Image Classification Model

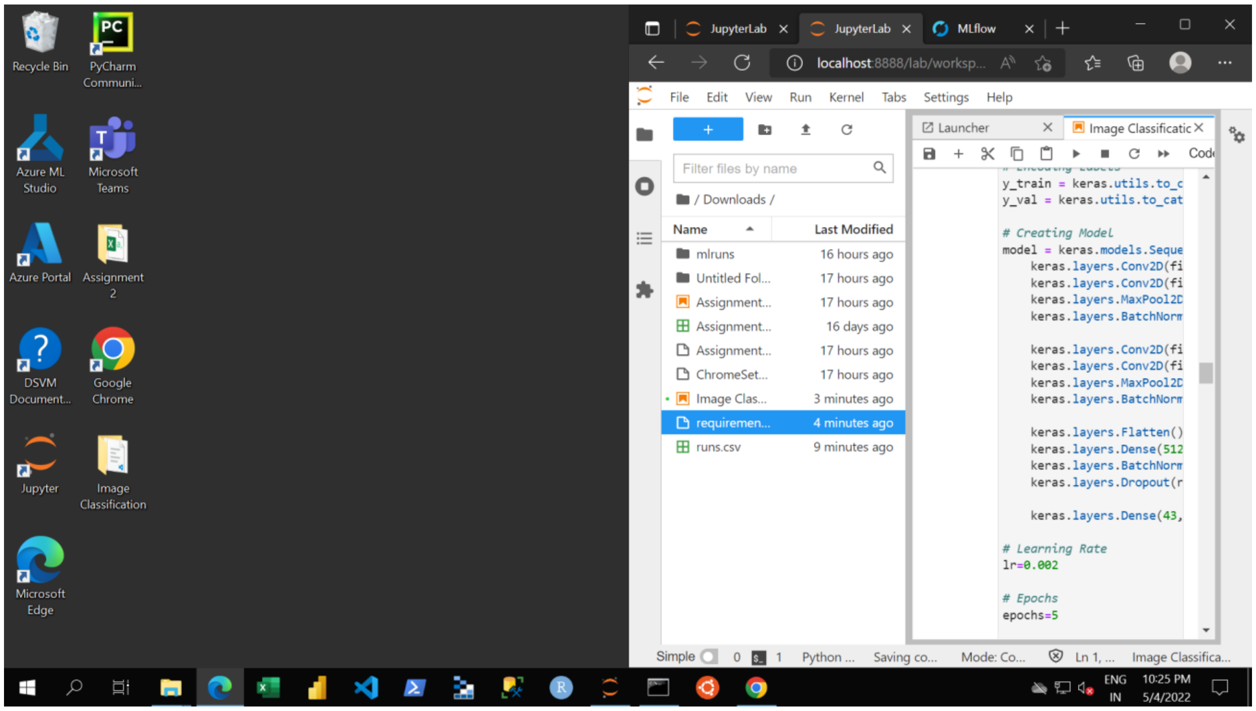
Description

Image classification is a very common problem today especially in the evolving field of self-driving cars. A Deep Learning ML model is used to classify the traffic sign images.

Getting Started

Dependencies

- Python 3
- Scikit-Learn
- Pandas
- Numpy
- TensorFlow
- OpenCV
- Current Web Browser
- Jupyter Notebook
- MLflow
- Azure/ AWS Cloud
- Virtual Machine Azure



Name	Last Modified
mlruns	16 hours ago
Untitled Fol...	17 hours ago
Assignment...	17 hours ago
Assignment...	16 days ago
Assignment...	17 hours ago
ChromeSet...	17 hours ago
Image Clas...	3 minutes ago
requiremen...	4 minutes ago
runs.csv	9 minutes ago

```
y_train = keras.utils.to_c
y_val = keras.utils.to_cat

# Creating Model
model = keras.models.Seque
    keras.layers.Conv2D(fi
    keras.layers.Conv2D(fi
    keras.layers.MaxPool2D
    keras.layers.BatchNorm

    keras.layers.Conv2D(fi
    keras.layers.Conv2D(fi
    keras.layers.MaxPool2D
    keras.layers.BatchNorm

    keras.layers.Flatten()
    keras.layers.Dense(512
    keras.layers.BatchNorm
    keras.layers.Dropout(r

    keras.layers.Dense(43,

# Learning Rate
lr=0.002

# Epochs
epochs=5
```

Installing

- www.anaconda.com/products/individual (Python 3, Scikit-Learn & Jupyter Notebook)
- pandas.pydata.org/pandas-docs/stable/getting_started/install.html
- www.mlflow.org/docs/latest/quickstart.html#installing-mlflow

Executing program

To Start MLflow, go to anaconda prompt and execute:

1) Start MLflow server:

```
mlflow server --backend-store-uri 'C:/temp/mlflow/localserver'
```

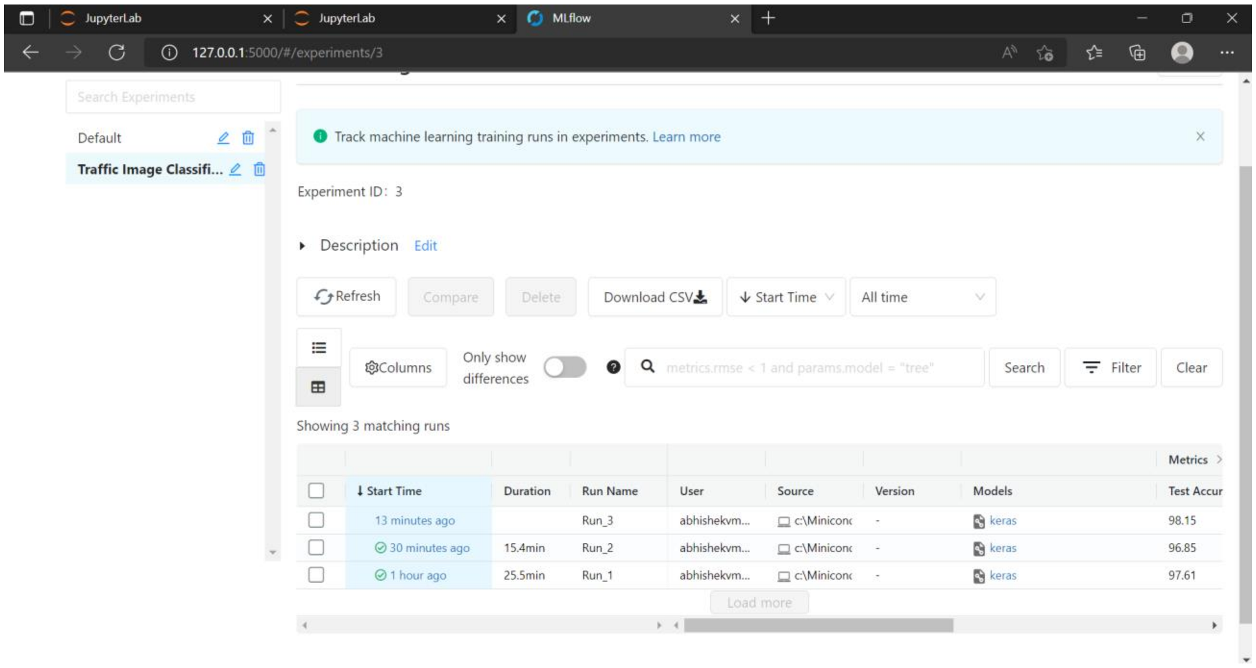
2) Open browser and go to http://localhost:5000 for MLflow UI

Once MLflow is running, open a new anaconda prompt and execute:

```
Jupyter Notebook
```

The above url will change as every virtual machine will take you to a different address. In our case, it was http://127.0.0.1:5000.

Expected Results



Track machine learning training runs in experiments. Learn more

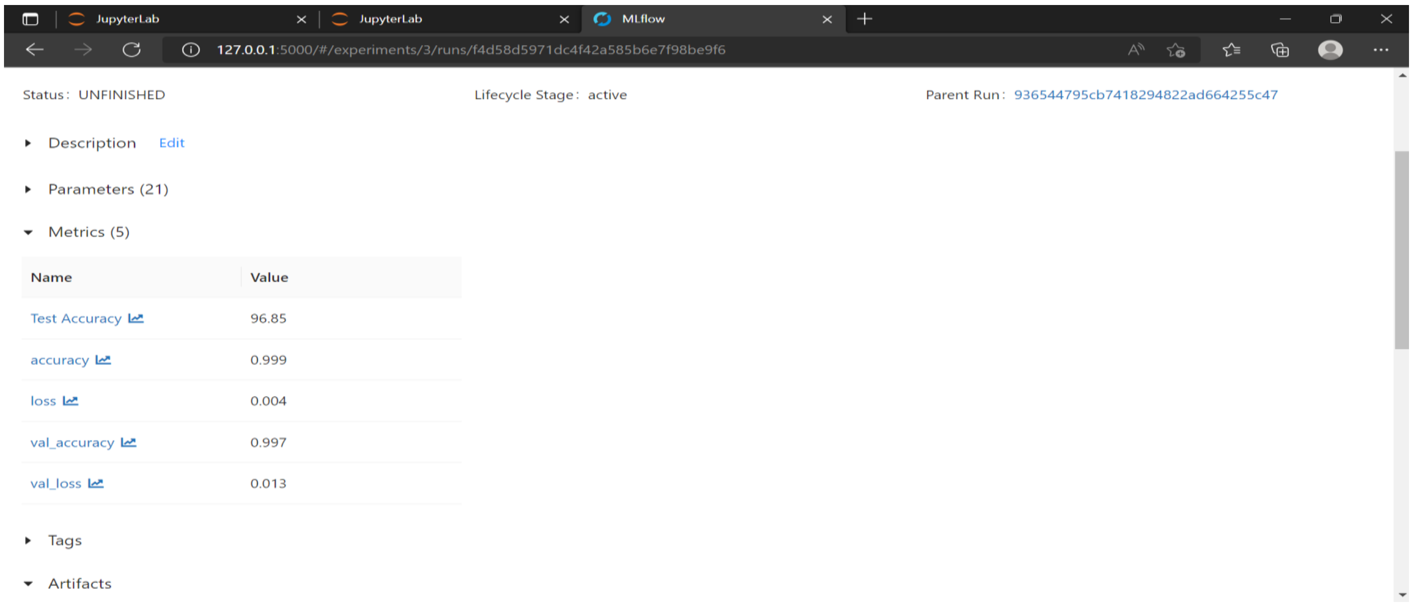
Experiment ID: 3

Description Edit

Showing 3 matching runs

							Metrics
Start Time	Duration	Run Name	User	Source	Version	Models	Test Accur
13 minutes ago		Run_3	abhishekvm...	c:\Minicon	-	keras	98.15
30 minutes ago	15.4min	Run_2	abhishekvm...	c:\Minicon	-	keras	96.85
1 hour ago	25.5min	Run_1	abhishekvm...	c:\Minicon	-	keras	97.61

Run the first experiment, and go to http://localhost:5000 and view the results in MLflow. After running all code blocks, you should see three runs with green checkmarks.


JupyterLab
JupyterLab
MLflow
127.0.0.1:5000/#/experiments/3/runs/f4d58d5971dc4f42a585b6e7f98be9f6

Status: UNFINISHED

Lifecycle Stage: active

Parent Run: 936544795cb7418294822ad664255c47

Description Edit

Parameters (21)

Metrics (5)

Name	Value
Test Accuracy	96.85
accuracy	0.999
loss	0.004
val_accuracy	0.997
val_loss	0.013

Tags

Artifacts

TODO

- Set up cloud environment to run TensorFlow on Azure/ AWS
- Upload your dataset to your cloud-based ML environment (Azure Blob or S3)
- Perform exploratory data analysis (EDA)
- Create a deep learning model to predict image class using TensorFlow
- Use MLFlow to track the experiment, log artifacts and model

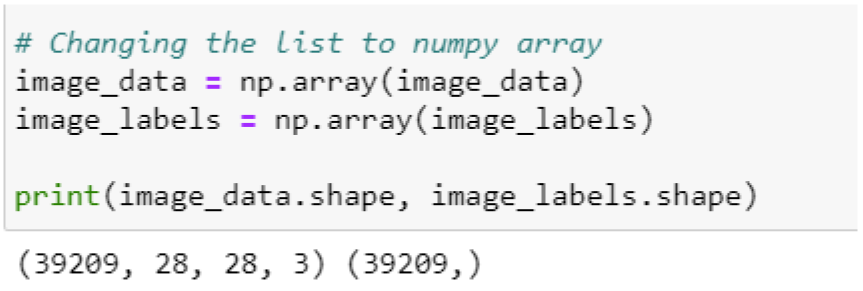
Authors

Abhishek Mote, Anurag Yadav

Approach:

- After the MLFlow was set, we started our experiment with assigning the path as we had csv as well as the image file as well.
- We provided the path directly to the notebook to access the entire dataset.
- Provided with labels already, so no need to generate separately, we jumped to data visualization to check how the data looks like,

- Our next step was to check how the testing data looks like, so we checked random data of 25 images.
-
- The approach should be to provide the path of directly, the traditional method won't work here.
- In addition, we didn't store the data into S3 or blob, instead, we created the virtual machine, downloaded the data into the VM, provided the path to the notebook.
- Next step is to convert the image data into array, in order to run the algorithm.
- The data will look like this



```
# Changing the list to numpy array
image_data = np.array(image_data)
image_labels = np.array(image_labels)

print(image_data.shape, image_labels.shape)
```

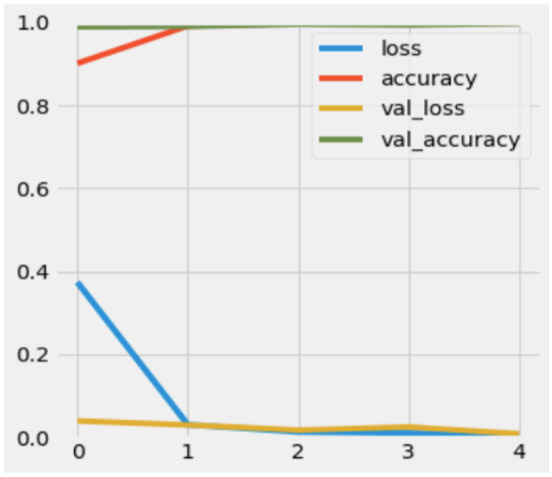
```
(39209, 28, 28, 3) (39209,)
```

- Then comes the modeling part, splitting the data into training and testing data set.
- We used keras.model.sequential to initiate the model, used 'relu' and 'softmax' as the activation functions, adam as optimizer and loss function as categorical cross entropy.

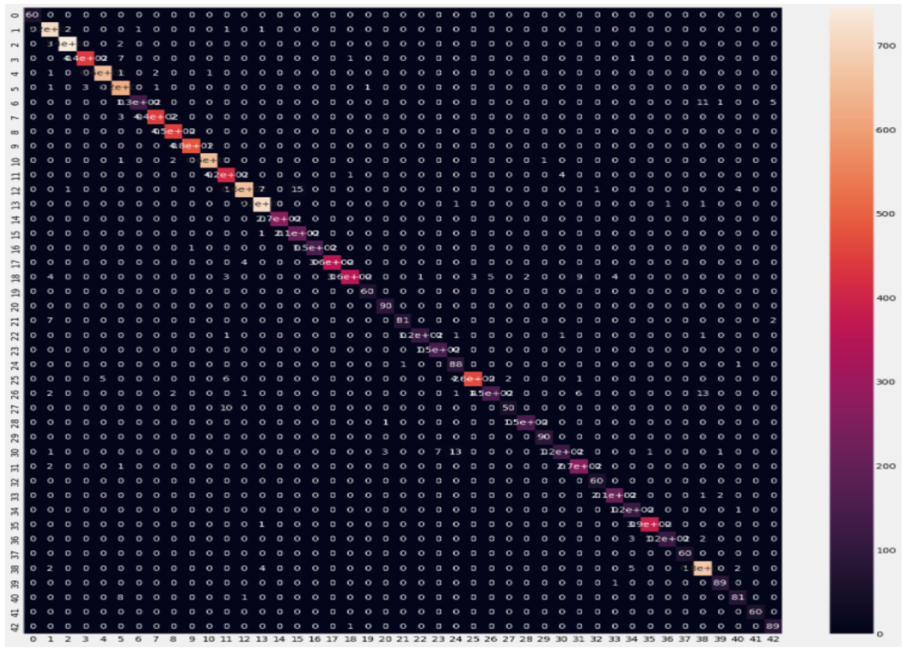
- We ran the experiment with 5 epochs.

```
Epoch 1/5
858/858 [==============================] - 59s 67ms/step - loss: 0.3739 - accuracy: 0.9011 - val_loss: 0.0400 - val_accuracy:
0.9888
Epoch 2/5
858/858 [==============================] - 56s 66ms/step - loss: 0.0309 - accuracy: 0.9913 - val_loss: 0.0303 - val_accuracy:
0.9901
Epoch 3/5
858/858 [==============================] - 54s 63ms/step - loss: 0.0136 - accuracy: 0.9965 - val_loss: 0.0178 - val_accuracy:
0.9953
Epoch 4/5
858/858 [==============================] - 54s 63ms/step - loss: 0.0107 - accuracy: 0.9968 - val_loss: 0.0253 - val_accuracy:
0.9935
Epoch 5/5
858/858 [==============================] - 55s 64ms/step - loss: 0.0055 - accuracy: 0.9986 - val_loss: 0.0090 - val_accuracy:
0.9972
```

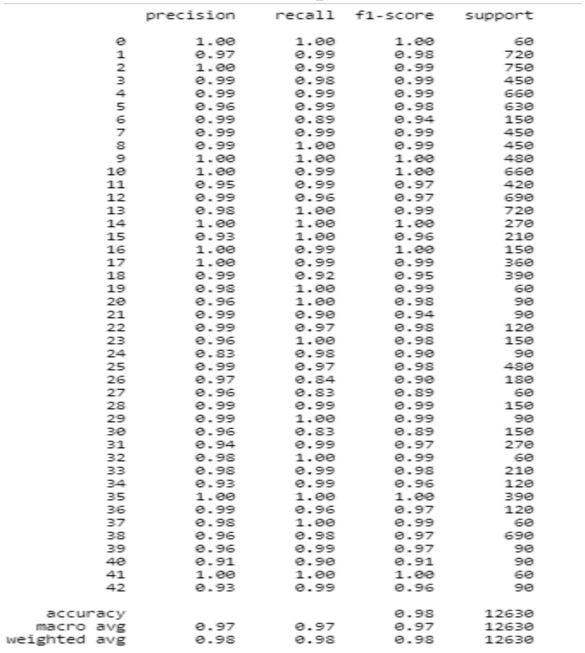
- Our model ran successfully maintaining the accuracy of 99% throughout the epochs.
- We kept 5 epochs because of the performance limitations of the VM. The plot captured the details of the experiment



- To check the model performance, we calculated the confusion matrix.



- And the classification report is as follows:

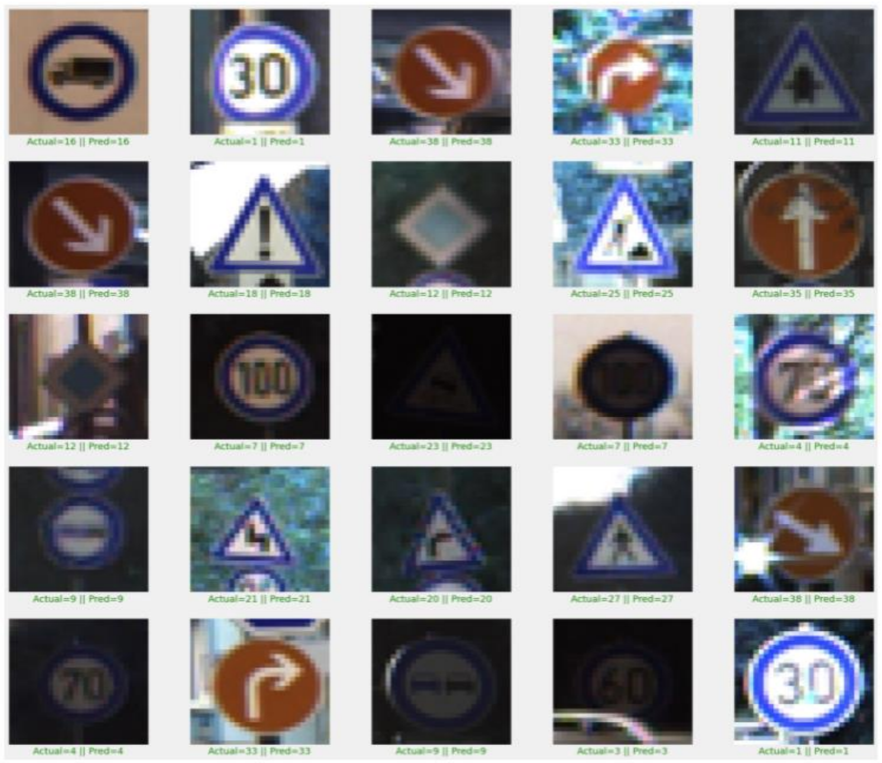


```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        60
           1       0.97      0.99      0.98       720
           2       1.00      0.99      0.99       750
           3       0.99      0.98      0.99       450
           4       0.99      0.99      0.99       660
           5       0.96      0.99      0.98       630
           6       0.99      0.89      0.94       150
           7       0.99      0.99      0.99       450
           8       0.99      1.00      0.99       450
           9       1.00      1.00      1.00       480
          10       1.00      0.99      1.00       660
          11       0.95      0.99      0.97       420
          12       0.99      0.96      0.97       690
          13       0.98      1.00      0.99       720
          14       1.00      1.00      1.00       270
          15       0.93      1.00      0.96       210
          16       1.00      0.99      1.00       150
          17       1.00      0.99      0.99       360
          18       0.99      0.92      0.95       390
          19       0.98      1.00      0.99        60
          20       0.96      1.00      0.98        90
          21       0.99      0.90      0.94        90
          22       0.99      0.97      0.98       120
          23       0.96      1.00      0.98       150
          24       0.83      0.98      0.90        90
          25       0.99      0.97      0.98       480
          26       0.97      0.84      0.90       180
          27       0.96      0.83      0.89        60
          28       0.99      0.99      0.99       150
          29       0.99      1.00      0.99        90
          30       0.96      0.83      0.89       150
          31       0.94      0.99      0.97       270
          32       0.98      1.00      0.99        60
          33       0.98      0.99      0.98       210
          34       0.93      0.99      0.96       120
          35       1.00      1.00      1.00       390
          36       0.99      0.96      0.97       120
          37       0.98      1.00      0.99        60
          38       0.96      0.98      0.97       690
          39       0.96      0.99      0.97        90
          40       0.91      0.90      0.91        90
          41       1.00      1.00      1.00        60
          42       0.93      0.99      0.96        90

    accuracy                           0.98     12630
   macro avg       0.97      0.97      0.97     12630
weighted avg       0.98      0.98      0.98     12630
```

- And to conclude, how our experiment worked, we checked the predicted images vs the actual images.



-
- The model predicted the images accurately, as it can be seen that actual = = predicted can be seen under each image.

Explain the performance differences and advantages of running TensorFlow on cloud platform vs running it on a local machine

The performance differences

- A virtual machine needs to be setup initially in order to run the experiment
- Virtual space like Blob or S3 needs to be purchased to store the data
- Both of these involves cost.
- VM comes with a limitation that it misses out on some of the libraries that needs to be pip installed before running.
- Azure or AWS, has the older version of python, that is 3.6 however the latest version is 3.10, so that needs to be updated in order to run few libraries.
- These problems are not associated while working in local machine as any update can be done through conda cmd and pip install.
- Also, can't keep VM machine running for a longer time as it may cost you a lot based on the usage whereas the local machine is free to use as long as we want.

The Advantages

- We can use virtual machine practically on any laptop or desktop, as it's a cloud service, we just need to connect it with Azure or AWS.
- Easy to install and create a VM with documentation
- Very fast when we provide multi-CPU's
- The model runs efficiently without affecting the local machine.
- Cost is as per the use, and not on the subscription model. So you need to pay for the time you used the machine, which is mostly hourly based and can be as low as $0.20 per hour.