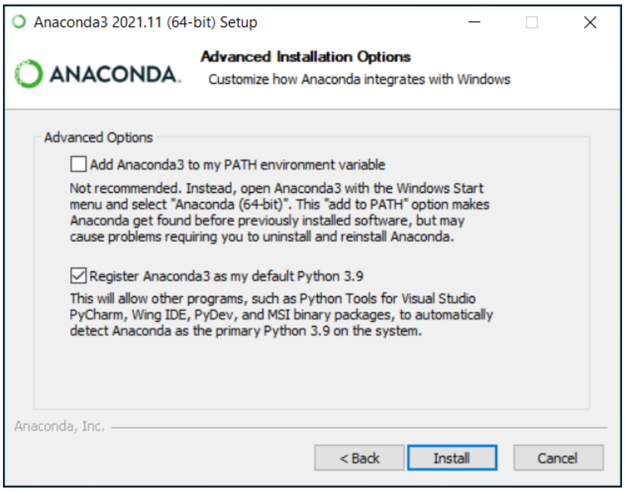
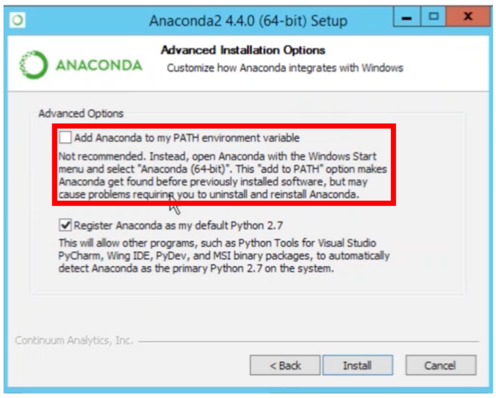
Experiment 1

Aim: Setting up the Jupyter IDE Environment and Executing a Python Program

Procedure to Install Anaconda :

1. Go to the Anaconda Website and chose a Python 3.x graphical installer
2. Locate your download and double click it.
3. Read the license agreement and click on I Agree.
4. Note your installation location and then click Next
5. This is an important part of the installation process. The recommended approach is to not check the box to add Anaconda to your path. This means you will have to use Anaconda Navigator or the Anaconda Command Prompt when you wish to use Anaconda. If you want to be able to use Anaconda in your command prompt please use the alternative approach and check the box.





6. This is an optional step. This is for the case where you didn't check the box in step 5 and now want to add Anaconda to your path in the environment variables

Integrating Jupyter with Anaconda:

7. Find and open the Anaconda Prompt app using the search bar.
8. Once the Anaconda Prompt app opens, navigate to the desired folder, using the cd command.
9. Once in the desired folder, type Jupyter notebook followed by the Enter key.
10. The Jupyter server will start. You should see some server logs printed. You may be prompted to select an application to open Jupyter in. Firefox or Chrome are preferred.
11. Shortly after, a browser window should open, showing the files and folders located in the folder where you started the Jupyter server.

Executing a Python Program:

```
n=int(input("Emter a number"))
s=0
r=0
t=n
while n>0:
  r=n%10
  s=s*10 +r
  n=n//10
if(s==t):
  print("Palindrome")
else:
  print("Not Palindrome")
```

```
Emter a number121
Palindrome
```

Result: We have successfully installed Anaconda and set up the Jupyter IDE and have executed a Python program to check whether an input number is Palindrome or not.

Experiment 2

Aim: Installing Tensor flow and PyTorch Libraries and make use of them

Procedure to install Tensor-flow in Anaconda:

Tensor-flow with conda is supported on 64-bit Windows 7 or later, 64-bit Ubuntu, Linux 14.04 or later, 64 bit CentOS Linux 6 or later and macOs 10.10 or later.

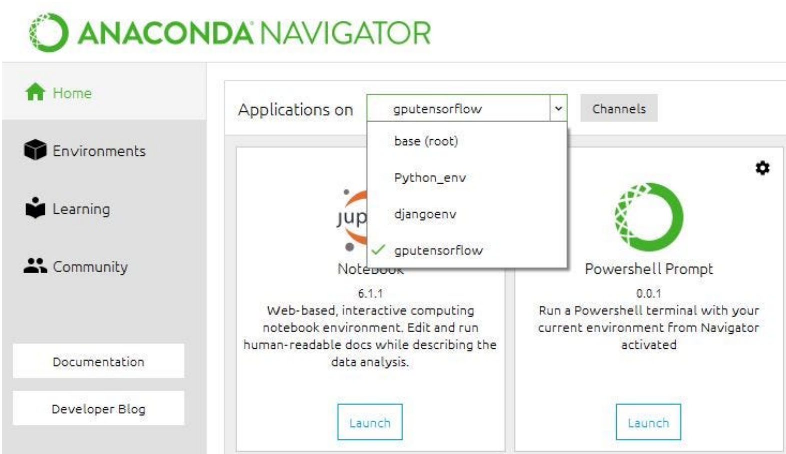
1. On Windows open the Start menu and open Anaconda Command Prompt.
2. Choose a name for your TensorFlow environment, such as “tf"
3. To install the current release of CPU-only TensorFlow, recommended for beginners.

   conda create -n tf tensorflow

   conda activate tf
4. Or, to install the current release of GPU TensorFlow on Linux or conda create Windows:

   conda create-n tf-gpu tensorflow-gpu

   conda activate tf-gpu
5. Now go to Anaconda Navigator and change the environment to tf-gpu from base.



6. Install Jupyter notebook and launch Jupyter in the new environment
7. Install numpy using pip install numpy==1.23.4

8. Now import tensorflow and check the version

```
import tensorflow as tf
print(tf.__version__)

2.6.0
```

9. Check the keras version using the following command

```
!pip show keras

Name: keras
Version: 2.13.1
Summary: Deep learning for humans.
Home -page: https://keras.io/
Author: Keras team
Author-email: keras -users@googlegroups.com
License: Apache 2.0
Location: c:\users\mgit\anaconda3\envs\tf- gpu\lib\site-packages
Requires:
Required-by: tensorflow
```

Example program for Tensorflow basics

```
import tensorflow as tf
x= tf.constant ([[1., 2., 3.],[4., 5., 6.]])
print(x)
print(x.shape)
print (x.dtype)
```

Output:

```
tf.Tensor([[1., 2., 3.][4., 5., 6.]],
shape=(2, 3), dtype=float32)
(2,3)
<dtype: 'float32'>
```

Installing Pytorch and importing it in Jupyter notebook

1. Use the command pip3 instal torch torch vision torch audio in anaconda command prompt to install pytorch.
2. Now import torch in Jupyter notebook
3. Write an example program in Jupyter

```
import torch
x = torch.rand(5, 3)
print (x)
```

Output:

```
tensor([[0.8338, 0.2921, 0.2501],
[0.8172, .9531, 0.9061],
[0.4925, .0952, 0.3532],
[0.3888, 0.7118, 0.3312],
[0.4027, 0.3560, 0.8726]])
```

Result: We have successfully installed Tensorflow and Keras and executed simple programs

Experiment 3

Aim: Applying Convolutional Neural Network on Computer Vision Algorithms

Importing The Libraries

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from tensorflow.keras.optimizers import Adam
```

Loading and Resizing the training datasets of dogs and cats

```
import PIL
import os
import os.path
from PIL import Image
f= r'C:\Users MGIT\Desktop\cd dataset\train\dog’
for file in os.listdir(f):
    f_img= f+ “/" +file
    img=Image.open(f_img)
    img=img.resize((112, 112))
    img.save(f_img)

import PIL
import os
import os.path
from PIL import Image
f= r'C:\Users MGIT\Desktop\cd dataset\train\cat’
```

```
for file in os.listdir(f):
        f_img= f+ "/" +file
        img=Image.open(f_img)
        img=img.resize((112, 112))
        img.save(f_img)
```

Loading and Resizing the testing datasets of dogs and cats

```
import PIL
import os
import os.path
from PIL import Image
f= r'C:\Users MGIT\Desktop\cd dataset\test\dog'
for file in os.listdir(f):
        f_img= f+ "/" +file
        img=Image.open(f_img)
        img=img.resize((112, 112))
        img.save(f_img)


import PIL
import os
import os.path
from PIL import Image
f= r'C:\Users MGIT\Desktop\cd dataset\test\cat'
for file in os.listdir(f):
        f_img= f+ "/" +file
        img=Image.open(f_img)
        img=img.resize((112, 112))
        img.save(f_img)
```

Image Preprocessing

```
IMAGE_SIZE = 112
BATCH_SIZE= 32
train_data_size = 180
test_data = 20

train= tf.keras.preprocessing.image. ImageDataGenerator(rescale=1./255, rotation_range = 90,
shear_range =0.2, zoom_range = 0.2, horizontal_flip = True,)
```

Output: Found 180 images belonging to 2 classes

```
test= tf.keras.preprocessing.image. ImageDataGenerator(rescale=1./255, rotation_range = 90,
shear_range =0.2, zoom_range = 0.2, horizontal_flip = True,)
```

Output: Found 20 images belonging to 2 classes

Model Building

```
model= Sequential([
        Conv2D(32,(3,3),activation='relu', input_shape=(112,112,3)),
        MaxPool2D(2,2),
        Conv2D(32,(3,3),activation='relu' ,input_shape=(112, 112, 3)),
        MaxPool2D(2,2),
        Flatten(),
        Dense(100, activation='relu' ),
        Dense(1, activation='sigmoid')
]
)

model.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 110, 110, 32)	896
max_pooling2d (MaxPooling2D)	(None, 55, 55, 32)	0
conv2d_1 (Conv2D)	(None, 53, 53, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 26, 26, 32)	0
flatten (Flatten)	(None, 21632)	0
dense (Dense)	(None, 100)	2,163,300
dense_1 (Dense)	(None, 1)	101

```
Total params: 2,173,545 (8.29 MB)

Trainable params: 2,173,545 (8.29 MB)

Non-trainable params: 0 (0.00 B)
```

```
model.compile('Adam', 'binary_crossentropy', ['accuracy'])
model.fit(train_data, epochs=10, validation_data=test_data)
```

```
Epoch 1/10
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/keras/src/trainers/data_adapters/py_dataset_ad
  self._warn_if_super_not_called()
251/251 ━━━━━━━━━━━━━━━━━━━━ 22s 83ms/step - accuracy: 0.5308 - loss: 0.7071 - val_accuracy: 0.5393 - val_loss: 0.6830
Epoch 2/10
251/251 ━━━━━━━━━━━━━━━━━━━━ 21s 84ms/step - accuracy: 0.5835 - loss: 0.6706 - val_accuracy: 0.6545 - val_loss: 0.6405
Epoch 3/10
251/251 ━━━━━━━━━━━━━━━━━━━━ 21s 83ms/step - accuracy: 0.6392 - loss: 0.6378 - val_accuracy: 0.6960 - val_loss: 0.5935
Epoch 4/10
251/251 ━━━━━━━━━━━━━━━━━━━━ 22s 84ms/step - accuracy: 0.6735 - loss: 0.6124 - val_accuracy: 0.7098 - val_loss: 0.5840
Epoch 5/10
251/251 ━━━━━━━━━━━━━━━━━━━━ 23s 89ms/step - accuracy: 0.6824 - loss: 0.5960 - val_accuracy: 0.6663 - val_loss: 0.6050
Epoch 6/10
251/251 ━━━━━━━━━━━━━━━━━━━━ 21s 84ms/step - accuracy: 0.7003 - loss: 0.5775 - val_accuracy: 0.6920 - val_loss: 0.5726
Epoch 7/10
251/251 ━━━━━━━━━━━━━━━━━━━━ 21s 82ms/step - accuracy: 0.6826 - loss: 0.5875 - val_accuracy: 0.7153 - val_loss: 0.5515
Epoch 8/10
251/251 ━━━━━━━━━━━━━━━━━━━━ 20s 77ms/step - accuracy: 0.7047 - loss: 0.5631 - val_accuracy: 0.7514 - val_loss: 0.5175
Epoch 9/10
251/251 ━━━━━━━━━━━━━━━━━━━━ 20s 77ms/step - accuracy: 0.7103 - loss: 0.5597 - val_accuracy: 0.7355 - val_loss: 0.5312
Epoch 10/10
251/251 ━━━━━━━━━━━━━━━━━━━━ 20s 80ms/step - accuracy: 0.7163 - loss: 0.5528 - val_accuracy: 0.7385 - val_loss: 0.5232

<keras.src.callbacks.history.History at 0x3089e31d0>
```

Result:Trained a neural network model to classify the dogs and cats images

Experiment 4

Aim: Image Classification on MNIST dataset (CNN model with Fully connected Layer)

Importing The Libraries

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from tensorflow.keras.optimizers import Adam
```

PreProcessing and Loading Images

```
image_size = 64
batch_size = 32
train = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255,
        rotation_range=90, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)

test = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
train_set = train.flow_from_directory(r' ./dataset_mnist/train',
        target_size=(image_size, image_size), batch_size= batch_size,
class_mode='categorical')

test_set= test.flow_from_directory( r' ./dataset_mnist/test',
         target_size=(image_size, image_size), batch_size= batch_size,
class_mode='categorical')
```

Output: Found 100 images belonging to 10 classes
Found 100 images belonging to 10 classes

Model Building

```
model= Sequential([
        Conv2D(32,(3,3),activation='relu', input_shape=(112,112,3)),
        MaxPool2D(2,2),
        Conv2D(64,(3,3),activation='relu'),
        MaxPool2D(2,2),
        Conv2D(64,(3,3),activation='relu'),
        MaxPool2D(2,2),
        Flatten(),
        Dense(100, activation='relu' ),
        Dense(1, activation='sigmoid')
]
)

model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d_4 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_5 (Conv2D)	(None, 29, 29, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_6 (Conv2D)	(None, 12, 12, 64)	36,928
max_pooling2d_6 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_2 (Flatten)	(None, 2304)	0
dense_4 (Dense)	(None, 128)	295,040
dense_5 (Dense)	(None, 10)	1,290

Total params: 352,650 (1.35 MB)

Trainable params: 352,650 (1.35 MB)

Non-trainable params: 0 (0.00 B)

```
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
model.fit(train_data, epochs=15, validation_data=test_data)
```

```
Epoch 1/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 16s 62ms/step - accuracy: 0.5368 - loss: 0.6922 - val_accuracy: 0.6006 - val_loss: 0.6679
Epoch 2/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 13s 50ms/step - accuracy: 0.6118 - loss: 0.6610 - val_accuracy: 0.6268 - val_loss: 0.6380
Epoch 3/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 16s 62ms/step - accuracy: 0.6484 - loss: 0.6303 - val_accuracy: 0.6886 - val_loss: 0.5883
Epoch 4/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 15s 59ms/step - accuracy: 0.6693 - loss: 0.6067 - val_accuracy: 0.6925 - val_loss: 0.5756
Epoch 5/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 15s 60ms/step - accuracy: 0.6911 - loss: 0.5915 - val_accuracy: 0.7173 - val_loss: 0.5667
Epoch 6/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 14s 53ms/step - accuracy: 0.6826 - loss: 0.5858 - val_accuracy: 0.7168 - val_loss: 0.5597
Epoch 7/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 13s 50ms/step - accuracy: 0.6988 - loss: 0.5701 - val_accuracy: 0.7415 - val_loss: 0.5262
Epoch 8/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 13s 53ms/step - accuracy: 0.7154 - loss: 0.5515 - val_accuracy: 0.7321 - val_loss: 0.5251
Epoch 9/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 14s 53ms/step - accuracy: 0.7171 - loss: 0.5420 - val_accuracy: 0.7528 - val_loss: 0.5172
Epoch 10/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 13s 51ms/step - accuracy: 0.7241 - loss: 0.5412 - val_accuracy: 0.7593 - val_loss: 0.5010
Epoch 11/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 13s 51ms/step - accuracy: 0.7374 - loss: 0.5293 - val_accuracy: 0.7449 - val_loss: 0.5215
Epoch 12/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 15s 60ms/step - accuracy: 0.7362 - loss: 0.5243 - val_accuracy: 0.7667 - val_loss: 0.4891
Epoch 13/15
...
Epoch 14/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 13s 51ms/step - accuracy: 0.7337 - loss: 0.5240 - val_accuracy: 0.7701 - val_loss: 0.4853
Epoch 15/15
251/251 ━━━━━━━━━━━━━━━━━━━━ 13s 50ms/step - accuracy: 0.7537 - loss: 0.4993 - val_accuracy: 0.7514 - val_loss: 0.5015
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Result: Performed Image classification on MNIST Dataset for numeric digits from 0 to 9

Experiment 5

Aim: **Applying the pre-trained model VGG16 for MNIST Dataset Classification**

Importing The Libraries

```
from tensorflow import keras
from keras.models import Sequential, Model
from keras. layers import Input, Dense, Dropout, Flatten,Conv2D, MaxPoo12D
from keras.layers import BatchNormalization
```

Loading the dataset

```
image_size = 64
batch_size = 32
train = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255,
        rotation_range=90, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)

test = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
```

Output: Found 100 images belonging to 10 classes
Found 100 images belonging to 10 classes

```
input = Input (shape (224, 224,3))
x=Conv2D (filters=64, kernel_size=3, padding='same' , activation='relu')(input)
x=Conv2D (filters=64, kernel_size=3, padding='same' , activation='relu')(x)
x=MaxPool2D (pool_size=2, strides=2 ,padding='same' )(x)
x=Conv2D (filters=128, kernel_size=3, padding='same' , activation='relu')(x)
x=Conv2D (filters=128, kernel_size=3, padding='same' , activation='relu')(x)
x=MaxPool2D (pool_size=2, strides=2 ,padding='same' )(x)
x=Conv2D (filters=256, kernel_size=3, padding='same' , activation='relu')(x)
x=Conv2D (filters=256, kernel_size=3, padding='same' , activation='relu')(x)
x=Conv2D (filters=256, kernel_size=3, padding='same' , activation='relu')(x)
x=MaxPool2D (pool_size=2, strides=2 ,padding='same' )(x)
```

	Sheet No. ______________________
Laboratory Record of	**Experiment No.** _________________
NN & DL	**Date** ____________________________

```
x=Conv2D (filters=512, kernel_size=3, padding='same' , activation='relu')(x)
x=Conv2D (filters=512, kernel_size=3, padding='same' , activation='relu')(x)
x=Conv2D (filters=512, kernel_size=3, padding='same' , activation='relu')(x)
x=MaxPool2D (pool_size=2, strides=2 ,padding='same' )(x)
x=Conv2D (filters=256, kernel_size=3, padding='same' , activation='relu')(x)
x=Conv2D (filters=256, kernel_size=3, padding='same' , activation='relu')(x)
x=Conv2D (filters=256, kernel_size=3, padding='same' , activation='relu')(x)
x=MaxPool2D (pool_size=2, strides=2 ,padding='same' )(x)
x = Flatten() (x)
x = Dense (units = 4096, activation ='relu')(x)
output=Dense(units=10, activation='softmax')(x)
model = Model (inputs=input, outputs =output)
model.summary()
```

```
Model: "model"
Layer (type)                 Output Shape              Param #
===============================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
conv2d (Conv2D)              (None, 224, 224, 64)      1792
conv2d_1 (Conv2D)            (None, 224, 224, 64)      36928
max_pooling2d (MaxPooling2   (None, 112, 112, 64)      0
D)
conv2d_2 (Conv2D)            (None, 112, 112, 128)     73856
conv2d_3 (Conv2D)            (None, 112, 112, 128)     147584
max_pooling2d_1 (MaxPoolin   (None, 56, 56, 128)       0
g2D)
conv2d_4 (Conv2D)            (None, 56, 56, 256)       295168
conv2d_5 (Conv2D)            (None, 56, 56, 256)       590080
conv2d_6 (Conv2D)            (None, 56, 56, 256)       590080
max_pooling2d_2 (MaxPoolin   (None, 28, 28, 256)       0
g2D)
conv2d_7 (Conv2D)            (None, 28, 28, 512)       1180160
conv2d_8 (Conv2D)            (None, 28, 28, 512)       2359808
conv2d_9 (Conv2D)            (None, 28, 28, 512)       2359808
max_pooling2d_3 (MaxPoolin   (None, 14, 14, 512)       0
g2D)
conv2d_10 (Conv2D)           (None, 14, 14, 512)       2359808
conv2d_11 (Conv2D)           (None, 14, 14, 512)       2359808
conv2d_12 (Conv2D)           (None, 14, 14, 512)       2359808
max_pooling2d_4 (MaxPoolin   (None, 7, 7, 512)         0
g2D)
flatten (Flatten)            (None, 25088)             0
dense (Dense)                (None, 4096)              102764544
dense_1 (Dense)              (None, 4096)              16781312
dense_2 (Dense)              (None, 10)                40970
===============================================================
Total params: 134301514 (512.32 MB)
Trainable params: 134301514 (512.32 MB)
```

Laboratory Record of

NN & DL

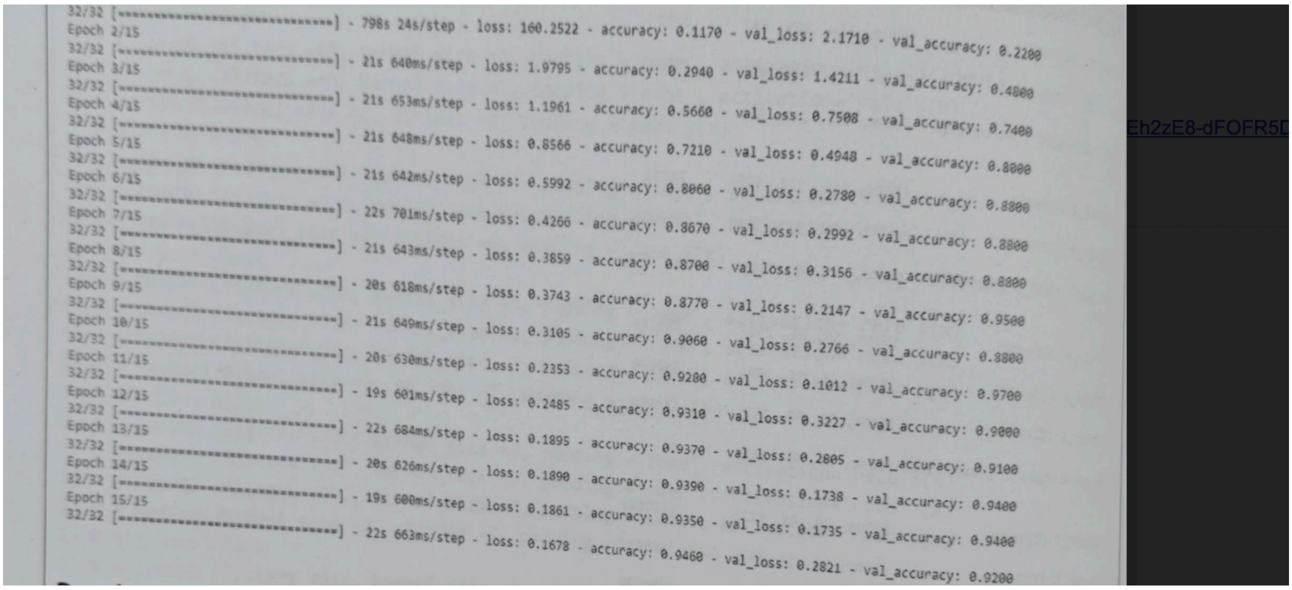
Sheet No. ____________________

Experiment No. ______________

Date ________________________

```
model.compile(loss="categorical_crossentropy", optimizer='Adam',metrics=['accuracy'])

history= model.fit(train_data, epochs=15, validation_data=test_data)
```



```
32/32 [==============================] - 798s 24s/step - loss: 160.2522 - accuracy: 0.1170 - val_loss: 2.1710 - val_accuracy: 0.2200
Epoch 2/15
32/32 [==============================] - 21s 640ms/step - loss: 1.9795 - accuracy: 0.2940 - val_loss: 1.4211 - val_accuracy: 0.4800
Epoch 3/15
32/32 [==============================] - 21s 653ms/step - loss: 1.1961 - accuracy: 0.5660 - val_loss: 0.7508 - val_accuracy: 0.7400
Epoch 4/15
32/32 [==============================] - 21s 648ms/step - loss: 0.8566 - accuracy: 0.7210 - val_loss: 0.4948 - val_accuracy: 0.8000
Epoch 5/15
32/32 [==============================] - 21s 642ms/step - loss: 0.5992 - accuracy: 0.8060 - val_loss: 0.2780 - val_accuracy: 0.8800
Epoch 6/15
32/32 [==============================] - 22s 701ms/step - loss: 0.4266 - accuracy: 0.8670 - val_loss: 0.2992 - val_accuracy: 0.8800
Epoch 7/15
32/32 [==============================] - 21s 643ms/step - loss: 0.3859 - accuracy: 0.8700 - val_loss: 0.3156 - val_accuracy: 0.8800
Epoch 8/15
32/32 [==============================] - 20s 618ms/step - loss: 0.3743 - accuracy: 0.8770 - val_loss: 0.2147 - val_accuracy: 0.9500
Epoch 9/15
32/32 [==============================] - 21s 649ms/step - loss: 0.3105 - accuracy: 0.9060 - val_loss: 0.2766 - val_accuracy: 0.8800
Epoch 10/15
32/32 [==============================] - 20s 630ms/step - loss: 0.2353 - accuracy: 0.9280 - val_loss: 0.1012 - val_accuracy: 0.9700
Epoch 11/15
32/32 [==============================] - 19s 601ms/step - loss: 0.2485 - accuracy: 0.9310 - val_loss: 0.3227 - val_accuracy: 0.9000
Epoch 12/15
32/32 [==============================] - 22s 684ms/step - loss: 0.1895 - accuracy: 0.9370 - val_loss: 0.2805 - val_accuracy: 0.9100
Epoch 13/15
32/32 [==============================] - 20s 626ms/step - loss: 0.1890 - accuracy: 0.9390 - val_loss: 0.1738 - val_accuracy: 0.9400
Epoch 14/15
32/32 [==============================] - 19s 600ms/step - loss: 0.1861 - accuracy: 0.9350 - val_loss: 0.1735 - val_accuracy: 0.9400
Epoch 15/15
32/32 [==============================] - 22s 663ms/step - loss: 0.1678 - accuracy: 0.9460 - val_loss: 0.2821 - val_accuracy: 0.9200
```

Result:Successfully implemented the pre-trained CNN model VGG16 for MNIST Dataset Classification

Roll No. ____________________ MGIT