Image Analytics in Marketing Code Examples

This notebook is a toturial 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, Arndt 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 Jupiter 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.

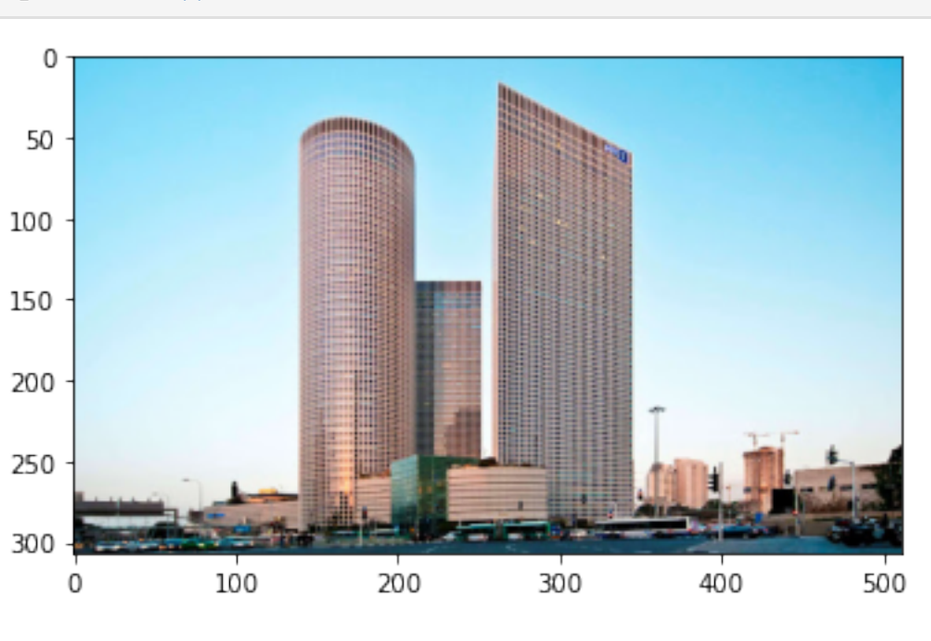
```
import numpy as np
import pandas as pd
import os
import shutil
import random
import cv2 # needs to be installed separately
import matplotlib.pyplot as plt
from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.feature import hog
from skimage import exposure
import 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.

```
img = mpimg.imread('azreali.jpg')
imgplot = plt.imshow(img)
plt.show()
```



Color Extraction

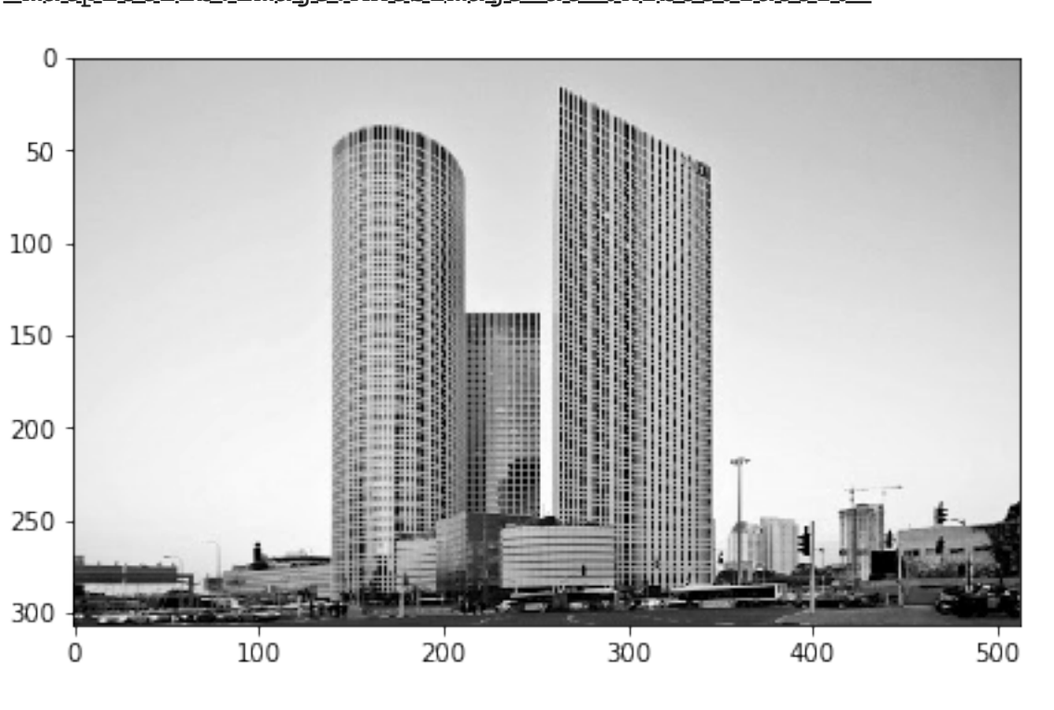
Color is a fundamental image trait. Here are some examples for creating color histograms, which capture the the color composition of the image.

Grayscale Histogram

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

```
img_gray = imread('azreali.jpg',as_gray=True)
imshow(img_gray)
```

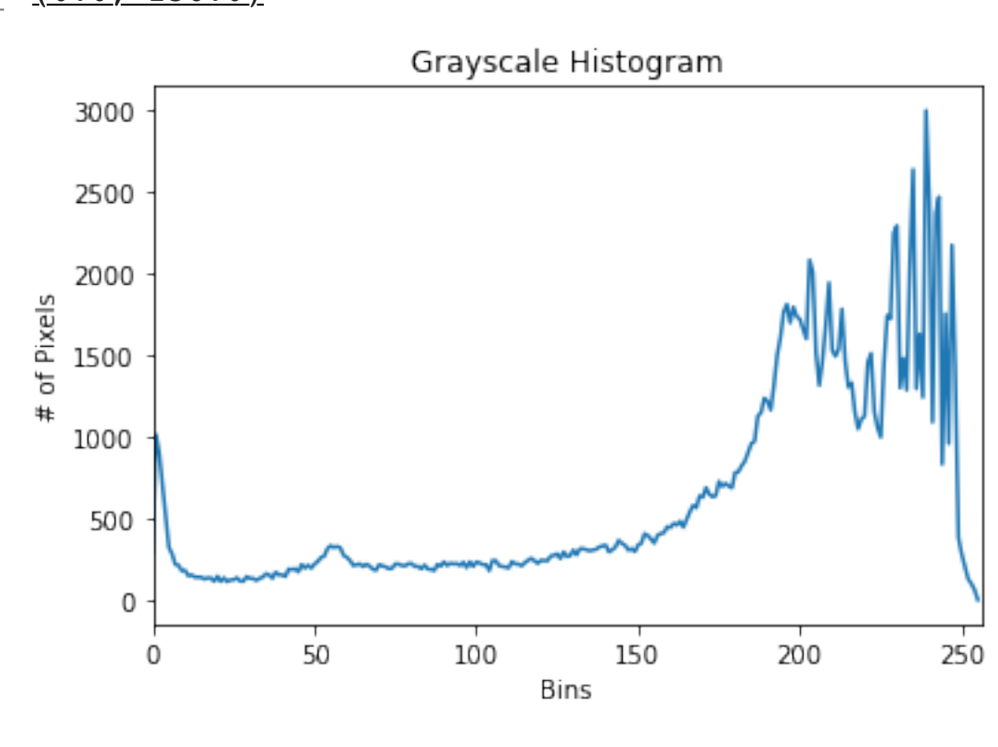
```
<matplotlib.image.AxesImage at 0x2bce01acc10>
```



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?

```
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])
```

```
(0.0, 256.0)
```

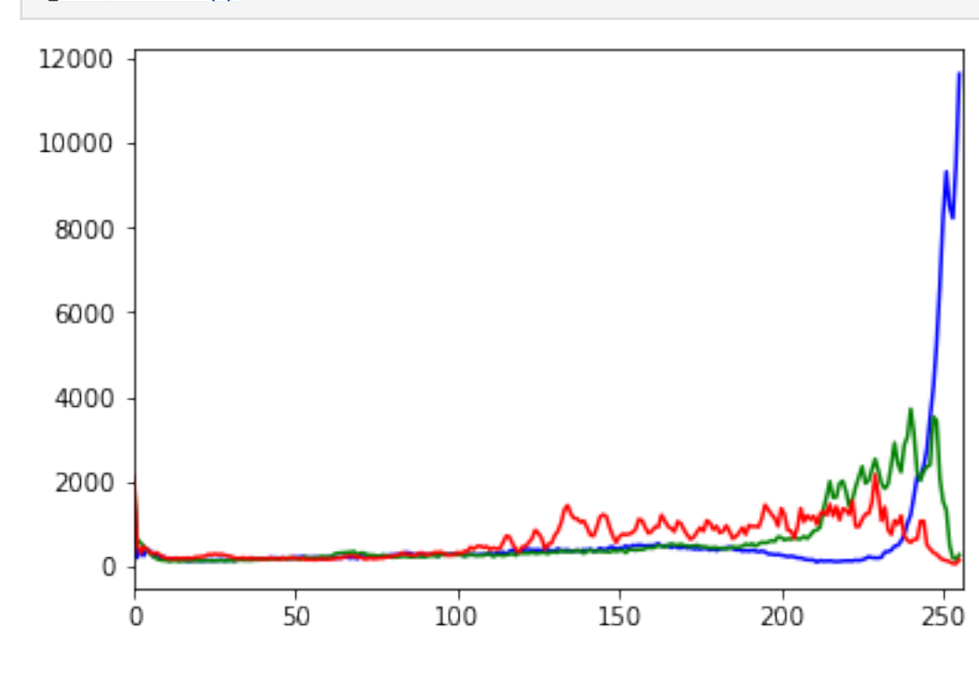


Indeed, the histogram is skewed to the right. The picture contains mainly pixels on the light side of the scale and relatively few dark pixels.

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?

```
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

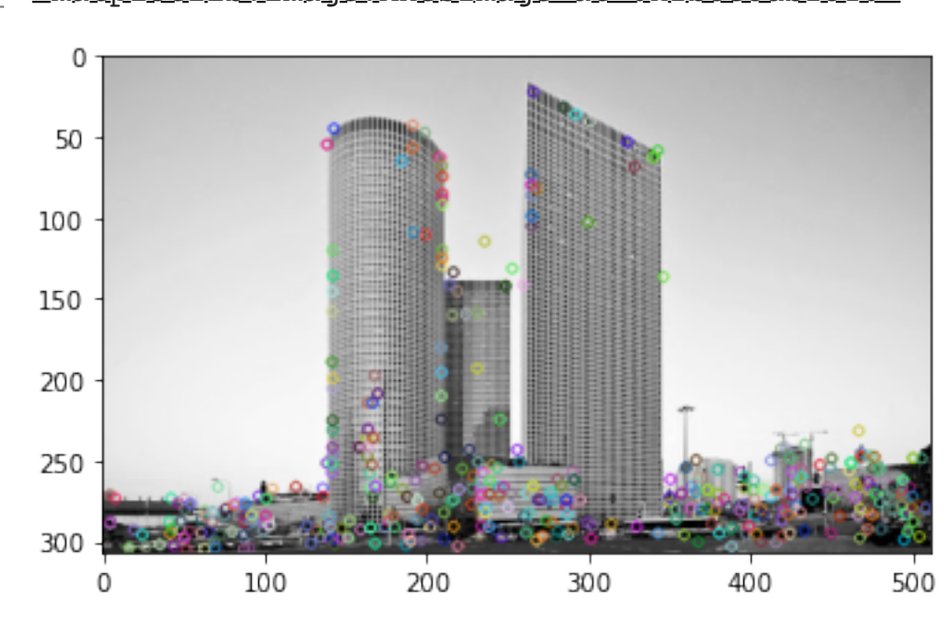
SIFT identifies the keypoints of the image, that usually lie on object edges. The extracted keypoints can be used to detect the object in the picture and separate it from the background.

```
#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)
```

```
<matplotlib.image.AxesImage at 0x2bce3dbf310>
```

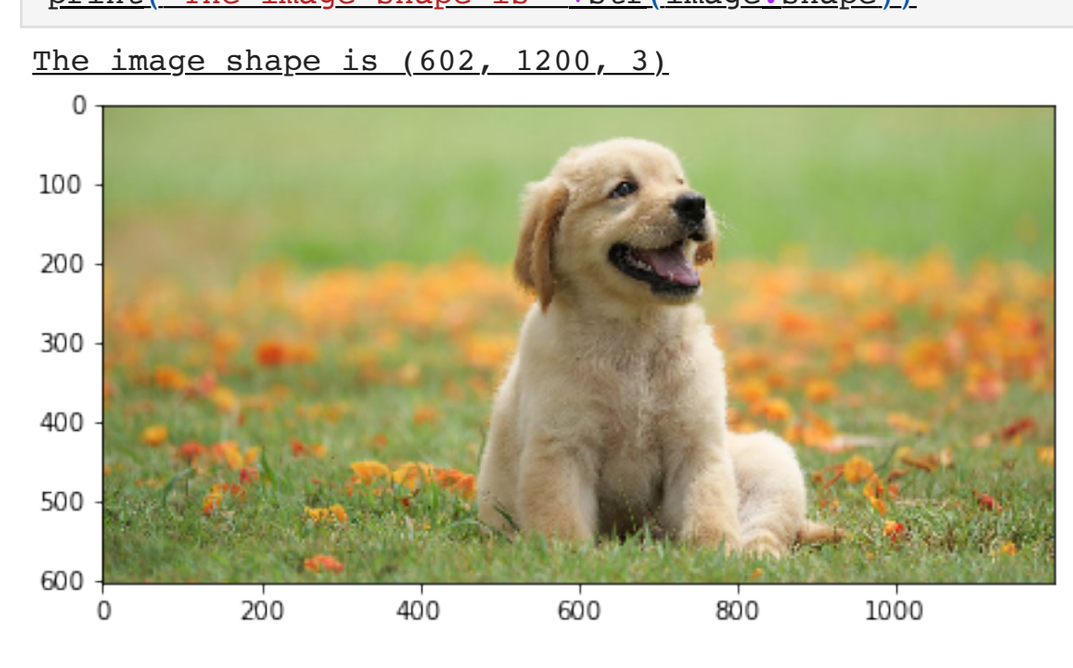


HOG: Histogram of Oriented Gradients

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

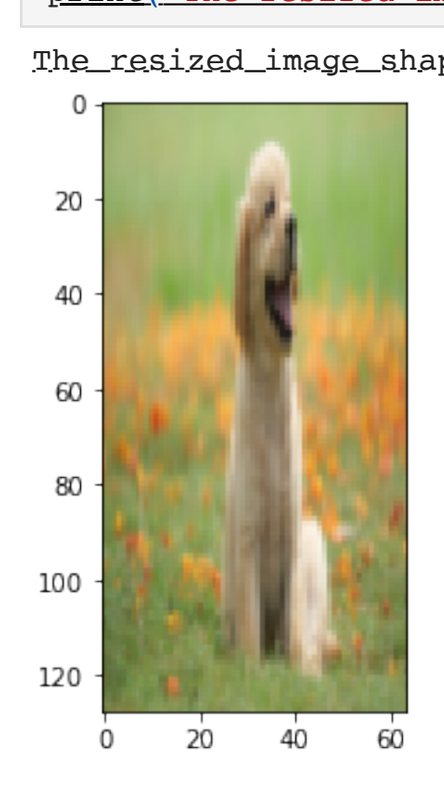
```
#reading the image
image = imread('dog.jpg')
imshow(image)
print('The image shape is '+str(image.shape))
```

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



```
#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)
```



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

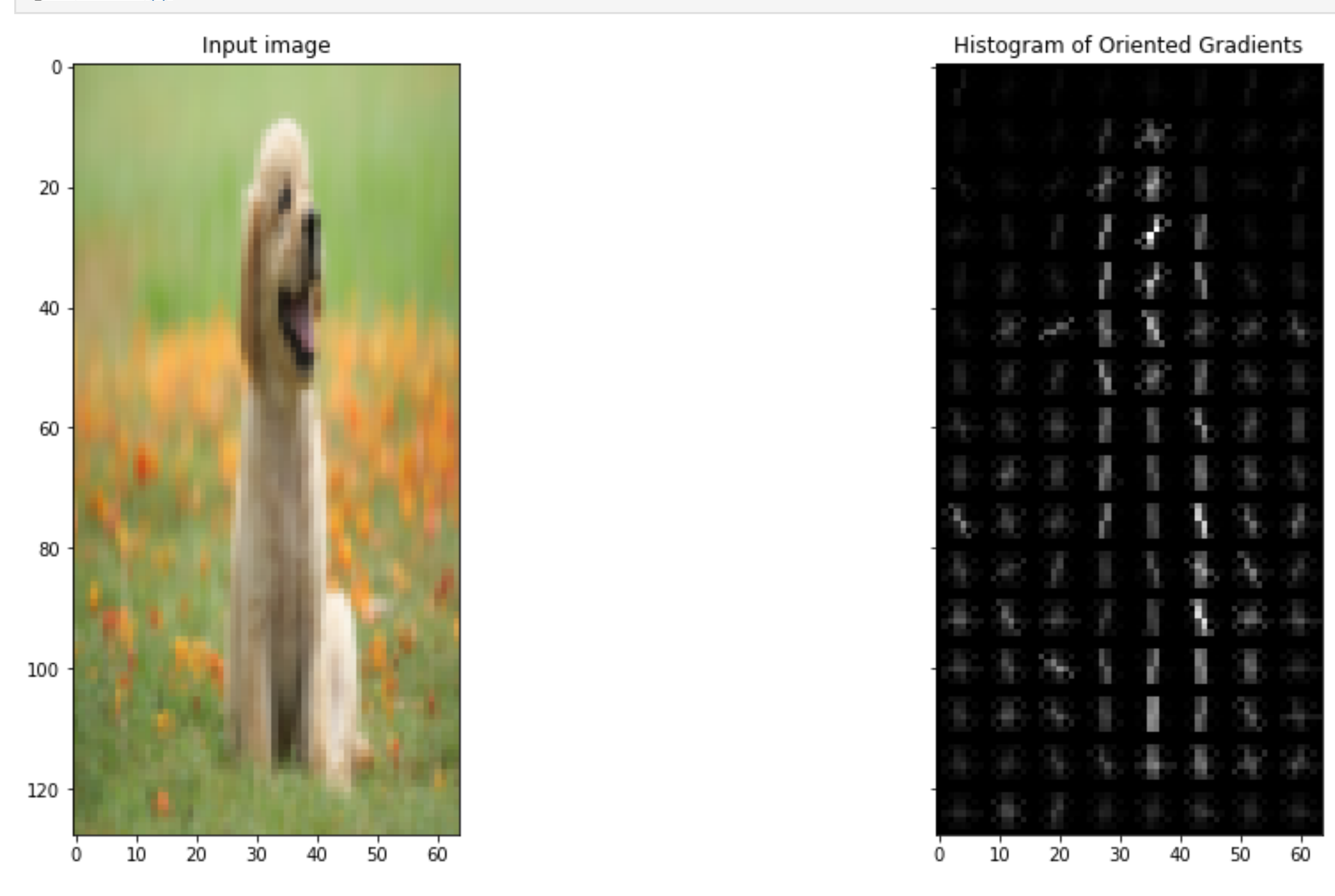
```
# Rescale histogram for better display
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8), sharex=True, sharey=True)

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

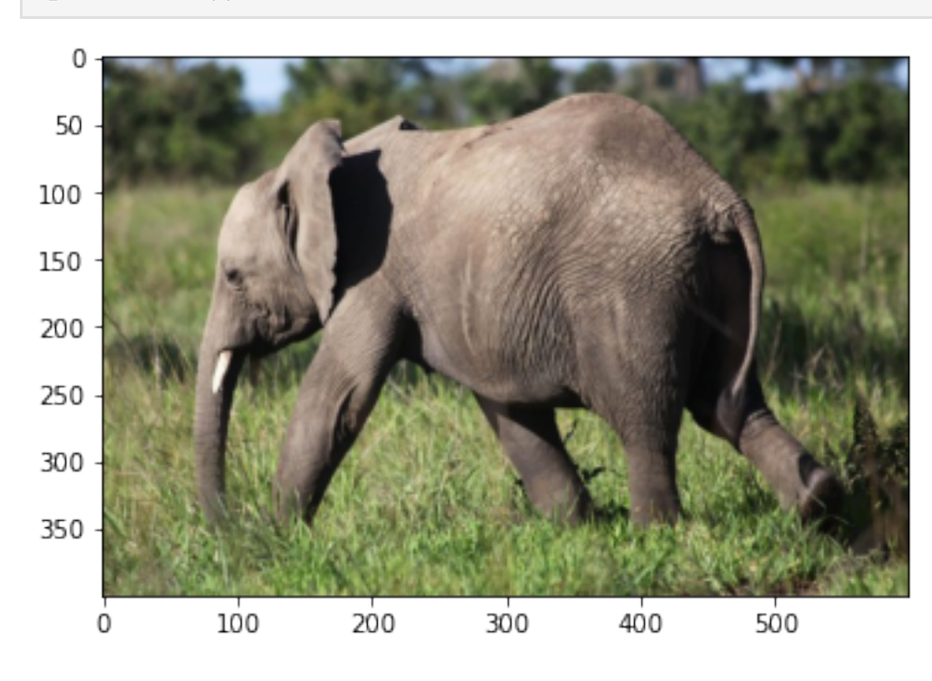
As described in the chapter, some image analytic tasks require interpretable features. One way to obtain these features is by using tagging software. Unlike colors, shape and texture, which decompose the image to its visual elements, image tags are "high level" features which identify objects, activities,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

```
import clarifai # needs to be installed separately
from clarifai.rest import ClarifaiApp

#creating an app of clarifai with the API key from the acount created
API_KEY='689d360c6a4342e19365bfc53c21593b'
app = ClarifaiApp(api_key=API_KEY)
```

```
#showing the image
img = mpimg.imread("elaphent.jpg")
imgplot = plt.imshow(img)
plt.show()
```



```
#using genreal model of Clarifai to extract taggs from the image
model = app.public_models.general_model
response = model.predict_by_filename("elaphent.jpg")
```

```
#printing tags and tag probability. The probability is the confidence level that the image indeed contains the tag.
if(response['status']['description'] == "Ok"):
    for concept in response["outputs"][0]["data"]["concepts"]:
        name = concept["name"]
        value = concept["value"]
        print(name + " " + str(value))
```

```
wildlife 0.99827564
elephant 0.9976928
mammal 0.9962359
no person 0.99166167
ivory 0.98950046
grass 0.98490936
african elephant 0.9844232
safari 0.9802783
nature 0.98003685
animal 0.9771493
large 0.9696601
wild 0.96826786
outdoors 0.953126
trunk 0.9451795
barbaric 0.944115
endangered species 0.94366425
savanna 0.9420985
Kruger 0.9286655
grassland 0.9276549
immense 0.92535317
```

Convolutional Neural Network Classifier

This part was written based on the tutorial: 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.

```
# Copy next 1000 elephants images to test_elephants_dir
# path=r'C:\Users\Owner\Dropbox\14- Market Research Book Chapter\ful data\train_elephants'
# files=os.listdir(path)
# src=np.random.choice(files,1000,replace=False)
# dst=r'C:\Users\Owner\Dropbox\14- Market Research Book Chapter\ful data\test_elephants'
# for f in src:
#       shutil.move(path+'\\'+f, dst)
```

```
# path=r'C:\Users\Owner\Dropbox\14- Market Research Book Chapter\ful data\train_zebras'
# files=os.listdir(path)
# src=np.random.choice(files,1000,replace=False)
# dst=r'C:\Users\Owner\Dropbox\14- Market Research Book Chapter\ful data\test_zebras'
# for f in src:
#       shutil.move(path+'\\'+f, dst)
```

```
# # Copy next 1000 elephants images to test_elephants_dir
# path=r'E:\Dropbox\14- Market Research Book Chapter\ful data\train\elephant'
# files=os.listdir(path)
# src=np.random.choice(files,200,replace=False)
# dst=r'E:\Dropbox\14- Market Research Book Chapter\ful data\test\elephant'
# for f in src:
#       shutil.move(path+'\\'+f, dst)
```

Importing libraries and defining directories

```
# tensorflow and keras need to be installed separately
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras.preprocessing import image
```

```
# place the name, change the name
base_dir = r'E:\Dropbox\14- Market Research Book Chapter\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)))
```

```
img_width, img_height = 224, 224  # Default input size for VGG16
train_size=10599
validation_size=2000
test_size=400
```

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.

```
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 the 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 subdirectories class_elephant and class_zebra,
    #together with labels 0 and 1 (0 corresponding to class_elephant and 1 corresponding 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
```

Instantiating the pre-trained convolutional base (VGG16)

```
#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 channels in RGB pictures
```

```
# Check architecture
conv_base.summary()
```

```
#consumes computational power
#creating training features
train_features, train_labels = extract_features(train_dir, train_size)
```

```
#creating validation features
validation_features, validation_labels = extract_features(validation_dir, validation_size)
```

```
train_labels
```

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.

```
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))
```

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.

```
# 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.

```
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_height, 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')
```

```
# Visualize predictions
visualize_predictions(model, 5)
```

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.