# Training the HOG- classification models: Cw/training_HOG_models

```
from google.colab import drive
drive.mount('/content/drive')
import os
GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/computerVision/Cw'
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
print(os.listdir(GOOGLE_DRIVE_PATH))


#code adapted from lab 7
# Identify path to zipped dataset
zip_path = os.path.join(GOOGLE_DRIVE_PATH, 'CW_Dataset.zip')
# Copy it to Colab
!cp '{zip_path}' .

# Unzip it
!yes|unzip -q CW_Dataset.zip

# Delete zipped version from Colab (not from Drive)
!rm CW_Dataset.zip


#import messesary libraries
import cv2
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from skimage import img_as_ubyte, io, color
from sklearn import svm, metrics
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
from joblib import dump, load
from skimage.feature import hog
from skimage import data, exposure

%matplotlib inline

# function to load and return images data and labels

def import_dataset(images_path, label_path):

  images_list = []  # empty images list

  labels_list = [] # empty label list

  #read all the image files

  for file in sorted(os.listdir(images_path)):
```

```
    if file.endswith('.jpg'):
      image = io.imread(os.path.join(images_path,file))     #read the image
      images_list.append(image)    #add it to the images list

  #get the labels for each image from the txt file
  data = np.genfromtxt(label_path)    #txt file to np
  labels = data[:,1]
  labels = labels.astype(int)    #convert label to int
  labels_list = labels.tolist()   #add to the label list

  return images_list, labels_list

#import training images and corresponding labels from the given files
train_images, train_labels = import_dataset('train','labels/list_label_train.txt')

#display 10 images of the train set
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 5), sharex=True, sharey=True)
ax = axes.ravel()
for i in range(10):
    ax[i].imshow(train_images[i], cmap='gray')
    ax[i].set_title(f'Label: {train_labels[i]}')
    ax[i].set_axis_off()
fig.tight_layout()
plt.show()

#show number of different label classes
unique, counts = np.unique(train_labels, return_counts=True)

print(np.asarray((unique, counts)))

print(np.array(train_images).dtype)
```

```
print(np.array(train_labels).dtype)

### HOG feature descriptors extraction

# adapted from lab 6
#a fuction to return extracted the HOG features from the images and their label
def extract_HOG_featuresDes(images , labels):
  HOG_descriptors=[]
  HOG_images= []
  label_list=[]

  for i in range(len(images)):
    image = images[i]
    HOG_des, HOG_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                cells_per_block=(1, 1), visualize=True, multichannel=True)
    if HOG_des is not None:
      HOG_descriptors.append(HOG_des)
      HOG_images.append(HOG_image)
      label_list.append(labels[i])

  return HOG_descriptors, HOG_images, label_list

# extracts the hog features of the train dataset
HOG_descriptors, HOG_images, labels = extract_HOG_featuresDes(train_images,train_labels)

  print(np.array(HOG_descriptors).shape)
  print(len(labels))
# plot 10 HOG images
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 5), sharex=True, sharey=True)
ax = axes.ravel()
```

```
for i in range(10):

  HOG_image_rescaled = exposure.rescale_intensity(HOG_images[i], in_range=(0, 10))
  ax[i].imshow(HOG_image_rescaled, cmap='gray')
  ax[i].set_title(f'Label: {labels[i]}')
  ax[i].set_axis_off()

fig.tight_layout()
plt.show()



##assinging the X_test and y_test
X_train = HOG_descriptors
y_train = labels

## Loading the Test dataset and extracting HOG features

# Load the test testdata
test_images, test_labels = import_dataset('test','labels/list_label_test.txt')

# extracts the hog features of the test dataset
test_HOG_descriptors, test_HOG_images, test_labels =
extract_HOG_featuresDes(test_images,test_labels)

# HOG-SVM

## Training the classifier

# shuffle the dataset to prevent biases,
# by preventing the moddel from learning the order of the training
X_train , y_train = shuffle(X_train, y_train, random_state=42)
```

```
# SVM classifier with HOG features descriptiors
svm_classifier= svm.SVC( kernel='rbf')
svm_classifier.fit(X_train,y_train)
```

## Evaluating the trained SVM classifier

```
X_test = test_HOG_descriptors
y_test = test_labels

#predicting the test set on HOG- SVM classifier
svm_predict = svm_classifier.predict(X_test).tolist()

#printing the classifier scores
print(f"""Classification report for classifier {svm_classifier}:
    {metrics.classification_report(y_test, svm_predict)}\n""")

#displaying the confusion metric of the classifier
#adapted from lab 7
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(svm_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("HOG-SVM Confusion Matrix")
plt.show()
```

## Improving the SVM classifier

Hyper Parametering tuning with Grid search will be used to improve the classifer by finding the best optimal parameters

### Hyper-Parameter tuning

```
# SVM Hyper parameter tuning using GridSearch to find the best optimal Hyper-parameter
```

```
#adapted from https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/

from sklearn.model_selection import GridSearchCV


# defining parameter range
svm_param_grid = {'C': [0.1, 1, 10],
            'gamma': [1, 0.5, 0.1, 0.01],
            'kernel': ['rbf','poly']}
#classifier
svm_grid_classifier = GridSearchCV(svm.SVC(), svm_param_grid, refit = True,cv = 5, verbose = 4)

# traning the classifier with grid parameters
svm_grid_classifier.fit(X_train, y_train)

# print best parameter after tuning
print(svm_grid_classifier.best_params_)
### Evaluating the tuned HOG-SVM

# predicting the test set with tuned SVM
svm_grid_predict = svm_grid_classifier.predict(X_test)

print(f"""Classification report for classifier {svm_grid_classifier}:
    {metrics.classification_report(y_test, svm_grid_predict)}\n""")

# confusion metrix of HOG-SVM classifier
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(svm_grid_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("Confusion Matrix of tuned HOG-SVM classifier")
plt.show()
```

## Save HOG-SVM classifier

the trained classifier model is saved on the drive with the help of joblib library

```
#save the classifier
from joblib import dump
path = os.path.join('drive', 'My Drive', 'Colab Notebooks/computerVision/Cw/HOG_SVM.joblib' )
dump(svm_grid_classifier, path)
```

# HOG-MLP

## Training MLP classifier

```
####### HOG MLP classification ##############
#adapted from lab 6
from sklearn.neural_network import MLPClassifier

# MLP, Multi-Layer Perceptron, classifier
mlp_classifier = MLPClassifier(hidden_layer_sizes=(50,), max_iter=100, alpha=1e-4,
                solver='sgd', verbose=True, random_state=1,
                learning_rate_init=.1)

#fit the train set to the classifier
mlp_classifier.fit(X_train, y_train)
```

## Evaluating the trained MLP classifier

```
# pridict the test set
mlp_predicted= mlp_classifier.predict(X_test)

# printing the score of the MLP classifier
print(f"""Classification report for classifier {mlp_classifier}:\n
    {metrics.classification_report(y_test, mlp_predicted)}""")
```

```
# confusion metrix of HOG-MLP classifier

fig, ax = plt.subplots(figsize=(8,8))

disp = metrics.plot_confusion_matrix(mlp_classifier, X_test, y_test, values_format = 'd', ax=ax)

disp.figure_.suptitle("Confusion Matrix of HOG-MLP classifier")

plt.show()
```

## Improving the MLP classifier

### Hyper-Parameter tuning

```
# adapted from https://datascience.stackexchange.com/questions/36049/how-to-adjust-the-hyperparameters-of-mlp-classifier-to-get-more-perfect-performa

#parameters

mlp_param_grid = {

    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],

    'activation': ['tanh', 'relu'],

    'solver': ['sgd', 'adam'],

    'alpha': [0.0001, 0.05],

    'learning_rate': ['adaptive']}



#classifier

mlp_grid_classifier = GridSearchCV(MLPClassifier(max_iter=100), mlp_param_grid, refit = True, verbose = 3)


# fitting the classifier for optimal hyper parameter

mlp_grid_classifier.fit(X_train, y_train)


# print best parameter after tuning

print(mlp_grid_classifier.best_params_)
```

```
# # print how our the classifier looks after hyper-parameter tuning
# print(mlp_grid_classifier.best_estimator_)

### Evaluating the tuned HOG-MLP

# pridict the test set
mlp_grid_predicted= mlp_grid_classifier.predict(X_test)

# printing the score of the MLP classifier
print(f"""Classification report for classifier {mlp_grid_classifier}:\n
    {metrics.classification_report(y_test, mlp_grid_predicted)}""")

# confusion metrix of HOG-MLP classifier after grid search
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(mlp_grid_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("Confusion Matrix of tuned HOG-MLP classifier")
plt.show()

## Save HOG-MLP

#save the classifier
path = os.path.join('drive', 'My Drive', 'Colab Notebooks/computerVision/Cw/HOG_MLP.joblib' )
dump(mlp_grid_classifier, path)

# HOG-Random Forest Classifier

## Training the Random Forest classifier

########### HOG-RANDOM FOREST CLASSIFIER ################
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier()
```

```
#Training the classifier with HOG features descriptors
rf_classifier.fit(X_train,y_train)

## Evaluating the trained Random Forest classifier

#predicting the test set
rf_predicted = rf_classifier.predict(X_test)

#eevaluating random forest classifier
print(f"""Classification report for classifier {rf_classifier}:
     {metrics.classification_report(y_test, rf_predicted)}\n""")

# confusion metrix of  classifier
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(rf_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("Confusion Matrix of HOG-Random forest classifier")
plt.show()

## Improving the SVM classifier

### Hyper-Parameter tuning

# Hyper parameter tuning for random forest classifier using grid search
rf_grid_params = {
    'n_estimators': [10,50,100, 150, 250],
    'min_samples_split': [2, 4, 6]
}

# finding the best hyper parameter
rf_grid_classifier = GridSearchCV(RandomForestClassifier(), rf_grid_params, cv=5, verbose=5)
```

```
rf_grid_classifier.fit(X_train, y_train)
# print best parameter after tuning
print(rf_grid_classifier.best_params_)

### Evaluating the tuned Random Forest Classifier

#predicting of test set
rf_grid_predicted = rf_grid_classifier.predict(X_test)

#printing the score of tuned random forest classifier
print(f"""Classification report for classifier {rf_grid_classifier}:
    {metrics.classification_report(y_test, rf_grid_predicted)}\n""")

# confusion metrix of the classifier
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(rf_grid_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("Confusion Matrix of tuned HOG-Random forest classifier")
plt.show()

## Save the HOG-Random Forest Classifier

#save the classifier
path = os.path.join('drive', 'My Drive', 'Colab Notebooks/computerVision/Cw/HOG_RFC.joblib' )
dump(rf_grid_classifier, path)
print(svm_grid_classifier.best_estimator_)
```

# Training SIFT_models: Cw/training_SIFT_models

# Google colab setup

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

import os
GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/computerVision/Cw'
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
print(os.listdir(GOOGLE_DRIVE_PATH))

!pip install opencv-python==4.4.0.46
```

# Introduction

this file contains all the classification models trainied with the SIFT feature descriptors.

the process involves :

1. unzipping the files
1. loading the dataset
1. extracting SIFT feature descriptors of the dataset
1. clustering
2. histogram of codewords
2. training the classifier
3. evaluating the classifier
4. Hyper-parameter tuning
5. Evaluation after tuning
6. save the best classifier model

# Loading and preparing the Dataset

this involves loading the train set and test sets and extracts SIFT features and convert them to histrogram od codewords

## unzipping the dataset on colab

zipped dataset is unziped directly on the colab server for faster data accessing

```
#adapted from lab 7

# Identify path to zipped dataset

zip_path = os.path.join(GOOGLE_DRIVE_PATH, 'CW_Dataset.zip')

# Copy it to Colab

!cp '{zip_path}' .


# Unzip it

!yes|unzip -q CW_Dataset.zip


# Delete zipped version from Colab (not from Drive)

!rm CW_Dataset.zip


#import messesary libraries

import cv2

import matplotlib.pyplot as plt

import numpy as np

from sklearn.utils import shuffle

from skimage import img_as_ubyte, io, color

from collections import Counter

from sklearn.cluster import MiniBatchKMeans

from sklearn import svm, metrics

from joblib import dump


%matplotlib inline
```

```
## Loading the train dataset

# function to load and return images data and labels
def import_dataset(images_path, label_path):
  images_list = []  # empty images list
  labels_list = [] # empty label list
  #read all the image files
  for file in sorted(os.listdir(images_path)):
    if file.endswith('.jpg'):
      image = io.imread(os.path.join(images_path,file))     #read the image
      images_list.append(image)    #add it to the images list

  #get the labels for each image from the txt file

  data = np.genfromtxt(label_path)    #txt file to np
  labels = data[:,1]
  labels = labels.astype(int)    #convert label to int
  labels_list = labels.tolist()   #add to the label list

  return images_list, labels_list

#import training images and corresponding labels
train_images, train_labels = import_dataset('train','labels/list_label_train.txt')



print(len(train_images))

#display 10 images of the train set
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 5), sharex=True, sharey=True)
ax = axes.ravel()
```

```
for i in range(10):
   ax[i].imshow(train_images[i], cmap='gray')
   ax[i].set_title(f'Label: {train_labels[i]}')
   ax[i].set_axis_off()
fig.tight_layout()
plt.show()

#show number of different label classes
unique, counts = np.unique(train_labels, return_counts=True)

print(np.asarray((unique, counts)))
```

### SIFT feature descriptors extraction

sift.detectandcompute() function was used to extract the descriptors of the dataset

```
#adapted from lab 7
# a funtion to extract the SHIFT feature descriptors of the images with corresponding labels

def extract_SIFT_featureDes(images, labels):
  sift = cv2.SIFT_create()
  SIFT_descriptors = []
  SIFT_keypoints= []
  label_list =[]

  for i in range(len(images)):
    img= img_as_ubyte(color.rgb2gray(images[i]))
    kp, des = sift.detectAndCompute(img,None)
    if des is not None:
      SIFT_descriptors.append(des)
      SIFT_keypoints.append(kp)
```

```
    label_list.append(labels[i])



  return SIFT_descriptors, SIFT_keypoints, label_list


#extracting the training SIFT descriptors of the train dataset
train_SIFT_descriptors, train_SIFT_kp, train_SIFT_labels =
extract_SIFT_featureDes(train_images,train_labels)


  print(np.array(train_SIFT_descriptors).shape)
  print(len(train_SIFT_labels))


## plots 5 images from traing set with SIFT decriptors keypoints
fig, ax = plt.subplots(1, 5, figsize=(10, 8), sharey=True)
for i in range(5):
  if train_SIFT_descriptors is not None:
    img = img_as_ubyte(color.rgb2gray(train_images[i]))
    img_with_SIFT = cv2.drawKeypoints(img,train_SIFT_kp[i] , img) #draw  key point on the image
    ax[i].imshow(img_with_SIFT)
    ax[i].set_title(f'Label: {train_labels[i]}')
    ax[i].set_axis_off()
fig.tight_layout()
plt.show()


### descriptor clustering


###clusterning the descriptor with MiniBatchKMeans###


train_des_array = np.vstack(train_SIFT_descriptors)  # convert the descriptors into arrays


#number of codewords/centroids (Classes)
```

```
k= len(np.unique(train_SIFT_labels)) *10    #idial k is number of classes * 10


Kmeans =
MiniBatchKMeans(n_clusters=k,batch_size=train_des_array.shape[0]//5).fit(train_des_array)


#save kmeans clustering
path = os.path.join('drive', 'My Drive', 'Colab Notebooks/computerVision/Cw/Kmeans.joblib' )
dump(Kmeans, path)


### Histogram of Codewords


#adapted from lab 7
# function to convert the desctriptors into histogram of codewords
def create_hist_words (descriptors):
  hist_list=[]
  for des in descriptors:
    hist = np.zeros(k)
    idx= Kmeans.predict(des)

    for j in idx:
      hist[j] = hist[j] + (1/len(des))
    hist_list.append(hist)

  return hist_list


#converto descriptors of each training images into histogram of codewords
train_hist_list = create_hist_words(train_SIFT_descriptors)


## Loading and preparing the Test dataset


## lets load the test set and extract SIFT descriptors of each images and create histogram of them
```

test_images, test_labels = import_dataset('test','labels/list_label_test.txt')

test_SIFT_descriptors, test_SIFT_kp, test_SIFT_labels = extract_SIFT_featureDes(test_images,test_labels)

test_hist_list = create_hist_words(test_SIFT_descriptors)

**Here the SIFT feture descriptors extracted above will be used to train different image classifiers.**

#assigning X_train and y_train

X_train= np.vstack(train_hist_list)

y_train= train_SIFT_labels

# shuffle the dataset to prevent biases,

# by preventing the moddel from learning the order of th etraining

X_train , y_train = shuffle(X_train, y_train, random_state=42)

#assinging the X_test and y_test

X_test= np.vstack(test_hist_list)

y_test = test_SIFT_labels

# SIFT-SVM

## Training SVM classifier

# training the svm classifier by passing the codeword histograms of the traing images and their labels

from sklearn import svm

#classifier

```
svm_classifier= svm.SVC(kernel='rbf')

svm_classifier.fit(X_train,y_train)
```

## Evaluating the trained SVM classifier

```
#predicting the test set

svm_predict = svm_classifier.predict(X_test).tolist()


# printing the classifier report

print(f"""Classification report for classifier {svm_classifier}:

    {metrics.classification_report(y_test, svm_predict)}\n""")


#displays the Confusion matrix of the classifer

fig, ax = plt.subplots(figsize=(8,8))

disp = metrics.plot_confusion_matrix(svm_classifier, X_test, y_test, values_format = 'd', ax=ax)

disp.figure_.suptitle("SIFT-SVM Confusion Matrix")

plt.show()
```

## Improving the SVM classifier

Hyper Parametering tuning with Grid search will be used to improve the classifer by finding the best optimal parameters

### Hyper-Parameter tuning

```
# SVM Hyper parameter tuning using GridSearch to find the best optimal Hyper-parameter

#adapted from https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/


from sklearn.model_selection import GridSearchCV
```

```
# defining parameter range
svm_param_grid = {'C': [0.1, 1, 10],
            'gamma': [1, 0.5, 0.1],
            'kernel': ['rbf','poly']}
#classifier
svm_grid_classifier = GridSearchCV(svm.SVC(), svm_param_grid, refit = True,cv = 5, verbose = 4)

# traning the classifier with grid parameters
svm_grid_classifier.fit(X_train, y_train)

# print best parameter after tuning
print(svm_grid_classifier.best_params_)

print(svm_grid_classifier.best_estimator_)



### Evaluating the tuned SVM

# predicting the test set with tuned SVM
svm_grid_predict = svm_grid_classifier.predict(X_test)

print(f"""Classification report of tuned SIFT-SVM classifier {svm_grid_classifier}:
     {metrics.classification_report(y_test, svm_grid_predict)}\n""")

# confusion metrix after tuning the classifier
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(svm_grid_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("Confusion Matrix of tuned SIFT-SVM classifier")
plt.show()

## Save the SIFT-SVM classifier
```

the trained classifier model is saved on the drive with the help of joblib library

```
#save the classifier
from joblib import dump
path = os.path.join('drive', 'My Drive', 'Colab Notebooks/computerVision/Cw/SIFT_SVM' )
dump(svm_classifier, path)
```

# SIFT-MLP

## Training MLP classifier

```
####### HOG MLP classification ##############
from sklearn.neural_network import MLPClassifier

# MLP, Multi-Layer Perceptron, classifier
mlp_classifier = MLPClassifier(hidden_layer_sizes=(50,), max_iter=100, alpha=1e-4,
                solver='sgd', verbose=True, random_state=1,
                learning_rate_init=.1)

#fit the train set to the classifier
mlp_classifier.fit(X_train, y_train)
```

## Evaluating the trained MLP classifier

```
# pridict the test set
mlp_predicted= mlp_classifier.predict(X_test)

# printing the score of the MLP classifier
print(f"""Classification report for classifier {mlp_classifier}:\n
    {metrics.classification_report(y_test, mlp_predicted)}""")
```

```
# confusion metrix of SIFT-MLP classifier
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(mlp_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("Confusion Matrix of SIFT-MLP classifier")
plt.show()
```

## Improving the MLP classifier

### Hyper-Parameter tuning

```
# adapted from https://datascience.stackexchange.com/questions/36049/how-to-adjust-the-hyperparameters-of-mlp-classifier-to-get-more-perfect-performa
#parameters
mlp_param_grid = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['adaptive']}

#classifier
mlp_grid_classifier = GridSearchCV(MLPClassifier(max_iter=100), mlp_param_grid, refit = True, verbose = 3)

# training to find the best parameter
mlp_grid_classifier.fit(X_train, y_train)



# print best parameter after tuning


print(mlp_grid_classifier.best_params_)
```

```
print(mlp_grid_classifier.best_estimator_)
```

### Evaluating the tuned SIFT-MLP

```
# pridict the test set
mlp_grid_predicted= mlp_grid_classifier.predict(X_test)



# printing the score of the tuned MLP classifier
print(f"""Classification report for tuned SIFT-MLP classifier {mlp_grid_classifier}:\n
    {metrics.classification_report(y_test, mlp_grid_predicted)}""")

# confusion metrix of HOG-MLP classifier after grid search
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(mlp_grid_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("Confusion Matrix of tuned SIFT-MLP classifier")
plt.show()
```

## Save SIFT-MLP

```
#save the classifier
path = os.path.join('drive', 'My Drive', 'Colab Notebooks/computerVision/Cw/SIFT_MLP' )
dump(mlp_grid_classifier, path)
```

# SIFT-Random Forest Classifier

## Training the Random Forest classifier

```
from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier()
```

```
#Training the classifier with HOG features descriptors
rf_classifier.fit(X_train,y_train)

## Evaluating the trained Random Forest classifier

#predicting the test set
rf_predicted = rf_classifier.predict(X_test)

#eevaluating random forest classifier
print(f"""Classification report for classifier {rf_classifier}:
    {metrics.classification_report(y_test, rf_predicted)}\n""")

# confusion metrix of random fore classifier
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(rf_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("Confusion Matrix of SIFT-Random forest classifier")
plt.show()

## Improving the SVM classifier

### Hyper-Parameter tuning

# Hyper parameter tuning for random forest classifier using grid search
rf_grid_params = {
   'n_estimators': [10,50,100, 150, 250],
   'min_samples_split': [2, 4, 6]
}

rf_grid_classifier = GridSearchCV(RandomForestClassifier(), rf_grid_params, cv=5, verbose=5)
rf_grid_classifier.fit(X_train, y_train)
```

```
# print best parameter after tuning
print(rf_grid_classifier.best_params_)



### Evaluating the tuned Random Forest Classifier

#predicting of test set
rf_grid_predicted = rf_grid_classifier.predict(X_test)

#printing the score of tuned random forest classifier
print(f"""Classification report for classifier {rf_grid_classifier}:
    {metrics.classification_report(y_test, rf_grid_predicted)}\n""")

# confusion metrix of  classifier
fig, ax = plt.subplots(figsize=(8,8))
disp = metrics.plot_confusion_matrix(rf_grid_classifier, X_test, y_test, values_format = 'd', ax=ax)
disp.figure_.suptitle("Confusion Matrix of tuned SIFT-Random forest classifier")
plt.show()

## Save SIFT-Random Forest Classifier

#save the classifier
path = os.path.join('drive', 'My Drive', 'Colab Notebooks/computerVision/Cw/SIFT_RFC' )
dump(rf_grid_classifier, path)
```

## Test function (EmotionRecognition): Cw/test_function

```
# Google colab setup

from google.colab import drive
```

drive.mount('/content/drive', force_remount=True)

import os

GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/computerVision/Cw'

GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)

print(os.listdir(GOOGLE_DRIVE_PATH))

!pip install opencv-python==4.4.0.46

# Introduction

This file implements the EmotionRecognition funtion that that allows to

visualise qualitative results (i.e., predictions) obtained with the different models on the test set images

4 random images from the test set must be displayed together with the

model’s predictions and the ground-truth labels.

it can also be used for Facial Emotion Recognition on images “in the wild”.

Trained models that are available are HOG-SVM, HOG-MLP, HOG-Random Forest, SIFT-SVM, SIFT-MLP and SIFT-Random Forest

from joblib import load

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

import os

import numpy as np

from skimage.feature import hog

from sklearn import svm, metrics

from skimage import img_as_ubyte, io, color

import cv2

# EmotionRecognition()

the EmotionRecognition function takes 2 parameter syntax being path_to_testset is a string pointing at the test set of the provided dataset and

model_type is a string used to select a given model

```
### EMOTIONRECOGNITION FUNCTION###

# model type : "HOG-SVM" "HOG-MLP", "HOG-RFC", "SIFT-SVM", "SIFT-MLP", "SIFT-RFC"


def EmotionRecognition(path_to_testset, model_type):

  path= path_to_testset

  print("1= Surprise, 2= Fear, 3= Disgust, 4= Happiness, 5= Sadness, 6= Anger, 7= Neutral")


#### LOAD THE DATASET ######

  test_images=[]

  test_labels=[]

  #predicted= []

  for file in sorted(os.listdir(path)):

    if file.endswith('.txt'):

      data = np.genfromtxt(path + '/' + file)    #txt file to np

      labels = data[:,1]

      labels = labels.astype(int)    #convert label to int

      test_labels = labels.tolist()   #add to the label list

    if file.endswith('.jpg'):

      image = io.imread(os.path.join(path,file))     #read the image

      test_images.append(image)    #add it to the images list


  # shuffle for randomnness

  if len(test_labels)==0 :
```

```
    test_images = shuffle(test_images)
  else:
    test_images, test_labels = shuffle(test_images,test_labels)

### HOG MODELS ####
  if model_type.startswith('HOG'):
    #extracts hog features of 4 random images
    X_test = []
    #y_test = []

    for i in range(4):
      image = test_images[i]
      HOG_des, HOG_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                    cells_per_block=(1, 1), visualize=True, multichannel=True)
      X_test.append(HOG_des)

    #load and test the selected classifer
    if model_type == 'HOG-SVM':
      #load the classifer
      classifier = load('drive/My Drive/Colab Notebooks/computerVision/Cw/HOG_SVM.joblib')
      predicted = classifier.predict(X_test).tolist()
      print("Model: HOG-SVM")



    if model_type == 'HOG-MLP':
      classifier = load('drive/My Drive/Colab Notebooks/computerVision/Cw/HOG_MLP.joblib')
      predicted = classifier.predict(X_test).tolist()
      print("Model: HOG-MLP")

    if model_type == 'HOG-RFC':
      classifier = load('drive/My Drive/Colab Notebooks/computerVision/Cw/HOG_RFC.joblib')
```

```
    predicted = classifier.predict(X_test).tolist()
    print("Model: HOG-Random Forest Classifier")

######## SIFT MODELS #################
  if model_type.startswith('SIFT'):

   #load teh saved Kmeans
   Kmeans = load('drive/My Drive/Colab Notebooks/computerVision/Cw/Kmeans.joblib')

   #extract SIFT fetaures of 4 random images
   sift = cv2.SIFT_create()
   SIFT_descriptors = []
   #SIFT_labels = []
   for i in range(4):
    img= img_as_ubyte(color.rgb2gray(test_images[i]))
    kp, des = sift.detectAndCompute(img,None)
    SIFT_descriptors.append(des)
    #SIFT_labels.append(test_labels[i])

   # histogram codewords
   hist_list=[]
   k = 70
   for des in SIFT_descriptors:

    hist = np.zeros(k)
    idx= Kmeans.predict(des)

    for j in idx:
     hist[j] = hist[j] + (1/len(des))
    hist_list.append(hist)
```

```
  X_test = np.vstack(hist_list)
  #test_labels = SIFT_labels
  #load and test the selected classifer

  if model_type=='SIFT-SVM':

    classifier = load('drive/My Drive/Colab Notebooks/computerVision/Cw/SIFT_SVM')
    predicted = classifier.predict(X_test).tolist()
    print("Model: SIFT-SVM")

  if model_type == 'SIFT-MLP':
    classifier = load('drive/My Drive/Colab Notebooks/computerVision/Cw/SIFT_MLP')
    predicted = classifier.predict(X_test).tolist()
    print("Model: SIFT-MLP")

  if model_type == 'SIFT-RFC':
    classifier = load('drive/My Drive/Colab Notebooks/computerVision/Cw/SIFT_RFC')
    predicted = classifier.predict(X_test).tolist()
    print("Model: SIFT-Random Forest Classifier")

fig, ax = plt.subplots(1, 4, figsize=(10, 8), sharey=True)

for i in range(4):
  ax[i].imshow(test_images[i])
  if len(test_labels) == 0:
    ax[i].set_title(f'Prediction: {predicted[i]}')
  else:
    ax[i].set_title(f'Prediction: {predicted[i]}\n Ground Truth: {test_labels[i]}')
  ax[i].set_axis_off()
```

```
  fig.tight_layout()
  plt.show()

# Test on test set

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/test', "HOG-SVM")

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/test', "HOG-MLP")

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/test', "HOG-RFC")

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/test', "SIFT-SVM")

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/test', "SIFT-MLP")

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/test', "SIFT-RFC")

# Test on inTheWild

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/inTheWild', "SIFT-SVM")

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/inTheWild', "SIFT-MLP")

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/inTheWild', "SIFT-RFC")

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/inTheWild', "HOG-
SVM")

EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/inTheWild', "HOG-
MLP")
```

```
EmotionRecognition('drive/My Drive/Colab Notebooks/computerVision/Cw/inTheWild', "HOG-SVM")
```