

```

import numpy as np
import cv2
import os
from google.colab import files
from sklearn import svm
from skimage import exposure
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from skimage.feature import hog
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Function to upload images
def upload_images(label):
    print(f"Please upload your {label} images.")
    uploaded = files.upload()
    images = []
    for filename in uploaded.keys():
        img = cv2.imdecode(np.frombuffer(uploaded[filename], np.uint8), cv2.IMREAD_COLOR)
        if img is not None:
            img = cv2.resize(img, (128, 128)) # Resize for consistency
            images.append(img)
    return np.array(images)

# Step 2: Upload real and fake images separately
real_images = upload_images("real")
fake_images = upload_images("fake")

# Step 3: Prepare labels
labels_real = np.zeros(len(real_images)) # 0 for real
labels_fake = np.ones(len(fake_images)) # 1 for fake

# Combine images and labels
X = np.concatenate((real_images, fake_images), axis=0)
y = np.concatenate((labels_real, labels_fake), axis=0)

# Step 4: Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    fill_mode='nearest'
)

datagen.fit(X)

# Generate augmented images
augmented_images = []
augmented_labels = []

for i in range(len(X)):
    for x in datagen.flow(X[i:i+1], batch_size=1):
        augmented_images.append(x[0])
        augmented_labels.append(y[i])
        if len(augmented_images) >= 1000:
            break

augmented_images = np.array(augmented_images)
augmented_labels = np.array(augmented_labels)

# Combine original and augmented images
X_augmented = np.concatenate((X, augmented_images), axis=0)
y_augmented = np.concatenate((y, augmented_labels), axis=0)

# Step 5: Feature extraction using HOG
def extract_hog_features(images):
    hog_features = []
    for img in images:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        features = hog(gray, pixels_per_cell=(16, 16), cells_per_block=(2, 2), visualize=False)
        hog_features.append(features)
    return np.array(hog_features)

# Extract HOG features
X_hog = extract_hog_features(X_augmented)

```

```

# Step 6: Feature Scaling
scaler = StandardScaler()
X_hog_scaled = scaler.fit_transform(X_hog)

# Step 7: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_hog_scaled, y_augmented, test_size=0.2, random_state=42, stratify=y_augmented)

# Step 8: Hyperparameter tuning using GridSearchCV
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto', 0.01, 0.1, 1],
    'kernel': ['linear', 'rbf', 'poly'],
    'degree': [2, 3, 4]
}

grid = GridSearchCV(svm.SVC(), param_grid, refit=True, verbose=2, cv=5)
grid.fit(X_train, y_train)

# Step 9: Make predictions with the best model
y_pred = grid.predict(X_test)

# Step 10: Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of SVM model: {accuracy * 100:.2f}%')

# Step 11: Generate confusion matrix and classification report
def plot_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title('Confusion Matrix')
    plt.show()

    report = classification_report(y_true, y_pred, target_names=['Real', 'Fake'])
    print("Classification Report:\n", report)

# Plot confusion matrix and print classification report
plot_confusion_matrix(y_test, y_pred)

# Step 12: Visualize some predictions with matplotlib chart
def plot_predictions(X_test, y_test, y_pred):
    plt.figure(figsize=(10, 5))
    for i in range(10): # Display the first 10 test images
        # Get the correct shape of each image
        image_shape = int(X_test[i].size ** 0.5)

        # Show each test image with predicted and actual labels
        plt.subplot(2, 5, i + 1)
        plt.imshow(X_test[i].reshape(image_shape, image_shape), cmap='gray')
        plt.title(f'True: {"Real" if y_test[i] == 0 else "Fake"}\nPred: {"Real" if y_pred[i] == 0 else "Fake"}')
        plt.axis('off')

def display_hog_features(images, num_samples=5):
    # Ensure num_samples does not exceed the number of images provided
    num_samples = min(num_samples, len(images))

    fig, axes = plt.subplots(num_samples, 2, figsize=(10, num_samples * 3))

    for i in range(num_samples):
        img = images[i]
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Extract HOG features and visualization
        hog_features, hog_image = hog(
            gray,
            pixels_per_cell=(16, 16),
            cells_per_block=(2, 2),
            orientations=9,
            visualize=True,
            block_norm='L2-Hys'
        )

        # Display original image
        axes[i, 0].imshow(gray, cmap='gray')
        axes[i, 0].set_title("Original Image")
        axes[i, 0].axis('off')

        # Adjust the HOG image for better contrast and display it
        hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
        axes[i, 1].imshow(hog_image_rescaled, cmap='gray')
        axes[i, 1].set_title("HOG Features")

```

```
axes[i, 1].axis('off')

plt.tight_layout()
plt.show()


# Display HOG features for the first 5 images in the dataset (you can change this number)
# Make sure you have enough images in X
display_hog_features(X, num_samples=5)
from sklearn.metrics import precision_recall_curve, auc

# Step 1: Get decision function scores to compute precision-recall for different thresholds
# Note: For GridSearchCV, we retrieve the best estimator from the grid and get its decision function
y_scores = grid.decision_function(X_test)

# Step 2: Compute precision, recall, and thresholds
precision, recall, thresholds = precision_recall_curve(y_test, y_scores)

# Step 3: Calculate the area under the precision-recall curve for reference
pr_auc = auc(recall, precision)

# Step 4: Plot Precision vs Recall
plt.figure(figsize=(10, 6))
plt.plot(recall, precision, label=f'Precision-Recall curve (area = {pr_auc:.2f})', color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision vs. Recall Curve using HOG Features')
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

 Please upload your real images.
 No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving person(1).png to person(1).png
 Saving person(2).png to person(2).png
 Saving person(3).png to person(3).png
 Saving person(4).png to person(4).png
 Saving person(5).png to person(5).png
 Saving person(6).png to person(6).png
 Saving person(7).png to person(7).png
 Saving person(8).png to person(8).png
 Saving person(9).png to person(9).png
 Saving person(10).png to person(10).png
 Saving person(11).png to person(11).png
 Saving person(12).png to person(12).png
 Saving person(13).png to person(13).png
 Saving person(14).png to person(14).png
 Saving person(15).png to person(15).png
 Saving person(16).png to person(16).png
 Saving person(17).png to person(17).png
 Saving person(18).png to person(18).png
 Please upload your fake images.

No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving person(51).jpg to person(51).jpg
 Saving person(52).jpg to person(52).jpg
 Saving person(53).jpg to person(53).jpg
 Saving person(54).jpg to person(54).jpg
 Saving person(55).jpg to person(55).jpg
 Saving person(56).jpg to person(56).jpg
 Saving person(57).jpg to person(57).jpg
 Saving person(58).jpg to person(58).jpg
 Saving person(59).jpg to person(59).jpg
 Saving person(60).jpg to person(60).jpg
 Saving person(61).jpg to person(61).jpg
 Saving person(62).jpg to person(62).jpg
 Saving person(63).jpg to person(63).jpg
 Saving person(64).jpg to person(64).jpg
 Saving person(65).jpg to person(65).jpg
 Saving person(66).jpg to person(66).jpg
 Saving person(67).jpg to person(67).jpg
 Saving person(68).jpg to person(68).jpg
 Fitting 5 folds for each of 180 candidates, totalling 900 fits

[CV] ENDC=0.1, degree=2, gamma=scale, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=rbf; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=rbf; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=rbf; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=rbf; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=rbf; total time=	0.3s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=scale, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=rbf; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=rbf; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=rbf; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=rbf; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=poly; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=poly; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=auto, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=rbf; total time=	0.5s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=rbf; total time=	0.5s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=rbf; total time=	0.5s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=rbf; total time=	0.5s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=rbf; total time=	0.5s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=0.01, kernel=poly; total time=	0.2s
[CV] ENDC=0.1, degree=2, gamma=0.1, kernel=linear; total time=	0.1s
[CV] ENDC=0.1, degree=2, gamma=0.1, kernel=linear; total time=	0.1s

<https://colab.research.google.com/drive/1VfPya2TSg-KGYXvY6THSwdHVMsjMWpt1?usp=sharing#printMode=true>

7/15

[illegible]

[illegible]

[illegible]

[illegible]

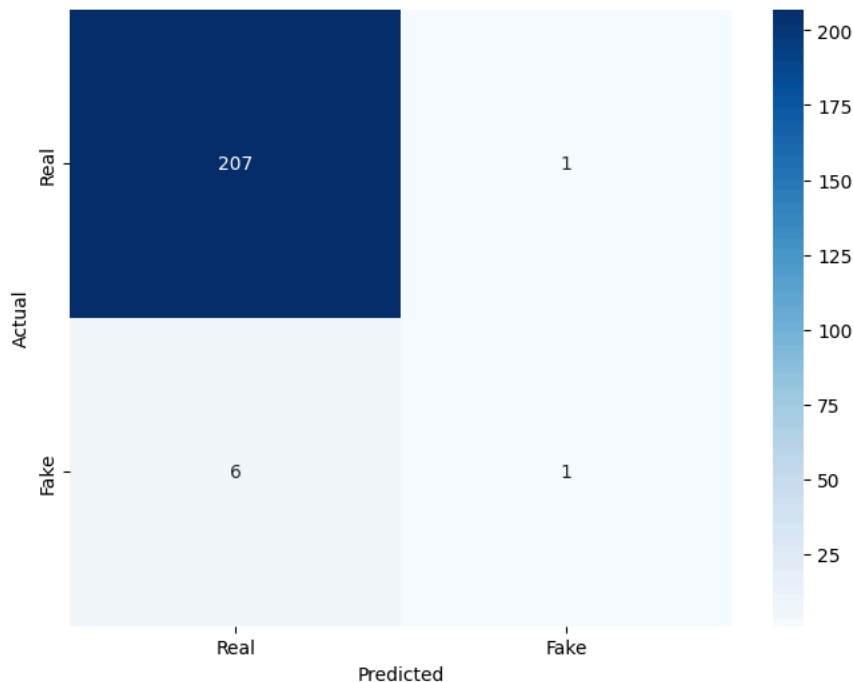
[illegible]

```

[CV] END .....C=100, degree=4, gamma=0.01, kernel=rbf; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=0.01, kernel=rbf; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=0.01, kernel=rbf; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=0.01, kernel=rbf; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=0.01, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=0.01, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=0.01, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=0.01, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=rbf; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=rbf; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=rbf; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=rbf; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=0.1, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=1, kernel=linear; total time= 0.1s
[CV] END .....C=100, degree=4, gamma=1, kernel=rbf; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=1, kernel=rbf; total time= 0.7s
[CV] END .....C=100, degree=4, gamma=1, kernel=rbf; total time= 0.6s
[CV] END .....C=100, degree=4, gamma=1, kernel=rbf; total time= 0.6s
[CV] END .....C=100, degree=4, gamma=1, kernel=rbf; total time= 0.6s
[CV] END .....C=100, degree=4, gamma=1, kernel=poly; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=1, kernel=poly; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=1, kernel=poly; total time= 0.5s
[CV] END .....C=100, degree=4, gamma=1, kernel=poly; total time= 0.4s
[CV] END .....C=100, degree=4, gamma=1, kernel=poly; total time= 0.4s
Accuracy of SVM model: 96.74%

```

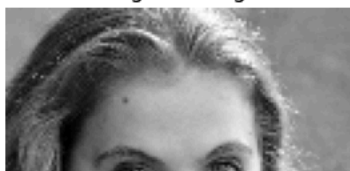
Confusion Matrix



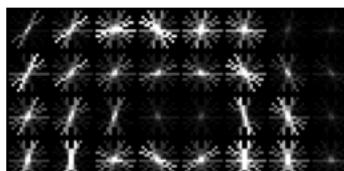
Classification Report:

	precision	recall	f1-score	support
Real	0.97	1.00	0.98	208
Fake	0.50	0.14	0.22	7
accuracy			0.97	215
macro avg	0.74	0.57	0.60	215
weighted avg	0.96	0.97	0.96	215

Original Image

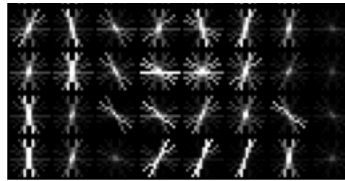


HOG Features





Original Image



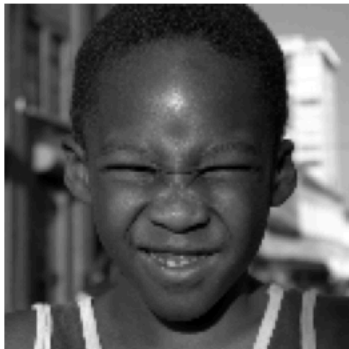
HOG Features



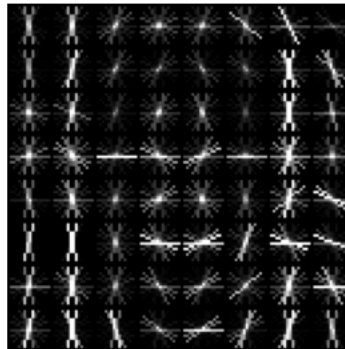
Original Image



HOG Features



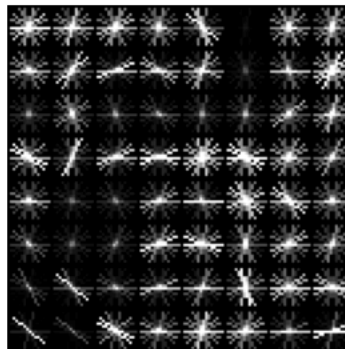
Original Image



HOG Features



Original Image



HOG Features

