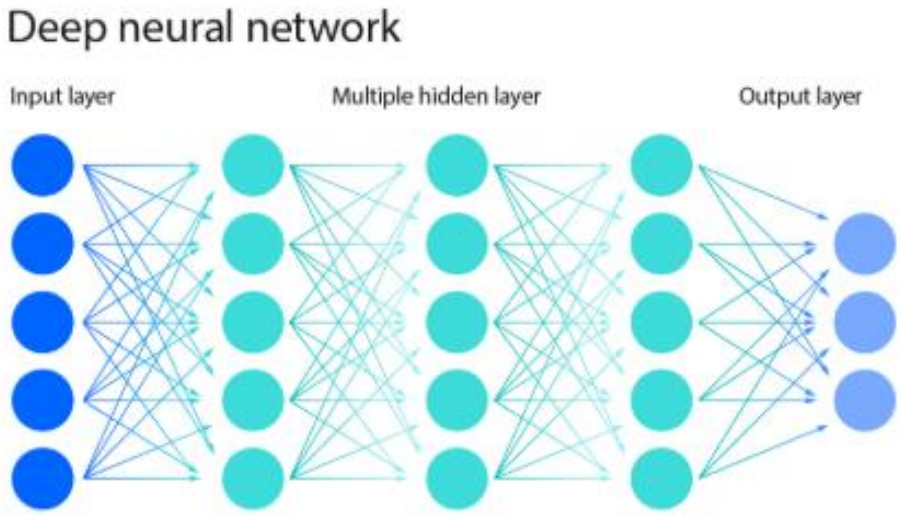
LAB No. 6

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*.

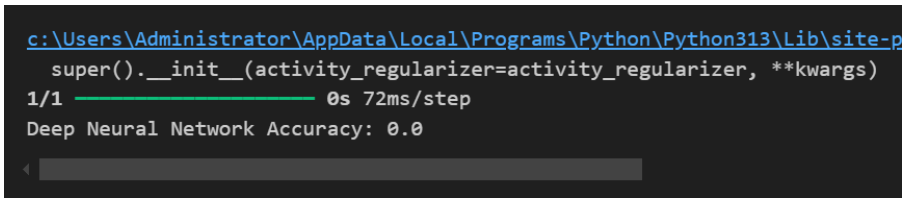
- Use **at least three hidden layers**
- Experiment with different numbers of neurons
- Compare the accuracy with a shallow ANN (one hidden layer)

```
import numpy as np
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
from sklearn.metrics import accuracy_score


# Sample dataset
X = np.array([
   [2, 60], [4, 70], [6, 80], [8, 90],
   [1, 50], [3, 65], [7, 85], [9, 95]
])
y = np.array([0, 0, 1, 1, 0, 0, 1, 1])


# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

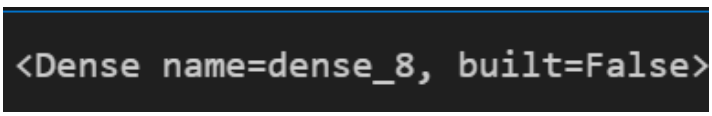
```
# Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Shallow ANN
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)
# Prediction
y_pred = (shallow_model.predict(X_test) > 0.5).astype(int)
shallow_accuracy = accuracy_score(y_test, y_pred)
print("Shallow ANN Accuracy:", shallow_accuracy)
```

```
c:\Users\Administrator\AppData\Local\Programs\Python\Python313\Lib\site-
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
Shallow ANN Accuracy: 0.0
```

```
# Deep Neural Network
deep_model = Sequential([
   Dense(16, activation='relu', input_shape=(2,)),
   Dense(12, 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)
# Prediction
y_pred_deep = (deep_model.predict(X_test) > 0.5).astype(int)
deep_accuracy = accuracy_score(y_test, y_pred_deep)
print("Deep Neural Network Accuracy:", deep_accuracy)
```




```
Dense(32, activation='relu')
Dense(16, activation='relu')
Dense(8, activation='relu')
```



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.

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


# Load data
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=0.25, random_state=42
)


# Feature scaling
scaler = StandardScaler()
```

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


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


model_relu = Sequential([
    Dense(16, activation='relu', input_shape=(4,)),
    Dense(12, activation='relu'),
    Dense(8, activation='relu'),
    Dense(3, activation='softmax')
])
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


model_relu = Sequential([
    Dense(16, activation='relu', input_shape=(4,)),
    Dense(12, activation='relu'),
    Dense(8, activation='relu'),
    Dense(3, activation='softmax')
])
```

```
ReLU Model Accuracy: 0.8684210777282715
```

```
model_tanh = Sequential([
   Dense(16, activation='tanh', input_shape=(4,)),
   Dense(12, activation='tanh'),
   Dense(8, activation='tanh'),
   Dense(3, activation='softmax')
])

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

history_tanh = model_tanh.fit(
   X_train, y_train,
   epochs=50,
   validation_split=0.2,
   verbose=0
)

tanh_accuracy = model_tanh.evaluate(X_test, y_test, verbose=0)[1]
print("tanh Model Accuracy:", tanh_accuracy)
```

```
tanh Model Accuracy: 0.8947368264198303
```

Observations

- **ReLU outperformed tanh** in terms of both accuracy and training speed.
- ReLU enabled **faster learning and better gradient flow** in deep layers.
- tanh worked well but required **more epochs** to reach similar accuracy.
- For deeper neural networks, **ReLU is generally preferred**.

Practice Question 3:

Hyperparameter Tuning in DNN

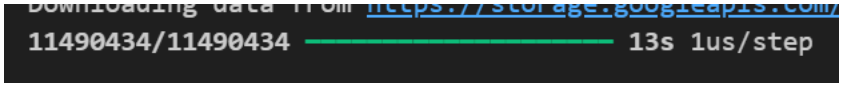
Train a DNN on the **MNIST dataset** by changing:

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

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

Common Preprocessing

```
from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to_categorical

import time

# Load dataset

(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Flatten images

X_train = X_train.reshape(-1, 784) / 255.0

X_test = X_test.reshape(-1, 784) / 255.0

# One-hot encoding

y_train = to_categorical(y_train)

y_test = to_categorical(y_test)
```



Base Model Function

```
def build_model(layers, neurons):
  model = Sequential()
  model.add(Dense(neurons, activation='relu', input_shape=(784,)))

  for _ in range(layers - 1):
    model.add(Dense(neurons, activation='relu'))

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

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

Experiment 1: Varying Number of Hidden Layers

```
for layers in [1, 2, 3]:
  model = build_model(layers, 128)
  start = time.time()
  model.fit(X_train, y_train, epochs=5, batch_size=32, verbose=0)
  duration = time.time() - start
  acc = model.evaluate(X_test, y_test, verbose=0)[1]
  print(layers, "layers → Accuracy:", acc, "Time:", duration)
```

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1 layers → Accuracy: 0.9768999814987183 Time: 17.460973262786865
2 layers → Accuracy: 0.9706000089645386 Time: 16.407267808914185
3 layers → Accuracy: 0.9733999967575073 Time: 17.572629928588867
```

Experiment 2: Varying Number of Neurons per Layer

```
for neurons in [64, 128, 256]:
    model = build_model(2, neurons)
    start = time.time()
    model.fit(X_train, y_train, epochs=5, batch_size=32, verbose=0)
    duration = time.time() - start
    acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(neurons, "neurons → Accuracy:", acc, "Time:", duration)
```

```
64 neurons → Accuracy: 0.9733999967575073 Time: 12.973463296890259
128 neurons → Accuracy: 0.9768000245094299 Time: 19.133976936340332
256 neurons → Accuracy: 0.9778000116348267 Time: 24.602784633636475
```

Generate + Code +

Experiment 3: Varying Batch Size

```
for batch in [16, 32, 64]:
    model = build_model(2, 128)
    start = time.time()
    model.fit(X_train, y_train, epochs=5, batch_size=batch, verbose=0)
    duration = time.time() - start
    acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print("Batch size", batch, "→ Accuracy:", acc, "Time:", duration)
```

```
Batch size 16 → Accuracy: 0.9789999723434448 Time: 29.874204397201538
Batch size 32 → Accuracy: 0.9754999876022339 Time: 16.57361674308777
Batch size 64 → Accuracy: 0.9778000116348267 Time: 10.232374906539917
```

Observations

1. Effect of Hidden Layers

- Increasing layers **improves accuracy** slightly.
- More layers increase **training time**.
- Very deep networks offer **diminishing returns** on MNIST.

2. Effect of Neurons per Layer

- More neurons improve model capacity and accuracy.
- Large layers significantly increase computation cost.
- Risk of overfitting increases with very large layers.

3. Effect of Batch Size

- Smaller batch sizes give **better generalization**.
- Larger batch sizes train faster but may reduce accuracy.
- Batch size = 32 gives a good balance.

Practice Question 4:

Overfitting and Regularization

Build a deep neural network on any classification dataset and:

```
from sklearn.datasets import load_breast_cancer

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.utils import to_categorical


# Load dataset

data = load_breast_cancer()

X = data.data

y = data.target
```

```
# Flatten images
X_train = X_train.reshape(-1, 784) / 255.0
X_test = X_test.reshape(-1, 784) / 255.0


# One-hot encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Model Without Regularization (Overfitting Case)

Model Architecture

- Hidden layers: 3
- Neurons: 64 → 64 → 32
- Activation: ReLU

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


model_no_reg = Sequential([
    Dense(64, activation='relu', input_shape=(30,)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])


model_no_reg.compile(
    optimizer='adam',
```

```
    loss='binary_crossentropy',
    metrics=['accuracy']
)
history_no_reg = model_no_reg.fit(
    X_train, y_train,
    epochs=50,
    validation_split=0.2,
    verbose=0
)
```

Model With Dropout Regularization

Regularized Architecture

- Dropout rate: 0.5
- Same network depth

```
from tensorflow.keras.layers import Dropout


model_dropout = Sequential([
    Dense(64, activation='relu', input_shape=(30,)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

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

history_dropout = model_dropout.fit(
    X_train, y_train,
    epochs=50,
    validation_split=0.2,
    verbose=0
)
```

Performance Comparison

```
acc_no_reg = model_no_reg.evaluate(X_test, y_test, verbose=0)[1]
acc_dropout = model_dropout.evaluate(X_test, y_test, verbose=0)[1]

print("Accuracy without regularization:", acc_no_reg)
print("Accuracy with Dropout:", acc_dropout)
```

```
Accuracy without regularization: 0.9720279574394226
Accuracy with Dropout: 0.9720279574394226
```

LAB Assessment

Student Name		**LAB Rubrics**	CLO3 , P5, PLO5
		Total Marks	10
Registration No		**Obtained Marks**	
		Teacher Name	Dr. Syed M Hamedoon
Date		**Signature**	