**Bank Telemarketing Classification Problem using Deep Learning**

**Introduction:**

We are given the data of direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (target variable y). This case study is inspired by this research paper where the researchers have used a very similar dataset as the one we will be using throughout this case study for determining the success of Bank Telemarketing. The result shows the highest accuracy of 89.27%.

**Methodology:**

I have utilized a ANN (Artificial Neural Network) architecture in my telemarketing model. ANN is generally used for image analysis, segmentation, classification, medical image analysis, photo and video recognition, etc. In this project, **I used TensorFlow with Keras. I** first applied feature engineering on data and processing technique to transform the raw data into an efficient and useful format. Later, the ANN architecture was applied. Here, it has been decomposed into two parts:

* Feature Extraction
* Classification

The Convolution and the pooling layers perform the feature extraction of images that extract information from the input for decision making. Finally, a fully connected layer serves as the classification part.

**Dataset:**

The dataset was Bank Telemarketing dataset with target label “y”. The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. <https://archive.ics.uci.edu/ml/datasets/bank+marketing>

**Pre-Processing:**

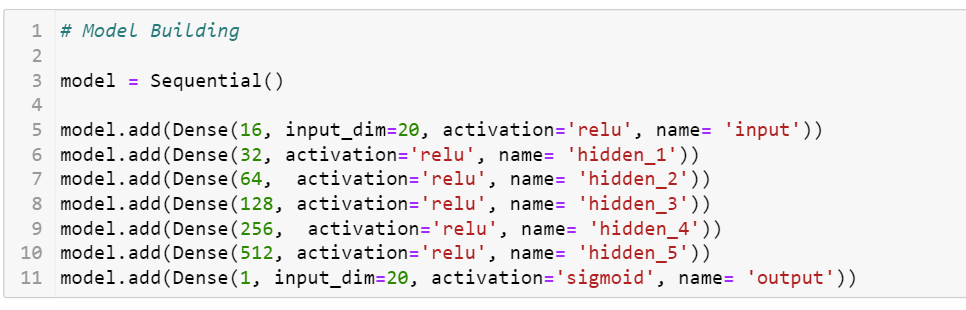
I checked for the null values in the data and found none. But interestingly the data was highly imbalance as there was more than 70% of the data which has “No” as a value for our response variable. Then I used balancer to make sure all the values in our response variable are equally balanced. I applied feature engineering to convert the categorical variable like age, month, year etc. and applied MinMaxScaler to make sure the data points are scaled for model.

**Project Workflow:**

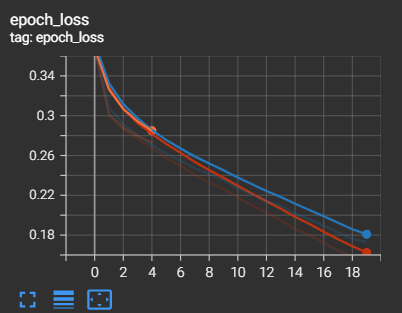
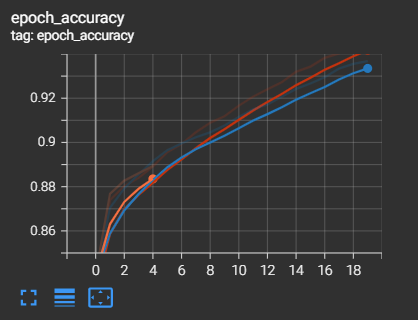
**TRAINING MODEL:**

I used Python for creating this ML model. Packages used:

* Matplotlib,
* Keras
* NumPy   
  were utilized for system implementation. Keras provides built-in functions such as activation functions, optimizers, layers, etc. TensorFlow was also used as the system's back-end.



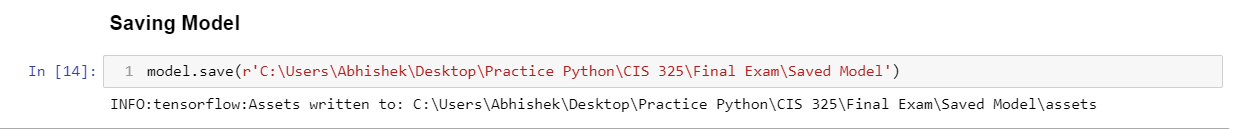
**MODEL ACCURACY REPRESENTATION:**

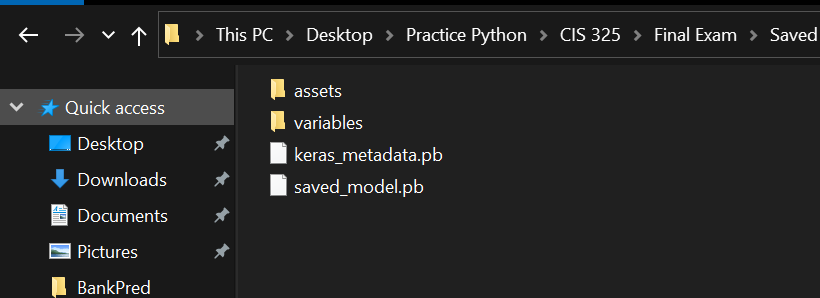


The above graph shows the accuracy of multiple models runs and how the accuracy has increased after each change in features, nodes, and variables.

**EXPORTING MODEL:**

You must first export your trained models in TensorFlow SavedModel format before deploying them to AI Platform Prediction and using them to deliver predictions. TensorFlow's recommended format for exporting models is a SavedModel, and it is required for deploying trained TensorFlow models on AI Platform Prediction. When you export your trained model as a SavedModel, you save your training graph, including its assets, variables, and metadata, in a format that AI Platform Prediction can ingest and restore for predictions.



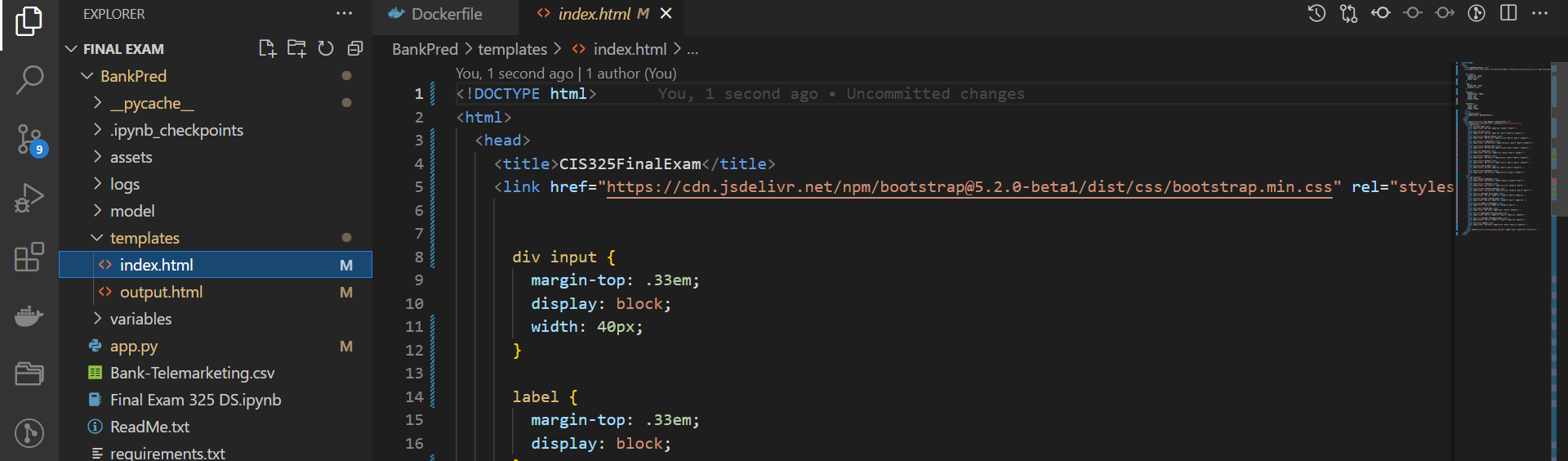


**CREATING HTTP & FLASK BASED WEB APPLICATION:**

In order to create a Gender Detection web application, we have created two different files:

* Flask for the back-end engine - app.py
* HTML for the front-end - /templates/index.html

The front-end HTML acts as a medium to interact with and accept input from the user, who then receives predictions from the model. First, the POST request is received from the HTML. Then, when a request is received, the ML model is loaded, and the input file is pre-processed according to the steps described in the Flask back-end engine. Finally, the model generates the prediction, which is subsequently returned to the user via the *render\_template()* function.





Snippets of the flask backend engine and HTML front end.

**GITHUB & SCM:**

Source code management (SCM) is a technique for tracking changes to a source code repository. SCM maintains a running history of modifications to a code base and aids in dispute resolution when integrating updates from various contributors.

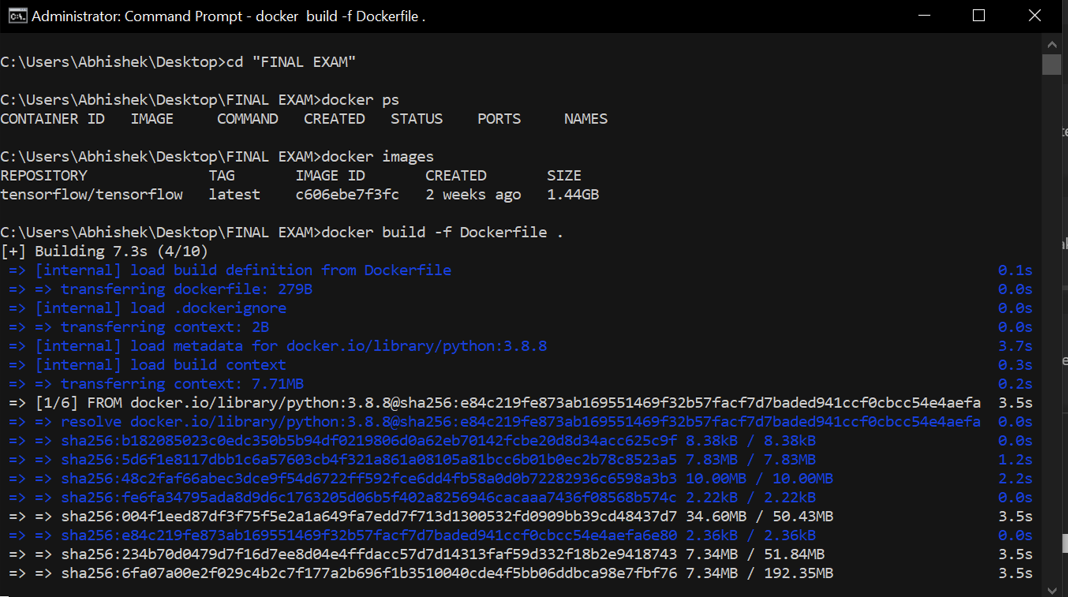
Visual Studio Code can be a valuable tool if you're working with a remote repository. You only need to add your remote repository to VS to be able to control the changes. Another option is to execute git commands from the terminal. Both of them can be seen in the snapshot below.

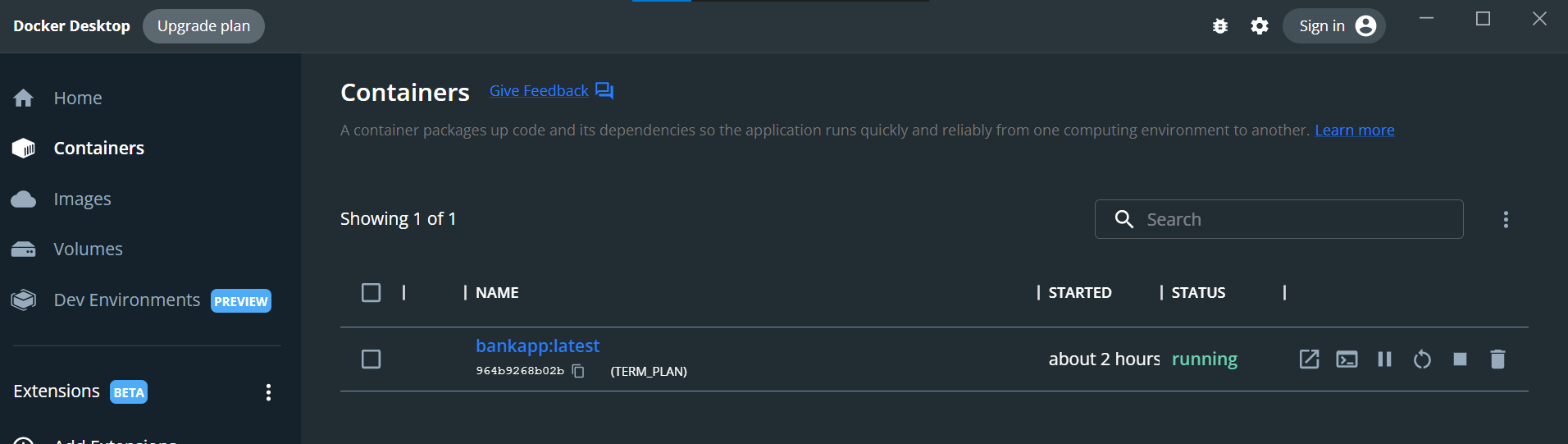
**CREATING A CUSTOM IMAGE FROM DOCKER:**

When we want to automate and operate a custom application, the easiest option to deploy your containerized application is to create your own docker image with the desired configuration. This custom configuration may be passed using the Dockerfile, and a container can then be created using this image.

Once the image is ready, it can be used to deploy containers in Docker. In order to test our application on Docker, we have created a container using *docker run -d -p 5000:5000 --name TERM PLAN <IMAGE\_NAME: VERSION>* command.

The application can be seen running at *localhost:5000*.





**PUSHING LOCAL DOCKER IMAGE TO AWS ELASTIC CONTAINER REGISTRY ECR:**

To push a local docker image to AWS, we must first configure AWS CLI for the first time. This can be performed by using the instructions listed in the [ReadMe file](https://github.com/anuragyadav16/model-latest/blob/main/GenderApp/ReadMe.txt). Once the CLI is ready, we must use the terminal to create a repository and push the local docker image to AWS ECR.

**DEPLOYING ELASTIC CONTAINER SERVICE ECS USING AWS FARGATE:**

AWS FARGATE is a technology that you can use with Amazon ECS to run containers without having to manage servers or clusters of Amazon EC2 instances. With AWS FARGATE, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters or optimize cluster packing.

When running your tasks and services with the FARGATE launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each FARGATE task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

**RESULTS:**

**Access this Application over Internet:** **http:// 54.87.120.37:5000/**

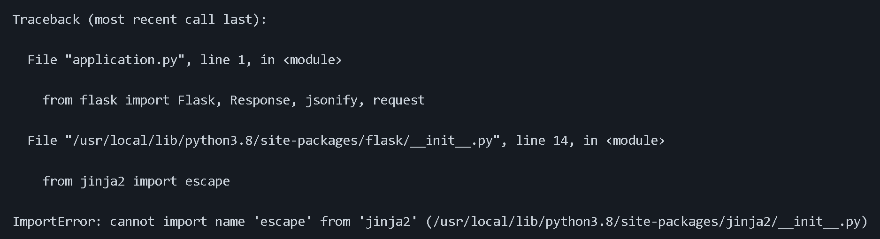
Github Link: <https://github.com/ABHISHEK9894/FINAL-EXAM-1.git>

**CHALLENGES / ERRORS ENCOUNTERED:**

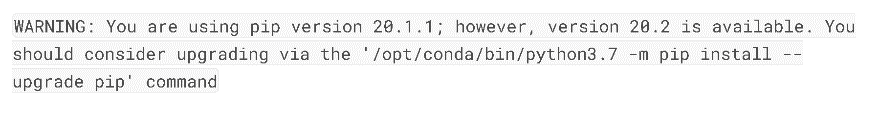
1. Invalid public key for CUDA apt repository

****

1. Flask failing to startup due to Jinja2 breaking change

****

1. Pip version error

****