```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

```
import os


dataset_path = '/content/drive/MyDrive/Dataset_ML'


# List folders (classes) in the dataset
print("Classes in the dataset:", os.listdir(dataset_path))
```

```
Classes in the dataset: ['0. Cut Shot', '1. Cover Drive', '2. Straight Drive', '3. Pull Shot', '4. Leg Glance Shot', '5.
```

```
# List class folders
classes = os.listdir(dataset_path)
print("Classes in the dataset:", classes)
```

```
Classes in the dataset: ['0. Cut Shot', '1. Cover Drive', '2. Straight Drive', '3. Pull Shot', '4. Leg Glance Shot', '5.
```

```
# Count images in each folder
for class_name in classes:
    class_folder = os.path.join(dataset_path, class_name)
    image_count = len(os.listdir(class_folder))
    print(f"Class '{class_name}' contains {image_count} images.")
```

```
Class '0. Cut Shot' contains 641 images.
Class '1. Cover Drive' contains 600 images.
Class '2. Straight Drive' contains 600 images.
Class '3. Pull Shot' contains 600 images.
Class '4. Leg Glance Shot' contains 600 images.
Class '5. Scoop Shot' contains 600 images.
```

```
import cv2
import numpy as np
from tensorflow.keras.utils import to_categorical


IMG_SIZE = 224  # Resize all images to 224x224
classes = ['0. Cut Shot', '1. Cover Drive', '2. Straight Drive', '3. Pull Shot', '4. Leg Glance :
class_to_label = {name: idx for idx, name in enumerate(classes)}  # Map class names to numeric l


data = []
labels = []


# Load images from each class folder
for class_name in classes:
    class_folder = os.path.join(dataset_path, class_name)
    label = class_to_label[class_name]  # Get numeric label

    for file_name in os.listdir(class_folder):
        file_path = os.path.join(class_folder, file_name)

        # Read the image
        img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)  # Load in grayscale
        if img is None:
```

```
            print(f"Failed to load {file_path}. Skipping...")
            continue

        # # Apply Bilateral Filter for noise reduction
        # img = cv2.bilateralFilter(img, 9, 75, 75)  # Using Bilateral Filter

        # Resize image
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))

        # Normalize image
        img = img / 255.0  # Scale pixel values to [0, 1]

        # Flatten the image to 1D (SVM requires 1D feature vectors)
        img = img.flatten()

        # Append to data and labels
        data.append(img)
        labels.append(label)


# Convert lists to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)
```

```
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.decomposition import PCA

from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
```

```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42
```

```
# Train an SVM classifier
svm_classifier = SVC(kernel='linear')  # You can try other kernels like 'rbf' or 'poly'
svm_classifier.fit(X_train, y_train)
```

```
▾ SVC
SVC(kernel='linear')
```

```
# Evaluate the model
y_pred = svm_classifier.predict(X_test)
```

```
# Print classification report and accuracy
print("Classification Report:\n", classification_report(y_test, y_pred, target_names=classes))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

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

        0. Cut Shot       0.71      0.63      0.66       150
     1. Cover Drive       0.62      0.63      0.62       123
  2. Straight Drive       0.67      0.72      0.69       127
       3. Pull Shot       0.65      0.62      0.64       109
 4. Leg Glance Shot       0.69      0.71      0.70       114
      5. Scoop Shot       0.75      0.79      0.77       106

           accuracy                           0.68       729
          macro avg       0.68      0.68      0.68       729
```

```
weighted avg       0.68      0.68      0.68       729

Accuracy Score: 0.6803840877914952
```

```
def preprocess_image(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Load the image in grayscale
    if img is None:
        raise ValueError(f"Failed to load image: {image_path}")
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))  # Resize to 224x224
    img = img / 255.0  # Normalize pixel values to [0, 1]
    img = img.flatten()  # Flatten to a 1D array
    return img



# Path to the new test image
new_image_path = '/content/test9.jpeg'

# Preprocess the test image
test_image = preprocess_image(new_image_path)

# Reshape the image to match the input shape of the SVM model (1, -1 for single sample)
test_image = test_image.reshape(1, -1)

# Predict the class
predicted_label = svm_classifier.predict(test_image)[0]

# Map the numeric label back to the class name
predicted_class = classes[predicted_label]

print(f"Predicted Class: {predicted_class}")
```

```
Predicted Class: 1. Cover Drive
```

```
# Define constants
class_to_label = {'2. Straight Drive': 0, '3. Pull Shot': 1}

# Filter data for the two classes
filtered_indices = np.where((labels == class_to_label['2. Straight Drive']) | (labels == class_t
filtered_data = data[filtered_indices]
filtered_labels = labels[filtered_indices]

# Map the labels to 0 and 1 for simplicity
filtered_labels = np.array([0 if label == class_to_label['2. Straight Drive'] else 1 for label i

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(filtered_data, filtered_labels, test_size=0.

# Reduce to 2 dimensions using PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Train an SVM classifier on the reduced data
svm = SVC(kernel='linear', C=1)
svm.fit(X_train_pca, y_train)

# Plot the decision boundary
plt.figure(figsize=(10, 8))

# Scatter plot of the training data
scatter = plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap=plt.cm.Paired, edgec
plt.legend(handles=scatter.legend_elements()[0], labels=['Straight Drive', 'Pull Shot'], title="

# Generate a grid to plot the decision boundary
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1
```

```
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

# Predict on the grid
Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.Paired)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Decision Boundary for Straight Drive vs Pull Shot')
plt.show()


#output visualization


from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()



from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# Binarize labels for ROC (One-vs-Rest)
y_test_binarized = label_binarize(y_test, classes=[0, 1, 2, 3, 4, 5])
y_pred_prob = svm_classifier.decision_function(X_test)

# Compute ROC curve for each class
fpr, tpr, thresholds = {}, {}, {}
roc_auc = {}

for i in range(len(classes)):
    fpr[i], tpr[i], thresholds[i] = roc_curve(y_test_binarized[:, i], y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
plt.figure()
for i in range(len(classes)):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for All Classes')
plt.legend(loc="lower right")
plt.show()



from sklearn.decomposition import PCA

# Apply PCA to reduce to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train)

# Fit SVM model on 2D PCA-transformed data
svm_classifier.fit(X_pca, y_train)

# Create meshgrid for decision boundary visualization
x_min, x_max = X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1
```

```
y_min, y_max = X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

Z = svm_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundary
plt.contourf(xx, yy, Z, alpha=0.8, cmap='coolwarm')

# Scatter plot of training points
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_train, cmap='coolwarm', s=30, edgecolors='k')
plt.title('Decision Boundary with PCA (All Classes)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()



# Plot support vectors (on PCA-reduced data)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_train, cmap='coolwarm', s=30, edgecolors='k')
plt.scatter(svm_classifier.support_vectors_[:, 0], svm_classifier.support_vectors_[:, 1],
            s=100, facecolors='none', edgecolors='k', linewidth=1.5, label='Support Vectors')
plt.title("Support Vectors (PCA-reduced 2D space)")
plt.legend()
plt.show()



# PCA feature importance visualization (variance ratio)
explained_variance = pca.explained_variance_ratio_
plt.bar(range(len(explained_variance)), explained_variance, alpha=0.7, align='center', color='sky
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('PCA Component Importance')
plt.show()
```