

# NNDL: ICP4

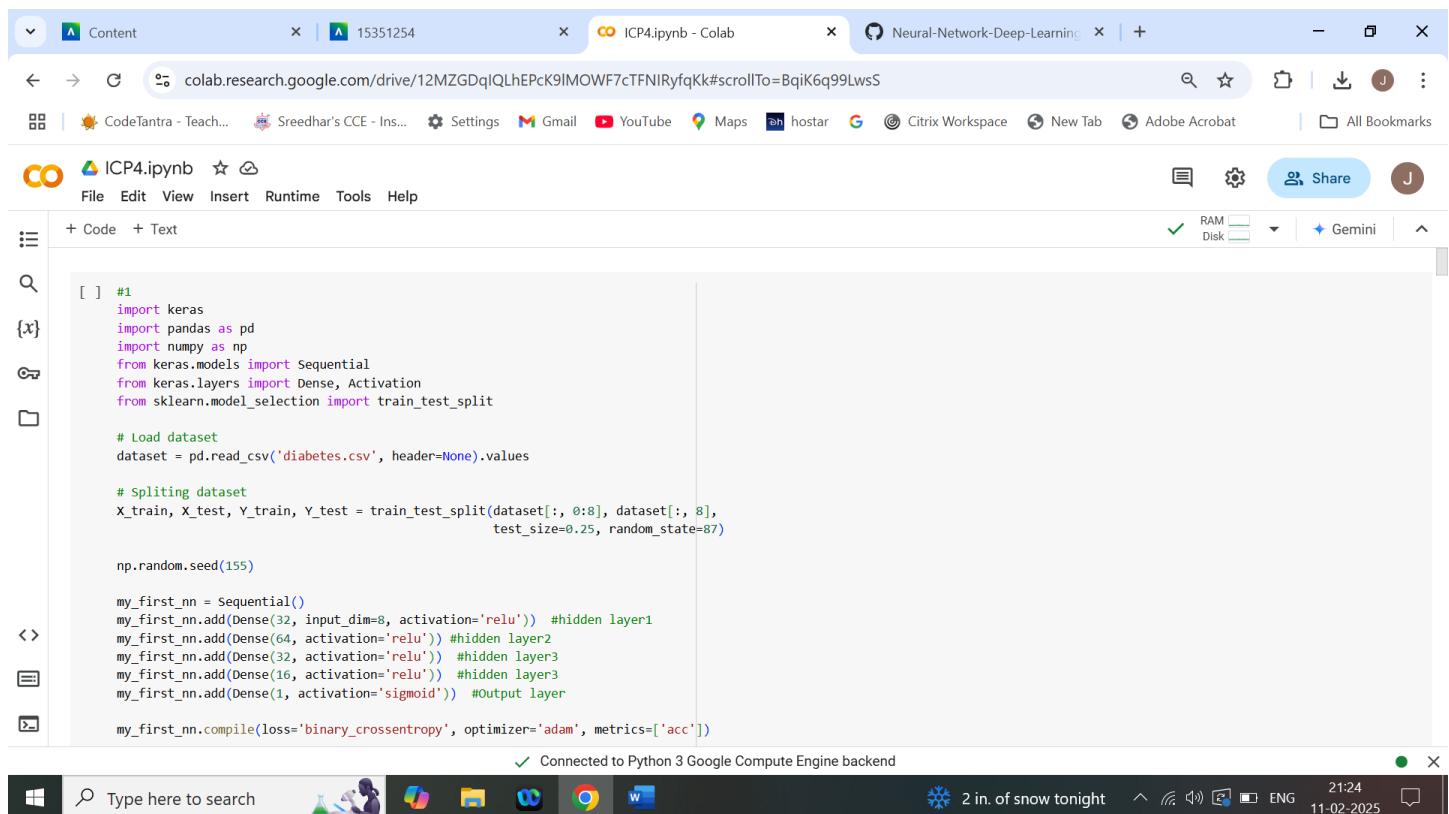
Jyothi Kiran Boddeda  
700769023

**GITHUB LINK :** <https://github.com/jyothikiranboddeda/Neural-Network-Deep-Learning.git>

**VIDEO LINK :**

[https://drive.google.com/file/d/1ifu6zSTop0harDN7IHqeI7e\\_WgKrPaGz/view?usp=sharing](https://drive.google.com/file/d/1ifu6zSTop0harDN7IHqeI7e_WgKrPaGz/view?usp=sharing)

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



The screenshot shows a Google Colab interface with the following details:

- Header:** Content, 15351254, ICP4.ipynb - Colab, Neural-Network-Deep-Learning.
- Toolbar:** Back, Forward, Refresh, CodeTantra - Teach..., Sreedhar's CCE - Ins..., Settings, Gmail, YouTube, Maps, hostar, Citrix Workspace, New Tab, Adobe Acrobat, All Bookmarks.
- File Menu:** File, Edit, View, Insert, Runtime, Tools, Help.
- Code Editor:** The code cell contains Python code for loading a diabetes dataset and creating a sequential neural network model with four hidden layers and one output layer. The code uses Keras, Pandas, NumPy, and Scikit-learn libraries.

```
[ ] #1
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.model_selection import train_test_split

# Load dataset
dataset = pd.read_csv('diabetes.csv', header=None).values

# Splitting dataset
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()
my_first_nn.add(Dense(32, input_dim=8, activation='relu')) #hidden layer1
my_first_nn.add(Dense(64, activation='relu')) #hidden layer2
my_first_nn.add(Dense(32, activation='relu')) #hidden layer3
my_first_nn.add(Dense(16, activation='relu')) #hidden layer3
my_first_nn.add(Dense(1, activation='sigmoid')) #Output layer

my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])

✓ Connected to Python 3 Google Compute Engine backend
```

- Bottom Bar:** Type here to search, system icons (calculator, file, etc.), and system status (2 in. of snow tonight, ENG, 21:24, 11-02-2025).

Content 15351254 ICP4.ipynb - Colab Neural-Network-Deep-Learning

```
[ ] # Splitting dataset
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()
my_first_nn.add(Dense(32, input_dim=8, activation='relu')) #hidden layer1
my_first_nn.add(Dense(64, activation='relu')) #hidden layer2
my_first_nn.add(Dense(32, activation='relu')) #hidden layer3
my_first_nn.add(Dense(16, activation='relu')) #hidden layer3
my_first_nn.add(Dense(1, activation='sigmoid')) #output layer

my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])

my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0, validation_data=(X_test, y_test))

print(my_first_nn.summary())

# Evaluating model performance
test_loss, test_acc = my_first_nn.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")

18/18 - 0s 6ms/step - acc: 0.7825 - loss: 0.4649 - val_acc: 0.6823 - val_loss: 0.6881
Epoch 84/100
18/18 - 0s 6ms/step - acc: 0.7650 - loss: 0.4688 - val_acc: 0.6979 - val_loss: 0.6029
```

Connected to Python 3 Google Compute Engine backend

Content 15351254 ICP4.ipynb - Colab Neural-Network-Deep-Learning

```
[ ] Epoch 98/100
18/18 - 0s 8ms/step - acc: 0.7796 - loss: 0.4390 - val_acc: 0.6927 - val_loss: 0.6219
Epoch 99/100
18/18 - 0s 6ms/step - acc: 0.7893 - loss: 0.4387 - val_acc: 0.6979 - val_loss: 0.6468
Epoch 100/100
18/18 - 0s 6ms/step - acc: 0.7789 - loss: 0.4518 - val_acc: 0.6927 - val_loss: 0.6410
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 32)	288
dense_5 (Dense)	(None, 64)	2,112
dense_6 (Dense)	(None, 32)	2,080
dense_7 (Dense)	(None, 16)	528
dense_8 (Dense)	(None, 1)	17

```
Total params: 15,077 (58.90 KB)
Trainable params: 5,025 (19.63 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 10,052 (39.27 KB)
None
6/6 - 0s 6ms/step - acc: 0.6839 - loss: 0.6653
Test Accuracy: 0.6927
```

Connected to Python 3 Google Compute Engine backend

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

```
#2
import keras
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# Load Breast Cancer dataset
data = load_breast_cancer()
X = data.data
Y = data.target

# Split data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)

np.random.seed(155)
model = Sequential()
model.add(Dense(32, input_dim=X.shape[1], activation='relu')) #hidden layer1
model.add(Dense(64, activation='relu')) #hidden layer2
model.add(Dense(32, activation='relu')) #hidden layer3
model.add(Dense(16, activation='relu')) #hidden layer4
model.add(Dense(1, activation='sigmoid')) #Output Layer

# Compile Model
```

Content 15351254 ICP4.ipynb - Colab Neural-Network-Deep-Learning

```
# Compile Model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])

model_fitted = model.fit(X_train, Y_train, epochs=100, initial_epoch=0, validation_data=(X_test, Y_test))
print(model.summary())

# Evaluate Model Accuracy
test_loss, test_acc = model.evaluate(X_test, Y_test)
print(f'Test Accuracy on Breast Cancer Dataset: {test_acc:.4f}')

14/14 ━━━━━━ 0s 8ms/step - acc: 0.9186 - loss: 0.1950 - val_acc: 0.8531 - val_loss: 0.3292
Epoch 76/100
14/14 ━━━━ 0s 11ms/step - acc: 0.9328 - loss: 0.1969 - val_acc: 0.8881 - val_loss: 0.2802
Epoch 77/100
14/14 ━━━━ 0s 9ms/step - acc: 0.9389 - loss: 0.1787 - val_acc: 0.8811 - val_loss: 0.3822
Epoch 78/100
14/14 ━━━━ 0s 8ms/step - acc: 0.9189 - loss: 0.2110 - val_acc: 0.8881 - val_loss: 0.3808
Epoch 79/100
14/14 ━━━━ 0s 7ms/step - acc: 0.9003 - loss: 0.2657 - val_acc: 0.8531 - val_loss: 0.3287
Epoch 80/100
14/14 ━━━━ 0s 7ms/step - acc: 0.9466 - loss: 0.1715 - val_acc: 0.9091 - val_loss: 0.3150
Epoch 81/100
14/14 ━━━━ 0s 8ms/step - acc: 0.9227 - loss: 0.1972 - val_acc: 0.9091 - val_loss: 0.3190
Epoch 82/100
14/14 ━━━━ 0s 8ms/step - acc: 0.9114 - loss: 0.1920 - val_acc: 0.8881 - val_loss: 0.2670
Epoch 93/100
```

Connected to Python 3 Google Compute Engine backend

Type here to search 2 in. of snow Wed 21:24 11-02-2025

Content 15351254 ICP4.ipynb - Colab Neural-Network-Deep-Learning

```
14/14 ━━━━ 0s 10ms/step - acc: 0.9291 - loss: 0.2004 - val_acc: 0.8601 - val_loss: 0.3155
Epoch 98/100
14/14 ━━━━ 0s 10ms/step - acc: 0.8193 - loss: 0.4797 - val_acc: 0.7133 - val_loss: 0.7859
Epoch 99/100
14/14 ━━━━ 0s 10ms/step - acc: 0.8523 - loss: 0.4111 - val_acc: 0.9021 - val_loss: 0.2649
Epoch 100/100
14/14 ━━━━ 0s 8ms/step - acc: 0.9101 - loss: 0.2907 - val_acc: 0.8462 - val_loss: 0.5071
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 32)	992
dense_10 (Dense)	(None, 64)	2,112
dense_11 (Dense)	(None, 32)	2,080
dense_12 (Dense)	(None, 16)	528
dense_13 (Dense)	(None, 1)	17

Total params: 17,189 (67.15 KB)  
Trainable params: 5,729 (22.38 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 11,460 (44.77 KB)

None  
5/5 ━━━━ 0s 11ms/step - acc: 0.8437 - loss: 0.5510  
Test Accuracy on Breast Cancer Dataset: 0.8462

Connected to Python 3 Google Compute Engine backend

Type here to search 2 in. of snow Wed 21:24 11-02-2025

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()
```

The screenshot shows a Google Colab interface. At the top, there are tabs for 'Content', '15351254', and 'ICP4.ipynb - Colab'. Below the tabs is a toolbar with various icons. The main area is a Jupyter notebook titled 'ICP4.ipynb'. The code in the notebook is:

```
[ ] #3  
import keras  
import numpy as np  
import pandas as pd  
from keras.models import Sequential  
from keras.layers import Dense, Activation  
from sklearn.datasets import load_breast_cancer  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import StandardScaler  
  
# Load Breast Cancer dataset  
data = load_breast_cancer()  
X = data.data  
Y = data.target  
  
# Split data  
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)  
  
# Normalize data  
sc = StandardScaler()  
X_train = sc.fit_transform(X_train)  
X_test = sc.transform(X_test)  
  
np.random.seed(155)  
model = Sequential()  
model.add(Dense(32, input_dim=X.shape[1], activation='relu')) #hidden layer1  
model.add(Dense(16, activation='relu')) #hidden layer2
```

The status bar at the bottom indicates "Connected to Python 3 Google Compute Engine backend". The taskbar at the very bottom shows the Windows logo, a search bar, and several pinned icons.

Content 15351254 ICP4.ipynb - Colab Neural-Network-Deep-Learning

# split data  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=87)  
  
# Normalize data  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)  
  
np.random.seed(155)  
model = Sequential()  
model.add(Dense(32, input\_dim=X.shape[1], activation='relu')) #hidden layer1  
model.add(Dense(64, activation='relu')) #hidden layer2  
model.add(Dense(32, activation='relu')) #hidden layer3  
model.add(Dense(16, activation='relu')) #hidden layer4  
model.add(Dense(1, activation='sigmoid')) #output layer  
  
model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['acc'])  
model\_fitted = model.fit(X\_train, Y\_train, epochs=100, initial\_epoch=0, validation\_data=(X\_test, Y\_test))  
  
print(model.summary())  
  
# Evaluate Model Accuracy  
test\_loss, test\_acc = model.evaluate(X\_test, Y\_test)  
print(f"Test Accuracy on Breast Cancer Dataset (with Normalization): {test\_acc:.4f}")

Connected to Python 3 Google Compute Engine backend

Type here to search 21:24 11-02-2025

Content 15351254 ICP4.ipynb - Colab Neural-Network-Deep-Learning

Epoch: 99/100  
14/14 0s 8ms/step - acc: 1.0000 - loss: 4.3094e-05 - val\_acc: 0.9720 - val\_loss: 0.2843  
Epoch 100/100  
14/14 0s 7ms/step - acc: 1.0000 - loss: 4.9047e-05 - val\_acc: 0.9720 - val\_loss: 0.2856  
Model: "sequential\_4"

Layer (type)	Output Shape	Param #
dense_14 (Dense)	(None, 32)	992
dense_15 (Dense)	(None, 64)	2,112
dense_16 (Dense)	(None, 32)	2,080
dense_17 (Dense)	(None, 16)	528
dense_18 (Dense)	(None, 1)	17

Total params: 17,189 (67.15 KB)  
Trainable params: 5,729 (22.38 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 11,460 (44.77 KB)  
None  
5/5 0s 9ms/step - acc: 0.9646 - loss: 0.5017  
Test Accuracy on Breast Cancer Dataset (with Normalization): 0.9720

[ ] #1

Connected to Python 3 Google Compute Engine backend

Type here to search 21:25 11-02-2025

## 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.

The screenshot shows a Google Colab interface with a Jupyter notebook titled 'ICP4.ipynb'. The code cell contains Python code for plotting training and validation accuracy and loss over epochs. The code uses the 'matplotlib.pyplot' library to create two subplots: one for accuracy and one for loss. The accuracy plot shows 'Training Accuracy' (blue line) increasing from approximately 0.93 to 0.99, and 'Validation Accuracy' (orange line) fluctuating between 0.95 and 0.98. The loss plot shows 'Training Loss' (blue line) decreasing from approximately 0.25 to 0.03, and 'Validation Loss' (orange line) fluctuating between 0.09 and 0.12. The notebook is connected to a Python 3 Google Compute Engine backend.

```
[ ] #1
import matplotlib.pyplot as plt

# Plot Training & Validation Accuracy and Loss
plt.figure(figsize=(12, 5))

# Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss Plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



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.

The screenshot shows a Google Colab notebook titled "ICP4.ipynb". The code cell contains Python code for plotting a sample test image and performing inference on it. The output cell displays the title "Sample Test Image" above a blacked-out image placeholder. The status bar at the bottom indicates "Connected to Python 3 Google Compute Engine backend".

```
[ ] #2
import numpy as np
import matplotlib.pyplot as plt

index = 0

# Plot the selected image
plt.imshow(X_test[index], cmap='gray')
plt.title("Sample Test Image")
plt.axis('off')
plt.show()

# Reshape image to match the input shape
image_reshaped = X_test[index].reshape(1, 28, 28)

prediction = np.argmax(model.predict(image_reshaped), axis=-1)

# Print predicted label
print(f"Model Prediction: {prediction[0]}")
```

Sample Test Image

Connected to Python 3 Google Compute Engine backend

The screenshot shows the same Google Colab notebook. The output cell now displays a grayscale image of a handwritten digit '7' on a black background. Below the image, a progress bar shows "1/1" and "0s 38ms/step", and the text "Model Prediction: 7". The status bar at the bottom indicates "Connected to Python 3 Google Compute Engine backend".

Sample Test Image

1/1 0s 38ms/step

Model Prediction: 7

Connected to Python 3 Google Compute Engine backend

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

The screenshot shows a Google Colab notebook titled 'ICP4.ipynb'. The code cell contains Python code for building a neural network. It imports necessary libraries like numpy, plt, Sequential, Dense, Flatten, and to\_categorical from tensorflow. It loads the MNIST dataset and preprocesses it by scaling the images to 255.0. The code then defines activation functions ('tanh', 'sigmoid') and hidden layers (2, 3). It prints a message for each training configuration and creates a Sequential model with a Flatten layer followed by two Dense layers with 128 units each and the specified activation function.

```
[ ] #3
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load and preprocess the MNIST dataset
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train = X_train / 255.0
X_test = X_test / 255.0
Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)

activations = ['tanh', 'sigmoid'] #activation functions
hidden_layers = [2, 3]

for activation in activations:
    for num_layers in hidden_layers:
        print(f"\nTraining with {num_layers} hidden layers and {activation} activation...\n")

        model = Sequential()
        model.add(Flatten(input_shape=(28, 28)))
        for _ in range(num_layers):
            model.add(Dense(128, activation=activation))
```

Connected to Python 3 Google Compute Engine backend

The screenshot shows the same Google Colab notebook 'ICP4.ipynb'. The code cell now includes training and evaluation steps. After defining the model and its layers, it compiles the model with Adam optimizer, categorical crossentropy loss, and accuracy metric. It then fits the model to the training data for 10 epochs with a batch size of 32. The test loss and accuracy are evaluated. Finally, it plots training and validation accuracy and loss over the 10 epochs.

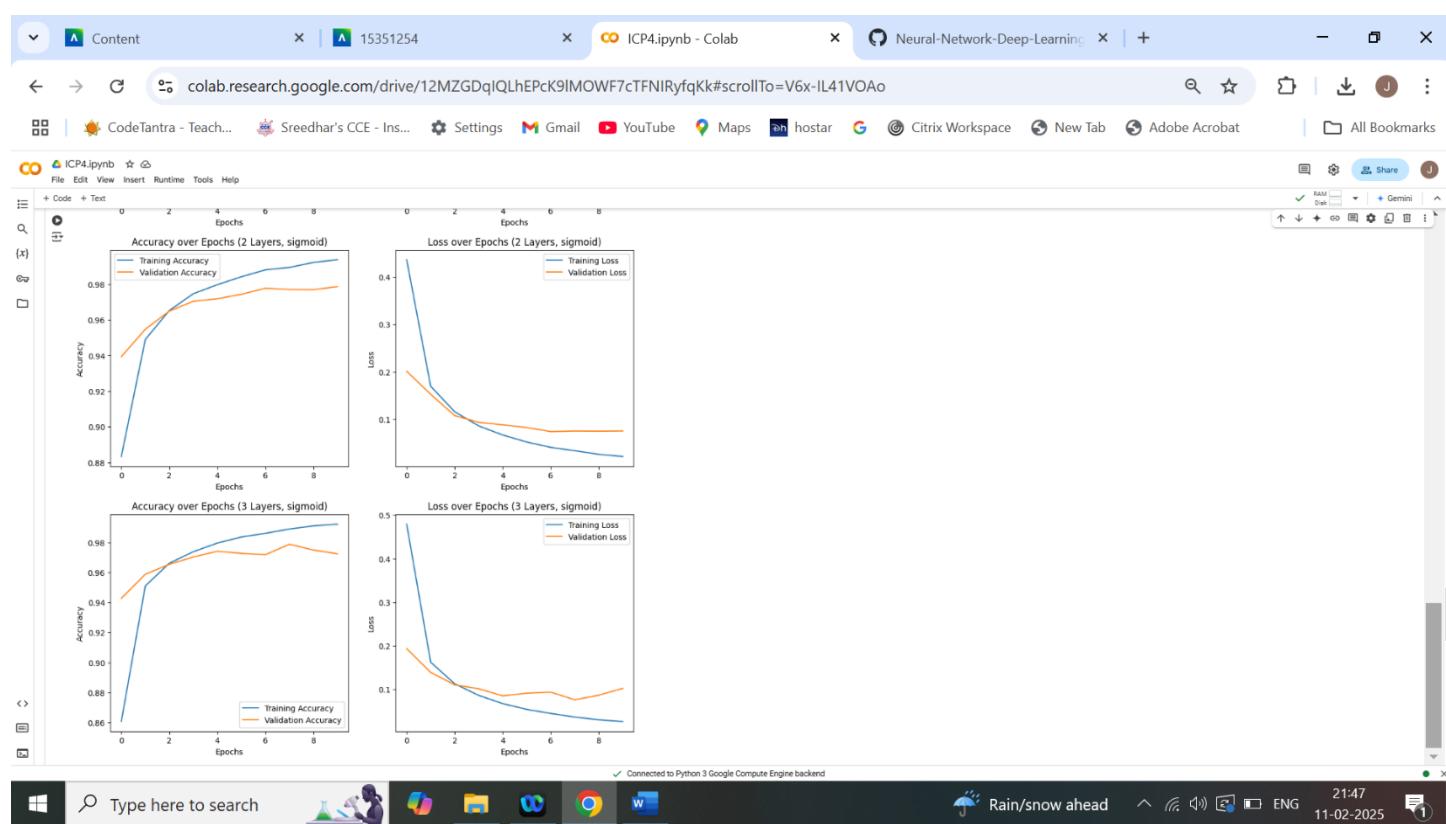
```
[ ] model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_size=32)
test_loss, test_acc = model.evaluate(X_test, Y_test)
print(f"Test Accuracy with {num_layers} hidden layers and {activation} activation: {test_acc:.4f}")

# Plot Loss and Accuracy
plt.figure(figsize=(12, 5))

# Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title(f'Accuracy over Epochs ({num_layers} Layers, {activation})')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss Plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title(f'Loss over Epochs ({num_layers} Layers, {activation})')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

Connected to Python 3 Google Compute Engine backend



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

The screenshot shows a Google Colab interface with a Jupyter notebook titled "ICP4.ipynb". The code cell contains Python code for loading the MNIST dataset, defining a neural network architecture, and training it. The code includes comments explaining the steps: not scaling the images, using one-hot encoding for labels, and specifying activation functions and hidden layers. The runtime configuration shows "Connected to Python 3 Google Compute Engine backend".

```
#4
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load the MNIST dataset without scaling
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

# Do not scale the images (no normalization)
# X_train = X_train / 255.0
# X_test = X_test / 255.0

# Convert labels to categorical (one-hot encoding)
Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)

activations = ['tanh', 'sigmoid'] # activation functions
hidden_layers = [2, 3]

for activation in activations:
    for num_layers in hidden_layers:
        print(f"\nTraining with {num_layers} hidden layers and {activation} activation")
        model = Sequential()
        model.add(Flatten(input_shape=(28, 28))) # Flatten input images
        for _ in range(num_layers):
            model.add(Dense(10, activation="softmax"))
        model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])
        history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_size=32)
        test_loss, test_acc = model.evaluate(X_test, Y_test)
        print(f"Test Accuracy with {num_layers} hidden layers and {activation} activation: {test_acc:.4f}")

        # Plot Loss and Accuracy
        plt.figure(figsize=(12, 5))

        # Accuracy Plot
        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'], label='Training Accuracy')
        plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
        plt.title(f'Accuracy over Epochs ({num_layers} Layers, {activation})')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()

        # Loss Plot
        plt.subplot(1, 2, 2)
        plt.plot(history.history['loss'], label='Training Loss')
        plt.plot(history.history['val_loss'], label='Validation Loss')
        plt.title(f'Loss over Epochs ({num_layers} Layers, {activation})')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()

        plt.show()
```

The screenshot shows the continuation of the Google Colab interface with the same Jupyter notebook "ICP4.ipynb". The code cell now includes the plotting logic for accuracy and loss over 10 epochs. The plots show the training and validation accuracy and loss for different numbers of hidden layers and activation functions. The runtime configuration again shows "Connected to Python 3 Google Compute Engine backend".

Content 15351254 ICP4.ipynb - Colab Neural-Network-Deep-Learning

colab.research.google.com/drive/12MZGDqIQLhEPcK9IMOWF7cTFNIRyfqKk#scrollTo=67DTI8x2WxT\_

CodeTantra - Teach... Sreedhar's CCE - Ins... Settings Gmail YouTube Maps hostar Citrix Workspace New Tab Adobe Acrobat All Bookmarks

**ICP4.ipynb** File Edit View Insert Runtime Tools Help Share RAM Disk Gemini

313/313 1s 2ms/step - accuracy: 0.9080 - loss: 0.2909  
Test Accuracy with 2 hidden layers and tanh activation: 0.9201

Accuracy over Epochs (2 Layers, tanh)

Training Accuracy Validation Accuracy

Epochs

Loss over Epochs (2 Layers, tanh)

Training Loss Validation Loss

Epochs

Training with 3 hidden layers and tanh activation...

```
Epoch 1/10
1875/1875 11s 5ms/step - accuracy: 0.7728 - loss: 0.7083 - val_accuracy: 0.8663 - val_loss: 0.4142
Epoch 2/10
1875/1875 10s 5ms/step - accuracy: 0.8575 - loss: 0.4376 - val_accuracy: 0.8808 - val_loss: 0.3780
Epoch 3/10
1875/1875 10s 5ms/step - accuracy: 0.8847 - loss: 0.3568 - val_accuracy: 0.9012 - val_loss: 0.3016
Epoch 4/10
1875/1875 9s 5ms/step - accuracy: 0.9010 - loss: 0.3061 - val_accuracy: 0.8971 - val_loss: 0.3178
Epoch 5/10
```

Connected to Python 3 Google Compute Engine backend

Type here to search Precip on Saturday 21:55 11-02-2025

Content 15351254 ICP4.ipynb - Colab Neural-Network-Deep-Learning

colab.research.google.com/drive/12MZGDqIQLhEPcK9IMOWF7cTFNIRyfqKk#scrollTo=67DTI8x2WxT\_

CodeTantra - Teach... Sreedhar's CCE - Ins... Settings Gmail YouTube Maps hostar Citrix Workspace New Tab Adobe Acrobat All Bookmarks

**ICP4.ipynb** File Edit View Insert Runtime Tools Help Share RAM Disk Gemini

313/313 1s 2ms/step - accuracy: 0.9341 - loss: 0.2109 - val\_accuracy: 0.9310 - val\_loss: 0.2154  
313/313 1s 2ms/step - accuracy: 0.9283 - loss: 0.2442  
Test Accuracy with 2 hidden layers and sigmoid activation: 0.9310

Accuracy over Epochs (2 Layers, sigmoid)

Training Accuracy Validation Accuracy

Epochs

Loss over Epochs (2 Layers, sigmoid)

Training Loss Validation Loss

Epochs

Training with 3 hidden layers and sigmoid activation...

```
Epoch 1/10
1875/1875 12s 6ms/step - accuracy: 0.7082 - loss: 0.9730 - val_accuracy: 0.8686 - val_loss: 0.4076
Epoch 2/10
1875/1875 10s 5ms/step - accuracy: 0.8797 - loss: 0.3880 - val_accuracy: 0.8916 - val_loss: 0.3527
Epoch 3/10
1875/1875 10s 5ms/step - accuracy: 0.8917 - loss: 0.3492 - val_accuracy: 0.9086 - val_loss: 0.2926
Epoch 4/10
```

Connected to Python 3 Google Compute Engine backend

Type here to search Precip on Saturday 21:55 11-02-2025



Type here to search Connected to Python 3 Google Compute Engine backend ENG 21:55 11-02-2025

