DIGIT RECOGNITION

Algorithm for recognition of handwritten digits

In many areas of life there is a need for the computers to recognize digits from a paper, including the handwritten digits.
For this purpouse, machine-learning algorithms on the basis of **image recognition** can be used.
Algorithms for image recognition use **convolutional networks** with different amount of layers, depending on the complexity of the task.

I. Importing keras methods for creating convolutional network from the tensorflow library

Tensorflow library comes with already created convolutional networks, that are ready to use.
The dataset, used for training is from MNIST (a database of handrwitten digits, 60 000 train and 10 000 test images). It is included inside the keras package.

In [1]:

```
# Import the modules
from sklearn.externals import joblib
from sklearn import datasets
from skimage.feature import hog
import numpy as np
from collections import Counter
import cv2
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Conv2D, BatchNormalization, Dense,Flatten, Drop
out, MaxPooling2D
from keras.datasets import mnist
```

```
D:\Marko\ML\Anaconda3\lib\site-packages\numpy\core\__init__.py:29: UserWarning:
loaded more than 1 DLL from .libs:
D:\Marko\ML\Anaconda3\lib\site-packages\numpy\.libs\libopenblas.CSRRD7HKRKC3T3YX
A7VY7TAZGLSWDKW6.gfortran-win_amd64.dll
D:\Marko\ML\Anaconda3\lib\site-packages\numpy\.libs\libopenblas.IPBC74C7KURV7CB2
PKT5Z5FNR3SIBV4J.gfortran-win_amd64.dll
  stacklevel=1)
Using TensorFlow backend.
```

Loading the MNIST train dataset inside of a zip object

In [2]:

```
(xtrain, ytrain), (xtest, ytest) = mnist.load_data()
union_list = list(zip(xtrain, ytrain))
np.random.shuffle(union_list)
train_numbers, train_number_labels = zip(*union_list)
train_numbers = np.array(train_numbers)
train_number_labels = np.array(train_number_labels)
```

II. Building the convolutional neural network model

Layers are ready to use from keras.
The size of the images from the MNIST database was 28, 28 (grayscale).
For image recognition, very usually are used the decreasing convolution networks with relu activation function.
Function, used for classification is softmax, because we have 10 classes of digits.

In [4]:

```
model = Sequential([
    Conv2D(128, (7,7), activation = 'relu', padding = 'Same', input_shape = (28, 28
, 1)),
    MaxPooling2D(),
    Conv2D(64, (5, 5), activation = 'relu', padding = 'Same'),
    MaxPooling2D(),
    Conv2D(32, (3, 3), activation = 'relu', padding = 'Same'),
    Flatten(),
    Dense(256, activation = 'relu'),
    Dropout(0.2),
    Dense(10, activation = 'softmax')
])
```

```
WARNING:tensorflow:From D:\Marko\ML\Anaconda3\lib\site-packages\tensorflow\pytho
n\ops\resource_variable_ops.py:435: colocate_with (from tensorflow.python.framew
ork.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From D:\Marko\ML\Anaconda3\lib\site-packages\tensorflow\pytho
n\keras\layers\core.py:143: 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`.
```

Sparse categorical crossentropy as a loss function is used, because there are the 10 classes of numbers, which are needed to be recognized as a separate individual objects, and the result is encoded as single integer, starting from 0 to the number of classes munis one.

The Adam optimizer is an adaptive learning rate algorithm that’s been designed specifically for training deep neural networks. It is the latest trend in deep learning optimization. Adam optimizing algorithm was created on the basis of the stochastic gradient descent with momentum, combined with RMSprop.

Accuracy metrics calculates the probability of correctly predicted classes from the complete set (softmax).

In [5]:

```
model.compile(loss = 'sparse_categorical_crossentropy', optimizer = Adam(lr = 0.000
1), metrics = ['accuracy'])
```

III. Training the model

Reshaping the train array as an input for the model

In [6]:

```
train_numbers = train_numbers.reshape(-1, 28, 28, 1)
xtest = xtest.reshape(-1, 28, 28, 1)
```

The training parameters can be reccorded to a file, so that after trainig they can be accessed afterwards and used without the need to train the model again.

```
In [32]: # SKIP IF TRAINED
         import os
         import tensorflow as tf
         checkpoint_path = "training_1/cp.ckpt"
         checkpoint_dir = os.path.dirname(checkpoint_path)
         # Create checkpoint callback
         cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path, save_weights_only
         =True, verbose=1)
         model.fit(train_numbers, train_number_labels, batch_size = 512, epochs = 3,
                   callbacks = [cp_callback])  # pass callback to training
```

```
Epoch 1/3
59904/60000 [============================>.] - ETA: 2s - loss: 0.0707 - acc: 0.9
778
Epoch 00001: saving model to training_1/cp.ckpt
WARNING:tensorflow:This model was compiled with a Keras optimizer (<tensorflow.p
ython.keras.optimizers.Adam object at 0x0000002BF3AE3390>) but is being saved in
TensorFlow format with `save_weights`. The model's weights will be saved, but un
like with TensorFlow optimizers in the TensorFlow format the optimizer's state w
ill not be saved.

Consider using a TensorFlow optimizer from `tf.train`.
WARNING:tensorflow:From D:\Marko\ML\Anaconda3\lib\site-packages\tensorflow\pytho
n\keras\engine\network.py:1436: update_checkpoint_state (from tensorflow.python.
training.checkpoint_management) is deprecated and will be removed in a future ve
rsion.
Instructions for updating:
Use tf.train.CheckpointManager to manage checkpoints rather than manually editin
g the Checkpoint proto.
60000/60000 [==============================] - 1388s 23ms/sample - loss: 0.0707
- acc: 0.9778
Epoch 2/3
59904/60000 [============================>.] - ETA: 2s - loss: 0.0551 - acc: 0.9
825
Epoch 00002: saving model to training_1/cp.ckpt
WARNING:tensorflow:This model was compiled with a Keras optimizer (<tensorflow.p
ython.keras.optimizers.Adam object at 0x0000002BF3AE3390>) but is being saved in
TensorFlow format with `save_weights`. The model's weights will be saved, but un
like with TensorFlow optimizers in the TensorFlow format the optimizer's state w
ill not be saved.

Consider using a TensorFlow optimizer from `tf.train`.
60000/60000 [==============================] - 1405s 23ms/sample - loss: 0.0551
- acc: 0.9825
Epoch 3/3
59904/60000 [============================>.] - ETA: 2s - loss: 0.0441 - acc: 0.9
854
Epoch 00003: saving model to training_1/cp.ckpt
WARNING:tensorflow:This model was compiled with a Keras optimizer (<tensorflow.p
ython.keras.optimizers.Adam object at 0x0000002BF3AE3390>) but is being saved in
TensorFlow format with `save_weights`. The model's weights will be saved, but un
like with TensorFlow optimizers in the TensorFlow format the optimizer's state w
ill not be saved.

Consider using a TensorFlow optimizer from `tf.train`.
60000/60000 [==============================] - 1422s 24ms/sample - loss: 0.0440
- acc: 0.9855
```

```
Out[32]: <tensorflow.python.keras.callbacks.History at 0x2bfe8e0390>
```

The pre-trained parameters can be loaded directly to the program.

```
In [7]:  checkpoint_path = "training_1/cp.ckpt"
         model.load_weights(checkpoint_path)

Out[7]:  <tensorflow.python.training.checkpointable.util.CheckpointLoadStatus at 0xe47850
         6cc0>
```

```
In [27]: # TRAIN MODEL
         model.fit(train_numbers, train_number_labels, batch_size = 512, epochs = 3)

         Epoch 1/3
         60000/60000 [==============================] - 1385s 23ms/sample - loss: 1.1667
         - acc: 0.7620
         Epoch 2/3
         60000/60000 [==============================] - 1382s 23ms/sample - loss: 0.1675
         - acc: 0.9478
         Epoch 3/3
         60000/60000 [==============================] - 1393s 23ms/sample - loss: 0.1002
         - acc: 0.9684

Out[27]: <tensorflow.python.keras.callbacks.History at 0x2bfd893d30>
```

Additionally, there is a classifier used speciffically for recognizing digits, that can be used as a mean to categorize images, based on a pre-trained model.

```
In [3]:  # Load the classifier
         clf = joblib.load("digits_cls.pkl")

         # Read the input image
         im = cv2.imread("photo_2.jpg")
```

IV. Finding the digits inside an image file and recognizing them

-> first, the image is loaded
-> next, the image is converted to grayscale
-> the threshold and contours are found
-> the individual digits are surrounded by a rectangle
-> the separated images are resized to an appropriate size for the recognition model
-> and, finally, the recognized digits are labelled

In [8]:

```
# Convert to grayscale and apply Gaussian filtering
im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
im_gray = cv2.GaussianBlur(im_gray, (5, 5), 0)

# Threshold the image
ret, im_th = cv2.threshold(im_gray, 90, 255, cv2.THRESH_BINARY_INV)
#cv2.imshow("Resulting Image", im_th)
# Find contours in the image
ctrs, hier = cv2.findContours(im_th.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIM
PLE)

# Get rectangles contains each contour
rects = [cv2.boundingRect(ctr) for ctr in ctrs]

# For each rectangular region, calculate HOG features and predict
# the digit using Linear SVM.
for rect in rects:
    # Draw the rectangles
    cv2.rectangle(im, (rect[0], rect[1]), (rect[0] + rect[2], rect[1] + rect[3]), (
0, 255, 0), 3)
    # Make the rectangular region around the digit
    leng = int(rect[3] * 1.6)
    pt1 = int(rect[1] + rect[3] // 2 - leng // 2)
    pt2 = int(rect[0] + rect[2] // 2 - leng // 2)
    roi = im_th[pt1:pt1+leng, pt2:pt2+leng]
    # Resize the image
    roi = cv2.resize(roi, (28, 28), interpolation=cv2.INTER_AREA)

    '''
    #If we want to use the pre-trained model, this code could be uncommented
    roi = cv2.dilate(roi, (3, 3))
    # Calculate the HOG features
    roi_hog_fd = hog(roi, orientations=9, pixels_per_cell=(14, 14), cells_per_block
=(1, 1), visualise=False)
    nbr = clf.predict(np.array([roi_hog_fd], 'float64'))
    '''

    roismall = roi.reshape(-1,28,28,1)
    result = model.predict_classes(roismall)

    cv2.putText(im, str(int(result[0])), (rect[0], rect[1]),cv2.FONT_HERSHEY_DUPLEX
, 2, (0, 255, 255), 3)

cv2.imshow("Resulting Image with Rectangular ROIs", im)
cv2.waitKey()
```

Out[8]:

```
-1
```

In [ ]: