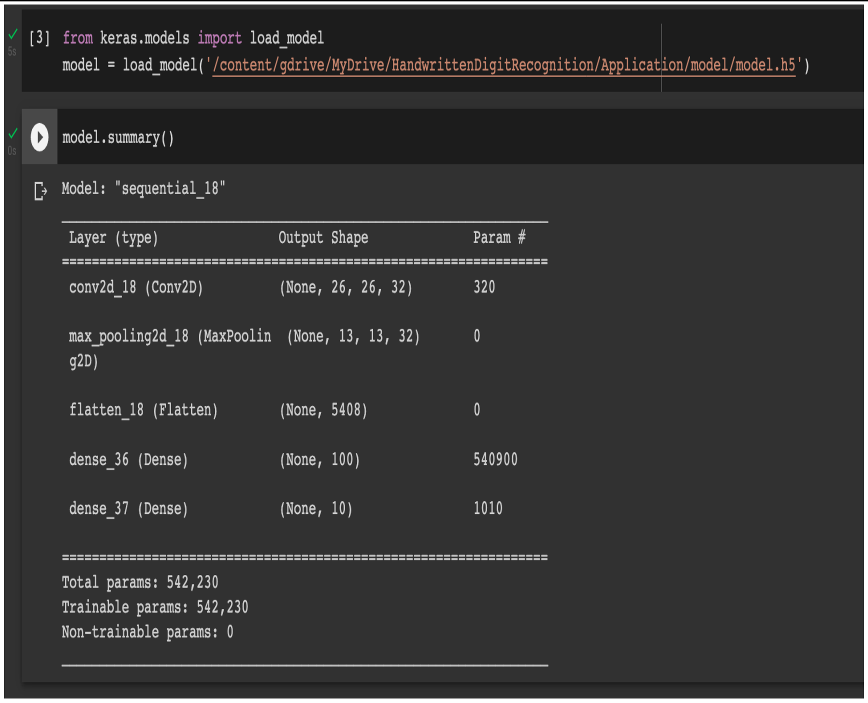
**Project Development Phase**
**Model Performance Test**

| | |
|---|---|
| Date | 10 November 2022 |
| Team ID | PNT2022TMID00805 |
| Project Name | Project - A Novel Method for Handwritten Digit Application |
| Maximum Marks | 10 Marks |

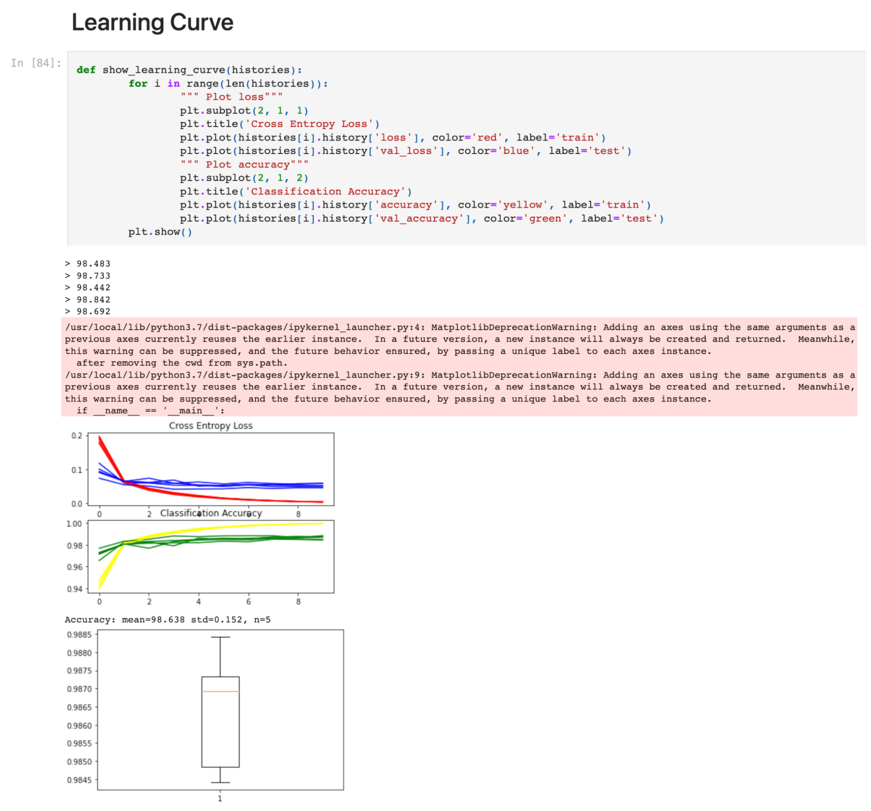
## Model Performance Testing:

| S.No. | Parameter | Values | Screenshot |
|---|---|---|---|
| 1. | Model Summary | **model.summary()** |  |

```
[3] from keras.models import load_model
    model = load_model('/content/gdrive/MyDrive/HandwrittenDigitRecognition/Application/model/model.h5')
```

```
model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_18 (MaxPoolin  (None, 13, 13, 32)       0
 g2D)

 flatten_18 (Flatten)        (None, 5408)              0

 dense_36 (Dense)            (None, 100)               540900

 dense_37 (Dense)            (None, 10)                1010

=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

|  |  |  |  |
|---|---|---|---|
| 2. | Accuracy | Training Accuracy -<br><br>Validation Accuracy - | 98.638<br> |

Learning Curve

```
In [84]: def show_learning_curve(histories):
                for i in range(len(histories)):
                        """ Plot loss"""
                        plt.subplot(2, 1, 1)
                        plt.title('Cross Entropy Loss')
                        plt.plot(histories[i].history['loss'], color='red', label='train')
                        plt.plot(histories[i].history['val_loss'], color='blue', label='test')
                        """ Plot accuracy"""
                        plt.subplot(2, 1, 2)
                        plt.title('Classification Accuracy')
                        plt.plot(histories[i].history['accuracy'], color='yellow', label='train')
                        plt.plot(histories[i].history['val_accuracy'], color='green', label='test')
                plt.show()
```

```
> 98.483
> 98.733
> 98.442
> 98.842
> 98.692
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a
previous axes currently reuses the earlier instance.  In a future version, a new instance will always be created and returned.  Meanwhile,
this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a
previous axes currently reuses the earlier instance.  In a future version, a new instance will always be created and returned.  Meanwhile,
this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  if __name__ == '__main__':
```

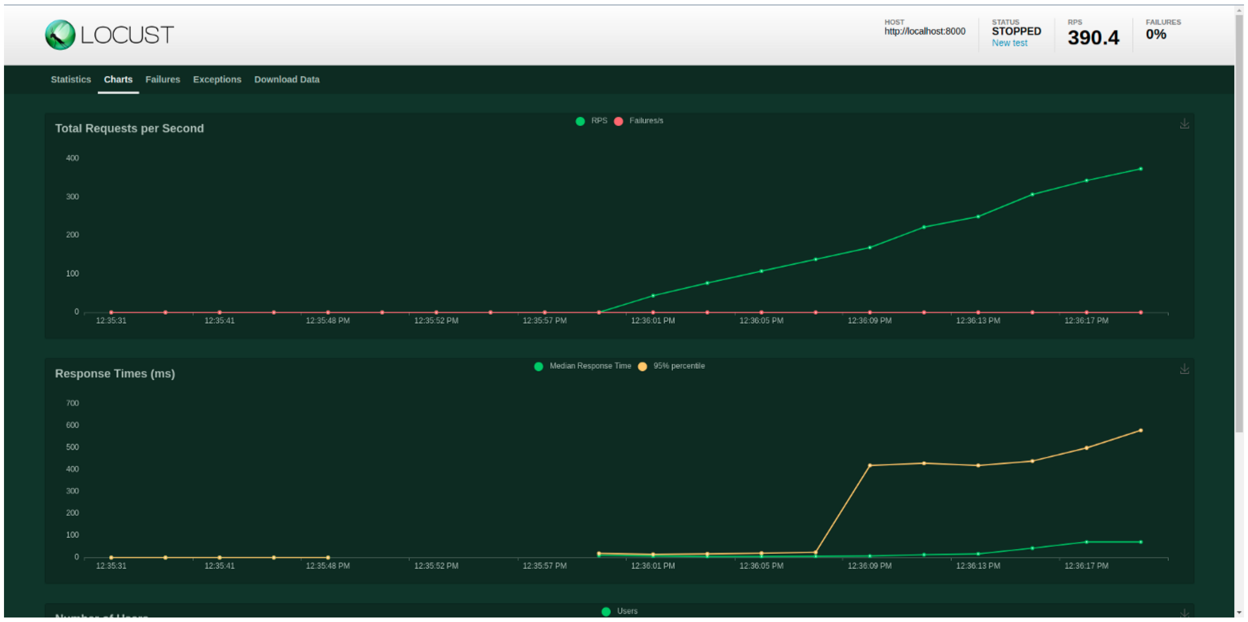
```
Accuracy: mean=98.638 std=0.152, n=5
```

## Performance Testing

It was done with locust.py

### locust.py

```
from locust
import HttpUser,
task, between
import os


def
getImg(filePath)
:
     _fileName =
os.path.basename
(filePath)

_fileContent =
open(filePath,'r
b')
     return
_fileName,
_fileContent,
'file'

_files= {

"Images":getImg(
"../Test
Images/2.jpeg"),

"Images":getImg(
"../Test
Images/3.jpeg"),
}
headers =
{'content-type':
'multipart/form-
data'}

class
PerformanceTest(
HttpUser):

   @task
   def
home(self):
        r =
self.client.post
("",
files=_files,
headers =
headers)
        print(r)
```


LOCUST
HOST
http://localhost:8000
STATUS
STOPPED
New test
RPS
390.4
FAILURES
0%
Statistics
Charts
Failures
Exceptions
Download Data
Total Requests per Second
● RPS ● Failures/s
400
300
200
100
0
12:35:31
12:35:41
12:35:48 PM
12:35:52 PM
12:35:57 PM
12:36:01 PM
12:36:05 PM
12:36:09 PM
12:36:13 PM
12:36:17 PM
Response Times (ms)
● Median Response Time ● 95% percentile
700
600
500
400
300
200
100
0
12:35:31
12:35:41
12:35:48 PM
12:35:52 PM
12:35:57 PM
12:36:01 PM
12:36:05 PM
12:36:09 PM
12:36:13 PM
12:36:17 PM
● Users