Summer 2024: CS5720

Neural Networks and Deep Learning - ICP-3

GitHub Link: https://github.com/gouthamthogaru/ICP3

Programming elements:

Keras Basics

In class programming:

1. Use the use case in the class:
   a. Add more Dense layers to the existing code and check how the accuracy changes.

In [9]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount
=True).
```

In [14]:
```
# Mounting my google drive in order to get the data.csv file that I uploaded in my drive
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
from sklearn.decomposition import PCA
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
```

```
Mounted at /content/drive
```

In [15]:

```
In [20]:
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np



dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/diabetes.csv', header=None).values

X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                    test_size=0.25, random_state=87)
np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                     initial_epoch=0)
print(my_first_nn.summary())
print(my_first_nn.evaluate(X_test, Y_test))
```

```
Epoch 1/100
18/18 [==============================] - 1s 2ms/step - loss: 39.6943 - accuracy: 0.6615
Epoch 2/100
18/18 [==============================] - 0s 2ms/step - loss: 27.5460 - accuracy: 0.6615
Epoch 3/100
18/18 [==============================] - 0s 2ms/step - loss: 15.6921 - accuracy: 0.6615
Epoch 4/100
18/18 [==============================] - 0s 2ms/step - loss: 4.7755 - accuracy: 0.6024
Epoch 5/100
```

2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_6 (Dense)             (None, 20)                180       
                                                                 
 dense_7 (Dense)             (None, 1)                 21        
                                                                 
=================================================================
Total params: 201 (804.00 Byte)
Trainable params: 201 (804.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
6/6 [==============================] - 0s 2ms/step - loss: 0.7013 - accuracy: 0.6354
[0.7013064026832581, 0.6354166865348816]
```

In [21]:

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Load the dataset
dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/diabetes.csv', header=None).values

# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8], test_size=0.25, random_state=87)
np.random.seed(155)

# Create the model
my_first_nn = Sequential()
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer 1
my_first_nn.add(Dense(15, activation='relu')) # hidden layer 2
my_first_nn.add(Dense(10, activation='relu')) # hidden layer 3
```

3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below).

```
my_first_nn.add(Dense(10, activation='relu')) # hidden layer 3
my_first_nn.add(Dense(1, activation='sigmoid')) # output layer

# Compile the model
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)

# Print the model summary
print(my_first_nn.summary())

# Evaluate the model
accuracy = my_first_nn.evaluate(X_test, Y_test)
print(f"Accuracy: {accuracy[1]}")
```

```
Epoch 1/100
18/18 [==============================] - 2s 2ms/step - loss: 1.2324 - accuracy: 0.6580
Epoch 2/100
18/18 [==============================] - 0s 2ms/step - loss: 0.8708 - accuracy: 0.6701
Epoch 3/100
18/18 [==============================] - 0s 2ms/step - loss: 0.7693 - accuracy: 0.6788
Epoch 4/100
18/18 [==============================] - 0s 2ms/step - loss: 0.7196 - accuracy: 0.6806
Epoch 5/100
18/18 [==============================] - 0s 2ms/step - loss: 0.7021 - accuracy: 0.6667
Epoch 6/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6278 - accuracy: 0.6840
Epoch 7/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6132 - accuracy: 0.6788
Epoch 8/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5904 - accuracy: 0.6944
Epoch 9/100
```

github.com/gouthamthogaru/ICP3/blob/main/ICP-3.ipynb

```python
# Preprocess the dataset
# Drop the 'id' column and the 'Unnamed: 32' column
dataset.drop(['id', 'Unnamed: 32'], axis=1, inplace=True)

# Convert the 'diagnosis' column to binary labels (M -> 1, B -> 0)
dataset['diagnosis'] = dataset['diagnosis'].map({'M': 1, 'B': 0})

# Split the dataset into features (X) and labels (Y)
X = dataset.iloc[:, 1:].values
Y = dataset.iloc[:, 0].values

# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)
np.random.seed(155)

# Create the model
my_first_nn = Sequential()
my_first_nn.add(Dense(20, input_dim=X.shape[1], activation='relu')) # hidden layer 1
my_first_nn.add(Dense(15, activation='relu')) # hidden layer 2
my_first_nn.add(Dense(10, activation='relu')) # hidden layer 3
my_first_nn.add(Dense(1, activation='sigmoid')) # output layer

# Compile the model
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)

# Print the model summary
print(my_first_nn.summary())

# Evaluate the model
accuracy = my_first_nn.evaluate(X_test, Y_test)
print(f"Accuracy: {accuracy[1]}")
```

github.com/gouthamthogaru/ICP3/blob/main/ICP-3.ipynb

CP3 / ICP-3.ipynb

Preview | Code | Blame | 2014 lines (2014 loc) · 318 KB

```
# Evaluate the model
accuracy = my_first_nn.evaluate(X_test, Y_test)
print(f"Accuracy: {accuracy[1]}")
```

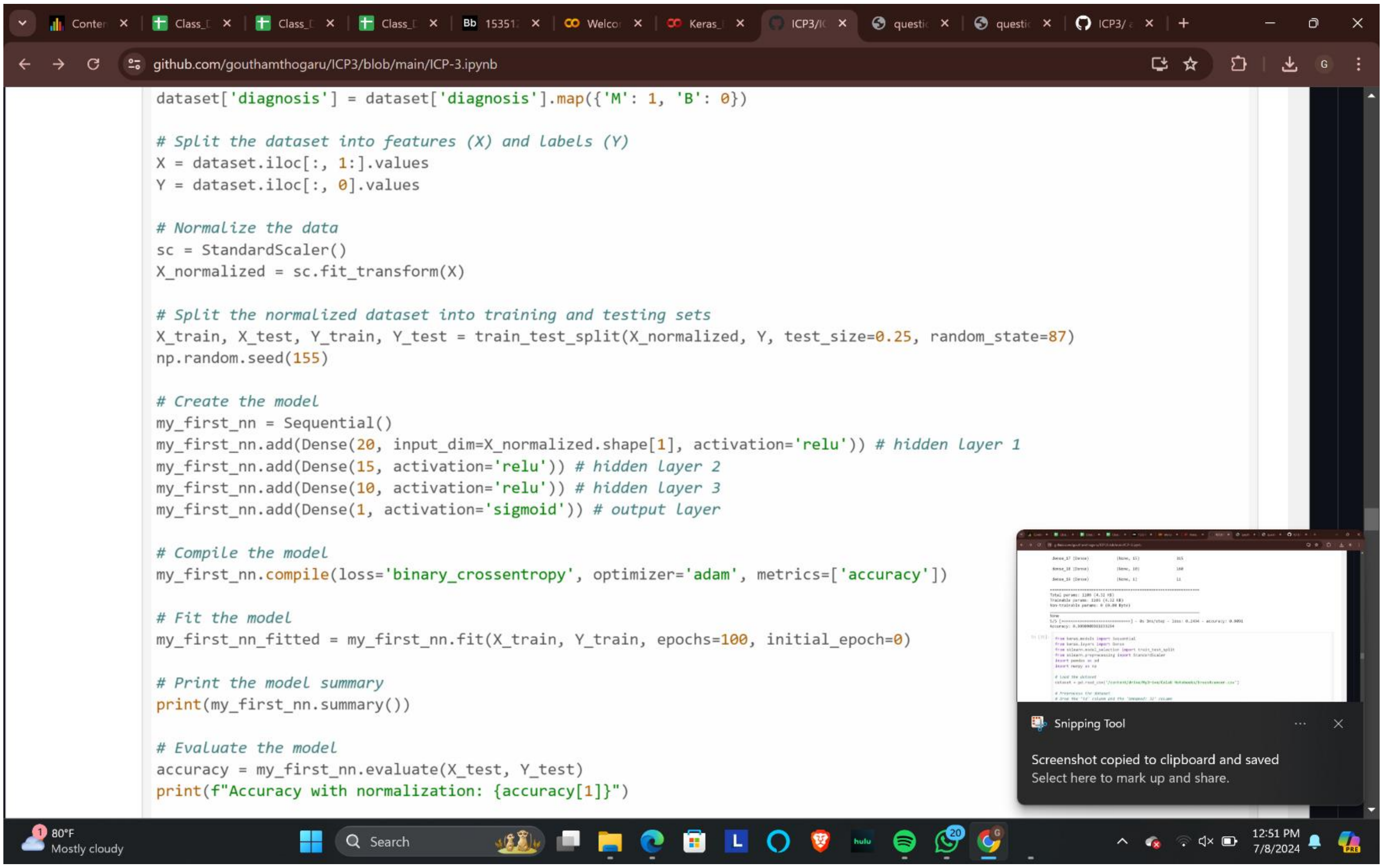
```
Epoch 1/100
14/14 [==============================] - 1s 2ms/step - loss: 1.5231 - accuracy: 0.4953
Epoch 2/100
14/14 [==============================] - 0s 3ms/step - loss: 0.6570 - accuracy: 0.6385
Epoch 3/100
14/14 [==============================] - 0s 2ms/step - loss: 0.4654 - accuracy: 0.7723
Epoch 4/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3459 - accuracy: 0.9085
Epoch 5/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3483 - accuracy: 0.8545
Epoch 6/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3000 - accuracy: 0.8944
Epoch 7/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2811 - accuracy: 0.9061
Epoch 8/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2502 - accuracy: 0.9249
Epoch 9/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2417 - accuracy: 0.9178
Epoch 10/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2291 - accuracy: 0.9202
Epoch 11/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2375 - accuracy: 0.9085
Epoch 12/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3273 - accuracy: 0.8685
Epoch 13/100
```

80°F Mostly cloudy   12:50 PM 7/8/2024

```
 dense_17 (Dense)            (None, 15)                315

 dense_18 (Dense)            (None, 10)                160

 dense_19 (Dense)            (None, 1)                 11

=================================================================
Total params: 1106 (4.32 KB)
Trainable params: 1106 (4.32 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
5/5 [==============================] - 0s 3ms/step - loss: 0.2494 - accuracy: 0.9091
Accuracy: 0.9090909361839294
```

In [31]:

```python
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# Load the dataset
dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/breastcancer.csv')

# Preprocess the dataset
# Drop the 'id' column and the 'Unnamed: 32' column
dataset.drop(['id', 'Unnamed: 32'], axis=1, inplace=True)

# Convert the 'diagnosis' column to binary labels (M -> 1, B -> 0)
dataset['diagnosis'] = dataset['diagnosis'].map({'M': 1, 'B': 0})

# Split the dataset into features (X) and labels (Y)
```



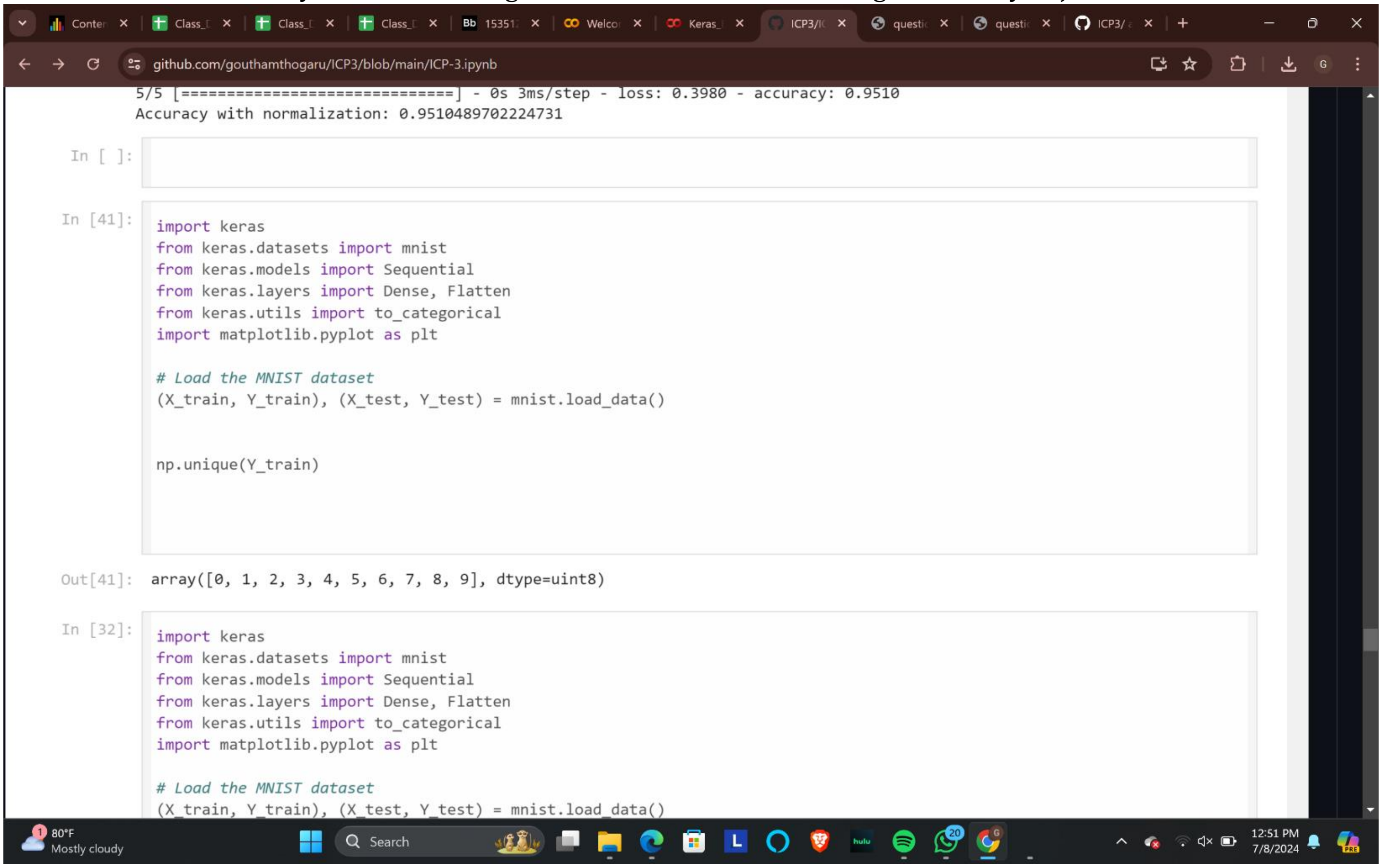

```python
dataset['diagnosis'] = dataset['diagnosis'].map({'M': 1, 'B': 0})

# Split the dataset into features (X) and labels (Y)
X = dataset.iloc[:, 1:].values
Y = dataset.iloc[:, 0].values

# Normalize the data
sc = StandardScaler()
X_normalized = sc.fit_transform(X)

# Split the normalized dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_normalized, Y, test_size=0.25, random_state=87)
np.random.seed(155)

# Create the model
my_first_nn = Sequential()
my_first_nn.add(Dense(20, input_dim=X_normalized.shape[1], activation='relu')) # hidden layer 1
my_first_nn.add(Dense(15, activation='relu')) # hidden layer 2
my_first_nn.add(Dense(10, activation='relu')) # hidden layer 3
my_first_nn.add(Dense(1, activation='sigmoid')) # output layer

# Compile the model
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)

# Print the model summary
print(my_first_nn.summary())

# Evaluate the model
accuracy = my_first_nn.evaluate(X_test, Y_test)
print(f"Accuracy with normalization: {accuracy[1]}")
```

github.com/gouthamthogaru/ICP3/blob/main/ICP-3.ipynb

```python
print(f"Accuracy with normalization: {accuracy[1]}")
```

```
Epoch 1/100
14/14 [==============================] - 1s 2ms/step - loss: 0.7172 - accuracy: 0.5446
Epoch 2/100
14/14 [==============================] - 0s 2ms/step - loss: 0.5851 - accuracy: 0.7676
Epoch 3/100
14/14 [==============================] - 0s 2ms/step - loss: 0.4920 - accuracy: 0.8451
Epoch 4/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3989 - accuracy: 0.8991
Epoch 5/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3168 - accuracy: 0.9272
Epoch 6/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2491 - accuracy: 0.9413
Epoch 7/100
14/14 [==============================] - 0s 3ms/step - loss: 0.1968 - accuracy: 0.9484
Epoch 8/100
14/14 [==============================] - 0s 3ms/step - loss: 0.1590 - accuracy: 0.9577
Epoch 9/100
14/14 [==============================] - 0s 3ms/step - loss: 0.1319 - accuracy: 0.9648
Epoch 10/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1126 - accuracy: 0.9695
Epoch 11/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0967 - accuracy: 0.9718
Epoch 12/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0855 - accuracy: 0.9765
Epoch 13/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0762 - accuracy: 0.9789
Epoch 14/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0688 - accuracy: 0.9836
Epoch 15/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0627 - accuracy: 0.9836
Epoch 16/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0578 - accuracy: 0.9883
```

1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.



```
5/5 [==============================] - 0s 3ms/step - loss: 0.3980 - accuracy: 0.9510
Accuracy with normalization: 0.9510489702224731
```

In [ ]:

In [41]:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load the MNIST dataset
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()


np.unique(Y_train)
```

Out[41]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

In [32]:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load the MNIST dataset
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

github.com/gouthamthogaru/ICP3/blob/main/ICP-3.ipynb

```python
from keras.layers import Dense, Flatten
from keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load the MNIST dataset
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

# Normalize the data
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# One hot encode the labels
Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)

# Create the model
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, Y_train, epochs=10, validation_split=0.2)

# Plot the loss and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
```

github.com/gouthamthogaru/ICP3/blob/main/ICP-3.ipynb

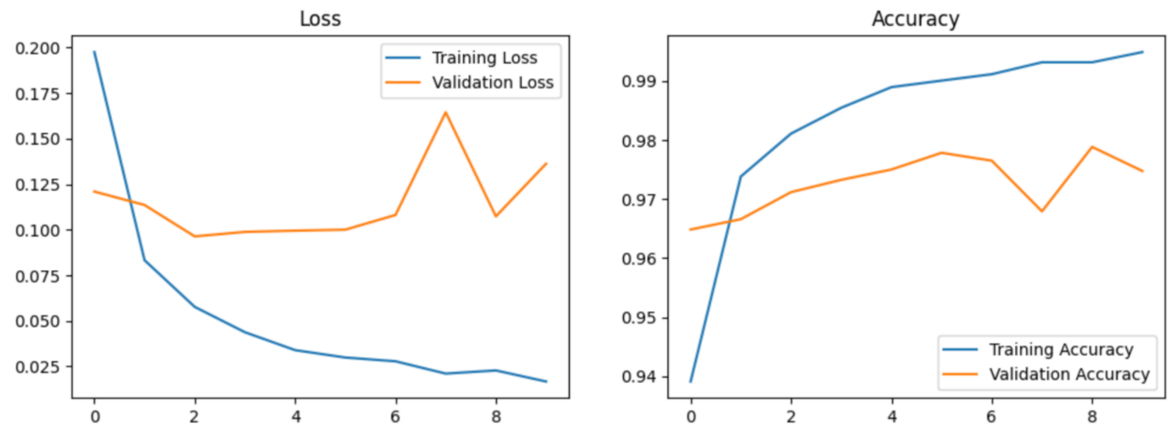
```python
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.legend()

plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.1975 - accuracy: 0.9391 - val_loss: 0.1209 - val_accur
acy: 0.9648
Epoch 2/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0833 - accuracy: 0.9738 - val_loss: 0.1136 - val_accur
acy: 0.9666
Epoch 3/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.0577 - accuracy: 0.9811 - val_loss: 0.0963 - val_accur
acy: 0.9712
Epoch 4/10
1500/1500 [==============================] - 19s 13ms/step - loss: 0.0438 - accuracy: 0.9855 - val_loss: 0.0988 - val_accur
acy: 0.9732
Epoch 5/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0340 - accuracy: 0.9890 - val_loss: 0.0995 - val_accur
acy: 0.9750
Epoch 6/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0299 - accuracy: 0.9901 - val_loss: 0.1000 - val_accur
acy: 0.9778
Epoch 7/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0278 - accuracy: 0.9911 - val_loss: 0.1081 - val_accur
acy: 0.9765
Epoch 8/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0211 - accuracy: 0.9932 - val_loss: 0.1644 - val_accur
acy: 0.9679
```

80°F Mostly cloudy 12:52 PM 7/8/2024

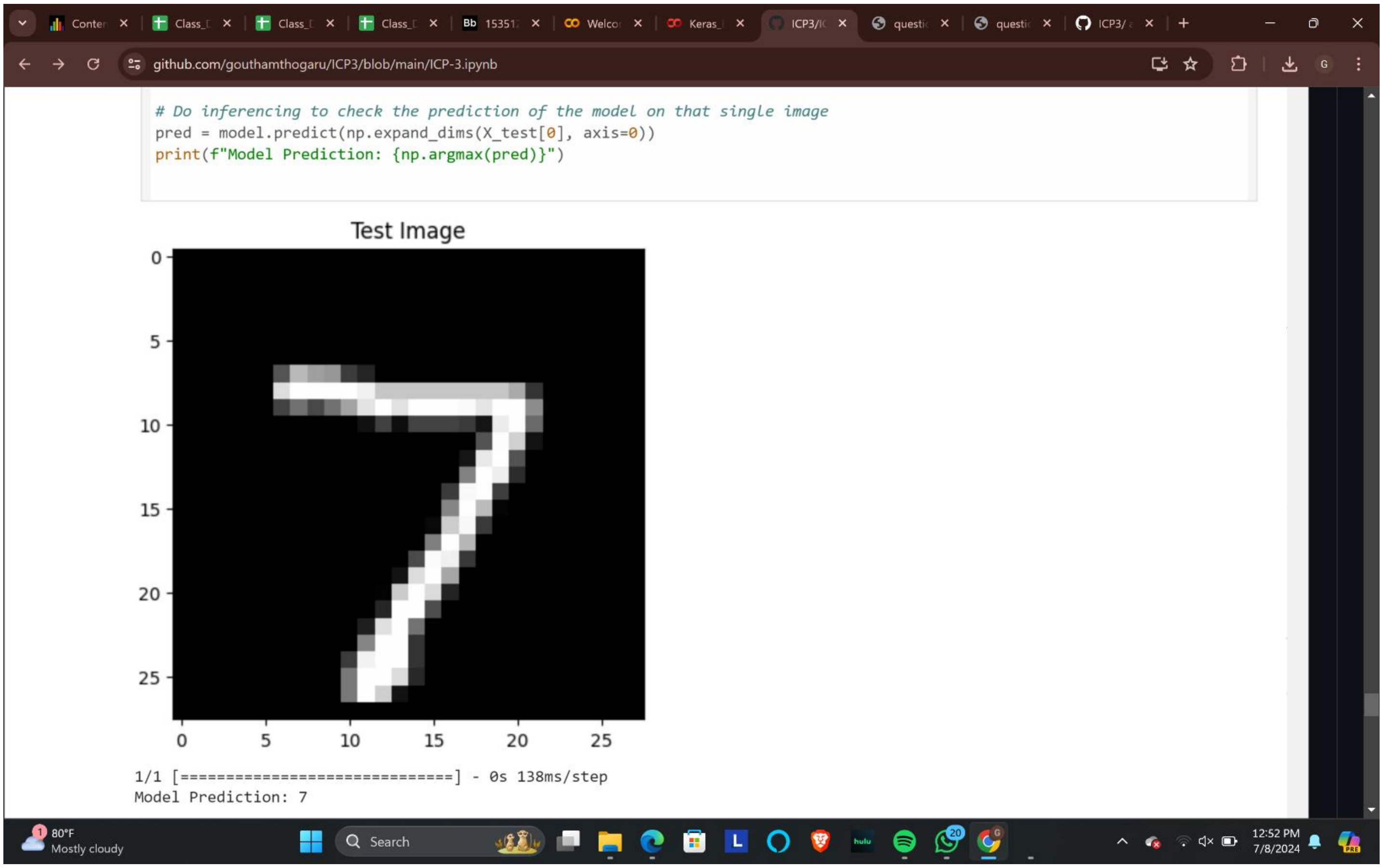
github.com/gouthamthogaru/ICP3/blob/main/ICP-3.ipynb

```
Epoch 8/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0211 - accuracy: 0.9932 - val_loss: 0.1644 - val_accuracy: 0.9679
Epoch 9/10
1500/1500 [==============================] - 19s 12ms/step - loss: 0.0228 - accuracy: 0.9932 - val_loss: 0.1073 - val_accuracy: 0.9788
Epoch 10/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.0168 - accuracy: 0.9949 - val_loss: 0.1362 - val_accuracy: 0.9747
```

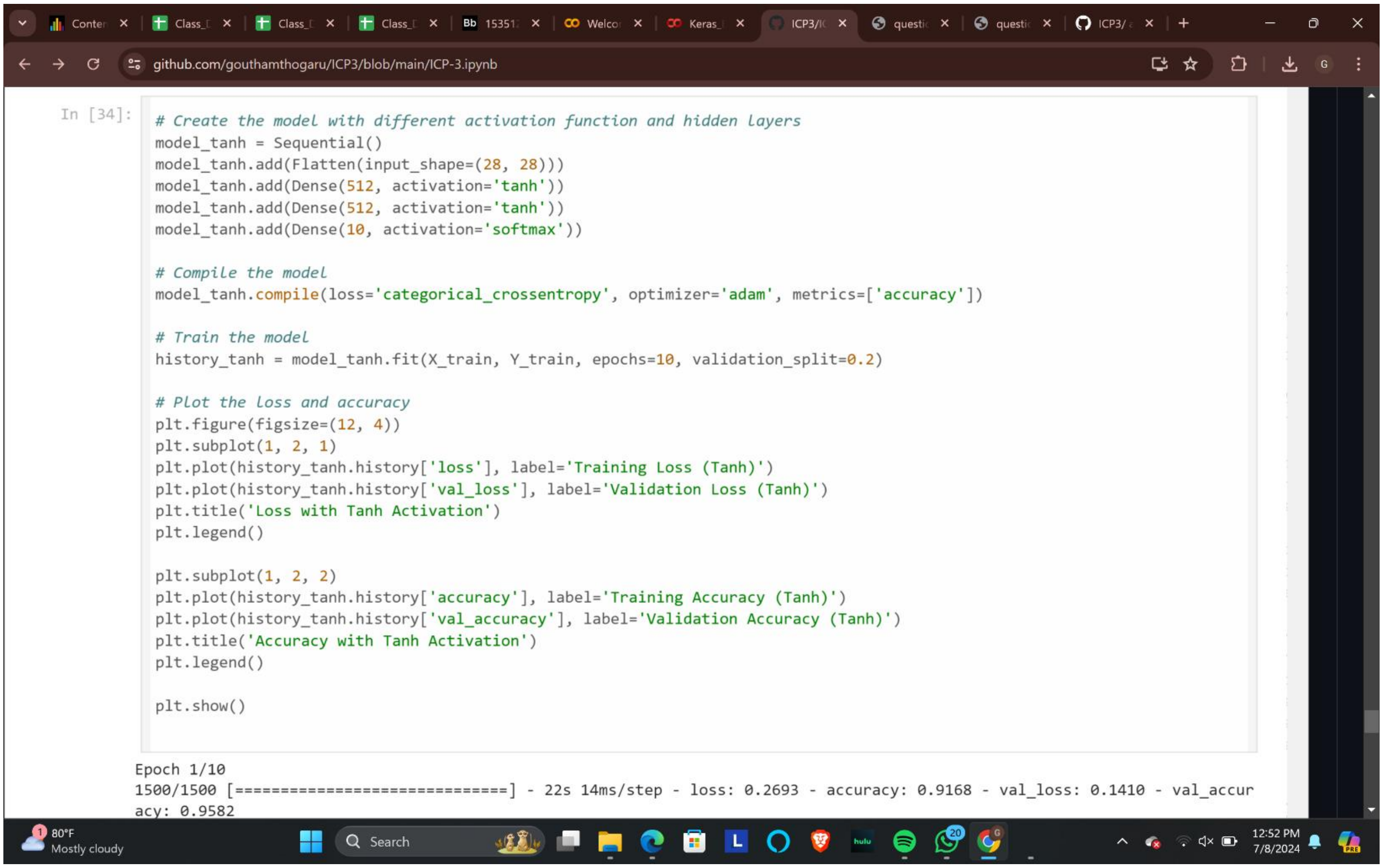


In [33]:

```python
import numpy as np

# Plot one of the images in the test data
plt.imshow(X_test[0], cmap='gray')
plt.title('Test Image')
```



Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image



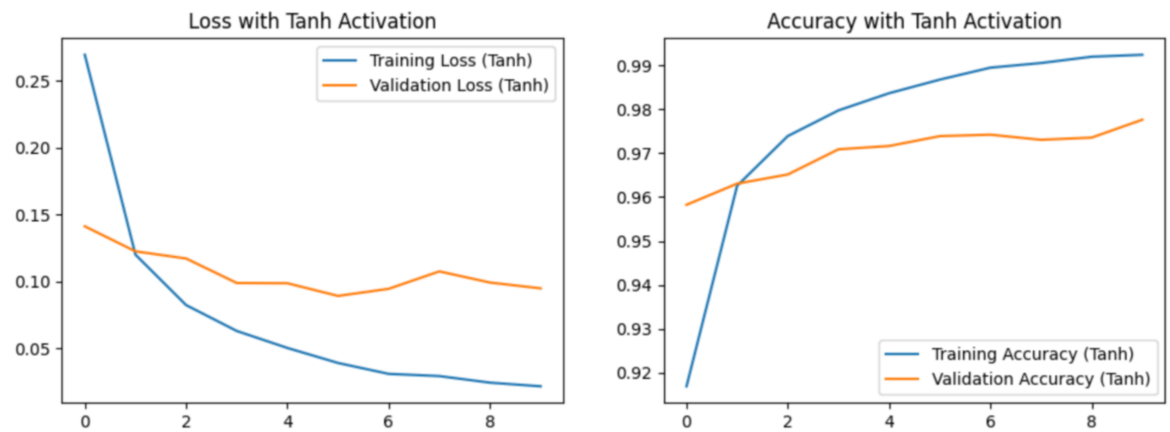

```python
# Create the model with different activation function and hidden layers
model_tanh = Sequential()
model_tanh.add(Flatten(input_shape=(28, 28)))
model_tanh.add(Dense(512, activation='tanh'))
model_tanh.add(Dense(512, activation='tanh'))
model_tanh.add(Dense(10, activation='softmax'))

# Compile the model
model_tanh.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history_tanh = model_tanh.fit(X_train, Y_train, epochs=10, validation_split=0.2)

# Plot the loss and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_tanh.history['loss'], label='Training Loss (Tanh)')
plt.plot(history_tanh.history['val_loss'], label='Validation Loss (Tanh)')
plt.title('Loss with Tanh Activation')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_tanh.history['accuracy'], label='Training Accuracy (Tanh)')
plt.plot(history_tanh.history['val_accuracy'], label='Validation Accuracy (Tanh)')
plt.title('Accuracy with Tanh Activation')
plt.legend()

plt.show()
```

```
Epoch 1/10
1500/1500 [==============================] - 22s 14ms/step - loss: 0.2693 - accuracy: 0.9168 - val_loss: 0.1410 - val_accur
acy: 0.9582
```

2. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.

github.com/gouthamthogaru/ICP3/blob/main/ICP-3.ipynb

```
Epoch 8/10
1500/1500 [==============================] - 17s 12ms/step - loss: 0.0291 - accuracy: 0.9906 - val_loss: 0.1073 - val_accur
acy: 0.9731
Epoch 9/10
1500/1500 [==============================] - 17s 12ms/step - loss: 0.0241 - accuracy: 0.9920 - val_loss: 0.0990 - val_accur
acy: 0.9736
Epoch 10/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0214 - accuracy: 0.9925 - val_loss: 0.0947 - val_accur
acy: 0.9777
```



In [44]:

```
# Load the MNIST dataset without scaling the images
(X_train_ns, Y_train_ns), (X_test_ns, Y_test_ns) = mnist.load_data()

# One hot encode the labels
```

```python
# One hot encode the labels
Y_train_ns = to_categorical(Y_train_ns, 10)
Y_test_ns = to_categorical(Y_test_ns, 10)

# Create the model
model_ns = Sequential()
model_ns.add(Flatten(input_shape=(28, 28)))
model_ns.add(Dense(512, activation='relu'))
model_ns.add(Dense(512, activation='relu'))
model_ns.add(Dense(10, activation='softmax'))

# Compile the model
model_ns.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history_ns = model_ns.fit(X_train_ns, Y_train_ns, epochs=10, validation_split=0.2)

# Plot the loss and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_ns.history['loss'], label='Training Loss (No Scaling)')
plt.plot(history_ns.history['val_loss'], label='Validation Loss (No Scaling)')
plt.title('Loss without Scaling')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_ns.history['accuracy'], label='Training Accuracy (No Scaling)')
plt.plot(history_ns.history['val_accuracy'], label='Validation Accuracy (No Scaling)')
plt.title('Accuracy without Scaling')
plt.legend()

plt.show()
```

Run the same code without scaling the images and check the performance?

github.com/gouthamthogaru/ICP3/blob/main/ICP-3.ipynb

```
plt.show()
```

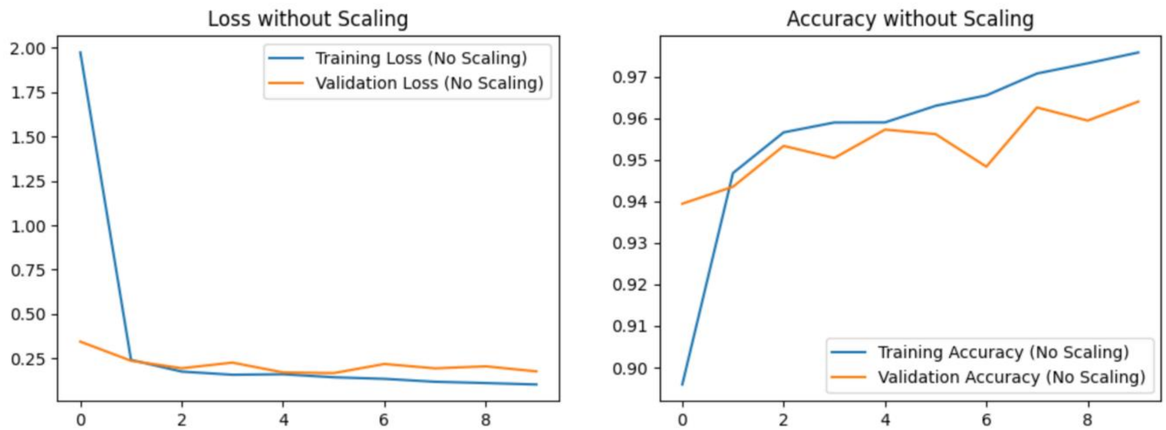
```
Epoch 1/10
1500/1500 [==============================] - 19s 12ms/step - loss: 1.9743 - accuracy: 0.8959 - val_loss: 0.3445 - val_accur
acy: 0.9394
Epoch 2/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.2415 - accuracy: 0.9468 - val_loss: 0.2367 - val_accur
acy: 0.9435
Epoch 3/10
1500/1500 [==============================] - 19s 13ms/step - loss: 0.1757 - accuracy: 0.9566 - val_loss: 0.1944 - val_accur
acy: 0.9533
Epoch 4/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.1579 - accuracy: 0.9590 - val_loss: 0.2264 - val_accur
acy: 0.9504
Epoch 5/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.1608 - accuracy: 0.9590 - val_loss: 0.1711 - val_accur
acy: 0.9572
Epoch 6/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.1437 - accuracy: 0.9630 - val_loss: 0.1671 - val_accur
acy: 0.9562
Epoch 7/10
1500/1500 [==============================] - 17s 12ms/step - loss: 0.1348 - accuracy: 0.9655 - val_loss: 0.2186 - val_accur
acy: 0.9483
Epoch 8/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.1184 - accuracy: 0.9707 - val_loss: 0.1940 - val_accur
acy: 0.9626
Epoch 9/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.1111 - accuracy: 0.9732 - val_loss: 0.2053 - val_accur
acy: 0.9594
Epoch 10/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.1030 - accuracy: 0.9758 - val_loss: 0.1767 - val_accur
acy: 0.9640
```

Loss without Scaling

Accuracy without Scaling

github.com/gouthamthogaru/ICP3/blob/main/ICP-3.ipynb

```
acy: 0.9626
Epoch 9/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.1111 - accuracy: 0.9732 - val_loss: 0.2053 - val_accur
acy: 0.9594
Epoch 10/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.1030 - accuracy: 0.9758 - val_loss: 0.1767 - val_accur
acy: 0.9640
```



In [ ]:

80°F
Mostly cloudy
Search
12:53 PM
7/8/2024