Build Deep Autoencoders model for Anomaly Detection in Python Project Overview

Objective:

Autoencoders are used to learn compressed representations of raw data with Encoder and decoder as sub-parts. As a part of a series of Deep Learning projects, this project briefs about Autoencoders and its architecture. In this project, we build a deep learning model based on Autoencoders for Anomaly detection and deploy it using Flask. If you haven't already visited, here is the previous project of the series Deep Learning Project for Beginners with Source Code Part 1.

Aim :

- To understand the theory behind Autoencoders
- To develop a deep learning model based on Autoencoders to learn distributions and relationships between features of normal transactions
- To deploy the model using Flask

Data Overview:

The dataset used is a transaction dataset, and it contains information for more than 100K transactions over several features.

Tech Stack:

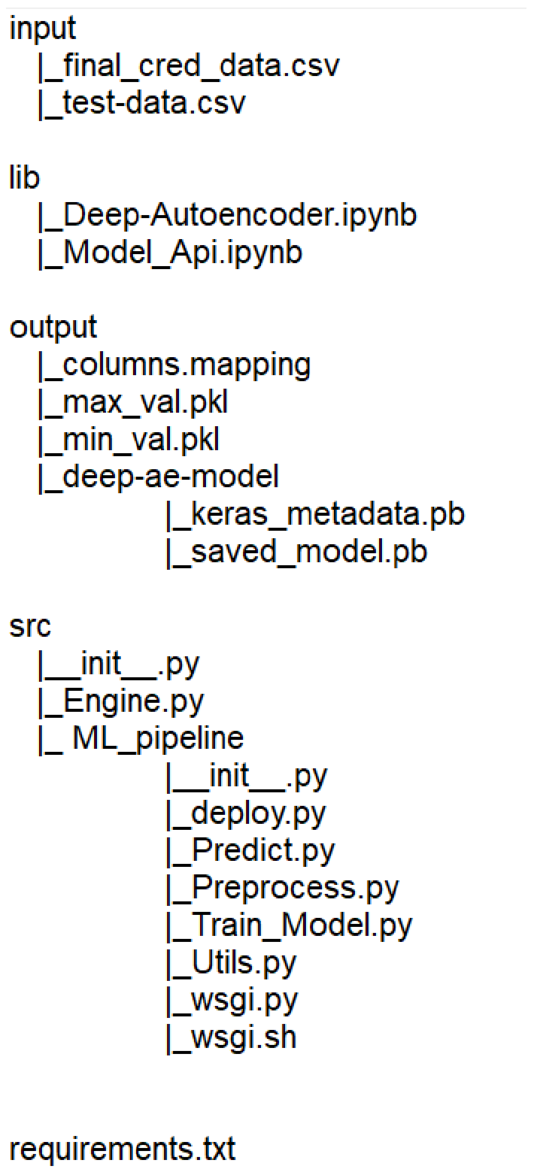
- ➔ Language: Python
- ➔ Packages: Pandas, Numpy, matplotlib, Keras, Tensorflow
- ➔ API service: Flask, Gunicorn

Approach

1. Understand business objective
2. Understand the data using EDA
3. Normalize and clean the data using Imputation
4. Understand idea behind Auto-encoders
5. Build a base auto-encoder model using Keras
6. Tune the model to extract the best performance
7. Extract predictions
8. Serve model as API endpoint using Flask

9. Perform real-time predictions

Modular code:



```
input
  |_final_cred_data.csv
  |_test-data.csv

lib
  |_Deep-Autoencoder.ipynb
  |_Model_Api.ipynb

output
  |_columns.mapping
  |_max_val.pkl
  |_min_val.pkl
  |_deep-ae-model
            |_keras_metadata.pb
            |_saved_model.pb

src
  |__init__.py
  |_Engine.py
  |_ ML_pipeline
            |__init__.py
            |_deploy.py
            |_Predict.py
            |_Preprocess.py
            |_Train_Model.py
            |_Utils.py
            |_wsgi.py
            |_wsgi.sh


requirements.txt
```

The ipython notebook is modularized into different functions so that the user can use those functions instantly whenever needed. The modularized code folder is structured in the following way.

Once you unzip the modular_code.zip file you can find the following folders within it.

1. input
2. src

3. output
4. lib

1. The input folder contains all the data that we have for analysis. In our case, it will contains two csv files which are
    a. final_cred_data.csv
    b. Test-data.csv

2. The src folder is the heart of the project. This folder contains all the modularized code for all the above steps in a modularized manner. It further contains the following.
    a. ML_pipeline
    b. engine.py

The ML_pipeline is a folder that contains all the functions put into different python files which are appropriately named. These python functions are then called inside the engine.py file

3. The output folder contains all the models that we trained for this data saved as .pkl files. These models can be easily loaded and used for future use and the user need not have to train all the models from the beginning.

4. The lib folder is a reference folder. It contains the original ipython notebook that we saw in the videos.

5. The requirements.txt file has all the required libraries with respective versions. Kindly install the file by using the command **pip install -r requirements.txt**

Project Takeaways:

1. What are Autoencoders?
2. How do Autoencoders work?
3. What is the architecture of Autoencoders?
4. How to build Autoencoders using Keras?
5. How to tune an Autoencoder model in Keras?
6. What are the different types of Autoencoders?
7. What is the loss function in Autoencoders?
8. What are the applications of Autoencoders?
9. What is PCA?
10. How is PCA different from Autoencoders?

11. What is anomaly detection?
12. How to perform EDA?
13. How to clean the data?
14. How to normalize the data?
15. How to build an API using Flask?
16. How to serve a model using Flask as an API endpoint?