Big Data Lab: Assignment-1
Digit Classification via Hand coded ML pipeline

Arjav Singh
IDDD Data Science
Indian Institute of Technology Madras
Chennai, India
mm20b007@smail.iitm.ac.in

***Abstract*—In this assessment, I have created a Machine Learning Pipeline for classifying the handwritten digits. The pipeline utilizes a Deep learning model and MNIST dataset of handwritten digits.**

I. Introduction

The MNIST handwritten digit dataset is a collection of 28x28 pixel grayscale images of handwritten digits (0-9). It serves as a benchmark in the field of machine learning and computer vision for developing and testing algorithms for digit recognition. MNIST contains 60,000 training and 10,000 testing images, making it a widely used dataset for training and evaluating models in image classification. The assessment involved the development of a pipeline for classifying the handwritten digits. To achieve it, the whole process is divided into a set of four tasks involving -

1) The first task is to create a function to rotate a 28 x 28 size image to one of the following angles: -30, -20, -10, 10, 20, 30.
2) Defining a function to create an over-sampled dataset of the rotated images for further training of the model to improve its performance.
3) Model building for the pipeline. I have used CNN as the classifier for the case. In this task, I created a single function that takes the dataset, trains the model, and performs hyperparameter tuning.
4) Finally, the last task involves the development of a function to monitor the model's deployment and check the model's performance using a small sample of ground truth data.

II. Description

Task 1 is necessary for the data augmentation step. I utilized the functional class of the torchvision package because it is simple to rotate an image with its in-build function, but first, it is important to verify that the shape of the input is compatible with the function, for which I used the assert statement.

Building on the rotate image function, Task 2 involves randomly selecting an image from the dataset, applying rotation using the rotate image, and appending the rotated image along with its label to the oversampled dataset.

Key Points to Verify:

1) Oversample Dataset Size: Ensure that the oversample rate, if less than 1, defaults to 2.

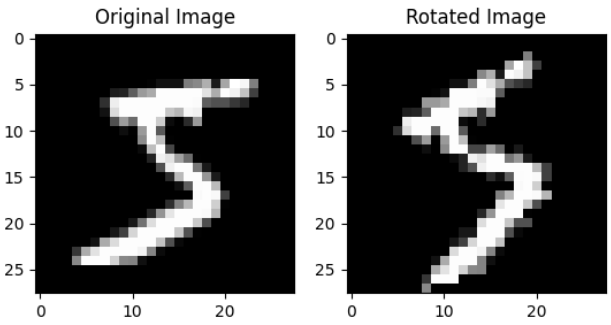

Fig. 1. Original image vs rotated image.

2) Data Structure of Oversampled Dataset: The oversampled dataset should be of list data type, with each element comprising a tuple containing the rotated image tensor and its corresponding label.

This task aims to efficiently increase dataset diversity by introducing random image rotations, contributing to enhanced training data for the subsequent machine learning model.

In the third task, I defined a CNN model for which the architecture comprises four layers. The initial layer consists of 16 convolutional kernels, each with a size of 5x5, resulting in a total of 416 parameters. The subsequent layer introduces 32 convolutional kernels of size 5x5, operating on the 16 images from the preceding layer. With 401 parameters per kernel, the total parameter count for this layer is 12,832. Following the same logic, the third layer, consisting of 64 kernels of size 9x9, accumulates a total of 18,496 parameters. The dense layer receives input from the flattened layer, representing 64 sets of 16 images from the pooling layer. Mapped to a flat array of 1024, the output size is 64, leading to 65,600 parameters. Finally, the last layer, with an input size of 64, has 650 parameters, resulting in a succinct yet comprehensive representation of the CNN's parameterization.

To improve the performance, I tuned two hyperparameters, the convolutional layer's activation function and pool type. Since my system was not supporting the grid search and any other established methods, I used nested for loops to test the different combinations of the hyperparameters.

Moving to the final task, I created the final function to monitor the model's performance. The function will check the model accuracy on the ground truth dataset and raise drift if the loss exceeds the threshold value. For the test purpose, I took the threshold loss to be 2.5.

To simulate potential drift scenarios, I iterated through each angle from -30 to 30, and whenever drift was detected, I added oversampled data to address it. The observed loss values during this process ranged between 2.3 and 2.7. Notably, no drift was observed after the dataset size exceeded 18,000 data points, likely due to the relatively high threshold loss value.

While there is room for improvement in the model and the overall pipeline, the assessment of the pipeline's development has been successfully completed.

REFERENCES

[1] Dhaneshwar, A., "MNIST - CNN + Grid Search + Data Augmentation," Kaggle, Available: https://www.kaggle.com/code/arjav007/mnist-cnn-grid-search-data-augmentation/edit.

[2] Keras. (n.d.). "Model Training APIs - Keras Documentation." Available: https://keras.io/api/models/model_training_apis/.