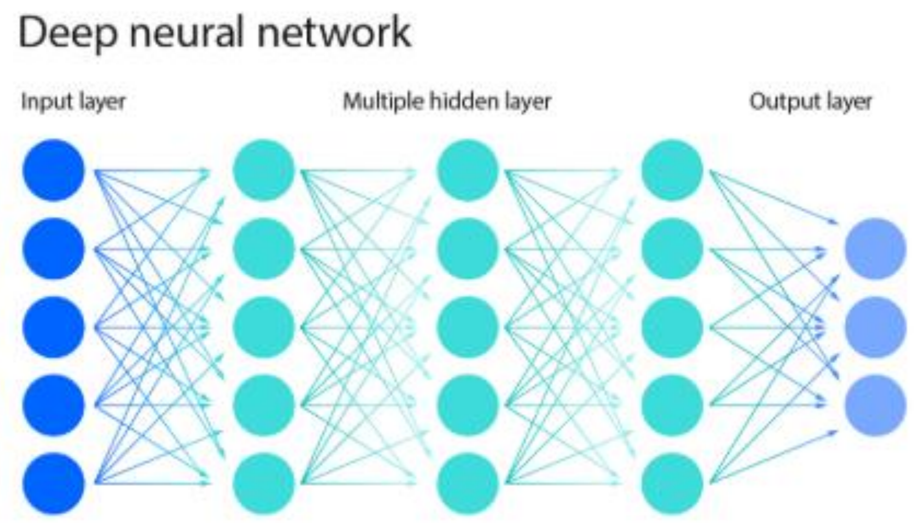
LAB No. 6

Implementation of Deep Neural Network

In this lab, students will study and implement a **Deep Neural Network (DNN)** for classification tasks. A DNN is an extension of Artificial Neural Networks that contains **multiple hidden layers**, enabling the model to learn complex and hierarchical patterns from data. Students will begin with a simple DNN on a small dataset to understand its structure and training process, and then apply DNN models to real-world datasets such as **Iris** and **MNIST**. Model performance will be evaluated using accuracy metrics.

Introduction

A **Deep Neural Network (DNN)** is a neural network with **more than one hidden layer** between the input and output layers. Each layer extracts increasingly complex features from the data. DNNs use **backpropagation** and **gradient descent–based optimizers** to update weights and minimize loss.



Key Components of DNN:

- **Input Layer** – receives raw data
- **Multiple Hidden Layers** – perform deep feature learning
- **Output Layer** – produces final prediction
- **Activation Functions** – ReLU, Sigmoid, Softmax
- **Loss Function** – measures prediction error
- **Optimizer** – Adam, SGD

DNNs are widely used in applications such as image recognition, speech processing, recommendation systems, and natural language processing.

Solved Examples

Example 1

Build a Deep Neural Network to predict whether a student **passes or fails** based on study hours and attendance **(Small Dataset)**

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Dataset
data = {
    'StudyHours': ['Low', 'High', 'High', 'Low', 'High'],
    'Attendance': ['Poor', 'Good', 'Poor', 'Good', 'Good'],
    'Result': ['Fail', 'Pass', 'Pass', 'Fail', 'Pass']
}

df = pd.DataFrame(data)

# Encode categorical data
encoder = LabelEncoder()
for col in df.columns:
    df[col] = encoder.fit_transform(df[col])

X = df[['StudyHours', 'Attendance']].values
y = df['Result'].values

# Build DNN
model = Sequential()
model.add(Dense(8, activation='relu', input_shape=(2,)))
model.add(Dense(6, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile and train
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X, y, epochs=150, verbose=0)
```

```
# Prediction
prediction = model.predict([[1, 1]])
print("Predicted Result (Pass=1, Fail=0):", int(prediction[0][0] > 0.5))
```

The multiple hidden layers enable the DNN to learn deeper patterns compared to a shallow ANN

Example 2:

Apply a Deep Neural Network to classify Iris flowers into three species.

Solution:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# One-hot encoding
y = to_categorical(y)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Build DNN
model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(4,)))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Compile and train
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=150, verbose=0)

# Evaluate
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy:", accuracy)
```

The deeper architecture improves feature representation and classification accuracy.

Example 3:

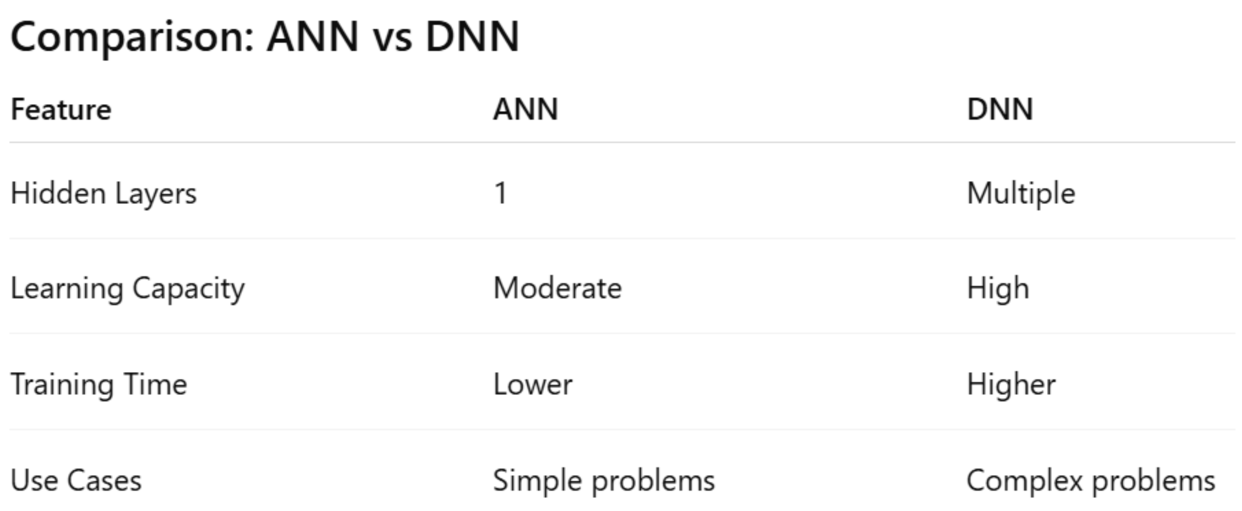
Use a Deep Neural Network to classify handwritten digits (0–9).

Solution:

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Preprocessing
X_train = X_train.reshape(-1, 28*28) / 255.0
X_test = X_test.reshape(-1, 28*28) / 255.0

# One-hot encode labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Build DNN
model = Sequential()
model.add(Dense(256, activation='relu', input_shape=(784,)))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compile and train
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=128, verbose=1)

# Evaluate
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", accuracy)
```

DNN learns hierarchical pixel features and achieves good accuracy, though CNNs are more suitable for image data.



Comparison: ANN vs DNN

Feature	ANN	DNN
Hidden Layers	1	Multiple
Learning Capacity	Moderate	High
Training Time	Lower	Higher
Use Cases	Simple problems	Complex problems

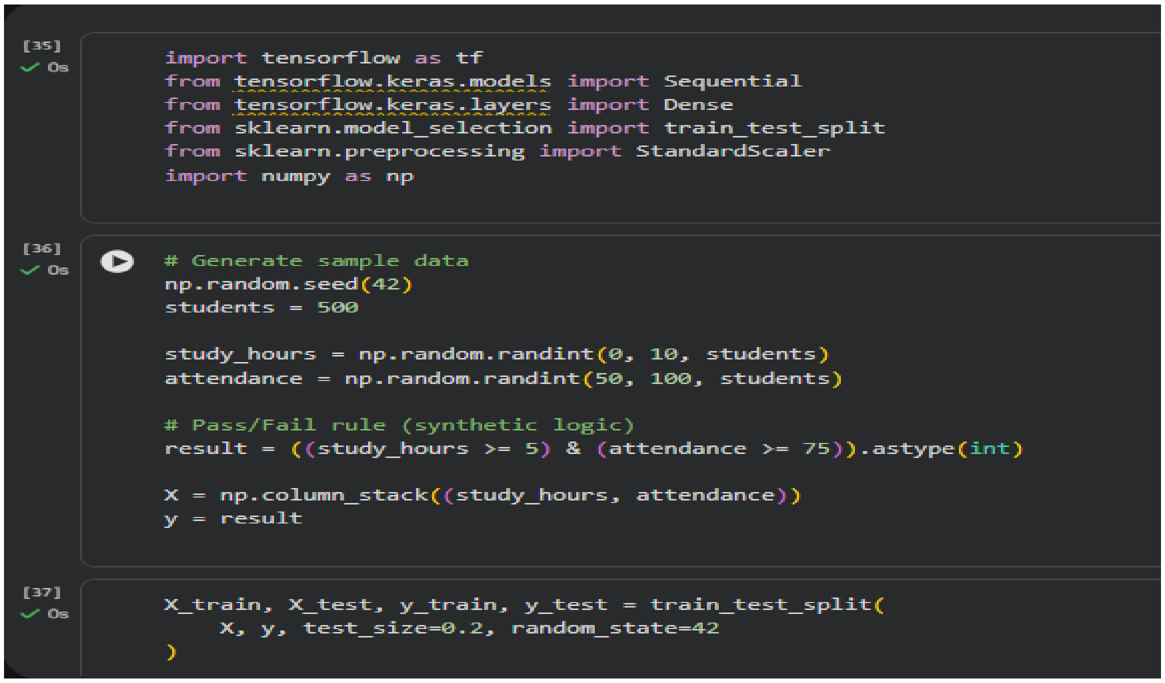
LAB Assignment No 6

Practice Question 1:

DNN Architecture Design

Create a Deep Neural Network to predict whether a student **passes or fails** using features such as *study hours* and *attendance*.

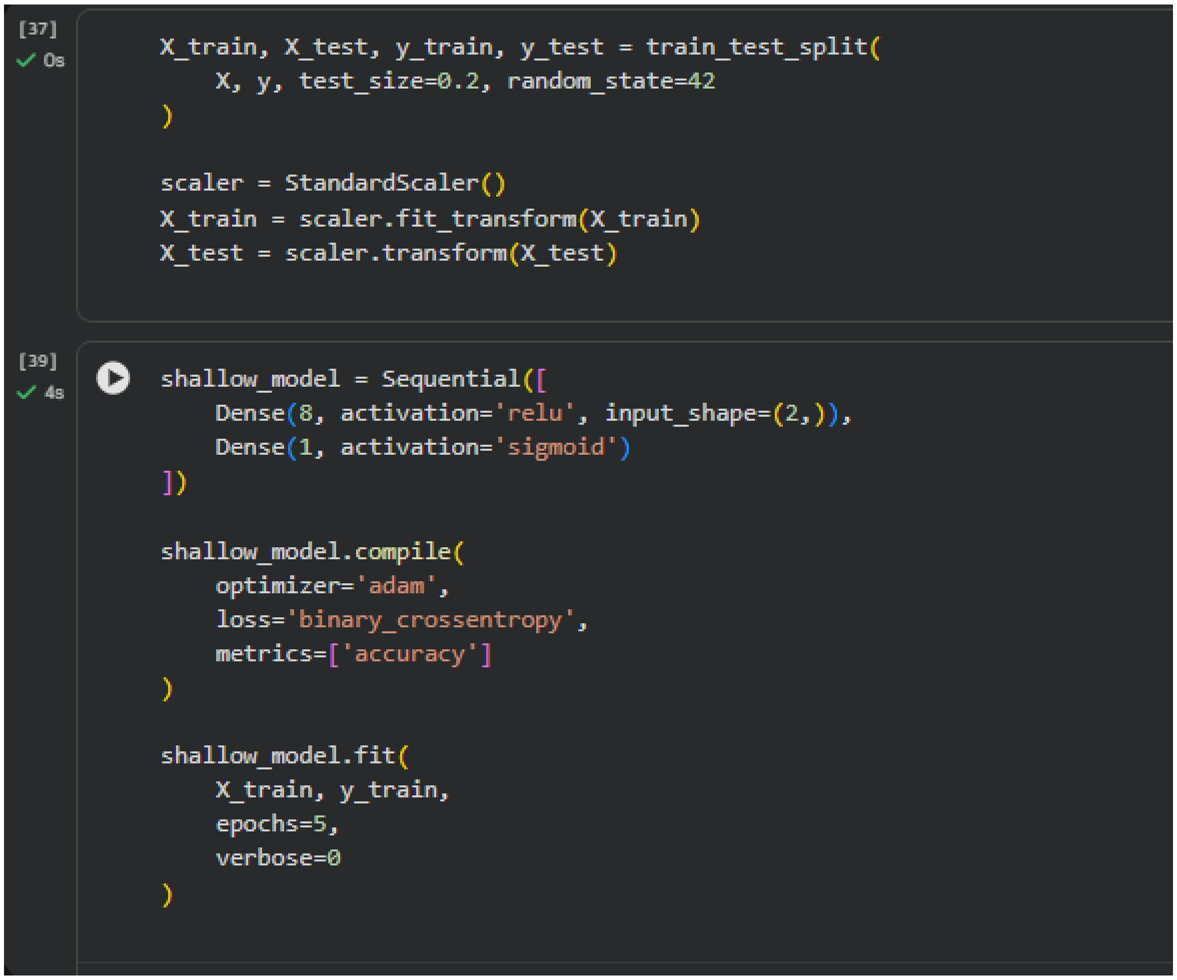
- Use **at least three hidden layers**
- Experiment with different numbers of neurons
- Compare the accuracy with a shallow ANN (one hidden layer)



```
[35]
✓ 0s
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
```

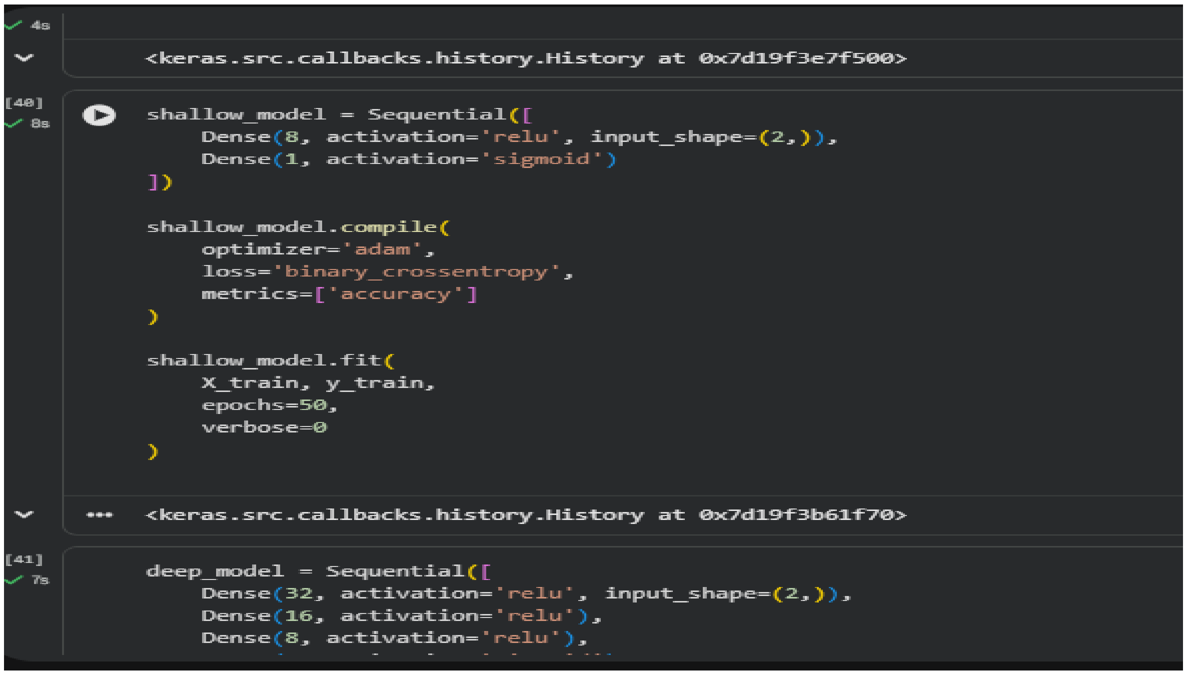
```
[36]
✓ 0s
# Generate sample data
np.random.seed(42)
students = 500

study_hours = np.random.randint(0, 10, students)
attendance = np.random.randint(50, 100, students)

# Pass/Fail rule (synthetic logic)
result = ((study_hours >= 5) & (attendance >= 75)).astype(int)

X = np.column_stack((study_hours, attendance))
y = result
```

```
[37]
✓ 0s
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```



```
[37]
✓ 0s
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
[39]
✓ 4s
shallow_model = Sequential([
    Dense(8, activation='relu', input_shape=(2,)),
    Dense(1, activation='sigmoid')
])

shallow_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

shallow_model.fit(
    X_train, y_train,
    epochs=5,
    verbose=0
)
```

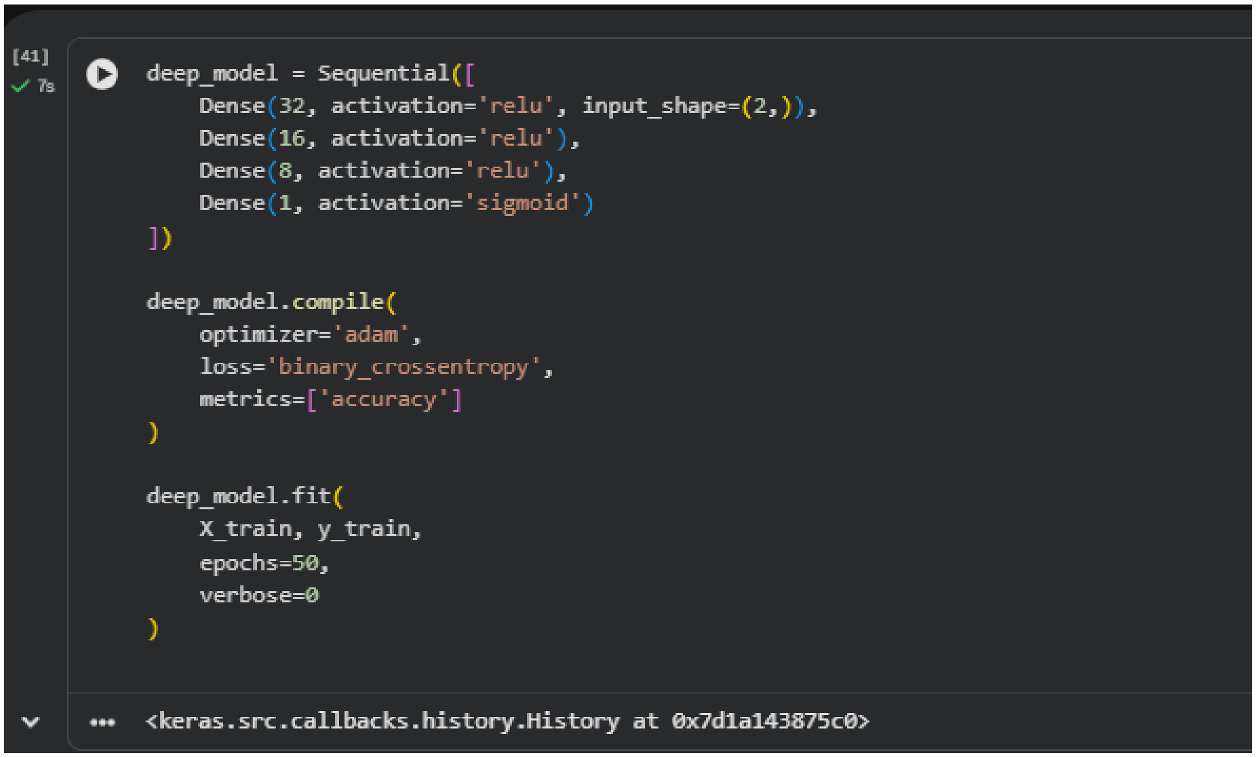


```
<keras.src.callbacks.history.History at 0x7d19f3e7f500>
```

```
shallow_model = Sequential([
    Dense(8, activation='relu', input_shape=(2,)),
    Dense(1, activation='sigmoid')
])

shallow_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

shallow_model.fit(
    X_train, y_train,
    epochs=50,
    verbose=0
)
```

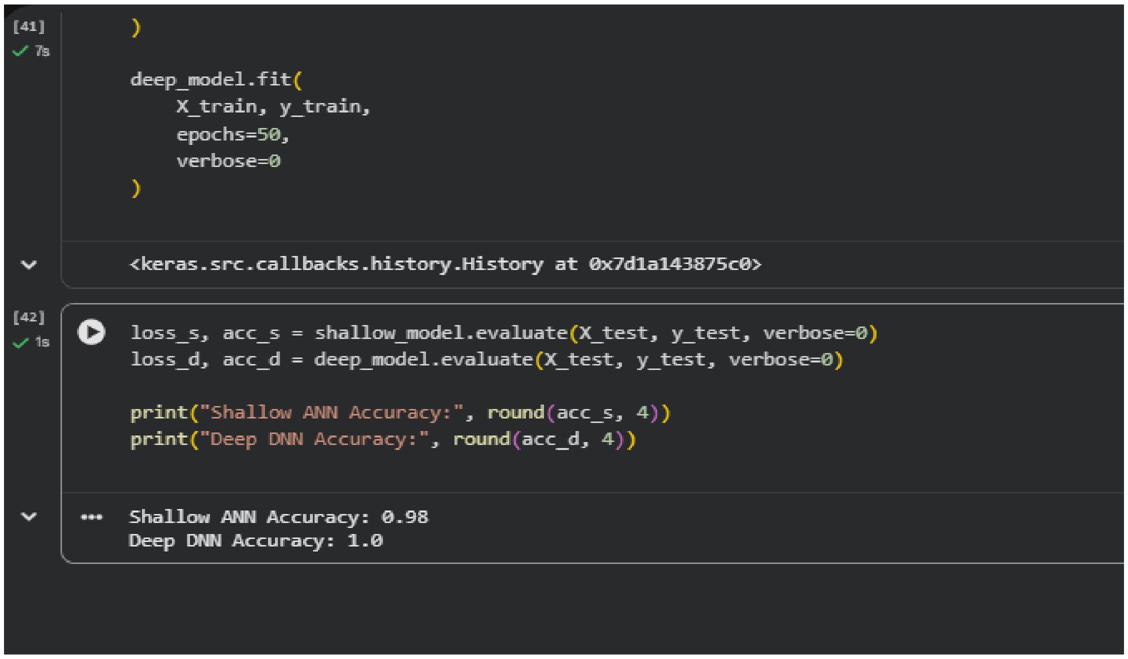
```
<keras.src.callbacks.history.History at 0x7d19f3b61f70>
```

```
deep_model = Sequential([
    Dense(32, activation='relu', input_shape=(2,)),
    Dense(16, activation='relu'),
    Dense(8, activation='relu'),
```



```
deep_model = Sequential([
    Dense(32, activation='relu', input_shape=(2,)),
    Dense(16, activation='relu'),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid')
])

deep_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

deep_model.fit(
    X_train, y_train,
    epochs=50,
    verbose=0
)
```

```
<keras.src.callbacks.history.History at 0x7d1a143875c0>
```



```
[41]    )

        deep_model.fit(
            X_train, y_train,
            epochs=50,
            verbose=0
        )

        <keras.src.callbacks.history.History at 0x7d1a143875c0>
```

```
[42]    loss_s, acc_s = shallow_model.evaluate(X_test, y_test, verbose=0)
        loss_d, acc_d = deep_model.evaluate(X_test, y_test, verbose=0)

        print("Shallow ANN Accuracy:", round(acc_s, 4))
        print("Deep DNN Accuracy:", round(acc_d, 4))

        Shallow ANN Accuracy: 0.98
        Deep DNN Accuracy: 1.0
```

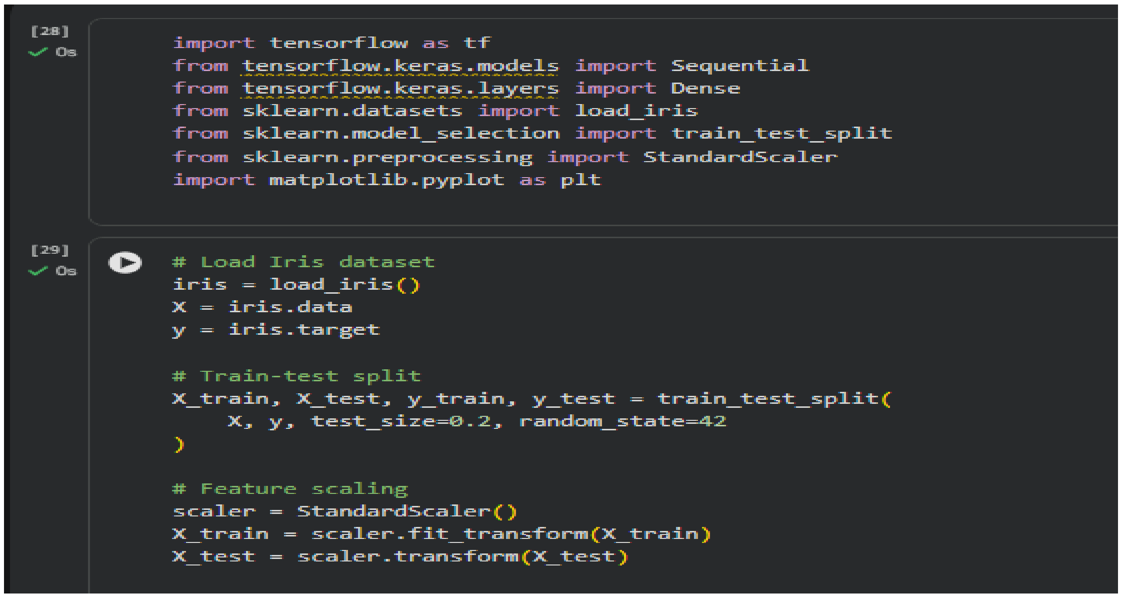
Practice Question 2:

Activation Function Analysis

Using the **Iris dataset**, build two DNN models:

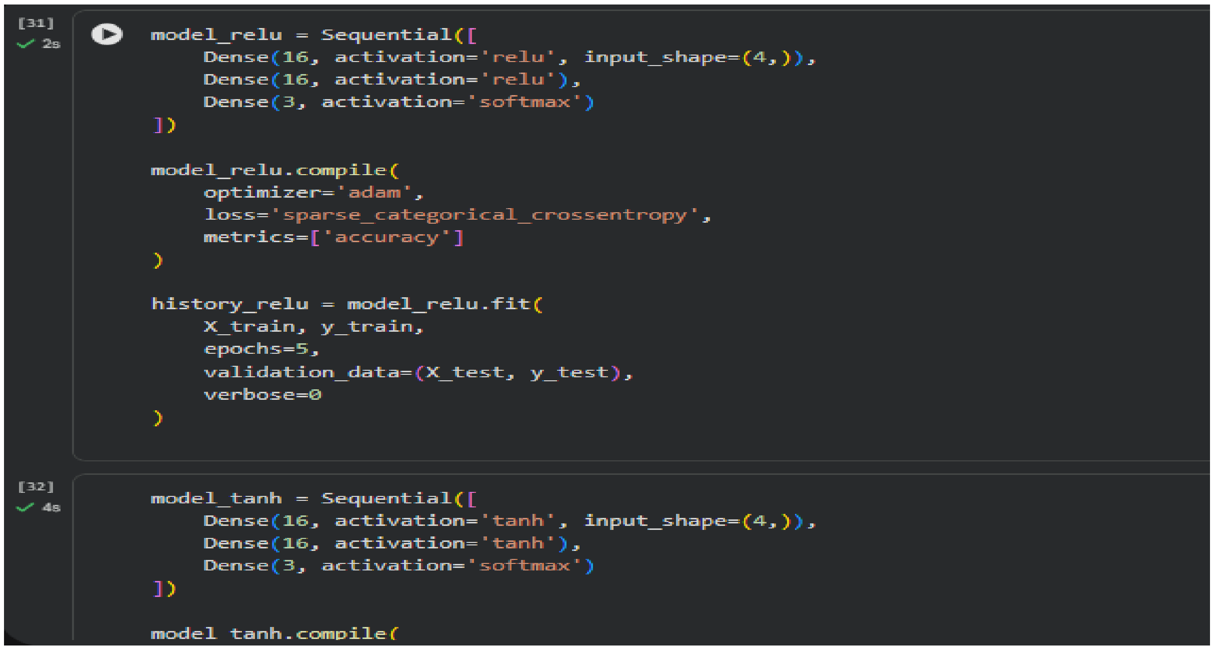
- Model A: Use **ReLU** activation in all hidden layers
- Model B: Use **tanh** activation in all hidden layers

Train both models and **compare their accuracy and training behavior**. Write your observation.

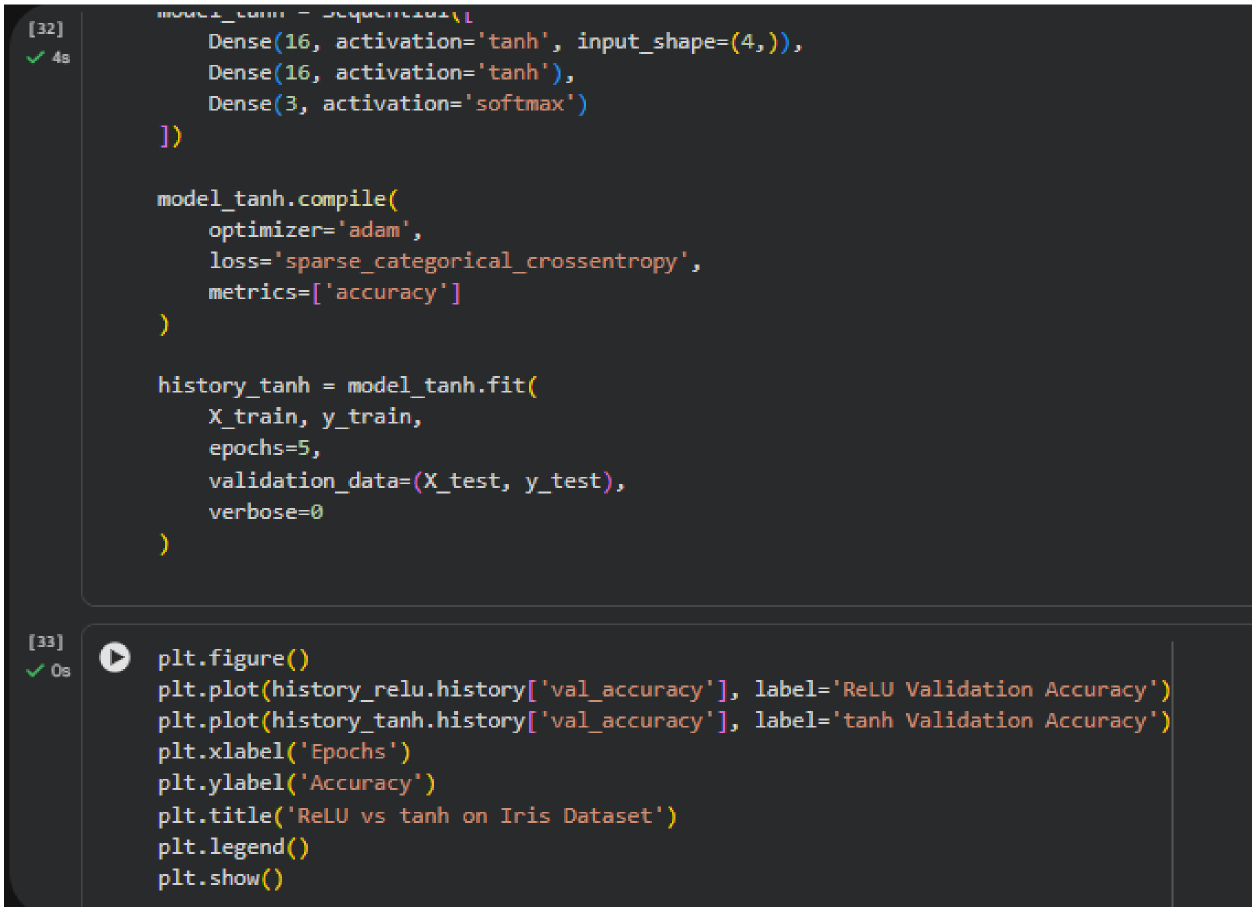


```
[28]    import tensorflow as tf
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense
        from sklearn.datasets import load_iris
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        import matplotlib.pyplot as plt
```

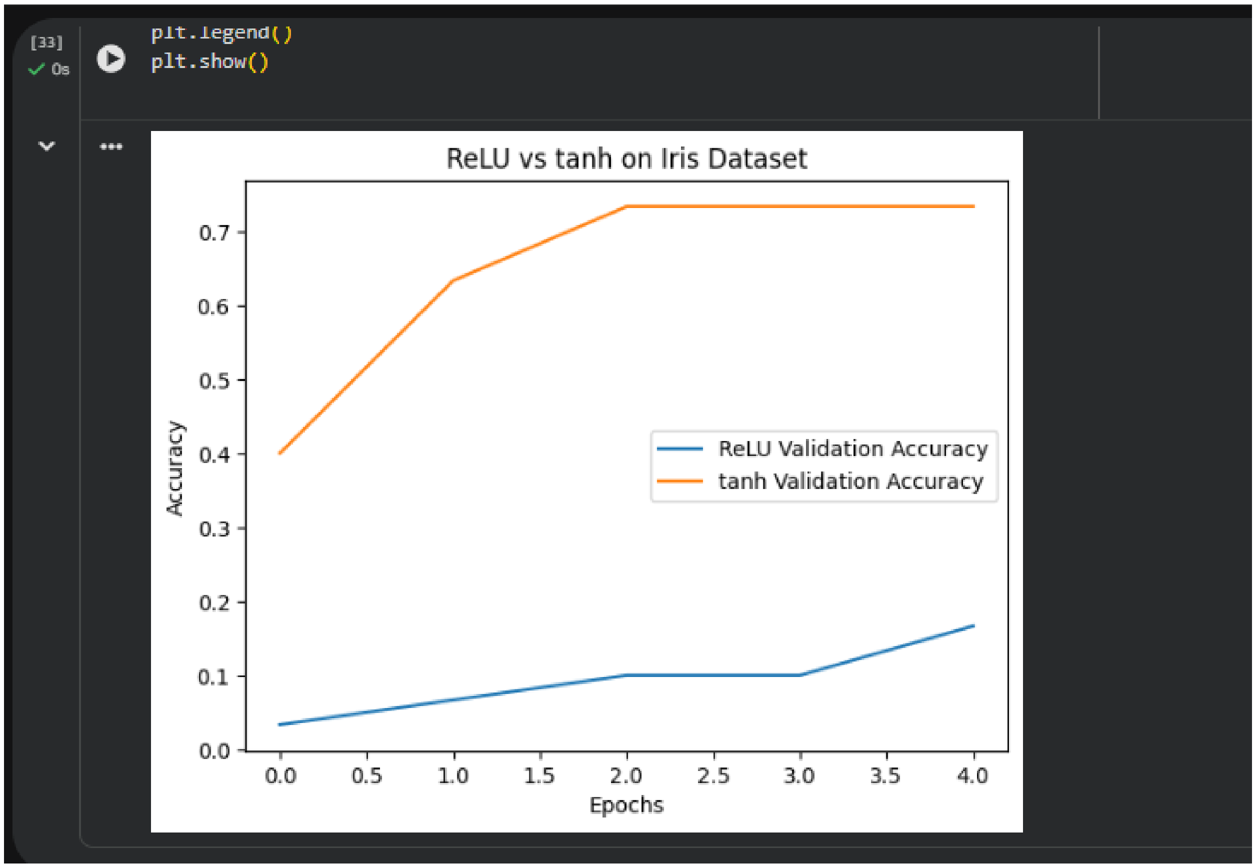
```
[29]    # Load Iris dataset
        iris = load_iris()
        X = iris.data
        y = iris.target

        # Train-test split
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        # Feature scaling
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)
```



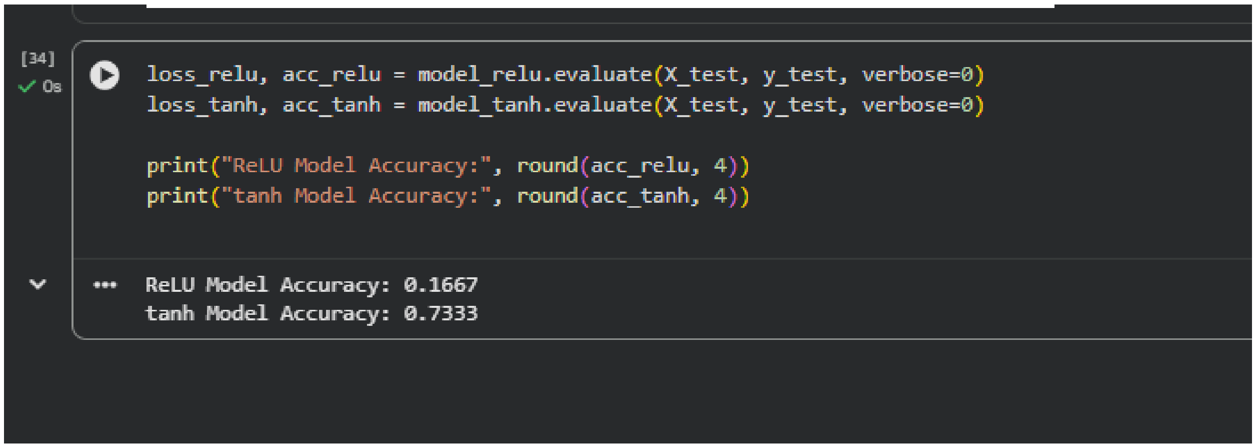
```
[31]
✓ 2s
model_relu = Sequential([
    Dense(16, activation='relu', input_shape=(4,)),
    Dense(16, activation='relu'),
    Dense(3, activation='softmax')
])

model_relu.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

history_relu = model_relu.fit(
    X_train, y_train,
    epochs=5,
    validation_data=(X_test, y_test),
    verbose=0
)

[32]
✓ 4s
model_tanh = Sequential([
    Dense(16, activation='tanh', input_shape=(4,)),
    Dense(16, activation='tanh'),
    Dense(3, activation='softmax')
])

model_tanh.compile(
```



```
[32]
✓ 4s
    Dense(16, activation='tanh', input_shape=(4,)),
    Dense(16, activation='tanh'),
    Dense(3, activation='softmax')
])

model_tanh.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

history_tanh = model_tanh.fit(
    X_train, y_train,
    epochs=5,
    validation_data=(X_test, y_test),
    verbose=0
)

[33]
✓ 0s
plt.figure()
plt.plot(history_relu.history['val_accuracy'], label='ReLU Validation Accuracy')
plt.plot(history_tanh.history['val_accuracy'], label='tanh Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('ReLU vs tanh on Iris Dataset')
plt.legend()
plt.show()
```



```
[33]
plt.legend()
plt.show()
```



```
[34]
loss_relu, acc_relu = model_relu.evaluate(X_test, y_test, verbose=0)
loss_tanh, acc_tanh = model_tanh.evaluate(X_test, y_test, verbose=0)

print("ReLU Model Accuracy:", round(acc_relu, 4))
print("tanh Model Accuracy:", round(acc_tanh, 4))
```

```
ReLU Model Accuracy: 0.1667
tanh Model Accuracy: 0.7333
```

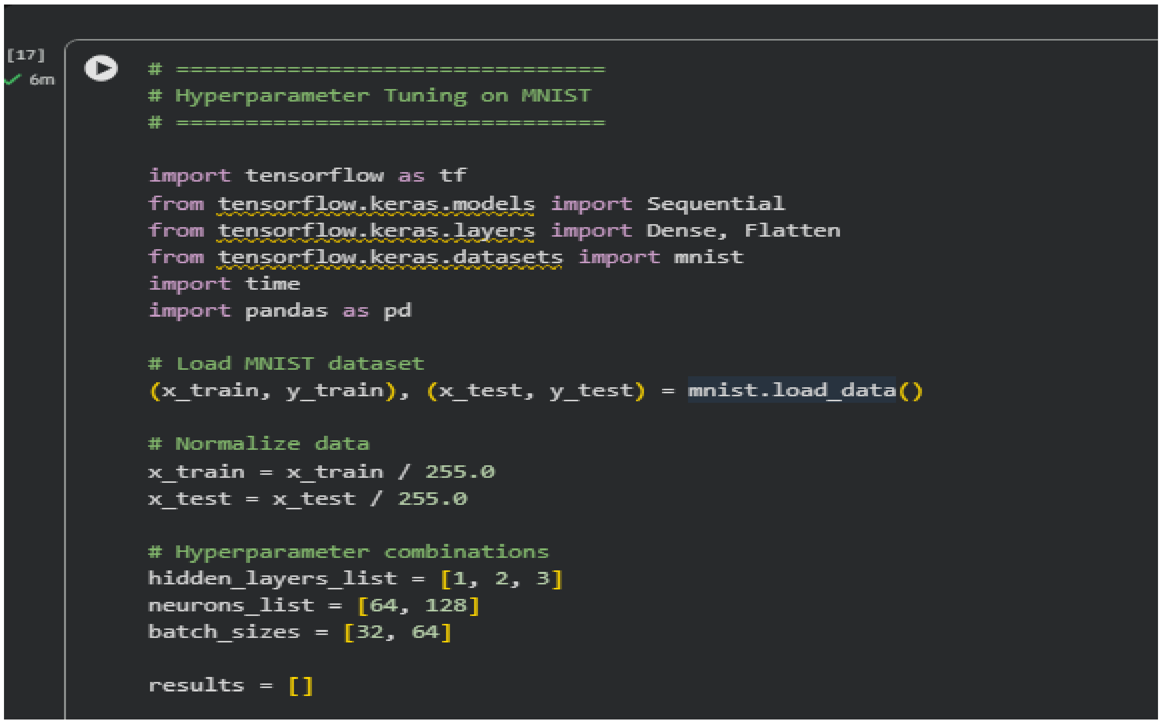
Practice Question 3:

Hyperparameter Tuning in DNN

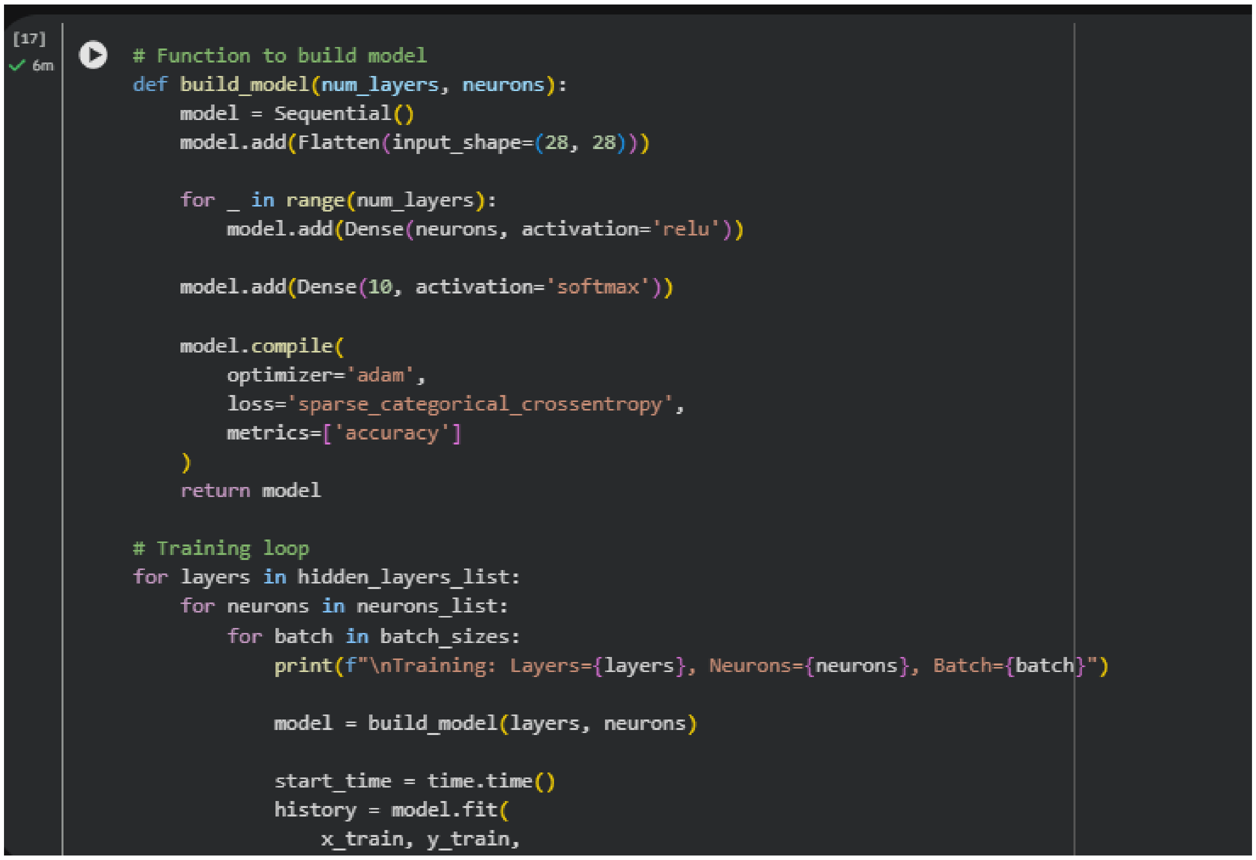
Train a DNN on the **MNIST dataset** by changing:

- Number of hidden layers
- Number of neurons per layer
- Batch size

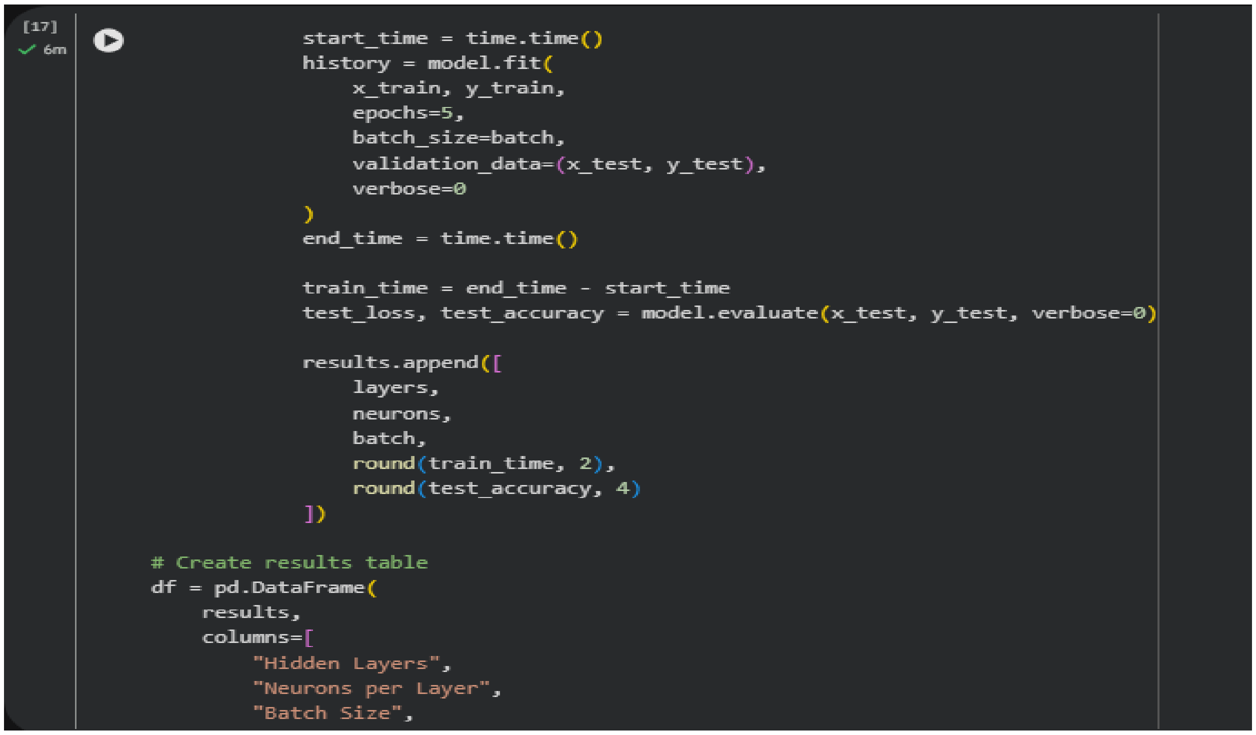
Record how these changes affect **training time and accuracy**



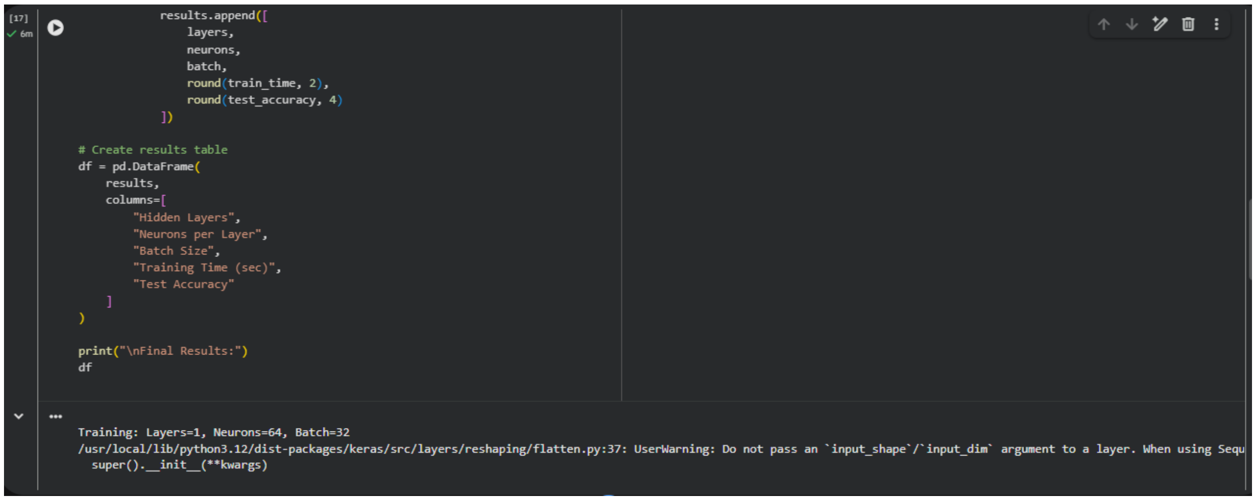
```python
# ================================
# Hyperparameter Tuning on MNIST
# ================================

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
import time
import pandas as pd

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize data
x_train = x_train / 255.0
x_test = x_test / 255.0

# Hyperparameter combinations
hidden_layers_list = [1, 2, 3]
neurons_list = [64, 128]
batch_sizes = [32, 64]

results = []
```



```python
# Function to build model
def build_model(num_layers, neurons):
    model = Sequential()
    model.add(Flatten(input_shape=(28, 28)))

    for _ in range(num_layers):
        model.add(Dense(neurons, activation='relu'))

    model.add(Dense(10, activation='softmax'))

    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

# Training loop
for layers in hidden_layers_list:
    for neurons in neurons_list:
        for batch in batch_sizes:
            print(f"\nTraining: Layers={layers}, Neurons={neurons}, Batch={batch}")

            model = build_model(layers, neurons)

            start_time = time.time()
            history = model.fit(
                x_train, y_train,
```

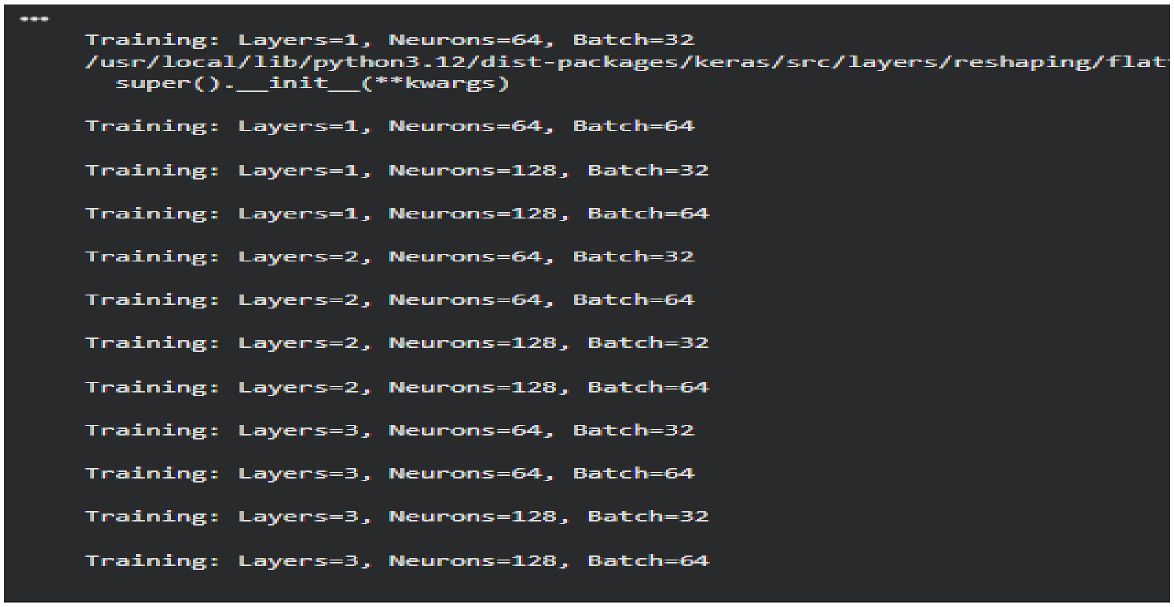


```
[17]
 6m
                start_time = time.time()
                history = model.fit(
                    x_train, y_train,
                    epochs=5,
                    batch_size=batch,
                    validation_data=(x_test, y_test),
                    verbose=0
                )
                end_time = time.time()

                train_time = end_time - start_time
                test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)

                results.append([
                    layers,
                    neurons,
                    batch,
                    round(train_time, 2),
                    round(test_accuracy, 4)
                ])

    # Create results table
    df = pd.DataFrame(
        results,
        columns=[
            "Hidden Layers",
            "Neurons per Layer",
            "Batch Size",
```

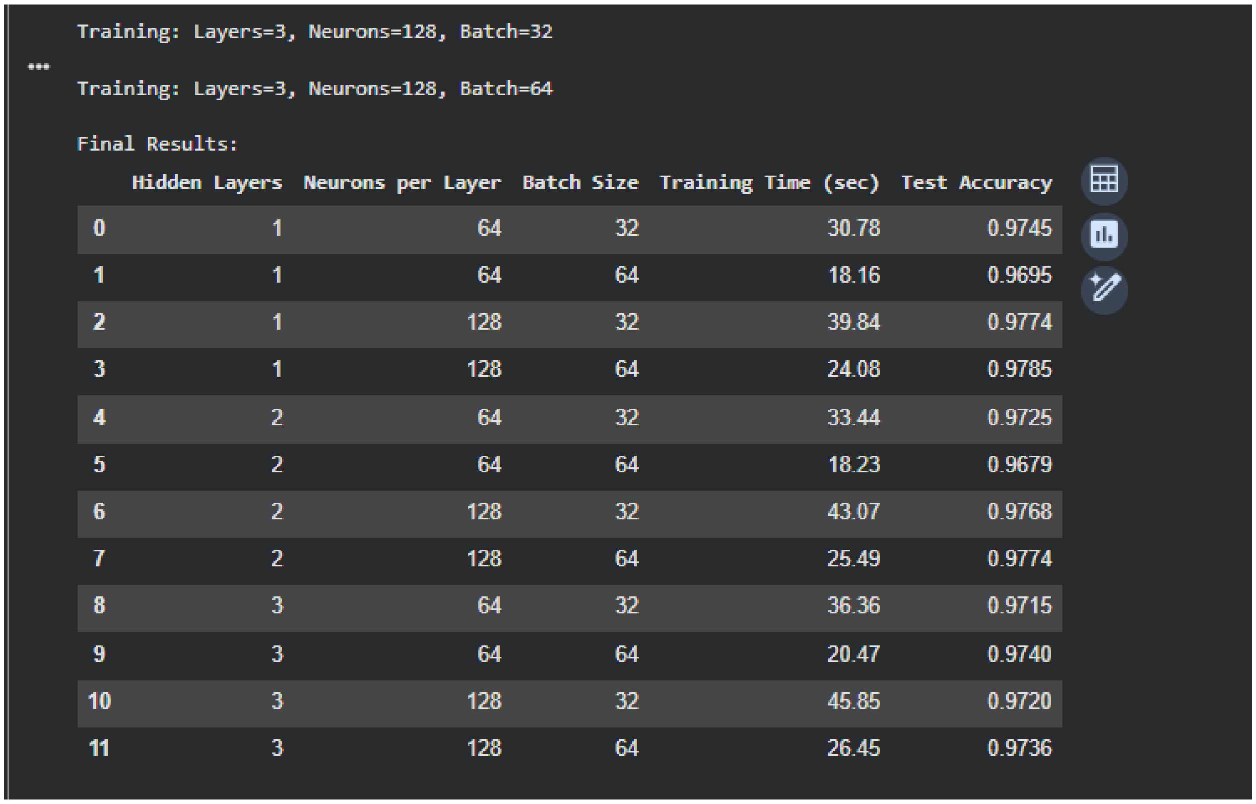


```
[17]
 6m
            results.append([
                layers,
                neurons,
                batch,
                round(train_time, 2),
                round(test_accuracy, 4)
            ])

    # Create results table
    df = pd.DataFrame(
        results,
        columns=[
            "Hidden Layers",
            "Neurons per Layer",
            "Batch Size",
            "Training Time (sec)",
            "Test Accuracy"
        ]
    )

    print("\nFinal Results:")
    df
```

```
Training: Layers=1, Neurons=64, Batch=32
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequ
  super().__init__(**kwargs)
```



```
...
    Training: Layers=1, Neurons=64, Batch=32
    /usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flat
      super().__init__(**kwargs)

    Training: Layers=1, Neurons=64, Batch=64

    Training: Layers=1, Neurons=128, Batch=32

    Training: Layers=1, Neurons=128, Batch=64

    Training: Layers=2, Neurons=64, Batch=32

    Training: Layers=2, Neurons=64, Batch=64

    Training: Layers=2, Neurons=128, Batch=32

    Training: Layers=2, Neurons=128, Batch=64

    Training: Layers=3, Neurons=64, Batch=32

    Training: Layers=3, Neurons=64, Batch=64

    Training: Layers=3, Neurons=128, Batch=32

    Training: Layers=3, Neurons=128, Batch=64
```



```
    Training: Layers=3, Neurons=128, Batch=32
...
    Training: Layers=3, Neurons=128, Batch=64

    Final Results:
```

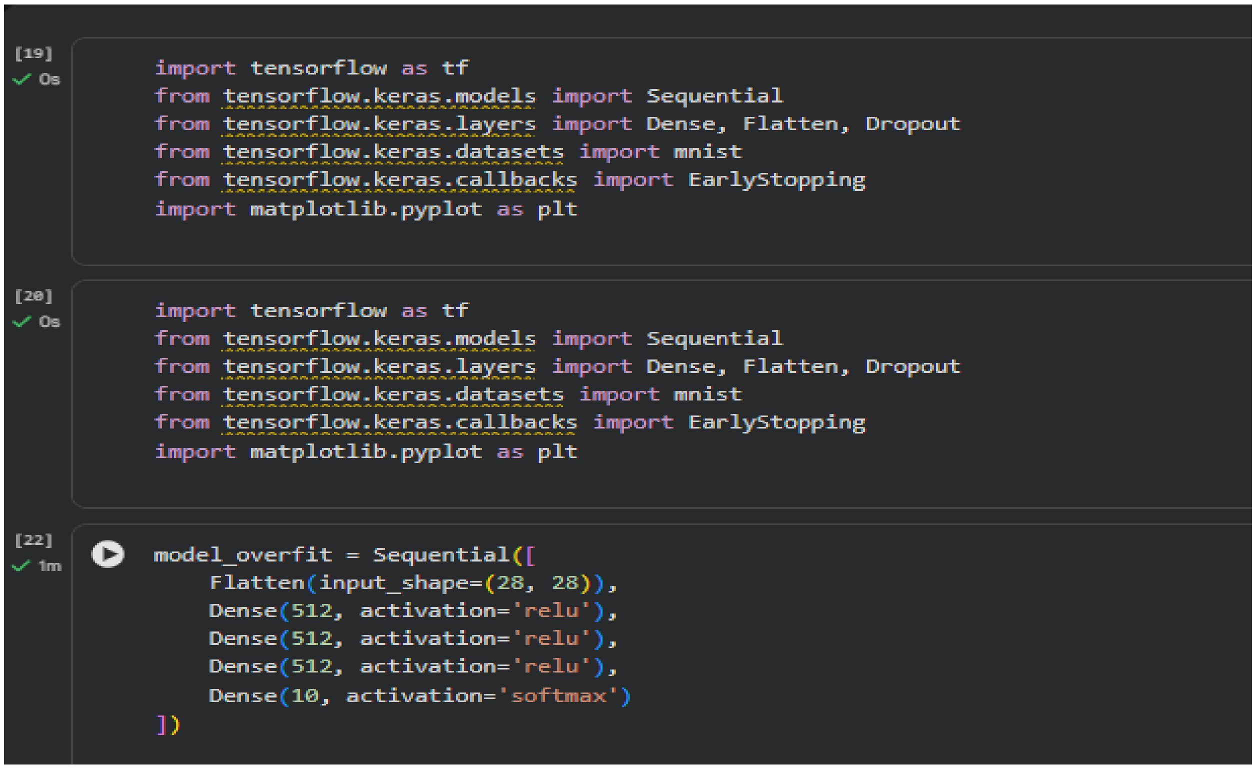
	Hidden Layers	Neurons per Layer	Batch Size	Training Time (sec)	Test Accuracy
0	1	64	32	30.78	0.9745
1	1	64	64	18.16	0.9695
2	1	128	32	39.84	0.9774
3	1	128	64	24.08	0.9785
4	2	64	32	33.44	0.9725
5	2	64	64	18.23	0.9679
6	2	128	32	43.07	0.9768
7	2	128	64	25.49	0.9774
8	3	64	32	36.36	0.9715
9	3	64	64	20.47	0.9740
10	3	128	32	45.85	0.9720
11	3	128	64	26.45	0.9736

Practice Question 4:

Overfitting and Regularization

Build a deep neural network on any classification dataset and:

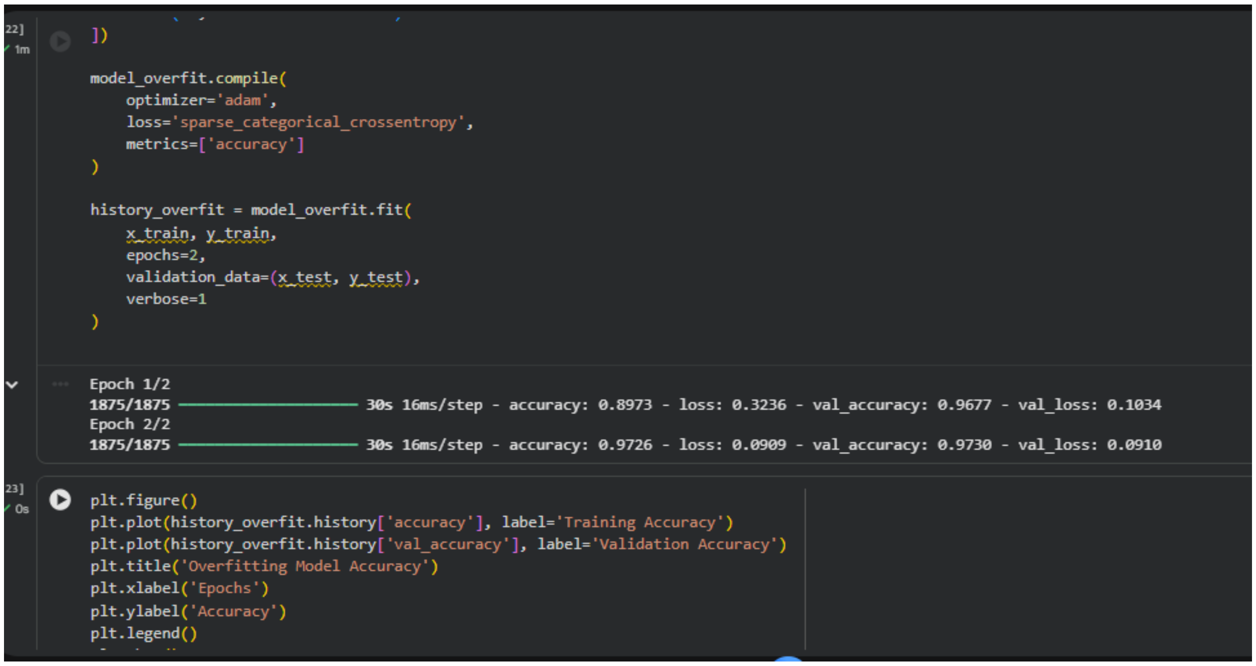
- Observe signs of **overfitting**
- Apply at least one regularization technique (Dropout or Early Stopping)
- Compare model performance **before and after regularization**



```
[19]
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
```

```
[20]
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
```

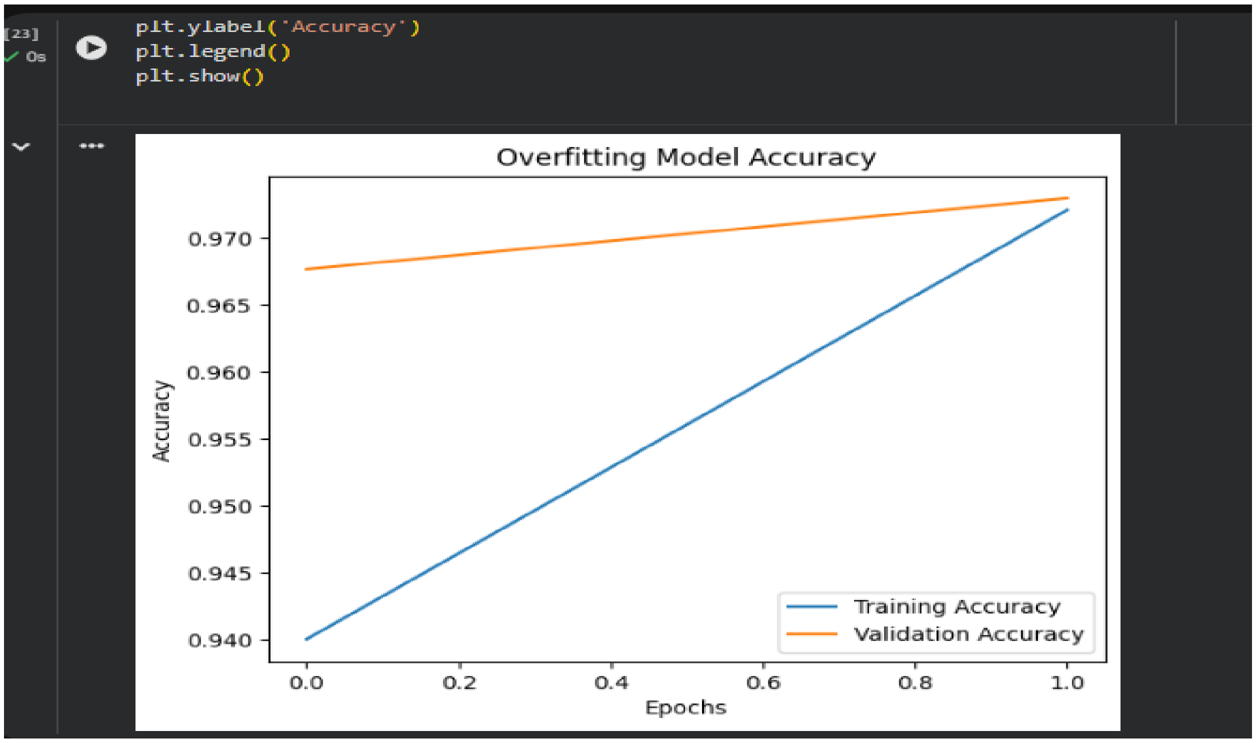
```
[22]
model_overfit = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(512, activation='relu'),
    Dense(512, activation='relu'),
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])
```



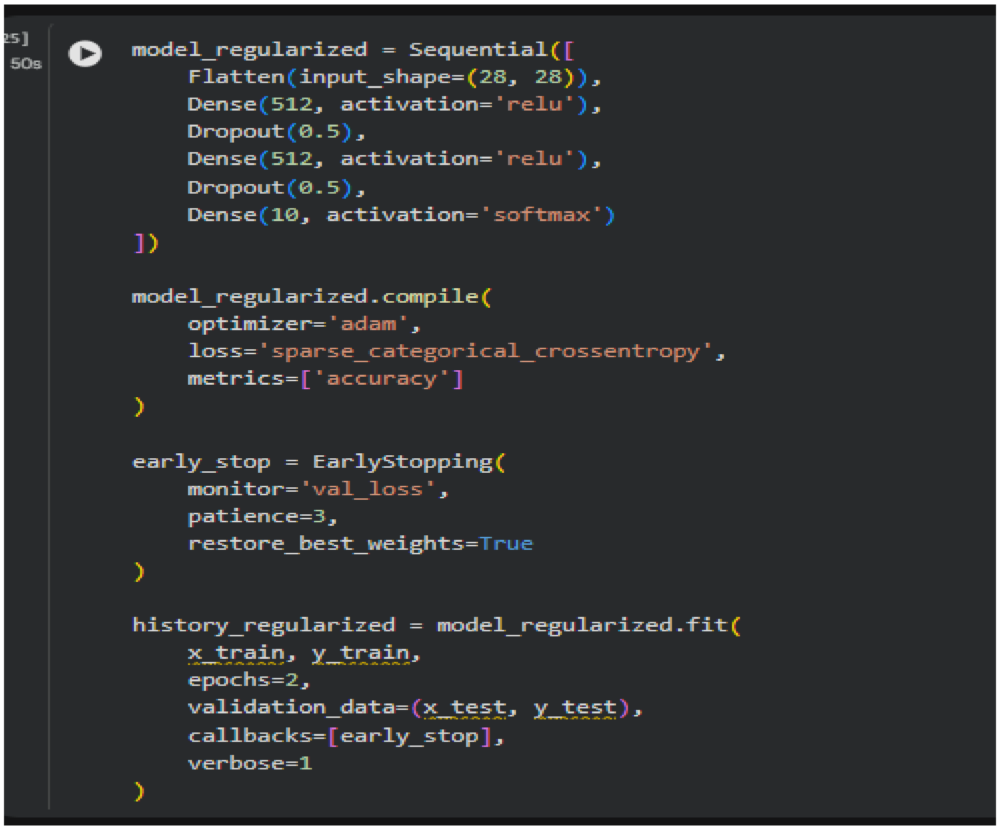
```
])

model_overfit.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

history_overfit = model_overfit.fit(
    x_train, y_train,
    epochs=2,
    validation_data=(x_test, y_test),
    verbose=1
)
```

```
Epoch 1/2
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step - accuracy: 0.8973 - loss: 0.3236 - val_accuracy: 0.9677 - val_loss: 0.1034
Epoch 2/2
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step - accuracy: 0.9726 - loss: 0.0909 - val_accuracy: 0.9730 - val_loss: 0.0910
```

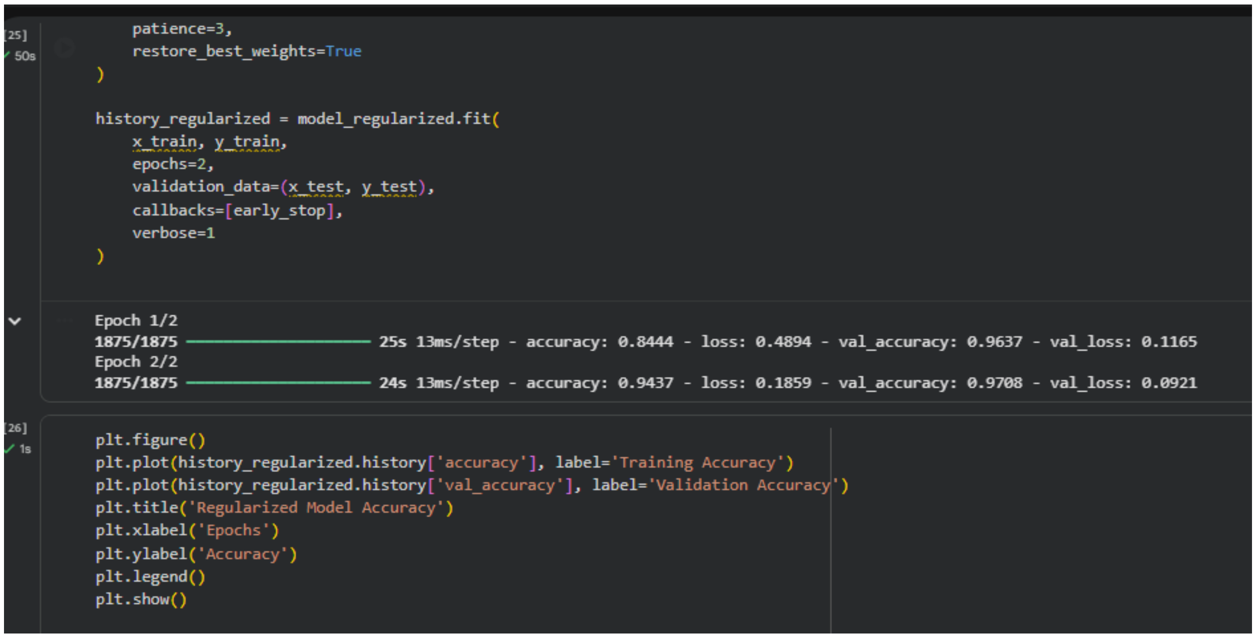
```
[23]
plt.figure()
plt.plot(history_overfit.history['accuracy'], label='Training Accuracy')
plt.plot(history_overfit.history['val_accuracy'], label='Validation Accuracy')
plt.title('Overfitting Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```



```
[23]
 0s
        plt.ylabel('Accuracy')
        plt.legend()
        plt.show()
```



```
25]
50s
        model_regularized = Sequential([
            Flatten(input_shape=(28, 28)),
            Dense(512, activation='relu'),
            Dropout(0.5),
            Dense(512, activation='relu'),
            Dropout(0.5),
            Dense(10, activation='softmax')
        ])

        model_regularized.compile(
            optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']
        )

        early_stop = EarlyStopping(
            monitor='val_loss',
            patience=3,
            restore_best_weights=True
        )

        history_regularized = model_regularized.fit(
            x_train, y_train,
            epochs=2,
            validation_data=(x_test, y_test),
            callbacks=[early_stop],
            verbose=1
        )
```

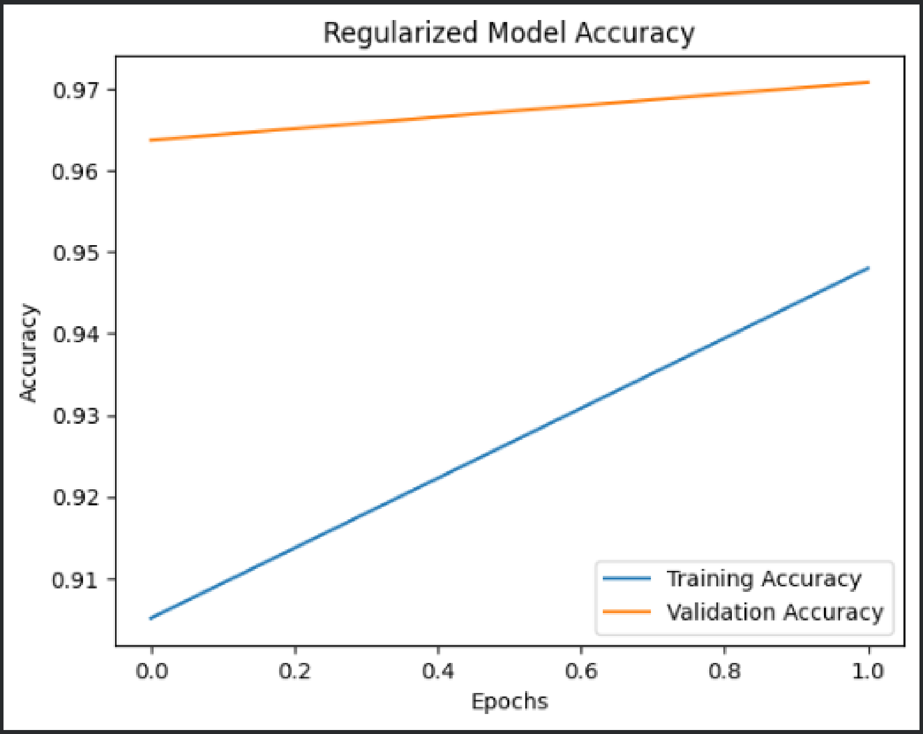


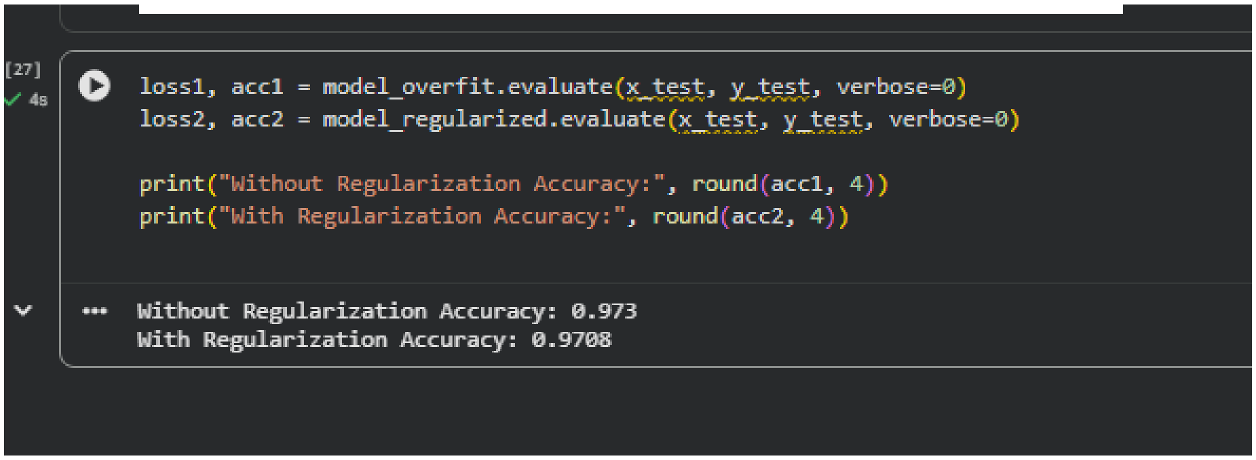
```
    patience=3,
    restore_best_weights=True
)

history_regularized = model_regularized.fit(
    x_train, y_train,
    epochs=2,
    validation_data=(x_test, y_test),
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/2
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 25s 13ms/step - accuracy: 0.8444 - loss: 0.4894 - val_accuracy: 0.9637 - val_loss: 0.1165
Epoch 2/2
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 24s 13ms/step - accuracy: 0.9437 - loss: 0.1859 - val_accuracy: 0.9708 - val_loss: 0.0921
```

```
plt.figure()
plt.plot(history_regularized.history['accuracy'], label='Training Accuracy')
plt.plot(history_regularized.history['val_accuracy'], label='Validation Accuracy')
plt.title('Regularized Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
plt.legend()
plt.show()
```





```
loss1, acc1 = model_overfit.evaluate(x_test, y_test, verbose=0)
loss2, acc2 = model_regularized.evaluate(x_test, y_test, verbose=0)

print("Without Regularization Accuracy:", round(acc1, 4))
print("With Regularization Accuracy:", round(acc2, 4))
```

```
Without Regularization Accuracy: 0.973
With Regularization Accuracy: 0.9708
```