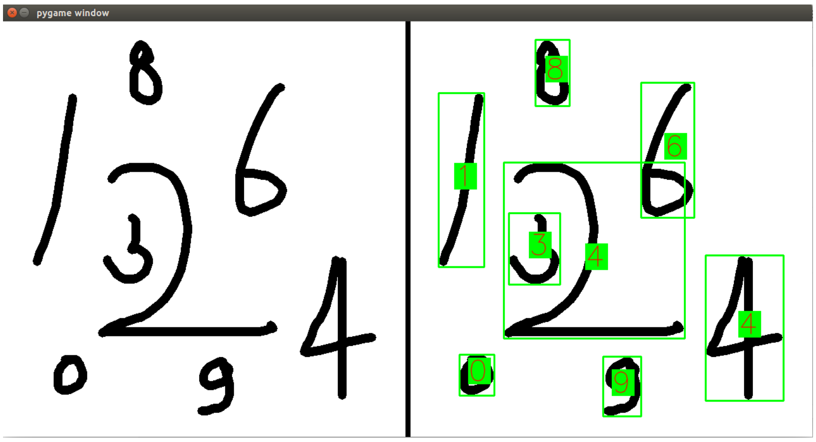
Digit OCR



In this project I am going to

build an OCR notebook.

It extracts every digit from the image and returns

the right number that containes 5 digits.

I devided this project in five steps

Step1:

data preparation

Step2:

Build our Convulatonal-Neural-Network

Step3:

Training our CNN on MNIST Dataset

Step4:

Test our Model with Random Picture

Step5:

Developing the OCR Engine Logic

Step1 : Data Preparation

In [2]:

```
'''
import all package needed in this project
tensorflor: a free and open-source software library for machine learning and artificial intelligence
keras: Keras acts as an interface for the TensorFlow library.
matplotlib: is a plotting library for the Python programming language and its numerical mathematics extension NumPy
numpy: the fundamental package for scientific computing in Python.
cv2:  is a library of programming functions mainly for real-time computer vision
'''


import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from keras.models import Sequential
import os
import numpy as np
from keras.models import load_model
```

```python
import cv2
from keras.datasets import mnist
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils
import keras.models as models
```

In [3]:

```python
'''

create a function that normalises the data to train the CNN model faster (on [0..1] data instead of on [0..255]).

'''
def normalize_x(train: np.ndarray, test: np.ndarray):
        train = train / 255
        test = test / 255
        return train, test

'''

load the the Mnist Digit dataset (already splitted)
X_train contains 70000 pictures used to train the model
X_test contains 10000 pictures used to evaluate the model

'''
(X_train, y_train), (X_test, y_test) = mnist.load_data()


'''
The cnn model needs a matrix with dimention (28,28,1) as input,
so I reshape all data to 28x28x1 3D matrices.

We need an extra dimension in the end which corresponds to channels.
MNIST images are gray scaled so it uses only one channel.
For RGB images, there are 3 channels. Here we would have reshaped 784px vectors to 28x28x3 3D matrices.


'''
X_train = X_train.reshape((-1, 28, 28, 1)).astype('float32')
X_test = X_test.reshape((-1, 28, 28, 1)).astype('float32')

'''

normalize inputs from 0-255 to 0-1

'''
X_train, X_test = normalize_x(X_train, X_test)


'''

Labels are 10 digits numbers from 0 to 9.
We need to encode these lables to one hot vector (ex : 2 -> [0,0,1,0,0,0,0,0,0,0]).


'''
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

In [4]:

```python
'''
https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

generating data augumentation with ImageDataGenerator

More data, more accuracy, better consistent estimators!


'''

VALIDATION_FRACTION_SIZE=0.10
ROTATION_RANGE=10
ZOOM_RANGE=0.10
SHIFT_RANGE=0.10
SHEAR_RANGE=0.10

datagen = ImageDataGenerator(
        rotation_range=ROTATION_RANGE,
        zoom_range=ZOOM_RANGE,
        width_shift_range=SHIFT_RANGE,
        height_shift_range=SHIFT_RANGE,
        shear_range=SHEAR_RANGE,
        validation_split=VALIDATION_FRACTION_SIZE,
)
```

Step2 : Build our Convolutional neural network

In [5]:

```python
"""
We can build an easy model composed of different layers such as:

Conv2D: (30 filters of size 5 by 5). The features will be "extracted" from the image.
MaxPooling2D: The images get half sized.
Flatten: Transforms the format of the images from a 2d-array to a 1d-array of 28 28 1 pixel values.
Relu : given a value x, returns max(x, 0).
Softmax: 10 neurons, probability that the image belongs to one of the classes.

"""

# Constants for hyperparameters
BATCH_SIZE=100
EPOCHS=10
```

```
model = Sequential()
model.add(Conv2D(30, (5, 5), input_shape=(28, 28, 1), activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(15, (3, 3), activation='relu'))
model.add(MaxPooling2D())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='softmax'))
# Compile model

'''

Then, we can compile it with some parameters such as:
Optimizer: adam
Loss function: we use sparse categorical crossentropy for classification, each images belongs to one class only

'''
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Step3: Training our CNN on MNIST dataset and the generated dataset

In [6]:
```
'''

We fit the model to the data from the training and the generated data set.
The neural network will learn the pattern by itself in order to distinguish each category.

'''


model.fit(
        datagen.flow(
                X_train, y_train,
                batch_size=BATCH_SIZE,
                subset='training'
        ),
        validation_data=datagen.flow(
                X_train, y_train,
                batch_size=BATCH_SIZE,
                subset='validation'
        ),
        steps_per_epoch=(X_train.shape[0] * (1 - VALIDATION_FRACTION_SIZE)) // BATCH_SIZE,
        epochs=EPOCHS,
)
```

```
Epoch 1/10
540/540 [==============================] - 75s 129ms/step - loss: 0.5097 - accuracy: 0.8349 - val_loss: 0.1498 - val_accuracy: 0.9575
Epoch 2/10
540/540 [==============================] - 48s 88ms/step - loss: 0.1688 - accuracy: 0.9482 - val_loss: 0.1103 - val_accuracy: 0.9683
Epoch 3/10
540/540 [==============================] - 44s 82ms/step - loss: 0.1236 - accuracy: 0.9611 - val_loss: 0.0887 - val_accuracy: 0.9732
Epoch 4/10
540/540 [==============================] - 52s 96ms/step - loss: 0.1023 - accuracy: 0.9677 - val_loss: 0.0787 - val_accuracy: 0.9775
Epoch 5/10
540/540 [==============================] - 41s 77ms/step - loss: 0.0875 - accuracy: 0.9732 - val_loss: 0.0592 - val_accuracy: 0.9820
Epoch 6/10
540/540 [==============================] - 43s 80ms/step - loss: 0.0823 - accuracy: 0.9749 - val_loss: 0.0637 - val_accuracy: 0.9800
Epoch 7/10
540/540 [==============================] - 44s 82ms/step - loss: 0.0731 - accuracy: 0.9773 - val_loss: 0.0580 - val_accuracy: 0.9840
Epoch 8/10
540/540 [==============================] - 45s 83ms/step - loss: 0.0685 - accuracy: 0.9793 - val_loss: 0.0538 - val_accuracy: 0.9828
Epoch 9/10
540/540 [==============================] - 42s 78ms/step - loss: 0.0659 - accuracy: 0.9786 - val_loss: 0.0513 - val_accuracy: 0.9857
Epoch 10/10
540/540 [==============================] - 40s 74ms/step - loss: 0.0623 - accuracy: 0.9810 - val_loss: 0.0512 - val_accuracy: 0.9857
```

Out[6]:
```
<keras.callbacks.History at 0x16e8935d9f0>
```

In [7]:
```
'''
save the model

'''

model.save('classifier.h5')
```

In [8]:
```
'''
We should evaluate the model performance on a test dataset set

'''
scores = model.evaluate(X_test, y_test, verbose=0)
print("Large CNN Error: %.2f%%" % (100-scores[1]*100))
print("Accuracy on the testing dataset: %.2f%%" % (scores[1]*100))
```

```
Large CNN Error: 0.88%
Accuracy on the testing dataset: 99.12%
```

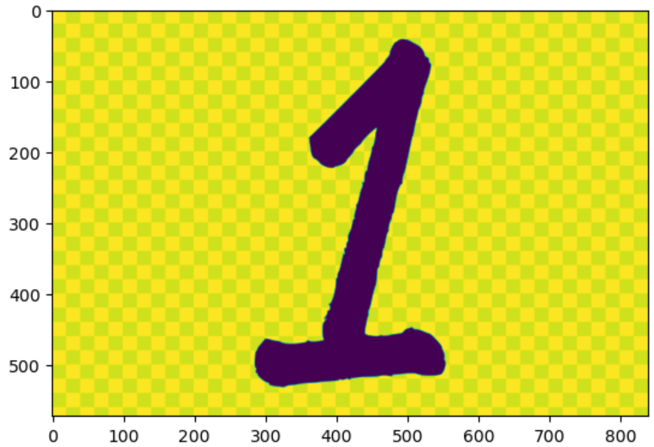
Step4: Test our model with a random picture

`In this step, I test my model on random number pictures.`

In [10]:
```
"""

    Before predicting the image we need to preprocess the image.
    The preprocessing of the incoming images shoud be performed in the same way that
    images in the MNIST dataset were processed.
    This code was adapted from this link
```

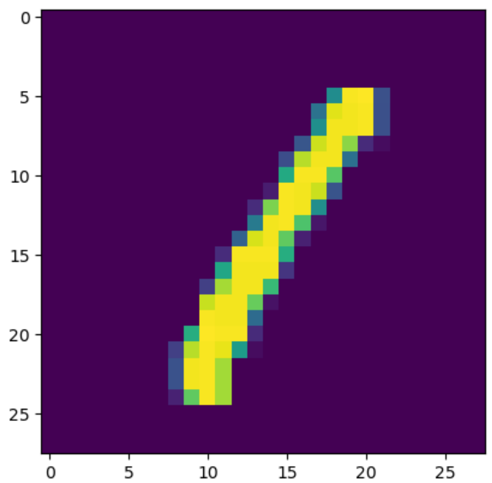
```
    http://opensourc.es/blog/tensorflow-mnist

    https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

    I take an example number picture and I predict the number

"""

'''
Preproceccing step:

    - read the image
    - image thresholding because the images doesn't look like the trained ones. Make the background black and the numbers white
    - resize the image
    - normalize the image

'''

img = cv2.imread("one.png", cv2.IMREAD_GRAYSCALE)
(thresh, gray) = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
gray = cv2.resize(255-gray, (28, 28))
data = (gray / 255).reshape((1, 28, 28, 1))



prediction = model.predict(data)
classes_x = np.argmax(prediction, axis=1)
print(f"The predicted number is ===> {classes_x[0]}")
fig = plt.figure
plt.imshow(img)
plt.show()
```

```
1/1 [==============================] - 0s 25ms/step
The predicted number is ===> 1
```



In [14]:

```
'''
show how a number 1 looks like in my training dataset

'''
fig = plt.figure
plt.imshow(X_train[3])
plt.show()
```



Step5: Developing the OCR engine logic

In this section, I used the contour detection technique with openCV to find numbers on images. I used this step to develop my OCR engine logic.

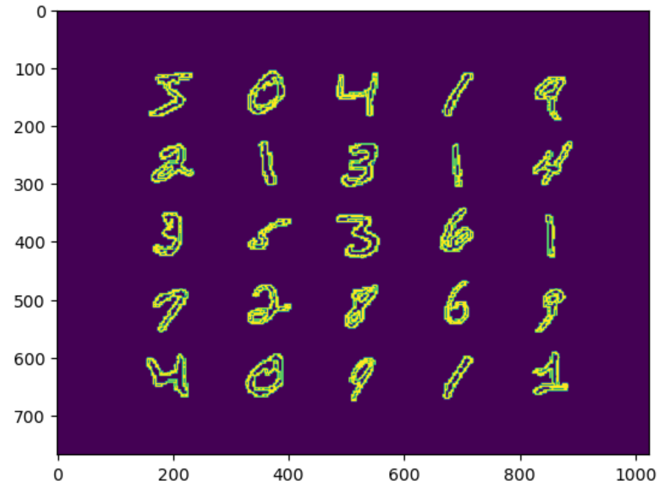
---

step1: Preprocess the main image with the multi-stage algorithm "canny edge detector"
step2: Find and preprocess the contours (like step 4)
step3: Predict and draw the numbers found

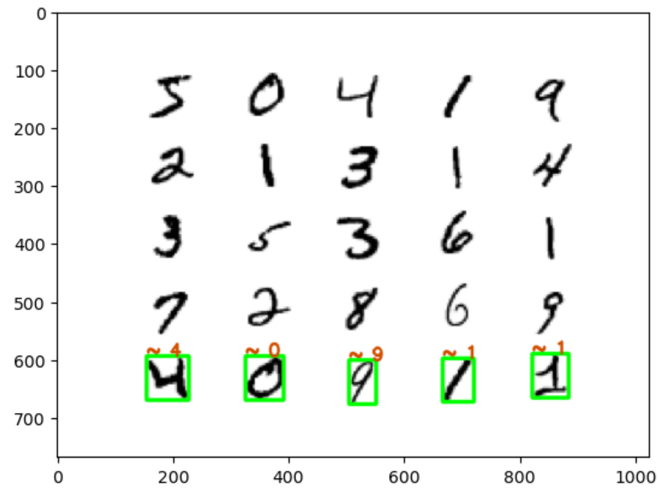
In [15]:
```
'''
Load the model
'''

import keras.models as models

filename = "classifier.h5"

model = models.load_model(filename)
```

In [17]:
```
'''

To find the contours I used the Canny Edge detector. It is one of the most popular techniques for edge detection,
not just because of its simplicity, but also because it generates high-quality results.


'''



#Loading our image
image = cv2.imread("numbers1.png")

#converting it to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# applying the cv2.GaussianBlur to blur the image and remove noise
blurred = cv2.GaussianBlur(gray, (3, 3), 0)

# apply the Canny edge detector. The first argument is the image on which we want to detect the edges.
# The second and third arguments are the thresholds used for the hysteresis procedure.
edged = cv2.Canny(blurred, 10, 100)

#plt.imshow(edged)
#plt.show()

# define a (3, 3) structuring element
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

# apply the dilation operation to the edged image
dilate = cv2.dilate(edged, kernel, iterations=1)


plt.imshow(dilate)
plt.show()

# find the contours in the dilated image
cont, _ = cv2.findContours(dilate, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

'''

In this for loop I tried to predict every contour whether a number was found or not.
If the prediction > 0.7 we draw the contour.


'''

numbers_found = []
for c in cont:

    # find the x coordinate, y coordinate, width and height of the contour
    x,y,w,h = cv2.boundingRect(c)

    # center the image to have better accuracy
    part = image[y-20:y+h+20,x-10:x+w+10]

    #plt.imshow(part)
    #plt.show()


    # int next five lines we write the same code logic like in step 4
    part = cv2.resize(255-part, (28, 28))
    img_ = cv2.cvtColor(part, cv2.COLOR_BGR2GRAY)
    (thresh, gray) = cv2.threshold(img_, 50, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

    #plt.imshow(gray)
    #plt.show()

    data = (gray / 255).reshape((1, 28, 28, 1))
    print(data.shape)
    prediction = model.predict(data)

    # skip low prediction
    if np.max(prediction) <0.7:
        continue
    classes_x = np.argmax(prediction, axis=1)
    numbers_found.append(classes_x)

    # draw contour (c) in green
    cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),5)

    # draw the text
    cv2.putText(
     image,
     "~ "+str(classes_x[0]),
    (x,y),
     cv2.FONT_HERSHEY_SIMPLEX,
     1, #font size
     (209, 80, 0, 255), #font color
     3)
    if len(numbers_found)==5:
        break

# show the result picture and print the 5 numbers
```

```
plt.imshow(image)
plt.show()

print("Die gefundenen Zfiffern sind : " , "".join([str(numbers_found[i][0]) for i in range(len(numbers_found))]))
cv2.imwrite('numbers1_OCR.jpg', image)
```



```
(1, 28, 28, 1)
1/1 [==============================] - 0s 30ms/step
(1, 28, 28, 1)
1/1 [==============================] - 0s 38ms/step
(1, 28, 28, 1)
1/1 [==============================] - 0s 18ms/step
(1, 28, 28, 1)
1/1 [==============================] - 0s 38ms/step
(1, 28, 28, 1)
1/1 [==============================] - 0s 31ms/step
```



```
Die gefundenen Zfiffern sind :  91041
```

Out[17]:

```
True
```

next possible step for this project:

`Billding a web application (Flask/Django) on which we can mount our model`

`Lasty deploying the web application on Heroku usings the connections to github`

In [ ]: