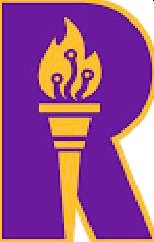
RAJALAKSHMI ENGINEERING COLLEGE

# RAJALAKSHMI NAGAR, THANDALAM - 602 105



LABORATORY LAB MANUAL

AI23521 — BUILD AND DEPLOY MACHINE LEARNING APPLICATIONS



REGISTER NT\*hIBER: 2116 231ñ010ñ 7

YE.OR / BR.4NC’II / SEC TIOh: III h’EAR / .CIVIL / A

SESIESTER: ¥" SEAIESTER

AC.4DEhIIC’ h"E.4R: 202S—2026

RAJALAKSHMI ENGINEERING COLLEGE

BONAFIDE CERTIFICATE

##### CERTIFIED THAT THIS LABORATORY RECORD REPORT FOR **“BUILD** AND **DEPLOY** MACHINE LEARNING APPLICATIONS" IS THE BONAFIDE WORK OF "HARISH KUMAR V [231501057]” WHO CARRIED OUT THE PRACTICAL WORK UNDER MY SUPERVISION.

Submitted for the Practical Examination held on

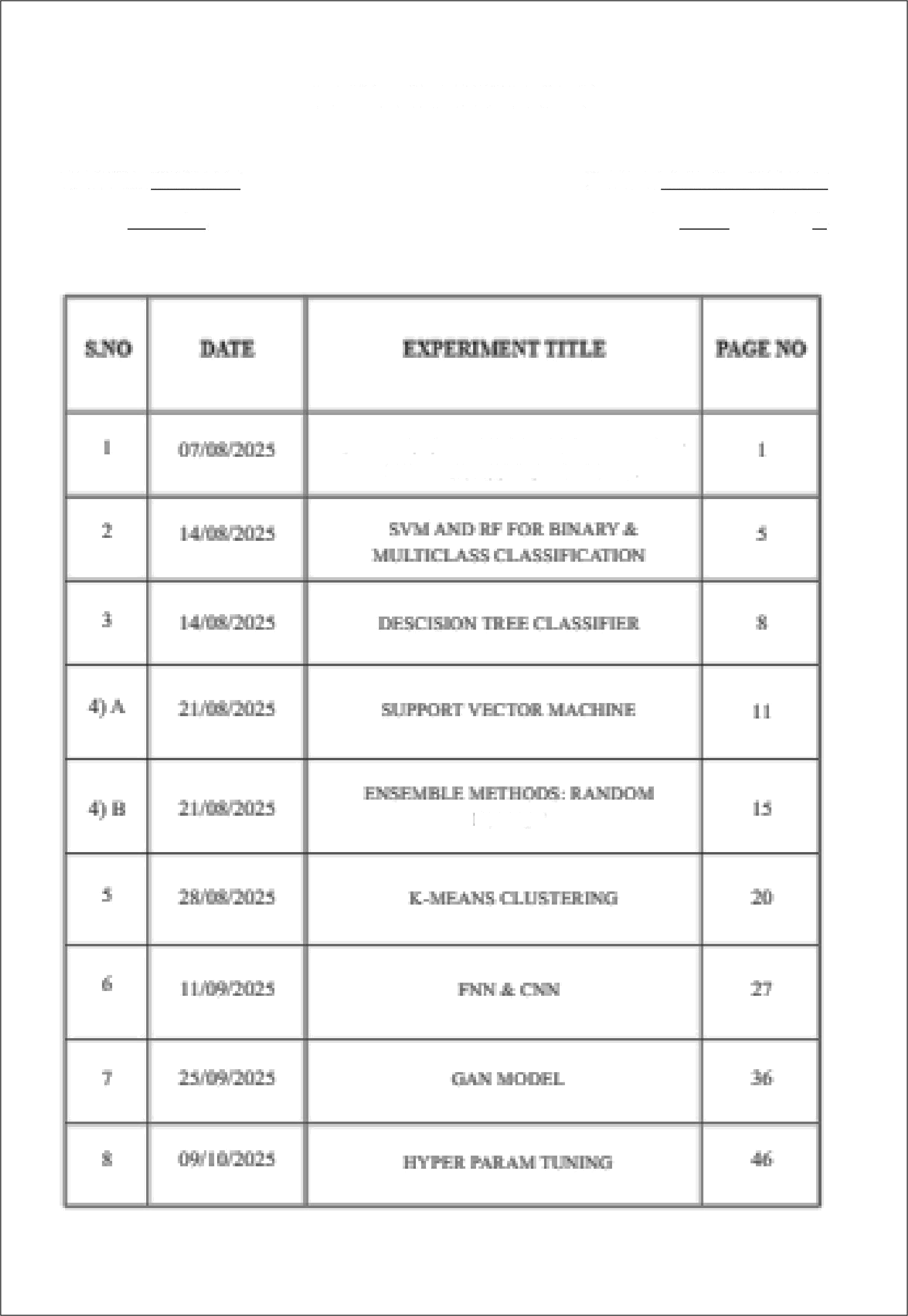
##### SIGNATURE

###### Mr. Vijayabhaskar V, AIML, REC (Autonomous) Thandalam,

Chennai - **602 105**

INTERNAL EXAMINER EXTERNAL EXAMINER

# TABLE OF CONTENTS

REG NO: 231501057 YEAR: III YEAR

NAME: HARISHKUMAR V BRANCH: AIML SEC: A

SETTTI¥ r M\*’I¥gE ENYf9€Ig¥MENTABA PBEPBZgZEgBEN4Z DATA

FOAEET

|  |  |
| --- | --- |
| **EXP NO: 1** | **SETTING UP THE ENVIRONMENT AND PREPROCESSING THE DATA** |
| **DATE: 07/08/2025** |

#### AIM:

To set up a fully functional machine learning development environment and to perform data preprocessing operations like handling missing values, encoding categorical variables, feature scaling, and splitting datasets.

### ALGORITHM:

STEP 1: Install **Required Libraries:**

* Install numpy, pandas, matplotlib, seaborn, and scikit-learn using pip. STEP 2: Import Libraries.

STEP 3: Load Dataset:

* Load any dataset (e.g., Titanic or Iris) using pandas. STEP 4: Data Exploration:
* Use df.info(), df.describe(), df.isnull().sum() to understand the data. STEP 5: Handle Missing Values:
* Use .fillna() or .dropna() depending on the strategy. STEP 6: Encode Categorical Data:
* Use pd.get dummies() or LabelEncoder. STEP 7: Feature Scaling:
* Normalize or standardize the numerical features using Standard Sealer or

**MinMaxScaler. STEP 8:** Split Dataset:

* Use **train test split()** from **sklearn** to create training and testing sets.

**STEP 9:** Display the **Preprocessed** Data.

1

#### CODE:

# 1. Install necessary libraries (if not already installed) # !pip install numpy pandas matplotlib seaborn scikit-learn

# 2. Import libraries import pandas as pd import numpy as np

from sklearn.model selection import train test split

from sklearn.preprocessing import StandardScaler, LabelEncoder import seaborn as sns

import matplotlib.pyplot as plt

# 3. Load d at a set

df = sns.load dataset(’titanic’) # Titanic dataset df.head()

# 4. Explore the dataset print(df.info()) print(df.describe()) print(df.isnull().sum())

# 5. Handle missing values

# Fill age with median, embark town with mode df[’age’].fillna(df[’age’].median(), inplace-True)

df[’embark town’].fillna(df[’embark town’].mode()[0], inplace=True) df.drop(columns=[’deck’], inplace=True) # too many missing values

# 6. Encode categorical variables

# Convert ’sex’ and ’embark town’ using LabelEncoder le = LabelEncoder()

df[’sex’] = le.fit transform(df[’sex’])

df[’embark town’] = le.fit transform(df[’embark town’])

# Drop non-informative or redundant columns

df.drop(columns=[’embarked’, ’class’, ’who’, ’alive’, ’adult male’, ’alone’], inplace=True)

# 7. Feature Scaling scaler = StandardScaler()

numerical cols - [’age’, ’fare’]

df[numerical cols] = scaler.fit transform(df[numerical cols])

# 8. Split dataset

# Define features (X) and label (y)

X = df.drop(’survived’, axis=1) y = df[’survived’]

X train, X test, y train, y test = train test split(X, y, test size=0.2,

random state=42)

# 9. Show final preprocessed data print(“Training Data Shape:”, X train.shape) print(“Test Data Shape:”, X test.shape)

X train.head()

#### OUTPUT:

‹ c la s s ' pa nd as . r one . £ rame . Da I of name ’ › Ra nge I ndcx : 8°3 I en t r i es , 0 t o 890

Oa t a r o1 umn s ( I ot a 1 1 5 c o1 mn s ) :

# L o1 r mn I Jon tiu11 € oun t Dt ype

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | s u cv i ve d | 891 | non - n u 1 1 | i n t 6d |
| 1 | pc 1 as s | 891 | non - nu 11 | i n t bd |
| 2 | sex | 89 1 | iaon - n u 1 1 | ob j ec I |
| *?* | age | 7 l4 | r4on - n u 11 | 11 oa t64 |
| 4 | s i bsp | 89 I | no'a - n u11 | i r› t64 |
| 5 | pa r c to | 89 1 | Jo-iJ n \ 11 | in t 64 |
| 6 | f a ne | 89 I | no-n rJu11 | f 1oa t 64 |
| 7 | emb anke d | 889 | n on- nu 11 | ob j e cI |
| 8 | c 1a s s | 89 1 | n on- nr 11 | c at e go n\‘ |
| 9 | ciii o | B9 1 | n on- nr 11 | ob j e cI |
| 10 | adu 1t ma 1e | 89 1 | non- nu T1 | bo a 1 |
| 11 | de c k | 203 | non- nu11 | c at e gon \’ |

12 *ebb ar k* I o\in B89 non - nu 11 ob j e cI

13 a1ive 89 1 no-n nu11 ob je c t

14 a1on e 89 1 no-n nu T1 boo 1

dt\’p e s : boo I ( 2) , c aI e go r\’ ( 2) , 11o a t6A ( 2) , ! n t64 ( 4) , ob je c t ( 5) memo ny u sa ge : 80 . 7+ KB

flone

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | survived | pclass | age | sibsp | parch | Care |
| count | 891.OOO0O0 | 891.000000 | 714.000000 | 891.000000 | 891.000000 | 891.000000 |
| mean | 0.3838?8 | 2.308642 | 29.699118 | 0. h23O08 | 0. 381594 | 32. 204208 |
| std | 0.486592 | 0. 836071 | IA. 526497 | 1. 1027d3 | 0. 806057 | 49.693429 |
| ruin | 0.000000 | 1.000000 | 0.420000 | 0.O0O00O | 0.000000 | 0.000000 |
| 25Z | 0.O0O0O0 | 2.000000 | 20.125000 | 0.0OO00O | 0.000080 | 7.910400 |
| 50% | 0.000000 | 3.000000 | 28.000000 | 0.OOO000 | 0.000000 | 14.J542O0 |
| 75% | 1.000000 | 3.000000 | 38.000000 | 1.000000 | 0.000000 | 31.000000 |
| max | 1.080000 | 3.000000 | 80.000000 | 8.000000 | 6.000000 | 512.329200 |

sur 'i.ed rclass sex

age silsp parch

embarked class v,ho

adult male deck

embark toi:n

ali.e

a I on e

0

0

G

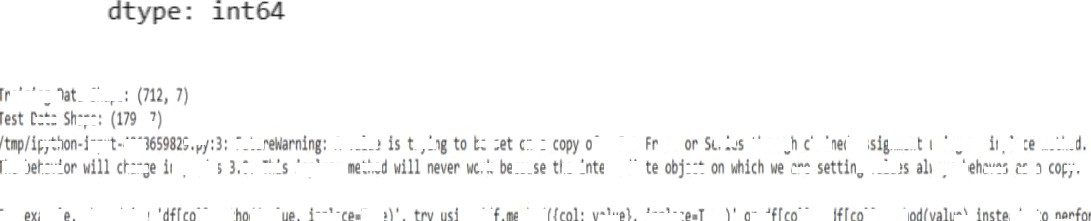
177

0

0

2

0



#### RESULT:

Thus, the execution successfully set up a complete machine learning environment and performed data preprocessing operations, including handling missing values, encoding categorical features, feature scaling, and splitting the dataset for model training and testing.

|  |  |  |
| --- | --- | --- |
|  | EXP NO: 2 | SUPPORT VECTOR MACHINE (SVM) AND RANDOM FOREST FOR BINARY & MULTICLASS CLASSIFICATION |
|  |
| DATE: 14/08/2025 | |

#### AIM:

To build classification models using Support Vector Machines (SVM) and Random Forest, apply them to a dataset, and evaluate the models using performance metrics like accuracy and confusion matrix.

### ALGORITHM:

* PART A: SVM MODEL

STEP 1: Import necessary libraries STEP 2: Load and explore the dataset STEP 3: Handle missing values if any STEP 4: Encode categorical variables

STEP 5: Split dataset into training and testing sets STEP 6: Build SVM classifier using SVC() STEP 7: Train and predict

STEP 8: Evaluate the model using accuracy and confusion matrix

* **PART B:** RANDOM FOREST **MODEL**

**STEP** 1: Initialize Random Forest using RandomForestClassifier()

**STEP** 2: Train and predict

**STEP** 3: Evaluate and compare with SVM

#### CODE:

# 1. Import libraries

|  |  |  |
| --- | --- | --- |
| import pandas as pd |  | |
| from sklearn.datasets import  from sklearn.model selection | load iris  import train | test split |

from sklearn.preprocessing import StandardScaler from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy score, confusion matrix import seaborn as sns

import matplotlib.pyplot as pit

# 2. Load dataset iris = load iris()

X = iris.data

y = iris.target

# 3. Feature scaling scaler = StandardScaler()

X scaled = scaler.fit transform(X)

# 4. Train-test split

X train, X test, y train, y test = train test split(X scaled, y, test size=0.3, random state=42)

# Part A: SUPPORT VECTOR MACHINE

# 5. Initialize and train SVM

svm model = SVC(kernel='linear') # You can also try 'rbf', 'poly' svm model.fit(X train, y train)

# 6. Predict and evaluate SVM

y pred svm = svm model.predict(X test)

print(“SVM Accuracy:“, accuracy score(y test, y pred svm))

print(“SVM Confusion Matrix:\n", confusion matrix(y test, y pred svm)) # Part B: RANDOM FOREST

# 7. Initialize and train Random Forest

rf model = RandomForestClassifier(n estimators=100, random state=42) rf model.fit(X train, y train)

# 8. Predict and evaluate Random Forest y pred rf = rf model.predict(X test)

print(“Random Forest Accuracy:“, accuracy score(y test, y pred rf))

print(“Random Forest Confusion Matrix:\n“, confusion matrix(y test, y pred rf))

# 9. Visual comparison using seaborn heatmap

plt.subplot(1, 2, 1)

sns.heatmap(confusion matrix(y test, y pred svm), e,

plt.title(”SVM Confusion Matrix”)

plt.subplot(1, 2, 2)

sns.heatmap(confusion matrix(y test, y pred rf), -True,

plt.title(”Random Forest Confusion Matrix”) plt.tight layout()

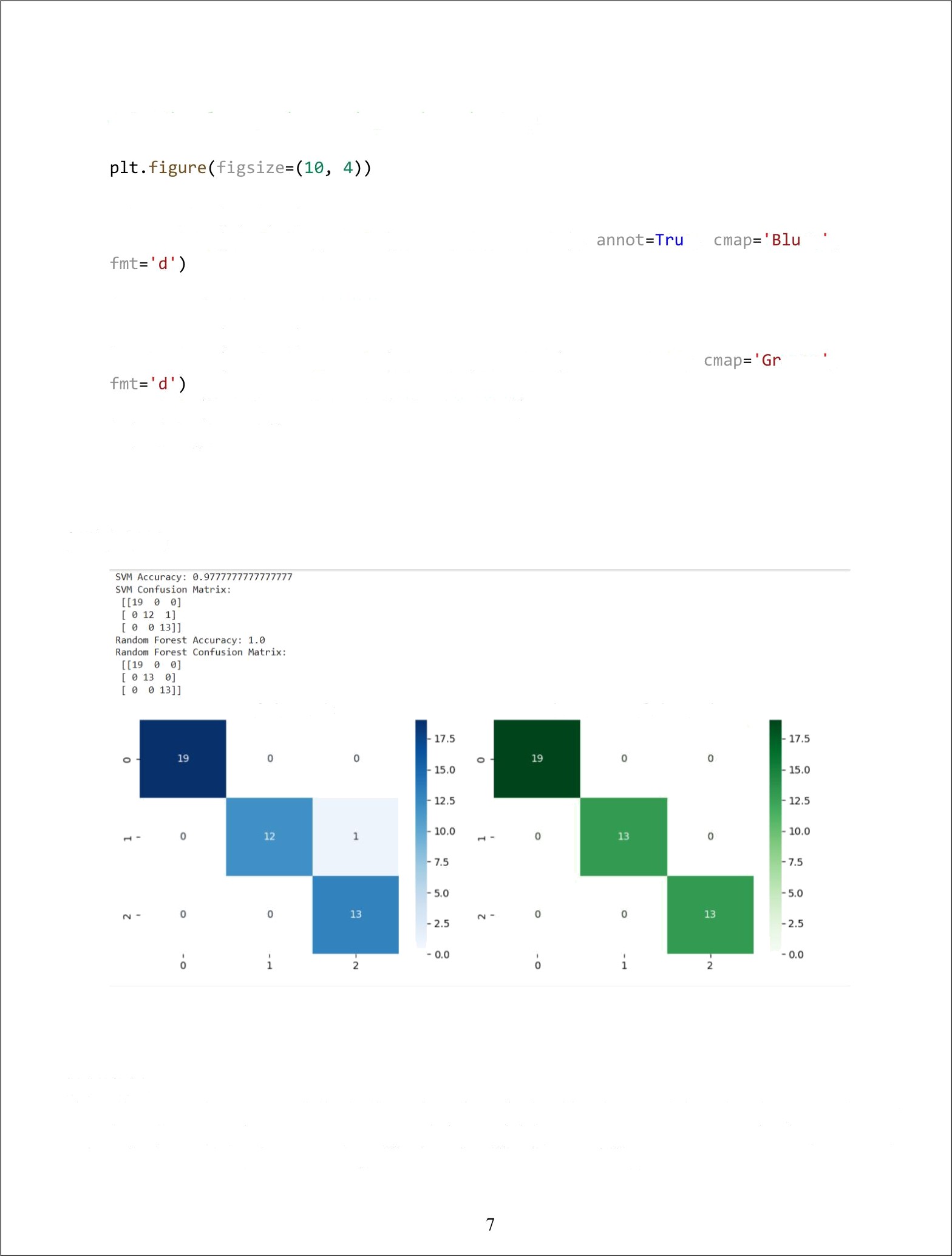
plt.show()

#### OUTPUT:

SVM Confusion Natrix Random Forest Confusion Matrix

es ,

eens ,



RESULT:

Thus, the execution successfully built and evaluated classification models using Support Vector Machines (SVM) and Random Forest, and the models' performance was assessed using accuracy and confusion matrix, demonstrating effective classification capability.

|  |  |
| --- | --- |
| EXP NO: 3 | **DECISION TREE CLASSIFIER** |
| DATE: 14/08/2025 |

#### AIM:

To implement a Decision Tree classified and evaluate its performance using accuracy score and confusion matrix on a real-world dataset.

### ALGORITHM:

**STEP** 1: Import necessary libraries

**STEP** 2: Load a classification dataset (e.g., Iris or Titanic)

**STEP 3:** Split the dataset into training and test sets

**STEP** 4: Preprocess data if needed

**STEP 5:** Train a DecisionTreeClassifier from sklearn.tree

**STEP 6:** Predict on test data

**STEP** 7: Evaluate using:

* Confusion Matrix
* Accuracy Score

**STEP 8:** Visualize the **Decision Tree** (optional)

#### CODE:

# Step 1: Import Libraries

from sklearn.datasets import load iris

from sklearn.tree import DecisionTreeClassifier, plot tree from sklearn.model selection import train test split

from sklearn.metrics import confusion matrix, accuracy score

###### 8

import matplotlib.pyplot as plt import seaborn as sns

# Step 2: Load Dataset iris = load iris()

X = iris.data

y = iris.target

# Step 3: Split the dataset

X train, X test, y train, y test = train test split(X, y,

=42)

# Step 4: Train the Decision Tree Classifier

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| dt  dt | model = DecisionTreeClassifier(.!  model.fit(X train, y train) | | ’ | -'gini', |
| # | Step | 5: Predict | | |
| y | pred | = dt model.predict(X test) | | |

# Step 6: Evaluate the Model

cm = confusion matrix(y test, y pred) acc = accuracy score(y test, y pred) print(”Confusion Matrix: ”, cm) print(”Accuracy Score:”, acc)

# Step 7: Visualize Confusion Matrix sns.heatmap(cm, =True, ’=”Blues”, ..

=iris.target names) plt.xlabel(”Predicted”) plt.ylabel(”Actual”) plt.title(”Confusion Matrix”) plt.show()

=iris.target names,

# Step 8: Visualize the Decision Tree plt.figure( ..’.’-(12,8))



plot tree(dt model, =True,

=iris.target names) plt.title(”Decision Tree Visualization”) plt.show()

9

: ›-iris.feature names,

#### OUTPUT:

0 0 1 3 ] ]

## RESULT:

Thus, the execution successfully implemented SVM and Random Forest classification models, applied them to the given dataset, and evaluated their performance using accuracy and confusion

matrix to measure classification effectiveness.

10

|  |  |
| --- | --- |
| **EXP NO: 4) A** | **SUPPORT VECTOR MACHINE (SVM)** |
| **DATE: 21/08/2025** |

#### AIM:

To build an SVM model for a binary classification task, tune its hyperparameters, and evaluate it using accuracy, precision, recall, F1-score, confusion matrix, and ROC-AUC.

### ALGORITHM:

**STEP** 1: Import libraries: **numpy,** pandas, **matplotlib,** sklearn.

**STEP** 2: Load data: Use a standard binary dataset (Breast Cancer Wisconsin) from sklearn.datasets.

**STEP 3:** Train/Test split: **80/20** split with a fixed random state.

**STEP 4:** Preprocess: Standardize features (StandardScaler).

**STEP 5:** SVMs are sensitive to feature scale.

**STEP 6:** Model selection: Use SVC **(RBF** kernel).

**STEP** 7: Hyperparameter tuning: Grid search on C and gamma with cross-validation (GridSearchCV).

**STEP 8:** Train final model: Fit on training data using best parameters. **STEP 9:** Evaluate: Predict on test set; compute metrics and plot ROC curve. **STEP 10:** Report: Best params, metrics, and brief observations.

CODE:

# EXPERIMENT 4A — SVM (RBF)

# 1) Imports import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load breast cancer

from sklearn.model selection import train test split, GridSearchCV from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import (

accuracy score, precision score, recall score, f1 score, confusion matrix, classification report, roc auc score, roc curve

# 2) Load dataset (binary classification) data = load breast cancer()

X = pd.DataFrame(data.data, ›.=data.feature names)

y = pd.Series(data.target, ›: .=”target”) # 0 = malignant, 1 = benign

# 3) Train/test split

X train, X test, y train, y test = train test split(

X, y, ’ =0.20, .-42, ! -y

# 4) Standardize features (important for SVMs) scaler = StandardScaler()

X train sc = scaler.fit transform(X train)

X test sc = scaler.transform(X test)

# 5) Define model

svm = SVC( r .='rbf', =Tnue,

# 6) Hyperparameter grid & tuning param grid = {

”C”: [0.1, 1, 10, 100],

”gamma”: [”scale”, 0.01, 0.001, 0.0001]

grid = GridSearchCv(

=42)

=5,

›‹-param grid,

='f1', # You can change to 'accuracy' or 'roc auc'

=-1,

=0



grid.fit(X train sc, y train)

print(”Best Parameters from Grid Search:”, grid.best params ) best svm = grid.best estimator

# 7) Train final model & predict best svm.fit(X train sc, y train)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| y pred | = | best | svm . p nedi ct (X test | sc ) |  | |
| y prob | = | best | svm . p nedi ct p noba (X | test | sc ) [ : , | 1] |

# 8) Evaluation

acc = accuracy score(y test, y pred) prec = precision score(y test, y pred, rec = recall score(y test, y pred)

f1 = f1 score(y test, y pred)

auc = roc auc score(y test, y prob) cm = confusion matrix(y test, y pred)

print(” === SVM (RBF) — Test Metrics ===”) print(f”Accuracy : {acc:.4f}”) print(f”Precision: {prec:.4f}”) print(f”Recall : {rec:.4f}”)

print(f”F1-Score : {f1:.4f}”)

print(f”ROC-AUC : {auc:.4f}”)

=8)



print(” Confusion Matrix: ”, cm)

print(” Classification Report: ”, classification report(y test, y pred,

# 9) Plot ROC Curve

fpr, tpr, thresholds = roc curve(y test, y prob) plt.figure()

plt.plot(fpr, tpr, =f”SVM (AUC = {auc:.3f})“)

plt.plot([0, 1], [0, 1], =”--”, ='gray') plt.xlabel(”False Positive Rate”)

plt.ylabel(”True Positive Rate”) plt.title(”ROC Curve — SVM (RBF)”) plt.legend()

plt.grid(True) plt.show()

OUTPUT:

Best Parameters \*rom Grid Search: ('C': 10, 'gamma' 0.01}

|  |  |
| --- | --- |
| P rec i s hon : | 8 . 9861 |
| Re c a L1 | 8 . 9861 |
| FI-Score : | 0.9861 |
| ROC - AUC : | B . 9977 |

Confusion Matrix:

[ 1 71) ]

Classification Report:

precision recall {I-score support

0 0.98

1 0 . 9 9

0.98

0.99

0.98 42

0.99 72

a *cc* u r acy

mscnosvg

weighted avg

B . 9 8

0.9B

0.9B

0.9B

0.98

0.98

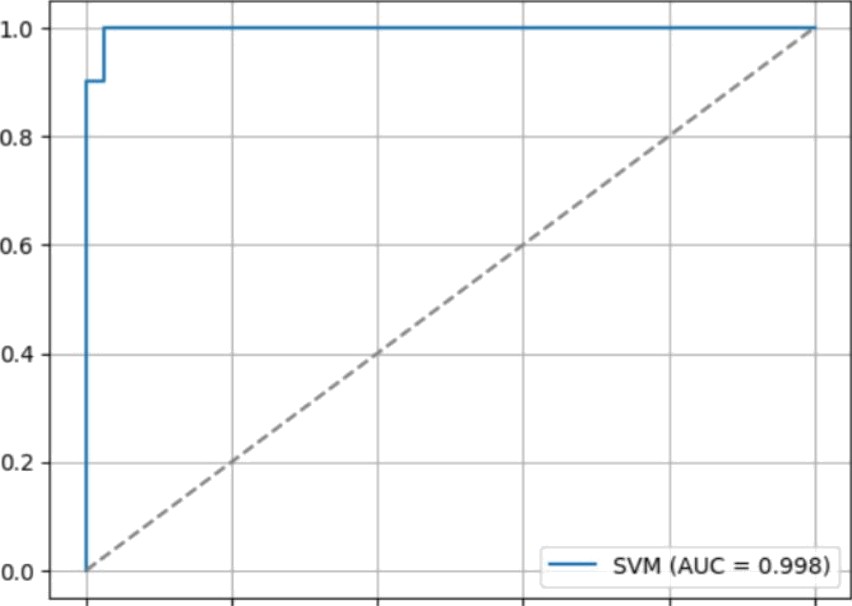
0.98

114

114

I14

ROC Curve — SVN (RBF)



0.0 02

0.4 0.6

False Positive Rate

0.8 1.0

##### RESULT:



True Positive Rate

Thus, the execution successfully built and tuned an SVM model for binary classification and evaluated its performance using accuracy, precision, recall, Fl-score, confusion matrix, and ROC-AUC, achieving effective model performance.

|  |  |
| --- | --- |
| **EXP NO: 4)** B | **ENSEMBLE METHODS: RANDOM FOREST** |
| **DATE: 21/08/2025** |

#### AIM:

To implement **a Random Forest classified** for a classification task, tune key hyperparameters, evaluate performance, and interpret feature **importance.**

### ALGORITHM:

**STEP 1:** Import libraries.

**STEP** 2: Load data (use same dataset to compare with SVM).

**STEP 3:** Train/Test split with stratification.

**STEP** 4: (Optional) Preprocess: Random Forests don't require scaling; we'll use raw features.

STEP 5: Model: RandomForestClassifier.

**STEP 6:** Hyperparameter tuning: Grid search over n estimators, max depth, min samples split, min samples leaf.

**STEP** 7: Train the best model on training data.

**STEP 8:** Evaluate with **Accuracy, Precision, Recall, F1, Confusion Matrix, ROC-AUC. STEP** 9: Interpretation: Plot top feature importances.

#### CODE:

# EXPERIMENT 4B — Random Forest Classifier

# 1) Imports import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load breast cancer

from sklearn.model selection import train test split, GridSearchCV

from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import (

accuracy score, precision score, recall score, f1 score,

confusion matrix, classification report, roc auc score, roc curve

# 2) Load dataset (same as 4A for comparison)

data = load bneast cancen ( )

X = pd . Data F name(data. data, .=data . -Featur'e names) y = pd . Senies (data . target, : ="I anget " )

# 3) Train/test split (no scaling needed for RF)

X train, X test, y train, y test = train test split(

X, y, ’ =0.20, .-42, ! -y

# 4) Define model

rf = RandomForestClassifier( .’ .’

# 5) Hyperparameter grid & tuning param grid = {

”n estimators“: [100], ”max depth“: [None, 10], ”min samples split”: [2], ”min samples leaf”: [1]

grid = GridSearchCv(

=3,

=param grid, ’=”f1“,

=-1,

=0)

grid.fit(X train, y train)



print(”Best Parameters (CV):”, grid.best params ) best rf = grid.best estimator

# 6) Train final model & predict best rf.fit(X train, y train)

y pred = best rf.predict(X test)

y prob = best rf.predict proba(X test)[:, 1]

# 7) Evaluate

acc = accuracy score(y test, y pred) prec = precision score(y test, y pred, rec = recall score(y test, y pred)

f1 = f1 score(y test, y pred)

auc = roc auc score(y test, y prob) cm = confusion matrix(y test, y pred)

print(” === Random Forest — Test Metrics ===” print(f”Accuracy : {acc:.4f}”) print(f”Precision: {prec:.4f}”)

print(f”Recall : {rec:.4f}”)

print(f”F1-Score : {f1:.4f}”)

print(f”ROC-AUC : {auc:.4f}”)

print(” Confusion Matrix: ", cm)

print(” Classification Report: ”, classification report(y test, y pred,

‹ ‹s‹ =0))

# 8) Feature Importance (Top 10)

importances = pd.Series(best rf.feature importances , .=X.columns) top10 = importances.sort values(. ‹: ›=False).head(10)

plt.figure()

top10[::-1].plot( :=”barh”) plt.xlabel(”Importance”)

plt.title(”Top 10 Feature Importances — Random Forest”)

plt.grid(.. :‹=”x”, =0.3) plt.show()

# 9) ROC Curve

fpr, tpr, thresholds = roc curve(y test, y prob) plt.figure()

plt.plot(fpr, tpr, =f”Random Forest (AUC = {auc:.3f})”)

plt.plot([0, l], [0, 1], -”--”,



plt.xlabel(”False Positive Rate") plt.ylabel(”True Positive Rate”) plt.title(”ROC Curve — Random Forest”) plt.legend()

plt.grid(True) plt.show()

! -'gray')

#### OUTPUT:

Best Parameters (CV): ('max depth': Tone, 'min samples leaf': 1, 'min samples split': 2, 'n estimators' : 100}

=== Random Forest- Test F\et ri c s ===

|  |  |
| --- | --- |
| Accuracy : | 0.9561 |
| Prec I s ion : | 0. 9389 |
| Rec a11 : | 0. 9722 |
| Fl-Score : | 0.9655 |
| ROC AUC : | 0.9937 |

Confusion Matrix:

[ [ 39 3

[ 2 70] ]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Classification | Report:  precision | recall | fl-score | support |
| 0 | 0.95 | 0.93 | B.94 | 42 |
| 1 | 0.96 | 0.97 | 0.97 | 72 |
| accuracy |  |  | 0.96 | 114 |
| macro avg | 0. 96 | 6. 93 | 0.95 | 114 |
| weighted avg | 6. 96 | 0. 96 | 0.96 | 114 |

worst area

worst concave points

mean concave points worst perimeter mean perimeter

mean radius mean cone avity

Top 10 Feature Importances — Random Forest

worst cone avity

000

6 0

importance

010 012

1.0

08

0.6

# 0.4

02

0.0

ROC Curve — Random Forest

Random Forest (AUC — 0.994)

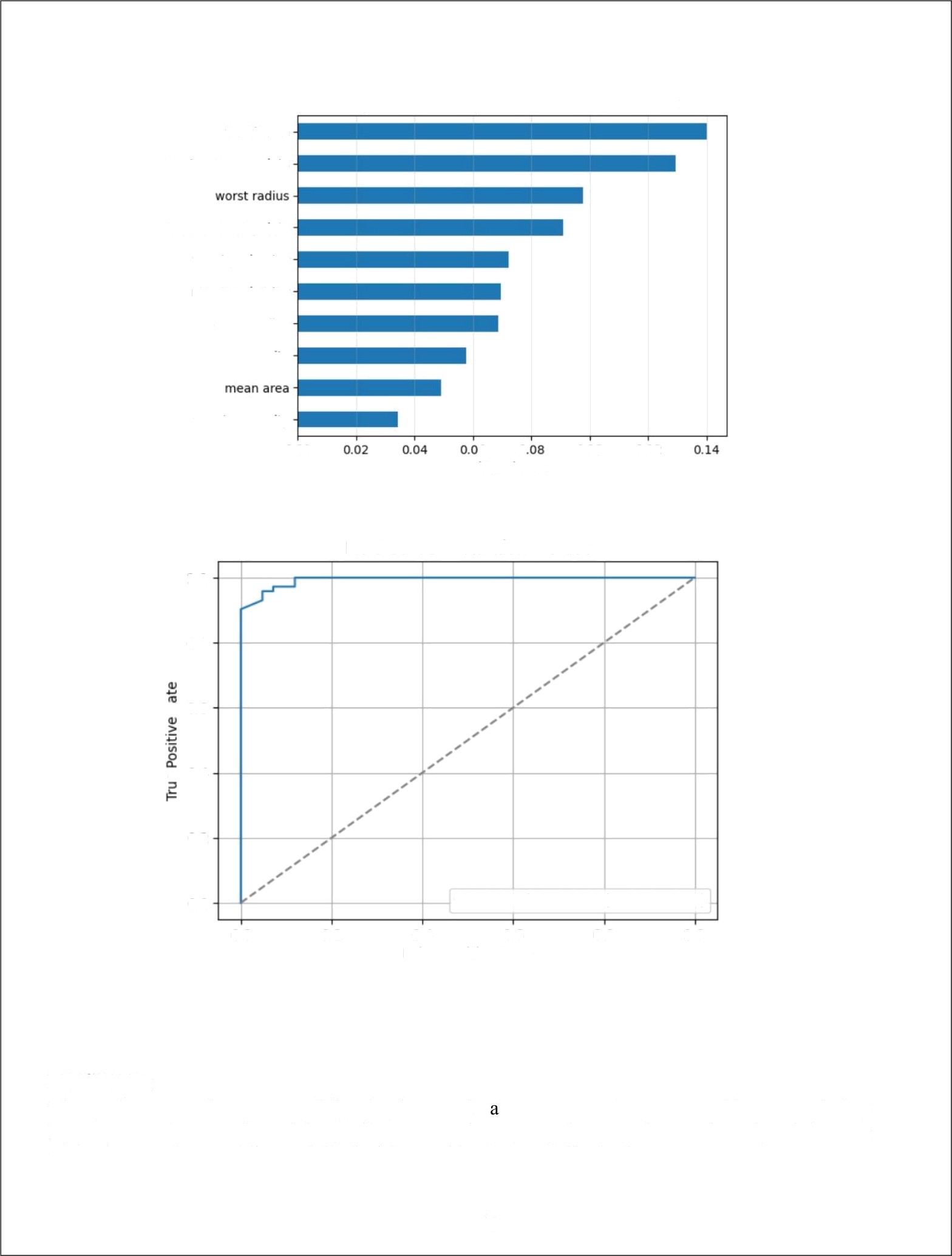
0.0

0.2

0.4 0.6

False Positive Rate

0.8 1.0



#### RESULT:

Thus, the execution successfully implemented Random Forest classifier, tuned key hyperparameters, evaluated its performance using appropriate metrics, and analyzed feature importance to interpret the model's decision-making process effectively.

19

|  |  |
| --- | --- |
| **EXP NO: 5** | **CLUSTERING WITH K-MEANS AND DIMENSIONALITY REDUCTION WITH PCA** |
| **DATE: 28/08/2025** |

AIM:

To demonstrate the application of Unsupervised Learning models, specifically K-Means clustering for grouping data points and Principal Component Analysis (PCA) for dimensionality reduction and visualization, using a suitable dataset.

ALGORITHM:

1. K-MEANS CLUSTERING

K-Means is an iterative clustering algorithm that aims to partition $n$ observations into $k$ clusters, where each observation belongs to the cluster with the nearest mean (centroid).

STEPS:

1. **INITIALIZATION:** Choose $k$ initial centroids randomly from the dataset.
2. **ASSIGNMENT:** Assign each data point to the cluster whose centroid is closest (e g., using Euclidean distance).
3. **UPDATE:** Recalculate the centroids as the mean of all data points assigned to that cluster.
4. **ITERATION:** Repeat steps 2 and 3 until the centroids no longer move significantly or a maximum number of iterations is reached.
5. PRINCIPAL COMPONENT ANALYSIS (PCA)

PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

STEPS:

1. **STANDARDIZATION:** Standardize the dataset (mean = 0, variance 1).
2. **COVARIANCE MATRIX CALCULATION:** Compute the covariance matrix of the standardized data.
3. **EIGENVALUE DECOMPOSITION:** Calculate the eigenvalues and eigenvectors of the covariance matrix.
4. **FEATURE VECTOR CREATION:** Sort the eigenvectors by decreasing eigenvalues and select the top $k$ eigenvectors to form a feature vector (projection matrix).
5. **PROJECTION:** Project the original data onto the new feature space using the feature vector.

CODE:

# EXPERIMENT — K-Means & PCA

# Import necessary libraries import numpy as np

import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.datasets import make blobs

from sklearn.preprocessing import StandardScaler from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from sklearn.metrics import silhouette score Part 1: K-Means Clustering ---

print(”--- Part 1: K-Means Clustering ---”

# 1. Generate dataset

|  |  |  |  |
| --- | --- | --- | --- |
| X, y = make blobs( =300, | =3, | ’ | =0.60, =42) |
| df kmeans = pd.DataFrame(X,  print(” Original K-Means Dataset print(df kmeans.head()) | =['Feature Head:”) | 1', | 'Feature 2']) |

# 2. Elbow Method

for i in range(1, 11): kmeans = KMeans(

=42)

kmeans.fit(X)

=i, -'k-means++',

=300, . -10,

wcss.append(kmeans.inertia )



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| plt.figure( | =(10, | 6)) |  | |
| plt.plot(range(l, | 11), | wcss, | ='o', | ='--' |

plt.title('Elbow Method for Optimal K (K-Means)') plt.xlabel('Number of Clusters (K)') plt.ylabel('WCSS')

plt.grid(True)

plt.show()

# 3. Apply K-Means with chosen K optimal k = 3

kmeans = KMeans( ‹=optimal k, ='k-means++',

.: =10, =42)

clusters = kmeans.fit predict(X) df kmeans['Cluster'] = clusters

=300,

# 4. Visualize K-Means clusters plt.figure( =(10, 8))

sns.scatterplot( -'Feature 1', ='Feature 2', .’-'Cluster', ,' -df kmeans,

='viridis', .-100, ,’ ,’=0.8)

plt.scatter(kmeans.cluster centers [:, 0], kmeans.cluster centers [:, l],

=300, ='red', ' ='X', n. ='Centroids') plt.title(f'K-Means Clustering with K={optimal k}') plt.xlabel('Feature 1')

plt.ylabel('Feature 2') plt.legend() plt.grid(True) plt.show()

# 5. Silhouette Score

silhouette avg = silhouette score(X, clusters)

print(f” Silhouette Score for K-Means (K={optimal k}): {silhouette avg:.3f}”) # --- Part 2: Dimensionality Reduction with PCA ---

print(” Part 2: Dimensionality Reduction with PCA ---”)

# 1. Generate 4D data set

X pca, y pca - make blobs(

=1. 0, , ’ =25)

;-500,

-4, =4,

df pca original = pd.DataFrame(X pca, range(X pca.shape[1])])

df pca original['True Cluster'] = y pca

print(” Original PCA Dataset Head:“) print(df pca original.head())

-[f'Feature {i+1}' for i in

print(f”Original PCA Dataset Shape: {df pca original.shape}”)

# 2. St a nd a cd ize

scaler = StandardScaler()

X pca scaled = scaler.fit transform(X pca)

# 3. PCA (4D a 2D)

pca = PCA( ’, ’ =2)

principal components = pca.fit transform(X pca scaled)



df principal components = pd.DataFrame(principal components,

:‹=['Principal Component 1', 'Principal Component 2']) df principal components['True Cluster'] = y pca

explained variance = pca.explained variance ratio print(” Principal Components Head:”)

print(df principal components.head())

print(f” Explained Variance Ratio: {explained variance}”)

print(f”Total Explained Variance by 2 PCs: {explained variance.sum():.3f}“)

# 4. Visualize PCA result plt.figure( ..’.’-(10, 8))

sns.scatterplot(’='Principal Component 1', -'Principal Component 2',

='True Cluster',

=df principal components, ;; -'Paired', =100,

’ ’=0.8)

plt.title('PCA - Dimensionality Reduction to 2 Components') plt.xlabel(f'PC1 ({explained variance[0]\*l00:.2f}%)') plt.ylabel(f'PC2 ({explained variance[l]\*l00:.2f}%)') plt.grid(True)

plt.show()

# 5. K-Means on PCA-reduced data

kmeans pca = KMeans( ‹=4, ='k-means++', =300,

clusters pca = kmeans pca.fit predict(principal components)

df principal components['KMeans Cluster on PCA'] = clusters pca

plt.figure( =(10, 8))

sns.scatterplot(’='Principal Component 1', ’ ='Principal Component 2',

='KMeans Cluster on PCA',

-df principal components, -'viridis', .-100,

, =0.8)

plt.scatter(kmeans pca.cluster centers [:, 0], kmeans pca.cluster centers [:, 1], =300, ='red', ’ ='X', -'Centroids')

plt.title('K-Means Clustering on PCA-Reduced Data') plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend() plt.grid(True) plt.show()

# 6. Silhouette Score for PCA-reduced KMeans

silhouette avg pca = silhouette score(principal components, clusters pca)

print(f“\nSilhouette Score for K-Means on PCA-Reduced Data (K=4):

{s1Ihouette avg pea: . 3f)")

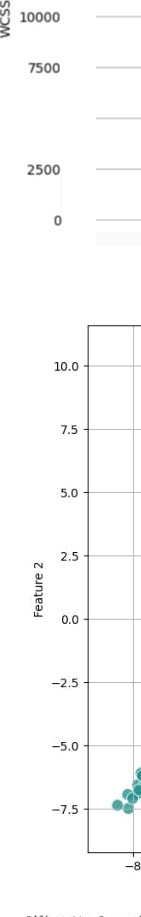
OUTPUT:

-- Part 1: K-Means Clustering ---

Original K-Means Dataset Head:

Feature 1 Feature 2

e -7.155244 -7.390016



1 -7.395875 -7. i ee4z

###### 2 -2.eiss71 8.28i7se

3 4.se927e 2.632436

4 -s.ie2se2 -7.484961

Elbon ›‹ethe for Optimal K (K-weans)



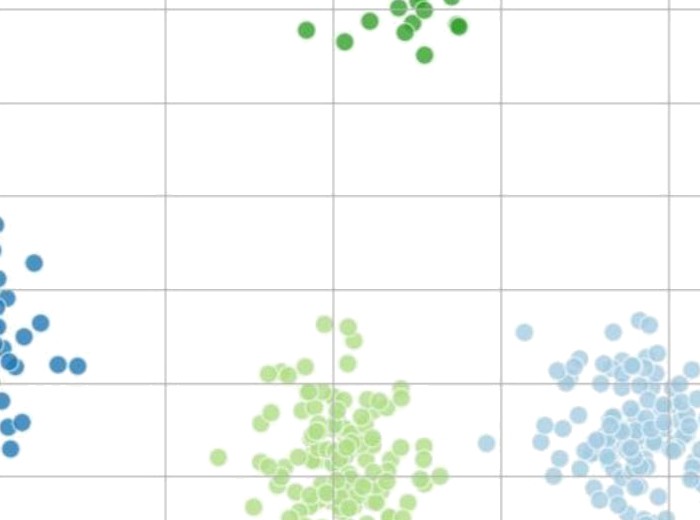
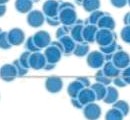
\*\*\* ’‘‘”‘ ‘””b ---- --e - --- - ------ ---- - - -•.... ...... .....

lO

Gleans Clustering with K=3

24

- Part 2 : Oine n s i on a I i ty gedu ct\*on ui t:h PCA



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0r | i g ina I PCA  F eat ure y | Oat a se t knead :  £eat-ure 2 Feat ure | | 3 | Feat | ure | 4 | T rue | ¢t | us1-er |
| 0 | -8. 63B667 | T . y y00S 7 | - 6. **4087** 22 | | -8.204990 | | | 3 | | |
| T | -2. 95T 5S6 | - 7. 65744S | 3. 844794 | | 8. 983S89 | | | 1 | | |
| 2 | -6.253J 77 | 2. MSte3 | -7. 86988 J | | B. 559678 | | | 3 | | |
| 3 | - 2. T51209 | 3. 4814W | - 5 .734930 | | 8. 965230 | | | 3 | | |
| 4 | —2. 3475 t9 | -7.238467 | 3. 47889 l | | - B. 443446 | | | I | | |

Or tgJ n at PHA 0a t aset Shape : ( SN. S ) Pr i n c i pa1 coeipon en t s klead :

Pr1s‹1pzI **Cosposen** I àinDpvl Coeponen 2 Inve CIurer B.4FF?eF I.62J?I? ?

2JW22 WUFW2 I

2 8I#2JA I.966926 ?

WA27 9 2dA926 ?

-L 4ÙB# B.£99?Se i

ExpTai ned var Tanc e Ratio: [8. S720 431 0. 336223a 2] kota I Exp) ai red var i anc e by 2 Pc s : 0. 908

PCA - Dimensionality Reduction to 2 **Components**

*y y* True Cluster

0

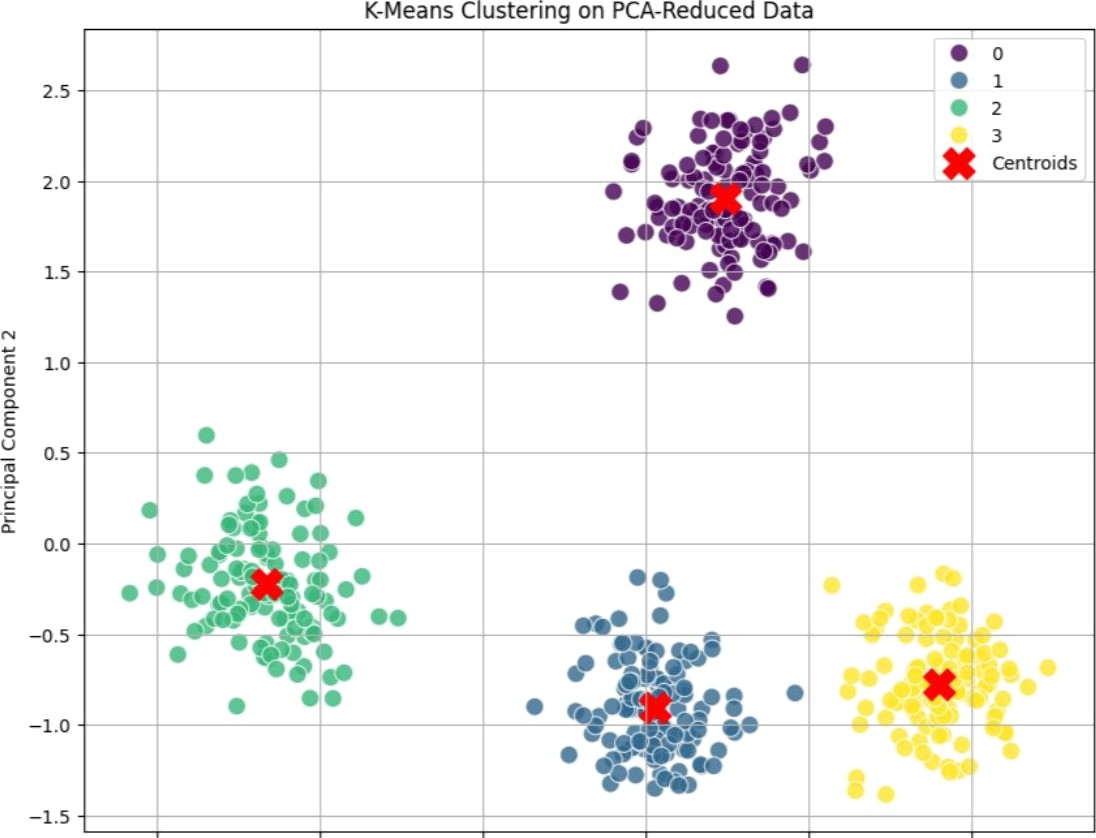
2

L5

-0.S

-2 -1 0 2

**PC1(SJ.Il’a)**





—3 -2 —T 0

Princ ipa IComponent T

SI T h a u e t t e 5ca n e Tor K - ' ea n s on PC s - Re d u ced Da t a ( K=4 › : 0 . T 7€'

## RESULT:

Thus, the execution successfully demonstrated Unsupervised Learning techniques by applying K- Means clustering to group data points and PCA for dimensionality reduction and visualization,

effectively revealing patterns and structure within the dataset.

|  |  |
| --- | --- |
| **EXP NO: 6** | **FEEDFORWARD AND CONVOLUTIONAL NEURAL**  **NETWORKS** |
| **DATE: 11/09/2025** |

AIM:

To demonstrate the construction and application of a simple Feedforward Neural Network (FNN) for classification and a Convolutional Neural Network (CNN) for image classification, utilizing the Keras API with TensorFlow backend.

ALGORITHM:

1. **FEEDFORWARD NEURAL NETWORK (FNN)**

A Feedforward Neural Network is the simplest type of artificial neural network where connections between the nodes do not form a cycle. It consists of an input layer, one or more hidden layers, and an output layer. Information flows only in one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes.

STEPS:

1. **Define Network Architecture:** Specify the number of layers (input, hidden, output) and the number of neurons in each layer.
2. **Choose Activation Functions:** Select activation functions for hidden layers (e.g., ReLU) and the output layer (e.g., Sigmoid for binary classification, Softmax for multi-class classification).
3. **Define Loss Function:** Choose a loss function appropriate for the task (e g., Binary Cross- entropy for binary classification, Categorical Cross-entropy for multi-class classification).
4. **Choose Optimizer:** Select an optimization algorithm (e g., Adam, SGD) to update network weights during training.
5. **Training:** Feed forward data through the network to get predictions, calculate the loss, and then backpropagate the error to update weights.
6. **Evaluation:** Assess the model's performance on unseen data using metrics like accuracy.
7. CONVOLUTIONAL NEURAL NETWORK (CNN)

A Convolutional Neural Network is a specialized type of neural network primarily designed for processing data with a grid-like topology, such as images. Key components include convolutional layers, pooling layers, and fully connected layers.

STEPS:

1. **Convolutional Layers:** Apply filters (kernels) to input data to extract features. Each filter detects a specific pattern (e.g., edges, textures).
2. **Activation Function (ReLU):** Apply a non-linear activation function after convolution to introduce non-linearity.
3. **Pooling Layers:** Downsample feature maps to reduce dimensionality, computational cost, and prevent overfitting (e g., Max Pooling).
4. **Flattening:** Convert the 2D pooled feature maps into a ID vector to be fed into a fully connected layer.
5. **Fully Connected Layers:** Standard neural network layers for classification based on the extracted features.
6. **Output Layer:** Final layer with an activation function (e g., Softmax) to output class probabilities.
7. **Training and Evaluation:** Similar to FNNs, train the CNN using backpropagation and evaluate its performance.

#### CODE:

# Import necessary libraries import numpy as np

import matplotlib.pyplot as plt import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.datasets import mnist, fashion mnist

from sklearn.metrics import classification report, confusion matrix import seaborn as sns

# Suppress TensorFlow warnings for cleaner output

tf.keras.utils.disable interactive logging()

# --- Part 1: Building a Simple Feedforward Neural Network --- print(”--- Part 1: Building a Simple Feedforward Neural Network ---”) # 1. Load and Preprocess Dataset (Using Fashion MNIST for FNN)

(x train fnn, y train fnn), (x test fnn, y test fnn) =

fashion mnist.load data()

print(f” Original FNN training data shape: {x train fnn.shape}“) print(f”Original FNN test data shape: {x test fnn.shape}”)

40

# Flatten images to lD array

x train fnn flat = x train fnn.reshape(-l, 28 \* 28)

x test fnn flat = x test fnn.reshape(-1, 28 \* 28) # Normalize pixel values

x train fnn norm = x train fnn flat / 255.0

1. test fnn norm = x test fnn flat / 255.0 print(f“Flattened & Normalized FNN training data shape:

{x train fnn norm.shape}“)

print(f“Flattened & Normalized FNN test data shape: {x test fnn norm.shape}“) # 2. Build FNN Model

model fnn = keras.Sequential([

layers.Dense(128, activation='relu', input shape=(784,)), layers.Dropout(0.2),

layers.Dense(64, activation='relu'), layers.Dense(10, activation='softmax')

# 3. Compile Model

model fnn.compile(optimizer='adam', loss='sparse categorical crossentropy', metrics=['accuracy'])

print(“\n--- FNN Model Summary ---“) model fnn.summary()

# 4. Train Model

print(“\n--- Training FNN Model ---“)

history fnn = model fnn.fit(x train fnn norm, y train fnn, epochs=10, validation split=0.1, verbose=1)

# 5. Evaluate Model

print(“\n--- Evaluating FNN Model ---")

loss fnn, accuracy fnn = model fnn.evaluate(x test fnn norm, y test fnn, verbose=0)

print(f“FNN Test Loss: {loss fnn:.4f}") print(f“FNN Test Accuracy: {accuracy fnn:.4f}") # Classification report & confusion matrix

1. pred fnn = np.argmax(model fnn.predict(x test fnn norm), axis=-1)

41

print(“\n--- FNN Classification Report ---“) print(classification report(y test fnn, y pred fnn)) print(“\n--- FNN Confusion Matrix ---“)

cm fnn = confusion matrix(y test fnn, y pred fnn) plt.figure(figsize=(10, 8))

sns.heatmap(cm fnn, annot=True, fmt=“d“, cmap=“Blues“, cbar=False) plt.title(“FNN Confusion Matrix“)

plt.xlabel(“Predicted Label“) plt.ylabel(“True Label“) plt.show()

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

plt.subplot(1, 2, 1)

plt.plot(history fnn.history['accuracy'], label='Training Accuracy') plt.plot(history fnn.history['val accuracy'], lnhel='Validation Accuracy') plt.title('FNN Model Accuracy')

plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.legend() plt.grid(True) plt.subplot(1, 2, 2)

plt.plot(history fnn.history['loss'], lahel='Training Loss') plt.plot(history fnn.history['val loss'], label='Validation Loss') plt.title('FNN Model Loss')

plt.xlabel('Epoch') plt.ylabel('Loss') plt.legend() plt.grid(True) plt.tight layout() plt.show()

# --- Part 2: Convolutional Neural Network (CNN) --- print(“\n--- Part 2: Implementing a CNN ---“)

# 1. Load MNIST for CNN

(x train cnn, y train cnn), (x test cnn, y test cnn) = mnist.load data() print(f“\nOriginal CNN training data shape: {x train cnn.shape}“)

42

print(f“Original CNN test data shape: {x test cnn.shape}“) # Reshape for channel dimension

x train cnn = x train cnn.reshape(x train cnn.shape[0], 28, 28, 1) x test cnn = x test cnn.reshape(x test cnn.shape[0], 28, 28, 1)

# Normalize

1. train cnn = x train cnn.astype('float32') / 255.0 x test cnn = x test cnn.astype('float32') / 255.0

print(f“Reshaped & Normalized CNN training data shape: {x train cnn.shape}“) print(f“Reshaped & Normalized CNN test data shape: {x test cnn.shape}")

num classes cnn = 10 # 2. Build CNN Model

model cnn = keras.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input shape=(28, 28, 1)),

layers.MaxPoo1ing2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)), layers.Flatten(),

layers.Dense(128, activztion='relu'), layers.Dropout(0.5),

layers.Dense(num classes cnn, activation='softmax')

# 3. Compile Model

model cnn.compile(optimizer='adam', loss='sparse categorical crossentropy', metrics=['accuracy'])

print(“\n--- CNN Model Summary ---“) model cnn.summary()

# 4. Train Model

print(“\n--- Training CNN Model ---“)

history cnn = model cnn.fit(x train cnn, y train cnn, epochs=10, validation split=0.1, verbose=1)

43

# 5. Evaluate Model

print(“\n--- Evaluating CNN Model ---“)

loss cnn, accuracy cnn = model cnn.evaluate(x test cnn, y test cnn, verbose=0) print(f“CNN Test Loss: {loss cnn:.4f}“)

print(f“CNN Test Accuracy: {accuracy cnn:.4f}“) # Classification report & confusion matrix

1. pred cnn = np.argmax(model cnn.predict(x test cnn), axis=-1)

print(“\n--- CNN Classification Report ---“) print(classification report(y test cnn, y pred cnn)) print(“\n--- CNN Confusion Matrix ---“)

cm cnn = confusion matrix(y test cnn, y pred cnn) plt.figure(figsize=(10, 8))

sns.heatmap(cm cnn, annot=True, fmt=“d“, cmap=“Blues“, cbar=False) plt.title(“CNN Confusion Matrix”)

plt.xlabel(“Predicted Label“)

plt.ylabel(“True Label”) plt.show()

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

plt.subplot(1, 2, 1)

plt.plot(history cnn.history['accuracy'], label='Training Accuracy') plt.plot(history cnn.history['val accuracy'], label='Validation Accuracy') plt.title('CNN Model Accuracy')

plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.legend() plt.grid(True) plt.subplot(1, 2, 2)

plt.plot(history cnn.history['loss'], label='Training Loss') plt.plot(history cnn.history['val loss'], label='Validation Loss') plt.title('CNN Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss') plt.legend()

plt.grid(True) plt.tight layout()

plt.show()

# Optional: Visualize predictions

print(” --- Sample CNN Predictions ---”) class names mnist = [str(i) for i in range(10)] plt.figure( =(10, 10))

for i in range(25): plt.subplot(5, 5, i + 1) plt.xticks([])

plt.yticks([]) plt.grid(False)

plt.imshow(x test cnn[i].reshape(28, 28), -plt.cm.binary) true label = y test cnn[i]

predicted label = y pred cnn[i]

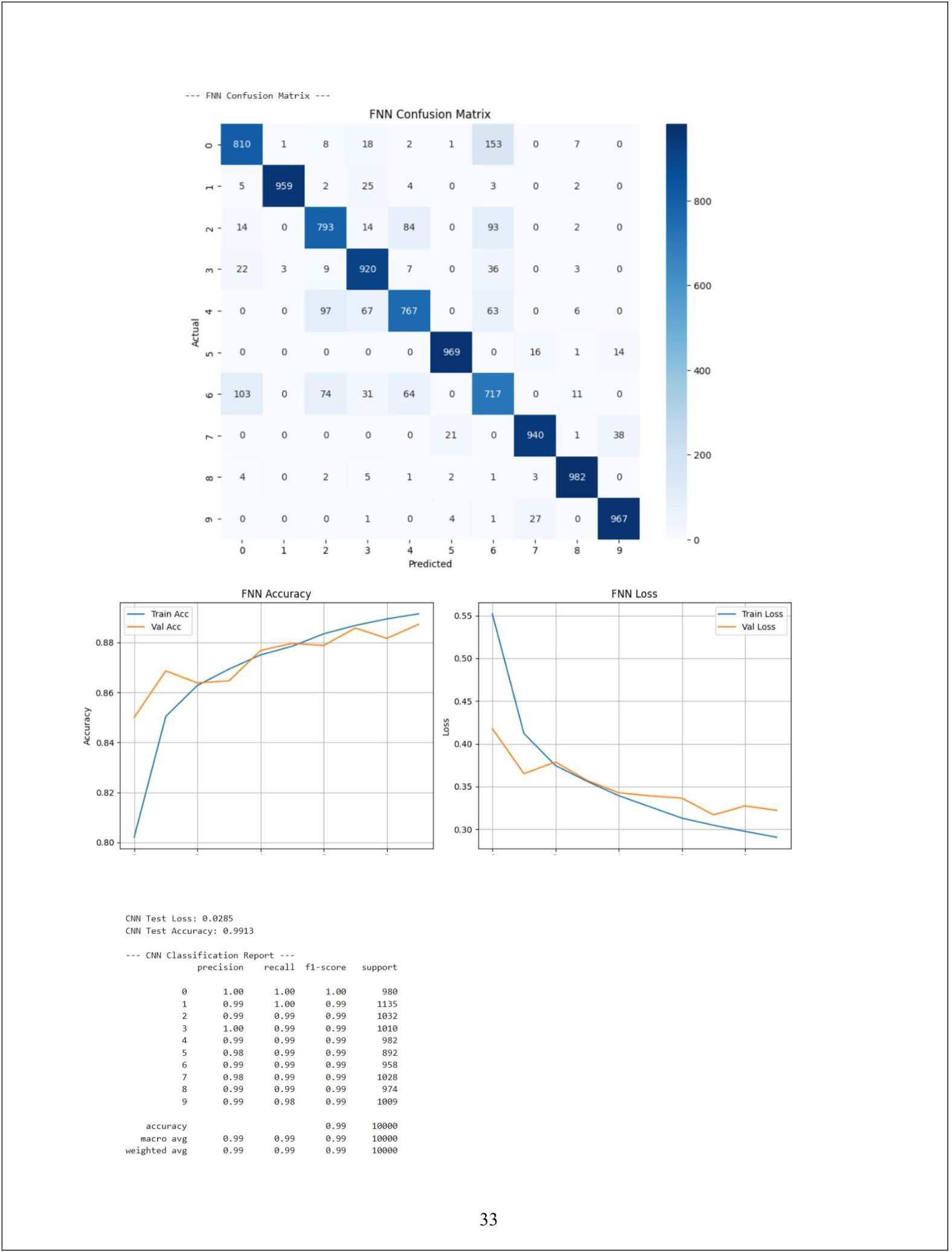
color = 'green' if true label == predicted label else 'red' plt.xlabel(f“True: {class names mnist[true label]} Pred:

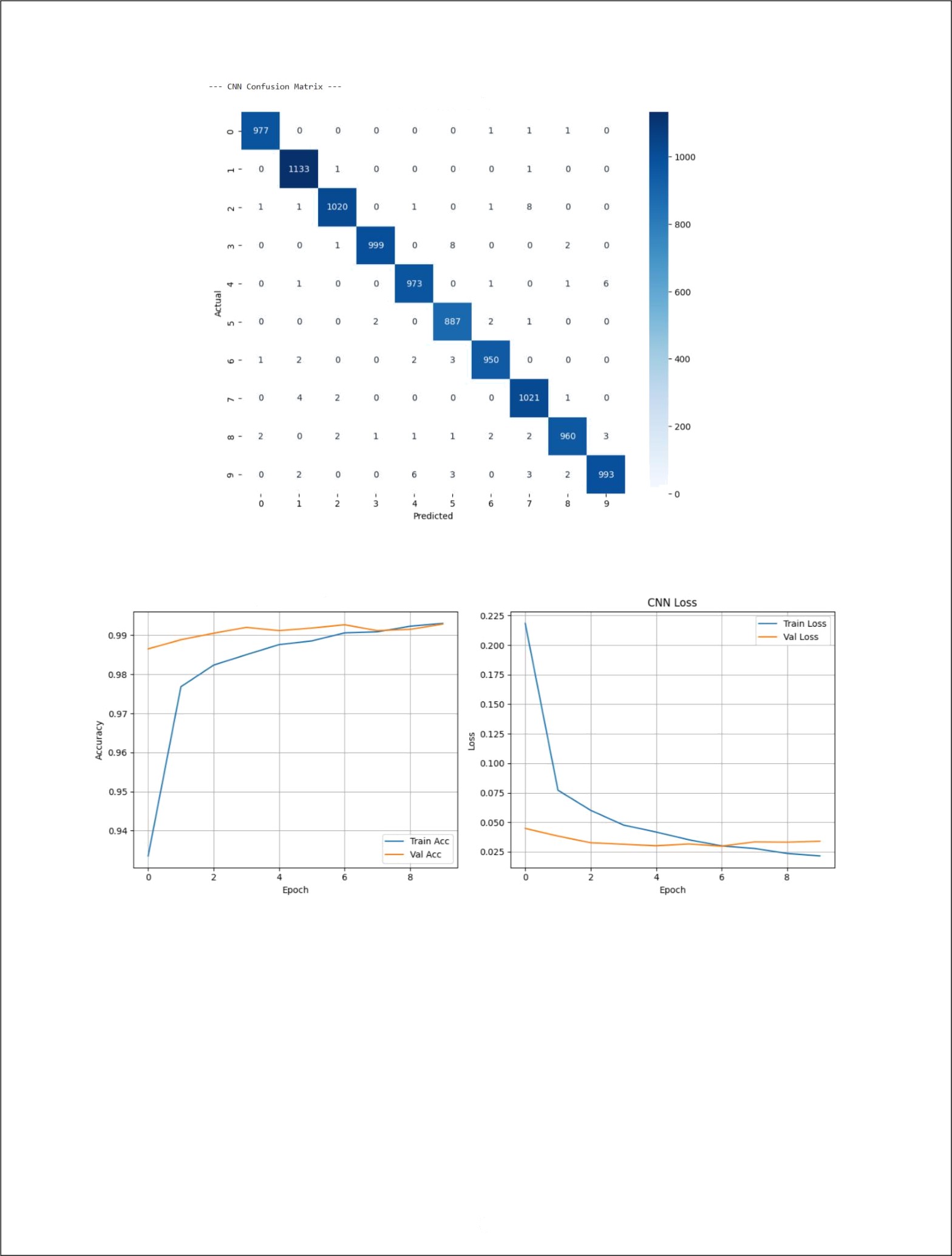
{class names mnist[predicted label]}“, =color) plt.suptitle(“Sample CNN Predictions (Green: Correct, Red: Incorrect)“,

=1.02, :=16)

plt.tight layout( =[0, 0, 1, 0.98]) plt.show()

|  |  |  |  |
| --- | --- | --- | --- |
| OUTPUT: |  |  | |
|  | FNN \*est Loss. 0.3404 |
|  | ANN Test Accuracy: 0.8824 |
|  | -- fNl Classification Report --- |
|  | precision recall | fl-score | support |
|  | 0 0.85 0 . 81 | 0.83 | IOOO |
|  | 1 1.00 0. 56 | 0.58 | 1000 |
|  | 2 0 . 81 0.79 | 0.88 | IOOO |
|  | 3 0 . 8S 0. ?2 | 8.88 | IOOO |
|  | 4 0.83 6. 77 | 0.80 | 1080 |
|  | 5 0.57 0. ^7 | 8.57 | IOOO |
|  | 6 0.67 6. 72 | 0.69 | 1000 |
|  | 7 0.95 6. ?4 | 0.55 | i0B0 |
|  | 8 0.97 0.98 | 0.57 | 1000 |
|  | 9 0.95 0.57 | 0.96 | 1000 |
|  | accuracy | 8. 88 | 18888 |
|  | macro avg 0.88 0. 88 | 8. 88 | 18888 |
|  | weighted avg 0 . 88 0. 88 | 8. 88 | 18888 |



CNN Confusion Matrix

34

CNN Predictions (Green = Correct, Red = Incorrect)

#### RESULT:

Thus, the execution successfully constructed and applied a Feedforward Neural Network (FNN) for classification and a Convolutional Neural Network (CNN) for image classification using the Keras API with TensorFlow backend, achieving efficient model training and evaluation.

|  |  |
| --- | --- |
| **EXP NO: 7** | **FEEDFORWARD AND CONVOLUTIONAL NEURAL**  **NETWORKS** |
| **DATE: 25/09/2025** |

#### AIM:

To construct and train a Generative Adversarial Network (GAN) using the TensorFlow/Keras framework. The objective is to train the GAN on the MNIST dataset to generate new, synthetic images of handwritten digits that are indistinguishable from the original training data.

### ALGORITHM:

GENERATIVE ADVERSARIAL NETWORKS (GANS)

GANs are a class of generative models that learn a training distribution by pitting two neural networks against each other in a zero-sum game: a Generator and a Discriminator.

1. **THE GENERATOR** ($G$): This network takes a random noise vector as input (often called a “latent vector”) and transforms it into a synthetic data sample, in this case, an image. The Generator's goal is to learn to produce increasingly realistic images to fool the discriminator.
2. **THE DISCRIMINATOR ($D$):** This is a binary classifier network. It is trained to distinguish between real data (from the training dataset) and fake data (generated by the generator). Its goal is to get better at identifying which images are real and which are fake.
3. THE ADVERSARIAL PROCESS:

**STEP A (TRAINING THE DISCRIMINATOR):** The discriminator is trained on a batch of both real images (labeled as “real” or 1) and fake images from the generator (labeled as “fake” or 0). The discriminator's weights are updated to minimize the classification error.

**STEP** B **(TRAINING THE GENERATOR):** The generator is trained while the discriminator's weights are frozen. The generator creates fake images and feeds them to the discriminator. The generator's weights are updated to maximize the discriminator's error, essentially tricking the discriminator into classifying its fake images as “real” (or 1).

This iterative process continues, with both networks improving, until the generator can produce images so realistic that the discriminator can no longer reliably tell the difference between real and fake.

## CODE:

# Import necessary libraries import numpy as np

import matplotlib.pyplot as plt import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.datasets import mnist import os

# Suppress TensorFlow warnings for cleaner output tf.keras.utils.disable interactive logging()

# --- Part 1: Dataset Loading and Preprocessing ---

print(”--- Part 1: Loading and Preprocessing the MNIST Dataset ---”) (x train, ), ( , ) = mnist.load data()

x train = x train.reshape(x train.shape[0], 28, 28, 1).astype('float32')

x train = (x train - 127.5) / 127.5 # Normalize to [-1, 1]

print(f”Normalized training data shape: {x train.shape}”) print(”Example of a normalized pixel value:”, x train[0, 0, 0, 0])

# --- Part 2: Building the Generator and Discriminator Models --- print(” --- Part 2: Building the GAN Components ---”)

latent dim = 100

# Generator

def build generator():

model = keras.Sequential( ,’ =”generator”) model.add(layers.Dense(7 7 256, -False,

=(latent dim,))) model.add(layers.BatchNormalization()) model.add(layers.LeakyReLU()) model.add(layers.Reshape((7, 7, 256)))

model.add(layers.Conv2DTranspose(128, (5, 5),

='same', =False)) model.add(layers.BatchNormalization()) model.add(layers.LeakyReLU())

=(1, 1),

model.add(layers.Conv2DTranspose(64, (5, 5), =(2, 2),

. .,='same', =False)) model.add(layers.BatchNormalization())

|  |  |  |
| --- | --- | --- |
| model.add(layers.LeakyReLU()) |  | |
| model.add(layers.Conv2DTranspose(1, (5, 5),  . .,='same', | = ( 2, | 2) , |
| =False, |  | ='tanh')) |
| return model |  |  |

generator = build generator()

print(” --- Generator Model Summary ---” generator.summary()

# Discriminator

def build discriminator():

model = keras.Sequential( :’=”discriminator”) model.add(layers.Conv2D(64, (5, 5), :-(2, 2), , .='same',

= [ 28, 28, 1] ) )

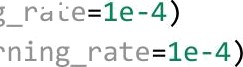
model.add(layers.LeakyReLU()) model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(128, (5, 5),

model.add(layers.LeakyReLU()) model.add(layers.Dropout(0.3)) model.add(layers.Flatten()) model.add(layers.Dense(1, return model

= ( 2, 2) , = ' same ' ) )

= sigmoid ))



discriminator = build discriminator()

print(” --- Discriminator Model Summary ---”) discriminator.summary()

# --- Part 3: Training Setup ---

cross entropy = keras.losses.BinaryCrossentropy( =False)

def discriminator loss( ,’ ,

real loss = cross entropy(tf.ones like( ), fake loss = cross entropy(tf.zeros like(!

return real loss + fake loss

def generator loss('’ ' '):

return cross entropy(tf.ones like( ),

generator optimizer tf.keras.optimizers.Adam( discriminator optimizer = tf.keras.optimizers.Adam(

@tf.function

def train step( ‹, =latent dim):

noise tf.random.normal([batch size, ])

with tf.GradientTape() as gen tape, tf.GradientTape() as disc tape: generated images = generator(noise, training=True)

real output = discriminator(images, training=True)

fake output = discriminator(generated images, training=True) gen loss = generator 1oss(fake output)

disc loss = discriminator 1oss(rea1 output, fake output)

gradients of generator = gen tape.gradient(gen loss, generator.trainab1e variables)

gradients of discriminator = disc tape.gradient(disc loss, discriminator.trainable variables)

generator optimizer.apply gradients(zip(gradients of generator,

generator.trainab1e variables))

discriminator optimizer.apply gradients(zip(gradients of discriminator, discriminator.trainable variables))

return gen loss, disc loss

def generate and save images(model, er°ch, test input): predictions model(test i put, training=Fa1se)

predictions rescaled = (predictions \* 0.5) 0.5 # Scale back to [0, 1] fig = plt.figure(fib•ize=(4, 4))

for i in range(predictions.shape[0]):

plt.subplot(4, 4, i + 1)

plt.imshow(predictions rescaled[i, :, :, 0], cmap='gray') plt.axis('off')

plt.suptitle(f"Epoch (epoch}“, fontnize=16) if not os.path.exists('images'):

os.makedirs('images')

plt.savefig(f'images/image at epoch {epech:04d}.png') plt.show()

# Training parameters EPOCHS = 200

batch size = 256

num examples to generate = 16

seed = tf.random.normal([num examples to generate, latent dim])

train dataset =

tf.data.Dataset.from tensor slices(x train).shuffle(x train.shape[0]).batch(ba tch size)

# Training loop

def train(dataset, epochs):

print(“\n--- Beginning GAN Training ---“) for epoch in range(epochs):

gen loss list = []

disc loss list = [] for image batch in



gen loss, disc loss = train step(image batch)

gen loss list.append(gen loss.numpy()) disc loss list.append(disc loss.numpy())

avg gen loss = np.mean(gen loss list)

avg disc loss = np.mean(disc loss list) print(f“Epoch {epoch + l}/{. } - Generator Loss:

{avg gen loss:.4f}, Discriminator Loss: {avg disc loss:.4f}”) if (epoch + 1) % 20 == 0:

generate and save images(generator, epoch + 1, seed)

print(” --- Training complete. Generating final images. ---” generate and save images(generator, ›, seed)

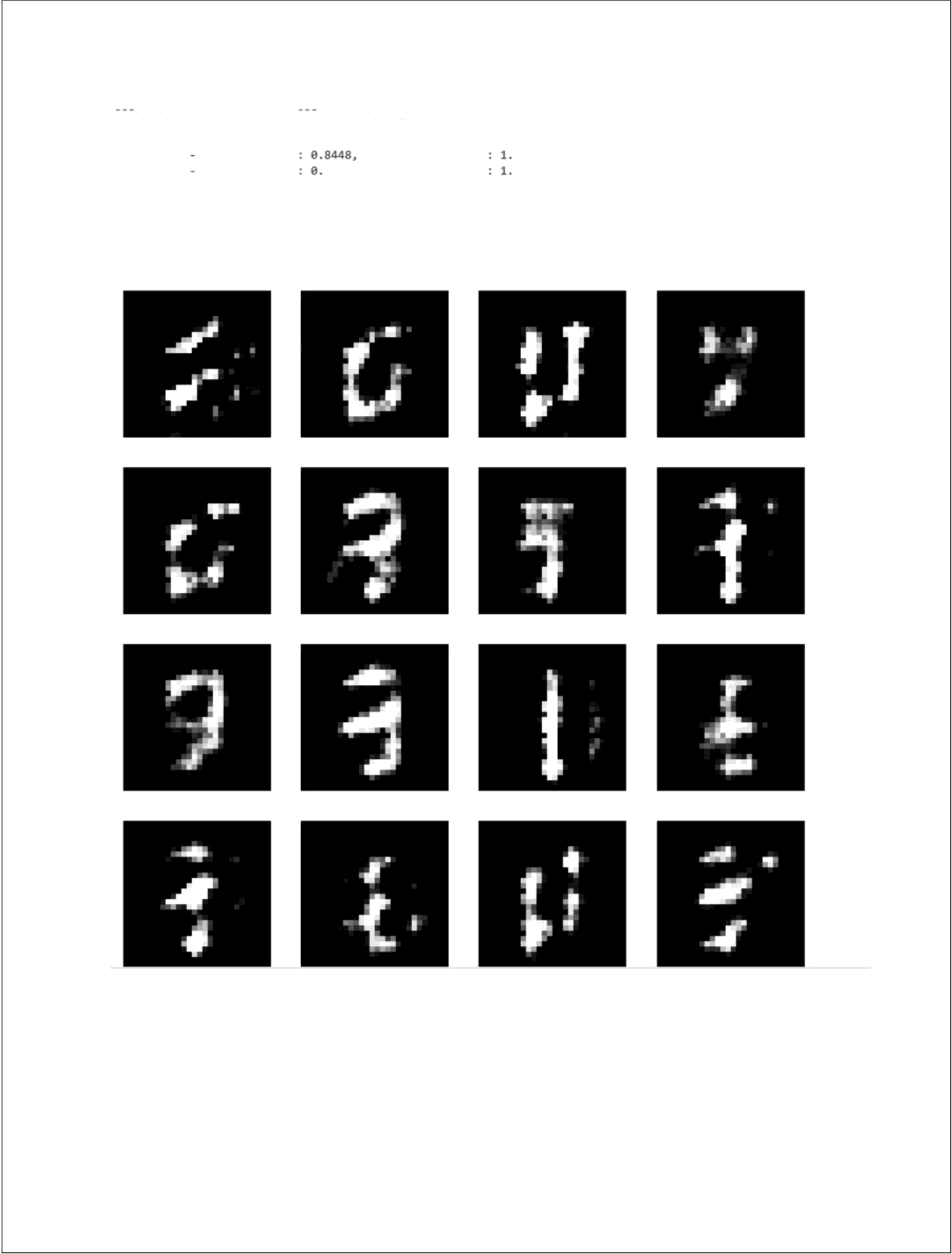
# Run training

train(train dataset, EPOCHS)

#### OUTPUT:

--- Part 1: Loading and Preprocessing the MNIST Dataset --- Normalized tra ning data shape: (60000, 28, 28, 1)

Example normalized pixel value: -1.B

Beginnsng GAN Tr aini ng

Epoch 1/ 38 - Generator Loss : 0.7877, Epoch 7/ 28 - Generator Los s : 6 . 81A8, Epoch 3/ 2e Generator Los s

Epoch 4/20 Generator Los s 8 5 3d, Epoch S/ 30 - Generator Los s : 8.8373,

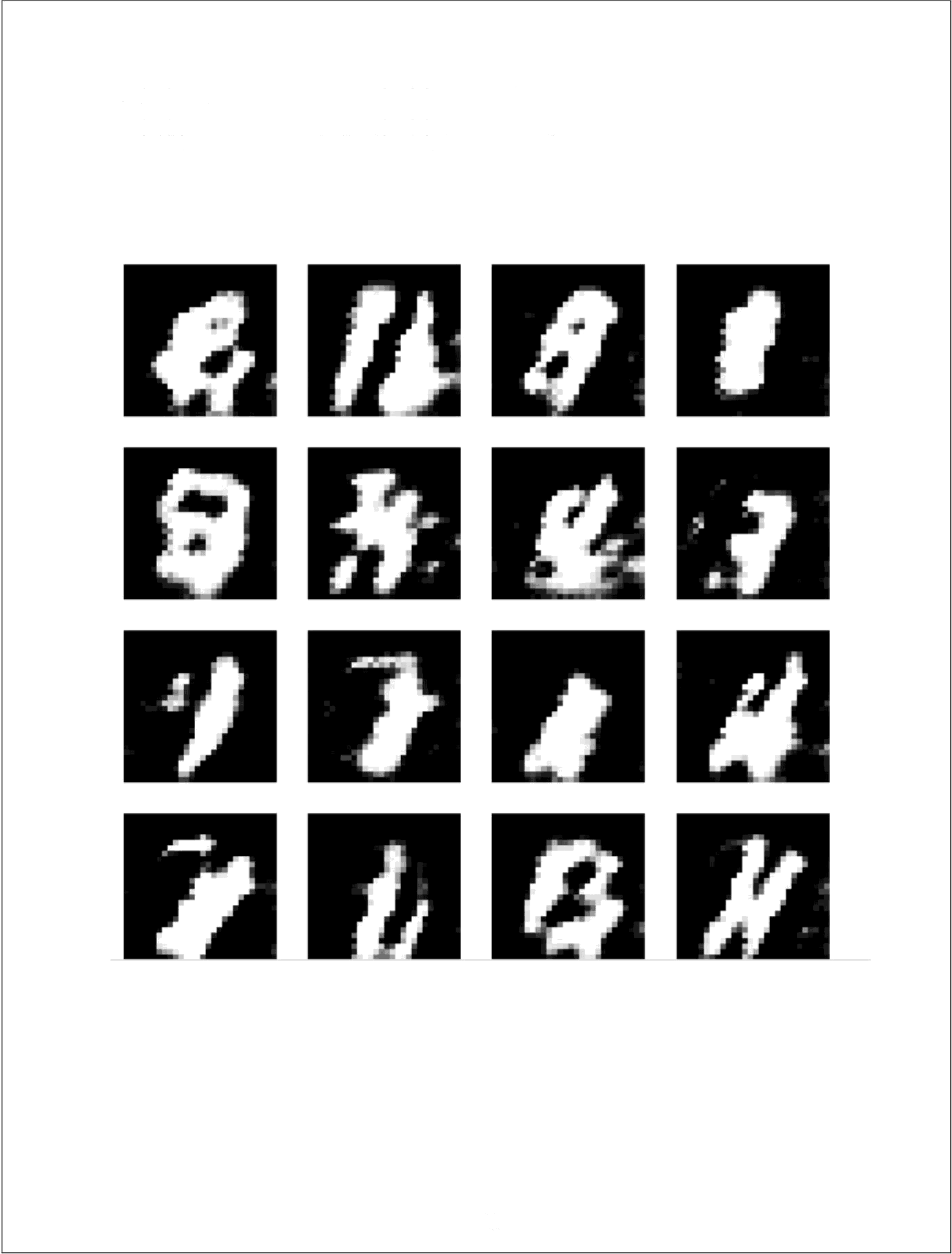
Di sc riminat.or Los s: 1. 0328 Di sc r iminator Loss: 1. 2225 Di sc r1minator• Loss 3634 Dt scrlmtnator Loss 2366

Di sc r iminat or Loss: 1.2497

## Epoch 5

41

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| E poc h | 6/2e | - Generator | Los s : | e. 8S›6. | D i s cr i ml nator | ras s : | › . z7eS |
| E poc h | 7/ 38 | - Generator | Los s: | 0.8888, | Di scr imi nator | joss: | 1. 3828 |
| E poc h | ă/ 20 | - Generator | Loss : | 0 . g739. | Di s rc i mJ nator | Los s : | 3. 2S32 |
| E poc h | 9/ 20 | - Generator | Loss : | 0.869a. | Di s cr i ml nator | Loss • | 3 . 3t30 |
| E poc h | 18/36 | - Generator | Lo s s: | 8. 8862, | Di sc r î ai nat or | Los s - | 1 . 2328 |

Epoch lO

42

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epoc h | 11/ 2e | - Gener atOP | LOVI : | 6. 9361. | Disc r ieiinator Los s : | 1.2244 |
| Epoc h | 12/ 20 | - Generator | Loss : | 6. 99a6. | Disc r iainat or Loss : | 1 . y7 y9 |
| Epoch  Epoc h | t3/ 30  14/ 26 | Generator  - Generator | Loss  ross : | B. 9786. | DSsc r leiinat or Loss  Di sc ri z\*nat or Loss : | 1. 1809 |
| Epoc h | 1S/ 26 | - Generator | Los s: | 1. 8a28. | Di sc r 1ninat or Loss: | 1 . 1B79 |
|  |  |  |  |  | Epoch 15 |  |

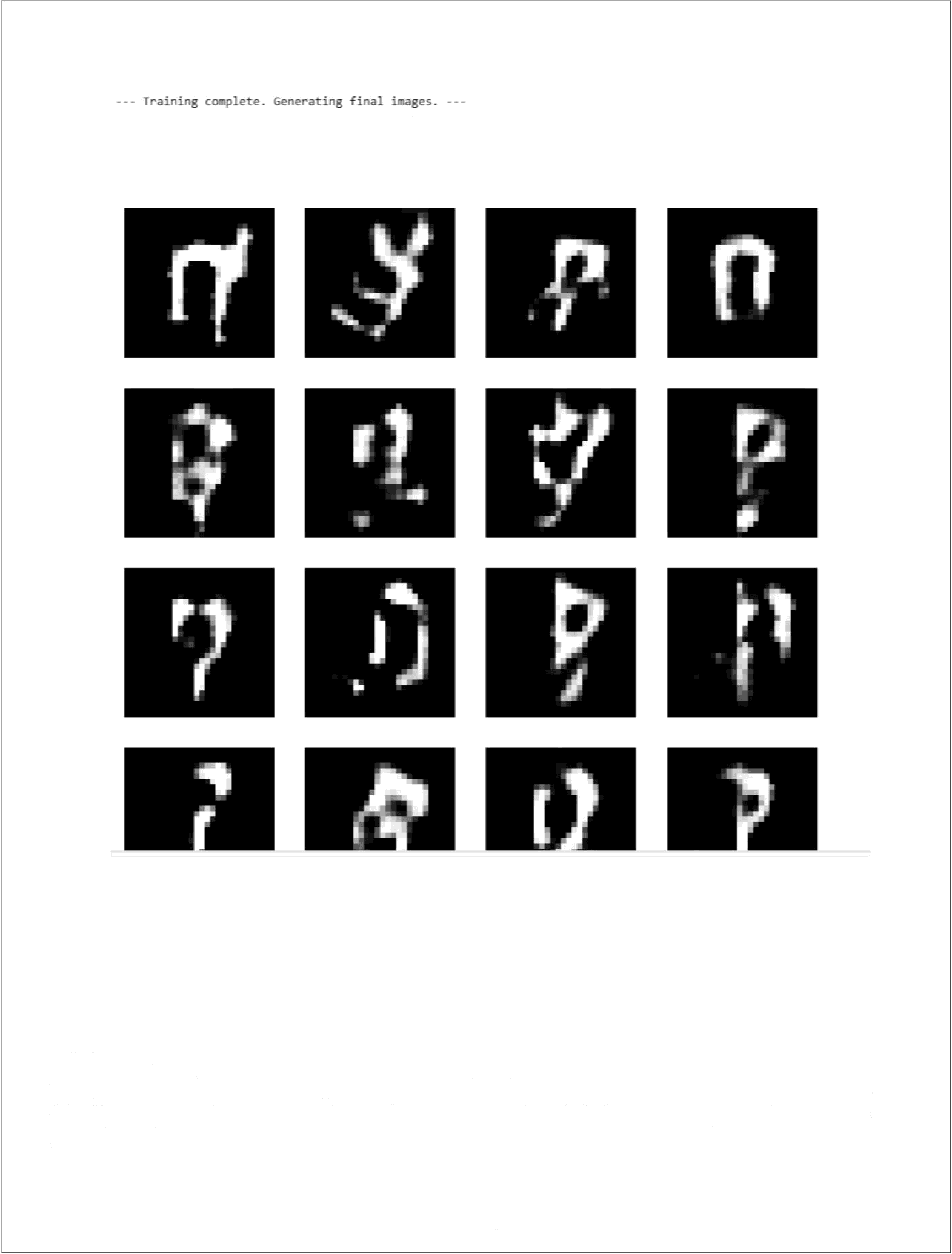


43

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Epoch | 16/28 | - Generator | Loss : | :t . 2826. | Disc rlulnator | Loss: | 1. B483 |
| Epoch | :t7/20 | - Generator | Loss : | 1 . 2648. | D:tsc r Emlnator | Loss • | 1. 06eS |
| Epoch | TB/ 26 | - Generator | Loss: | 1. T6S7, | D1sc rlnlnator | Loss: | y . 8a8a |
| Epoch | 19/26 | - Generator | Loss : | 1 . y644, | DIsc rJ•1nator | Loss: | y . 6897 |
| Epocn | 2e/le | - cenerator | Loss: | t. 3776. | Disc rlminator | Loss: | 3 . 0938 |

# Epoch 20

44

Epoch 20

#### RESULT:

Thus, the execution successfully constructed and trained a Generative Adversarial Network

(GAN) using the TensorFlow/Keras framework on the MNIST dataset, generating realistic synthetic handwritten digit images similar to the original data.

45

|  |  |
| --- | --- |
| **EXP NO: 8** | **MODEL EVALUATION AND IMPROVEMENT: HYPERPARAMETER** TUNING **WITH GRID SEARCH AND**  CROSS-VALIDATION |
| **DATE: 09/10/2025** |

#### AIM:

To demonstrate key techniques for model evaluation and improvement:

* **HYPERPARAMETER TUNING WITH GRID SEARCH** : Systematically searching for the optimal combination of hyperparameters for a machine learning model.
* **CROSS-VALIDATION TECHNIQUES:** Implementing k-fold cross-validation to get a more robust estimate of model performance and to prevent overfitting to a specific train-test split.

### ALGORITHM:

1. HYPERPARAMETER TUNING WITH GRID SEARCH

Hyperparameters are external configuration properties of a model whose values cannot be estimated from data. Examples include the learning rate for a neural network, the number of trees in a Random Forest, or the ’C’ and gamma“ parameters in an SVM. Tuning these parameters is crucial for optimal model performance.

**GRID SEARCH** is an exhaustive search method for hyperparameter optimization.

STEPS:

1. Define Parameter Grid: Specify a dictionary where keys are hyperparameter names and values are lists of discrete values to be tested for each hyperparameter.
2. **Instantiate Model:** Choose a machine learning model.
3. **Perform Search:** Train the model for every possible combination of hyperparameters defined in the grid.
4. Evaluate: For each combination, evaluate the model&#39;s performance using a specified scoring metric (e.g., accuracy, F1-score) and often in conjunction with cross-validation.
5. Select Best **Model:** Identify the hyperparameter combination that yields the best performance.
6. CROSS-VALIDATION TECHNIQUES

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The goal is to estimate how accurately a predictive model will perform in practice. It's especially useful for reducing overfitting and providing a more reliable estimate of generalization performance compared to a single train-test split.

K-FOLD CROSS-VALIDATION:

STEPS:

1. **Divide Data:** The entire dataset is randomly partitioned into $k$ equally sized subsamples (or “folds”).
2. **Iterate ‘k’ Times:** In each iteration, one fold is used as the validation (or test) set, and the remaining $k-1$ folds are used as the training set The model is trained on the training set and evaluated on the validation set.
3. **Aggregate Results:** The performance metric (e.g., accuracy) from each of the $k$ iterations is collected.
4. **Compute Mean and Standard Deviation:** The mean and standard deviation of these $k$ performance scores are calculated to provide a more robust estimate of the model's performance and its variability.

## CODE:

# Import necessary libraries import numpy as np

import pandas as pd

|  |  |  |  |
| --- | --- | --- | --- |
| import | matplotlib.pyplot | as | plt |
| import | seaborn as sns |  |  |

from sklearn.datasets import load iris # A classic dataset for classification from sklearn.model selection import train test split, KFold, cross val score,

GridSearchCV

from sklearn.svm import SVC # Support Vector Classifier, a common model for tuning

from sklearn.metrics import accuracy score, classification report, confusion matrix

from sklearn.preprocessing import StandardScaler

# --- Part 1: Hyperparameter Tuning with Grid Search --- print(”--- Part 1: Hyperparameter Tuning with Grid Search ---”) # 1. Load a Dataset (Iris Dataset for classification)

# The Iris dataset is a classic and simple dataset for classification tasks.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # It contains | measurements | of iris flowers | (sepal | length, sepal | width, | petal |
| length, petal | width) |  |  |  |  |  |

# and their corresponding species (Setosa, Versicolor, Virginica). iris = load iris()

X = iris.data

y = iris.target

feature names = iris.feature names target names = iris.target names

print(f“\nDataset Features (X) shape: {X.shape}“) print(f“Dataset Labels (y) shape: {y.shape}“) print(f“Feature Names: {feature names}“) print(f“Target Names: {target names}“)

# 2. Split Data into Training and Testing Sets

# It's crucial to split the data before scaling to prevent data leakage. # The test set will be used for final model evaluation, after tuning.

X train, X test, y train, y test = train test split(X, y, test size=0.3, rar^om state=42, vLratify=y)

print(f“\nTraining set size: (X train.shape[0]} samples“) print(f“Test set size: {X test.shape[0]} samples")

# 3. Standardize Features

# Scaling features is important for SVMs as they are sensitive to feature scales.

# Fit scaler only on training data to prevent data leakage.

scaler = StandardScaler()

X train scaled = scaler.fit transform(X train)

X test scaled = scaler.transform(X test) print(“\nFeatures standardized.“)

# 4. Define the Model and Hyperparameter Grid

# We'll use a Support Vector Classifier (SVC) as our model.

# Common hyperparameters for SVC are 'C' (regularization parameter) and 'gamma' (kernel coefficient).

# 'kernel' also can be tuned (e.g., 'linear', 'rbf').

# Define the parameter grid for Grid Search param grid = {

'C': [0.1, 1, 10, 100], # Regularization parameter

'gamma': [1, 0.1, 0.01, 0.001], # Kernel coefficient for 'rbf', 'poly' and 'sigmoid'

'kernel': ['rbf', 'linear'] # Type of kernel function

print(“\nHyperparameter grid defined:“) for param, values in param grid.items():

print(f“ {param}: {values}“)

# 5. Perform Grid Search with Cross-Validation

# GridSearchCV automatically performs k-fold cross-validation for each combination.

# cv=5 means 5-fold cross-validation.

# scoring='accuracy' means we want to optimize for accuracy.

grid search = GridSearchCV(SVC(), param grid, cv=5, scoring='accuracy', verbose=1, r 'obs=-1)

print(“\nStarting Grid Search with 5-fold Cross-Validation...") # Fit GridSearchCV on the scaled training data

grid search.fit(X train scaled, y train)

print(“\nGrid Search completed.“)

# 6. Get the Best Parameters and Best Score

print(f“\nBest hyperparameters found: {grid search.best params }“) print(f“Best cross-validation accuracy: {grid search.best score :.4f}“)

# 7. Evaluate the Best Model on the Test Set

# The best estimator attribute provides the model trained with the best parameters.

best model = grid search.best estimator

y pred tuned = best model.predict(X test scaled)

test accuracy tuned = accuracy score(y test, y pred tuned) print(f“\nTest set accuracy with tuned model: {test accuracy tuned:.4f}")

print(“\n--- Classification Report for Tuned Model ---")

print(classification report(y test, y pred tuned, target les=target names))

print(“\n--- Confusion Matrix for Tuned Model ---“) cm tuned = confusion matrix(y test, y pred tuned) plt.figure(fig 'ze=(8, 6))

sns.heatmap(cm tuned, r--ot=True, fmt='d', cmap='Blues', xtic)'aL ls=target names, yticklzbels=target names) plt.title('Confusion Matrix (Tuned SVM)') plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# Visualize Grid Search results (optional, but good for understanding) # Convert results to a DataFrame for easier analysis

results df = pd.DataFrame(grid search.cv results ) print(“\n--- Top S Grid Search Results ---“)

print(results df[['param C', 'param gamma', 'param kernel', 'mean test score', 'rank test score']].sort values(by='rank test score').head())

# --- Part 2: Cross-Validation Techniques (k-fold) --- print(“\n--- Part 2: Cross-Validation Techniques (k-fold) ---“)

# We will demonstrate k-fold cross-validation on a simple SVM without explicit tuning for clarity,

# to focus solely on the CV process.

# 1. Instantiate a Model (using default or chosen parameters)

model cv = SVC(random state=42) # Using default parameters for simplicity

# 2. Define k-fold Cross-Validation Strategy # We'll use 5-fold cross-validation.

# KFold ensures that each fold is distinct.

# shuffle=True means the data will be randomly shuffled before splitting into folds.

# random state for reproducibility.

k folds = 5

kf = KFold(r splits=k folds, sh. ffle=True, random state=42)

print(f“\nPerforming {k folds}-fold cross-validation...“) # 3. Perform Cross-Validation and Get Scores

# cross va1 score performs the KFold splitting, training, and evaluation automatically.

# It returns an array of scores, one for each fold.

cv scores = cross val score(model cv, X train scaled, y train, cv=kf, scoring='accuracy')

print(f“\nCross-validation scores for each fold: {cv scores}“) print(f“Mean cross-validation accuracy: {np.mean(cv scores):.4f}“) print(f“Standard deviation of cross-validation accuracy:

{np.std(cv scores):.4f}“)

# 4. Visualize Cross-Validation Scores plt.figure(figsize=(8, 5))

plt.bar(range(1, k folds + 1), cv scores, ='skyblue') plt.axhline( =np.mean(cv scores), ='r', ='--',

Accuracy ({np.mean(cv scores):.4f})')

plt.title(f'{k folds}-Fold Cross-Validation Accuracy Scores') plt.xlabel('Fold Number')

plt.ylabel('Accuracy')

plt.ylim(0.8, 1.0) # Set y-axis limits for better visualization plt.legend()

=f'Mean

plt.grid( .. ='y', plt.show()

='--')

# 5. Discuss why CV is useful

print(” --- Why is Cross-Validation Important? ---”)

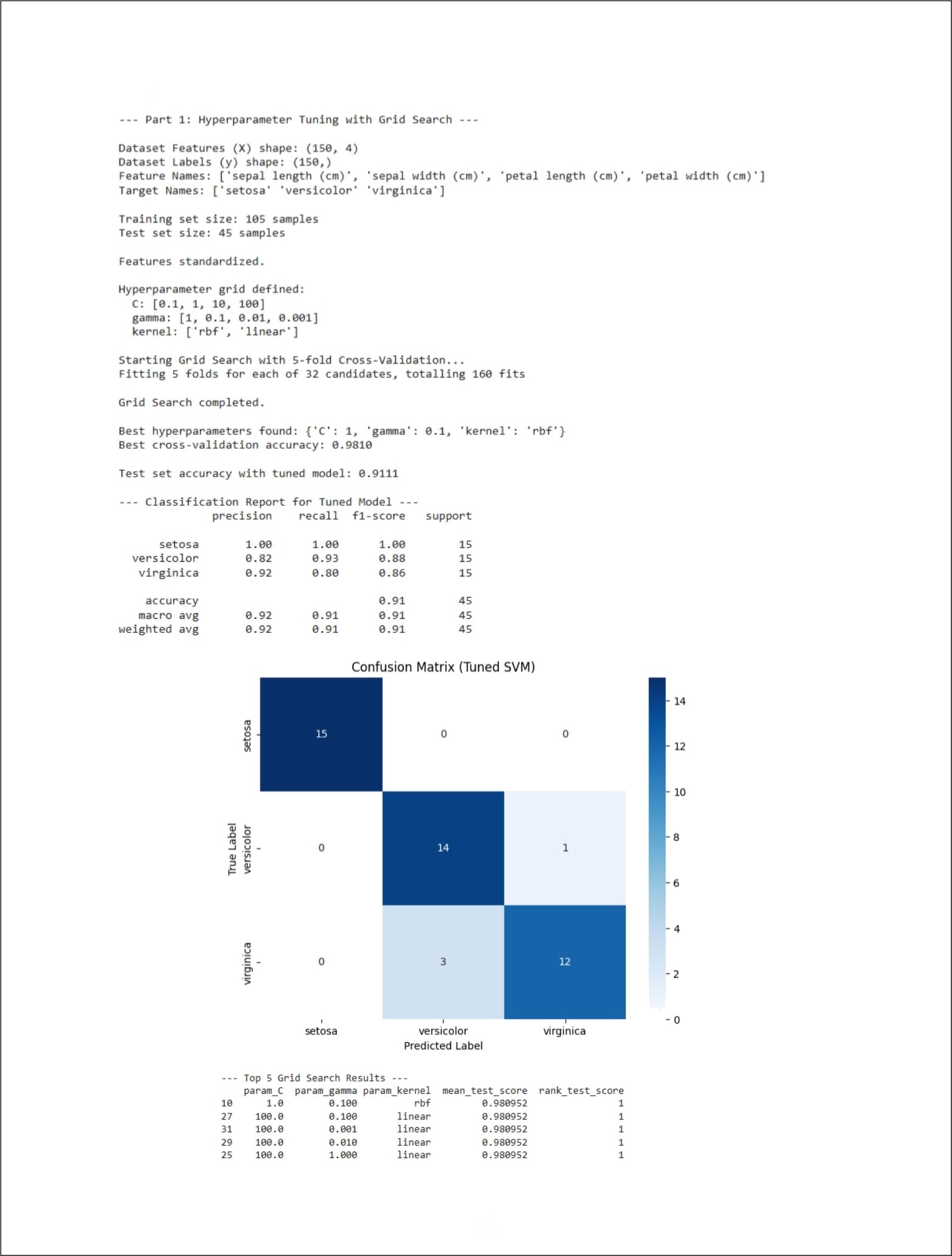
print(”1. More Reliable Performance Estimate: Reduces bias from a single train-test split.”)

print(”2. Better Generalization: Helps ensure the model performs well on unseen data.”)

print(”3. Efficient Data Usage: All data points are used for both training and

validation across different folds.”)

print(”4. Detects Overfitting/Underfitting: Variability in scores can indicate instability.”)

OUTPUT:

52

Pa ct 2 : c r'os s -Va 11d a t ion Tec h n res ( k (o ld ) P e mo r ml ng S- (of d c co ss - va I i d as io n . ..

C no s s - va J i d at i on s co ne s +o r e ach +o Td : [ 1 . '-Te an c nos s - va T i d at i on ac c ura c y : 0 . 97t'4

0 . ". s23809 T 0. ".52380."S 0 . 9S 2 38095 1.

S r and a nd de v i a t i on oT c r as s - a T i d at i on ac c u na cy \* 0 . 023 3

5-Fold Cross-Validation Accuracy Scores

1. 000

— - - Mea n Ac cu rack 10.9 7 lA)

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

0. 97 5

0. 96 0

**0.925**

0. 900

**0875**

**0850**

**0.825**

**0800**

1

2

Fold Number



- \'Jfry i s Cr as s -va 1id a t hon I mpo r r an t"

1. . ' lo ne Re Tla bye Pe n+or mance E st Tm ate : Red uc e s bi a s f-roo a sT ng Te t n a Tn - te s t sp Tit .
2. . Bet t er Gen er‘a T T: at ion : He 1ps ens ure t he mode) per I- oros vze) T on unse en d a ta .

3. E -fN•i cient Dat a U s age : All dat a pod rt s ai e us ed -fo r bat h tra Tn Tng and \ a1i da t io n ac r as s d i I-I er ent God d s .

4 . Oet ect s 0v er I Tt t ink/Undenf itt i ng : va r i ab i I ity i n s core s can Tndi cat e i ns t abT TQty .

#### RESULT:

Thus, the execution successfully demonstrated model evaluation and improvement techniques by performing Hyperparameter Tuning with Grid Search to find the optimal parameters and applying Cross-Validation to obtain a robust estimate of model performance while preventing overfitting.

53