Training CNN on MNIST Dataset

The task is to build a program software for convolution neural network on a given dataset.

Problem Definition

Goal is to correctly identify digits from a dataset of tens of thousands of handwritten images.

Dataset

The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been **size-normalized and centered in a fixed-size image.**

This classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

Source

Data is taken from the Digit Recognizer: Learn computer vision fundamentals with the famous MNIST data contest from Kaggle

Detailed Explanation

The data files `train.csv` and `test.csv` contain gray-scale images of hand-drawn digits, from zero through nine.

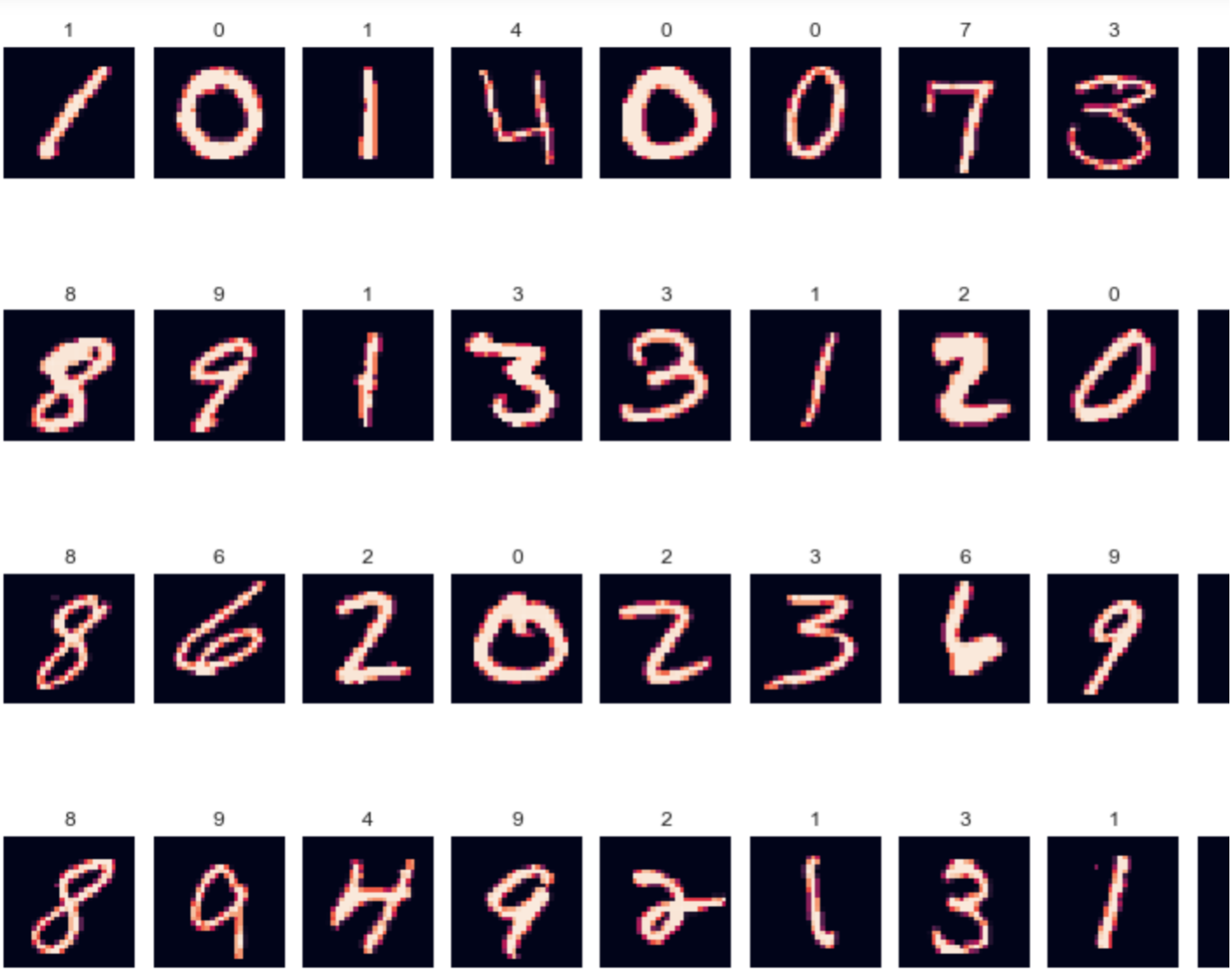
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

Each pixel column in the training set has a name like `pixelx`, where `x` is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed `x` as `x = i * 28 + j`, where `i` and `j` are integers between 0 and 27, inclusive. Then `pixelx` is located on row `i` and column `j` of a `28 x 28` matrix, (indexing by zero).

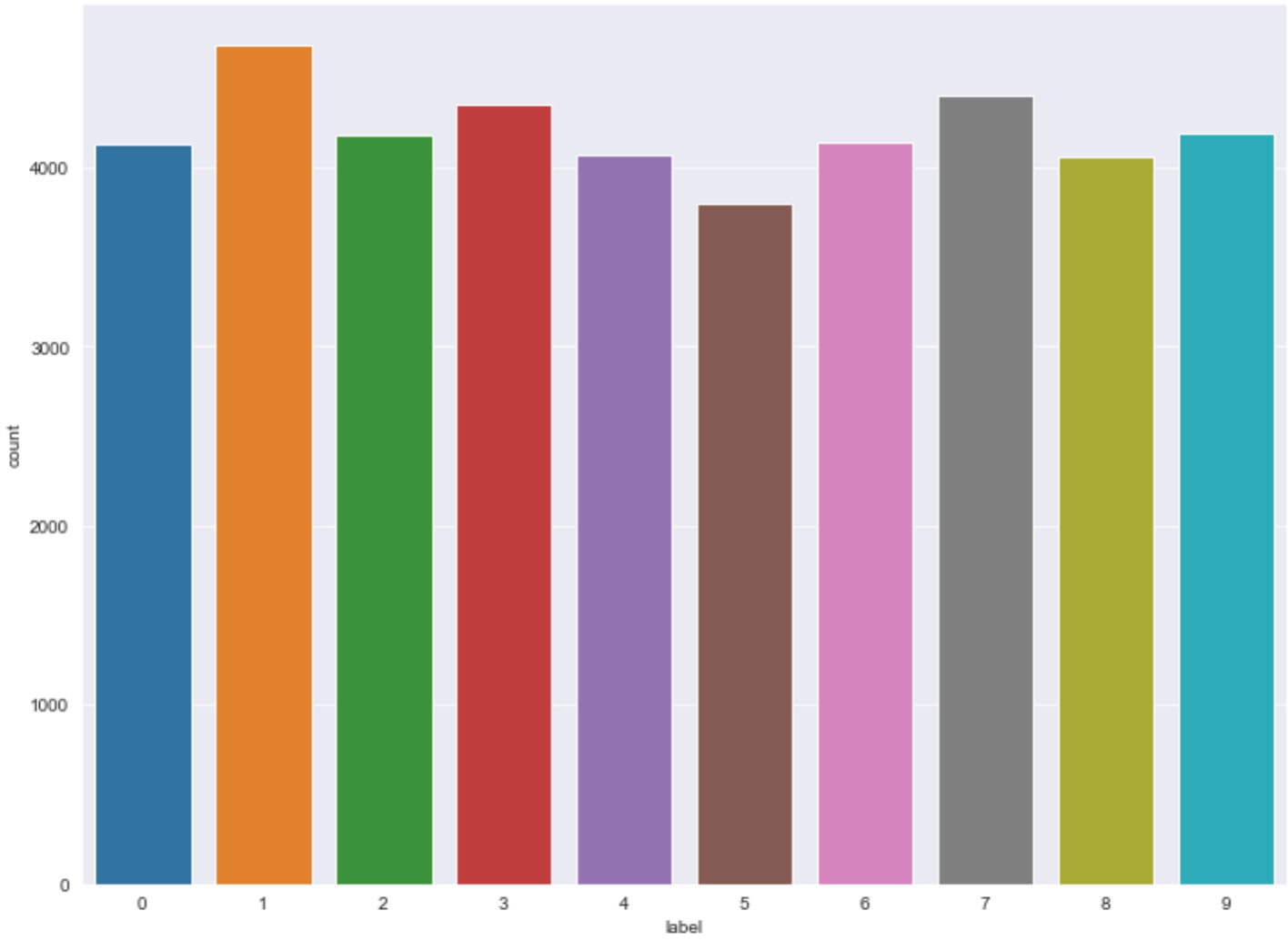
```
000 001 002 003 ... 026 027
028 029 030 031 ... 054 055
056 057 058 059 ... 082 083
 |   |   |   |  ...  |   |
728 729 730 731 ... 754 755
756 757 758 759 ... 782 783
```

Training Data

The training data set `(train.csv)`, has 785 columns. The first column, called **"label"**, is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.



Training Data is such that the number of samples for each class are uniform.



Testing Data

The test data set (`test.csv`), is the same as the training set, except that it does not contain the "label" column.

CNN

Methodology

TensorFlow Sequential API

We use TensorFlow Sequential API to define our model. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

Convolution Layer

The first is the convolutional (`Conv2D`) layer. It is like a set of learnable filters. Here, I set 32 filters for the two firsts Conv2D layers and 64 filters for the last two Conv2D layers.

Each filter transforms a part of the image (defined by the *kernel size*) using the kernel filter. The kernel filter matrix is applied on the whole image. **Filters can be seen as a transformation of the image.**

This allows the model to isolate features that are useful from anywhere in the image.

MaxPooling Layer

The second important layer is the pooling (`MaxPool2D`) layer. This layer simply acts as a downsampling filter. It looks at the `n` neighboring pixels (defined by the pool_size) and picks the **maximum value**.

These are used to reduce computational cost and also indirectly reduce overfitting. **Combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.**

Dropout

The Dropout layer randomly sets input units to 0 with a frequency of `rate` at each step during training time, which **helps prevent overfitting**. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

> Dropout layer only applies when training is set to True such that no values are dropped during inference.

ReLU

`relu` is an activation function (mathematically - `max(0, x)`). The rectifier activation function is **used to add non linearity to the network.**

Flatten

The Flatten layer is use to convert the final feature maps into a one single 1D vector. This flattening step is needed so that you can make use of fully connected layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers.

Final Classifier

Finally, we add two fully-connected (`Dense`) layers which is just an artificial neural networks (ANN) classifier. The last layer (`Dense(10, activation="softmax")`) uses the softmax activation function which **outputs distribution of probability of each class.**

Evaluation

The evaluation metric used is the categorization accuracy, or the proportion of test images that are correctly classified. For example, a categorization accuracy of 0.97 indicates that you have correctly classified all but 3% of the images.

Mathematically, it can be expressed as -

```
Accuracy = Number of Correct Predictions / Total number of Predictions
```

Read more about it here

Loss Function

Loss function measures how poorly a model performs on images with known labels. **It is the error rate between the oberved labels and the predicted ones.**. **CategoricalCrossentropy** computes the crossentropy loss between the labels and predictions.

This loss function when there are two or more label classes. It expects labels to be provided in a **one_hot** representation.

Read more about it here

Optimizer

Optimizer iteratively improves parameters (filter kernel values, weights and bias of neurons, etc) in order to minimise the loss.

We use Adam as an optimizer, as it works well for most tasks out of the box & does not need much tuning.

Read more about it here

Libraries Used

Tensorflow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 in 2015

> TensorFlow version used - 2.3

Pandas

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

> Pandas version used - 1.1.15

Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

> Numpy version used -

Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

> Matplotlib version used - 3.4

Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

> Seaborn version used - 0.11.1

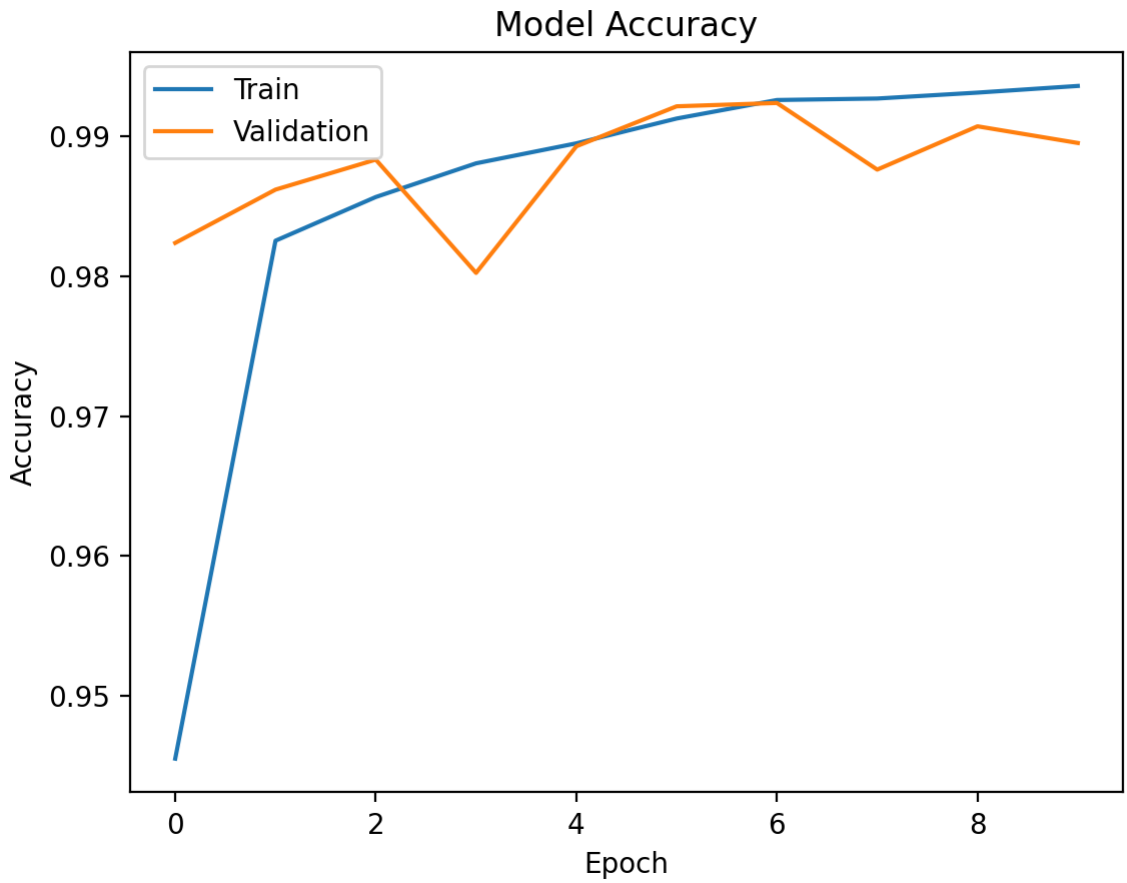
Results

Train Data Result

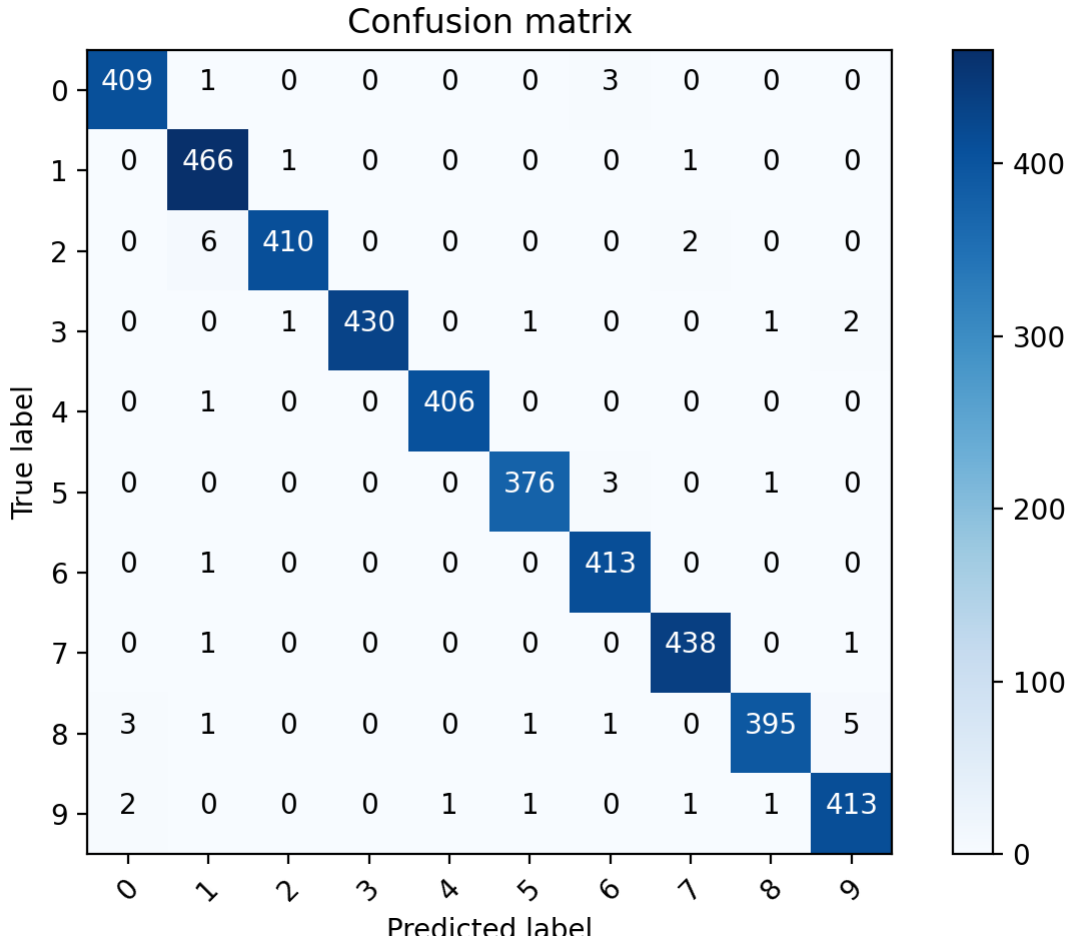
```
 Final Training Loss: 0.012
Final Training Accuracy: 0.995

Final Validation Loss: 0.040
Final Validation Accuracy: 0.989
```

Training & Validation Metric still seemed to be improving, that implies the model could be trained a bit longer.



Looking at the confusion matrix below, we could infer that the model performed very well in general. However, it had some difficultly in the following pairs of labels - 1. Digits 2 & 3 2. Digits 8 & 9 3. Digits 6 & 0



Test Data Result

Submitting the result on test data gives a score of `0.98978` on Kaggle. This gives a position of 1502 on the current leaderboard ranking.

Your most recent submission

Name	Submitted	Wait time	Execution
submission.csv	a few seconds ago	9 seconds	1 second

Complete

Jump to your position on the leaderboard

Next Steps

Some of the following steps could help achieve a better score.

Increase Model Size

The CNN model we have is not very deep in nature. Adding more layers would help achieve better performance.

Train Longer

Training the CNN for a longer time will help improve the score. Currently, training is done only for 10 epochs. We can notice in the plot above that the training & validation loss were still decreasing whereas the accuracy was increasing when we stopped at 10 epochs. This implies that the model still has scope for learning further. Increase the number of epochs in `config.py` & try again.

Data Augmentation

The above two mechanisms can lead to overfitting if not careful. In order to avoid overfitting problem, we need to expand artificially our handwritten digit dataset. We can make your existing dataset even larger. The idea is to alter the training data with small transformations to reproduce the variations occuring when someone is writing a digit.

Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as data augmentation techniques. Some popular augmentations are `grayscaling, horizontal & vertical flips, random crops, color jitters, translations, rotations`, and much more.

Summary

How to get started

This project runs on `python 3.7`

1. Create a separate virtual environment
2. Install the dependencies

```
pip install -r requirements.txt
```

3. Training the CNN

```
cd scripts
python train.py
```

4. Make prediction on test data. This will generate a `submission.csv` file which can be submitted on Kaggle

```
cd scripts
python eval.py
```

Project Structure

```
.
├── README.md
├── data
│   ├── test.csv    # testing data
│   └── train.csv   # training data
├── models          # stores models
├── notebooks
│   └── data_exploration.ipynb  # explores data
├── requirements.txt    # stores python dependencies
├── resources       # stores images
│   ├── test.png
│   ├── train.png
│   └── training_distribution.png
└── scripts
    ├── config.py   # allows configuring training
    ├── eval.py     # uses trained CNN to predict on test data
    ├── train.py    # trains CNN on MNIST data
    └── utils.py    # contains utility functions
```