Task-03

Implement a support vector machine (SVM) to classify images of cats and dogs from the Kaggle data set.

Dataset :- https://www.kaggle.com/c/dogs-vs-cats/data (https://www.kaggle.com/c/dogs-vs-cats/data)

In [1]:
```
import zipfile
import os

# Path to the ZIP file
zip_file_path = 'archive (2).zip'

# Directory where you want to extract the contents
extract_dir = 'archive_extracted'

# Check if the ZIP file exists
if os.path.exists(zip_file_path):
    # Create the directory if it doesn't exist
    if not os.path.exists(extract_dir):
        os.makedirs(extract_dir)

    # Extract the ZIP file
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extract_dir)
    print(f"ZIP file extracted to {extract_dir}")
else:
    print(f"The ZIP file {zip_file_path} does not exist.")
```

```
ZIP file extracted to archive_extracted
```

In [2]:
```
def print_directory_structure(startpath, level=0):
    for root, dirs, files in os.walk(startpath):
        indent = ' ' * 4 * (level)
        print(f"{indent}{os.path.basename(root)}/")
        subindent = ' ' * 4 * (level + 1)
        for f in files:
            print(f"{subindent}{f}")
        for d in dirs:
            print_directory_structure(os.path.join(root, d), level + 1)

# Inspect the directory structure
print_directory_structure(extract_dir)
```

```
            0.jpg
            1.jpg
            10.jpg
            100.jpg
            101.jpg
            102.jpg
            103.jpg
            104.jpg
            105.jpg
            106.jpg
            107.jpg
            108.jpg
            109.jpg
            11.jpg
            110.jpg
            111.jpg
            112.jpg
            113.jpg
            114.jpg
```

In [3]:
```
train_dir = os.path.join(extract_dir, 'cats_and_dogs_filtered', 'train'
validation_dir = os.path.join(extract_dir, 'cats_and_dogs_filtered', 'v
```

In [4]:
```
# Path to the extracted directory
extracted_dir = 'archive_extracted/dogvscat'

# Define the directories
train_dir = os.path.join(extracted_dir, 'train')
test_dir = os.path.join(extracted_dir, 'test')

# Ensure the train and test directories exist
if not os.path.exists(train_dir):
    print(f"The directory {train_dir} does not exist.")
if not os.path.exists(test_dir):
    print(f"The directory {test_dir} does not exist.")

# Check contents of the train directory
print("Contents of the train directory:", os.listdir(train_dir))

# Check contents of the test directory
print("Contents of the test directory:", os.listdir(test_dir))
```

```
Contents of the train directory: ['0', '1']
Contents of the test directory: ['0', '1']
```

In [5]:
```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\senap\anacond
a3\lib\site-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.17.0 in c:\users\senap\anacond
a3\lib\site-packages (from opencv-python) (1.21.5)
```

In [6]:
```
import os
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score,confu
```

In [7]:

```
# Function to load images and labels
def load_images_and_labels(directory, label, image_size=(64, 64)):
    images = []
    labels = []
    for root, dirs, files in os.walk(directory):
        for filename in files:
            img_path = os.path.join(root, filename)
            img = cv2.imread(img_path)
            if img is not None:
                img = cv2.resize(img, image_size)
                images.append(img)
                labels.append(label)
    return np.array(images), np.array(labels)

# Load test data
test_data, test_labels = load_images_and_labels(test_dir, 1)
```

In [8]:

```
import os
import cv2
import numpy as np

def load_images_and_labels(directory, label):
    images = []
    labels = []
    for filename in os.listdir(directory):
        img_path = os.path.join(directory, filename)
        img = cv2.imread(img_path)
        if img is not None:
            # Resize image to a consistent size if needed
            img = cv2.resize(img, (64, 64))  # Adjust desired_width and
            # Normalize or preprocess the image as required for your SV
            # Example: img = preprocess_image(img)
            # Append image and corresponding label
            images.append(img)
            labels.append(label)
    return np.array(images), np.array(labels)

# Define paths to your dataset directories
train_cats_dir = 'archive_extracted/dogvscat/train/0'
train_dogs_dir = 'archive_extracted/dogvscat/train/1'
test_cats_dir = 'archive_extracted/dogvscat/test/0'
test_dogs_dir = 'archive_extracted/dogvscat/test/1'

# Load images and labels for cats and dogs
cats_train, cats_labels_train = load_images_and_labels(train_cats_dir,
dogs_train, dogs_labels_train = load_images_and_labels(train_dogs_dir,

# Concatenate labels
train_labels = np.concatenate((cats_labels_train, dogs_labels_train), a

# Check shapes to ensure consistency
print("Shape of cats_train:", cats_train.shape)
print("Shape of dogs_train:", dogs_train.shape)
print("Shape of train_labels:", train_labels.shape)

# Flatten the features to feed into SVM
train_data_flattened = np.reshape(train_labels, (train_labels.shape[0],

# Standardize the features
scaler = StandardScaler()
train_data_scaled = scaler.fit_transform(train_data_flattened)

# Train the SVM classifier (you can train on the entire dataset for dem
svm = SVC(kernel='linear')
svm.fit(train_data_scaled, train_labels)

# Continue with prediction or evaluation steps as needed
```

```
Shape of cats_train: (250, 64, 64, 3)
Shape of dogs_train: (250, 64, 64, 3)
Shape of train_labels: (500,)
```

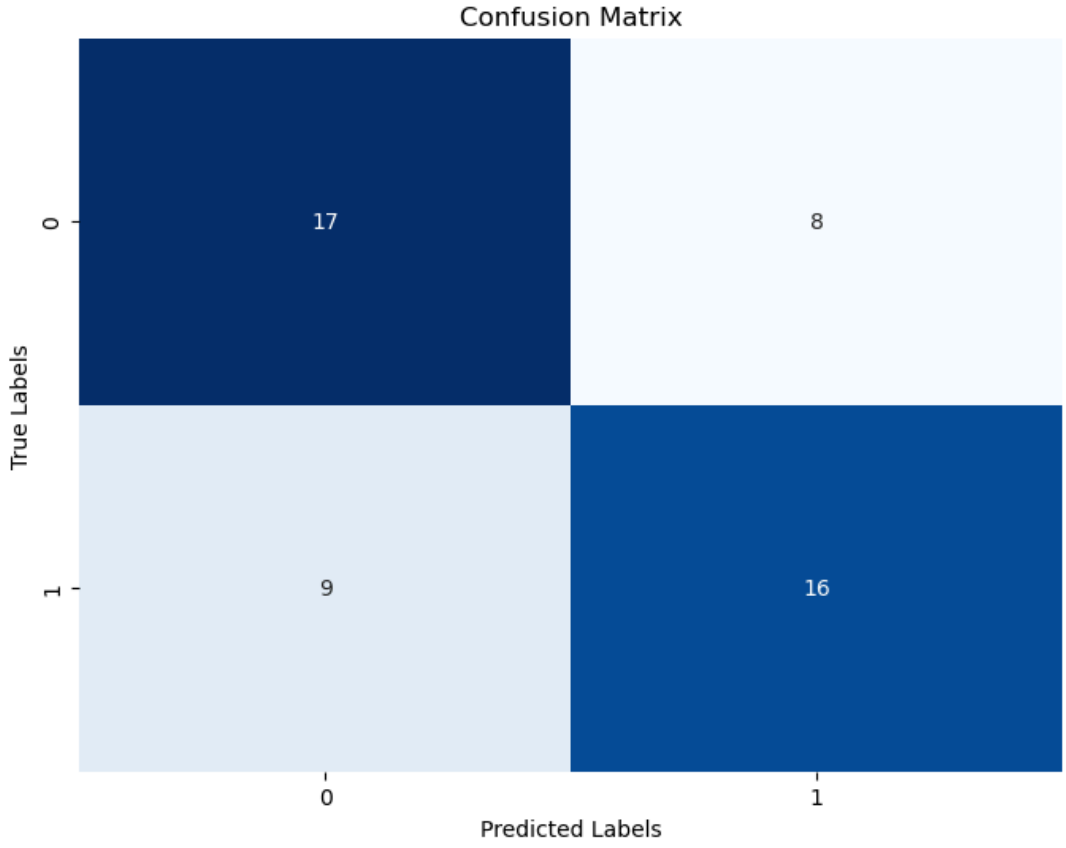
Out[8]:

```
SVC(kernel='linear')
```

In [9]:

```
# Load testing data
cats_test, cats_labels_test = load_images_and_labels(test_cats_dir, 0)
dogs_test, dogs_labels_test = load_images_and_labels(test_dogs_dir, 1)

# Combine training data
train_data = np.concatenate((cats_train, dogs_train), axis=0)
train_labels = np.concatenate((cats_labels_train, dogs_labels_train), a

# Combine testing data
test_data = np.concatenate((cats_test, dogs_test), axis=0)
test_labels = np.concatenate((cats_labels_test, dogs_labels_test), axis

# Flatten the features to feed into SVM
train_data_flattened = train_data.reshape(train_data.shape[0], -1)
test_data_flattened = test_data.reshape(test_data.shape[0], -1)

# Standardize the features
scaler = StandardScaler()
train_data_scaled = scaler.fit_transform(train_data_flattened)
test_data_scaled = scaler.transform(test_data_flattened)

# Train the SVM classifier
svm = SVC(kernel='linear')
svm.fit(train_data_scaled, train_labels)

# Predict on the test set
y_pred = svm.predict(test_data_scaled)

# Evaluate the model
print("Accuracy:", accuracy_score(test_labels, y_pred))
print(classification_report(test_labels, y_pred))

# Print confusion matrix
conf_mat = confusion_matrix(test_labels, y_pred)
print("Confusion Matrix:")
print(conf_mat)

import matplotlib.pyplot as plt
import seaborn as sns
# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.66
              precision    recall  f1-score   support

           0       0.65      0.68      0.67        25
           1       0.67      0.64      0.65        25

    accuracy                           0.66        50
   macro avg       0.66      0.66      0.66        50
weighted avg       0.66      0.66      0.66        50

Confusion Matrix:
[[17  8]
 [ 9 16]]
```



In [10]:

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate precision, recall, and F1-score for each class
precision = precision_score(test_labels, y_pred, average=None)
recall = recall_score(test_labels, y_pred, average=None)
f1 = f1_score(test_labels, y_pred, average=None)

# Define the class labels
class_labels = ['Cats', 'Dogs']

# Plot the metrics
plt.figure(figsize=(10, 6))
plt.plot(class_labels, precision, marker='o', label='Precision')
plt.plot(class_labels, recall, marker='o', label='Recall')
plt.plot(class_labels, f1, marker='o', label='F1-score')

# Adding labels and title
plt.xlabel('Class')
plt.ylabel('Score')
plt.title('Precision, Recall, and F1-score for Each Class')
plt.legend()
plt.grid(True)
plt.ylim(0, 1)  # Since these scores are between 0 and 1

# Show the plot
plt.show()
```