To Recognize Capitalized Hand-written Alphabets

Author: CHI YUFENG, JI RUI, JIANG BOXUAN, WANG BOQIAN

Instructor: Mrs. Neera Tyagi

I. Abstract

Our project's aim is to program the computer to identify hand-written alphabets via matrix operations. Each alphabet image contains 28*28 pixels, and we create a matrix using these pixels. By multiply the matrix to several sample matrixes, the pixels are converted into a three-layer 1024x512x128 fully-connected deep neural network. And finally, we employ gradient decent method so that the computer can predict the highest possibility of the alphabet written.

II. Introduction

Have you ever faced a problem, where you wanted to transform a paper document into a digital one, but you have to type it character by character? People have been trying to find a easy way to solve the problem, and we have offered a solution. The purpose of our project is to recognize hand-written alphabets, so the computer can automatically identify the characters without any manual input.

III. Progress

1. Getting Started

To begin with, we need Python 3.6 installed. Then, we need to import several libraries that will be used.

```
import tflearn
import keras
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

*Note: if you are using a brand new Python, you need to first install these libraries using pip install:

```
> pip install numpy pandas tensorflow tflearn keras matplotlib
```

2. Gather Dataset

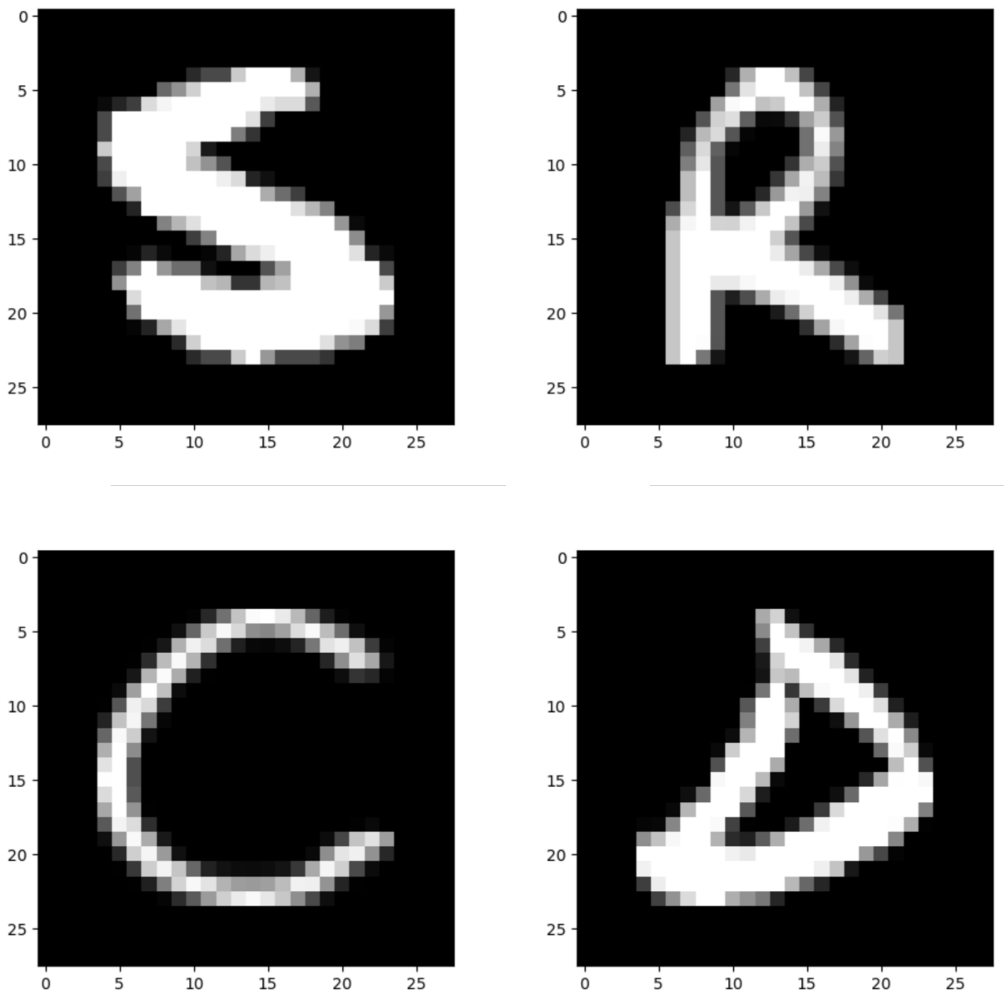
The A_Z Handwritten Data contains capitalized handwritten alphabet images (A-Z) in size of 28x28 pixels. Each alphabet in the image is centered at 20x20 pixel box. There are 372451 images in total, or approximately 14325 images for each of the alphabet, in the data file.

As it is a csv file after decompression, we can use the loadtxt() method from numpy to load from the dataset.

```
train_file = 'data/a_z_handwritten.csv'
num_classes = 26

np = np.loadtxt(train_file, delimiter=',')
```

The images looks like this:



3. Preprocess Data

In order to achieve maximum efficient, we need to preprocess the data before we feed it into the neural network.

```
def preprocess_data(raw):
    np.random.shuffle(raw)
    y_label = raw[:, 0]
    out_y = keras.utils.to_categorical(y_label, num_classes)
    num_images = raw.shape[0]
    out_x = raw[:, 1:785]
    return out_x, out_y

X, Y = load_data()

testX, testY = X[-1000:], Y[-1000:]
X, Y = X[:80000], Y[:80000]
```

In the code above, we first use the shuffle() method to randomize the data (the original one is sorted from A-Z, and it is not machine-friendly). Then, we extract the label, the number indicating which letter is this row representing, from the raw data. keras.utils.to_categorical() is used to convert the number, range from 0-25, into the one-hot array.

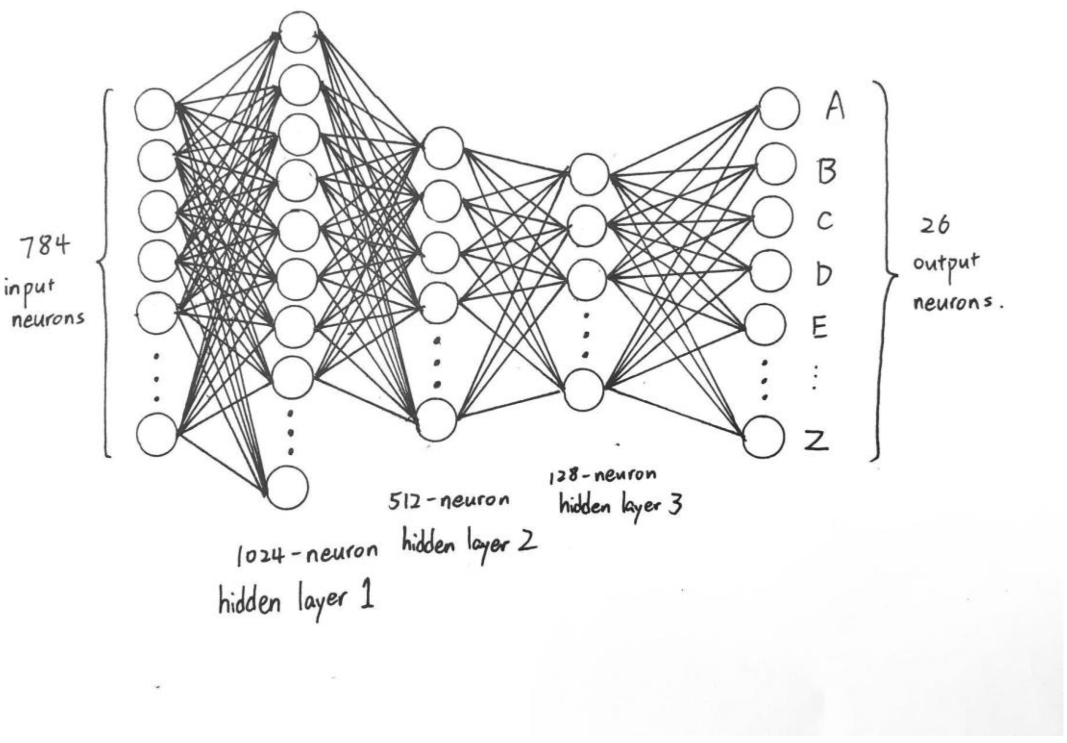
The image is extracted from the raw data, and the function returns the processed X and Y.Finally, we use the last 1000 images of the dataset as the test set, and the first 80000 rows (this is limited by my 16G RAM) as the train set.

4. Building the Network

```
def build_model():
    net = tflearn.input_data(shape=[None, 784])       # input_layer
    net = tflearn.fully_connected(net, 1024, activation='relu', regularize
r='L2', weight_decay=0.001)                  # dense1
    net = tflearn.dropout(net, 0.9)                   # dropout1
    net = tflearn.fully_connected(net, 512, activation='relu', regularizer
='L2', weight_decay=0.001)                  # dense2
    net = tflearn.dropout(net, 0.9)                   # dropout2
    net = tflearn.fully_connected(net, 128, activation='relu', regularizer
='L2', weight_decay=0.001)                  # dense3
    net = tflearn.dropout(net, 0.9)                   # dropout3
    softmax = tflearn.fully_connected(net, 26, activation='softmax')
    # Regression using SGD with learning rate decay and Top-3 accuracy
    sgd = tflearn.SGD(learning_rate=0.04, lr_decay=0.98, decay_step=1000)
    net = tflearn.regression(softmax, optimizer=sgd, metric=tflearn.metric
s.Accuracy(), loss='categorical_crossentropy')
    model = tflearn.DNN(net, tensorboard_verbose=0)
    return model
```

This is a three-layer 1024x512x128 fully-connected deep neural network.

In the image below this what the neural network looks like, when visualized:



5. Train the model

```
model = build_model()
model.fit(X, Y, n_epoch=4, validation_set=(testX, testY), show_metric=True
)
```

This step is surprisingly simple.

First, we build the model. Then, we use the fit() method to train the network.

6. Evaluation

We will predict and show a random image in the test dataset to test the accuracy.

```
import random
preds = model.predict([testX[random.randint(0, len(testX))]])
pos = n_hi = 0
for i, n in enumerate(testY[2]):
 if n > n_hi:
  pos, n_hi = i, n
print(chr(pos+65))
pos = n_hi = 0
for i, n in enumerate(preds[0]):
 if n > n_hi:
  pos, n_hi = i, n
print(chr(pos+65))
plt.imshow(testX[2].reshape(28, 28), cmap=plt.cm.gray)
plt.pause(2)
plt.close()
```

IV. Result

```
Training Step: 12500  | total loss: [1m[32m0.15304[0m[0m | time: 61.852s
SGD | epoch: 004 | loss: 0.15304 - acc: 0.9735 | val_loss: 0.03337 - val_a
cc: 0.9910 -- iter: 200000/200000
```

We achieve an accuracy of 0.9910 in the validation dataset after 4 epochs of training, the images that the neural network has not seen before. This is actually a surprising outcome with such a simple network structure.

V. Reference

Patel S. (2018). A-Z Handwritten Alphabets in .csv format. *Kaggle*. Retrieved from https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format