Exploring handwritten digit recognition by applying different ML/DL techniques

Anonymous CVP submission

Paper ID ***

Abstract

Exploring handwritten digit recognition by applying different differnt ML/DL techniques for different datasets The handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavors. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.

1. Introduction

In this project we will try to recognize handwritten digits from various sources like the MNIST database, Gurmukhi language etc and will apply multiple ML and DL techniques and compare their results. The OCR techniques will also be explored and compared for the results. A detailed report, link to github with codes and a youtube video will be presented with the major results.

Authored by : Saumitra Tarey M20AIE299

1.1. Language

All manuscripts must be in English.

1.2. Using Neural Networks MNIST

A Neural Network is coded using PyTorch Library in Python.

For this, we will be using the popular MNIST database. It is a collection of 70000 handwritten digits split into training and test set of 60000 and 10000 images respectively. We will do the below transformations on the dataset.

transforms.ToTensor() — converts the image into numbers, that are understandable by the system. It separates the image into three color channels (separate images): red, green and blue. Then it converts the pixels of each image to the brightness of their color between 0 and 255. These values are then scaled down to a range between 0 and 1. The image is now a Torch Tensor.

transforms.Normalize() — normalizes the tensor with a mean and standard deviation which goes as the two parameters respectively. The Neural network is made by:

Input Layer having 784 neurons, dense layer having 128 neurons, dense layer having 64 neurons, output layer having 10 neurons with softmax, cross entropy loss

Using the MNIST dataset we train this model using the split trainset of 60000 images and test it on the test set

The results are as below. They are a combination of varying the hyperparameters of the model like learning rate, loss and number of layers

Case 1 - learning rate = 0.003, layers = 2, loss = NLLLoss

time taken to train = 2m 42s accuracy achieved = .9707

==========

Case 2 - learning rate = 0.03, layers = 2, loss = NLLLoss

time taken to train = 2m 45s accuracy achieved = .9671

Hence increasing the learning rate in this case reduces the accuracy and increases the time taken to train

==========

Case 3 - learning rate = 0.3, layers = 2, loss = NLLLoss

time taken to train = 3m 06s accuracy achieved = .1028

Hence increasing the learning rate in this case reduces the accuracy and increases the time taken to train

==========

Case 4 - learning rate = 0.003, layers = 4, loss = NLLLoss

time taken to train = 3m 53s accuracy achieved = .9767

Hence increasing the number of layers from 2 to 4 improves the accuracy by a factor of 0.006 in this case

==========

Case 5 - learning rate = 0.003, layers = 4, loss = CrossEntropyLoss

time taken to train = 3m 53s accuracy achieved = .9799

Cross entropy gives better result then NLLLoss in this case while the training time remains the same

1.3. Using Neural Networks Gurmukhi

Gurmukhi is one of the popular Indian scripts widely used in Indian state of Punjab. In this part of the assignment,

our goal is to develop a neural network solution for classifying Gurmukhi digits. The code written for the MNIST dataset for the previous section of classification has been modified to adapt to Gurmukhi.

Input Layer having 1024 neurons, dense layer having 512 neurons, dense layer having 128 neurons, output layer having 10 neurons with softmax

Case 1 - learning rate = 0.003, layers = 2, loss = NLLLoss

time taken to train = 1m 31s accuracy achieved = .8632

==========

Case 2 - learning rate = 0.03, layers = 2, loss = NLLLoss

time taken to train = 2m 36s accuracy achieved = .7899

Hence increasing the learning rate in this case reduces the accuracy and increases the time taken to train

==========

Case 3 - learning rate = 0.3, layers = 2, loss = NLLLoss

time taken to train = 6m 13s accuracy achieved = .0909

Hence increasing the learning rate in this case reduces the accuracy and increases the time taken to train

==========

Case 4 - learning rate = 0.003, layers = 4, loss = NLLLoss

time taken to train = 3m 16s accuracy achieved = .8887

Hence increasing the number of layers from 2 to 4 improves the accuracy

==========

Case 5 - learning rate = 0.003, layers = 4, loss = CrossEntropyLoss

time taken to train = 3m 13s accuracy achieved = .8771

Cross entropy gives better result then NLLLoss in this case while the training time remains the same

1.4. OCR using Tensorflow

Optical character recognition or optical character reader is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo or from subtitle text superimposed on an image.

The datasets that we use are the standard MNIST 0-9 dataset by LECun et al and The Kaggle A-Z dataset by Sachin Patel. The Standard MNIST dataset is already builtin in many deep learning frameworks like tensorflow, Pytorch, keras. MNIST dataset allow us to recognize the digits 0–9. Each image containing single digits of 28 x 28 grayscale images.

MNIST doesn't include A-Z so for that we're using dataset released by Sachin Patel on kaggle. This dataset takes the capital letters A-Z from NIST Special Database 19 and rescales them to be 28 x 28 grayscale pixels to be in the same format as our MNIST data.

The process involves loading the datasets, combining the datasets, training the dataset using Resnet and then testing for accuracy.

Residual Network (ResNet) is a deep learning model used for computer vision applications. It is a Convolutional Neural Network (CNN) architecture designed to support hundreds or thousands of convolutional layers. The architecture has been built in the code from scratch.

The testing result of the above model approach is posted below

precision recall f1-score support

0 0.14 0.96 0.25 1381

1 0.97 0.98 0.97 1575

2 0.89 0.95 0.92 1398

3 0.93 1.00 0.96 1428

4 0.86 0.97 0.91 1365

5 0.48 0.94 0.64 1263

6 0.93 0.97 0.95 1375

7 0.93 0.99 0.96 1459

8 0.91 0.99 0.95 1365

9 0.94 0.99 0.97 1392

A 0.99 0.99 0.99 2774

B 0.98 0.97 0.98 1734

C 0.99 0.97 0.98 4682

D 0.94 0.95 0.95 2027

E 0.99 0.98 0.99 2288

F 0.96 1.00 0.98 232

G 0.94 0.94 0.94 1152

H 0.97 0.96 0.97 1444

I 0.97 0.98 0.98 224

J 0.97 0.96 0.97 1699

K 0.97 0.98 0.97 1121

L 0.97 0.98 0.98 2317

M 0.98 0.99 0.99 2467

N 0.99 0.98 0.99 3802

O 0.99 0.31 0.48 11565

P 1.00 0.98 0.99 3868

Q 0.94 0.97 0.95 1162

R 0.99 0.98 0.98 2313

S 0.99 0.85 0.92 9684

T 0.99 0.98 0.99 4499

U 0.99 0.97 0.98 5802

V 0.95 0.99 0.97 836

W 0.99 0.98 0.99 2157

X 0.98 0.99 0.98 1254

Y 0.97 0.92 0.95 2172

Z 0.95 0.91 0.93 1215

accuracy 0.87 88491

macro avg 0.93 0.95 0.92 88491

weighted avg 0.96 0.87 0.89 88491

Number of Epochs used here is 10, as we increase the number of epochs, the accuracy goes up. For e.g. the ac-

curacy for 50 epochs which took about 5 hours of training using GPU is around .96

1.5. OCR using EasyOCR

EasyOCR is actually a python package that holds PyTorch as a backend handler. EasyOCR like any other OCR(tesseract of Google or any other) detects the text from images but in my reference, while using it I found that it is the most straightforward way to detect text from images also when high end deep learning library(PyTorch) is supporting it in the backend which makes it accuracy more credible. EasyOCR supports 42+ languages for detection purposes. EasyOCR is created by the company named Jaided AI company.

We have used MNIST data set here to predict the digits using EasyOCR and test its accuracy. The output is provided in the form of a co-ordinates bounding box, predicted digit and the confidence level.

1.6. OCR using pytesseract

Pytesseract or Python-tesseract is an Optical Character Recognition (OCR) tool for Python. It will read and recognize the text in images, license plates etc. Python-tesseract is actually a wrapper class or a package for Google's Tesseract-OCR Engine. It is also useful and regarded as a stand-alone invocation script to tesseract, as it can easily read all image types supported by the Pillow and Leptonica imaging libraries.

Using the same test images used for EasyOCR sampled from the MNIST database, we loop though the images and predict the MNIST digit. The image is read and first converted to grayscale and processed further with thresholding and noise removal.

2. Summary

We have explored the topic of Handwritten digit recognition by applying various ML/DL techniques in this project. We have seen a basic neural nets based model to recognize digits for MNIST and the Gurmukhi language with reasonable accuracy.

Then we have seen an approach to use tensorflow to create a custom OCR model and training it using Resnet. Finally we have tested the readily available OCR like EasyOCR and Pytesseract.

2.1. Inference

Using the sequential model as mentioned in the section 1.2 we found that the ANN deep learning model gives reasonably good results on the MNIST and Gurumukhi datasets. The training time is in few minutes and the accuracy varies depending on the loss function, learning rates and varying the dense layers.

The second model is made using tensorflow and a Resnet model is used to train on the input data. Input is a combination of MNIST and AZ kaggle dataset, which makes it more versatile than the ANN model. The training time is around 20 minutes for 10 epochs and the model is able to predict the digits and alphabets with high accuracy.

For the readymade library EasyOCR, we have used a subset of MNIST database sample images for classification. To classify 300+ images, the code takes about 40 seconds which make is quite fast and easy to use, however the results are not as accurate as the previously custom made models. There are many images which the library is not able to classify.

Pytesseract does not work well with the MNIST dataset and is not able to classify the images at all.

References

https://pypi.org/project/easyocr/
https://pypi.org/project/pytesseract/
https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format