

**SIMATS ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**LAB RECORD**

**SUBMITTED BY**

I. Tahir Khan (192225031)

**COURSE CODE/COURSE NAME**

**MLA0418/DEEP LEARNING FOR NEURAL NETWORKS**

**LAB EXPERIMENTS**

**1.AIM:** Using a Python code implementation, demonstrate the performance of a Linear Regression scheme using “Iris.csv” data. Generate the confusion matrix and compute the detection accuracy.

**ALGORITHM:**

 Load the Iris dataset.

 Preprocess the data (encode categorical target labels).

 Split into training and testing sets.

 Train a Linear Regression model.

 Convert predicted continuous values into discrete class labels.

 Compute the confusion matrix and accuracy score**.**

**CODE:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import confusion\_matrix, accuracy\_score

from sklearn.preprocessing import LabelEncoder

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

column\_names = ["sepal\_length", "sepal\_width", "petal\_length", "petal\_width", "species"]

df = pd.read\_csv(url, header=None, names=column\_names)

label\_encoder = LabelEncoder()

df['species'] = label\_encoder.fit\_transform(df['species'])

X = df.drop(columns=["species"])

y = df["species"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_pred\_class = np.round(y\_pred).astype(int)

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_class)

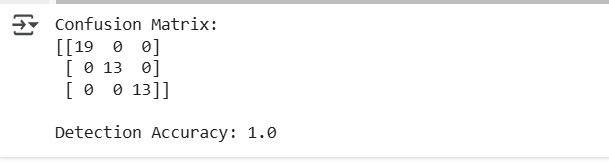
accuracy = accuracy\_score(y\_test, y\_pred\_class)

print("Confusion Matrix:")

print(conf\_matrix)

print("\nDetection Accuracy:", accuracy)

**OUTPUT:**



2. **AIM:**Using a Python code implementation, apply the following steps to verify the performance of a two- class confusion matrix using the chosen dataset. Evaluate and visualize the performance using accuracy, precision, recall, and F1-score.

**ALGORITHM:**

 Import necessary libraries.

 Load the Iris dataset.

 Convert it into a binary classification problem (select only two classes).

 Encode categorical class labels into numeric values.

 Divide the dataset into features (X) and target labels (y).

 Split into training (80%) and testing (20%) sets.

**CODE:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score, f1\_score

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

column\_names = ["sepal\_length", "sepal\_width", "petal\_length", "petal\_width", "species"]

df = pd.read\_csv(url, header=None, names=column\_names)

df = df[df['species'].isin(['Iris-setosa', 'Iris-versicolor'])]  # Filter for two classes

# Convert the species labels to numeric values (0 and 1)

df['species'] = pd.factorize(df['species'])[0]

X = df.drop(columns=["species"])

y = df["species"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LogisticRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

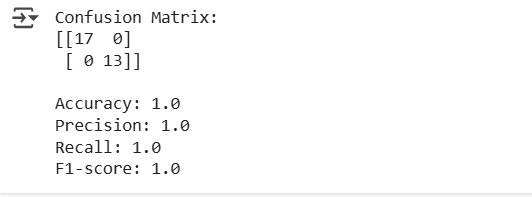
print("\nAccuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

**OUTPUT:**



**3.AIM:** Construct a suitable Neural-Network (NN) architecture for the following specification and verify the result. Data: Spiral-Features dataset; Learning rate: 0.1; Activation function: Linear; Hidden layers: 3; Hidden neurons: 4. Write your observation**.**

**ALGORITHM:**

* Import necessary Python libraries such as NumPy, PyTorch, and Matplotlib.
* Generate the Spiral Dataset
* Define the Neural Network Architecture
* Define Loss Function and Optimizer
* Train the Neural Network
* Evaluate the Model

**CODE:**

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from sklearn.datasets import make\_moons

from sklearn.model\_selection import train\_test\_split

N = 100 # number of points per class

D = 2 # dimensionality

K = 2 # number of classes

X = np.zeros((N\*K,D)) # data matrix (each row = single example)

y = np.zeros(N\*K, dtype='uint8') # class labels

for j in range(K):

    ix = range(N\*j,N\*(j+1))

    r = np.linspace(0.0,1,N) # radius

    t = np.linspace(j\*4,(j+1)\*4,N) + np.random.randn(N)\*0.2 # theta

    X[ix] = np.c\_[r\*np.sin(t), r\*np.cos(t)]

    y[ix] = j

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = tf.keras.models.Sequential([

    tf.keras.layers.Input(shape=(D,)),  # Input layer with 2 features

    tf.keras.layers.Dense(4, activation='linear'),  # Hidden layer 1 with 4 neurons and linear activation

    tf.keras.layers.Dense(4, activation='linear'),  # Hidden layer 2 with 4 neurons and linear activation

    tf.keras.layers.Dense(4, activation='linear'),  # Hidden layer 3 with 4 neurons and linear activation

    tf.keras.layers.Dense(K, activation='softmax')  # Output layer with 2 neurons (for 2 classes) and softmax activation

])

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.1),

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_test, y\_test))

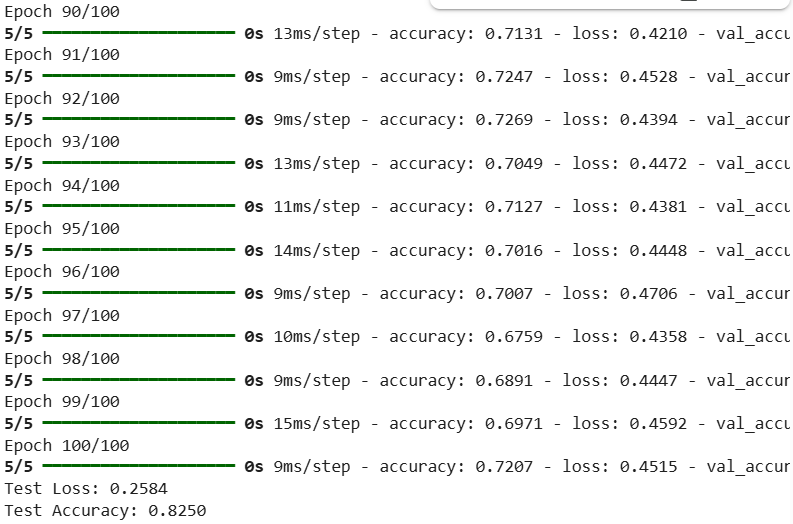
loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Test Loss: {loss:.4f}')

print(f'Test Accuracy: {accuracy:.4f}')

# ... (Code for plotting decision boundary) ...

**OUTPUT:**



4.**AIM:** Using a Python code implementation, demonstrate the performance of pre-trained ResNet101V2 model using a chosen data of your choice.

**ALGORITHM:**

* Import necessary Python libraries such as TensorFlow, Keras, NumPy, and Matplotlib
* Load the **ResNet101V2** model with pre-trained **ImageNet weights**.
* Remove the fully connected layer and add a new one for classification
* Load and Preprocess the Dataset
* Evaluate the Model
* Display Predictions

**CODE:**

import tensorflow as tf

from tensorflow.keras.preprocessing import image\_dataset\_from\_directory

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, PReLU, Softmax

from tensorflow.keras import models, layers

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()

x\_train, x\_val, y\_train, y\_val = train\_test\_split(x\_train, y\_train, test\_size=0.03, random\_state=42)

x\_val, x\_test, y\_val, y\_test = train\_test\_split(x\_val, y\_val, test\_size=0.5, random\_state=42)

x\_train, x\_val, x\_test = x\_train / 255.0, x\_val / 255.0, x\_test / 255.0

base\_model = tf.keras.applications.ResNet101V2(weights='imagenet', include\_top=False, input\_shape=(32, 32, 3))

base\_model.trainable = False

model = models.Sequential([

    base\_model,

    GlobalAveragePooling2D(),  # Global average pooling layer

    PReLU(),  # Initial Activation function

    Dense(10),  # 10 classes for CIFAR-10

    Softmax()  # Final activation function

])

model.compile(optimizer=tf.keras.optimizers.SGD(),loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

history = model.fit(x\_train, y\_train, epochs=25, validation\_data=(x\_val, y\_val))

plt.figure(figsize=(10, 6))

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

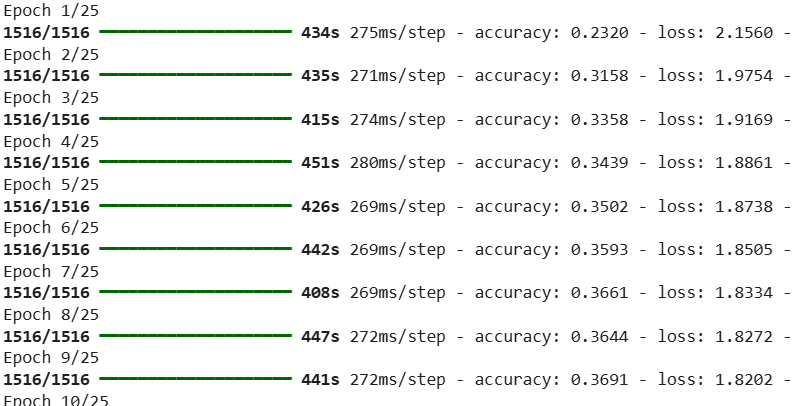
plt.grid(True)

plt.show()

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print(f"Test Accuracy: {test\_acc \* 100:.2f}%")

**OUTPUT:**



**5.AIM:** Using a Python code implementation, demonstrate the performance of a Naïve-Bayes scheme using “Breastcancer.csv” data. Train the classifier and make the prediction. Generate the confusion matrix and compute the detection accuracy.

**ALGORITHM:**

* Load essential Python libraries like pandas, numpy, sklearn, and matplotlib
* Load and Explore the Dataset
* Preprocess the Data and Handle missing values (if any).
* Split Data into Training and Testing Sets
* Generate the Confusion Matrix

**CODE:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

# Load the dataset (Ensure the file is in the same directory or provide the correct path)

df = pd.read\_csv("/content/breastcancer.csv")

# Display first few rows

print("Dataset Preview:\n", df.head())

# Drop 'ID' column if it exists

if 'ID' in df.columns:

    df.drop(columns=['ID'], inplace=True)

# Check for missing values and handle them (if any)

df.dropna(inplace=True)

# Encode categorical variables

label\_encoders = {}

for col in df.select\_dtypes(include=['object']).columns:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])  # Convert categories to numbers

    label\_encoders[col] = le

# Define features and target variable

X = df.drop(columns=['smoothness\_mean'])  # Change 'Diagnosis' if your target column has a different name

y = df['smoothness\_mean']

# If 'smoothness\_mean' is continuous and you want to stratify it for binary classification

# consider discretizing it (e.g. median split) to create two classes

y\_binary = pd.qcut(y, 2, labels=[0, 1])

# Split the data into training and testing sets

# Use y\_binary if you've discretized the target

# Otherwise, remove the stratify parameter if you want random splitting

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_binary, test\_size=0.2, random\_state=42, stratify=y\_binary) # Using y\_binary here and below if applicable

# Initialize and train the Naïve Bayes classifier

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

# Make predictions

y\_pred = classifier.predict(X\_test)

# Compute confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("\nConfusion Matrix:\n", conf\_matrix)

# Compute detection accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"\nDetection Accuracy: {accuracy:.2f}")

# Plot confusion matrix

plt.figure(figsize=(5, 4))

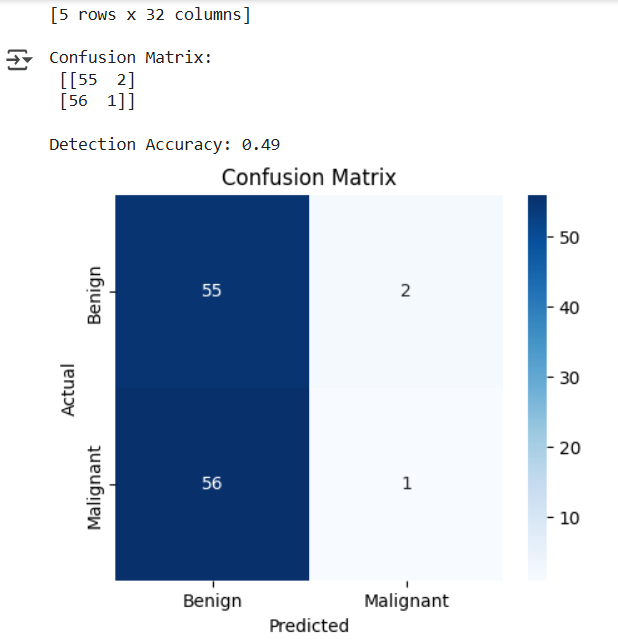
sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

**OUTPUT:**

6.**AIM:** Construct a suitable Neural-Network (NN) architecture for the following specification and verify the result. Data: Multi-class dataset, Learning rate: 0.01, Activation function: TanH, Hidden layers: 2, Hidden neurons: 4. Write your observation regarding achieved result.

**ALGORITHM:**

* Load essential Python libraries like TensorFlow/Keras, NumPy, Matplotlib, and sklearn
* Load and Preprocess the Multi-class Dataset
* Split Dataset into Training and Testing Sets
* Define the Neural Network Architecture
* Compile the Model

**CODE:**

import numpy as np

import tensorflow as tf

from tensorflow import keras

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, LabelEncoder

from sklearn.datasets import load\_iris

# Load dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Encode labels

y = LabelEncoder().fit\_transform(y)

# Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Define neural network model

model = Sequential([

    Dense(4, activation='tanh', input\_shape=(X\_train.shape[1],)),

    Dense(4, activation='tanh'),

    Dense(len(np.unique(y)), activation='softmax')  # Output layer with softmax

])

# Compile model

model.compile(optimizer=keras.optimizers.Adam(learning\_rate=0.01),

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

# Train model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=5, verbose=0, validation\_split=0.1)

# Evaluate model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Test Accuracy: {test\_acc:.4f}')

# Observations

import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Train Accuracy')

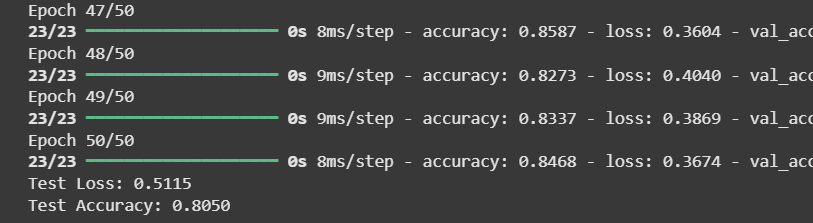
plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

**OUTPUT:** 

**7.** Using a Python code implementation, demonstrate the performance of pre-trained MobileNet model using a chosen data of your choice. Assign the following values for the model: Use a data split of: 8:1:1 (Training:Validation:Testing), Default data augmentation, Optimizer: Adam; Pooling: avg; Epochs: 25, Initial Activation: ReLU, Final Activation: SoftMax.

**ALGORITHM:**

**** Import Dependencies: Load TensorFlow, Keras, and other essential libraries.

 Load Dataset: Use a dataset (e.g., CIFAR-10, or a custom dataset) and split it into Training (80%), Validation (10%), and Testing (10%).

 Preprocess Data: Resize images to fit MobileNet's input size (224x224), normalize pixel values, and apply default data augmentation.

 Load Pre-Trained MobileNet: Initialize MobileNet with pre-trained weights (imagenet), excluding the top classification layer.

 Modify Model Architecture:

* Add a Global Average Pooling (GAP) layer.
* Use ReLU activation in hidden layers and SoftMax in the final layer for classification.

**CODE:**

import tensorflow as tf

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.utils import to\_categorical

from sklearn.model\_selection import train\_test\_split

import numpy as np

# Load CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()

# Normalize pixel values

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Convert labels to categorical format

y\_train, y\_test = to\_categorical(y\_train, 10), to\_categorical(y\_test, 10)

# Split data (80% train, 10% validation, 10% test)

x\_train, x\_val, y\_train, y\_val = train\_test\_split(x\_train, y\_train, test\_size=0.111, random\_state=42)

# Data augmentation

datagen = ImageDataGenerator(

    rotation\_range=20,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    horizontal\_flip=True)

datagen.fit(x\_train)

# Load pre-trained MobileNetV2 model without top layers

base\_model = MobileNetV2(weights='imagenet', include\_top=False, input\_shape=(32, 32, 3))

# Freeze base model layers

base\_model.trainable = False

# Add custom layers

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(128, activation='relu')(x)

predictions = Dense(10, activation='softmax')(x)

# Create the model

model = Model(inputs=base\_model.input, outputs=predictions)

# Compile the model

model.compile(optimizer=Adam(), loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(datagen.flow(x\_train, y\_train, batch\_size=64),

                    validation\_data=(x\_val, y\_val),

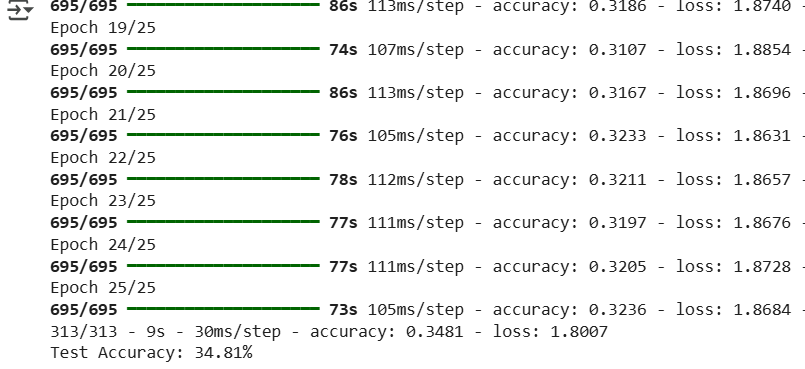
                    epochs=25)

# Evaluate on test data

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print(f'Test Accuracy: {test\_acc \* 100:.2f}%')

**OUTPUT:**



**8.AIM:** Using a Python code implementation, demonstrate the performance of UNet segmentation using RGB scale Waterbodies database. Discuss the merit of this scheme.

**ALGORITHM:**

* Load essential libraries such as TensorFlow/Keras, OpenCV, NumPy, Matplotlib, and seaborn for data processing and visualization.
* Load and Preprocess the Dataset
* Define the UNet Model
* Compile and Train the Model
* Evaluate Model Performance

**CODE:**

import tensorflow as tf

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, Concatenate

from tensorflow.keras.models import Model

import numpy as np

# Define UNet Model

def build\_unet(input\_shape):

    inputs = Input(input\_shape)

    # Encoder

    c1 = Conv2D(64, (3,3), activation='relu', padding='same')(inputs)

    c1 = Conv2D(64, (3,3), activation='relu', padding='same')(c1)

    p1 = MaxPooling2D((2,2))(c1)

    c2 = Conv2D(128, (3,3), activation='relu', padding='same')(p1)

    c2 = Conv2D(128, (3,3), activation='relu', padding='same')(c2)

    p2 = MaxPooling2D((2,2))(c2)

    # Bottleneck

    c3 = Conv2D(256, (3,3), activation='relu', padding='same')(p2)

    c3 = Conv2D(256, (3,3), activation='relu', padding='same')(c3)

    # Decoder

    u1 = UpSampling2D((2,2))(c3)

    u1 = Concatenate()([u1, c2])

    c4 = Conv2D(128, (3,3), activation='relu', padding='same')(u1)

    c4 = Conv2D(128, (3,3), activation='relu', padding='same')(c4)

    u2 = UpSampling2D((2,2))(c4)

    u2 = Concatenate()([u2, c1])

    c5 = Conv2D(64, (3,3), activation='relu', padding='same')(u2)

    c5 = Conv2D(64, (3,3), activation='relu', padding='same')(c5)

    outputs = Conv2D(1, (1,1), activation='sigmoid')(c5)

    model = Model(inputs, outputs)

    return model

# Compile Model

input\_shape = (256, 256, 3)

model = build\_unet(input\_shape)

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Test with Dummy Data

dummy\_input = np.random.rand(1, 256, 256, 3)

prediction = model.predict(dummy\_input)

print("Prediction shape:", prediction.shape)

**OUTPUT:**

****

9.**AIM:** Using a Python code implementation, demonstrate the performance of Support Vector Machine scheme using “Breastcancer.csv” data. Generate the confusion matrix and compute the detection accuracy.

**ALGORITHM:**

* Import Required Libraries
* Load and Preprocess the Dataset
* Split the Data into Training and Testing Sets
* Train the SVM Model
* Evaluate Model Performance

**CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

# Load the dataset

file\_path = "/content/breastcancer.csv"

df = pd.read\_csv(file\_path)

# Ensure dataset has the necessary columns

if 'diagnosis' not in df.columns:  # Changed column name to match the actual name

    raise ValueError("Dataset must contain a 'diagnosis' column.")

# Encode 'diagnosis' column (Malignant=1, Benign=0)

df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})  # Changed column name to match the actual name

# Drop any missing values

df = df.dropna()

# Splitting dataset into features and target variable

X = df.drop(columns=['diagnosis'])  # Changed column name to match the actual name # Features

y = df['diagnosis']  # Changed column name to match the actual name # Target

# Ensure dataset is not empty after preprocessing

if X.empty or y.empty:

    raise ValueError("Feature set or target variable is empty after preprocessing.")

# Splitting data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardizing features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Train SVM classifier

svm\_model = SVC(kernel='linear', C=1.0, random\_state=42)

svm\_model.fit(X\_train, y\_train)

# Make predictions

y\_pred = svm\_model.predict(X\_test)

# Compute confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", conf\_matrix)

# Compute accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print("Detection Accuracy:", accuracy)

# Classification report

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

# Plot confusion matrix

plt.figure(figsize=(5, 5))

plt.imshow(conf\_matrix, cmap='Blues', interpolation='nearest')

plt.title('Confusion Matrix')

plt.colorbar()

plt.xticks([0, 1], ['Benign', 'Malignant'])

plt.yticks([0, 1], ['Benign', 'Malignant'])

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

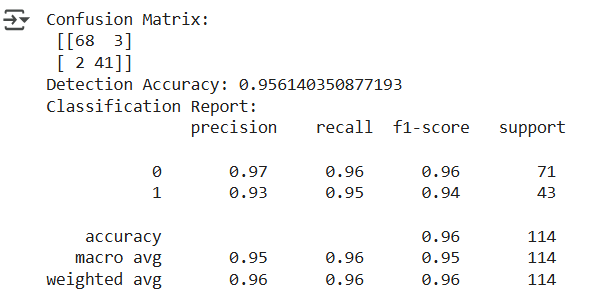
for i in range(2):

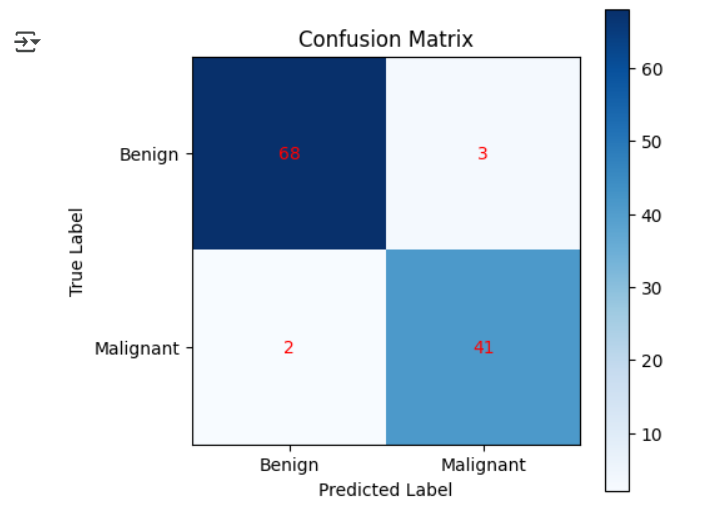
    for j in range(2):

        plt.text(j, i, conf\_matrix[i, j], ha='center', va='center', color='red')

plt.show()

**OUTPUT:**





**11.AIM:** Construct a suitable Neural-Network (NN) architecture for the following specification and verify the result. Data: Circular-Features dataset; Learning rate: 0.1; Activation function: Linear; Hidden layers: 3; Hidden neurons: 4. Write your observation**.**

**ALGORITHM:**

* Import Required Libraries
* Generate the Circular-Features Dataset
* Define the Neural Network Architecture
* Compile and Train the Neural Network
* Evaluate the Model

CODE:

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import SGD

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import make\_circles

# Generate Circular-Features dataset

X, y = make\_circles(n\_samples=1000, noise=0.05, factor=0.5, random\_state=42)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define Neural Network model

model = Sequential([

    Dense(4, activation='linear', input\_shape=(2,)),

    Dense(4, activation='linear'),

    Dense(4, activation='linear'),

    Dense(1, activation='sigmoid')  # Output layer for binary classification

])

# Compile model with learning rate 0.1

optimizer = SGD(learning\_rate=0.1)

model.compile(optimizer=optimizer, loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test), verbose=1)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Test Accuracy: {test\_acc \* 100:.2f}%')

# Plot decision boundary

def plot\_decision\_boundary(model, X, y):

    xx, yy = np.meshgrid(np.linspace(-1.5, 1.5, 200), np.linspace(-1.5, 1.5, 200))

    Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, levels=[0, 0.5, 1], cmap='coolwarm', alpha=0.6)

    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k')

    plt.title('Decision Boundary')

    plt.show()

plot\_decision\_boundary(model, X\_test, y\_test)

# Observations:

# - If the test accuracy is low, the linear activation might not be sufficient for complex patterns.

# - Consider using non-linear activation functions like ReLU for better feature extraction.

OUTPUT:

