Text Recognition in Images

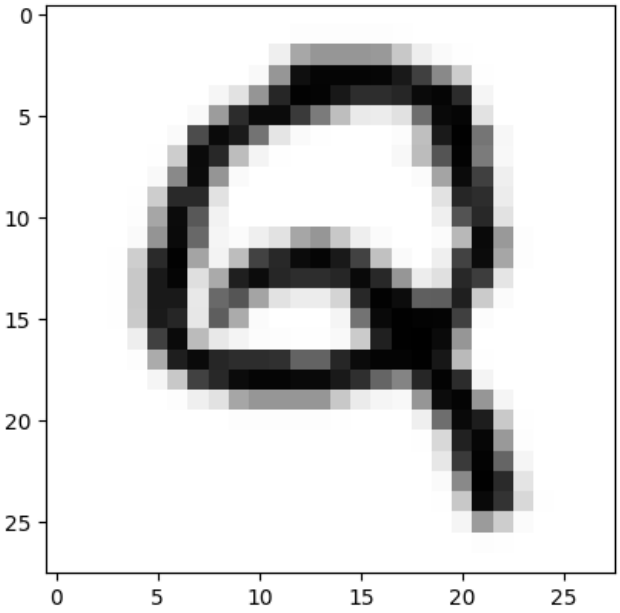
May 30, 2019

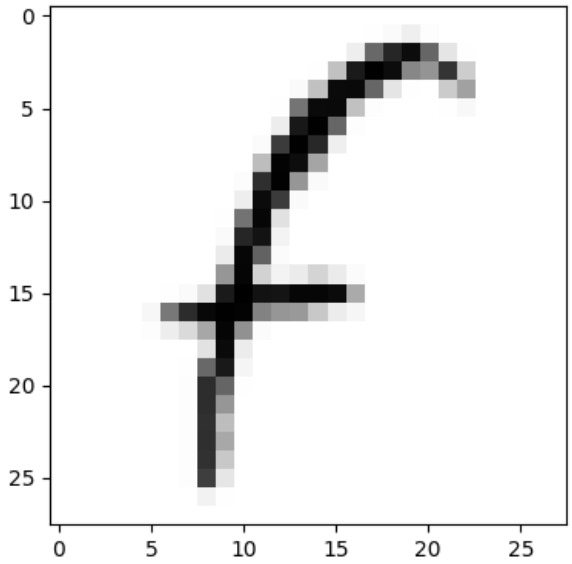
1 Recognizing single characters

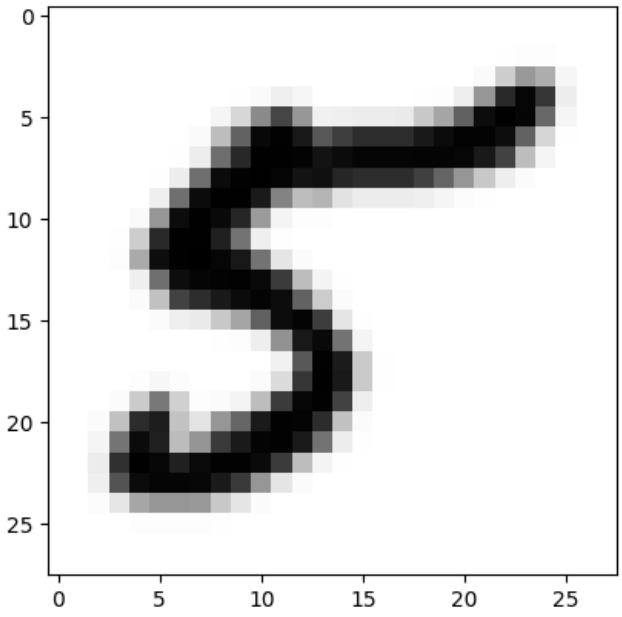
In this brief analysis we focus on the problem of reading some text from an image. To be more precise, in this first stage we will concentrate on recognizing single handwritten characters.

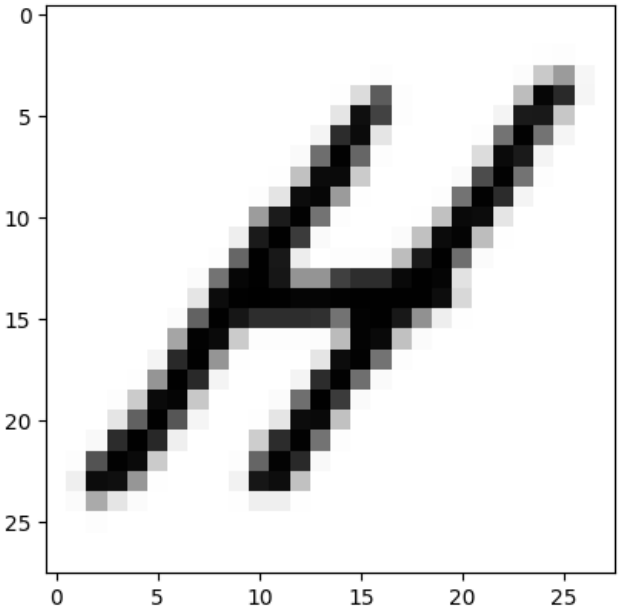
1.1 The dataset

The dataset we used in this preliminary stage is a quite commmon one. It is the eMNIST training and test set, which is based on the MNIST set but extended with letters. All the data is presented in the same format as the MNIST dataset, which means that each image is 28x28 pixels big, in grayscale and containst one single handwritten character. We attach below some examples of the images included in this set.




0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25



1.2 Feed-forward neural network

The most simple approach to this problem is by building a linear, feed-forward neural network. Being a first try, we didn't expect much from this experiment, but the network turned out to perform quite good. We managed to reach 80.1% accuracy on the eMNIST test set with a 6-layers feed-forward neural network. The code follows.

[ ]:
```
# Import necessary libraries
import numpy as np
import pickle

# Function that returns the value of the sigmoid function for a certain input␣
 ↪value
def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

# Function that returns the value of the derivative of the sigmoid function for␣
 ↪a certain input value
def sigmoid_prime(x):
    return sigmoid(x) * (1 - sigmoid(x))

# Mnist class

class Mnist(object):

    # Function that loads the mnist training and test data from csv files and␣
 ↪prepares two arrays to be pickled
```

```
    @staticmethod
    def _load_(source_tr, source_te, destination):
        # Array of couples (image, result) where 'image' is an array of 784␣
↪floats between 0 and 1, and 'result' is an array of 36 bits (either 0 or 1)␣
↪representing the neural netork output
        mnist_training_data = []
        # Array of couples (image, result) where 'image' is the same as above,␣
↪and 'result' is an integer between 0 and 35 representing the neural network␣
↪output
        mnist_test_data = []
        # Training set array
        with open(source_tr, "r") as f_tr:
            lines_tr = f_tr.readlines()
        for line_tr in lines_tr:
            line_tr = line_tr.replace("\n", "")
            bits_tr = line_tr.split(",")
            result_tr = np.array([[float(0 if i != int(bits_tr[0]) else 1)] for␣
↪i in range(36)])
            mnist_training_data.append((np.array([[float(bit_tr) / 255] for␣
↪bit_tr in bits_tr[1:]], dtype = "float32"), result_tr))
        # Test set array
        with open(source_te, "r") as f_te:
            lines_te = f_te.readlines()
        for line_te in lines_te:
            line_te = line_te.replace("\n", "")
            bits_te = line_te.split(",")
            mnist_test_data.append((np.array([[float(bit_te) / 255] for bit_te␣
↪in bits_te[1:]], dtype = "float32"), int(bits_te[0])))
        return (mnist_training_data, mnist_test_data)

    # Function that pickles the arrays into a binary file
    @staticmethod
    def pickle(source_tr, source_te, destination):
        mnist_training_data, mnist_test_data = Mnist._load_(source_tr,␣
↪source_te, destination)
        with open(destination, "wb") as f:
            pickle.dump((mnist_training_data, mnist_test_data), f, pickle.
↪HIGHEST_PROTOCOL)

    # Function that, given a binary file, unpickles the content, returning the␣
↪arrays
    @staticmethod
    def unpickle(source):
        with open(source, "rb") as f:
            mnist_training_data, mnist_test_data = pickle.load(f)
        return (mnist_training_data, mnist_test_data)
```

```
# Ann class

class Ann(object):

    # Function that pickles a neural network class instance into a binary file
    @staticmethod
    def pickle(ann, destination):
        with open(destination, "wb") as f:
            pickle.dump(ann, f, pickle.HIGHEST_PROTOCOL)

    # Function that unpickles the given ANN pickled file
    @staticmethod
    def unpickle(source):
        with open(source, "rb") as f:
            ann = pickle.load(f)
        return ann

# NeuralNetwork class

class NeuralNetwork(object):

    # Constructor - initializes a network with the given dimensions (dimens␣
 ↪must be a list of the neurons contained in each layer)

    def __init__(self, dimens):
        self.num_layers = len(dimens)
        self.dimens = dimens
        self.weights = [np.random.randn(y, x) for x, y in zip(self.dimens[:-1],␣
 ↪self.dimens[1:])]
        self.biases = [np.random.randn(y, 1) for y in self.dimens[1:]]

    # Function that, given an input "a" for the network, returns the␣
 ↪corresponding output

    def feedforward(self, a):
        for w, b in zip(self.weights, self.biases):
            a = sigmoid(np.dot(w, a) + b)
        return a

    # Function that, for every epoch, shuffles the training data and creates␣
 ↪mini batches (according to the stochastic gradient descent algoritm) with␣
 ↪the given size, then it calls the "update_mini_batch" function. If␣
 ↪"test_data" is given, a test against it is done at the end of each epoch of␣
 ↪training
```

```
    def sgd(self, training_data, epochs, mini_batch_size, eta, test_data =␣
↪None):
        if test_data: n_test = len(test_data)
        n = len(training_data)
        for e in range(epochs):
            np.random.shuffle(training_data)
            mini_batches = [training_data[k:k+mini_batch_size] for k in␣
↪range(0, n, mini_batch_size)]
            for mini_batch in mini_batches:
                self.update_mini_batch(mini_batch, eta)
            if test_data:
                print("Epoch %s: %s / %s" % (e, self.evaluate(test_data),␣
↪n_test))
            else:
                print("Epoch %s complete" % (e))
            eta *= 0.95

    # Function that updates weights and biases basing on the nabla_w and␣
↪nabla_b vertors, which are computed by the "backprop" function

    def update_mini_batch(self, mini_batch, eta):
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        for x, y in mini_batch:
            delta_nabla_w, delta_nabla_b = self.backprop(x, y)
            nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
            nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        self.weights = [w - (eta / len(mini_batch)) * nw for w, nw in zip(self.
↪weights, nabla_w)]
        self.biases = [b - (eta / len(mini_batch)) * nb for b, nb in zip(self.
↪biases, nabla_b)]

    # Function that calculates the gradient of the cost function (with respect␣
↪to weights and biases) by using the backpropagation algorithm

    def backprop(self, x, y):
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        # Feedforward
        activation = x
        activations = [x]
        zs = []
        for w, b in zip(self.weights, self.biases):
            z = np.dot(w, activation) + b
            zs.append(z)
            activation = sigmoid(z)
```

```
            activations.append(activation)
        # Backward pass
        delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1])
        nabla_w[-1] = np.dot(delta, activations[-2].transpose())
        nabla_b[-1] = delta
        for l in range(2, self.num_layers):
            z = zs[-l]
            sp = sigmoid_prime(z)
            delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
            nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
            nabla_b = delta
        return (nabla_w, nabla_b)

    # Function that returns the derivative of the cost function, given the␣
 ↪output of the net and the expected result

    def cost_derivative(self, output_activations, y):
        return (output_activations - y)

    # Function that evaluates the neural network precision against a test␣
 ↪dataset

    def evaluate(self, test_data):
        test_results = [(np.argmax(self.feedforward(x)), y) for (x, y) in␣
 ↪test_data]
        return sum(int(x == y) for (x, y) in test_results)

if __name__ == "__main__":
    training_set, test_set = Mnist.unpickle("res/mnist/emnist-balanced.pkl")
    network = NeuralNetwork([784, 100, 80, 60, 40, 36])
    network.sgd(training_set, 50, 10, 1.0, test_set)
    Ann.pickle(network, "res/network/ann_4h100806040_%s.pkl" % (network.
 ↪evaluate(test_set)*100/14400))
```

1.3 Convolutional neural network

We also tried using a CNN to try getting better results, but actually there was quite little improvement.

[5]:
```
# Import necessary libraries
import os.path
import pandas as pd
import keras
from keras.preprocessing.image import *
from keras.models import *
from keras.layers import *
from keras.utils import *
```

```
from keras.optimizers import *
from keras.callbacks import *
import numpy as np
import pandas as pd
from skimage.io import imread
import os
import matplotlib.pyplot as plt
import gc; gc.enable()
from cv2 import *

# Test values
showImages = False
baseDir = ""
mnistDir = os.path.join(baseDir, "mnist")

toId = {}
toChar = {}

with open(os.path.join(mnistDir, "emnist-balanced-mapping-to-char.txt")) as f:
    for line in f.read().split("\n"):
        a,b = line.split(" ")
        toChar[int(a)] = b
        toId[b] = a

numClasses = len(toChar)

# print(numClasses, "classes")
# print("toChar =", toChar)
# print("toId =", toId)

train = pd.read_csv(os.path.join(mnistDir, "emnist-balanced-train-uppercase.
 ↪csv"), header=None).to_numpy()

x = np.transpose(train[:, 1:].reshape((-1, 28, 28, 1)), (0, 2, 1, 3))
y = to_categorical(train[:, 0], num_classes=numClasses)

x = (x - x.mean()) / x.std()

# print(x.shape)
# print(y.shape)

generator = ImageDataGenerator(
    rotation_range=15,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False,
```

```
    vertical_flip=False
)

generator.fit(x)

if showImages:
    for i in range(100):
        d, l = x[i], y[i]
        #d = d.reshape((28, 28))
        # print(toChar[np.argmax(l)])
        #print(d)
        plt.title("" + str(toChar[np.argmax(l)]) + " - " + str(l))
        plt.imshow(d[:, :, 0], cmap="Greys")
        plt.show()

lrDecay = LearningRateScheduler(lambda epoch: 0.001 * np.power(0.95, epoch))
saver = ModelCheckpoint("model1.model")

model = Sequential()

model.add(InputLayer(input_shape=(28, 28, 1)))

model.add(Conv2D(32, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = 5, strides=2, padding='same',␣
 ↪activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(64, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size = 5, strides=2, padding='same',␣
 ↪activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(128, kernel_size = 4, activation='relu'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(numClasses, activation='softmax'))
```

```
model.compile(optimizer=Adam(0.001), loss="categorical_crossentropy",␣
 ↪metrics=["acc"])

model.summary()

stepsPerEpoch = 128
model.fit_generator(generator.flow(x, y, batch_size=stepsPerEpoch),
                    steps_per_epoch=len(x) // stepsPerEpoch,
                    epochs=50, callbacks=[lrDecay, saver])
```

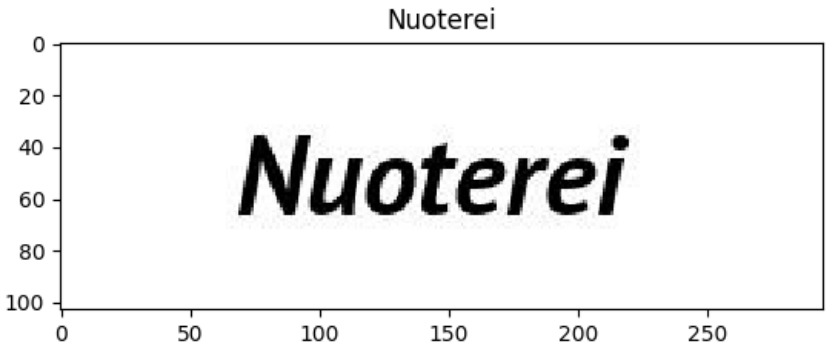
2 Recognizing strings

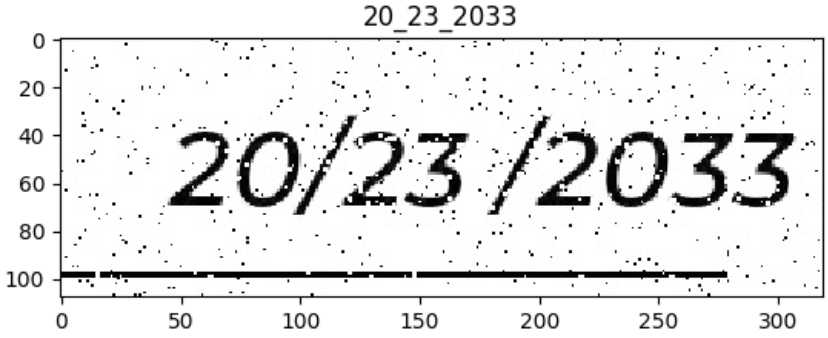
In this second part we concentrate on the task of reading some text (which is composed by more that one single character) from an image. We will assume that all the images contain texts written in standard fonts (there are no handwritten characters).

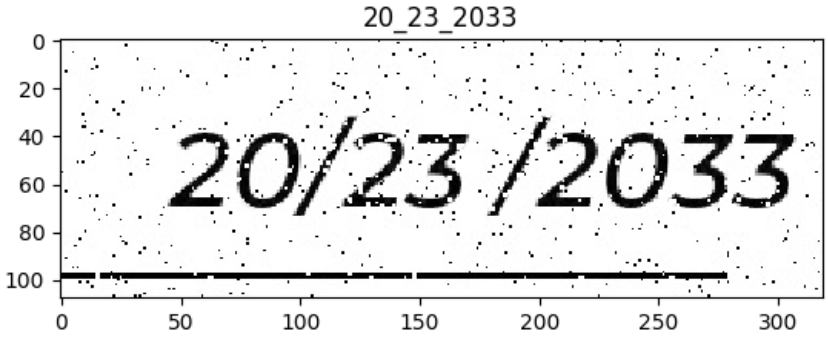
In order to achieve this result, we divided this complex problem into three subtasks: - try to cut the entire image into the single letters and characters; - run a neural network on the single characters and recognize them; - merge the two results and return the complete text contained in the original image.

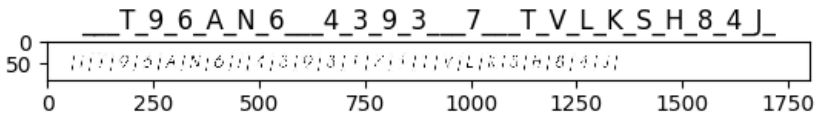
2.1 The dataset

As already said, the dataset is composed of images of texts written in standar fonts. They include dates, words from a dictionary, and even names or IBANs. Below there are some examples.




20_23_2033
20/23/2033
0
20
40
60
80
100
0
50
100
150
200
250
300


20_23_2033
20/23/2033
0
20
40
60
80
100
0
50
100
150
200
250
300



We can note that many of the images may contain random lines and even some noise.

2.2 Splitting the image into the single characters

This is actually the hardest part of the problem, as there is no algorithm that is guaranteed to work in all the situations that may occur. So, we had to formulate different euristics and try to do the best to separate the single characters.

First of all, we focused on the idea that, when a text is clearly written, a single character is made of single pixels which are all contiguous (apart from the lowercase 'i', that can be recognised even if considered without the little dot on the top). We developed this idea and made slight adjustments looking at the pixels next to each border of the recognised characters, merging contiguous boxes which looked like one single character, and discarding boxes which have incompatible dimensions (in order to ignore dots and random lines).

Follows a snippet of the code we used.

[4]:
```
# Import necessary libraries
import os.path
import queue
import numpy as np
import PIL
from PIL import Image

# Test values (the first of the images above)
DIR = "imgs"
IMAGE = "ACJOWW40Z13U616M.jpg"

# Function that checks if a given pixel is black or not
```

```
def isBlack(x):
    return x < 100

# Function that processes an image, splitting it in the single characters
def process(dir="", name="img.jpg"):
    img = Image.open(os.path.join(dir, name))
    name = name.split(".")[0]
    img = img.convert("L")
    img = np.array(img)
    img = img * 255. / img.max()
    img[img > 220] = 255
    img[img < 255] = 0
    img = PIL.Image.fromarray(np.uint8(img))
    w, h = img.size
    img = np.array(img)
    mask = np.zeros(img.shape)
    boxes = []

    for j in range(1, w-1):
        for i in range(1, h-1):
            s = True
            for a in range(-1, 2):
                for b in range(-1, 2):
                    if (a != 0 or b != 0) and img[i+a][j+b] == 0:
                        s = False
            if s:
                mask[i][j] = 1
                img[i][j] = 255

    for j in range(w):
        for i in range(h):
            if isBlack(img[i][j]) and mask[i][j] == 0:
                q = queue.Queue(maxsize=w*h)
                q.put((i, j))
                xmin, xmax, ymin, ymax = i, i, j, j
                while not q.empty():
                    x, y = q.get()
                    for a in range(-2, 2+1):
                        for b in range(-1, 1+1):
                            if x+a >= 0 and x+a < h and y+b >= 0 and y+b < w␣
 ↪and mask[x+a][y+b] == 0 and isBlack(img[x+a][y+b]):
                                mask[x+a][y+b] = 1
                                q.put((x+a, y+b))
                                xmin = min(xmin, x+a)
                                xmax = max(xmax, x+a)
                                ymin = min(ymin, y+b)
                                ymax = max(ymax, y+b)
```

```
                dex = xmax - xmin
                dey = ymax - ymin
                if dex > 5 and dey > 5 and 1.3 * dex > dey:
                    boxes.append((xmin, xmax, ymin, ymax))

    n = len(boxes)
    imgBoxes = np.copy(img)
    # print(n)
    for box in boxes:
        xmin, xmax, ymin, ymax = box
        for k in range(xmin, xmax+1):
            imgBoxes[k][ymin] = 192
            imgBoxes[k][ymax] = 192
        for k in range(ymin, ymax+1):
            imgBoxes[xmin][k] = 192
            imgBoxes[xmax][k] = 192
    # PIL.Image.fromarray(np.uint8(imgBoxes)).show()
    img = PIL.Image.fromarray(np.uint8(img))

    if len(boxes) == len(name):
        i = 0
        for box in boxes:
            xmin, xmax, ymin, ymax = box
            s = img.crop((ymin-2, xmin-2, ymax+2, xmax+2))
            s.save("out/%s_%s.jpg" % (name, i))
            i += 1

if __name__ == "__main__":
    process(DIR, IMAGE)
```

This piece of code processes an image and draws grey boxes around each character it recognizes, then saves a new image for each of them. We run this small script over all the dataset we were given in order to generate a suitable training dataset for a neural network which is able to recognize the single characters.

2.3 Recognizing the single characters

After having a set of single characters, we were able to train a neural network to recognize them. Actually, we used a network which is pretty similar to the CNN we used to recognize the eMNIST set of characters.

[2]:
```
# Import necessary libraries
import os.path
import pandas as pd
import PIL
from PIL import Image
import keras
from keras.preprocessing.image import *
from keras.models import *
```

```
from keras.layers import *
from keras.utils import *
from keras.optimizers import *
from keras.callbacks import *
import numpy as np
import pandas as pd
from skimage.io import imread
import os
import matplotlib.pyplot as plt
import gc; gc.enable()
from cv2 import *

# Test data
showImages = False
baseDir = ""
wordsDir = os.path.join(baseDir, "words")

#mappings
toId = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8,␣
 ↪'j': 9, 'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r':␣
 ↪17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25,␣
 ↪'A': 26, 'B': 27, 'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33, 'I':␣
 ↪34, 'J': 35, 'K': 36, 'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42,␣
 ↪'R': 43, 'S': 44, 'T': 45, 'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z':␣
 ↪51, '0': 52, '1': 53, '2': 54, '3': 55, '4': 56, '5': 57, '6': 58, '7': 59,␣
 ↪'8': 60, '9': 61, '_': 62}
toChar = {0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e', 5: 'f', 6: 'g', 7: 'h', 8:␣
 ↪'i', 9: 'j', 10: 'k', 11: 'l', 12: 'm', 13: 'n', 14: 'o', 15: 'p', 16: 'q',␣
 ↪17: 'r', 18: 's', 19: 't', 20: 'u', 21: 'v', 22: 'w', 23: 'x', 24: 'y', 25:␣
 ↪'z', 26: 'A', 27: 'B', 28: 'C', 29: 'D', 30: 'E', 31: 'F', 32: 'G', 33: 'H',␣
 ↪34: 'I', 35: 'J', 36: 'K', 37: 'L', 38: 'M', 39: 'N', 40: 'O', 41: 'P', 42:␣
 ↪'Q', 43: 'R', 44: 'S', 45: 'T', 46: 'U', 47: 'V', 48: 'W', 49: 'X', 50: 'Y',␣
 ↪51: 'Z', 52: '0', 53: '1', 54: '2', 55: '3', 56: '4', 57: '5', 58: '6', 59:␣
 ↪'7', 60: '8', 61: '9', 62: '_'}

numClasses = len(toChar)

# print(numClasses, "classes")
# print("toChar =", toChar)
# print("toId =", toId)

#read dataset
trainData = pd.read_csv("dataset.csv", header=None)

#code provided by Corrado Alessio (HumanProtein challlenge)
# https://www.dropbox.com/sh/hsgzowq3elk7n68/AABM_cJ_r3zxkljYNRv8YPRra?dl=0)
#wraps input in a keras sequence
```

```
class MyDataGenerator(keras.utils.Sequence):

    def __init__(self, paths, labels, batch_size, shape, shuffle=False,␣
↪use_cache=False, augment=None, directory="all"):
        self.augment = augment
        self.directory = directory
        self.paths, self.label = paths, to_categorical(labels)
        self.batch_size = batch_size
        self.shape = shape
        self.shuffle = shuffle
        self.use_cache = use_cache
        if use_cache == True:
            self.cache = np.zeros((paths.shape[0], shape[0], shape[1],␣
↪shape[2]))
            self.is_cached = np.zeros((paths.shape[0]))
        self.on_epoch_end()

    def __len__(self):
        return int(np.ceil(len(self.paths) / float(self.batch_size)))

    def __getitem__(self, idx):
        indexes = self.indexes[idx * self.batch_size: (idx + 1) * self.
↪batch_size]

        paths = self.paths[indexes]
        X = np.zeros((paths.shape[0], self.shape[0], self.shape[1], self.
↪shape[2]))
        # Generate data
        if self.use_cache == True:
            X = self.cache[indexes]
            for i, path in enumerate(paths[np.where(self.is_cached[indexes] ==␣
↪0)]):
                image = self.__load_image(path)
                self.is_cached[indexes[i]] = 1
                self.cache[indexes[i]] = image
                X[i] = image
        else:
            for i, path in enumerate(paths):
                X[i] = self.__load_image(path)

        y = np.stack(self.labels[indexes])

        return X, y

    def on_epoch_end(self):

        # Updates indexes after each epoch
```

```
        self.indexes = np.arange(len(self.paths))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)

    def __iter__(self):
        """Create a generator that iterate over the Sequence."""
        for item in (self[i] for i in range(len(self))):
            yield item

    def __load_image(self, path):
        return augment.random_transform(np.array(Image.open(path).convert("L").
 ↪resize((self.shape[0], self.shape[1]), PIL.Image.ANTIALIAS))\
            .reshape(self.shape))

augment = ImageDataGenerator(
    rotation_range=15,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False,
    vertical_flip=False
)

gen = MyDataGenerator(trainData[0], trainData[1], 32, (28, 28, 1),␣
 ↪augment=augment)

#annealing and model saving
lrDecay = LearningRateScheduler(lambda epoch: 0.001 * np.power(0.95, epoch))
saver = ModelCheckpoint("train_2.model")

model = Sequential()
#automatic input normalization
model.add(BatchNormalization(input_shape=(28, 28, 1)))

model.add(Conv2D(32, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = 5, strides=2, padding='same',␣
 ↪activation='relu')) #differentiable pooling
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(64, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
```

```
model.add(Conv2D(64, kernel_size = 5, strides=2, padding='same',␣
 ↪activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(128, kernel_size = 4, activation='relu'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(numClasses, activation='softmax'))  #one hot result

model.compile(optimizer=Adam(0.001), loss="categorical_crossentropy",␣
 ↪metrics=["acc"])  #standard optimizer with annealing

model.summary()

model.fit_generator(gen, epochs=50, callbacks=[lrDecay, saver])
```

The neural network trained with this data was able to reach 91.0% accuracy in our tests.

2.4 String recognition

Given an image containing some text, the program executes three main steps: - tries to split it in images containing single characters; - processes all these images with the trained neural network, recognizing the characters; - combines all the characters in a string, which is then returned.

The software, actually, is just a combination of the two parts already described in sections 2.2 and 2.3. We attach here the complete code.

[3]:
```
# Import necessary libraries
import numpy as np
import PIL
from PIL import Image
import queue
import os.path
import keras
from keras.preprocessing.image import *
from keras.models import *
from keras.layers import *
from keras.utils import *
from keras.optimizers import *
from keras.callbacks import *
import os
import matplotlib.pyplot as plt
```

```
toId = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8,␣
 ↪'j': 9, 'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r':␣
 ↪17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25,␣
 ↪'A': 26, 'B': 27, 'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33, 'I':␣
 ↪34, 'J': 35, 'K': 36, 'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42,␣
 ↪'R': 43, 'S': 44, 'T': 45, 'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z':␣
 ↪51, '0': 52, '1': 53, '2': 54, '3': 55, '4': 56, '5': 57, '6': 58, '7': 59,␣
 ↪'8': 60, '9': 61, '_': 62}
toChar = {0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e', 5: 'f', 6: 'g', 7: 'h', 8:␣
 ↪'i', 9: 'j', 10: 'k', 11: 'l', 12: 'm', 13: 'n', 14: 'o', 15: 'p', 16: 'q',␣
 ↪17: 'r', 18: 's', 19: 't', 20: 'u', 21: 'v', 22: 'w', 23: 'x', 24: 'y', 25:␣
 ↪'z', 26: 'A', 27: 'B', 28: 'C', 29: 'D', 30: 'E', 31: 'F', 32: 'G', 33: 'H',␣
 ↪34: 'I', 35: 'J', 36: 'K', 37: 'L', 38: 'M', 39: 'N', 40: 'O', 41: 'P', 42:␣
 ↪'Q', 43: 'R', 44: 'S', 45: 'T', 46: 'U', 47: 'V', 48: 'W', 49: 'X', 50: 'Y',␣
 ↪51: 'Z', 52: '0', 53: '1', 54: '2', 55: '3', 56: '4', 57: '5', 58: '6', 59:␣
 ↪'7', 60: '8', 61: '9', 62: '_'}

def isBlack(x):
    return x < 100

def process(dir="", name="img.jpg"):
    img = Image.open(os.path.join(dir, name))
    name = name.split(".")[0]
    img = img.convert("L")
    img = np.array(img)
    img = img * 255. / img.max()
    img[img > 220] = 255
    img[img < 255] = 0
    #img = np.array([list(map(lambda x: 0 if x < 220 else 255, img_row)) for␣
 ↪img_row in img])
    img = PIL.Image.fromarray(np.uint8(img))
    w, h = img.size
    img = np.array(img)
    mask = np.zeros(img.shape)
    boxes = []

    for j in range(1, w-1):
        for i in range(1, h-1):
            s = True
            for a in range(-1, 2):
                for b in range(-1, 2):
                    if (a != 0 or b != 0) and img[i+a][j+b] == 0:
                        s = False
            if s:
                mask[i][j] = 1
                img[i][j] = 255
```

```
    for j in range(w):
        for i in range(h):
            if isBlack(img[i][j]) and mask[i][j] == 0:
                q = queue.Queue(maxsize=w*h)
                q.put((i, j))
                xmin, xmax, ymin, ymax = i, i, j, j
                while not q.empty():
                    x, y = q.get()
                    for a in range(-2, 2+1):
                        for b in range(-1, 1+1):
                            if x+a >= 0 and x+a < h and y+b >= 0 and y+b < w␣
↪and mask[x+a][y+b] == 0 and isBlack(img[x+a][y+b]):
                                mask[x+a][y+b] = 1
                                q.put((x+a, y+b))
                                xmin = min(xmin, x+a)
                                xmax = max(xmax, x+a)
                                ymin = min(ymin, y+b)
                                ymax = max(ymax, y+b)
                dex = xmax - xmin
                dey = ymax - ymin
                if dex > 5 and dey > 5 and 1.3 * dex > dey:
                    boxes.append((xmin, xmax, ymin, ymax))

    n = len(boxes)
    """l = 0
    for box in boxes:
        xmin, xmax, ymin, ymax = box
        l += xmax
    l /= n"""

    imgBoxes = np.copy(img)

    # print(n)
    for box in boxes:
        xmin, xmax, ymin, ymax = box
        for k in range(xmin, xmax+1):
            imgBoxes[k][ymin] = 192
            imgBoxes[k][ymax] = 192
        for k in range(ymin, ymax+1):
            imgBoxes[xmin][k] = 192
            imgBoxes[xmax][k] = 192

    #PIL.Image.fromarray(np.uint8(imgBoxes)).show()
    img = PIL.Image.fromarray(np.uint8(img))

    res = []
```

```
    for box in boxes:
        xmin, xmax, ymin, ymax = box
        res.append(np.array(img.crop((ymin-2, xmin-2, ymax+2, xmax+2)).
↪resize((28,28))).reshape((28,28,1)))
        #s.save("out/%s_%s.jpg" % (name, i))

    return np.array(res)

'''
for d in data:
    plt.imshow(d)
    plt.show()
'''

model = load_model("train_1_1.model")
model.summary()

for dir in ["words/all"]:  #predict all images
    for name in os.listdir(dir):
        data = process("words/all", name)

        res = model.predict(data, verbose=1)

        res = np.argmax(res, axis=1)

        pred = "".join([toChar[x] for x in res])

        #show results (one at a time)
        plt.imshow(Image.open(os.path.join("words/all", name)))
        plt.title(pred)
        plt.show()
```