Revathi Atchi

700742168

NEURAL NETWORK AND DEEP LEARNING ASSIGNMENT-6

GITHUB LINK: https://github.com/revathiatchi/NeuralAssignment6.git

RECORDINGLINK:
https://github.com/revathiatchi/NeuralAssignment6/assets/156601745/3b6d7dfb-7279-478c-8540-5fafc6b51a24

Use Case Description: Predicting the diabetes disease

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.

2. Change the data source to Breast Cancer dataset * available in the source code folder and make required

changes. Report accuracy of the model.

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

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = B cancer

```python
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Loading the diabetes dataset
dataset = pd.read_csv('diabetes.csv')

# Splitting the dataset into the dependent and independent variables
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)

# Normalizing the data using StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Building the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

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

# Training the model
history = model.fit(X_train, y_train, epochs=100,
                    batch_size=10, validation_split=0.2)

# Evaluating the model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

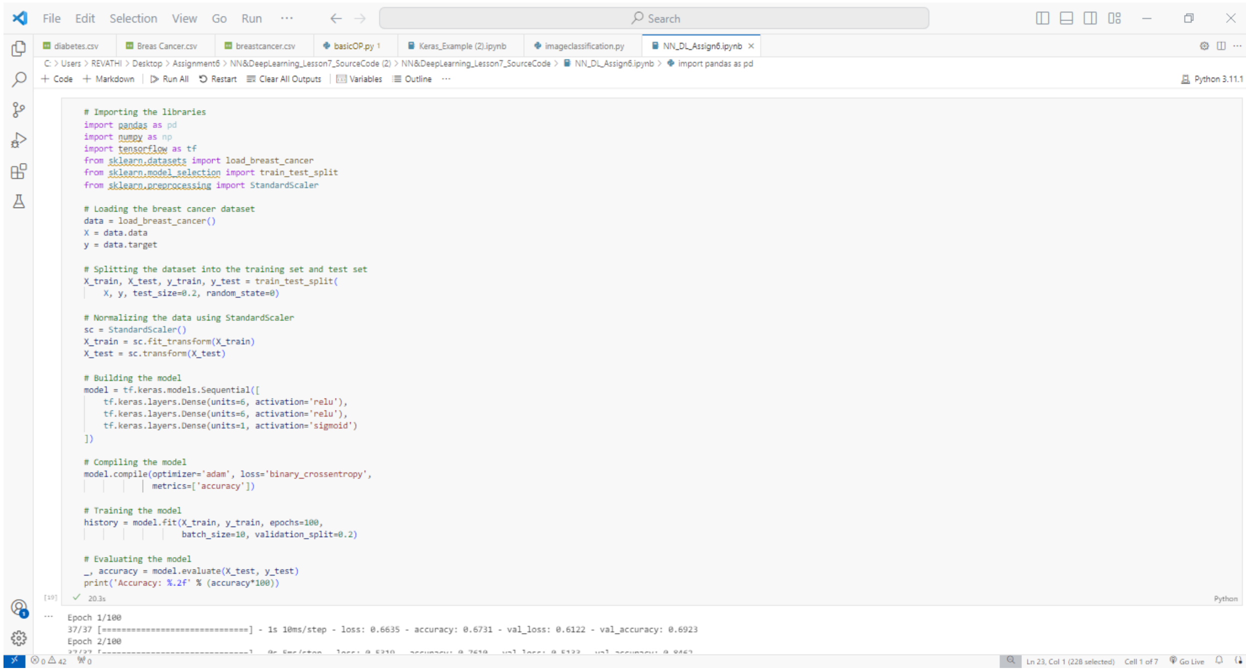
```
Epoch 1/100
49/49 [==============================] - 1s 8ms/step - loss: 0.7811 - accuracy: 0.3531 - val_loss: 0.7344 - val_accuracy: 0.4146
Epoch 2/100
```

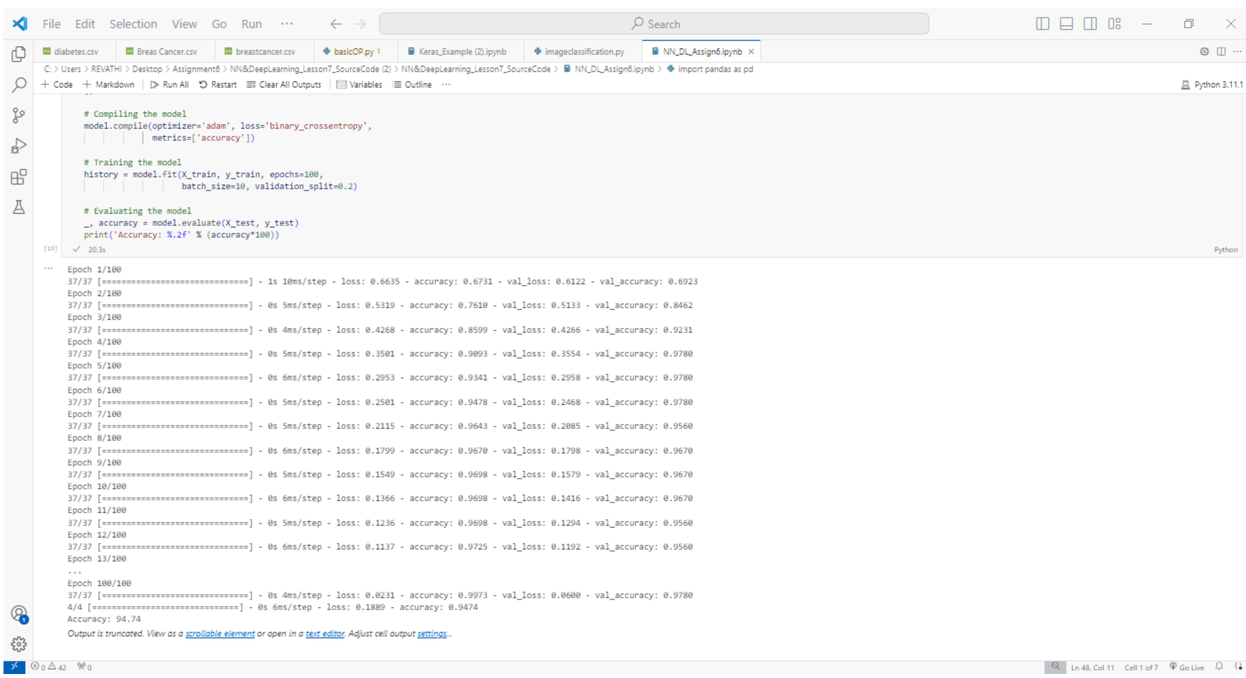
```python
# Training the model
history = model.fit(X_train, y_train, epochs=100,
                    batch_size=10, validation_split=0.2)

# Evaluating the model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Epoch 1/100
49/49 [==============================] - 1s 8ms/step - loss: 0.7811 - accuracy: 0.3531 - val_loss: 0.7344 - val_accuracy: 0.4146
Epoch 2/100
49/49 [==============================] - 0s 4ms/step - loss: 0.7197 - accuracy: 0.4327 - val_loss: 0.7095 - val_accuracy: 0.5122
Epoch 3/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6861 - accuracy: 0.5755 - val_loss: 0.6945 - val_accuracy: 0.5610
Epoch 4/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6596 - accuracy: 0.7000 - val_loss: 0.6830 - val_accuracy: 0.5935
Epoch 5/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6340 - accuracy: 0.7102 - val_loss: 0.6725 - val_accuracy: 0.5610
Epoch 6/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6076 - accuracy: 0.7082 - val_loss: 0.6636 - val_accuracy: 0.5610
Epoch 7/100
49/49 [==============================] - 0s 4ms/step - loss: 0.5827 - accuracy: 0.7122 - val_loss: 0.6553 - val_accuracy: 0.5610
Epoch 8/100
49/49 [==============================] - 0s 4ms/step - loss: 0.5603 - accuracy: 0.7184 - val_loss: 0.6477 - val_accuracy: 0.5854
Epoch 9/100
49/49 [==============================] - 0s 4ms/step - loss: 0.5408 - accuracy: 0.7245 - val_loss: 0.6407 - val_accuracy: 0.5854
Epoch 10/100
49/49 [==============================] - 0s 4ms/step - loss: 0.5237 - accuracy: 0.7245 - val_loss: 0.6334 - val_accuracy: 0.6016
Epoch 11/100
49/49 [==============================] - 0s 4ms/step - loss: 0.5106 - accuracy: 0.7224 - val_loss: 0.6279 - val_accuracy: 0.6260
Epoch 12/100
49/49 [==============================] - 0s 3ms/step - loss: 0.4987 - accuracy: 0.7306 - val_loss: 0.6208 - val_accuracy: 0.6260
Epoch 13/100
...
Epoch 100/100
49/49 [==============================] - 0s 4ms/step - loss: 0.4067 - accuracy: 0.8041 - val_loss: 0.6680 - val_accuracy: 0.6667
5/5 [==============================] - 0s 3ms/step - loss: 0.4858 - accuracy: 0.7792
Accuracy: 77.92
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
# Importing the libraries
import pandas as pd
import numpy as np
```

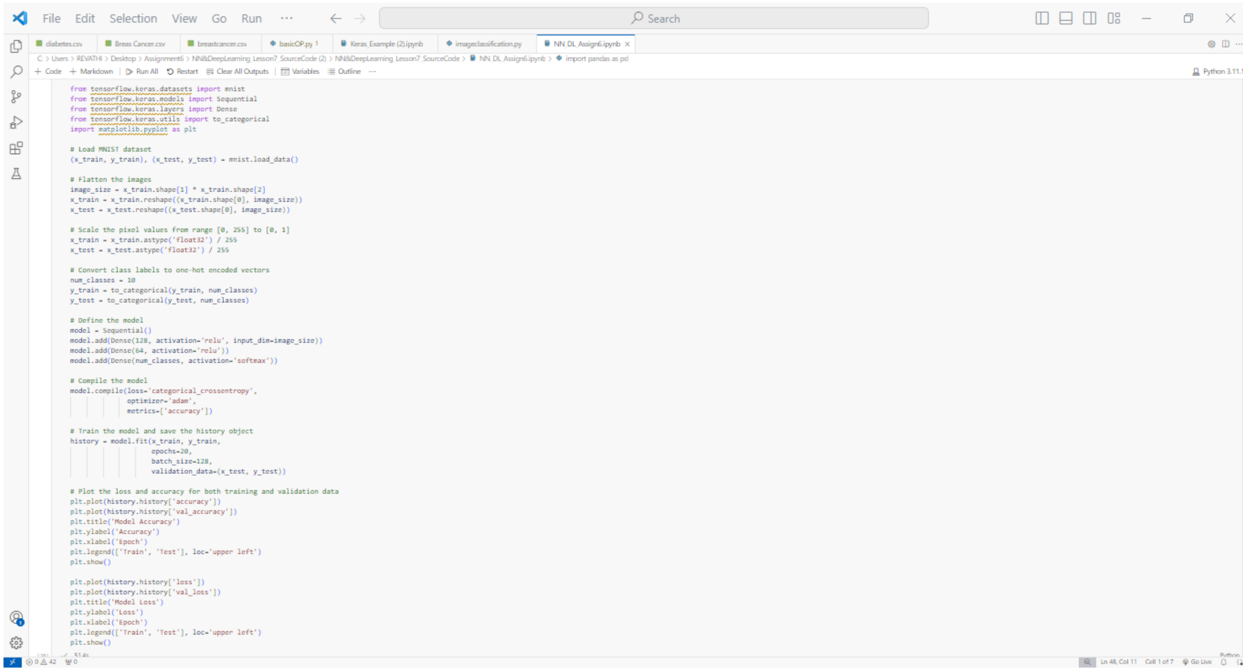


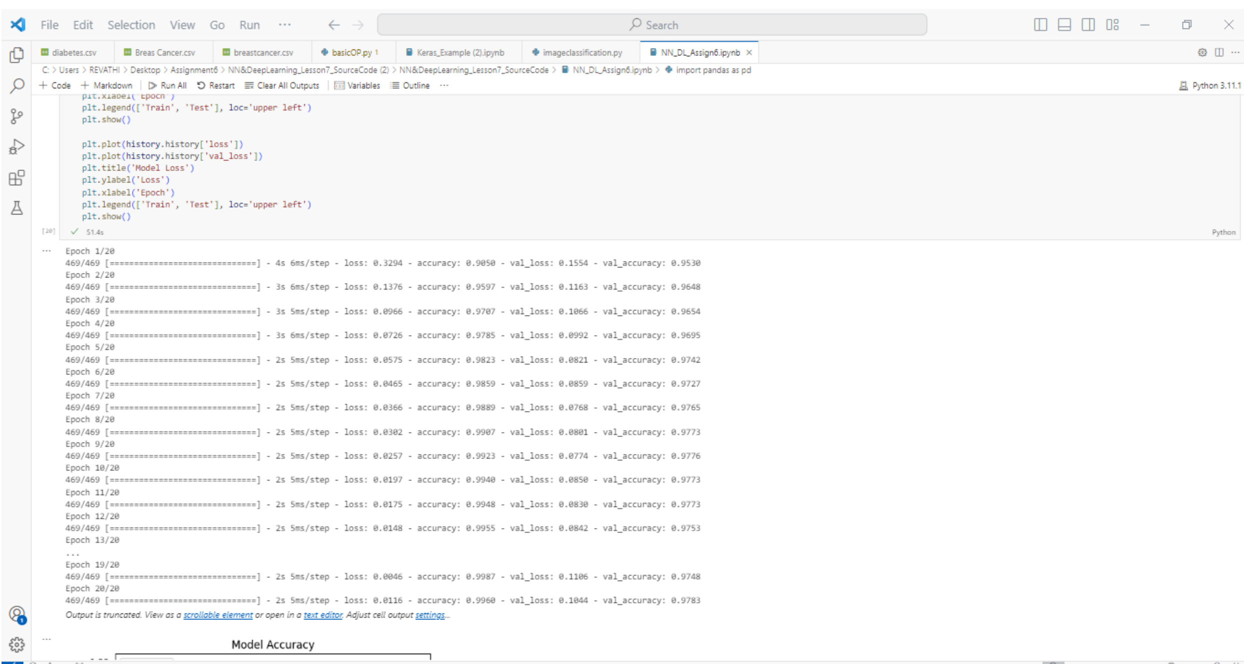


In class programming: Use Image Classification on the hand written digits data set (mnist)

1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.
2. 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.
3. 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.

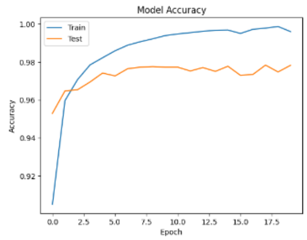
4. Run the same code without scaling the images and check the performance?

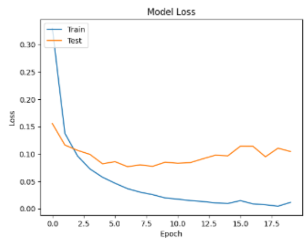




```
...
Epoch 19/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0046 - accuracy: 0.9987 - val_loss: 0.1106 - val_accuracy: 0.9748
Epoch 20/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0116 - accuracy: 0.9960 - val_loss: 0.1044 - val_accuracy: 0.9783
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...






```python
import keras
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
```

[21] 0.0s

```python
# load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# reshape the data to a 1D array of pixels
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

# convert data type to float32 and normalize the data to a range between 0 and 1
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# convert labels to categorical one-hot encoding
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# define the model architecture
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

# print the model summary
model.summary()

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

# train the model
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=20,
                    verbose=1,
                    validation_data=(x_test, y_test))

# plot the training and validation loss and accuracy
plt.figure(figsize=(10, 5))
```

File Edit Selection View Go Run ··· Search

diabetes.csv | Breas Cancer.csv | breastcancer.csv | basicOP.py 1 | Keras_Example (2).ipynb | imageclassification.py | NN_DL_Assign6.ipynb

C: > Users > REVATHI > Desktop > Assignment6 > NN&DeepLearning_Lesson7_SourceCode (2) > NN&DeepLearning_Lesson7_SourceCode > NN_DL_Assign6.ipynb > import pandas as pd

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· Python 3.11.1

```python
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

# print the model summary
model.summary()

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

# train the model
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=20,
                    verbose=1,
                    validation_data=(x_test, y_test))

# plot the training and validation loss and accuracy
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# select a random image from the test data
idx = np.random.randint(x_test.shape[0])
image = x_test[idx].reshape(28, 28)

# plot the selected image
plt.figure()
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.title('Selected Image')

# do inferencing to check the model prediction on the selected image
prediction = model.predict(image.reshape(1, 784))
prediction = np.argmax(prediction)

# print the predicted label
print('Predicted label:', prediction)
```

[22] 2m 43.5s Python

File Edit Selection View Go Run ··· Search

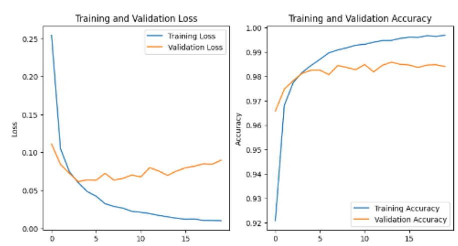
diabetes.csv | Breas Cancer.csv | breastcancer.csv | basicOP.py 1 | Keras_Example (2).ipynb | imageclassification.py | NN_DL_Assign6.ipynb

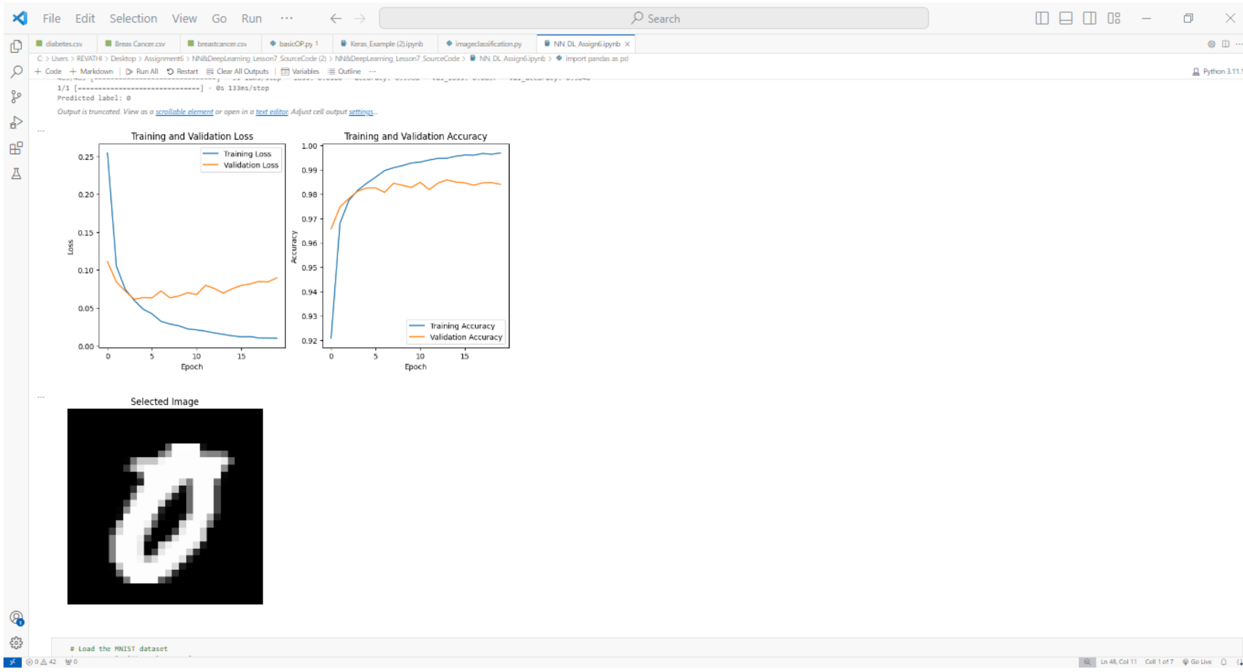
C: > Users > REVATHI > Desktop > Assignment6 > NN&DeepLearning_Lesson7_SourceCode (2) > NN&DeepLearning_Lesson7_SourceCode > NN_DL_Assign6.ipynb > import pandas as pd

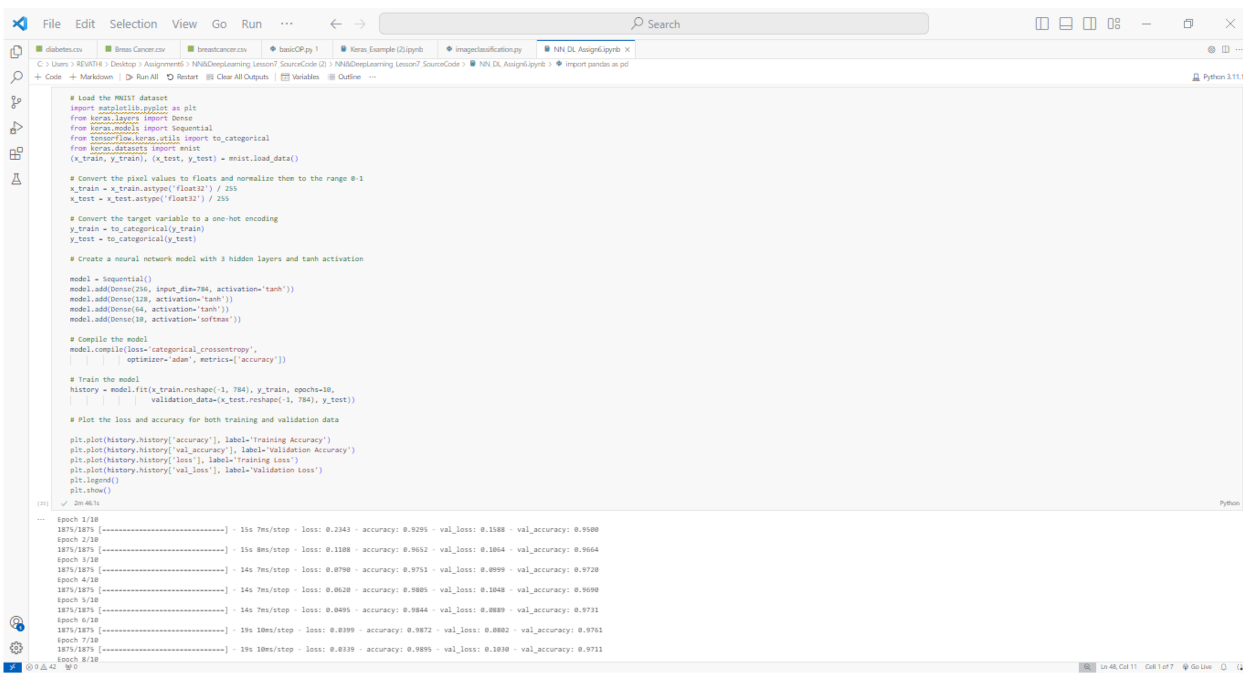
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· Python 3.11.1

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_50 (Dense)            (None, 512)               401920

 dropout_2 (Dropout)         (None, 512)               0

 dense_51 (Dense)            (None, 512)               262656

 dropout_3 (Dropout)         (None, 512)               0

 dense_52 (Dense)            (None, 10)                5130

=================================================================
Total params: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 10s 18ms/step - loss: 0.2544 - accuracy: 0.9209 - val_loss: 0.1110 - val_accuracy: 0.9657
Epoch 2/20
469/469 [==============================] - 8s 17ms/step - loss: 0.1050 - accuracy: 0.9680 - val_loss: 0.0842 - val_accuracy: 0.9747
Epoch 3/20
469/469 [==============================] - 8s 18ms/step - loss: 0.0745 - accuracy: 0.9774 - val_loss: 0.0723 - val_accuracy: 0.9782
...
Epoch 20/20
469/469 [==============================] - 9s 18ms/step - loss: 0.0100 - accuracy: 0.9968 - val_loss: 0.0897 - val_accuracy: 0.9840
1/1 [==============================] - 0s 133ms/step
Predicted label: 0
```

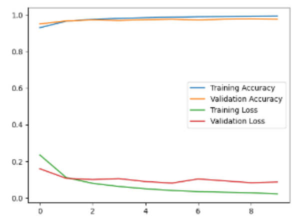
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...




1/1 [==============================] - 0s 133ms/step
Predicted label: 0
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch
Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
Selected Image
Load the MNIST dataset


# Load the MNIST dataset
import matplotlib.pyplot as plt
from keras.layers import Dense
from keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Convert the pixel values to floats and normalize them to the range 0-1
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
# Convert the target variable to a one-hot encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
# Create a neural network model with 3 hidden layers and tanh activation
model = Sequential()
model.add(Dense(256, input_dim=784, activation='tanh'))
model.add(Dense(128, activation='tanh'))
model.add(Dense(64, activation='tanh'))
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.reshape(-1, 784), y_train, epochs=10,
validation_data=(x_test.reshape(-1, 784), y_test))
# Plot the loss and accuracy for both training and validation data
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
2m 46.1s
Epoch 1/10
1875/1875 [==============================] - 15s 7ms/step - loss: 0.2343 - accuracy: 0.9295 - val_loss: 0.1588 - val_accuracy: 0.9500
Epoch 2/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.1108 - accuracy: 0.9652 - val_loss: 0.1064 - val_accuracy: 0.9664
Epoch 3/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0790 - accuracy: 0.9751 - val_loss: 0.0999 - val_accuracy: 0.9720
Epoch 4/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0620 - accuracy: 0.9805 - val_loss: 0.1048 - val_accuracy: 0.9690
Epoch 5/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0495 - accuracy: 0.9844 - val_loss: 0.0889 - val_accuracy: 0.9731
Epoch 6/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0399 - accuracy: 0.9872 - val_loss: 0.0802 - val_accuracy: 0.9761
Epoch 7/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0339 - accuracy: 0.9895 - val_loss: 0.1030 - val_accuracy: 0.9711
Epoch 8/10

```
Epoch 1/10
1875/1875 [==============================] - 15s 7ms/step - loss: 0.2343 - accuracy: 0.9295 - val_loss: 0.1588 - val_accuracy: 0.9500
Epoch 2/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.1108 - accuracy: 0.9652 - val_loss: 0.1064 - val_accuracy: 0.9664
Epoch 3/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0790 - accuracy: 0.9751 - val_loss: 0.0999 - val_accuracy: 0.9720
Epoch 4/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0620 - accuracy: 0.9805 - val_loss: 0.1048 - val_accuracy: 0.9690
Epoch 5/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0495 - accuracy: 0.9844 - val_loss: 0.0889 - val_accuracy: 0.9731
Epoch 6/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0399 - accuracy: 0.9872 - val_loss: 0.0802 - val_accuracy: 0.9761
Epoch 7/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0339 - accuracy: 0.9895 - val_loss: 0.1030 - val_accuracy: 0.9711
Epoch 8/10
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0303 - accuracy: 0.9904 - val_loss: 0.0926 - val_accuracy: 0.9764
Epoch 9/10
1875/1875 [==============================] - 23s 12ms/step - loss: 0.0273 - accuracy: 0.9913 - val_loss: 0.0823 - val_accuracy: 0.9776
Epoch 10/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0216 - accuracy: 0.9928 - val_loss: 0.0862 - val_accuracy: 0.9760
```




```python
# Load the MNIST dataset
import matplotlib.pyplot as plt
from keras.layers import Dense
from keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Flatten the input images
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

# Convert the target variable to a one-hot encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Create a neural network model with 2 hidden layers and Relu activation

model = Sequential()
model.add(Dense(256, input_dim=784, activation='relu'))
model.add(Dense(128, 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_data=(x_test, y_test))

# Plot the loss and accuracy for both training and validation data

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
1875/1875 [==============================] - 16s 8ms/step - loss: 1.3584 - accuracy: 0.8887 - val_loss: 0.3575 - val_accuracy: 0.9129
Epoch 2/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.2403 - accuracy: 0.9376 - val_loss: 0.2320 - val_accuracy: 0.9389
Epoch 3/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1763 - accuracy: 0.9517 - val_loss: 0.1875 - val_accuracy: 0.9491
Epoch 4/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1557 - accuracy: 0.9569 - val_loss: 0.1747 - val_accuracy: 0.9556
Epoch 5/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1358 - accuracy: 0.9620 - val_loss: 0.1862 - val_accuracy: 0.9518
Epoch 6/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1231 - accuracy: 0.9668 - val_loss: 0.1622 - val_accuracy: 0.9587
Epoch 7/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1091 - accuracy: 0.9717 - val_loss: 0.1540 - val_accuracy: 0.9617
Epoch 8/10
```

```python
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
1875/1875 [==============================] - 16s 8ms/step - loss: 1.3584 - accuracy: 0.8887 - val_loss: 0.3575 - val_accuracy: 0.9129
Epoch 2/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.2403 - accuracy: 0.9376 - val_loss: 0.2320 - val_accuracy: 0.9389
Epoch 3/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1763 - accuracy: 0.9517 - val_loss: 0.1875 - val_accuracy: 0.9491
Epoch 4/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1557 - accuracy: 0.9569 - val_loss: 0.1747 - val_accuracy: 0.9556
Epoch 5/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1358 - accuracy: 0.9620 - val_loss: 0.1862 - val_accuracy: 0.9518
Epoch 6/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1231 - accuracy: 0.9668 - val_loss: 0.1622 - val_accuracy: 0.9587
Epoch 7/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1091 - accuracy: 0.9717 - val_loss: 0.1540 - val_accuracy: 0.9617
Epoch 8/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1054 - accuracy: 0.9725 - val_loss: 0.1408 - val_accuracy: 0.9679
Epoch 9/10
1875/1875 [==============================] - 12s 7ms/step - loss: 0.0919 - accuracy: 0.9753 - val_loss: 0.1598 - val_accuracy: 0.9643
Epoch 10/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0862 - accuracy: 0.9771 - val_loss: 0.1197 - val_accuracy: 0.9712
```