Handwritten Digit Recognition with Neural Networks

Course Project CS 419 M: Introduction to Machine Learning

Github Repository Link -

https://github.com/aakash2314/Handwritten-digit-recognition-using-CNN

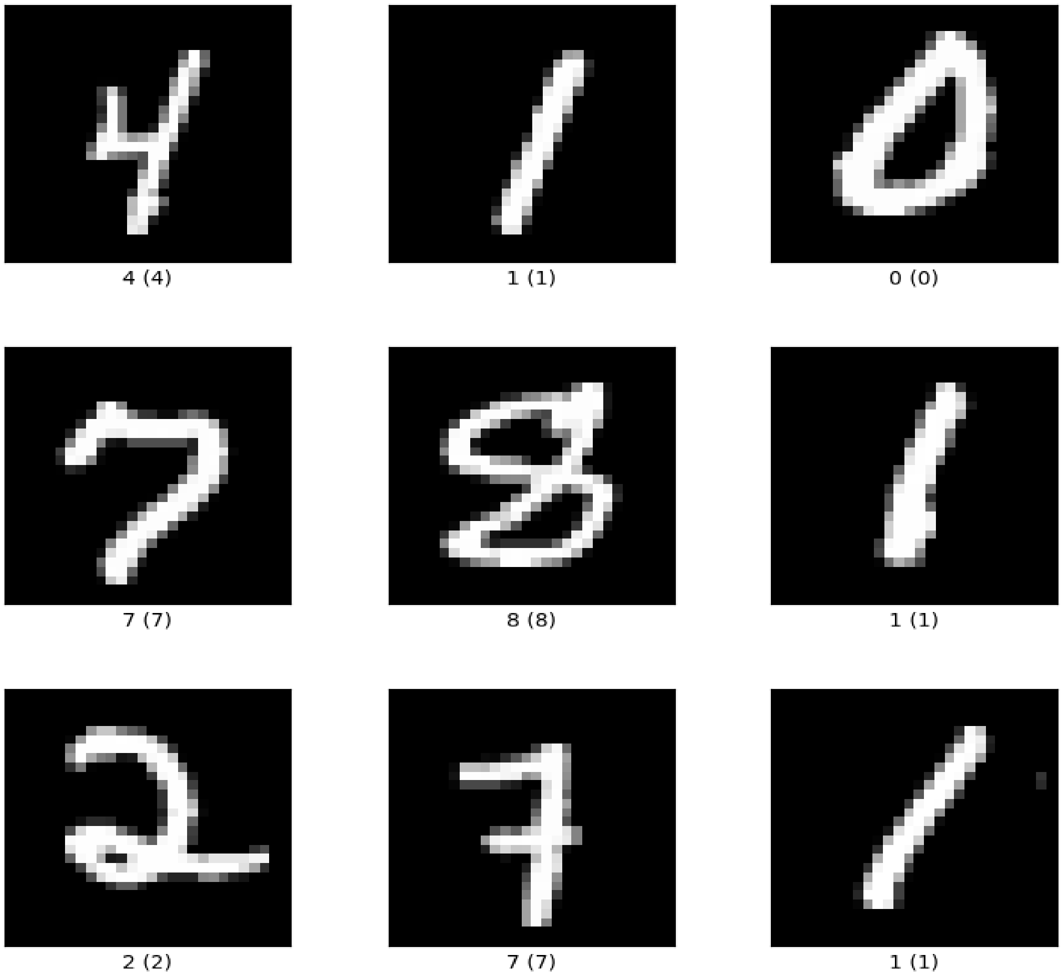
Introduction

Handwritten digit recognition refers to a computer's capacity to recognise human handwritten digits from a variety of sources, such as photographs, papers, and touch displays, and classify them into ten specified categories (0-9). In the realm of deep learning, this has been a topic of inexhaustible investigation. Numerous applications exist for digit recognition, including number plate identification, postal mail sorting, and bank check processing.
Because handwritten digit recognition is not an optical character recognition, we face numerous challenges due to the various styles of writing used by different people. This study compares and contrasts various machine learning and deep learning methods for handwritten digit recognition. We employed Support Vector Machines, Multilayer Perceptrons, and Convolutional Neural Networks to achieve this.
The algorithms are compared based on their accuracy, mistakes, and testing-training time, which are supported by plots and charts created with matplotlib for visualisation. Any model's correctness is critical because more accurate models produce better decisions. Low-accuracy models are unsuitable for real-world applications. For example, great accuracy is important in an automated bank cheque processing system that recognises the amount and date on the check.
If the system reads a digit wrongly, it can cause significant damage, which is not desirable. As a result, in these real-world applications, a high-accuracy algorithm is necessary. As a result, we've put together a comparison of several algorithms based on their accuracy, so that the most accurate method with the fewest errors can be used in a variety of handwritten digit recognition applications.
For handwritten digit recognition, this paper explains how to use machine learning and deep learning algorithms such as SVM, CNN, and MLP. It also tells you which algorithm is the most effective at recognising digits. We will describe similar work that has been done in this subject in the following portions of this paper, as well as the methodology and implementation of all three algorithms for a better understanding of them. The conclusion and result are then presented, which are supported by the research

conducted in this paper. Furthermore, it will show you some possible future improvements in this subject. This paper's citations and references are found in the final section.

MNIST Handwritten Digit Classification DataSet

The Modified National Institute of Standards and Technology dataset is an acronym for Modified National Institute of Standards and Technology dataset.
It's a collection of 60,000 little square grayscale photos of handwritten single numbers ranging from 0 to 9.
The goal is to sort a handwritten digit image into one of ten groups that represent integer values ranging from 0 to 9, inclusively.
The hold out test dataset is a widely used and well-understood dataset that has been "solved" for the most part. Top-performing models are deep learning convolutional neural networks, which achieve a classification accuracy of above 99 percent with an error rate of between 0.4 percent and 0.2 percent.



<u>Baseline Model</u>

The first step is to create a model that serves as a baseline.
This is crucial because it requires both constructing the infrastructure for the test harness so that any model we design can be assessed on the dataset and establishing a baseline in model performance on the problem against which all advances can be measured.
With five basic parts, we can create this test harness. They are the loading of the dataset, the preparation of the dataset, the model definition, the model evaluation, and the results display.

Pre-Processing Dataset

We have some information about the dataset.
For example, we know that the photos are all pre-aligned (e.g., each image only contains a hand-drawn digit), that they are all 2828 pixels square, and that they are grayscale.
As a result, we may import the photos and rearrange the data arrays so that they only have one colour channel.

The pixel values for each image in the dataset are unsigned integers in the range of 0 to 255, or black and white.
We are unsure of the ideal method for scaling pixel values for modelling, however we do know that some scaling will be necessary.
Normalizing the pixel values of grayscale photos, such as rescaling them to the range [0,1], is an excellent place to start. To do so, switch the data type from unsigned integers to floats, then divide the pixel values by the maximum value.

Model building and evaluation

Next, we must define a problem-specific baseline convolutional neural network model. The feature extraction front end, which consists of convolutional and pooling layers, and the classifier backend, which will generate a prediction, are the two key components of the model.
We can start with a single convolutional layer with a small filter size (3,3) and a small number of filters (32) for the convolutional front-end, followed by a max pooling layer. The filter maps can then be flattened to give the classifier features.
We know that in order to forecast the probability distribution of an image belonging to each of the ten classes, we'll need an output layer with 10 nodes because the problem is a multi-class classification task. This will necessitate the usage of the softmax activation function as well. We can add a dense layer between the feature extractor and the output layer to interpret the features, in this case with 100 nodes, between the feature extractor and the output layer.
The ReLU activation function and the He weight initialization approach will be used by all layers, as they are both great practises.
We must evaluate the model after it has been defined.
Five-fold cross-validation will be used to assess the model. The value of k=5 was chosen to serve as a baseline for both repeated evaluation and to avoid requiring a long run time. Each test set will be 20% of the training dataset, or approximately 12,000 samples, which is similar to the size of the real test set for this problem.

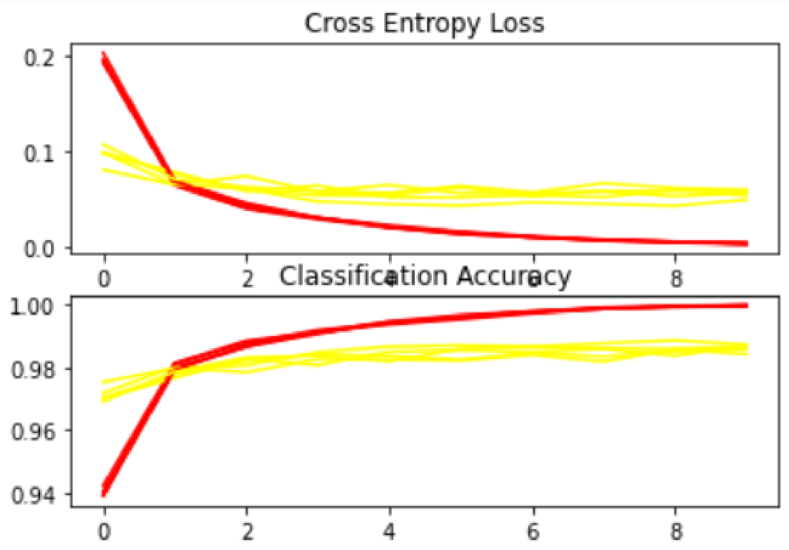
The training dataset is shuffled before being split, and the sample shuffling is done each time, ensuring that each model we evaluate has the same train and test datasets in each fold, allowing us to compare models apples to apples.
With a default batch size of 32 examples, we'll train the baseline model for a modest 10 training epochs. The test set for each fold will be used to evaluate the model throughout each epoch of the training run so that learning curves can be created afterwards, as well as at the end of the run to estimate the model's performance. As a result, we'll keep track of the results from each run, as well as the fold classification accuracy.

Present Results

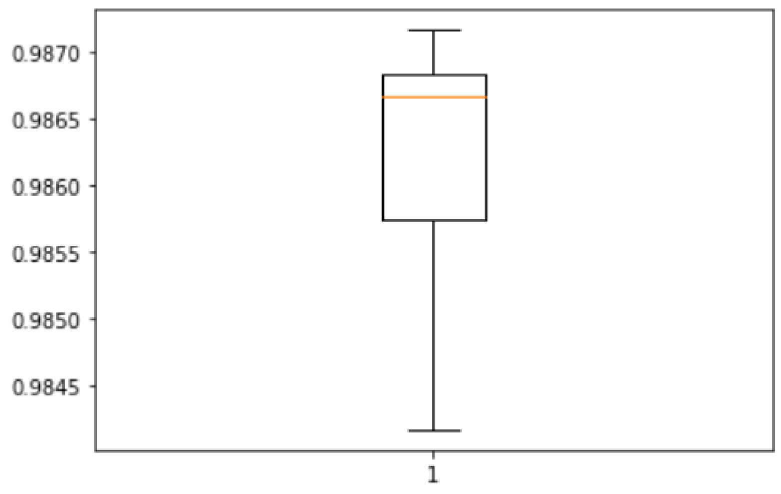
We can present the results after the model has been evaluated.
The diagnostics of the model's learning behaviour during training and the estimation of the model's performance are the two main features to present. Separate functions can be used to implement them.
The diagnostics begin with the creation of a line plot depicting model performance on the train and test set throughout each of the k-fold cross-validation folds. These graphs are useful for determining whether a model is overfitting, underfitting, or has a decent fit for a dataset.

```
run_test_harness()
```

```
> 98.417
> 98.575
> 98.683
> 98.717
> 98.667
```



```
Accuracy: mean=98.612 std=0.108, n=5
```



Depthening of the Model

There are a variety of ways to tweak the model setup to see whether it can improve on the baseline model.
Modifying the capacity of the feature extraction element of the model or changing the capacity or function of the classifier part of the model are two frequent techniques. A tweak to the feature extractor is likely to have the most impact.
We can deepen the feature extractor section of the model by following a VGG-like pattern of adding more convolutional and pooling layers with the same filter size while increasing the number of filters. We'll use a double convolutional layer with 64 filters each, followed by another max pooling layer in this..

Finalizing Model

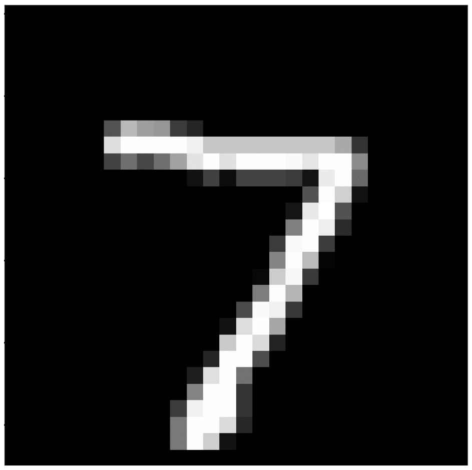
Model improvement may go on indefinitely as long as we have ideas and the time and resources to test them out.
A final model configuration must be determined and implemented at some time. We will use the deeper model as our final model in this scenario.
We'll start by finalising our model by fitting it to the whole training dataset and saving it to a file for future usage. After that, we'll load the model and assess its performance on the holdout test dataset to see how well the chosen model performs in practise. Finally, we'll apply the saved model to a single image and generate a forecast.

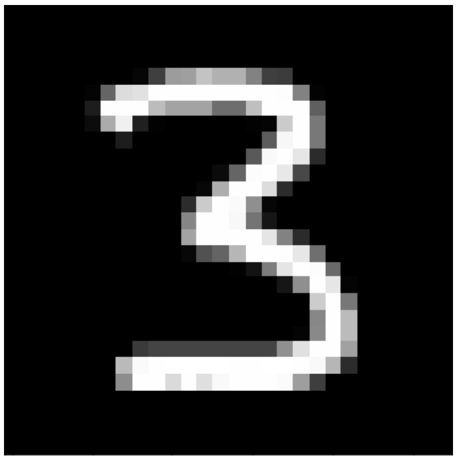
Final Model saving- We can now load and test the final model on the holdout test dataset.
This is something we might do if we wanted to show project stakeholders the performance of the chosen model.

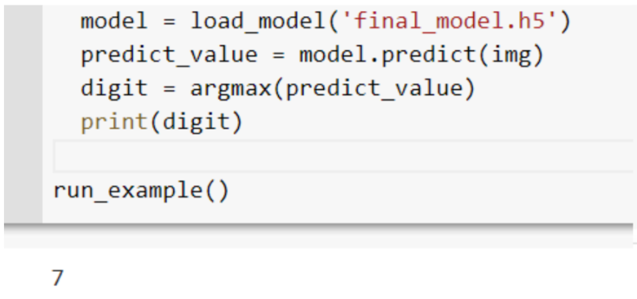
Evaluate Model- We can now load and test the final model on the holdout test dataset. This is something we might do if we wanted to show project stakeholders the performance of the chosen model.

Make Prediction - We can utilise our previously saved model to forecast new photos. The model assumes that the new photos are grayscale, that they have been aligned so that each image contains one centred handwritten number, and that the image size is square (28x28 pixels).

Testing our model using random inputs-



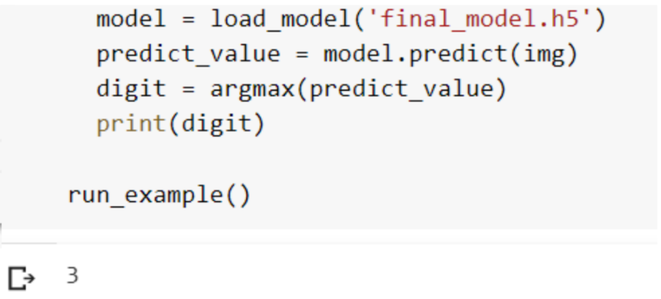




```
    model = load_model('final_model.h5')
    predict_value = model.predict(img)
    digit = argmax(predict_value)
    print(digit)

run_example()
```

```
7
```



```
    model = load_model('final_model.h5')
    predict_value = model.predict(img)
    digit = argmax(predict_value)
    print(digit)

run_example()
```

```
3
```

Prepared by -

1. Prateek Jha (200040106)
2. Pratham Kingar(200040075)
3. Gautam Asodiya(200040054)
4. Aakash Jha(200040002)
5. Dylan Rodrigues(200040049)