first idea of the meaning of individual convnet filters.
- Visualizing convnets filters. This is useful to understand precisely what visual pattern or concept each filter in a convnet is receptive to.
- Visualizing heatmaps of class activation in an image. This is useful to understand which part of an image where identified as belonging to a given class, and thus allows to localize objects in images.

For the first method -- activation visualization -- we will use the small convnet that we trained from scratch on the cat vs. dog classification problem two sections ago. For the next two methods, we will use the VGG16 model that we introduced in the previous section.

Visualizing intermediate activations

Visualizing intermediate activations consists in displaying the feature maps that are output by various convolution and pooling layers in a network, given a certain input (the output of a layer is often called its "activation", the output of the activation function). This gives a view into how an input is decomposed unto the different filters learned by the network. These feature maps we want to visualize have 3 dimensions: width, height, and depth (channels). Each channel encodes relatively independent features, so the proper way to visualize these feature maps is by independently plotting the contents of every channel, as a 2D image. Let's start by loading the model that we saved in section 5.2:

In [2]:

```
from keras.models import load_model

model = load_model('cats_and_dogs_small_2.h5')
model.summary()  # As a reminder.
```

```
WARNING: Logging before flag parsing goes to stderr.
W0716 21:01:56.411632 4456719808 deprecation_wrapper.py:119] From
/anaconda3/envs/pytf/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:517: The name
tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

W0716 21:01:56.436995 4456719808 deprecation_wrapper.py:119] From
/anaconda3/envs/pytf/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:4138: The nam
e tf.random_uniform is deprecated. Please use tf.random.uniform instead.

W0716 21:01:56.469735 4456719808 deprecation_wrapper.py:119] From
/anaconda3/envs/pytf/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:3976: The nam
e tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

W0716 21:01:56.585834 4456719808 deprecation_wrapper.py:119] From
```

```
/anaconda3/envs/pytf/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:131: The name
tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0716 21:01:56.587511 4456719808 deprecation_wrapper.py:119] From
/anaconda3/envs/pytf/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:133: The name
tf.placeholder_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default instea
d.

W0716 21:01:56.603757 4456719808 deprecation.py:506] From /anaconda3/envs/pytf/lib/python3.6/site-
packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
W0716 21:01:56.755331 4456719808 deprecation_wrapper.py:119] From
/anaconda3/envs/pytf/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:174: The name
tf.get_default_session is deprecated. Please use tf.compat.v1.get_default_session instead.

W0716 21:01:56.965307 4456719808 deprecation_wrapper.py:119] From
/anaconda3/envs/pytf/lib/python3.6/site-packages/keras/optimizers.py:790: The name
tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

W0716 21:01:57.000504 4456719808 deprecation.py:323] From /anaconda3/envs/pytf/lib/python3.6/site-
packages/tensorflow/python/ops/nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_5 (Conv2D)            (None, 148, 148, 32)      896       
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 74, 74, 32)        0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 72, 72, 64)        18496     
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 36, 36, 64)        0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 34, 34, 128)       73856     
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 17, 17, 128)       0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 15, 15, 128)       147584    
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (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
_________________________________________________________________
```

This will be the input image we will use -- a picture of a cat, not part of images that the network was trained on:

In [3]:

```python
img_path = '/Users/ly/Downloads/tiger.jpg'

# We preprocess the image into a 4D tensor
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)
# Remember that the model was trained on inputs
# that were preprocessed in the following way:
```

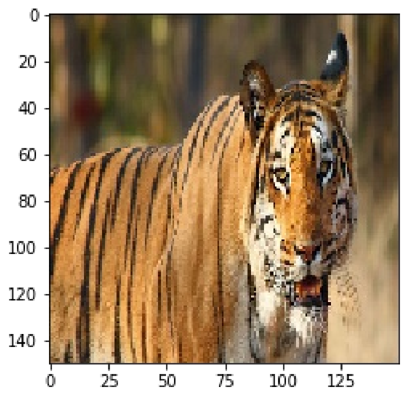
```
img_tensor /= 255.

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

```
(1, 150, 150, 3)
```

Let's display our picture:

In [4]:

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



In [5]:

```
from keras import models
# 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)
```

In [6]:

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

Experiment with the first activation layer

In [31]:

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

```
(1, 148, 148, 32)
```

It's a 148x148 feature map with 32 channels.

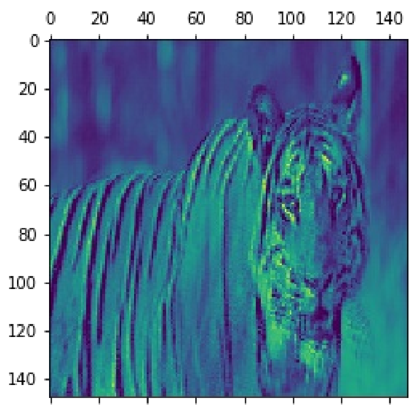
In [ ]:

```
import matplotlib.pyplot as plt
```

14th channel - learns bright background and lines

In [25]:

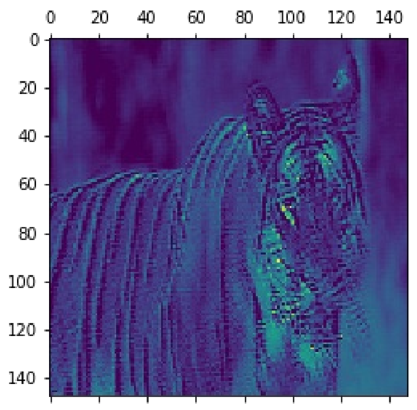
```
plt.matshow(first_layer_activation[0, :, :, 14], cmap='viridis')
plt.show()
```



27th channel - learns dark background and lines

In [24]:

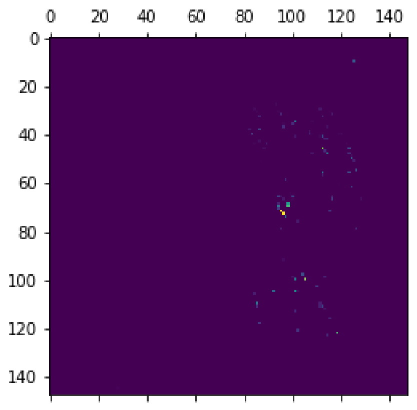
```
plt.matshow(first_layer_activation[0, :, :, 27], cmap='viridis')
plt.show()
```



10th channel - learns eyes!

In [29]:

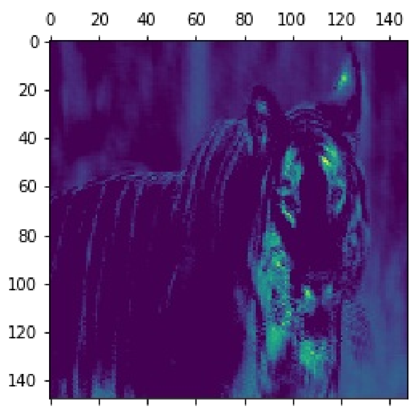
```
plt.matshow(first_layer_activation[0, :, :, 10], cmap='viridis')
plt.show()
```



15th Channel - learns white color

In [30]:

```
plt.matshow(first_layer_activation[0, :, :, 15], cmap='viridis')
plt.show()
```



Experiment with the second Activation Layer

In [32]:

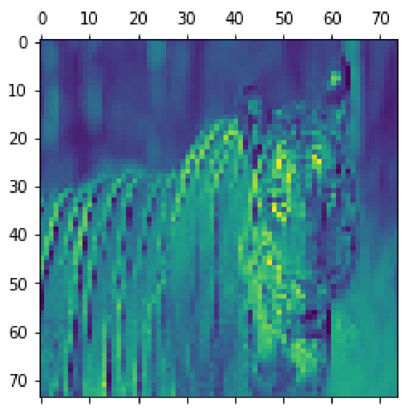
```
second_layer_activation = activations[1]
print(second_layer_activation.shape)
```

```
(1, 74, 74, 32)
```

14th channel - learns bright background color (image becomes fuzzy)

In [33]:

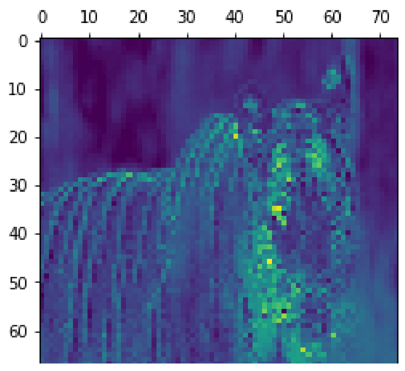
```
plt.matshow(second_layer_activation[0, :, :, 14], cmap='viridis')
plt.show()
```

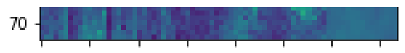


27th channel - learns (a bit)dark background and fuzzy lines

In [34]:

```
plt.matshow(second_layer_activation[0, :, :, 27], cmap='viridis')
plt.show()
```

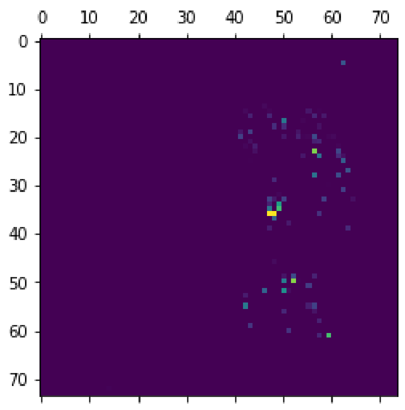




10th channel - captures more bright area, original eyes become fuzzy

In [36]:

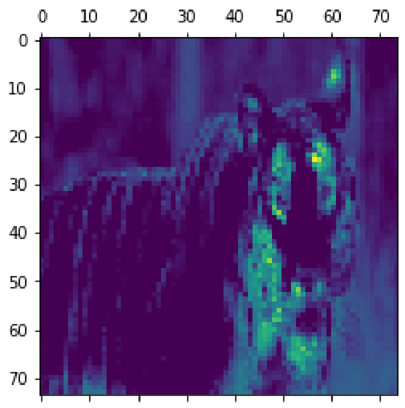
```
plt.matshow(second_layer_activation[0, :, :, 10], cmap='viridis')
plt.show()
```



15th channel - fuzzy image, large area of dark colors without distinct borders

In [37]:

```
plt.matshow(second_layer_activation[0, :, :, 15], cmap='viridis')
plt.show()
```



The experiments shows that the deeper the activation layer, the more "fuzzy" or generalized features are learnt. Also, different filters extract different features from the input image.