In [ ]:
```
print ("Bismillah")
```

```
Bismillah
```

Imports

In [ ]:
```
import cv2
import os
import pandas as pd
import numpy as np
import sklearn

%matplotlib inline
import matplotlib.pyplot as plt

from google.colab.patches import cv2_imshow
```

Loading Data

In [ ]:
```
#Helper Function to Load data
def load_images_from_folder(folder, label):
  images = []
  for filename in os.listdir(folder):
    # i=i+1
    # print(i)
    img = cv2.imread(os.path.join(folder,filename),0)
    if img is not None:
      resized = cv2.resize(img, (150,150))
      # plt.imshow(img1)
      images.append(resized)
  labelLength = len(images)
  labelList = [label] * labelLength
  return (images, labelList)
```

In [ ]:
```
# Getting Train Data
train_data_pos, train_label_pos = load_images_from_folder("/content/drive/MyDrive/CV/As
train_data_neg, train_label_neg = load_images_from_folder("/content/drive/MyDrive/CV/As
```

In [ ]:
```
#Getting Test Data
test_data_pos, test_label_pos = load_images_from_folder("/content/drive/MyDrive/CV/Assi
test_data_neg, test_label_neg = load_images_from_folder("/content/drive/MyDrive/CV/Assi
```

Pre-Processing Data

In [ ]:
```
print(len(train_label_pos))
print(len(train_label_neg))
print(len(test_label_pos))
print(len(test_label_neg))

print(test_data_pos[0].dtype)
print(test_data_neg[81].shape)
cv2_imshow(test_data_pos[81])
```

```
2416
```

```
1218
1132
453
uint8
(150, 150)
```

In [ ]:
```python
# Calling DataFrame constructor after zipping
# both lists, with columns specified
pos_train = pd.DataFrame(list(zip(train_data_pos, train_label_pos)),
               columns =['Train_img', 'Train_label'])
neg_train = pd.DataFrame(list(zip(train_data_neg, train_label_neg)),
               columns =['Train_img', 'Train_label'])

# pos_train.head()
# neg_train.head()
```

In [ ]:
```python
pos_test = pd.DataFrame(list(zip(test_data_pos, test_label_pos)),
               columns =['Test_img', 'Test_label'])
neg_test = pd.DataFrame(list(zip(test_data_neg, test_label_neg)),
               columns =['Test_img', 'Test_label'])
# pos_test.head()
# neg_test.head()
```

In [ ]:
```python
train = pos_train.append(neg_train, ignore_index=True)
train
```

Out[ ]:

	Train_img	Train_label
0	[[191, 191, 190, 191, 192, 192, 192, 192, 192,...	human
1	[[115, 115, 116, 117, 117, 118, 118, 119, 121,...	human
2	[[109, 110, 110, 110, 109, 108, 113, 116, 118,...	human
3	[[148, 148, 148, 147, 148, 152, 150, 151, 155,...	human
4	[[107, 108, 110, 110, 108, 104, 110, 113, 113,...	human
...	...	...
3629	[[109, 111, 115, 113, 110, 193, 167, 130, 151,...	non-human
3630	[[95, 92, 84, 87, 90, 91, 104, 91, 97, 120, 10...	non-human
3631	[[142, 143, 142, 143, 145, 146, 146, 147, 146,...	non-human
3632	[[86, 107, 124, 106, 116, 100, 94, 110, 96, 97...	non-human
3633	[[193, 205, 206, 207, 209, 211, 216, 216, 216,...	non-human

3634 rows × 2 columns

In [ ]:
```
test = pos_test.append(neg_test, ignore_index=True)
test
```

Out[ ]:

	Test_img	Test_label
0	[[26, 26, 26, 27, 27, 28, 27, 27, 27, 27, 27, ...	human
1	[[129, 131, 135, 139, 144, 149, 151, 153, 151,...	human
2	[[43, 43, 43, 44, 46, 49, 49, 50, 44, 39, 53, ...	human
3	[[18, 18, 18, 18, 17, 17, 17, 18, 18, 18, 18, ...	human
4	[[104, 104, 104, 104, 104, 104, 104, 104, 104,...	human
...	...	...
1580	[[109, 123, 129, 160, 177, 153, 126, 120, 128,...	non-human
1581	[[241, 170, 184, 227, 226, 187, 230, 237, 233,...	non-human
1582	[[138, 98, 59, 72, 115, 137, 117, 128, 121, 92...	non-human
1583	[[224, 221, 224, 220, 222, 223, 223, 224, 224,...	non-human
1584	[[204, 187, 179, 179, 178, 163, 143, 112, 101,...	non-human

1585 rows × 2 columns

In [ ]:
```
# train_shuffled= train.iloc[np.random.permutation(train.index)].reset_index(drop=True)
# train_shuffled = sklearn.utils.shuffle(train).reset_index(drop=True)
train_shuffled = train.sample(frac=1).reset_index(drop=True)
train_shuffled
```

Out[ ]:

	Train_img	Train_label
0	[[74, 74, 74, 74, 72, 71, 75, 83, 96, 105, 107...	human
1	[[132, 134, 131, 133, 129, 129, 131, 133, 131,...	non-human
2	[[175, 173, 171, 171, 171, 170, 168, 166, 164,...	human
3	[[52, 54, 57, 68, 69, 64, 55, 50, 49, 56, 61, ...	human
4	[[52, 52, 54, 65, 65, 57, 50, 46, 45, 46, 48, ...	human
...	...	...
3629	[[51, 64, 115, 144, 139, 0, 52, 72, 0, 0, 0, 0...	non-human
3630	[[198, 205, 209, 210, 212, 216, 213, 212, 214,...	non-human
3631	[[253, 33, 14, 18, 25, 37, 102, 87, 90, 92, 88...	non-human
3632	[[101, 91, 79, 80, 86, 96, 88, 83, 80, 77, 71,...	human
3633	[[105, 105, 105, 106, 106, 106, 107, 108, 107,...	non-human

3634 rows × 2 columns

In [ ]:
```
# test_shuffled = sklearn.utils.shuffle(test).reset_index(drop=True)
test_shuffled = test.sample(frac=1).reset_index(drop=True)
test_shuffled
```

Out[ ]:

	Test_img	Test_label
0	[[185, 185, 185, 185, 185, 186, 185, 185, 184,...	human
1	[[253, 254, 254, 253, 254, 254, 254, 254, 254,...	non-human
2	[[100, 100, 100, 99, 96, 96, 112, 128, 127, 12...	human
3	[[144, 146, 152, 151, 133, 119, 126, 133, 148,...	human
4	[[29, 29, 28, 28, 29, 29, 28, 27, 26, 26, 26, ...	human
...	...	...
1580	[[154, 162, 164, 159, 157, 159, 163, 161, 148,...	non-human
1581	[[62, 63, 64, 65, 64, 62, 59, 56, 53, 50, 50, ...	human
1582	[[110, 102, 85, 71, 68, 71, 112, 153, 149, 145...	human
1583	[[24, 24, 25, 25, 25, 26, 27, 29, 29, 30, 31, ...	human
1584	[[129, 122, 117, 129, 145, 126, 129, 127, 118,...	non-human

1585 rows × 2 columns

Computing HoG

HoG for Training Data

In [ ]:
```
#Imports plus Initializing
from skimage import feature
from skimage import exposure
from tqdm.notebook import tqdm
```

In [ ]:
```
#initialize list that contains training images
train_data = []
train_hog_img = []
# count of all training images to use in loop
train_img_count = len(train_shuffled)
train_img_count
```

Out[ ]: 3634

In [ ]:
```
def compute_HOG(image):
  (H1, hogImage1) = feature.hog(image, orientations = 3,
                                pixels_per_cell  = (2, 2), cells_per_block  = (2, 2), t
                                block_norm  = 'L1' , visualize=True)
  return (H1, hogImage1)
```

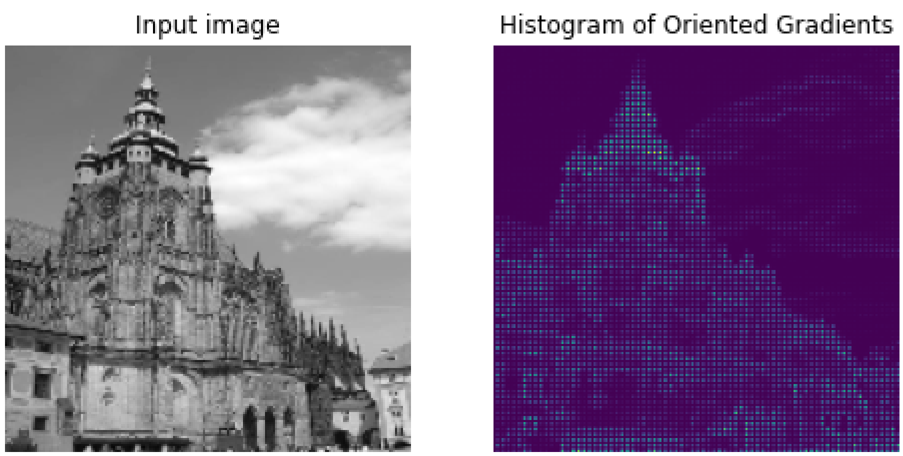
In [ ]:
```
# loop over the images
for i in tqdm(range(0,train_img_count)):
  # pre-process image here if needed
  # Computing the HOG features. Also Keep and eye on the parameters used in this functi
  (h_vector, h_image) = compute_HOG(train_shuffled.Train_img[i])
  #append computed HOGs in train data
  train_data.append(h_vector)
  train_hog_img.append(h_image)
```

```
#get train labels
train_label = train_shuffled.Train_label[0:train_img_count]
```

In [ ]:
```
train_shuffled['HoG_vector'] = train_data
train_shuffled['HoG_img'] = train_hog_img
train_shuffled.to_csv('/content/drive/MyDrive/CV/Assignment 2/TrainData.csv')
```

Visualize an example of computed HOG

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)

ax1.axis('off')
ax1.imshow(train_shuffled.Train_img[i], cmap=plt.cm.gray) #i==499
ax1.set_title('Input image')

# Rescale histogram for better display
hog_image_rescaled = exposure.rescale_intensity(h_image, in_range=(0, 255))

ax2.axis('off')
ax2.imshow(hog_image_rescaled)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()

# Length of computed feature vector
print("Length of computed vector\n",len(h_vector))
```




```
Length of computed vector
 65712
```

HoG for Testing Data

In [ ]:
```
test_data = []
test_hog_img = []
test_img_count = len(test_shuffled)

# Loop over the images
for i in tqdm(range(test_img_count)):
  # pre-process image here if needed
  # Computing the HOG features. Also Keep and eye on the parameters used in this functi
  (h_vector, h_image) = compute_HOG(test_shuffled.Test_img[i])
  #append computed HOGs in train data
  test_data.append(h_vector)
  test_hog_img.append(h_image)
```

```
#get labels
test_labels = test_shuffled.Test_label[0:test_img_count]
```

In [ ]:
```
test_shuffled['HoG_vector'] = test_data
test_shuffled['HoG_img'] = test_hog_img
test_shuffled.to_csv('/content/drive/MyDrive/CV/Assignment 2/TestData.csv')
```

//When Session dropped, using saved values in tables

In [ ]:
```
# train_shuffled = pd.read_csv('/content/drive/MyDrive/CV/Assignment 2/TrainData.csv')
# test_shuffled = pd.read_csv('/content/drive/MyDrive/CV/Assignment 2/TestData.csv')
```

In [ ]:
```
# train_data = train_shuffled.HoG_vector
# train_label = train_shuffled.Train_label
# test_data = test_shuffled.HoG_vector
# test_labels = test_shuffled.Test_label
# df.infer_objects().dtypes
```

SVM

Training

In [ ]:
```
#train_data --> contains vector histogram of train images
#train_label --> contains labels of train images
#test_data --> contains histogram of test images
#test_labels --> contains labels of test

from sklearn.svm import LinearSVC

# Load linear SVM
modelSVC = LinearSVC(max_iter=3000)
modelSVC.fit(train_data, train_label)
print("SVC training completed")
```

```
SVC training completed
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Lib
linear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Saving Model

In [ ]:
```
from joblib import dump, load
```

In [ ]:
```
dump(modelSVC, '/content/drive/MyDrive/CV/Assignment 2/modelSVM.joblib')
```

Out[ ]:
```
['/content/drive/MyDrive/CV/Assignment 2/modelSVM.joblib']
```

Loading Model

In [ ]:
```
modelSVC = load('/content/drive/MyDrive/CV/Assignment 2/modelSVM.joblib')
```

Testing

In [ ]:
```
# Create predictions
predicted_labels = modelSVC.predict(test_data)
```

```
print("Prediction completed")
# uncomment below lines to get the predicted labesl and the actural labels printed.
print("Comparing predicted and actual labels")
print(predicted_labels[0:10])
print(test_labels[0:10])
```

```
Prediction completed
Comparing predicted and actual labels
['human' 'non-human' 'non-human' 'human' 'human' 'human' 'non-human'
 'human' 'human' 'non-human']
0        human
1    non-human
2        human
3        human
4        human
5        human
6    non-human
7        human
8    non-human
9    non-human
Name: Test_label, dtype: object
```

Performance Report with SVM

Computing performance measures

In [ ]:
```
mask = predicted_labels==test_labels
correct = np.count_nonzero(mask)
print (correct*100.0/predicted_labels.size)
```

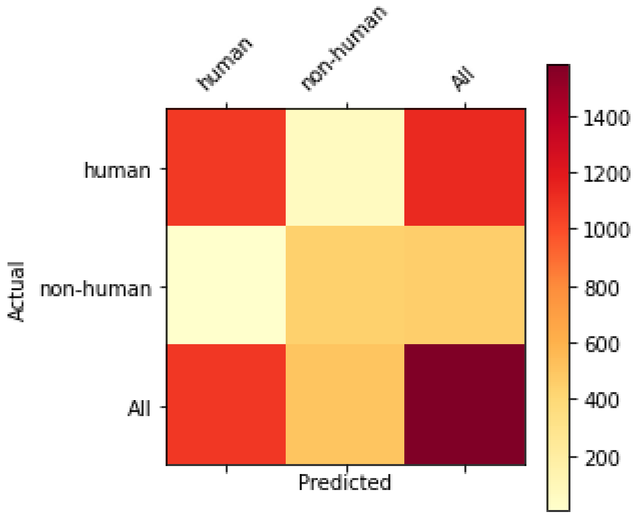
```
95.58359621451105
```

Confusion Matrix

In [ ]:
```
act = pd.Series(test_labels,name='Actual')
pred = pd.Series(predicted_labels,name='Predicted')
confusion_matrix = pd.crosstab(act, pred,margins=True)
print("Confusion matrix:\n%s" % confusion_matrix)
```

```
Confusion matrix:
Predicted  human  non-human   All
Actual
human       1071         61  1132
non-human      9        444   453
All         1080        505  1585
```

In [ ]:
```
#Plotting Above Confusion Matrix
def plot_confusion_matrix(df_confusion, title='Confusion matrix', cmap=plt.cm.YlOrRd):
  plt.matshow(df_confusion, cmap=cmap) # imshow
  plt.colorbar()
  tick_marks = np.arange(len(df_confusion.columns))
  plt.xticks(tick_marks, df_confusion.columns, rotation=45)
  plt.yticks(tick_marks, df_confusion.index)
  plt.ylabel(df_confusion.index.name)
  plt.xlabel(df_confusion.columns.name)


#call function
plot_confusion_matrix(confusion_matrix)

confusion_matrix.to_csv('/content/drive/MyDrive/CV/Assignment 2/SVM_ConfusionMatrix.csv
```



Classification Report

In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(test_labels, predicted_labels, target_names=["human","non-h
```

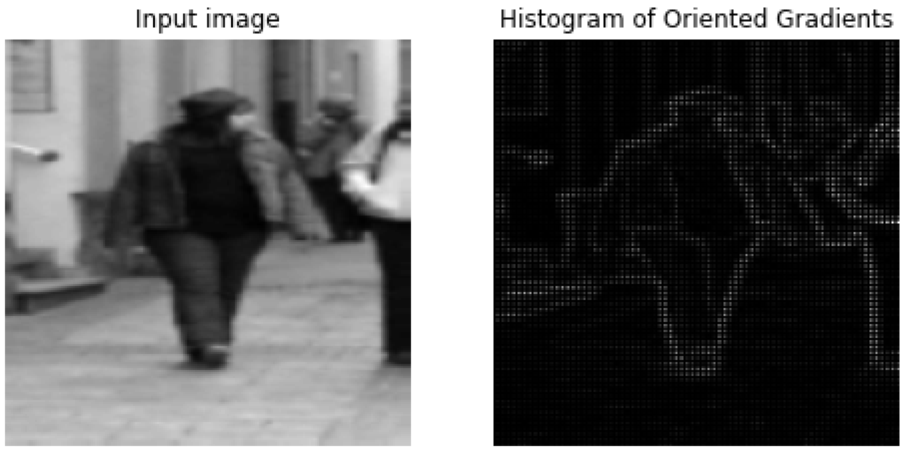
```
              precision    recall  f1-score   support

       human       0.99      0.95      0.97      1132
   non-human       0.88      0.98      0.93       453

    accuracy                           0.96      1585
   macro avg       0.94      0.96      0.95      1585
weighted avg       0.96      0.96      0.96      1585
```

Classifying Few correctly classified images

In [ ]:
```
# set index out of 10000 test images
index = 5
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelSVC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
```

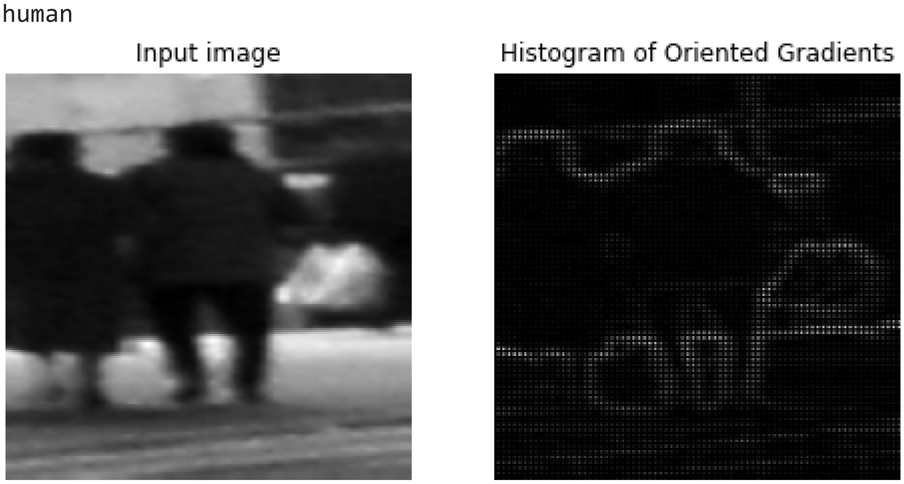
```
Predicted Label =
human
```



In [ ]:

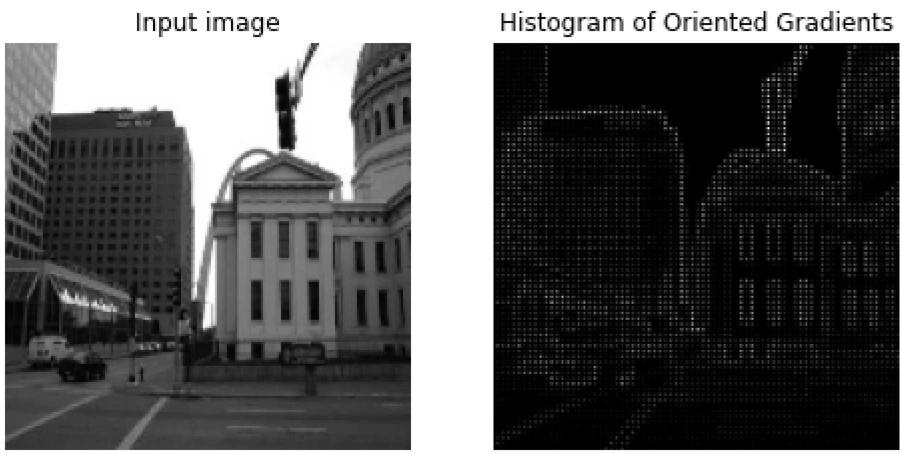
```python
# set index out of 10000 test images
index = 157
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelSVC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```



In [ ]:
```
# set index out of 10000 test images
index = 927
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelSVC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

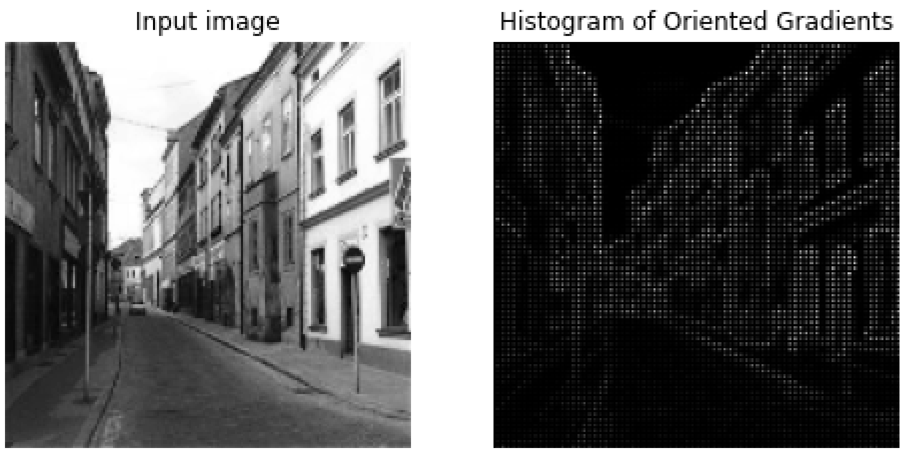
```
Actual Label =
non-human
Predicted Label =
non-human
```



In [ ]:
```
# set index out of 10000 test images
index = 741
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelSVC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
```

```
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

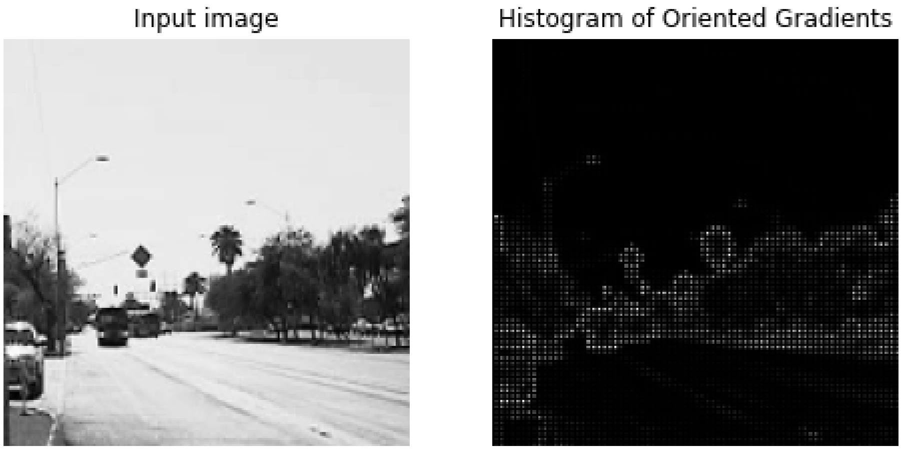
```
Actual Label =
non-human
Predicted Label =
non-human
```



In [ ]:

```
# set index out of 10000 test images
index = 139
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelSVC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

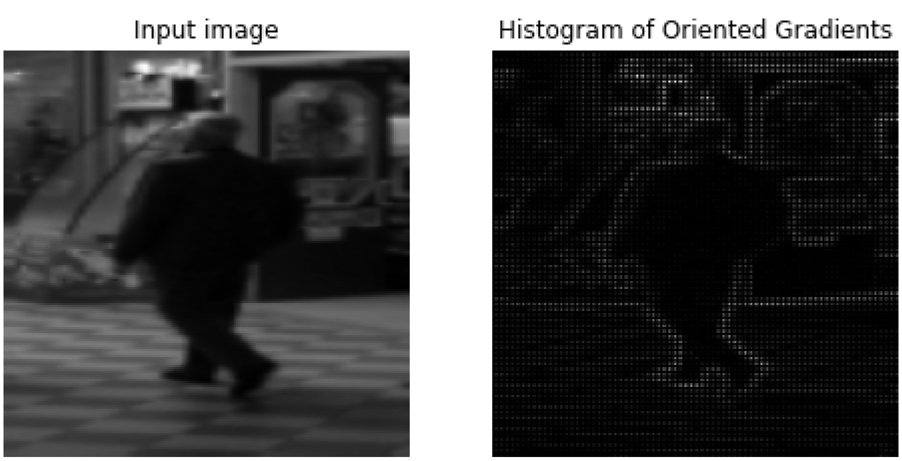
```
Actual Label =
non-human
Predicted Label =
non-human
```



In [ ]:

```
# set index out of 10000 test images
index = 789
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelSVC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```



In [ ]:

```
# set index out of 10000 test images
```

```
index = 456
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelSVC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```

Random Forrest Classifier

Training

In [ ]:
```
#train_data --> contains vector histogram of train images
#train_label --> contains labels of train images
#test_data --> contains histogram of test images
#test_labels --> contains labels of test

from sklearn.ensemble import RandomForestClassifier

# load linear Random Forrest Classifier
modelRFC = RandomForestClassifier(max_depth=2, random_state=0)
modelRFC.fit(train_data, train_label)
print("Random Forrest training completed")
```

```
Random Forrest training completed
```

Saving Model

In [ ]:
```
dump(modelRFC, '/content/drive/MyDrive/CV/Assignment 2/modelRFC.joblib')
```

Out[ ]:
```
['/content/drive/MyDrive/CV/Assignment 2/modelRFC.joblib']
```

Loading Model

In [ ]:
```
modelRFC = load('/content/drive/MyDrive/CV/Assignment 2/modelRFC.joblib')
```

Testing

In [ ]:
```
# Create predictions
predicted_RFC_labels = modelRFC.predict(test_data)
print("Prediction completed")
# uncomment below lines to get the predicted labesl and the actural labels printed.
print("Comparing predicted and actual labels")
print(predicted_RFC_labels[0:10])
print(test_labels[0:10])
```

```
Prediction completed
Comparing predicted and actual labels
['human' 'non-human' 'human' 'human' 'human' 'human' 'non-human' 'human'
 'human' 'non-human']
0        human
1    non-human
2        human
3        human
4        human
5        human
6    non-human
7        human
8    non-human
9    non-human
Name: Test_label, dtype: object
```

Performance Report with Random Forrest

Computing performance measures

In [ ]:
```
mask = predicted_RFC_labels==test_labels
correct = np.count_nonzero(mask)
print (correct*100.0/predicted_RFC_labels.size)
```

```
79.87381703470031
```

Confusion Matrix

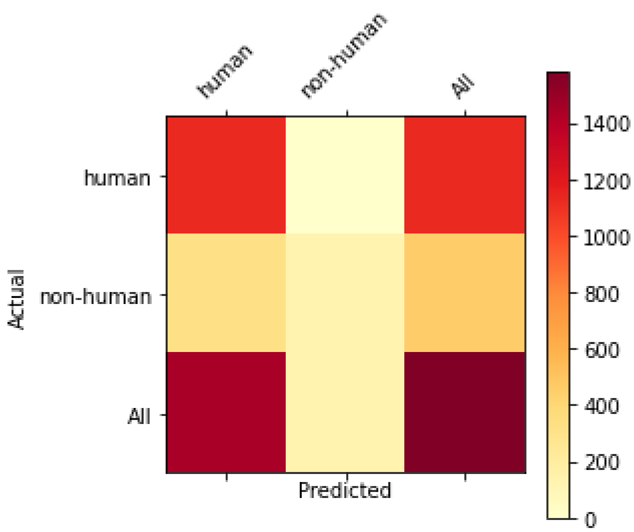
In [ ]:
```
act = pd.Series(test_labels,name='Actual')
pred = pd.Series(predicted_RFC_labels,name='Predicted')
confusion_matrix = pd.crosstab(act, pred,margins=True)
print("Confusion matrix:\n%s" % confusion_matrix)

confusion_matrix.to_csv('/content/drive/MyDrive/CV/Assignment 2/RandomForrest_Confusion
```

```
Confusion matrix:
Predicted  human  non-human   All
Actual
human       1132          0  1132
```

```
non-human     319        134   453
All          1451        134  1585
```

In [ ]:

```
#Plotting Above Confusion Matrix
def plot_confusion_matrix(df_confusion, title='Confusion matrix', cmap=plt.cm.YlOrRd):
  plt.matshow(df_confusion, cmap=cmap) # imshow
  plt.colorbar()
  tick_marks = np.arange(len(df_confusion.columns))
  plt.xticks(tick_marks, df_confusion.columns, rotation=45)
  plt.yticks(tick_marks, df_confusion.index)
  plt.ylabel(df_confusion.index.name)
  plt.xlabel(df_confusion.columns.name)


#call function
plot_confusion_matrix(confusion_matrix)
```



Classification Report

In [ ]:

```
from sklearn.metrics import classification_report
print(classification_report(test_labels, predicted_RFC_labels, target_names=["human","n
```

```
              precision    recall  f1-score   support

       human       0.78      1.00      0.88      1132
   non-human       1.00      0.30      0.46       453

    accuracy                           0.80      1585
   macro avg       0.89      0.65      0.67      1585
weighted avg       0.84      0.80      0.76      1585
```

Classifying Few correctly classified images

In [ ]:

```
# set index out of 10000 test images
index = 5
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelRFC.predict([h_vector])

#comparison
```

```
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```

In [ ]:

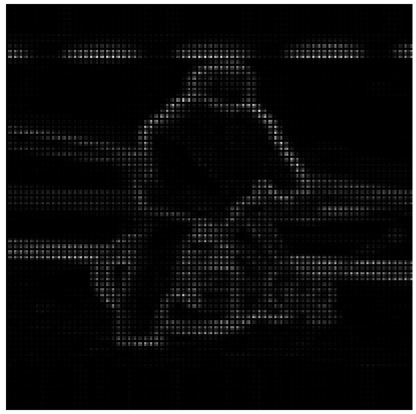
```
# set index out of 10000 test images
index = 10
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelRFC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
```

```
Predicted Label =
human
```

Input image

Histogram of Oriented Gradients



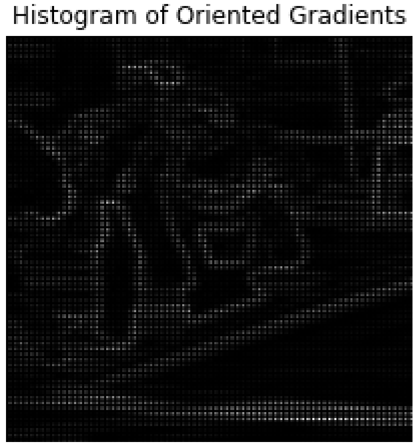
In [ ]:

```python
# set index out of 10000 test images
index = 156
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelRFC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```

Input image

Histogram of Oriented Gradients



In [ ]:

```
# set index out of 10000 test images
index = 271
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelRFC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```

Input image

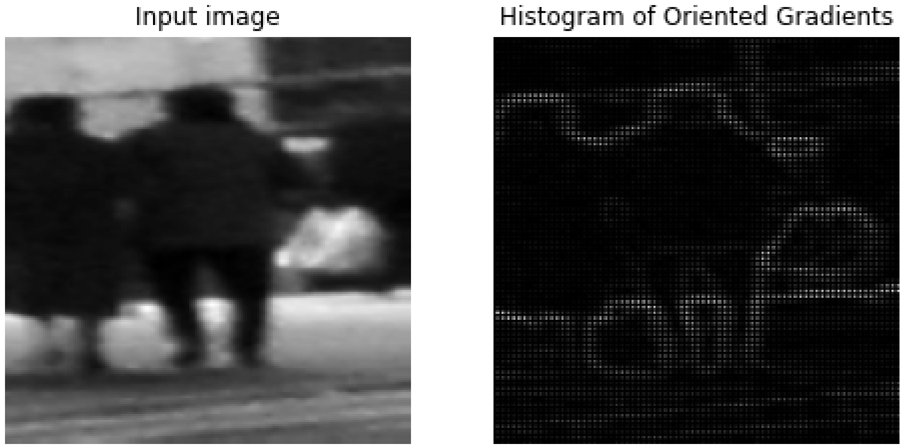
Histogram of Oriented Gradients

In [ ]:

```
# set index out of 10000 test images
index = 157
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelRFC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
```

```
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

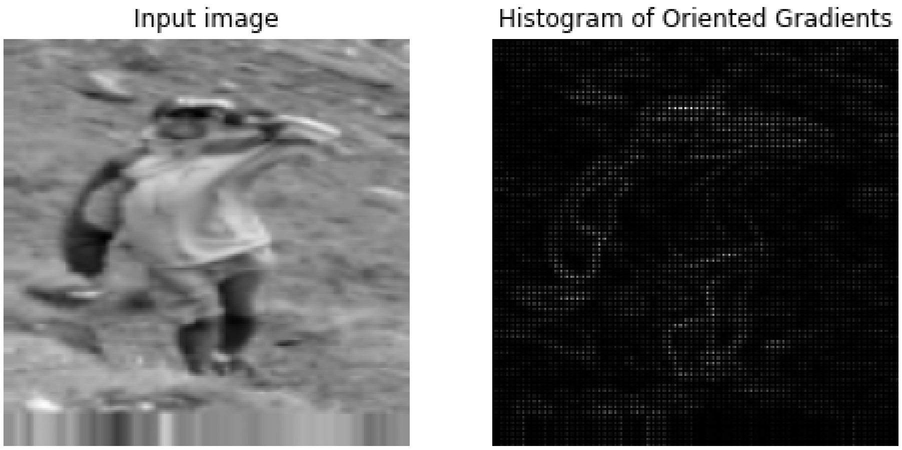
```
Actual Label =
human
Predicted Label =
human
```



In [ ]:

```
# set index out of 10000 test images
index = 1357
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelRFC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

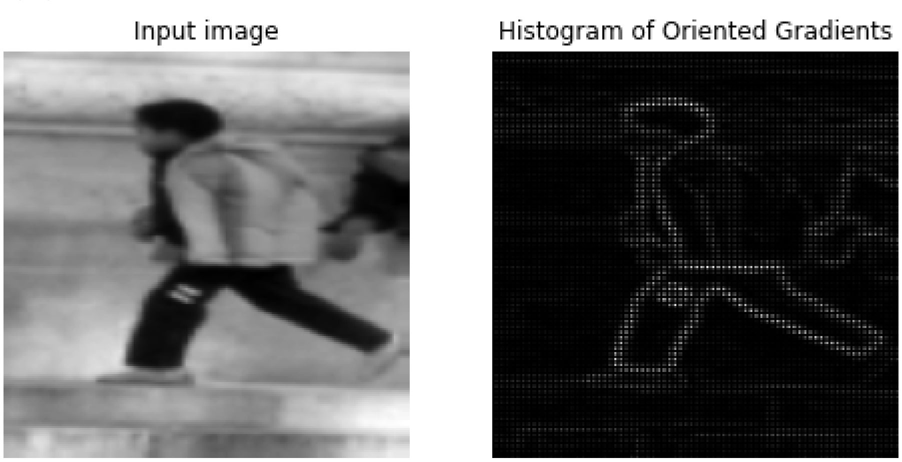
```
Actual Label =
human
Predicted Label =
human
```



In [ ]:

```
# set index out of 10000 test images
index = 875
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelRFC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```



Gradient Boosting Classifier

Training

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier

#train_data --> contains vector histogram of train images
#train_label --> contains labels of train images
#test_data --> contains histogram of test images
#test_labels --> contains labels of test

# load Gradient Boost
# gb = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_f
modelGBC = GradientBoostingClassifier(random_state=0)
modelGBC.fit(train_data, train_label)
print("Gradient Boosting training completed")
```

```
Gradient Boosting training completed
```

Saving Model

In [ ]:
```
dump(modelGBC, '/content/drive/MyDrive/CV/Assignment 2/modelGBC.joblib')
```

Out[ ]:
```
['/content/drive/MyDrive/CV/Assignment 2/modelGBC.joblib']
```

Loading Model

In [ ]:
```
modelGBC = load('/content/drive/MyDrive/CV/Assignment 2/modelGBC.joblib')
```

Testing

In [ ]:
```
# Create predictions
predicted_GBC_labels = modelGBC.predict(test_data)
print("Prediction completed")
# uncomment below lines to get the predicted labesl and the actural labels printed.
print("Comparing predicted and actual labels")
print(predicted_GBC_labels[0:10])
print(test_labels[0:10])
```

```
Prediction completed
Comparing predicted and actual labels
['human' 'non-human' 'human' 'human' 'human' 'human' 'non-human' 'human'
 'non-human' 'non-human']
0        human
1    non-human
2        human
3        human
4        human
5        human
6    non-human
7        human
8    non-human
9    non-human
Name: Test_label, dtype: object
```

Performance Report with Gradient Boosting Classifier

Computing performance measures

In [ ]:
```
mask = predicted_GBC_labels==test_labels
correct = np.count_nonzero(mask)
print (correct*100.0/predicted_GBC_labels.size)
```

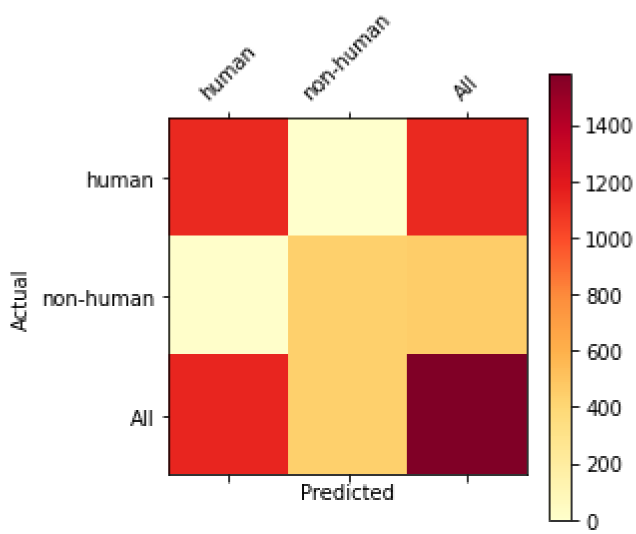
```
99.43217665615143
```

Confusion Matrix

In [ ]:
```
act = pd.Series(test_labels,name='Actual')
pred = pd.Series(predicted_GBC_labels,name='Predicted')
confusion_matrix = pd.crosstab(act, pred,margins=True)
print("Confusion matrix:\n%s" % confusion_matrix)
```

```
Confusion matrix:
Predicted  human  non-human   All
Actual
human       1132          0  1132
non-human      9        444   453
All         1141        444  1585
```

In [ ]:
```
#Plotting Above Confusion Matrix
def plot_confusion_matrix(df_confusion, title='Confusion matrix', cmap=plt.cm.YlOrRd):
  plt.matshow(df_confusion, cmap=cmap) # imshow
  plt.colorbar()
  tick_marks = np.arange(len(df_confusion.columns))
  plt.xticks(tick_marks, df_confusion.columns, rotation=45)
  plt.yticks(tick_marks, df_confusion.index)
  plt.ylabel(df_confusion.index.name)
  plt.xlabel(df_confusion.columns.name)


#call function
plot_confusion_matrix(confusion_matrix)

confusion_matrix.to_csv('/content/drive/MyDrive/CV/Assignment 2/GBC_ConfusionMatrix.csv
```



Classification Report

In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(test_labels, predicted_GBC_labels, target_names=["human","n
```

```
              precision    recall  f1-score   support

       human       0.99      1.00      1.00      1132
```

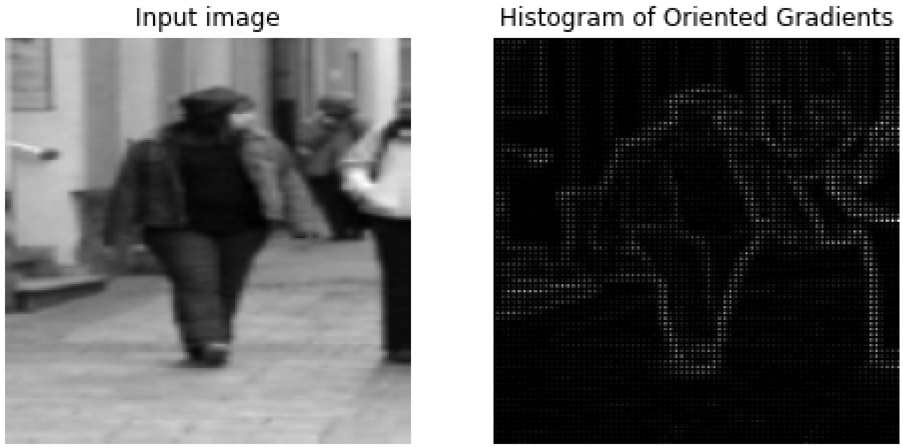
```
   non-human       1.00      0.98      0.99       453

    accuracy                           0.99      1585
   macro avg       1.00      0.99      0.99      1585
weighted avg       0.99      0.99      0.99      1585
```

Classifying Few correctly classified images

In [ ]:

```
# set index out of 10000 test images
index = 5
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelGBC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```

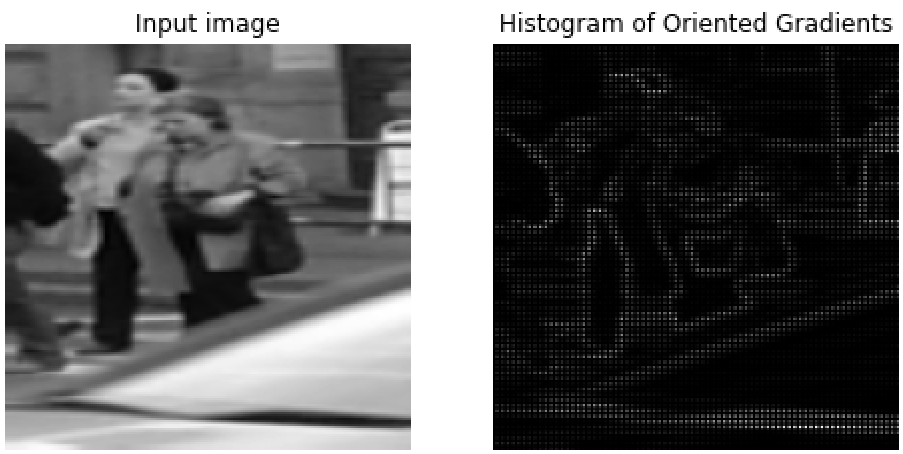


In [ ]:

```
# set index out of 10000 test images
index = 156
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelGBC.predict([h_vector])
```

```
#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

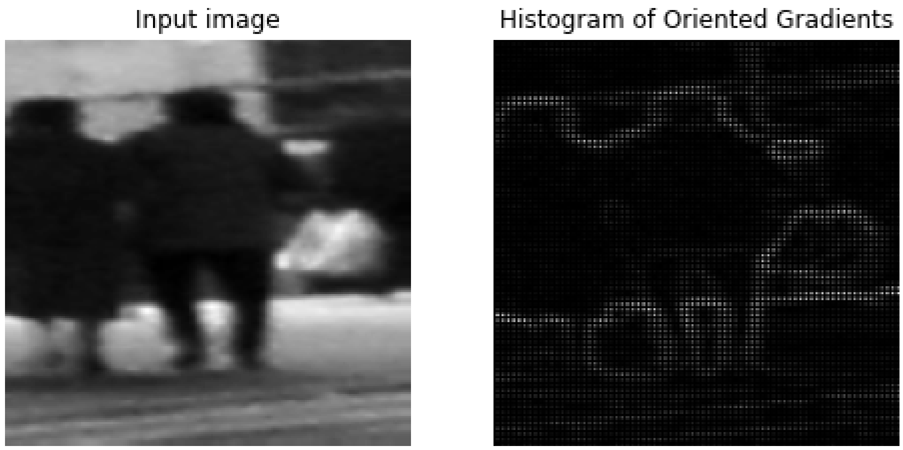
```
Actual Label =
human
Predicted Label =
human
```



In [ ]:

```
# set index out of 10000 test images
index = 157
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelGBC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

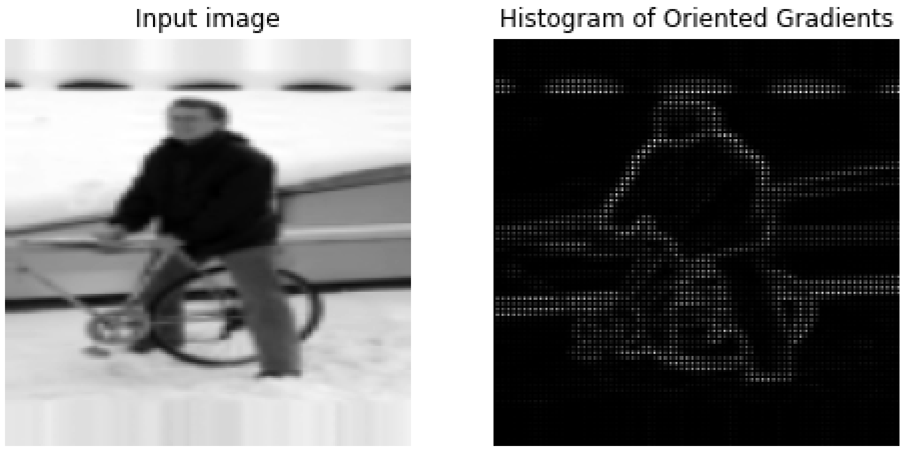
```
Actual Label =
human
Predicted Label =
human
```



In [ ]:

```python
# set index out of 10000 test images
index = 1255
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelGBC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

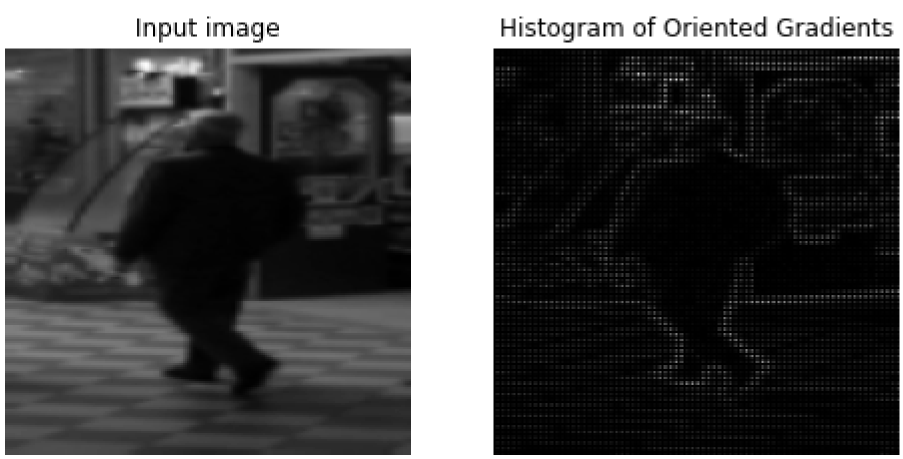
```
Actual Label =
human
Predicted Label =
human
```



In [ ]:

```
# set index out of 10000 test images
index = 789
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelGBC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```

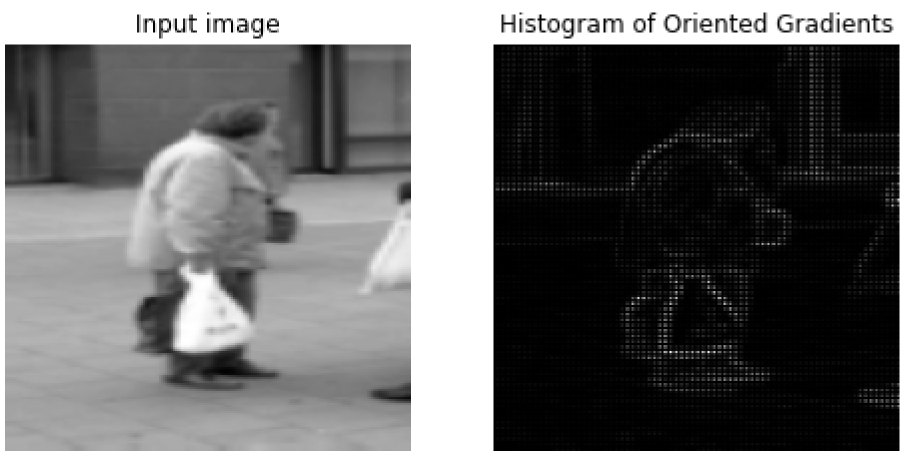


In [ ]:

```
# set index out of 10000 test images
```

```
index = 666
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelGBC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
human
Predicted Label =
human
```

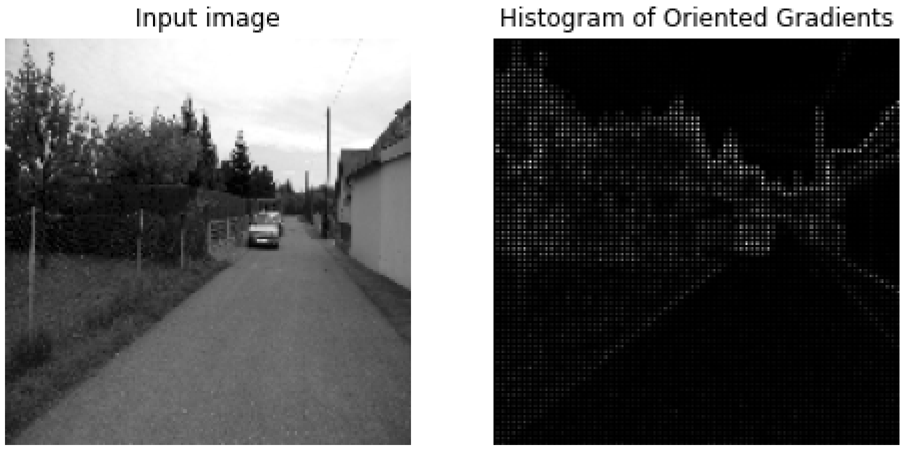


In [ ]:

```
# set index out of 10000 test images
index = 1171
#get image
image = test_shuffled.Test_img[index]

#compute hog feature vector for above image
(h_vector, h_image) = compute_HOG(image)
obtained_label = modelGBC.predict([h_vector])

#comparison
print("Actual Label =")
print(test_labels[index])
print("Predicted Label =")
print(obtained_label[0])

#visualize
figr, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
```

```
ax1.axis('off')
ax1.imshow(image, cmap="gray")
ax1.set_title('Input image')
ax2.axis('off')
ax2.imshow(h_image,cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```

```
Actual Label =
non-human
Predicted Label =
non-human
```



Google Colab Link:

https://colab.research.google.com/drive/1rTEbSQmwgYRg7alArT5wlOIftXAhaGn7?usp=sharing