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

```
Mounted at /content/drive
```

```
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import shutil
import seaborn as sns

from PIL import Image
%matplotlib inline
from collections import Counter
import cv2
import os
from skimage import color, feature, img_as_ubyte
from sklearn.metrics import classification_report


from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from skimage.feature import hog


import tensorflow as tf
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
from torch.utils.data import DataLoader,Dataset,TensorDataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input, decode_predictions
from joblib import load,dump
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from retinaface import RetinaFace
```

```
# Lab 09
GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = '/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG'
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
print(os.listdir(GOOGLE_DRIVE_PATH))
```

```
['Code', 'Video', 'Models', 'CV2023_CW_Dataset']
```

```
GOOGLE_DRIVE_PATH
```

```
'/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG'
```

```
training_data_path = os.path.join(GOOGLE_DRIVE_PATH, '/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/CV2023_CW
testing_data_path = os.path.join(GOOGLE_DRIVE_PATH, '/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/CV2023_CW_
```

```
if os.path.isdir(os.path.join(training_data_path,'all_classes_train')) is False:
  os.mkdir(os.path.join(training_data_path,'all_classes_train'))
if os.path.isdir(os.path.join(testing_data_path,'all_classes_test')) is False:
  os.mkdir(os.path.join(testing_data_path,'all_classes_test'))
```

```
train_dataset_path = os.path.join(training_data_path,'all_classes_train')
test_dataset_path = os.path.join(testing_data_path,'all_classes_test')
```

```
class_list = ['0', '1', '2']
for i in class_list:
  train_dir=train_dataset_path+'/'+i
  test_dir=test_dataset_path+'/'+i
  if os.path.isdir(train_dir) is False:
    os.mkdir(train_dir)
  if os.path.isdir(test_dir) is False:
    os.mkdir(test_dir)
```

```
#Lab 07,the way how to load the images and their respective labels
def import_selected_data(path):

    train_image_paths, train_labels, test_image_paths, test_labels = [], [], [], []

    for folder in os.listdir(path):
        if folder == "train":
            train_images_path = os.path.join(path, folder, "images")
            train_labels_path = os.path.join(path, folder, "labels")
            for file in sorted(os.listdir(train_images_path)):
                if file.endswith(".jpeg"):
                    train_image_paths.append(os.path.join(train_images_path, file))
                    label_path = os.path.join(train_labels_path, file.split(".")[0] + ".txt")
                    with open(label_path, "r") as f:
                        train_labels.append(f.read().strip())

        if folder == "test":
            test_images_path = os.path.join(path, folder, "images")
            test_labels_path = os.path.join(path, folder, "labels")
            for file in sorted(os.listdir(test_images_path)):
                if file.endswith(".jpeg"):
                    test_image_paths.append(os.path.join(test_images_path, file))
                    label_path = os.path.join(test_labels_path, file.split(".")[0] + ".txt")
                    with open(label_path, "r") as f:
                        test_labels.append(f.read().strip())

    return train_image_paths, train_labels, test_image_paths, test_labels
```

```
train_image_paths, train_labels, test_image_paths, test_labels = import_selected_data('/content/drive/MyDrive/Colab Notebooks/
```

```
#Lab 07
print("Number of training images: ", len(train_image_paths))
print("Number of training labels: ", len(train_labels))
print("Number of test images: ", len(test_image_paths))
print("Number of test labels: ", len(test_labels))
```

```
Number of training images:  2393
Number of training labels:  2393
Number of test images:  458
Number of test labels:  458
```

```
# Lab 07
print(Counter(train_labels))
```

```
Counter({'1': 1939, '0': 376, '2': 78})
```

```
for k in range(0,len(train_labels)):
    filename = train_image_paths[k].split('/')[len(train_image_paths[k].split('/'))-1]
    label = train_labels[k]
    start = os.path.abspath(train_image_paths[k])
    target = os.path.join(train_dataset_path, label, filename)
    shutil.copyfile(start, target)
```

```
for k in range(0,len(test_labels)):
    filename = test_image_paths[k].split('/')[len(test_image_paths[k].split('/'))-1]
    label = test_labels[k]
    start = os.path.abspath(test_image_paths[k])
    target = os.path.join(test_dataset_path, label, filename)
    shutil.copyfile(start, target)
```

```
def bgr_to_rgb(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Define the data generator with data augmentation
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=bgr_to_rgb, validation_split=0.2)
```

```
size = 100
# Split the training data into a training set and a validation set
train_batches = datagen.flow_from_directory(directory=train_dataset_path,
                                            target_size=(size,size),
                                            batch_size=5,
                                            shuffle=True,

                                             )
```

```
# The validation set can be accessed like this:
valid_batches = datagen.flow_from_directory(directory=train_dataset_path,
                                            target_size=(size,size),
                                            batch_size=5,
                                            shuffle=True,
                                            subset="validation",
                                             )

test_batches =datagen.flow_from_directory(test_dataset_path,
                                           target_size=(size,size),
                                           batch_size=1,
                                           shuffle=False,
                                           )
```

```
Found 2393 images belonging to 3 classes.
Found 477 images belonging to 3 classes.
Found 458 images belonging to 3 classes.
```

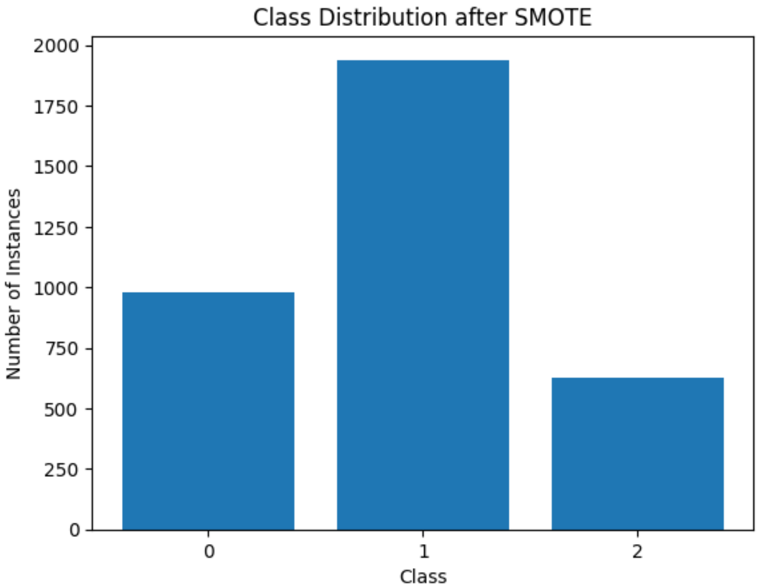
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import OneHotEncoder
from imblearn.over_sampling import SMOTE

# Convert directory iterator to arrays
def get_data_from_iterator(iterator, num_samples):
    images, labels = [], []
    for batch in iterator:
        images.extend(batch[0])
        labels.extend(batch[1])
        if len(images) >= num_samples:
            break
    return np.array(images), np.array(labels)

num_train_samples = train_batches.n
num_valid_samples = valid_batches.n

train_X, train_y = get_data_from_iterator(train_batches, num_train_samples)
valid_X, valid_y = get_data_from_iterator(valid_batches, num_valid_samples)

# Get the number of samples in each class
num_samples_class0 = np.sum(np.argmax(train_y, axis=1) == 0)
num_samples_class1 = np.sum(np.argmax(train_y, axis=1) == 1)
num_samples_class2 = np.sum(np.argmax(train_y, axis=1) == 2)

# Define the desired number of samples for each class after SMOTE
sampling_strategy = {
    0: int(num_samples_class0 * 2.6),
····2:·int(num_samples_class2·*·8),
}

# Apply SMOTE
smote = SMOTE(sampling_strategy=sampling_strategy)
train_X_resampled, train_y_resampled = smote.fit_resample(train_X.reshape(train_X.shape[0], -1), np.argmax(train_y, axis=1))
valid_X_resampled, valid_y_resampled = smote.fit_resample(valid_X.reshape(valid_X.shape[0], -1), np.argmax(valid_y, axis=1))

# Reshape the data back to the original shape
train_X_resampled = train_X_resampled.reshape(train_X_resampled.shape[0], train_X.shape[1], train_X.shape[2], train_X.shape[3]
valid_X_resampled = valid_X_resampled.reshape(valid_X_resampled.shape[0], valid_X.shape[1], valid_X.shape[2], valid_X.shape[3]

# One-hot encode the labels
encoder = OneHotEncoder(sparse=False)
train_y_resampled = encoder.fit_transform(train_y_resampled.reshape(-1, 1))
valid_y_resampled = encoder.transform(valid_y_resampled.reshape(-1, 1))

# Create new directory iterators with balanced data
datagen = ImageDataGenerator()

train_batches = datagen.flow(train_X_resampled, train_y_resampled, batch_size=5, shuffle=True)
valid_batches = datagen.flow(valid_X_resampled, valid_y_resampled, batch_size=5, shuffle=True)
```

```
/usr/local/lib/python3.9/dist-packages/imblearn/utils/_validation.py:313: UserWarning: After over-sampling, the number of
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/imblearn/utils/_validation.py:313: UserWarning: After over-sampling, the number of
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sp
  warnings.warn(
```

```
original_class_counts = Counter(np.argmax(train_y, axis=1))
unique, counts = np.unique(train_y_resampled.argmax(axis=1), return_counts=True)
```

```
resampled_class_counts = dict(zip(unique, counts))

# Print the counters
print("Original class counts:", original_class_counts)
print("Resampled class counts:", resampled_class_counts)

# Create the bar plot
plt.bar(resampled_class_counts.keys(), resampled_class_counts.values())
plt.xlabel('Class')
plt.ylabel('Number of Instances')
plt.title('Class Distribution after SMOTE')
plt.xticks(list(resampled_class_counts.keys()))
plt.show()
```

```
Original class counts: Counter({1: 1939, 0: 376, 2: 78})
Resampled class counts: {0: 977, 1: 1939, 2: 624}
```



CNN

```
# Define the model
model = models.Sequential([
    layers.Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(100, 100, 3)),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(512, activation='LeakyReLU', kernel_regularizer=l2(0.001)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(256, activation='LeakyReLU', kernel_regularizer=l2(0.001)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
```

```
    layers.Dense(3, activation='softmax')
])
```

```
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, restore_best_weights=True)

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(train_batches,callbacks=[early_stopping], validation_data = valid_batches,epochs=100)
```

```
Epoch 1/100
479/479 [==============================] - 33s 31ms/step - loss: 2.3969 - accuracy: 0.7225 - val_loss: 1.6400 - val_accu
Epoch 2/100
479/479 [==============================] - 13s 28ms/step - loss: 1.5630 - accuracy: 0.8491 - val_loss: 1.1053 - val_accu
Epoch 3/100
479/479 [==============================] - 13s 28ms/step - loss: 1.1091 - accuracy: 0.8692 - val_loss: 0.9403 - val_accu
Epoch 4/100
479/479 [==============================] - 14s 28ms/step - loss: 0.8975 - accuracy: 0.8763 - val_loss: 1.0124 - val_accu
Epoch 5/100
479/479 [==============================] - 14s 28ms/step - loss: 0.8411 - accuracy: 0.8897 - val_loss: 0.6602 - val_accu
Epoch 6/100
479/479 [==============================] - 14s 29ms/step - loss: 0.7892 - accuracy: 0.8838 - val_loss: 0.7727 - val_accu
Epoch 7/100
479/479 [==============================] - 14s 29ms/step - loss: 0.7393 - accuracy: 0.8930 - val_loss: 0.7034 - val_accu
Epoch 8/100
479/479 [==============================] - 14s 29ms/step - loss: 0.7833 - accuracy: 0.8792 - val_loss: 0.6682 - val_accu
Epoch 9/100
479/479 [==============================] - 14s 29ms/step - loss: 0.7803 - accuracy: 0.8796 - val_loss: 0.8861 - val_accu
Epoch 10/100
479/479 [==============================] - 14s 29ms/step - loss: 0.7275 - accuracy: 0.8751 - val_loss: 0.7880 - val_accu
Epoch 11/100
479/479 [==============================] - 14s 29ms/step - loss: 0.7321 - accuracy: 0.8918 - val_loss: 0.9349 - val_accu
Epoch 12/100
479/479 [==============================] - 14s 29ms/step - loss: 0.7603 - accuracy: 0.8817 - val_loss: 0.7149 - val_accu
Epoch 13/100
479/479 [==============================] - 14s 30ms/step - loss: 0.7188 - accuracy: 0.8897 - val_loss: 0.7637 - val_accu
Epoch 14/100
479/479 [==============================] - 14s 29ms/step - loss: 0.7310 - accuracy: 0.8872 - val_loss: 0.6936 - val_accu
Epoch 15/100
479/479 [==============================] - 14s 29ms/step - loss: 0.7285 - accuracy: 0.8842 - val_loss: 0.7102 - val_accu
```

```
prediction_cnn=model.evaluate(test_batches)
```

```
458/458 [==============================] - 2s 5ms/step - loss: 0.6978 - accuracy: 0.9214
```

```
predictions=model.predict(test_batches)
```

```
458/458 [==============================] - 2s 4ms/step
```

```
# Obtain classification report
y_pred=np.argmax(predictions,axis=1)
targetnames=['0','1','2']
y_true=test_batches.classes
y_true=to_categorical(y_true)
y_pred=to_categorical(y_pred)
report = classification_report(y_true,y_pred,target_names=targetnames)
print("Classification report:\n", report)
```

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

           0       0.83      0.88      0.86        51
           1       0.94      0.97      0.96       388
           2       0.20      0.05      0.08        19

   micro avg       0.92      0.92      0.92       458
   macro avg       0.66      0.63      0.63       458
weighted avg       0.90      0.92      0.91       458
 samples avg       0.92      0.92      0.92       458
```

```
from joblib import load,dump
```

```
dump(model,'cnn_final_model.joblib')
```

    ['cnn_final_model.joblib']

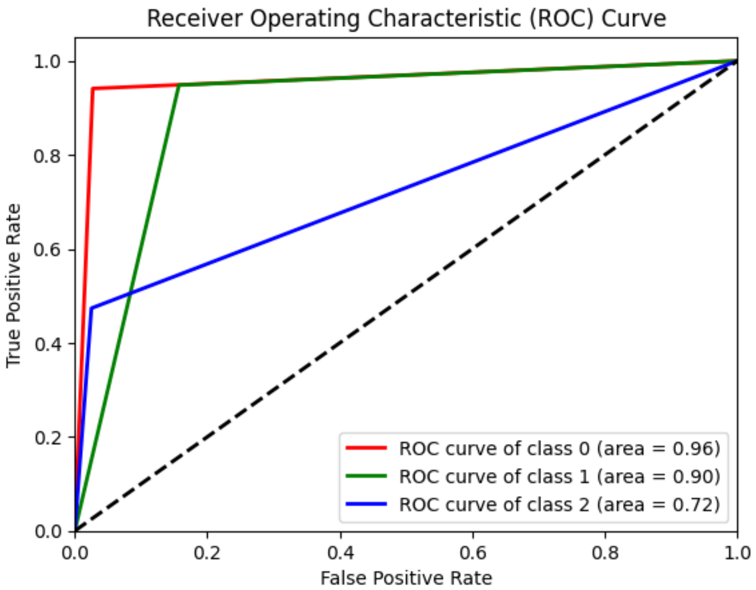
```
dump(model,'/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Models/cnn_final_model.joblib')
```

    ['/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Models/cnn_final_model.joblib']

```
cnn_loaded1 = load('/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Models/cnn_final_model.joblib')
```

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

# Make predictions on the test set
test_predictions = cnn_loaded1.predict(test_batches)

# Convert y_true and test_predictions to binary form
y_true_binary = label_binarize(test_batches.classes, classes=[0, 1, 2])
test_predictions_binary = label_binarize(np.argmax(test_predictions, axis=1), classes=[0, 1, 2])

# Compute ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(3):
    fpr[i], tpr[i], _ = roc_curve(y_true_binary[:, i], test_predictions_binary[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve
plt.figure()
lw = 2
colors = ['red', 'green', 'blue']
for i, color in zip(range(3), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], color='black', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

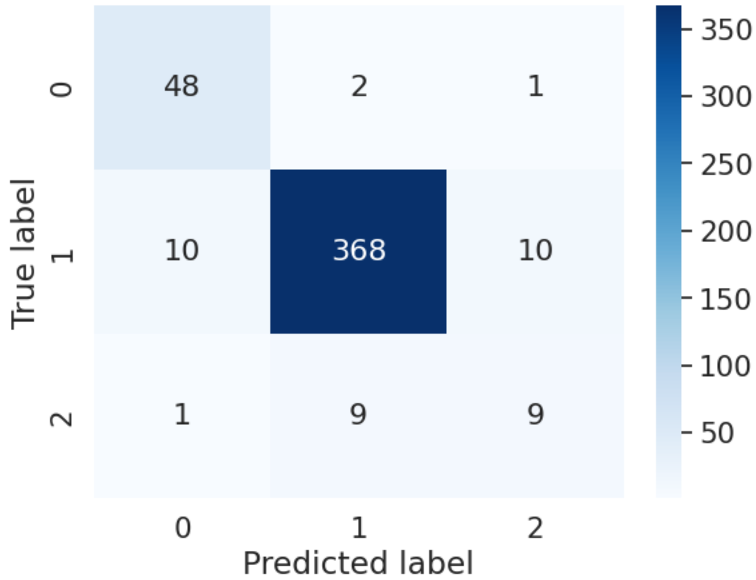
    458/458 [==============================] - 6s 12ms/step




```
# Create confusion matrix
y_true = np.argmax(y_true_binary, axis=1)
y_pred = np.argmax(test_predictions_binary, axis=1)
conf_matrix = confusion_matrix(y_true, y_pred)
print("Confusion matrix:\n", conf_matrix)
```

```
print( Confusion matrix:\n , conf_matrix)

     Confusion matrix:
      [[ 48   2   1]
      [ 10 368  10]
      [  1   9   9]]
```

```
# Plot confusion matrix as heatmap
sns.set(font_scale=1.4)
sns.heatmap(conf_matrix, annot=True, annot_kws={"size": 16}, cmap='Blues', fmt='g')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```



HOG+SVM

```
def extract_hog_features(image):
    features,image = hog(image, orientations=8, pixels_per_cell=(16,16), cells_per_block=(1, 1), multichannel =True,visualize=
    return features,image
```

the HOG (Histogram of Oriented Gradients) feature extraction technique to convert each image into a set of features that can be used for training the model.

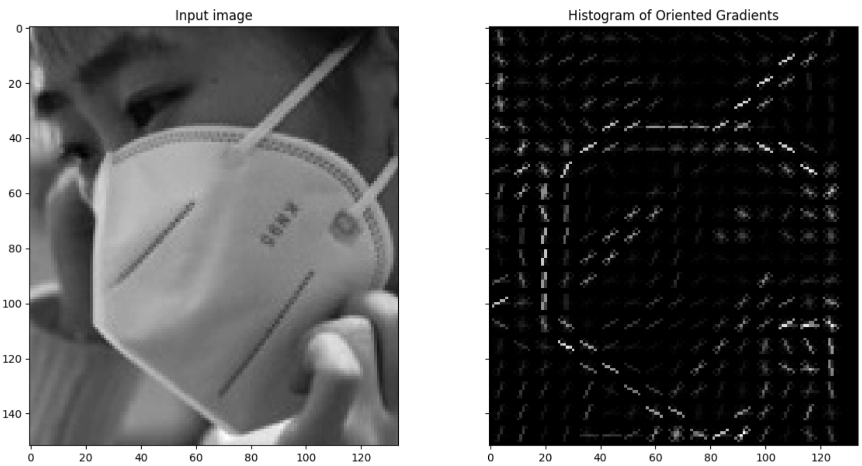
```
size = 100
train_images = []
train_images_img = []
train_labels = []
test_images = []
test_labels = []
train_images_orig=[]
counter = 0

# Training data
for imgs, labels in train_batches:
    for img, label in zip(imgs, labels):
        train_images_orig.append(img)
        img = cv2.resize(img,(size,size))
        features,image = extract_hog_features(img)
        train_images_img.append(image)
        train_images.append(features)
        train_labels.append(np.argmax(label))
        counter += 1
    if counter >= train_batches.n:
        break

counter = 0

# Testing data
for imgs, labels in test_batches:
    for img, label in zip(imgs, labels):
        img = cv2.resize(img,(size,size))
        features,image = extract_hog_features(img)
```

```
        test_images.append(features)
        test_labels.append(np.argmax(label))
        counter += 1
    if counter >= test_batches.n:
        break
```

```
 <ipython-input-58-6017f9a41c04>:2: FutureWarning: `multichannel` is a deprecated argument name for `hog`. It will be remo
   features,image = hog(image, orientations=8, pixels_per_cell=(16,16), cells_per_block=(1, 1), multichannel =True,visuali
```

```
# List files in the directory
image_directory = '/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/CV2023_CW_Dataset/train/images'
image_files = os.listdir(image_directory)

# Load the first image in the directory
image_file_path = os.path.join(image_directory, image_files[700])
image = cv2.imread(image_file_path)

if image is None:
    print("Error: Image not loaded. Check the file path.")
else:
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Compute HOG features
    fd, hog_image = hog(gray_image, orientations=9, pixels_per_cell=(8, 8),
                        cells_per_block=(2, 2), visualize=True, multichannel=False)

    # Normalize the HOG image for better visualization
    hog_image_rescaled = (hog_image - np.min(hog_image)) / (np.max(hog_image) - np.min(hog_image))

    # Visualize the first image and its HOG features
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 7), sharex=True, sharey=True)

    ax1.imshow(gray_image, cmap=plt.cm.gray)
    ax1.set_title('Input image')

    ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
    ax2.set_title('Histogram of Oriented Gradients')

    plt.show()
```

```
 <ipython-input-60-026191c73f59>:15: FutureWarning: `multichannel` is a depreca
   fd, hog_image = hog(gray_image, orientations=9, pixels_per_cell=(8, 8),
```




```
from sklearn.model_selection import GridSearchCV
from sklearn import svm

# Set up the parameter grid
param_grid = {
    'C': [0.00001,0.1, 1, 10, 100],
```

```
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': [1, 0.1, 0.01, 0.001]
}

# Initialize the SVM classifier
clf = svm.SVC(random_state=42)

# Create the GridSearchCV object
grid_search = GridSearchCV(clf, param_grid, cv=2)

# Fit the GridSearchCV object to the data
grid_search.fit(train_images, train_labels)

# Get the best parameters
best_params = grid_search.best_params_
print("Best parameters found: ", best_params)

# Train the SVM classifier with the best parameters
best_clf = svm.SVC(**best_params, random_state=42)
best_clf.fit(train_images, train_labels)
```

```
Best parameters found:  {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
▾                 SVC
SVC(C=100, gamma=0.1, random_state=42)
```

```
from sklearn.metrics import accuracy_score, classification_report

predicted_labels = best_clf.predict(test_images)
# Calculate accuracy
accuracy = accuracy_score(test_labels, predicted_labels)
print("Accuracy: {:.2f}%".format(accuracy * 100))
# Obtain classification report
report = classification_report(test_labels, predicted_labels)
print("Classification report:\n", report)
```

```
Accuracy: 87.55%
Classification report:
               precision    recall  f1-score   support

           0       0.57      0.53      0.55        51
           1       0.91      0.95      0.93       388
           2       0.71      0.26      0.38        19

    accuracy                           0.88       458
   macro avg       0.73      0.58      0.62       458
weighted avg       0.87      0.88      0.87       458
```

```
dump(best_clf,'/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Models/svm_final_model.joblib')
```

```
['/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Models/svm_final_model.joblib']
```

```
svm_loaded=load('/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Models/svm_final_model.joblib')
```

HOG+MLP

```
def extract_hog_features(image):
    features,image = hog(image, orientations=8, pixels_per_cell=(16,16), cells_per_block=(1, 1), multichannel =True,visualize=
    return features, image

def convert_to_rgb(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
size = 100
train_images_hog = []
train_images_img = []
train_labels_hog= []
test_images_hog = []
test_labels_hog = []
train_images_orig=[]
counter = 0

# Training data
for imgs, labels in train_batches:
    for img, label in zip(imgs, labels):
```

```
            train_images_orig.append(img)
            img = cv2.resize(img,(size,size))
            features,image = extract_hog_features(img)
            train_images_img.append(image)
            train_images_hog.append(features)
            train_labels_hog.append(np.argmax(label))
            counter += 1
        if counter >= train_batches.n:
            break

    counter = 0

    # Testing data
    for imgs, labels in test_batches:
        for img, label in zip(imgs, labels):
            img = cv2.resize(img,(size,size))
            features,image = extract_hog_features(img)
            test_images_hog.append(features)
            test_labels_hog.append(np.argmax(label))
            counter += 1
        if counter >= test_batches.n:
            break
```

```
<ipython-input-78-0f5790cc7810>:2: FutureWarning: `multichannel` is a deprecated argument name for `hog`. It will be remo
  features,image = hog(image, orientations=8, pixels_per_cell=(16,16), cells_per_block=(1, 1), multichannel =True,visuali
```

```
# Lab 07
# Initialize the MLP classifier
mlp = MLPClassifier(random_state=42)

# Create the parameter grid
param_grid_mlp = {
    'hidden_layer_sizes': [(100,), (200,), (300,), (100, 100), (200, 100), (100, 100, 100), (200, 100, 100)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'sgd'],
    'alpha': [0.0001, 0.001, 0.01],
}

# Create the GridSearchCV object
grid_search_mlp = GridSearchCV(mlp, param_grid_mlp, cv=2)

# Fit the GridSearchCV object to the data
grid_search_mlp.fit(train_images_hog, train_labels_hog)

# Get the best parameters
best_params_mlp = grid_search_mlp.best_params_
print("Best parameters found: ", best_params_mlp)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochast
  warnings.warn(
Best parameters found:  {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (200, 100), 'solver': 'adam'}
```

```
# Initialize MLP classifier with best hyperparameters
mlp = MLPClassifier(hidden_layer_sizes=best_params_mlp['hidden_layer_sizes'],
                    activation=best_params_mlp['activation'],
                    solver=best_params_mlp['solver'],
                    alpha=best_params_mlp['alpha'],
                    random_state=42)

mlp.fit(train_images_hog, train_labels)

predicted_labels = mlp.predict(test_images_hog)

accuracy = accuracy_score(test_labels, predicted_labels)
print("Accuracy: {:.2f}%".format(accuracy * 100))
# Obtain classification report
report = classification_report(test_labels, predicted_labels)
print("Classification report:\n", report)
```

```
Accuracy: 43.89%
Classification report:
               precision    recall  f1-score   support

           0       0.09      0.27      0.14        51
           1       0.82      0.48      0.61       388
           2       0.00      0.00      0.00        19

    accuracy                           0.44       458
   macro avg       0.30      0.25      0.25       458
weighted avg       0.71      0.44      0.53       458
```

```
dump(mlp,'/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Models/mlp_final_model.joblib')
```

```
['/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Models/mlp_final_model.joblib']
```

```
!pip install retinaface --no-deps
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: retinaface in /usr/local/lib/python3.9/dist-packages (1.1.1)
```

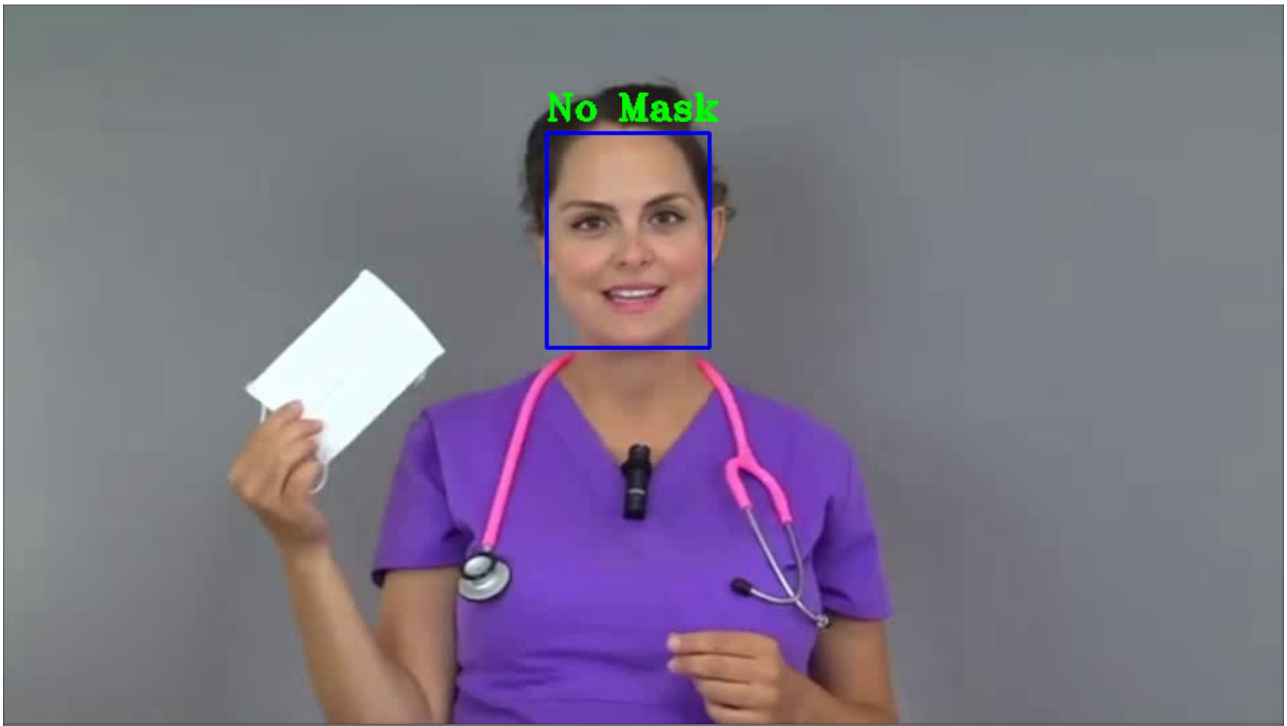
VIDEO

```
# https://medium.com/@stepanfilonov/tracking-your-eyes-with-python-3952e66194a6, TO UNDERSTND BETTER HOT TO IMPLEMNET FOR TH E
#https://www.google.com/search?q=RETINA+LIBRARY+VIDEO+PYTHON+MDEIUM&rlz=1C5CHFA_enGB1052GB1052&ei=ijJFZJ7KMZ-YhbIPoJSimAs&ved=
video_path = '/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Video/myvideo.mp4'
output_video_path = '/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Video/myvideo_output.mp4'

retinaface = RetinaFace()

cap = cv2.VideoCapture(video_path)

fps = int(cap.get(cv2.CAP_PROP_FPS))
frame_size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)), int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))

fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter(output_video_path, fourcc, fps, frame_size)
while True:
    ret, frame = cap.read()
```

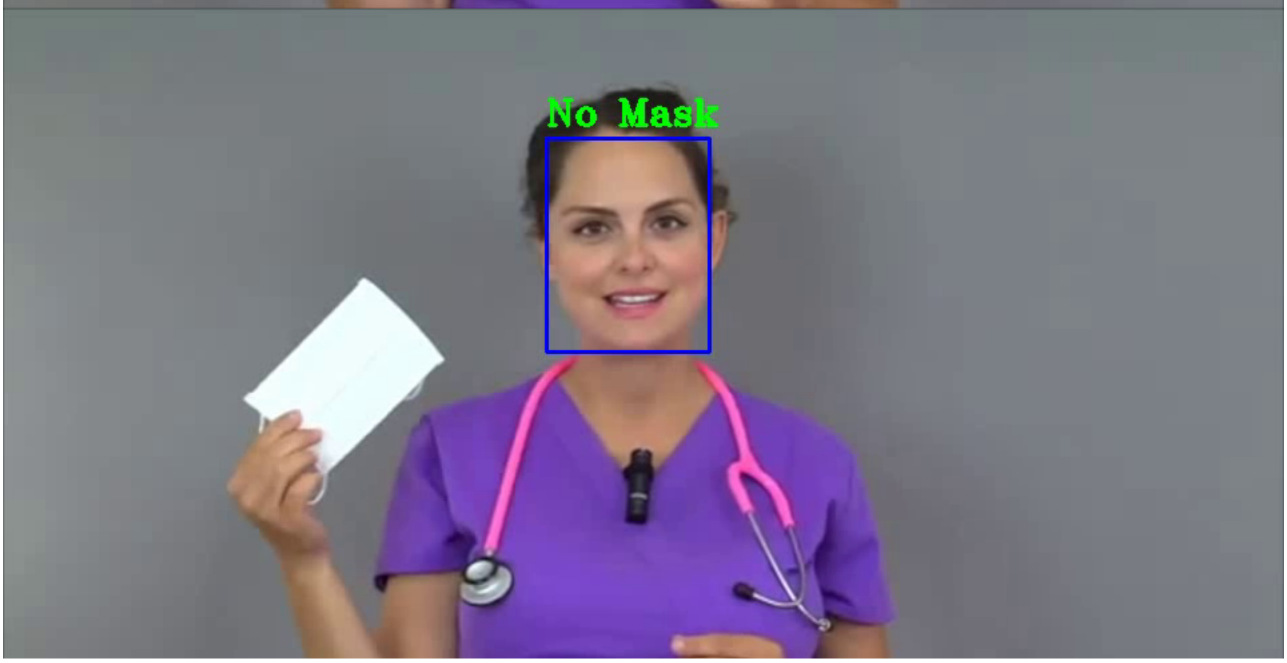
```
    if not ret:
        break
    faces = retinaface.predict(frame)
    if faces is not None:
        for face in faces:
            x, y, x2, y2 = face['x1'],face['y1'],face['x2'],face['y2']
            face_roi = frame[y:y2, x:x2]
            try:
                face_roi = cv2.resize(face_roi, (100, 100))
            except:
                continue
            face_roi = np.expand_dims(face_roi, axis=0)
            video_output = cnn_loaded.predict(face_roi, steps=1,verbose=0)
            video_output = np.argmax(video_output[0])
            if(video_output==0):
                cv2.rectangle(frame, (x, y), (x2, y2), (255, 0, 0), 2)
                cv2.putText(frame, "No Mask", (x, y - 10), cv2.FONT_HERSHEY_PLAIN, 0.9, (0, 255, 0), 2)
            elif(video_output==1):
                cv2.rectangle(frame, (x, y), (x2, y2), (0, 255, 0), 2)
                cv2.putText(frame, "Proper Mask", (x, y - 10), cv2.FONT_HERSHEY_PLAIN, 0.9, (0, 255, 0), 2)
            else:
                cv2.rectangle(frame, (x, y), (x2, y2), (0, 0, 255), 2)
                cv2.putText(frame, "Improper Mask", (x, y - 10), cv2.FONT_HERSHEY_PLAIN, 0.9, (0, 255, 0), 2)
    out.write(frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
out.release()
cv2.destroyAllWindows()
print("Output video saved successfully.")
```

```
model[normal quality] init ..
model success !
Output video saved successfully.
```

```
os.environ['DISPLAY'] = ':0'
os.environ['PYVISTA_OFF_SCREEN'] = 'true'
output_video_path = '/content/drive/MyDrive/Colab Notebooks/CV 2023 CW/CW_Folder_PG/Video/myvideo_output.mp4'
out_cap = cv2.VideoCapture(output_video_path)

while True:
    # Read the current frame
    ret, frame = out_cap.read()
    if not ret:
        break
   cv2_imshow(frame)
    key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
        break

out_cap.release()
cv2.destroyAllWindows()
```


No Mask


No Mask


No Mask