EXPERIMENT 6(A)

AIM: Support Vector Machines (SVM): Implement SVM on a dataset and evaluate the model's performance.

THEORY:

- SVM is a supervised machine learning algorithm used for both classification and regression.
- It tries to find an optimal hyperplane that separates data points of different classes with the maximum margin.
- For non-linearly separable data, SVM uses the kernel trick to map the data into higher dimensions where a separating hyperplane exists**.**

Key equations:

1. Decision Function:
   F(x) = w.x + b
   Where w is the weight vector and b is the bias.
2. Optimization Objective:

$$min\frac{1}{2}\left|\left|w\right|\right|^2$$

3. Kernel Function:

$$K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$$

ABOUT THE DATASET:

- A small image dataset available in sklearn.datasets.
- Consists of 1,797 grayscale images of handwritten digits (0–9).
- Each image is 8×8 pixels (64 features after flattening).
- Target labels correspond to the digit value.

SOURCE CODE:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
```

```
digits = load_digits()
X, y = digits.data, digits.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42, stratify = y)
```

```
svm_clf = SVC(kernel = 'rbf', C = 10, gamma = 0.0001)
svm_clf.fit(X_train,y_train)
```

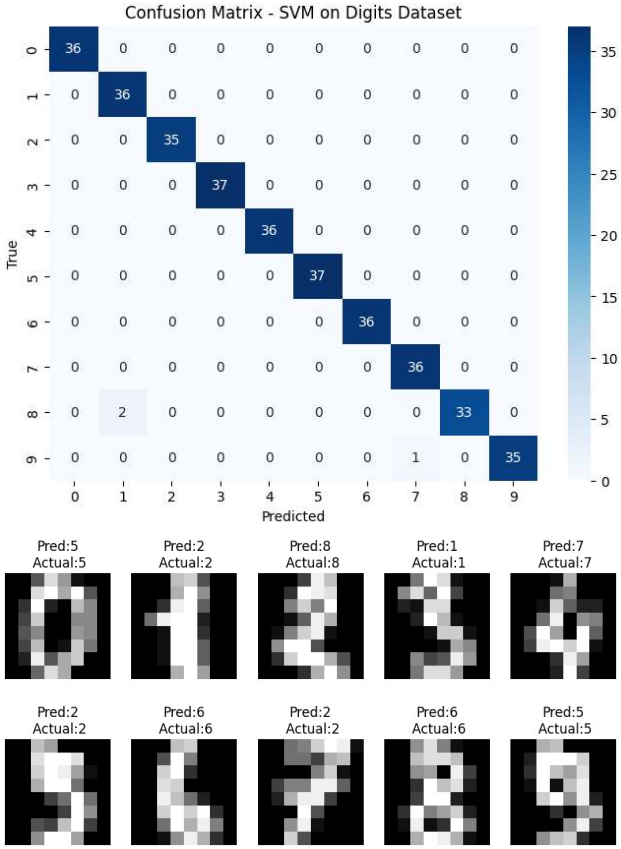
```
y_pred = svm_clf.predict(X_test)

# Accuracy
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.4f}")

# Classification Report
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Heatmap
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=digits.target_names,
            yticklabels=digits.target_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix - SVM on Digits Dataset")
plt.show()
```

```
fig, axes= plt.subplots(2,5,figsize=(10,5))
for i,ax in enumerate(axes.flat):
      ax.imshow(digits.images[i],cmap='gray')
      ax.set_title(f"Pred:{y_pred[i]}\n Actual:{y_test[i]}")
      ax.axis('off')
plt.show()
```

OUTPUT:

```
SVC
SVC(C=10, gamma=0.0001)
```

```
Accuracy: 0.9917

Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       0.95      1.00      0.97        36
           2       1.00      1.00      1.00        35
           3       1.00      1.00      1.00        37
           4       1.00      1.00      1.00        36
           5       1.00      1.00      1.00        37
           6       1.00      1.00      1.00        36
           7       0.97      1.00      0.99        36
           8       1.00      0.94      0.97        35
           9       1.00      0.97      0.99        36

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```



LEARNING OUTCOMES:

EXPERIMENT 6(B)

AIM: Support Vector Machines (SVM): Implement SVM on a dataset and evaluate the model's performance.

THEORY:

- SVR is the regression counterpart of Support Vector Machine (SVM).
- Instead of predicting discrete labels, SVR predicts a continuous target variable.
- It tries to fit a function f(x) such that:
  - The deviations from the actual targets are at most ε (epsilon).
  - Complexity of the model is minimized.

Mathematical formulation:

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

Where:

C: Regularization parameter (controls trade-off between flatness and tolerance to deviations).

ϵ: Tube size for acceptable error.

γ: Parameter of RBF kernel controlling the influence of each support vector.

RBF Kernel:

$$K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$$

ABOUT THE DATASET:

- Available in sklearn.datasets.
- Contains 442 samples and 10 baseline features (e.g., age, BMI, blood pressure, serum measurements).
- Target is a continuous measure of disease progression after one year.

SOURCE CODE:

```
from sklearn.datasets import load_diabetes
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pyplot as plt
```

```
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
svm_reg = SVR(kernel = "rbf", C = 100, gamma = 0.1, epsilon = 0.1)
```

```
svm_reg.fit(X_train, y_train)
```

```
y_pred = svm_reg.predict(X_test)
```

```
df_results_reg = pd.DataFrame({"Actual": y_test[:20], "Predicted":y_pred[:20]})
print("\n Diabetes Regression Results (first 20 rows):\n")
print(df_results_reg)
```

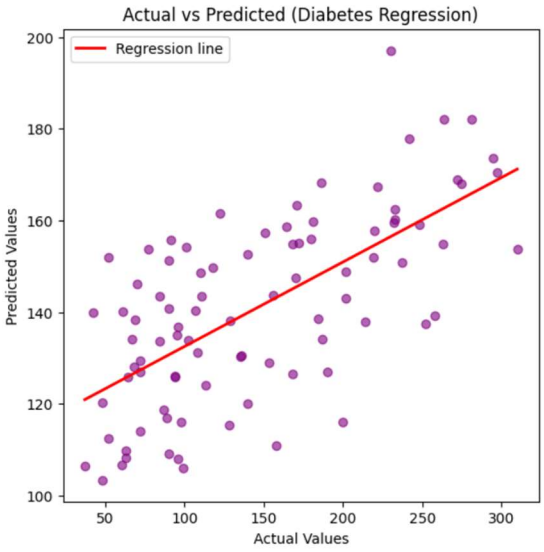
```
from sklearn.linear_model import LinearRegression
import numpy as np

plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred, alpha=0.6, color="purple")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted (Diabetes Regression)")

# ---- Fit a regression line: y_pred = a * y_test + b ----
reg = LinearRegression()
reg.fit(y_test.reshape(-1,1), y_pred)

# Prepare smooth x values
x_line = np.linspace(y_test.min(), y_test.max(), 100).reshape(-1,1)
y_line = reg.predict(x_line)

plt.plot(x_line, y_line, color="red", linewidth=2, label="Regression line")
plt.legend()
plt.show()
```

OUTPUT:

```
SVR
SVR(C=100, gamma=0.1)
```

Diabetes Regression Results (first 20 rows):

	Actual	Predicted
0	219.0	151.899882
1	70.0	146.113035
2	202.0	148.785149
3	230.0	197.092018
4	111.0	143.426537
5	84.0	133.719900
6	242.0	177.756316
7	272.0	168.992880
8	94.0	125.797882
9	96.0	136.814002
10	94.0	126.099704
11	252.0	137.418360
12	99.0	106.074805
13	297.0	170.465613
14	135.0	130.274215
15	67.0	134.149339
16	295.0	173.618414
17	264.0	182.018866
18	170.0	147.478972
19	275.0	168.038570



LEARNING OUTCOMES:

EXPERIMENT 6(C)

AIM: Support Vector Machines (SVM): Implement SVM on a dataset and evaluate the model's performance.

THEORY:

SVM is a supervised machine learning algorithm used for both classification and regression.

It tries to find an optimal hyperplane that separates data points of different classes with the maximum margin.

For non-linearly separable data, SVM uses the kernel trick to map the data into higher dimensions where a separating hyperplane exists.

Key equations:

1. Decision Function:

   $$F(x) = w.x + b$$

   Where w is the weight vector and b is the bias.

2. Optimization Objective:

   $$min\frac{1}{2}||w||^2$$

3. Kernel Function:

   $$K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$$

ABOUT THE DATASET:

- 1,797 grayscale images of size 8×8 pixels.
- Each image represents a handwritten digit (0–9).
- Features: 64 pixel intensity values per sample.
- Target: digit label (0–9).

SOURCE CODE:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
```

```
digits = load_digits()
X, y = digits.data, digits.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42, stratify = y)
```

```
svm_clf = SVC(kernel='rbf', C=10, gamma=0.001)
svm_clf.fit(X_train, y_train)
```

```
y_pred = svm_clf.predict(X_test)
```

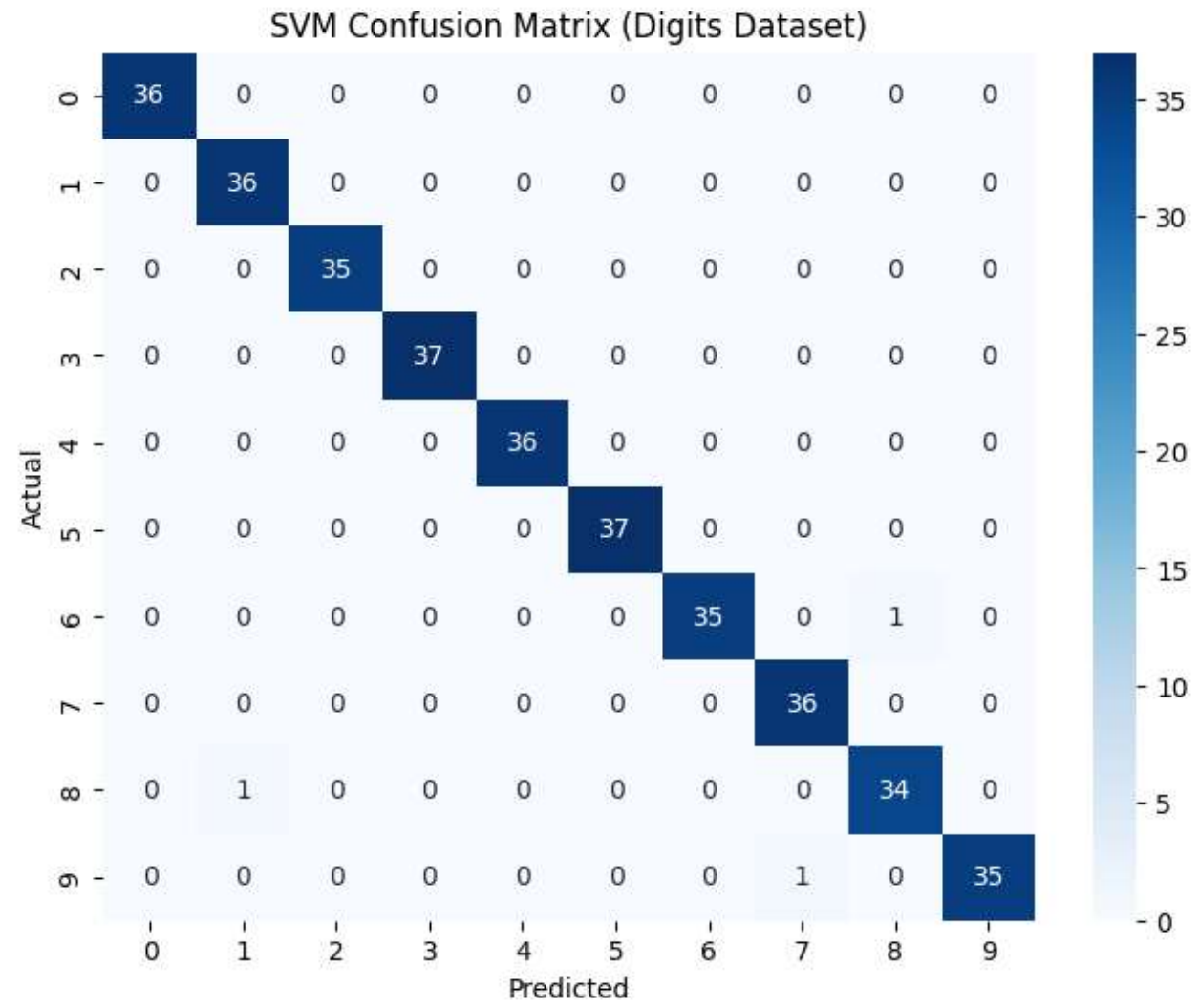
```
print("\n Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("SVM Confusion Matrix (Digits Dataset)")
plt.show()
```

OUTPUT:

```
SVC
SVC(C=10, gamma=0.001)
```

```
 Accuracy: 0.9916666666666667

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       0.97      1.00      0.99        36
           2       1.00      1.00      1.00        35
           3       1.00      1.00      1.00        37
           4       1.00      1.00      1.00        36
           5       1.00      1.00      1.00        37
           6       1.00      0.97      0.99        36
           7       0.97      1.00      0.99        36
           8       0.97      0.97      0.97        35
           9       1.00      0.97      0.99        36

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```



LEARNING OUTCOMES:

EXPERIMENT 7

AIM: Naïve Bayes: Implement Naïve Bayes on a dataset and evaluate the model's performance.

THEORY:

- Naïve Bayes is a probabilistic supervised learning algorithm based on Bayes' theorem.
- It assumes that all features are conditionally independent given the class label (the "naïve" assumption).
- Bayes' theorem:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

Where:

- P(C|X): Posterior probability of class CCC given data XXX.
- P(X|C): Likelihood of data given class.
- P(C): Prior probability of class.
- P(X): Probability of data.

Gaussian Naïve Bayes assumes that the likelihood P(X|C)P(X|C)P(X|C) follows a normal distribution:

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(x_i - \mu_k)^2}{2\sigma_k^2}\right)$$

ABOUT THE DATASET:

The Iris dataset is a classical dataset in machine learning with:

- 150 samples of iris flowers.
- 3 species: Setosa, Versicolor, Virginica.
- 4 features: Sepal Length, Sepal Width, Petal Length, Petal Width.

Goal: Classify flowers into their species based on features.

SOURCE CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pandas as pd
import seaborn as sns
from sklearn.naive_bayes import GaussianNB
```

```
data = pd.read_csv(r"C:\Users\Vasudev Grover\Downloads\Iris.csv")
print(data.head())
```

```
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

```
if y.dtype == "object":
    y = y.astype('category').cat.codes
```

```
plt.figure(figsize = (6,4))
sns.countplot(x = y, palette = "Set2")
plt.title("Class distribution in Iris Dataset")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```

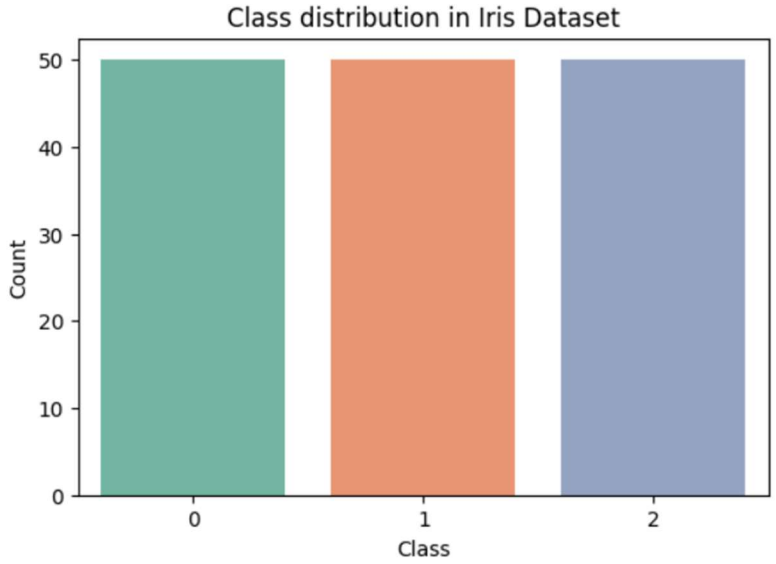
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
nb = GaussianNB()
nb.fit(X_train, y_train)
```

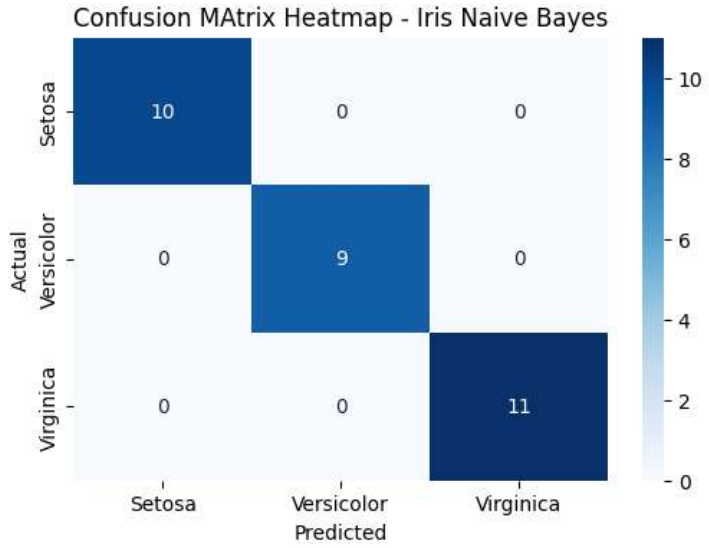
```
y_pred = nb.predict(X_test)

print("\n Accuracy: ", accuracy_score(y_test, y_pred))
print("\n Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\n Classification Report:\n", classification_report(y_test, y_pred))
```

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (6,4))
sns.heatmap(cm, annot = True, fmt = "d", cmap = "Blues",
            xticklabels = ["Setosa", "Versicolor", "Virginica"],
            yticklabels = ["Setosa", "Versicolor", "Virginica"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion MAtrix Heatmap - Iris Naive Bayes")
plt.show()
```

OUTPUT:




```
Accuracy:  1.0

Confusion Matrix:
[[10  0  0]
[ 0  9  0]
[ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```



LEARNING OUTCOMES:

EXPERIMENT 8

AIM: Gradient Boosting: Implement gradient boosting algorithm on a dataset and evaluate the model's performance.

THEORY:
Gradient Boosting is an ensemble learning technique that combines multiple weak learners (typically decision trees) to form a strong predictive model. It builds trees sequentially, where each new tree focuses on minimizing the residual errors of the previous ensemble.

Working Steps:

- Initialize the model with a constant prediction (e.g., the mean of target values).
- Compute residuals (errors) from the current model.
- Train a new weak learner (decision tree) on these residuals.
- Update the model by adding this new learner, scaled by a learning rate (η).
- Repeat steps 2–4 for a fixed number of iterations (n_estimators).

Advantages:

- Reduces bias and variance simultaneously.
- Handles complex nonlinear relationships.
- Provides feature importance measures.

Disadvantages:

- Computationally expensive.
- Sensitive to hyperparameters and overfitting if not tuned.

ABOUT THE DATASET:

- The experiment uses the Titanic Survival Dataset, available from DataScienceDojo's open repository.
- The dataset consists of 891 rows and 12 columns, combining both numerical and categorical features.
- **Key features include:**
  - **Pclass:** Passenger class (1st, 2nd, or 3rd)
  - **Sex:** Gender of the passenger
  - **Age:** Age of the passenger
  - **SibSp:** Number of siblings/spouses aboard
  - **Parch:** Number of parents/children aboard
  - **Fare:** Fare paid for the ticket
  - **Embarked:** Port of embarkation (C, Q, S)

SOURCE CODE:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

✓ 0.0s

```
# Load dataset
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
df = pd.read_csv(url)
print("Dataset shape:", df.shape)
print(df.head())
```

✓ 0.9s

```
# Select features
features = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]
df = df[features + ["Survived"]]
```

✓ 0.0s

```
# Handle missing values
df["Age"].fillna(df["Age"].median(), inplace=True)
df["Embarked"].fillna(df["Embarked"].mode()[0], inplace=True)

# Encode categorical variables
df["Sex"] = df["Sex"].map({"male": 0, "female": 1})
df = pd.get_dummies(df, columns=["Embarked"], drop_first=True)

# Split data
X = df.drop("Survived", axis=1)
y = df["Survived"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

✓ 0.1s

```
# Train Gradient Boosting Classifier
gb = GradientBoostingClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=3,
    random_state=42
)
gb.fit(X_train, y_train)

# Predictions
y_pred = gb.predict(X_test)
```

✓ 0.6s

```
accuracy = accuracy_score(y_test, y_pred)
print(f"\nGradient Boosting Accuracy: {accuracy:.3f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix - Gradient Boosting (Titanic Dataset)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

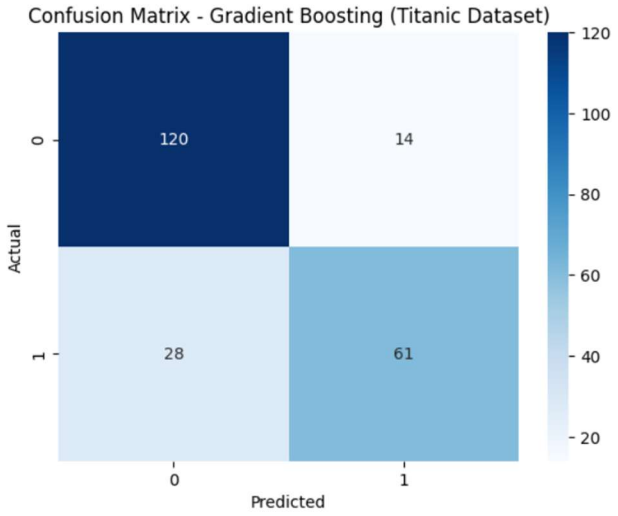
✓ 0.2s

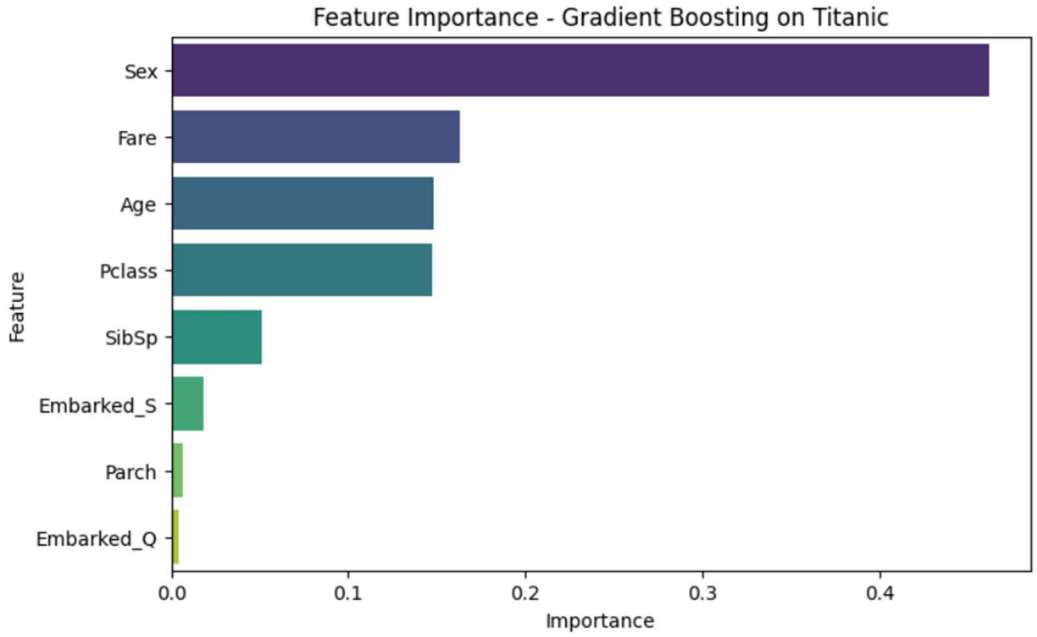
```
feature_importance = pd.Series(gb.feature_importances_, index=X.columns).sort_values(ascending=False)
plt.figure(figsize=(8,5))
sns.barplot(x=feature_importance, y=feature_importance.index, palette="viridis")
plt.title("Feature Importance - Gradient Boosting on Titanic")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

✓ 0.4s

OUTPUT:

```
Dataset shape: (891, 12)
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S


Gradient Boosting Accuracy: 0.812

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.90      0.85       134
           1       0.81      0.69      0.74        89

    accuracy                           0.81       223
   macro avg       0.81      0.79      0.80       223
weighted avg       0.81      0.81      0.81       223
```





LEARNING OUTCOMES:

EXPERIMENT 9

AIM: To build and evaluate a Convolutional Neural Network (CNN) model for image classification using the Fashion MNIST dataset, and to visualize model performance through accuracy, loss, and confusion matrix plots.

THEORY:
A Convolutional Neural Network (CNN) is a specialized deep learning architecture designed to automatically and adaptively learn spatial hierarchies of features from image data. Unlike traditional Artificial Neural Networks (ANNs), which treat images as flattened vectors, CNNs preserve the spatial structure of images — enabling them to efficiently capture local patterns such as edges, textures, and shapes.

Key components of a CNN:

(a) Convolutional Layer:
This is the core building block of a CNN. It applies a set of filters (kernels) that slide over the input image to detect low-level features (like edges and curves) in early layers and high-level features (like clothing patterns or outlines) in deeper layers.
Operation: Convolution → element-wise multiplication followed by summation.
Output: Feature Map, representing activation intensity for each learned feature.

(b) Activation Function (ReLU):
The Rectified Linear Unit (ReLU) introduces non-linearity, helping the network learn complex relationships by suppressing negative values while retaining positive signals.

(c) Pooling Layer:
This layer reduces the spatial dimensions of feature maps (down-sampling) to minimize computational complexity and prevent overfitting.
Max Pooling selects the highest value in each region, preserving key features.

(d) Flatten Layer:
Converts the 2D feature maps into a 1D vector so that it can be passed to fully connected (dense) layers for classification.

(e) Fully Connected (Dense) Layer:
These layers perform high-level reasoning by combining extracted features to make final predictions.

(f) Output Layer (Softmax):
The softmax activation function converts output logits into probability scores for each class, ensuring the sum equals 1.

Why CNNs for Image Classification?

- CNNs require fewer parameters than fully connected networks due to weight

sharing, making them computationally efficient.

- They automatically learn important features without the need for manual feature extraction.
- CNNs achieve high generalization performance on unseen image data.

ABOUT THE DATASET:

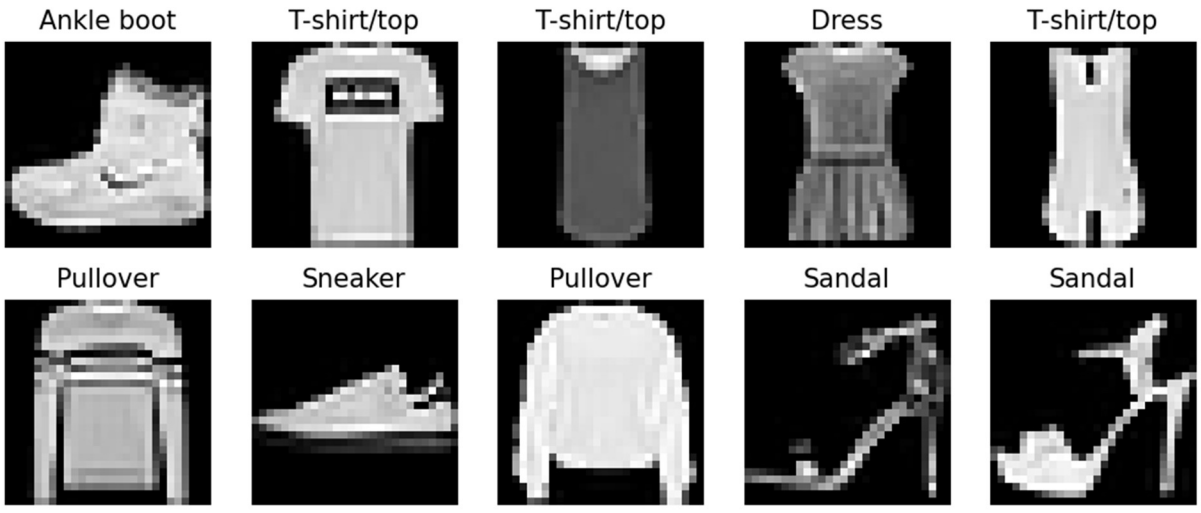
- **The Total Samples:** 70,000 (60,000 train + 10,000 test)
- **Image Size:** 28×28 pixels (grayscale)
- **Number of Classes:** 10
- **Features:** Pixel intensity values (0–255, normalized to 0–1)
- **Target:** Class label of the clothing item

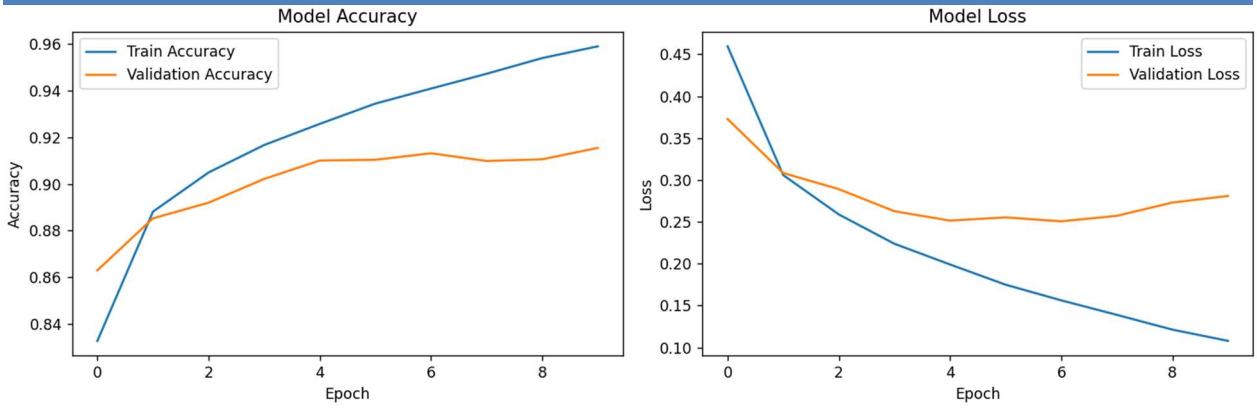
SOURCE CODE:

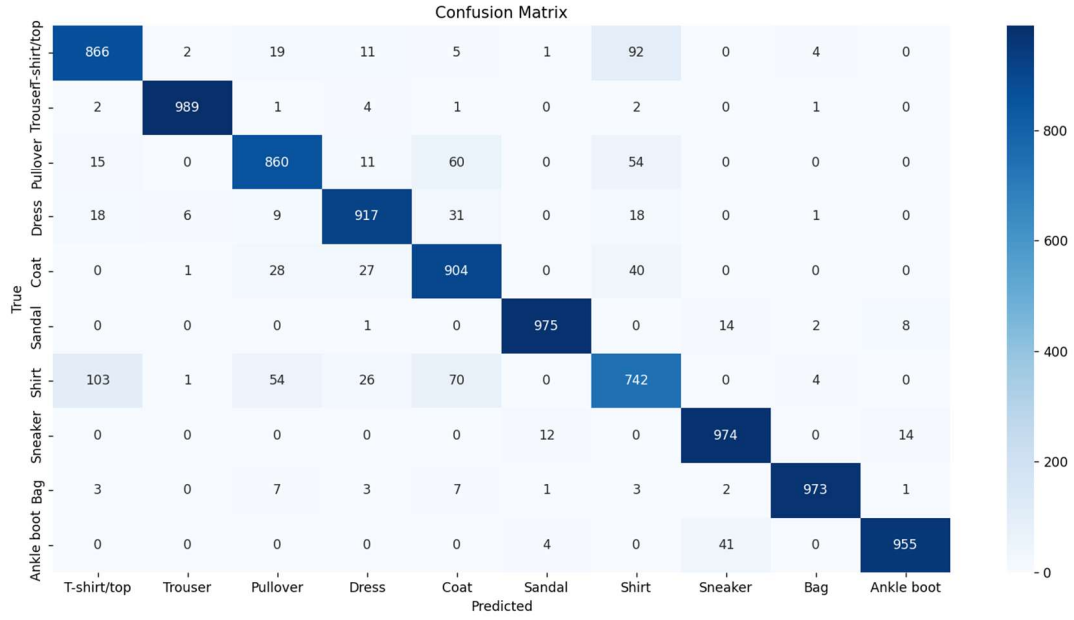
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

(x_train, y_train), (x_test, y_test) = datasets.fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

plt.figure(figsize=(10,4))
for i in range(10):
    plt.subplot(2,5,i+1)
    plt.imshow(x_train[i].reshape(28,28), cmap='gray')
    plt.title(class_names[y_train[i]])
    plt.axis('off')
plt.show()

model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\nTest Accuracy: {test_acc:.4f}")
```

```
43    plt.figure(figsize=(12,4))
44    plt.subplot(1,2,1)
45    plt.plot(history.history['accuracy'], label='Train Accuracy')
46    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
47    plt.title("Model Accuracy")
48    plt.xlabel("Epoch")
49    plt.ylabel("Accuracy")
50    plt.legend()
51
52    plt.subplot(1,2,2)
53    plt.plot(history.history['loss'], label='Train Loss')
54    plt.plot(history.history['val_loss'], label='Validation Loss')
55    plt.title("Model Loss")
56    plt.xlabel("Epoch")
57    plt.ylabel("Loss")
58    plt.legend()
59    plt.tight_layout()
60    plt.show()
61
62    y_pred = np.argmax(model.predict(x_test), axis=1)
63    cm = confusion_matrix(y_test, y_pred)
64
65    plt.figure(figsize=(10,8))
66    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
67                xticklabels=class_names, yticklabels=class_names)
68    plt.xlabel("Predicted")
69    plt.ylabel("True")
70    plt.title("Confusion Matrix")
71    plt.show()
72
73    print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=class_names))
74
```

OUTPUT:








```
Classification Report:
              precision    recall  f1-score   support

 T-shirt/top       0.86      0.87      0.86      1000
     Trouser       0.99      0.99      0.99      1000
    Pullover       0.88      0.86      0.87      1000
       Dress       0.92      0.92      0.92      1000
        Coat       0.84      0.90      0.87      1000
      Sandal       0.98      0.97      0.98      1000
       Shirt       0.78      0.74      0.76      1000
     Sneaker       0.94      0.97      0.96      1000
         Bag       0.99      0.97      0.98      1000
  Ankle boot       0.98      0.95      0.97      1000

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

LEARNING OUTCOMES:

EXPERIMENT 10

AIM: Recurrent Neural Networks (RNN): Implement RNN on a text classification dataset and evaluate the model's performance.

THEORY:

Recurrent Neural Networks (RNNs) are designed to handle **sequential data** by maintaining a **hidden state** that captures information from previous inputs. In text classification, RNNs process each word in a sentence sequentially, learning context and meaning from word order. The final hidden state helps predict the class label (e.g., sentiment or topic).
Variants like **LSTM** and **GRU** improve performance by solving the **vanishing gradient problem** and handling long-term dependencies.

ABOUT THE DATASET:

- **Source:** Collected from Usenet newsgroups.
- **Total Documents:** ~20,000 text samples.
- **Number of Classes:** 20 categories.
- **Type:** Multi-class text classification dataset.
- **Data Split:** Commonly divided into **training** and **test** subsets.
- **Content:** Raw text data, often requiring preprocessing (tokenization, stop-word removal, etc.).

SOURCE CODE:

[1] ✓ 23s
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
import numpy as np
```

[2] ✓ 15s
```
newsgroups = fetch_20newsgroups(subset='all', remove=('headers','footers','quotes'))
texts, labels = newsgroups.data, newsgroups.target
num_classes = np.unique(labels).shape[0]
```

[3] ✓ 0s
```
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2, random_state=42)
```

[4] ✓ 6s
```
VOCAB_SIZE = 20000
MAX_LEN = 300
tokenizer = Tokenizer(num_words=VOCAB_SIZE, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)
X_train_pad = pad_sequences(X_train_seq, maxlen=MAX_LEN)
X_test_pad = pad_sequences(X_test_seq, maxlen=MAX_LEN)
```

[5] ✓ 0s
```
# One-hot encode labels
y_train_cat = to_categorical(y_train, num_classes)
y_test_cat = to_categorical(y_test, num_classes)
```

```
[6]
✓ 0s
# Build GRU model
model = models.Sequential([
    layers.Embedding(VOCAB_SIZE, 128, input_length=MAX_LEN),
    layers.GRU(128, dropout=0.3, recurrent_dropout=0.3),
    layers.Dense(num_classes, activation='softmax')
])

Show hidden output
```

```
[10]
# Compile & train
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
model.fit(X_train_pad, y_train_cat, epochs=5, batch_size=64, validation_split=0.2, verbose=2)

Show hidden output
```

```
[9]
✓ 13s
# Evaluate
loss, acc = model.evaluate(X_test_pad, y_test_cat, verbose=0)
print(f"Test Accuracy: {acc:.4f}")

Show hidden output
```

OUTPUT:

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 300, 128)	2,560,000
gru (GRU)	(None, 128)	99,072
dense (Dense)	(None, 20)	2,580

```
Total params: 2,661,652 (10.15 MB)
Trainable params: 2,661,652 (10.15 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/5
189/189 - 203s - 1s/step - accuracy: 0.7382 - loss: 0.8525 - val_accuracy: 0.5030 - val_loss: 1.7356
Epoch 2/5
189/189 - 195s - 1s/step - accuracy: 0.8015 - loss: 0.6655 - val_accuracy: 0.5199 - val_loss: 1.7578
Epoch 3/5
189/189 - 205s - 1s/step - accuracy: 0.8461 - loss: 0.5214 - val_accuracy: 0.5365 - val_loss: 1.7448
Epoch 4/5
189/189 - 198s - 1s/step - accuracy: 0.8783 - loss: 0.4172 - val_accuracy: 0.5434 - val_loss: 1.7645
Epoch 5/5
189/189 - 204s - 1s/step - accuracy: 0.9052 - loss: 0.3360 - val_accuracy: 0.5613 - val_loss: 1.8103
<keras.src.callbacks.history.History at 0x7e24e878c290>
```

```
Test Accuracy: 0.4504
```

LEARNING OUTCOMES:

EXPERIMENT 11

AIM: Autoencoders: Implement autoencoders on an image dataset and evaluate the model's performance.

THEORY:

An **Autoencoder** is an unsupervised neural network used for **dimensionality reduction**, **feature learning**, and **data reconstruction**.

It consists of two main parts:

1. **Encoder:** Compresses the input image into a lower-dimensional latent representation (feature space).
2. **Decoder:** Reconstructs the original image from this compressed representation.

The network is trained to minimize the **reconstruction error**, i.e., the difference between the input and the output image.

Autoencoders learn to capture the **essential features** of images, removing noise and redundancy.

ABOUT THE DATASET:

- **Total Images:** 70,000
  - **Training Set:** 60,000
  - **Test Set:** 10,000
- **Image Size:** 28 × 28 pixels (grayscale)
- **Number of Classes:** 10 (digits 0–9)
- **Type:** Labeled image dataset for supervised learning

SOURCE CODE:

[1] ✓ 14s

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
```

[2] ✓ 0s

```
# Load and normalize MNIST dataset
(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
```

> Show hidden output

[3] ✓ 0s

```
# Flatten 28x28 → 784
x_train = x_train.reshape((len(x_train), 28 * 28))
x_test = x_test.reshape((len(x_test), 28 * 28))
```

[4] ✓ 0s

```
# Define Autoencoder architecture
encoding_dim = 64  # latent space dimension

input_img = tf.keras.Input(shape=(784,))
encoded = layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(encoding_dim, activation='relu')(encoded)

decoded = layers.Dense(128, activation='relu')(encoded)
decoded = layers.Dense(784, activation='sigmoid')(decoded)

autoencoder = tf.keras.Model(input_img, decoded)
```

[5]

```
# Compile
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

[6]

```
# Train model
history = autoencoder.fit(
    x_train, x_train,
    epochs=20,
    batch_size=256,
    shuffle=True,
    validation_data=(x_test, x_test),
    verbose=2
)
```

Show hidden output

[7] ✓ 0s

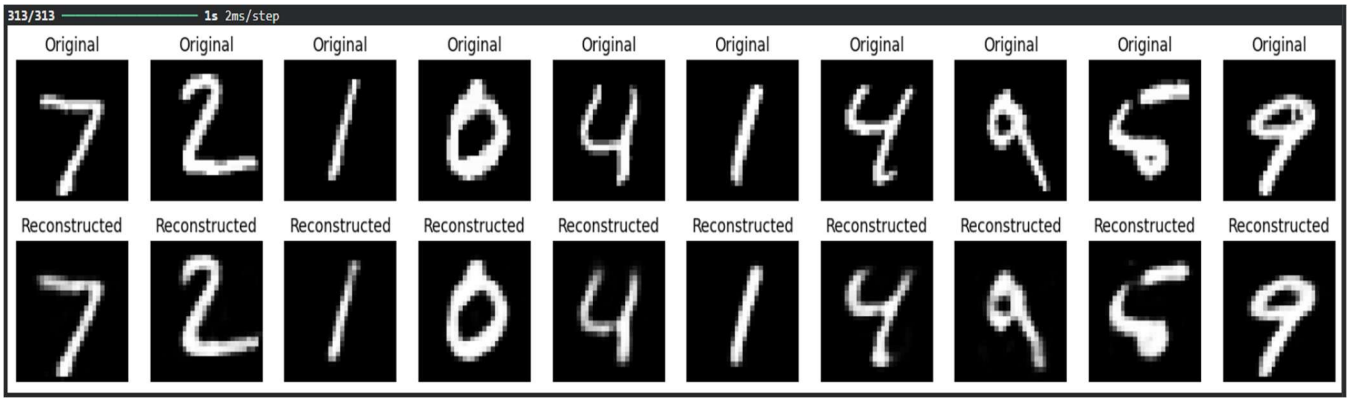
```
#  Evaluate model
test_loss = autoencoder.evaluate(x_test, x_test)
print(f"Reconstruction Loss (Test): {test_loss:.4f}")
```

Show hidden output

[8] ✓ 1s

```
#  Visualize original vs reconstructed images
decoded_imgs = autoencoder.predict(x_test)

n = 10  # display 10 images
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis("off")

    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.title("Reconstructed")
    plt.axis("off")

plt.show()
```

OUTPUT:



LEARNING OUTCOMES:

EXPERIMENT 12

AIM: Generative Adversarial Networks (GANs): Implement GANs on an image dataset and evaluate the model's performance.

THEORY:

A **GAN** is a deep learning model consisting of two competing neural networks:

1. **Generator (G):** Creates fake images from random noise.
2. **Discriminator (D):** Tries to distinguish between real and generated (fake) images.

Both networks are trained simultaneously — the generator improves to fool the discriminator, while the discriminator improves to detect fakes. This adversarial process helps the generator learn to produce highly realistic images.

ABOUT THE DATASET:

- **Total Images:** 70,000
  - **Training Set:** 60,000
  - **Test Set:** 10,000
- **Image Size:** 28 × 28 pixels (grayscale)
- **Number of Classes:** 10 (different clothing categories)
- **Type:** Labeled image dataset for supervised learning

SOURCE CODE:

[1]
```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
```

[2] ✓ 1s
```
# Load Fashion-MNIST dataset
(x_train, _), (_, _) = tf.keras.datasets.fashion_mnist.load_data()
x_train = x_train.reshape(-1, 28*28).astype("float32")
x_train = (x_train - 127.5) / 127.5  # Normalize to [-1, 1]
BUFFER_SIZE = 60000
BATCH_SIZE = 128
```

> Show hidden output

[3] ✓ 0s
```
dataset = tf.data.Dataset.from_tensor_slices(x_train).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

[4] ✓ 0s
```
#  Build the Generator
def build_generator():
    model = tf.keras.Sequential([
        layers.Dense(256, activation="relu", input_shape=(100,)),
        layers.BatchNormalization(),
        layers.Dense(512, activation="relu"),
        layers.BatchNormalization(),
        layers.Dense(1024, activation="relu"),
        layers.BatchNormalization(),
        layers.Dense(28*28, activation="tanh")
    ])
    return model
```

[5] ✓ 0s

```
#  Build the Discriminator
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Dense(512, input_shape=(784,)),
        layers.LeakyReLU(0.2),
        layers.Dense(256),
        layers.LeakyReLU(0.2),
        layers.Dense(1, activation="sigmoid")
    ])
    return model

generator = build_generator()
discriminator = build_discriminator()
```

Show hidden output

[6] ✓ 0s

```
#  Compile the discriminator
discriminator.compile(loss="binary_crossentropy",
                      optimizer=tf.keras.optimizers.Adam(0.0002, 0.5),
                      metrics=["accuracy"])
```

[7] ✓ 0s

```
# Build and compile the combined model
discriminator.trainable = False
z = layers.Input(shape=(100,))
img = generator(z)
validity = discriminator(img)
combined = tf.keras.Model(z, validity)
combined.compile(loss="binary_crossentropy", optimizer=tf.keras.optimizers.Adam(0.0002, 0.5))
```

[8]

```
# Training loop
EPOCHS = 5000
SAVE_INTERVAL = 500
noise_dim = 100

for epoch in range(1, EPOCHS + 1):
    # Train discriminator
    noise = np.random.normal(0, 1, (BATCH_SIZE, noise_dim))
    gen_imgs = generator.predict(noise, verbose=0)
    real_imgs = x_train[np.random.randint(0, x_train.shape[0], BATCH_SIZE)]

    real_labels = np.ones((BATCH_SIZE, 1))
    fake_labels = np.zeros((BATCH_SIZE, 1))

    d_loss_real = discriminator.train_on_batch(real_imgs, real_labels)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake_labels)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    # Train generator
    noise = np.random.normal(0, 1, (BATCH_SIZE, noise_dim))
    valid_y = np.ones((BATCH_SIZE, 1))
    g_loss = combined.train_on_batch(noise, valid_y)

    # Print progress
    if epoch % 500 == 0:
        print(f"{epoch} [D loss: {d_loss[0]:.4f}, acc.: {100*d_loss[1]:.2f}%] [G loss: {g_loss:.4f}]")

    # Save generated samples
    if epoch % SAVE_INTERVAL == 0:
        noise = np.random.normal(0, 1, (16, noise_dim))
        gen_imgs = generator.predict(noise, verbose=0)
        gen_imgs = 0.5 * gen_imgs + 0.5

        plt.figure(figsize=(4,4))
        for i in range(16):
            plt.subplot(4,4,i+1)
            plt.imshow(gen_imgs[i].reshape(28,28), cmap="gray")
            plt.axis('off')
        plt.suptitle(f"Fashion-MNIST GAN: Epoch {epoch}")
        plt.show()
```

OUTPUT:


500 [D loss: 2.6108, acc.: 49.75%] [G loss: 0.0246]
Fashion-MNIST GAN: Epoch 500
2500 [D loss: 4.0559, acc.: 49.73%] [G loss: 0.0052]
Fashion-MNIST GAN: Epoch 2500
1000 [D loss: 3.2644, acc.: 49.72%] [G loss: 0.0126]
Fashion-MNIST GAN: Epoch 1000
3000 [D loss: 4.2140, acc.: 49.72%] [G loss: 0.0043]
Fashion-MNIST GAN: Epoch 3000
1500 [D loss: 3.6188, acc.: 49.73%] [G loss: 0.0085]
Fashion-MNIST GAN: Epoch 1500
3500 [D loss: 4.3496, acc.: 49.73%] [G loss: 0.0037]
Fashion-MNIST GAN: Epoch 3500
2000 [D loss: 3.8651, acc.: 49.73%] [G loss: 0.0064]
Fashion-MNIST GAN: Epoch 2000
4000 [D loss: 4.4687, acc.: 49.72%] [G loss: 0.0032]
Fashion-MNIST GAN: Epoch 4000

LEARNING OUTCOMES: