MNIST data classification

Preprocessing

In order to train this multiclass dataset, a custom neural network has been applied. At first, the loaded dataset is split into train and test categories where train images are implemented for training the custom CNN model. Both the test and train images are reshaped for the CNN model to compute the matrix operation and convert this into the grayscale image to reduce the computation. Therefore, a categorical function is applied to convert the vector label into a binary format class.

Model build and train

In this segment, a custom CNN model is built with input, a convolutional hidden layer, flattened, and two dense layers. Here, max pooling is implemented to reduce the dimensionality and the number of parameters of the feature maps created by each convolution step. Relu is implemented as an activation function in order to add non-linearity and reduce the vanishing gradient problem. The dropout layer has been added to minimize the overfitting problem by cutting unnecessary connecting nodes. Therefore, the Fully connected layer here works to convert the multi-dimensional feature maps of input images into a one-dimensional vector for a dense layer. The softmax activation function is used in the final dense layer to convert the multiclass classification vector into multiclass vector probabilities. There, which value gives close probabilities to input values counted as predicted output.

Here, several Callback technique is applied.EarlyStopping is applied to monitor the validation loss of the model to check whether the model is learning from the input images or not in the large training epoch. Here, patience level is also provided to check the learning growth up to 10 epochs.ModelCHeckpoint is applied to save the model weights so that it can be utilized later and don't have to train the model repeatedly for classification. During forward and backward propagation of the neural network, the model weights are updated and the learning rate plays a significant role here. So, ReduceLROnPlateau working here to monitor the model improvement and if it halts to a certain patient level. Then the function reduces the learning rate so that the weights can update properly.

Model Evaluation

For evaluation, the model performance, classification report, and confusion matrix are utilized. The model provides 98% accuracy in classifying the mnist datasets. From the classification report. it also shows that the precision and recall matrices show promising scores. That means the trained model efficiently predicts actual positive values and less negative values.

In order to implement this, applied libraries in code will be useful. No need to import data from outside. The dataset is already there and doesn't have to train the model as model weights are saved in the first training.

Working with databases

Here, I have created a PostgreSQL database named ‘product_mangement’. Product and Stock location, two tables have been created and inserted data into it. The product table has a total of 9 columns and Stock Location has a total of 7 columns.

I have structured this database keeping in mind a warehouse scenario where different types of products have been stocked in different locations with their unique id. That's why, the product table contains a unique ID, product, product type (books, electronics, furniture, etc.), price of the product, sold quantity and to track the location I have used stock_location id and stock_location name to the corresponding product. There’s an entry date column for each product. A stock location table is created to track location information such as which location has which type of product, how many slots is available in that particular location, and how many slots are occupied in a location by product and last entry date on that location. This information will benefit during product allocation into the warehouse.

The code is divided into several functions based on creating a table, data retrieving, updating, deleting, and fetching. A library named pysopg2 is applied to connect to the database and do the above operation. The user should know the column info of a table and which function needs to be assigned data to activate the operation. For data retrieving, updating, deleting, and fetching user should provide data into the function.

API Operation

Here, Google Sheet API is used for the integration. At first, a new project and service account had to be created for API service from where the credential JSON file is retrieved. After that, some data is added in Google Sheets, and from that spreadsheet id is collected.

To connect with the API first an HTTPS link is added which specifies the access level needed for the Google Sheets API. It loads the credentials from the specified service_account_file and is used for authentication with Google's API services. The variable spreadsheet_id is set to the unique ID of the Google Spreadsheet from which data will be retrieved. After that, created a Google Sheets API service object using the build method. The sheets argument specifies the API to build,’v4’ is the version of the API.
Then, a get method is used to make the API request and called to retrieve data from the specified range in the spreadsheet.

The retrieved data remains in JSON format and in a key named valued as a list of lists. From that, I implemented a panda to process that data and formed a data frame with columns and data. The data range could vary according to selection.Some, Google API CLient library is needed to operate this and will work any valid data.