

COSC-6376 Assignment-4

Shaili Dave

PSID: 2052659

For your 4th Assignment You will do transfer learning on a well-known dataset in Sagemaker (or Vertex AI), and expose that inference engine via an API. The base data would be stored in a database of your choice

Task in Detail:

- You will use the MNIST dataset (<https://huggingface.co/datasets/mnist>, <https://deepai.org/dataset/mnist>, <https://www.kaggle.com/datasets/oddrationale/mnist-in-csv>)
- The dataset must be in a database
- You will have a sagemaker backend interacting with this data
- You will do transfer learning on your own handwritten digits on top of the existing dataset
- Generate an API which takes an image data (hosted image so it should take a URL of image as input) and give you the result

Steps followed in this assignment using Vertex AI on Google Cloud Platform

Upload Data to Database (Big Query)

1. Create cloud_assignment_4_data Bucket on Google Cloud Platform and Upload MNIST Data csv files to the bucket

The screenshot shows the Google Cloud Storage interface. On the left, there's a sidebar with 'Cloud Storage' selected under 'Buckets'. The main area displays the 'Bucket details' for 'cloud_assignment_4_data'. It shows the location as 'us (multiple regions in United States)', storage class as 'Standard', public access as 'Not public', and protection as 'None'. Below this, there are tabs for 'OBJECTS', 'CONFIGURATION', 'PERMISSIONS', 'PROTECTION', 'LIFECYCLE', and 'OBSERVABILITY'. The 'OBJECTS' tab is active, showing a list of objects: 'job-dir/' (Folder), 'mnist_train.csv' (text/csv, 104.9 MB, created Nov 19, 2022, last modified Nov 19, 2022, standard storage, not public), and 'mnist_test.csv' (text/csv, 17.5 MB, created Nov 18, 2022, last modified Nov 18, 2022, standard storage, not public). At the bottom of the list, there are filters for 'Name', 'Size', 'Type', 'Created', 'Storage class', 'Last modified', 'Public access', 'Vers', and a 'Show deleted data' button.

The screenshot shows the 'Create a bucket' wizard in the Google Cloud Platform. On the left, there's a sidebar with 'Cloud Storage' selected, showing 'Buckets', 'Monitoring (NEW)', and 'Settings'. The main area has a title 'Create a bucket' with a back arrow. It lists several steps: 'Name your bucket' (with input field 'cloud_assignment_4_data'), 'Choose where to store your data' (Location: us, Multi-region), 'Choose a storage class for your data' (Standard), 'Choose how to control access to objects' (Public access prevention: On, Access control: Uniform), and 'Choose how to protect object data' (None). To the right, a 'Good to know' section includes 'Location pricing' (Storage rates vary depending on location) and a table for 'Current configuration: Multi-region / Standard'. The table shows costs for 'us (multiple regions in United States)' and 'With default replication'. A 'HELP ASSISTANT' button is at the top right.

Also, create bucket `loud_assignment_4_model` in which specify region `us-central 1` (Iowa). This bucket is created to store model and we will specify region same in our vertex ai workbench notebook.

It is necessary in google cloud that all resources should be same region otherwise it will give error

This screenshot is identical to the one above, showing the 'Create a bucket' wizard. The sidebar shows 'Buckets', 'Monitoring (NEW)', and 'Settings'. The main area shows the same five steps: naming the bucket 'cloud_assignment_4_model', choosing storage location 'us' (Multi-region), selecting 'Standard' storage class, setting access control to 'Uniform', and choosing no protection tools. The 'Good to know' section, location pricing table, and help assistant button are also present.

2. For database we are using Big Query so go to BigQuery and create database `cloud_assignment_4`

3. Create Table `mnist_train` and `mnist_test` and transfer dataset from cloud storage to BigQuery Database

Google Cloud

BigQuery

Analysis

SQL workspace

Data transfers

Scheduled queries

Analytics Hub

Dataform

Migration

SQL translation

Administration

Monitoring

Capacity management

BI Engine

Release Notes

Create table

Source

Create table from Google Cloud Storage

Select file from GCS bucket or [use a URI pattern](#) *

cloud_assignment_4_data/mnist_test.csv [BROWSE](#) [?](#)

File format CSV

Source Data Partitioning

Destination

Project * logical-app-367922 [BROWSE](#)

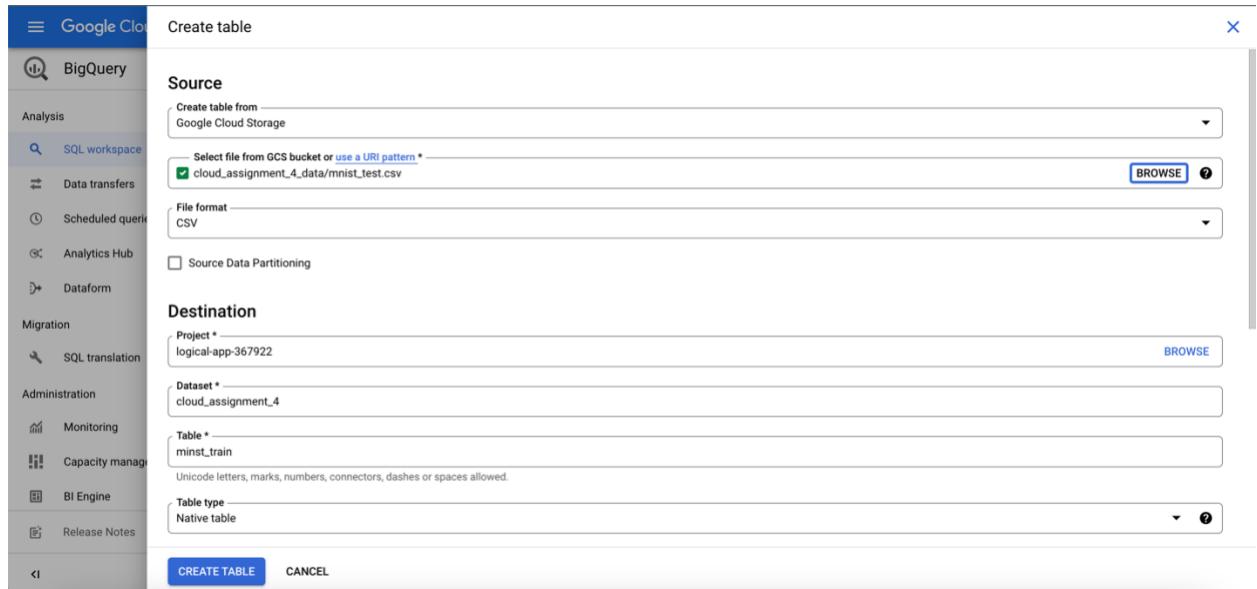
Dataset * cloud_assignment_4

Table * mnist_train

Unicode letters, marks, numbers, connectors, dashes or spaces allowed.

Table type Native table

[CREATE TABLE](#) [CANCEL](#)



Google Cloud

BigQuery

Analysis

SQL workspace

Data transfers

Scheduled queries

Analytics Hub

Dataform

Migration

SQL translation

Administration

Monitoring

Capacity management

BI Engine

Release Notes

Create table

Source

Create table from Google Cloud Storage

Select file from GCS bucket or [use a URI pattern](#) *

cloud_assignment_4_data/mnist_test.csv [BROWSE](#) [?](#)

File format CSV

Source Data Partitioning

Destination

Project * logical-app-367922 [BROWSE](#)

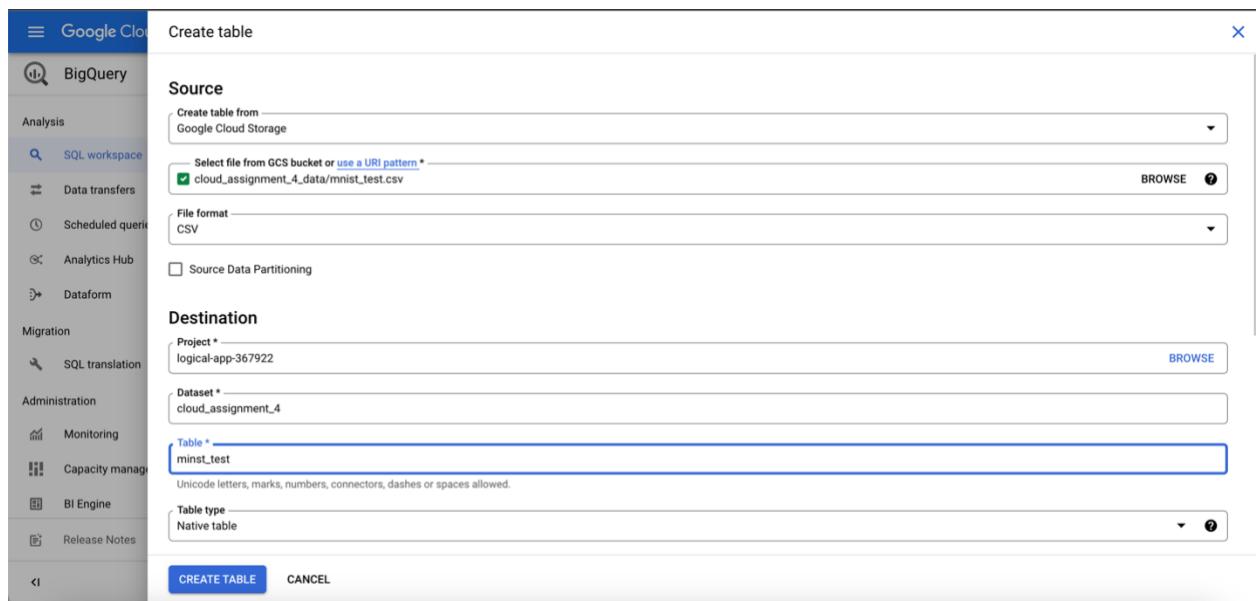
Dataset * cloud_assignment_4

Table * mnist_test

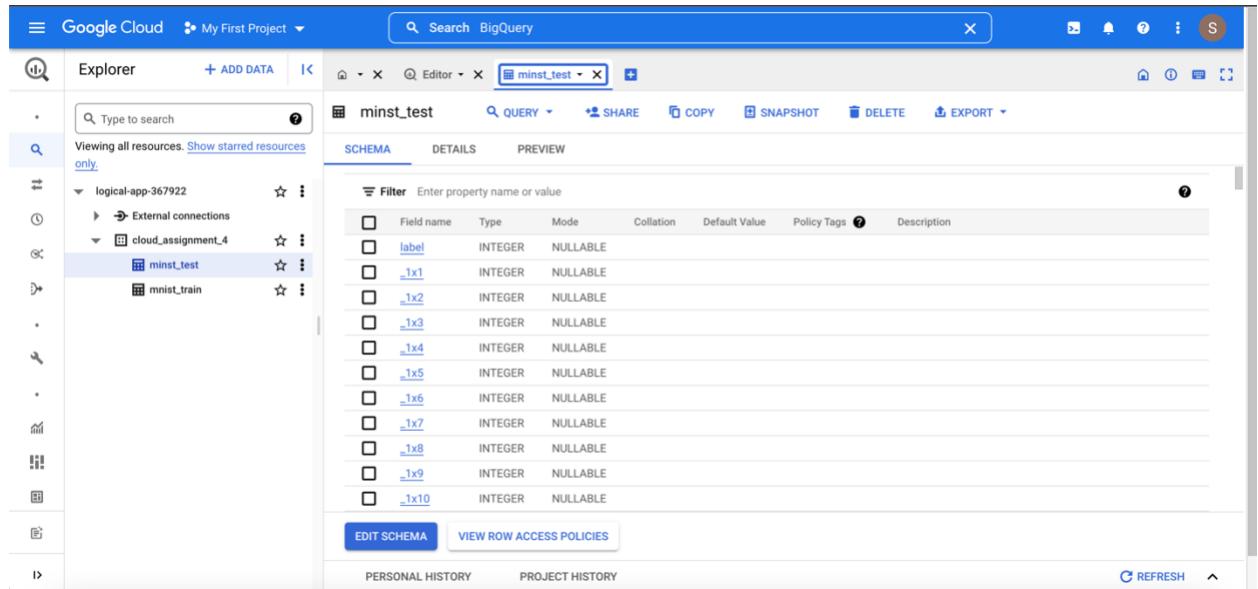
Unicode letters, marks, numbers, connectors, dashes or spaces allowed.

Table type Native table

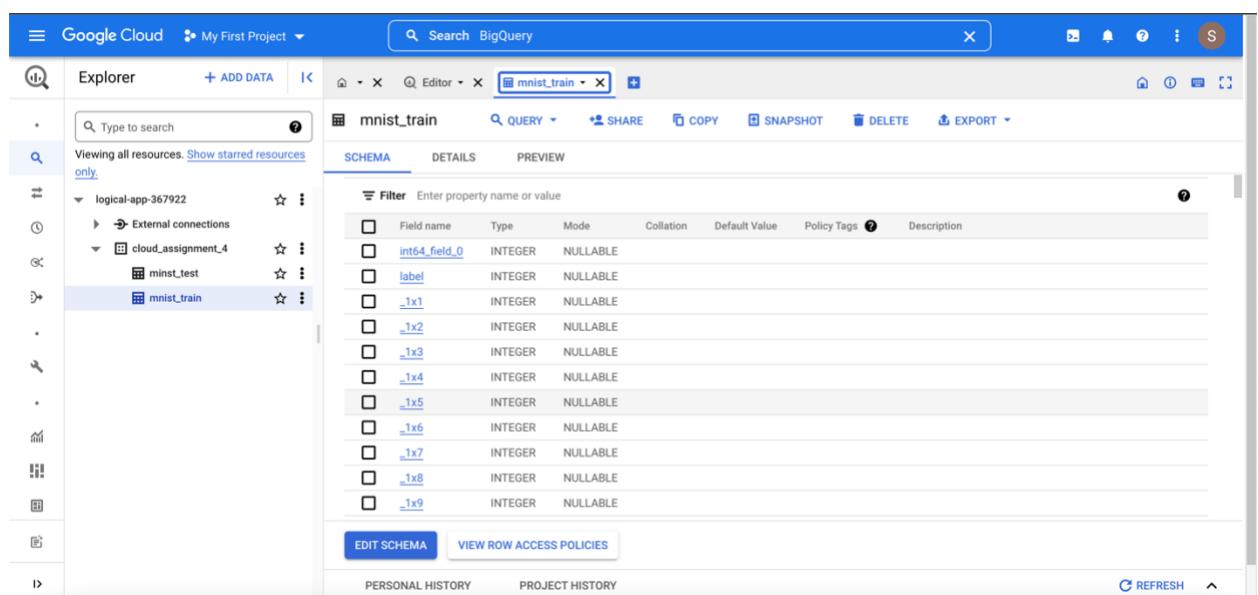
[CREATE TABLE](#) [CANCEL](#)



MNIST test and train Data in Database



The screenshot shows the Google Cloud BigQuery interface. The left sidebar displays the project navigation, including 'My First Project' and various resources like 'logical-app-367922' and 'cloud_assignment_4'. The main area is titled 'mnist_test' and shows the schema for this dataset. The schema consists of 10 columns, all of which are of type INTEGER and are nullable. The columns are labeled '_1x1', '_1x2', '_1x3', '_1x4', '_1x5', '_1x6', '_1x7', '_1x8', '_1x9', and '_1x10'. There are tabs for 'SCHEMA', 'DETAILS', and 'PREVIEW'. Below the schema table are buttons for 'EDIT SCHEMA' and 'VIEW ROW ACCESS POLICIES'. At the bottom are 'PERSONAL HISTORY' and 'PROJECT HISTORY' links, along with a 'REFRESH' button.



This screenshot shows the Google Cloud BigQuery interface again, but this time for the 'mnist_train' dataset. The schema is identical to the 'mnist_test' dataset, featuring 10 columns labeled '_1x1' through '_1x10' as INTEGER types with NULLABLE modes. The interface includes the same navigation bar, schema editor, and history features as the previous screenshot.

Vertex AI Backend Interacting with data

1. Go to vertex AI on Google Cloud Platform, go to Managed Notebook, and create new notebook

The screenshot shows the 'Create a managed notebook' dialog in the Vertex AI Workbench section of the Google Cloud Platform interface. The left sidebar shows 'Vertex AI' selected under 'Workbench'. The main form has the following fields:

- Notebook name ***: managed-notebook-1669143177
- Region**: us-central1 (Iowa)
- Permission**:
 - Service account
 - Single user only
- User email ***: shailidave09@gmail.com
- Advanced settings** (button)
- CREATE** (button)
- CANCEL** (button)

On the right, there's a summary: **\$202.09 monthly estimate**, with a note: "That's about \$0.277 hourly". Below it, a note says: "Pay for what you use: No upfront costs and per second billing Networking cost also applies. [Learn more](#)". A 'DETAILS' button is also present.

2. Assign 30GB RAM in Machine type, Hardware Configuration

The screenshot shows the 'Create a managed notebook' dialog with the 'Hardware configuration' section expanded. The left sidebar shows 'Vertex AI' selected under 'Workbench'. The main form has the following settings:

- Environment**:
 - Custom docker images
 - Provide custom docker images
- Hardware configuration**:
 - Machine type ***: n1-standard-8 (8 vCPUs, 30 GB RAM)
 - GPU type**: None
 - Data disk type**: Standard Persistent Disk
 - Data disk size in GB ***: 100
 - Delete to trash?
 - Disk encryption**:
 - Google-managed encryption key
 - No configuration required
 - Customer-managed encryption key (CMEK)

On the right, there's a summary: **\$401.38 monthly estimate**, with a note: "That's about \$0.550 hourly". Below it, a note says: "Pay for what you use: No upfront costs and per second billing Networking cost also applies. [Learn more](#)". A 'DETAILS' button is also present.

3. Activate managed notebook instance by clicking on start button as shown in screen shot and open TensorFlow Notebook

Status	Region	Machine type	GPU	Owner
ACTIVE	us-central1	n1-standard-8 (8 vCPUs, 30 GB RAM)	None	shailidave09@gmail.com

The screenshot shows a Jupyter Notebook interface running on an n1-standard-8 machine. The left sidebar displays a file tree with several files and images. The main area shows a code cell with Python imports and a log of TensorFlow compilation errors related to CUDA and TensorRT.

```
[3]: import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers as L
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

2022-11-19 19:16:06.043115: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-11-19 19:16:10.732243: E tensorflow/stream_executor/cuda/cuda_blas.cc:298] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2022-11-19 19:16:18.966927: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer.so.7'; dlsym: libnvinfer.so.7: cannot open shared object file: No such file or directory; LD_LIBRARY_PATH: /usr/local/cuda/lib64:/usr/local/cuda/lib:/usr/local/lib/x86_64-linux-gnu:/usr/local/nvidia/lib64
2022-11-19 19:16:18.967295: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer_plugin.so.7'; dlsym: libnvinfer_plugin.so.7: cannot open shared object file: No such file or directory; LD_LIBRARY_PATH: /usr/local/cuda/lib64:/usr/local/cuda/lib:/usr/local/lib/x86_64-linux-gnu:/usr/local/nvidia/lib64
2022-11-19 19:16:18.967321: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with TensorRT, please make sure the missing libraries
```

4. Vertex AI is interacted with Big Query Database, click on the icon shown in screen shoot it will load our created database cloud_assignment_4

The screenshot shows the BigQuery interface within a Jupyter Notebook. The left sidebar shows a 'Resources' section with a table named 'mnist_test' selected. The main area displays the table's details, including its ID, size, and creation date.

Table ID	logical-app-367922.cloud_assignment_4.mnist_test
Table size	59.89 MB
Number of rows	10,000
Created	Nov 18, 2022, 2:59:04 PM
Table expiration	Never
Last modified	Nov 18, 2022, 2:59:04 PM
Data location	US

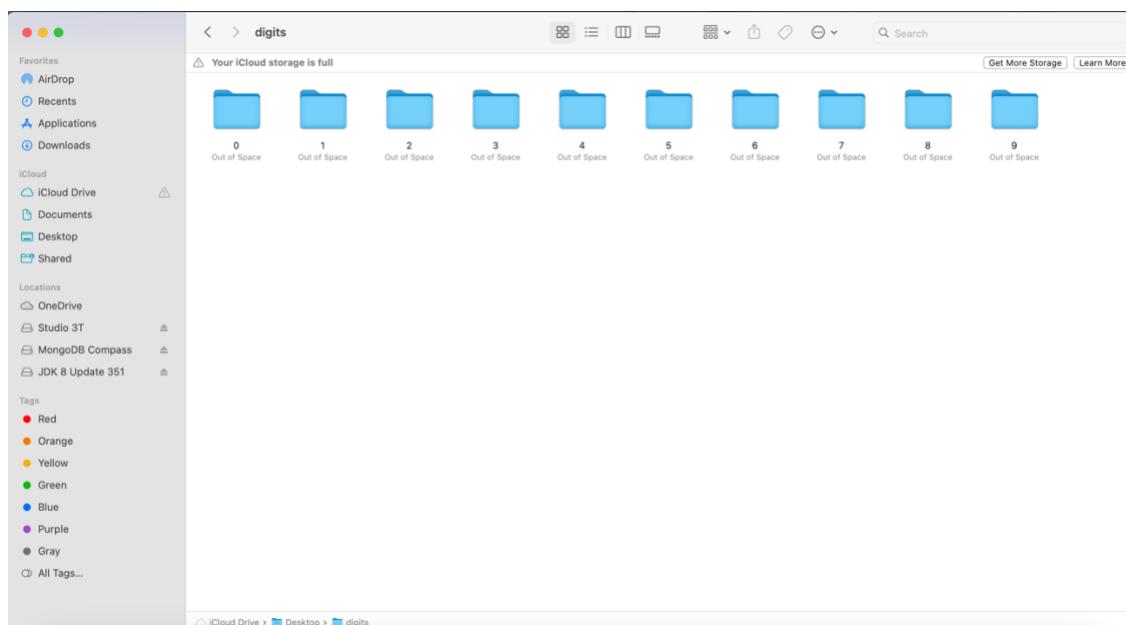
5. Click on the database, then click Submit Query and retrieve code of query to create tables in our database

The screenshot shows the BigQuery interface with the following details:

- Project:** managed-notebook-1668820099
- Environment:** n1-standard-8
- Query Editor 2:** Contains the query: `SELECT * FROM `logical-app-367922.cloud_assignment_4.mnist_test` LIMIT 1000`.
- Resources:** Shows datasets: `bigrquery-public-data`, `logical-app-367922`, and `cloud_assignment_4`. Under `cloud_assignment_4`, there are two tables: `mnist_test` and `mnist_train`.
- Query results:** A table with 7 rows and 14 columns. The columns are labeled: Row, label, _1x1, _1x2, _1x3, _1x4, _1x5, _1x6, _1x7, _1x8, _1x9, _1x10, _1x11, _1x12, _1x13, _1x14.

Transfer Learning on my own handwritten digits on top of existing dataset

1. Open Paint S in Mac and create 20 handwritten images of 28 * 28 pixels, each digit comprises two images
2. Create folder for each digit's image as shown in screen shot and store all these images in one folder Digits



3. Open jupyter notebook locally and create code to convert all images to grey scale and then flatten image and convert it to list

```

In [29]: # read dataset from local file
df = pd.read_csv("/Users/shailidave/Desktop/Cloud Computing COSC 6376/Assignment 4/archive/mnist_train.csv")

In [11]: # Created folder digits and subfolder for each digits image
dir_list = os.listdir("/Users/shailidave/Desktop/digits")
dir_list

Out[11]: ['DS_Store', '9', '0', '7', '6', '1', '8', '4', '3', '2', '5']

In [30]: # Loop through digits airectory
for i in range(1,len(dir_list)):
    path = "/Users/shailidave/Desktop/digits"+"/"+str(dir_list[i])
    images = os.listdir(path)

    # Loop through each number's folder
    for img in images:
        img_path = "/Users/shailidave/Desktop/digits"+"/"+str(dir_list[i])+"/"+img

    # Open image
    image = Image.open(img_path).convert('L')

    # Flatten Image
    flatten_image = np.array(image).flatten()

    # Convert it to list
    flatten_image = flatten_image.tolist()
    flatten_image.insert(0, i-1)
    print(flatten_image)

    # Append image array to dataset
    df.loc[len(df)] = flatten_image

```

4. Append this list along with label to mnist_train.csv

```

In [31]: # Updated dataframe with hand written digits images
df

Out[31]:
label 1x1 1x2 1x3 1x4 1x5 1x6 1x7 1x8 1x9 ... 28x19 28x20 28x21 28x22 28x23 28x24 28x25 28x26 28x27 28x28
0 5 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
2 4 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
3 1 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
4 9 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
...
... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
60015 7 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
60016 8 0 0 0 0 0 1 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
60017 8 48 48 48 48 48 48 48 48 47 ... 48 48 48 48 48 48 48 48 48 85
60018 9 48 48 48 48 48 48 48 48 48 ... 48 48 48 48 48 48 48 48 48 85
60019 9 48 48 48 48 48 48 48 48 48 ... 48 48 48 48 48 48 48 48 48 85

60020 rows × 785 columns

In [32]: # Save csv file |
df.to_csv("/Users/shailidave/Desktop/mnist_train.csv")

```

5. Upload updated mnist_train.csv to cloud_assignment_4 google cloud bucket
6. Also, create new table in Big Query Database named mnist_train and remove the old one then read dataset in vertex AI jupyter notebook

```
[12]: # Read dataset from bigquery
from google.cloud.bigquery import Client, QueryJobConfig
client = Client()

[46]: query1 = """SELECT * FROM `logical-app-367922.cloud_assignment_4.minst_test`"""
job1 = client.query(query1)
test = job1.to_dataframe()
test
```

	label	_1x1	_1x2	_1x3	_1x4	_1x5	_1x6	_1x7	_1x8	_1x9	...	_28x19	_28x20	_28x21	_28x22	_28x23	_28x24	_28x
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0

7. Now we read data from database, remove labels from dataset and reshape training array as shown in screenshot

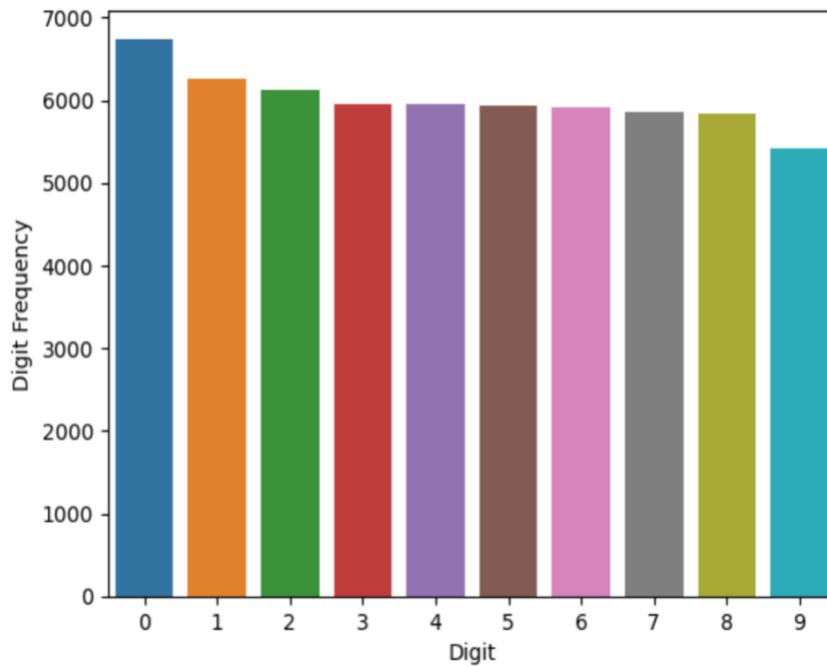
```
[16]: train_arr = np.array(train).reshape(-1, 28, 28, 1)

[17]: train_arr.shape
```

[17]: (60020, 28, 28, 1)

Data Visualization

8. Create bar plot to visualize frequency of each digit



Data Preprocessing

9. Converting array to image, resize image, convert it to RGB and converting it back to array

```
[19]: from tensorflow.keras.preprocessing.image import load_img, img_to_array, array_to_img

def change_size(image):
    img = array_to_img(image, scale=False) #returns PIL Image
    img = img.resize((75, 75)) #resize image
    img = img.convert(mode='RGB') #makes 3 channels
    arr = img_to_array(img) #convert back to array
    return arr.astype(np.float64)

[20]: train_arr_75 = [change_size(img) for img in train_arr]
del train_arr
train_arr_75 = np.array(train_arr_75)
train_arr_75.shape

[20]: (60020, 75, 75, 3)
```

Data Transformation

10. Transform training and validation data using ImageDataGenerator

```
[21]: image_gen = ImageDataGenerator(rescale=1./255, #easier for network to interpret numbers in range [0,1]
                                    zoom_range=0.1,
                                    width_shift_range=0.2,
                                    height_shift_range=0.2,
                                    validation_split=0.2) # 80/20 train/val split

train_generator = image_gen.flow(train_arr_75,
                                 y,
                                 batch_size=32,
                                 shuffle=True,
                                 subset='training',
                                 seed=42)
valid_generator = image_gen.flow(train_arr_75,
                                 y,
                                 batch_size=16,
                                 shuffle=True,
                                 subset='validation')
del train_arr_75 #saves RAM
```

Modelling

11. Add following layers to Sequential model and compile model

```
[23]: model = Sequential()

model.add(tf.keras.applications.resnet50.ResNet50(input_shape = (75, 75, 3),
                                                    include_top = False,
                                                    weights = 'imagenet'))

model.add(L.Flatten())
model.add(L.Dense(128, activation='relu'))
model.add(L.Dense(10, activation='softmax'))

model.compile(optimizer=keras.optimizers.Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
#Do not use default learning rate since it is too high!
```

12. Train model

```
[25]: history = model.fit(train_generator, validation_data=valid_generator, epochs=5,
    steps_per_epoch=train_generator.n//train_generator.batch_size,
    validation_steps=valid_generator.n//valid_generator.batch_size)

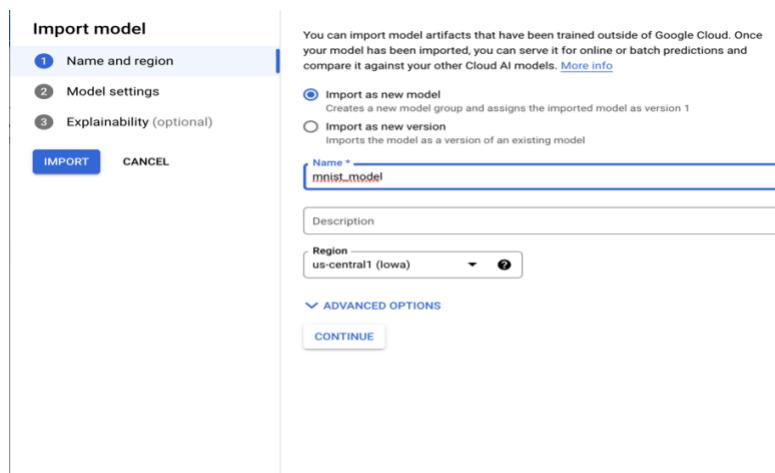
Epoch 1/5
1500/1500 [=====] - 2051s 1s/step - loss: 0.1566 - accuracy: 0.9540 - val_loss: 0.0722 - val_accuracy: 0.9801
Epoch 2/5
1500/1500 [=====] - 1889s 1s/step - loss: 0.0842 - accuracy: 0.9781 - val_loss: 0.2151 - val_accuracy: 0.9745
Epoch 3/5
1500/1500 [=====] - 1924s 1s/step - loss: 0.0512 - accuracy: 0.9857 - val_loss: 0.0289 - val_accuracy: 0.9925
Epoch 4/5
1500/1500 [=====] - 1980s 1s/step - loss: 0.0419 - accuracy: 0.9885 - val_loss: 0.0346 - val_accuracy: 0.9913
Epoch 5/5
1500/1500 [=====] - 1945s 1s/step - loss: 0.0470 - accuracy: 0.9876 - val_loss: 0.0412 - val_accuracy: 0.9908
```

13. Save trained model to our cloud bucket (cloud_assignment_model) which should be in same region as our notebook

```
[29]: from datetime import datetime
now = datetime.now()
current_time = now.strftime("%H.%M.%S")
tf.compat.v1.keras.experimental.export_saved_model(model, 'gs://cloud_assignment_4_model/keras-export'+current_time)
```

Create Endpoint and Deploy Model

14. On Vertex AI, go to model registry then click import and give name to model region should be same as cloud bucket for us-central1



15. Select TensorFlow version and path to cloud storage directory where model is stored then click import

Import model

- Name and region
- Model settings**
- Explainability (optional)

IMPORT CANCEL

Pre-built container settings

In order to run in a pre-built container, your code needs to be in Python 3.7.

Model framework * TensorFlow

Model framework version * 2.9

Accelerator type * None

Model artifact location (Cloud storage path) * gs://cloud_assignment_4/model/keras-export22.44.32 **BROWSE**

Path to the Cloud Storage directory where the exported model file is stored (not the path to the model file itself). The model name must be one of: saved_model.pb, model.pkl, model.joblib, or model.bst, depending on which library you used.

Predict schema

Optional. [Learn more about the predict schema](#)

Import model

- Name and region
- Model settings
- Explainability (optional)**

IMPORT CANCEL

Explainability options **LEARN**

In Vertex AI, models are made explainable through feature attribution, which tells you how much each feature contributed to the predicted result. You can use this information to verify that the model is behaving as expected, recognize bias in your models, and get ideas for ways to improve your model and your training data. Explainability will incur a minor additional cost. [Learn more](#)

Select a feature attribution method

Your model's data type determines which attribution methods are available to use. [Learn more about attribution methods](#)

None
 Sampled Shapley (for tabular models)
 Integrated gradients (for tabular models)
 Integrated gradients (for image classification models)
 XRAI (for image classification models)

Model Deployed

The screenshot shows the Google Cloud Vertex AI Model Registry interface. On the left, a sidebar lists options like Datasets, Labeling tasks, MODEL DEVELOPMENT (Training, Experiments, Metadata), and DEPLOY AND USE (Endpoints, Batch predictions, Matching Engine, Marketplace). The 'Model Registry' option is selected. The main area displays a table of models. One row is highlighted for 'mnist_model', which is listed under 'Name'. The 'Deployment status' column shows a green checkmark and the text 'Deployed on Vertex AI'. Other columns include 'Description' (empty), 'Default version' (1), 'Type' (Imported), 'Source' (Custom training), 'Updated' (Nov 19, 2022, 4:49:23 PM), and 'Labels' (empty). A 'Region' dropdown at the top is set to 'us-central1 (Iowa)'. A 'CREATE' and 'IMPORT' button are visible at the top right.

16. Once the model is deployed, click on model, and create new endpoint

This screenshot is similar to the previous one, showing the Vertex AI Model Registry. The 'mnist_model' row is selected. A context menu is open over this row, with the 'Edit labels' option highlighted. Other options in the menu include 'Edit name and description', 'Delete model', and 'Deploy to endpoint'.

17. Give name to endpoint

Deploy to endpoint

1 Define your endpoint 2 Model settings 3 Model monitoring

Endpoint name * [?](#)

Location

Region [?](#)

Access

Determines how your endpoint can be accessed. By default, endpoints are available for prediction serving through a REST API. Endpoint access can't be changed after the endpoint is created.

Standard
Makes the endpoint available for prediction serving through a REST API. AutoML and custom-trained models can be added to standard endpoints.

Private
Create a private connection to this endpoint using a VPC network and [private services access](#). Only custom-trained and tabular models can be added to private endpoints.
[Learn more](#)

ADVANCED OPTIONS

[CONTINUE](#)

18. Select Machine type from standard with 15 GB ram, keep everything default and click on deploy

Deploy to endpoint

1 Define your endpoint 2 Model settings 3 Model monitoring

Minimum number of compute nodes *

Once scaling settings are set, they can't be changed unless you redeploy the model. [Pricing guide](#)

Default is 1. If set to 1 or more, then compute resources will continuously run even without traffic demand. This can increase cost but avoid dropped requests due to node initialization.

Maximum number of compute nodes (optional)
Enter a number equal to or greater than the minimum nodes. Can reduce costs but may cause reliability issues for high traffic.

ADVANCED SCALING OPTIONS

Machine type * [?](#)

Accelerator type [?](#)

Accelerator count [?](#)

Service account [?](#)

A service account determines what Google Cloud resources your service code can access. By default, a Google-managed service account is used with permissions appropriate for most models. You can also use a user-managed service account to

Deployed Endpoint cloud_assignment_4

The screenshot shows the Google Cloud Vertex AI Endpoints page. On the left, there's a sidebar with options like Datasets, Labeling tasks, MODEL DEVELOPMENT (Training, Experiments, Metadata), DEPLOY AND USE (Model Registry, Endpoints, Batch predictions, Matching Engine, Marketplace). The 'Endpoints' option is selected. In the main area, there's a 'CREATE ENDPOINT' button and a search bar. Below it, a table lists endpoints. One row is highlighted for 'cloud_assignment_4'. The table columns include Name, ID, Status, Models, Region, Monitoring, Most recent alerts, Last updated, API, Notification, and Labels. The 'Region' dropdown is set to 'us-central1 (Iowa)'. A filter bar above the table allows entering a property name. The URL in the browser bar is <https://console.cloud.google.com/vertex-ai?authuser=2&project=logical-app-367922>.

Predict From Endpoint

19. Create new jupyter notebook, retrieve project id and endpoint id from sample request then specify endpoint link

This screenshot is similar to the previous one, showing the Vertex AI Endpoints page. However, a 'Sample Request' panel is open on the right side. The panel has tabs for 'REST' and 'PYTHON', with 'PYTHON' selected. It contains instructions and sample code for making predictions using Python. The code is as follows:

```
predict_custom_trained_model_sample(  
    project="591498791142",  
    endpoint_id="6694921903481552896",  
    location="us-central1",  
    instance_dict={ "instance_key_1": "value", ...}  
)
```

Below the code, there's a 'DONE' button.

The screenshot shows a Jupyter Notebook interface titled "managed-notebook-1668820099". The left sidebar displays a file tree with several notebooks and images. The main area contains a code cell [1] with the command `from google.cloud import aiplatform`. Below it, cell [2] shows the creation of a model endpoint. Cells [27] through [29] demonstrate image preprocessing: loading "six.jpg", converting it to a numpy array, and reshaping it to (28, 28). The notebook is running on a "TensorFlow 2 (Local)" kernel.

```
from google.cloud import aiplatform
model_endpoint = aiplatform.Endpoint('projects/591498791142/locations/us-central1/endpoints/6694921903481552896')

from PIL import Image
import numpy as np
img = Image.open('six.jpg').convert('L')
img = np.array(img)
img.shape

(28, 28)

test_arr = np.array(img).reshape(-1, 28, 28, 1)

test_arr.shape

(1, 28, 28, 1)
```

20. Upload handwritten image to workbench, do above mentioned preprocessing steps and give image to model to predict using endpoint

```
[31]: train_arr_75 = [change_size(img) for img in train_arr]
train_arr_75 = np.array(train_arr_75)
train_arr_75.shape

[31]: (1, 75, 75, 3)

• [32]: np.argmax(model_endpoint.predict(train_arr_75.tolist()).predictions)

[32]: 8
```

Predict using Cloud Function Serverless API

21. On cloud functions, create new function -> Give function name and make sure it is in same region as cloud bucket and for authentication select allow unauthenticated invocations

The screenshot shows the 'Create function' page in the Google Cloud console. The 'Configuration' tab is selected. In the 'Basics' section, the environment is set to '1st gen', the function name is 'function', and the region is 'us-central1'. A callout box highlights the 'function' input field. Below this, the 'Trigger' section shows an 'HTTP' trigger type selected, with 'HTTP' as the trigger type and the URL as 'https://us-central1-logical-app-367922.cloudfunctions.net/function'. At the bottom are 'NEXT' and 'CANCEL' buttons.

The screenshot shows the 'Create function' page in the Google Cloud console, continuing from the previous step. The 'Trigger' section remains the same. In the 'Authentication' section, the 'Allow unauthenticated invocations' checkbox is checked. Below it, the 'Require HTTPS' checkbox is also checked. At the bottom are 'SAVE' and 'CANCEL' buttons.

22. Allocate 1 GB memory from Runtime

The screenshot shows the 'Create function' page in the Google Cloud Platform. The 'Runtime, build, connections and security settings' section is open, showing:

- RUNTIME:** Memory allocated is set to 1 GB.
- BUILD:** Timeout is set to 60 seconds.
- CONNECTIONS:** Not visible in the screenshot.
- SECURITY AND:** Not visible in the screenshot.

Autoscaling: Minimum number of instances is 0, Maximum number of instances is 3000.

Runtime service account: Runtime service account is set to App Engine default service account.

At the bottom, there are 'NEXT' and 'CANCEL' buttons.

23. Select Python 3.10 for runtime and give entry point function name which will be triggered when we will hit our API

The screenshot shows the 'Create function' page in the Google Cloud Platform. The 'Configuration' tab is selected, showing:

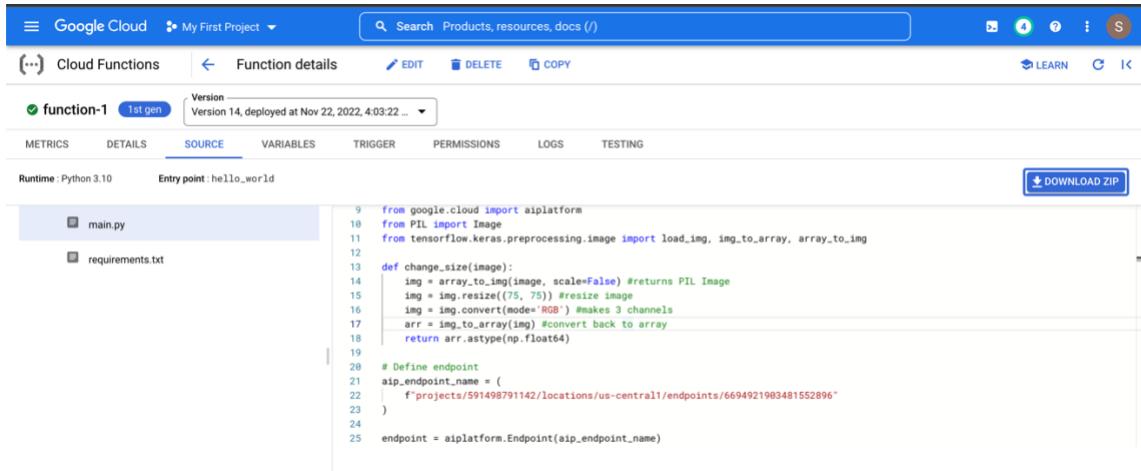
- Runtime:** Python 3.10.
- Source code:** Inline Editor.
- Entry point:** hello_world.

The code editor displays the following Python code:

```
Press Alt+F1 for Accessibility Options.
1 def hello_world(request):
2     """Responds to any HTTP request.
3     Args:
4         request (flask.Request): HTTP request object.
5     Returns:
6         The response text or any set of values that can be turned into a
7         Response object using
8         `make_response <http://flask.pocoo.org/docs/1.0/api/#flask.Flask.make_response>`_.
9     """
10    request_json = request.get_json()
11    if request.args and 'message' in request.args:
12        return request.args.get('message')
13    elif request_json and 'message' in request_json:
14        return request_json['message']
15    else:
16        return f'Hello World!'
```

At the bottom, there are 'PREVIOUS', 'DEPLOY', and 'CANCEL' buttons.

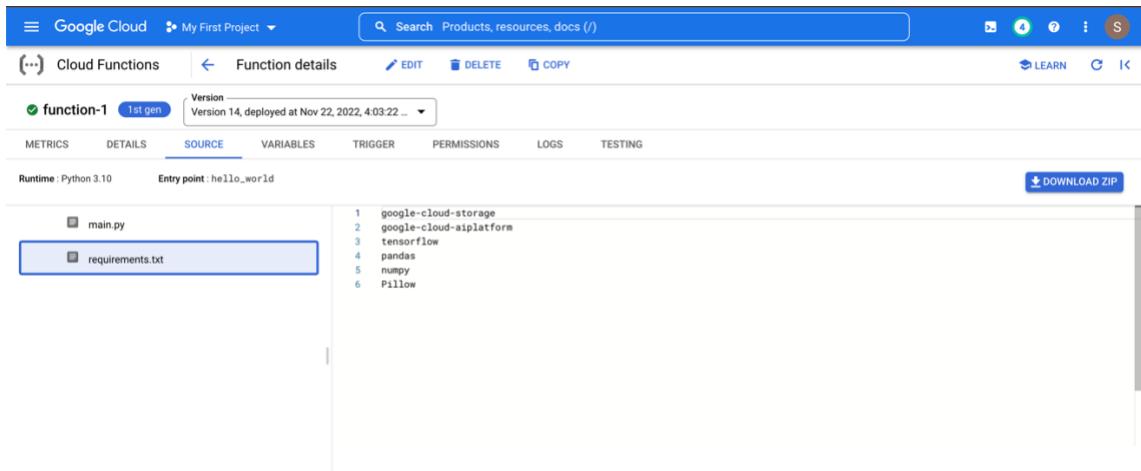
24. Mention required packages in requirements.txt



The screenshot shows the Google Cloud Functions interface for a project named "My First Project". A function named "function-1" is selected, showing its details. The "SOURCE" tab is active, displaying the code for main.py and requirements.txt. The main.py code includes imports for aiplatform, PIL, and tensorflow, along with a resize function and endpoint definition. The requirements.txt file lists dependencies: google-cloud-storage, google-cloud-aiplatform, tensorflow, pandas, numpy, and Pillow.

```
9 from google.cloud import aiplatform
10 from PIL import Image
11 from tensorflow.keras.preprocessing.image import load_img, img_to_array, array_to_img
12
13 def change_size(image):
14     img = array_to_img(image, scale=False) #returns PIL Image
15     img = img.resize((75, 75)) #resize image
16     img = img.convert(mode='RGB') #makes 3 channels
17     arr = img_to_array(img) #convert back to array
18     return arr.astype(np.float64)
19
20 # Define endpoint
21 aip_endpoint_name = (
22     f'projects/{PROJECT_ID}/locations/{LOCATION}/endpoints/{ENDPOINT_ID}'
23 )
24
25 endpoint = aiplatform.Endpoint(aip_endpoint_name)
```

```
1 google-cloud-storage
2 google-cloud-aiplatform
3 tensorflow
4 pandas
5 numpy
6 Pillow
```



This screenshot is identical to the one above, showing the Google Cloud Functions interface for "function-1". The "requirements.txt" file is highlighted with a blue selection bar, indicating it is the current file being edited or viewed.

25. Finally Deploy Cloud Function

Testing

26. For testing provide link of image in cloud bucket, for that I have made cloud bucket public
27. To make cloud bucket public, select bucket -> click on Permissions -> Add Principals -> Add allUsers -> Role -> Add Storage Object Viewer
28. There are **5 images** in bucket, anyone can be used for testing

The screenshot shows the Google Cloud Storage interface with the 'Cloud Storage' tab selected. Under 'Buckets', the 'cloud_assignment_4_model' bucket is highlighted. On the right, the 'Permissions' section is open, titled 'Grant access to "cloud_assignment_4_model"'. It shows '1 bucket selected' and '1 user' with 'cloud_assignment_4_model' checked. The 'Add principals' section has 'allUsers' selected. The 'Assign roles' section shows 'Storage Object Viewer' assigned with the condition 'Read access to GCS objects'. At the bottom are 'SAVE' and 'CANCEL' buttons.

Go to test and run API, provide url of image in cloud bucket

The screenshot shows the Google Cloud Functions interface with 'function-1' selected. The 'TESTING' tab is active. A code editor window displays a curl command to test the function:
curl -m 70 -X POST https://us-central1-logical-app-367922.cloudfunctions.net/function-1
-H "Authorization: bearer \$(gcloud auth print-identity-token)" \
-H "Content-Type: application/json" \
-d '{"url":"https://storage.googleapis.com/cloud_assignment_4_model/six.jpg"}'

The screenshot shows the Google Cloud Functions interface for a project named "My First Project". A specific function, "function-1", is selected. The "TESTING" tab is active, displaying a button labeled "(...) TEST THE FUNCTION". Below the button, a note states: "Testing in the Cloud Console has a 5 minute timeout. Note that this is different from the limit set in the function configuration." At the bottom of the testing section, there is an "Output" panel showing "Complete" status with a log entry: "S 8". Below the output panel is a "Logs" section with a note: "Logs Fetched (up to 100 entries). View all logs".

The screenshot shows the Google Cloud Storage interface for a project named "My First Project". A bucket named "cloud_assignment_4_model" is selected. The "OBJECTS" tab is active, showing a list of objects. The table includes columns for Name, Size, Type, Created, Storage class, Last modified, and Public access. The objects listed are: "eight.jpg", "eight_1.jpg", "keras-export22.44.32/", "nine.jpg", "one.jpg", and "six.jpg". Each object has a download icon and a more options menu icon.

Name	Size	Type	Created	Storage class	Last modified	Public access
eight.jpg	3.6 KB	image/jpeg	Nov 22, 2022, 7:48:16 PM	Standard	Nov 22, 2022, 7:48:16 PM	Public to internet
eight_1.jpg	3.6 KB	image/jpeg	Nov 22, 2022, 7:48:26 PM	Standard	Nov 22, 2022, 7:48:26 PM	Public to internet
keras-export22.44.32/	—	Folder	—	—	—	—
nine.jpg	3.5 KB	image/jpeg	Nov 22, 2022, 7:49:25 PM	Standard	Nov 22, 2022, 7:49:25 PM	Public to internet
one.jpg	3.2 KB	image/jpeg	Nov 22, 2022, 7:48:01 PM	Standard	Nov 22, 2022, 7:48:01 PM	Public to internet
six.jpg	3.6 KB	image/jpeg	Nov 21, 2022, 10:02:19 AM	Standard	Nov 21, 2022, 10:02:19 AM	Public to internet

Cloud bucket link:

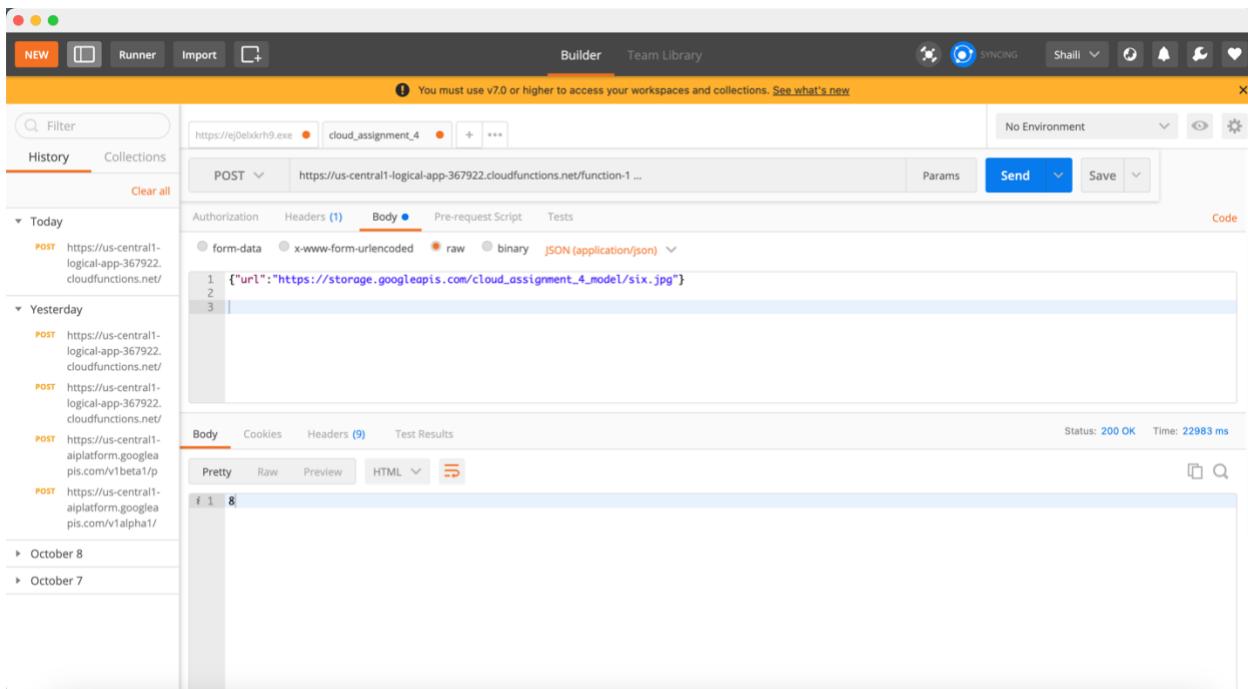
https://console.cloud.google.com/storage/browser/cloud_assignment_4_model;tab=objects?forceOnBucketsSortingFiltering=false&authuser=0&project=logical-app-367922&prefix=&forceOnObjectsSortingFiltering=false

Testing Using Postman

Link: <https://us-central1-logical-app-367922.cloudfunctions.net/function-1>

json input

```
{"url":"https://storage.googleapis.com/cloud_assignment_4_model/six.jpg"}
```

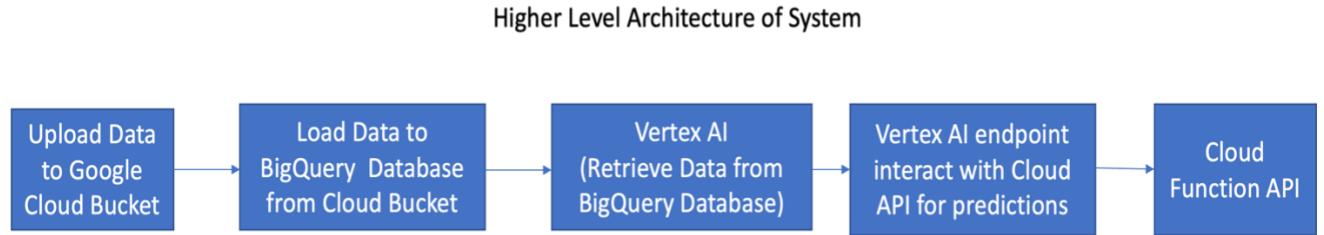


Testing using Curl request

```
curl -m 70 -X POST https://us-central1-logical-app-367922.cloudfunctions.net/function-1 \
-H "Authorization: bearer $(gcloud auth print-identity-token)" \
-H "Content-Type: application/json" \
-d '{"url":"https://storage.googleapis.com/cloud_assignment_4_model/six.jpg"}'
```

```
(base) shailidave@Shailis-MacBook-Air ~ % curl -m 70 -X POST https://us-central1-logical-app-367922.cloudfunctions.net/function-1 \
-H "Authorization: bearer $(gcloud auth print-identity-token)" \
-H "Content-Type: application/json" \
-d '{"url":"https://storage.googleapis.com/cloud_assignment_4_model/six.jpg"}'
zsh: command not found: gcloud
[8]
(base) shailidave@Shailis-MacBook-Air ~ % curl -m 70 -X POST https://us-central1-logical-app-367922.cloudfunctions.net/function-1 \
-H "Content-Type: application/json" \
-d '{"url":"https://storage.googleapis.com/cloud_assignment_4_model/six.jpg"}'
[8]
```

Architecture Diagram



Limitations

- ⇒ Model requires improvement as current model does not give correct predictions every time

References

- [1] https://www.youtube.com/watch?v=-9fU1xwBQYU&ab_channel=GoogleCloudTech
- [2] <https://stackoverflow.com/questions/73141754/vertex-ai-endpoint-call-with-json-invalid-json-payload-received>
- [3] <https://cloud.google.com/vertex-ai/docs/tutorials/image-recognition-custom/serving>
- [4] <https://towardsdatascience.com/machine-learning-model-as-a-serverless-endpoint-using-google-cloud-function-a5ad1080a59e>
- [5] <https://cloud.google.com/bigquery/docs/introduction>
- [6] <https://stackoverflow.com/questions/7391945/how-do-i-read-image-data-from-a-url-in-python>
- [7] <https://stackoverflow.com/questions/70773518/fail-to-install-module-pil-error-could-not-find-a-version-that-satisfies-the>
- [8] <https://cloud.google.com/functions/docs/writing>
- [9] <https://www.kaggle.com/code/saumandas/intro-to-transfer-learning-with-mnist>
- [10] <https://www.analyticsvidhya.com/blog/2021/05/transfer-learning-using-mnist/>