Image Analytics in Marketing Code Examples

This notebook is a tutorial for some of the methods described in Dzyabura, El Kihal and Peres (2020), "Image Analytics in Marketing", in *The Handbook of Market Research*, Ch 14, Editors: Christian Homburg, Martin Klarmann, Arnd Vomberg. Springer, 2021. March 2021.

We thank Efrat Naor for her great contribution to this tutorial.

Introduction

This tutorial describes several basic examples of the techniques described in the chapter. We present several techniques for feature extraction, and show an example of a Neural Network classifier.

The code is written in Python. The ideal interface for programming and running the code is Jupyter Notebook via Anaconda. On top of the standard Anaconda libraries, there are several additional libraries that need to be installed. We mark them in the code.

In [1]:
```
import numpy as np
import pandas as pd
import os
import shutil
import random
import cv2 # needs to be installed separately
import matplotlib.pyplot as plt
from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.feature import hog
from skimage import exposure
import matplotlib.image as mpimg
%matplotlib inline
```

Predefined Feature Extraction

We start with several types of basic features that can be extracted from an image.

First, we load a picture and display it.

In [2]:
```
img = mpimg.imread('azreali.jpg')
imgplot = plt.imshow(img)
plt.show()
```

Color Extraction

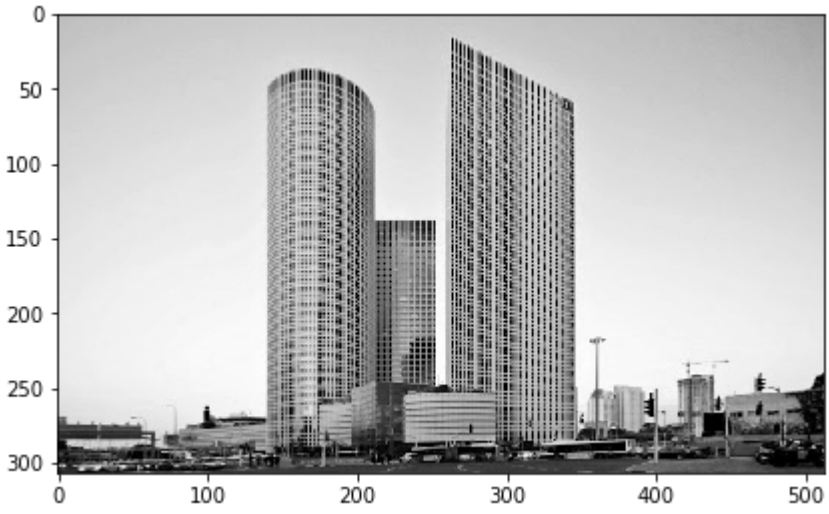
Color is a fundamental image trait. Here are some examples for creating color histograms, which capture the the color composition of the image.

Grayscale Histogram

This is our photo of the three towers in black and white display:

In [3]:
```
img_gray = imread('azreali.jpg',as_gray=True)
imshow(img_gray)
```

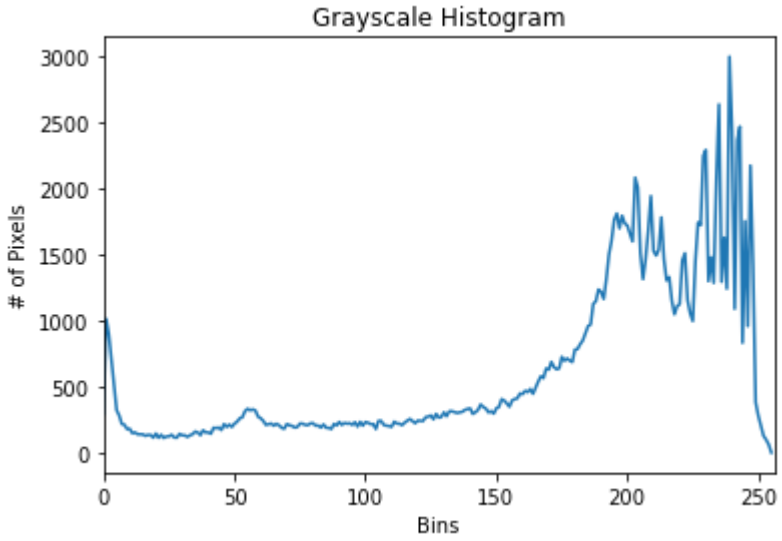
Out[3]: `<matplotlib.image.AxesImage at 0x21b7140bbb0>`



We can creat a color histogram for the grayscale image, counting the intensity of each pixel, ranging between 0 as black and 256 as white. Can you guess how the histogram will look like?

In [4]:
```
image = cv2.imread('azreali.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
hist = cv2.calcHist([gray], [0], None, [256], [0, 256])
plt.figure()
plt.title("Grayscale Histogram")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
plt.plot(hist)
plt.xlim([0, 256])
```

Out[4]: (0.0, 256.0)

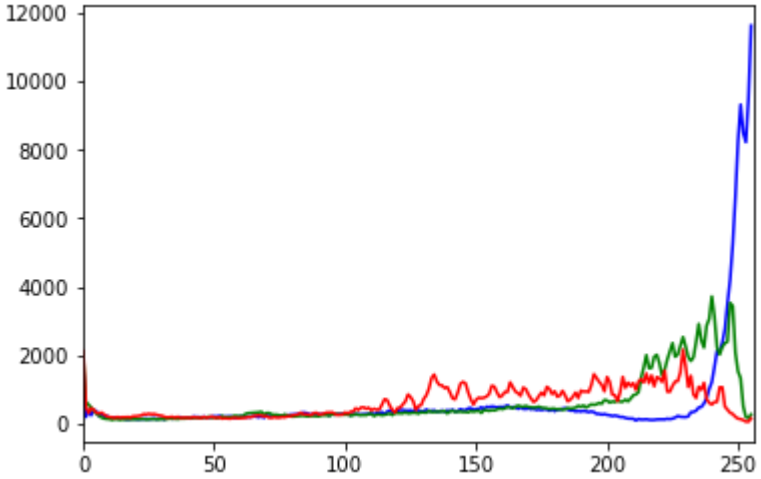


Indeed, the histogram is skewed to the right. The picture contains mainly pixels on the light side of the scale and relatively few dark pixels.

RGB Histogram

Next, we can create a color histogam for the RGB color space, with 0 representing the lowest intensity of the color and 256 the highest intensity. Can you guess what will be the dominant color in the three towers picture?

In [5]:
```
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([image],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```



Indeed. The picture has a lot of blue in it, and the histogram detects these blue pixels.

Another dimension of interest is shape, where the features are line directions, corners, and curves. We will give two examples for extraction of these types of features, which could be classified as "medium" level features.

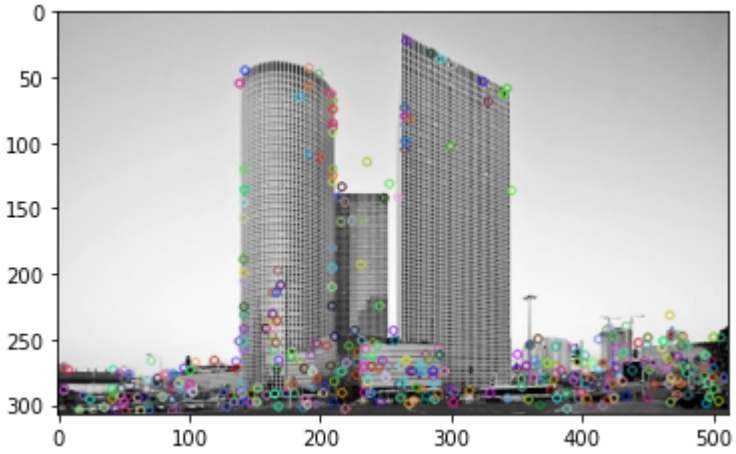
Shape Extraction

SIFT: Scale-invariant feature transform

SIFT identifies the keypoints of the image, that usually lie on object edges. The extracted keypoints can be used to detect the object in the picture and separate it from the background.

In [6]:
```
#SIFT
#reading image
img1 = cv2.imread('azreali.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#keypoints
sift = cv2.xfeatures2d.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)

img_1 = cv2.drawKeypoints(gray1,keypoints_1,img1)
plt.imshow(img_1)
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x21b71673fa0>
```

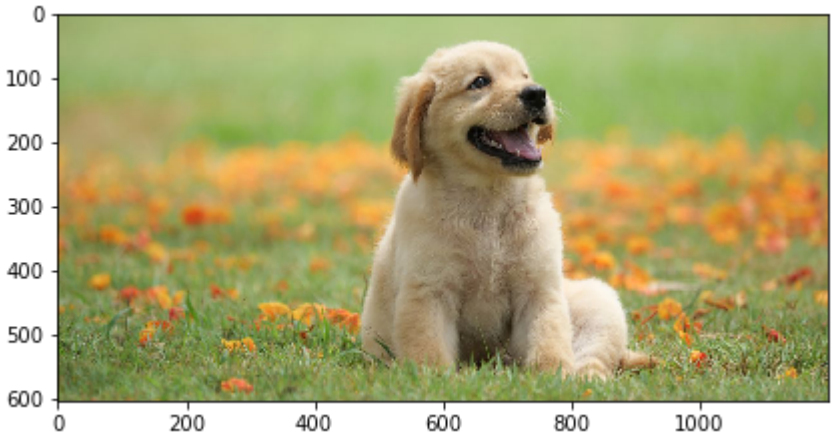


HOG: Histogram of Oriented Gradients

HOG descriptor extracts the edge directions in an image, and is used for object detection.

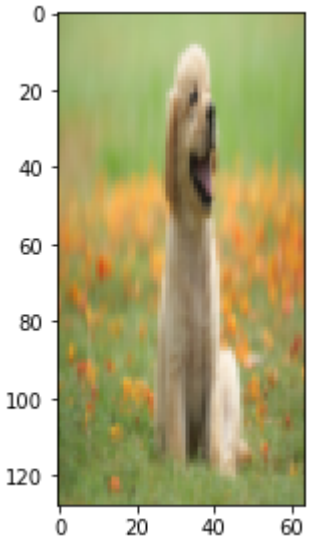
In [7]:
```
#reading the image
image = imread('dog.jpg')
imshow(image)
print('The image shape is '+str(image.shape))
```

```
The image shape is (602, 1200, 3)
```



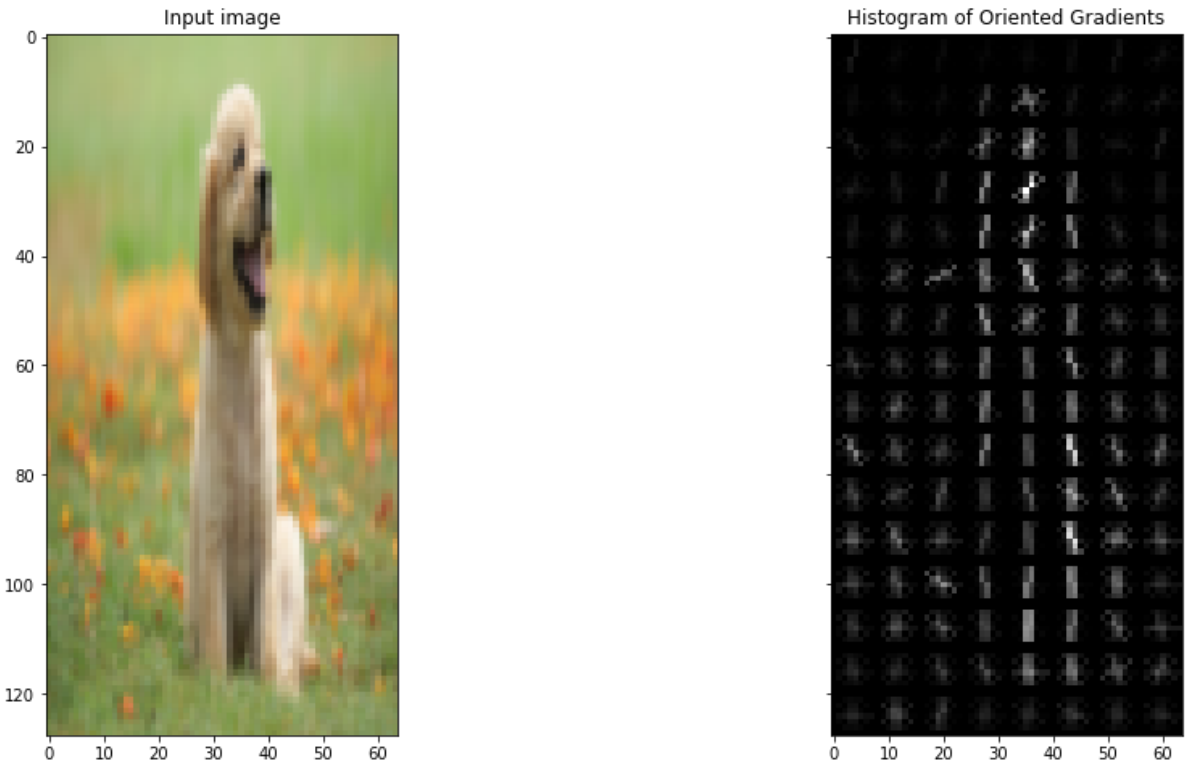
In [8]:
```
#resizing image
resized_img = resize(image, (128,64))
imshow(resized_img)
print('The resized image shape is '+str(resized_img.shape))
```

```
The resized image shape is (128, 64, 3)
```



In [9]:
```
#creating hog features
fd, hog_image = hog(resized_img, orientations=9, pixels_per_cell=(8, 8),
                    cells_per_block=(2, 2), visualize=True)
```

In [10]:
```
# Rescale histogram for better display
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8), sharex=True, sharey=True
)

ax1.imshow(resized_img, cmap=plt.cm.gray)
ax1.set_title('Input image')

# Rescale histogram for better display
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')

plt.show()
```

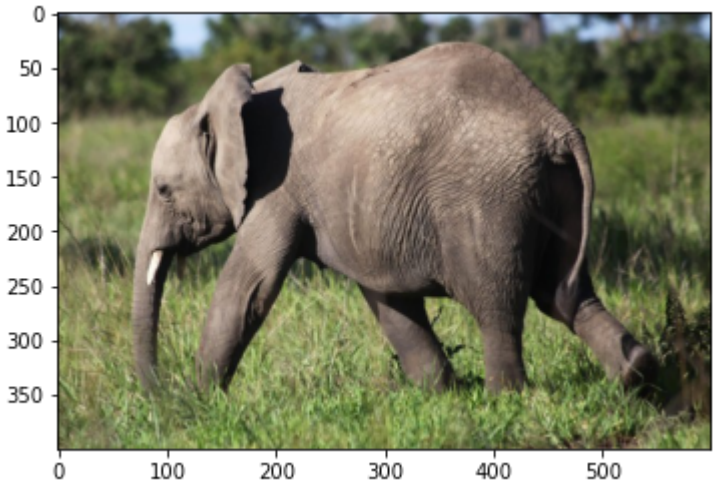


Extracting Interpretable Features - Tagging

As described in the chapter, some image analytic tasks require interpretable features. One way to obtain these features is by using tagging software. Unlike colors, shape and texture, which decompose the image to its visual elements, image tags are "high level" features which identify objects, activities, sceneries, and themes in the image.

An example of tagging software is "Clarifai", which also has a Python API. To use Clarifai, open an account and create an API key for your application

In [11]:
```
#showing the image
img = mpimg.imread("elaphent.jpg")
imgplot = plt.imshow(img)
plt.show()
```



In [11]:
```
import clarifai_grpc # needs to be installed separately
from clarifai_grpc.channel.clarifai_channel import ClarifaiChannel
from clarifai_grpc.grpc.api import service_pb2_grpc

stub = service_pb2_grpc.V2Stub(ClarifaiChannel.get_grpc_channel())
# This is how you authenticate.
metadata = (('authorization', 'Key YOUR_CLARIFAI_API_KEY'),)
```

In [12]:

```
#using genreal model of Clarifai to extract taggs from the image
from clarifai_grpc.grpc.api import service_pb2, resources_pb2
from clarifai_grpc.grpc.api.status import status_code_pb2

#general model id
model_id='aaa03c23b3724a16a56b629203edc62c'

with open("elaphent.jpg","rb") as f:
    file_bytes = f.read()

post_model_outputs_response = stub.PostModelOutputs(
    service_pb2.PostModelOutputsRequest(
        model_id=model_id,
        inputs=[
            resources_pb2.Input(
                data=resources_pb2.Data(
                    image=resources_pb2.Image(
                        base64=file_bytes
                    )
                )
            )
        ]
    ),
    metadata=metadata
)

if post_model_outputs_response.status.code != status_code_pb2.SUCCESS:
    raise Exception("Post model outputs failed, status: " + post_model_outputs
_response.status.description)

# Since we have one input, one output will exist here.
output = post_model_outputs_response.outputs[0]

#printing tags and tag probability. The probability is the confidence level th
at the image indeed contains the tag.
print("Predicted concepts:")
for concept in output.data.concepts:
    print("%s %.2f" % (concept.name, concept.value))
```

```
Predicted concepts:
wildlife 1.00
elephant 1.00
mammal 1.00
no person 0.99
ivory 0.99
grass 0.98
african elephant 0.98
safari 0.98
nature 0.98
animal 0.98
large 0.97
wild 0.97
outdoors 0.95
trunk 0.95
barbaric 0.94
endangered species 0.94
savanna 0.94
Kruger 0.93
grassland 0.93
immense 0.93
```

Convolutional Neural Network Classifier

This part was written based on the tutorial: [https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751 (https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751)](https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751)

We present an example of a Convolutional Neural Network (CNN) classifier that uses transfer learning- namely, applying a network that has already been trained by someone else for a different purpose.

The task is to classify images into two types: elephants and zebras. We execute transfer learning by fine tuning the pre-trained CNN. The fine tuning approach trains the CNN, but instead of initializing the model parameters with random numbers, the model is initialized with parameters learned from another NN. Our model is built from a convolutional base (a VGG16 Neural Network, pre-trained on ImageNet) which generates features from the image, and a classifier (a stack of fully-connected layers), which classifies the image based on the detected features.

Importing libraries and defining directories

In [15]:
```
# tensorflow and keras need to be installed separately
import tensorflow.keras
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras.preprocessing import image
```

In [16]:
```
# the training, validation and test data are under the convolutional_neural_ne
twork_data folder
PATH='YOUR_PATH'
base_dir = PATH+'\convolutional_neural_network_data'
train_dir = os.path.join(base_dir,'train')
test_dir = os.path.join(base_dir,'test')
validation_dir = os.path.join(base_dir, 'validation')
elephant_train_dir=os.path.join(train_dir,'elephant')
zebra_train_dir= os.path.join(train_dir,'zebra')
elephant_validation_dir=os.path.join(validation_dir,'elephant')
zebra_validation_dir=os.path.join(validation_dir,'zebra')
elephant_test_dir = os.path.join(test_dir,'elephant')
zebra_test_dir = os.path.join(test_dir,'zebra')
print('Total training elephant images:', len(os.listdir(elephant_train_dir)))
print('Total training zebra images:', len(os.listdir(zebra_train_dir)))
print('Total validation elephant images:', len(os.listdir(elephant_validation_
dir)))
print('Total validation zebra images:', len(os.listdir(zebra_validation_dir)))
print('Total test elephant images:', len(os.listdir(elephant_test_dir)))
print('Total test zebra images:', len(os.listdir(zebra_test_dir)))
```

```
Total training elephant images: 4800
Total training zebra images: 5797
Total validation elephant images: 1000
Total validation zebra images: 1000
Total test elephant images: 200
Total test zebra images: 200
```

In [17]:
```
elephant_train_size=len(os.listdir(elephant_train_dir))
zebra_train_size=len(os.listdir(zebra_train_dir))
elephant_validation_size=len(os.listdir(elephant_validation_dir))
zebra_validation_size=len(os.listdir(zebra_validation_dir))
elephant_test_size=len(os.listdir(elephant_test_dir))
zebra_test_size=len(os.listdir(zebra_test_dir))
```

In [18]:
```
img_width, img_height = 224, 224  # Default input size for VGG16
train_size=elephant_train_size+zebra_train_size
validation_size=elephant_validation_size+zebra_validation_size
test_size=elephant_test_size+zebra_test_size
```

Extracting features from the images using the convolutional base

The first step in the process is creating the convolutional base, which is a pre-trained CNN, that extracts the features from the images. The convolutional base selected is the VGG16 model, which is available on Keras library, pre-trained on the ImageNet database.

In [19]:
```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(rescale=1./255)
batch_size = 32
def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 7, 7, 512))  # Must be equal to t
he output of the convolutional base
    labels = np.zeros(shape=(sample_count))
    # Preprocess data
    # return a tf.data.Dataset that yields batches of images from the subdirec
tories class_elephant and class_zebra,
    #together with labels 0 and 1 (0 corresponding to class_elephant and 1 cor
responding to class_zebra).
    generator = datagen.flow_from_directory(directory,
                                            target_size=(img_width,img_height
),
                                            batch_size = batch_size,
                                            class_mode='binary')
    # Pass data through convolutional base
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size: (i + 1) * batch_size] = features_batch
        labels[i * batch_size: (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            break
    return features, labels
```

```
Using TensorFlow backend.
```

Instantiating the pre-trained convolutional base (VGG16)

In [20]:
```
#Instantiate convolutional base
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(img_width, img_height, 3))  # 3 = number of cha
nnels in RGB pictures
```

In [21]:
```
# Check architecture
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [22]:
```
#consumes computational power
#creating training features
train_features, train_labels = extract_features(train_dir, train_size)
```

```
Found 10597 images belonging to 2 classes.
```

In [23]:
```
#creating validation features
validation_features, validation_labels = extract_features(validation_dir, vali
dation_size)
```

```
Found 2000 images belonging to 2 classes.
```

Creating the classifier

After extracting the features from the convolutional base, we create the classifier, which is made of fully connected layers. The classifier determines whether the picture is from class "elephant" or from class "zebra". The classifier's predictions are probabilities between 0 to 1 of the image to belong to the zebra class.

In [24]:

```
epochs = 100
# from tensorflow.keras.models import Sequential
model = models.Sequential()
model.add(layers.Flatten(input_shape=(7,7,512)))
model.add(layers.Dense(256, activation='relu', input_dim=(7*7*512)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()

# Compile model
model.compile(optimizer=optimizers.Adam(),
              loss='binary_crossentropy',
              metrics=['acc'])

# Train model
history = model.fit(train_features, train_labels,
                    epochs=epochs,
                    batch_size=batch_size,
                   validation_data=(validation_features, validation_labels))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 256)               6422784
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 6,423,041
Trainable params: 6,423,041
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
332/332 [==============================] - 6s 15ms/step - loss: 0.6414 - acc:
0.8008 - val_loss: 0.1551 - val_acc: 0.9360
Epoch 2/100
332/332 [==============================] - 5s 14ms/step - loss: 0.1739 - acc:
0.9299 - val_loss: 0.0981 - val_acc: 0.9680
Epoch 3/100
332/332 [==============================] - 5s 14ms/step - loss: 0.1456 - acc:
0.9411 - val_loss: 0.0826 - val_acc: 0.9690
Epoch 4/100
332/332 [==============================] - 5s 14ms/step - loss: 0.1116 - acc:
0.9537 - val_loss: 0.0792 - val_acc: 0.9725
Epoch 5/100
332/332 [==============================] - 5s 14ms/step - loss: 0.1004 - acc:
0.9561 - val_loss: 0.0878 - val_acc: 0.9635
Epoch 6/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0926 - acc:
0.9636 - val_loss: 0.0643 - val_acc: 0.9800
Epoch 7/100
332/332 [==============================] - 5s 14ms/step - loss: 0.1002 - acc:
0.9558 - val_loss: 0.0729 - val_acc: 0.9735
Epoch 8/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0869 - acc:
0.9650 - val_loss: 0.0646 - val_acc: 0.9765
Epoch 9/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0962 - acc:
0.9563 - val_loss: 0.0720 - val_acc: 0.9755
Epoch 10/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0959 - acc:
0.9514 - val_loss: 0.0746 - val_acc: 0.9740
Epoch 11/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0836 - acc:
0.9593 - val_loss: 0.1014 - val_acc: 0.9605
Epoch 12/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0932 - acc:
0.9608 - val_loss: 0.0712 - val_acc: 0.9765
Epoch 13/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0628 - acc:
0.9748 - val_loss: 0.0632 - val_acc: 0.9835
Epoch 14/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0624 - acc:
```

```
0.9745 - val_loss: 0.0570 - val_acc: 0.9815
Epoch 15/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0661 - acc:
0.9710 - val_loss: 0.0609 - val_acc: 0.9815
Epoch 16/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0550 - acc:
0.9787 - val_loss: 0.0556 - val_acc: 0.9835
Epoch 17/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0581 - acc:
0.9713 - val_loss: 0.0549 - val_acc: 0.9815
Epoch 18/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0538 - acc:
0.9781 - val_loss: 0.1132 - val_acc: 0.9590
Epoch 19/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0585 - acc:
0.9734 - val_loss: 0.0588 - val_acc: 0.9800
Epoch 20/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0750 - acc:
0.9627 - val_loss: 0.0710 - val_acc: 0.9825
Epoch 21/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0571 - acc:
0.9738 - val_loss: 0.0590 - val_acc: 0.9835
Epoch 22/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0571 - acc:
0.9727 - val_loss: 0.0869 - val_acc: 0.9810
Epoch 23/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0514 - acc:
0.9768 - val_loss: 0.0750 - val_acc: 0.9815
Epoch 24/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0474 - acc:
0.9820 - val_loss: 0.0785 - val_acc: 0.9855
Epoch 25/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0384 - acc:
0.9839 - val_loss: 0.0575 - val_acc: 0.9860
Epoch 26/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0547 - acc:
0.9795 - val_loss: 0.0570 - val_acc: 0.9825
Epoch 27/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0702 - acc:
0.9705 - val_loss: 0.0656 - val_acc: 0.9855
Epoch 28/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0510 - acc:
0.9744 - val_loss: 0.0619 - val_acc: 0.9845
Epoch 29/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0467 - acc:
0.9805 - val_loss: 0.0878 - val_acc: 0.9750
Epoch 30/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0475 - acc:
0.9819 - val_loss: 0.0741 - val_acc: 0.9830
Epoch 31/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0496 - acc:
0.9802 - val_loss: 0.0664 - val_acc: 0.9860
Epoch 32/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0370 - acc:
0.9863 - val_loss: 0.0650 - val_acc: 0.9855
Epoch 33/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0300 - acc:
```

```
0.9889 - val_loss: 0.0779 - val_acc: 0.9835
Epoch 34/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0373 - acc:
0.9851 - val_loss: 0.0673 - val_acc: 0.9875
Epoch 35/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0379 - acc:
0.9855 - val_loss: 0.0610 - val_acc: 0.9840
Epoch 36/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0319 - acc:
0.9866 - val_loss: 0.0529 - val_acc: 0.9885
Epoch 37/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0397 - acc:
0.9801 - val_loss: 0.0757 - val_acc: 0.9830
Epoch 38/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0367 - acc:
0.9817 - val_loss: 0.0596 - val_acc: 0.9875
Epoch 39/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0373 - acc:
0.9846 - val_loss: 0.0611 - val_acc: 0.9870
Epoch 40/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0410 - acc:
0.9835 - val_loss: 0.0784 - val_acc: 0.9830
Epoch 41/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0377 - acc:
0.9840 - val_loss: 0.0749 - val_acc: 0.9850
Epoch 42/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0426 - acc:
0.9814 - val_loss: 0.0805 - val_acc: 0.9835
Epoch 43/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0364 - acc:
0.9850 - val_loss: 0.0752 - val_acc: 0.9840
Epoch 44/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0298 - acc:
0.9869 - val_loss: 0.0879 - val_acc: 0.9805
Epoch 45/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0336 - acc:
0.9844 - val_loss: 0.1078 - val_acc: 0.9845
Epoch 46/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0334 - acc:
0.9878 - val_loss: 0.0913 - val_acc: 0.9860
Epoch 47/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0251 - acc:
0.9895 - val_loss: 0.0795 - val_acc: 0.9895
Epoch 48/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0253 - acc:
0.9891 - val_loss: 0.0661 - val_acc: 0.9870
Epoch 49/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0372 - acc:
0.9827 - val_loss: 0.0728 - val_acc: 0.9885
Epoch 50/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0294 - acc:
0.9867 - val_loss: 0.0811 - val_acc: 0.9840
Epoch 51/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0288 - acc:
0.9878 - val_loss: 0.0765 - val_acc: 0.9850
Epoch 52/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0221 - acc:
```

```
0.9894 - val_loss: 0.0705 - val_acc: 0.9880
Epoch 53/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0205 - acc:
0.9911 - val_loss: 0.0882 - val_acc: 0.9905
Epoch 54/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0258 - acc:
0.9902 - val_loss: 0.0711 - val_acc: 0.9885
Epoch 55/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0249 - acc:
0.9886 - val_loss: 0.0764 - val_acc: 0.9875
Epoch 56/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0335 - acc:
0.9861 - val_loss: 0.0804 - val_acc: 0.9885
Epoch 57/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0294 - acc:
0.9849 - val_loss: 0.0760 - val_acc: 0.9910
Epoch 58/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0292 - acc:
0.9866 - val_loss: 0.0636 - val_acc: 0.9895
Epoch 59/100
332/332 [==============================] - 4s 13ms/step - loss: 0.0280 - acc:
0.9855 - val_loss: 0.0799 - val_acc: 0.9875
Epoch 60/100
332/332 [==============================] - 4s 13ms/step - loss: 0.0202 - acc:
0.9919 - val_loss: 0.0962 - val_acc: 0.9760
Epoch 61/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0366 - acc:
0.9842 - val_loss: 0.0845 - val_acc: 0.9875
Epoch 62/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0315 - acc:
0.9851 - val_loss: 0.0678 - val_acc: 0.9870
Epoch 63/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0316 - acc:
0.9854 - val_loss: 0.0803 - val_acc: 0.9870
Epoch 64/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0303 - acc:
0.9866 - val_loss: 0.1402 - val_acc: 0.9780
Epoch 65/100
332/332 [==============================] - 4s 13ms/step - loss: 0.0376 - acc:
0.9822 - val_loss: 0.0871 - val_acc: 0.9845
Epoch 66/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0314 - acc:
0.9866 - val_loss: 0.0810 - val_acc: 0.9885
Epoch 67/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0246 - acc:
0.9920 - val_loss: 0.0830 - val_acc: 0.9870
Epoch 68/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0185 - acc:
0.9918 - val_loss: 0.0872 - val_acc: 0.9875
Epoch 69/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0269 - acc:
0.9893 - val_loss: 0.0931 - val_acc: 0.9895
Epoch 70/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0171 - acc:
0.9921 - val_loss: 0.0816 - val_acc: 0.9880
Epoch 71/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0202 - acc:
```

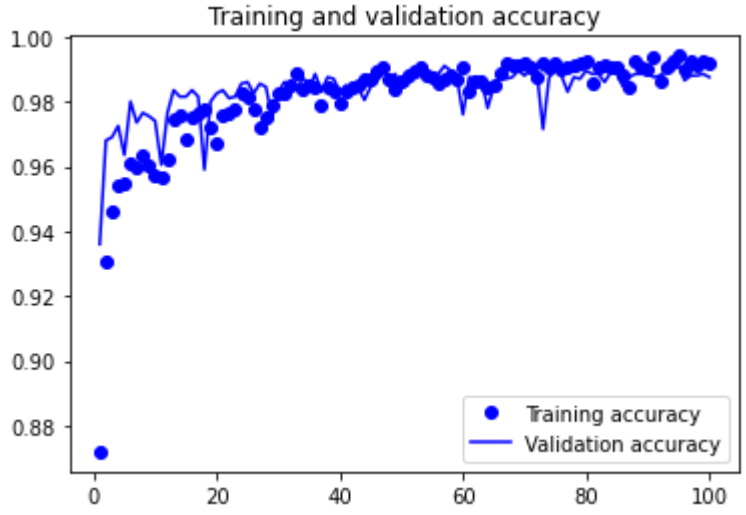
```
0.9917 - val_loss: 0.0734 - val_acc: 0.9900
Epoch 72/100
332/332 [==============================] - 4s 13ms/step - loss: 0.0292 - acc:
0.9879 - val_loss: 0.0611 - val_acc: 0.9890
Epoch 73/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0207 - acc:
0.9909 - val_loss: 0.1677 - val_acc: 0.9715
Epoch 74/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0286 - acc:
0.9896 - val_loss: 0.0809 - val_acc: 0.9875
Epoch 75/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0177 - acc:
0.9915 - val_loss: 0.0773 - val_acc: 0.9895
Epoch 76/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0186 - acc:
0.9899 - val_loss: 0.0702 - val_acc: 0.9880
Epoch 77/100
332/332 [==============================] - 4s 13ms/step - loss: 0.0246 - acc:
0.9910 - val_loss: 0.1221 - val_acc: 0.9830
Epoch 78/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0189 - acc:
0.9912 - val_loss: 0.0845 - val_acc: 0.9875
Epoch 79/100
332/332 [==============================] - 4s 13ms/step - loss: 0.0179 - acc:
0.9921 - val_loss: 0.0813 - val_acc: 0.9870
Epoch 80/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0195 - acc:
0.9922 - val_loss: 0.1271 - val_acc: 0.9895
Epoch 81/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0402 - acc:
0.9849 - val_loss: 0.0759 - val_acc: 0.9885
Epoch 82/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0236 - acc:
0.9899 - val_loss: 0.0849 - val_acc: 0.9885
Epoch 83/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0160 - acc:
0.9936 - val_loss: 0.0880 - val_acc: 0.9860
Epoch 84/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0212 - acc:
0.9926 - val_loss: 0.0741 - val_acc: 0.9890
Epoch 85/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0211 - acc:
0.9907 - val_loss: 0.0868 - val_acc: 0.9885
Epoch 86/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0223 - acc:
0.9902 - val_loss: 0.0884 - val_acc: 0.9845
Epoch 87/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0353 - acc:
0.9832 - val_loss: 0.0762 - val_acc: 0.9875
Epoch 88/100
332/332 [==============================] - 4s 13ms/step - loss: 0.0211 - acc:
0.9916 - val_loss: 0.0936 - val_acc: 0.9885
Epoch 89/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0186 - acc:
0.9915 - val_loss: 0.0906 - val_acc: 0.9885
Epoch 90/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0278 - acc:
```

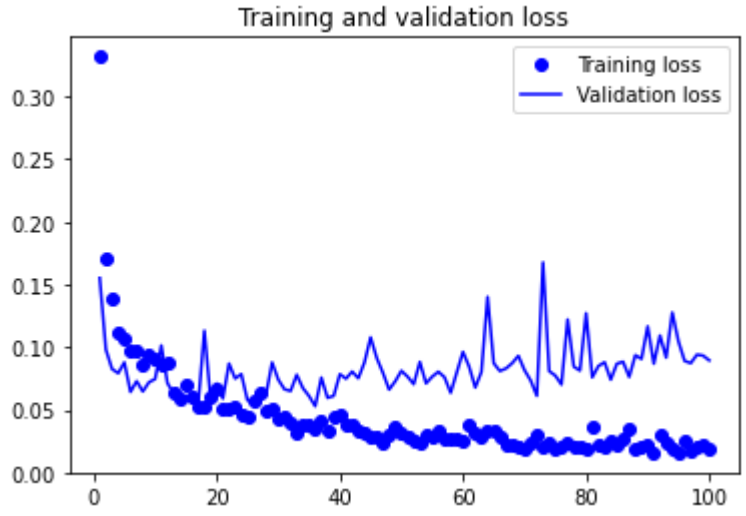
```
0.9861 - val_loss: 0.1166 - val_acc: 0.9885
Epoch 91/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0164 - acc:
0.9936 - val_loss: 0.0868 - val_acc: 0.9875
Epoch 92/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0309 - acc:
0.9874 - val_loss: 0.1093 - val_acc: 0.9880
Epoch 93/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0249 - acc:
0.9893 - val_loss: 0.0917 - val_acc: 0.9895
Epoch 94/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0245 - acc:
0.9891 - val_loss: 0.1276 - val_acc: 0.9895
Epoch 95/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0147 - acc:
0.9942 - val_loss: 0.1046 - val_acc: 0.9910
Epoch 96/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0214 - acc:
0.9915 - val_loss: 0.0890 - val_acc: 0.9865
Epoch 97/100
332/332 [==============================] - 4s 13ms/step - loss: 0.0222 - acc:
0.9910 - val_loss: 0.0872 - val_acc: 0.9880
Epoch 98/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0194 - acc:
0.9920 - val_loss: 0.0940 - val_acc: 0.9880
Epoch 99/100
332/332 [==============================] - 5s 14ms/step - loss: 0.0168 - acc:
0.9938 - val_loss: 0.0933 - val_acc: 0.9885
Epoch 100/100
332/332 [==============================] - 4s 14ms/step - loss: 0.0202 - acc:
0.9909 - val_loss: 0.0894 - val_acc: 0.9875
```

Evaluating the Model

In the evaluation of the model, we calculate the loss and the accuracy over the training set, compared to the validation set (which includes the images the classifier has not seen yet). The loss and the accuracy are calculated for each of the epochs. The loss function used is the binary Cross-Entropy loss, and the accuracy is computed as the rate of the correct predictions out of the total number of predictions made.

In [25]:
```
# Plot results
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc)+1)

plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



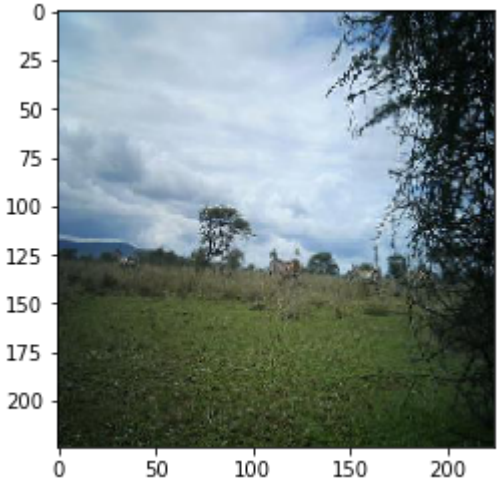


Testing the Model

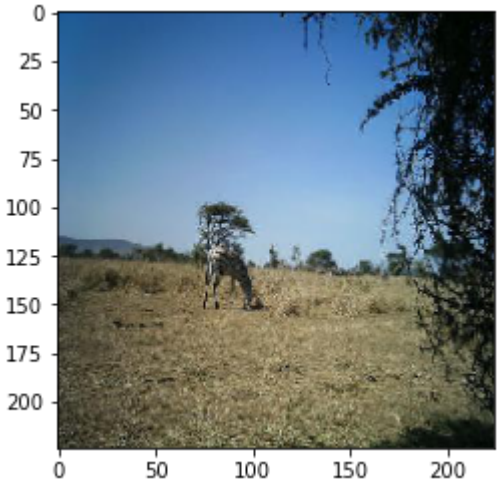
We test the classifier on a new image-if the prediction probability is smaller than 0.5, the prediction will be elephant, and zebra otherwise. We plot the new image to see whether the prediction was correct.

In [26]:
```
def visualize_predictions(classifier, n_cases):
    for i in range(0,n_cases):
        path = random.choice([elephant_test_dir, zebra_test_dir])

        # Get picture
        random_img = random.choice(os.listdir(path))
        img_path = os.path.join(path, random_img)
        img = image.load_img(img_path, target_size=(img_width, img_height))
        img_tensor = image.img_to_array(img)  # Image data encoded as integers
in the 0-255 range
        img_tensor /= 255.  # Normalize to [0,1] for plt.imshow application

        # Extract features
        features = conv_base.predict(img_tensor.reshape(1,img_width, img_heigh
t, 3))

        # Make prediction
        try:
            prediction = classifier.predict(features)
        except:
            prediction = classifier.predict(features.reshape(1, 7*7*512))

        # Show picture
        plt.imshow(img_tensor)
        plt.show()

        # Write prediction
        if prediction < 0.5:
            print('elephant')
        else:
            print('zebra')
```

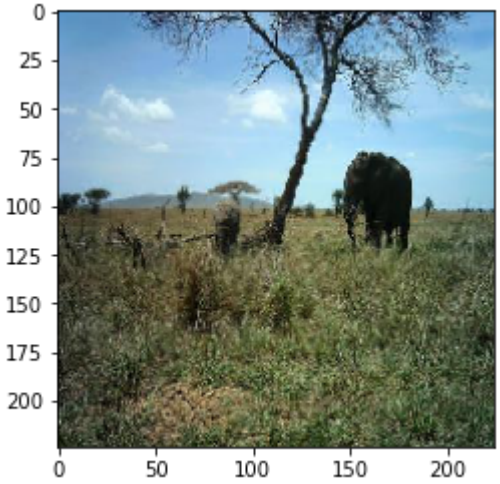
In [28]:
```
# Visualize predictions
visualize_predictions(model, 5)
```



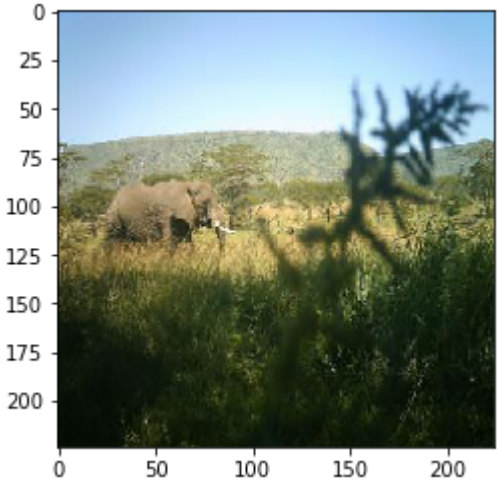
zebra



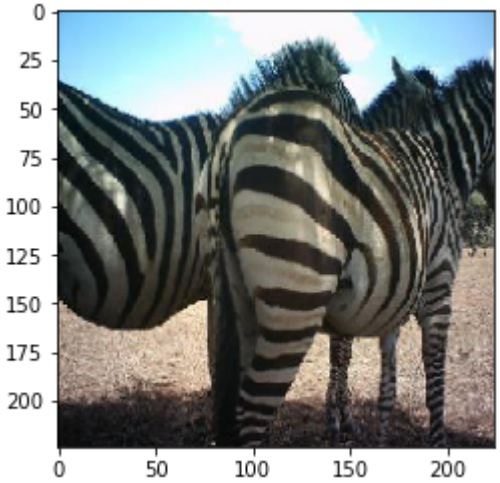
zebra



elephant




```
elephant
```




```
zebra
```

Dictionary

ResNet: A type of convolutional neural network.

VGGNet: A type of convolutional neural network.

ImageNet: database of images, contains over 1.2 million images, organized into hierarchical categories.

MNIST: handwritten digit database.

CelebA: a large-scale (200K) face attribute dataset.

Google Cloud Vision: computer vision software, pre-trained machine learning models.

Clarifai: image content identification and analyzation software.

YOLOV2: real-time object detection software.

PyTorch: free and open-source machine learning python library, used for computer vision and NLP.

Tensorflow: free and open-source machine learning programming library, focuses on deep neural network.

scikit-image: free and open-source python image processing library.

OpenCV: free and open-source computer vison aimed programming library.