With fewer than 500 North Atlantic right whales left in the world's oceans, knowing the health and status of each whale is integral to the efforts of researchers working to protect the species from extinction.

Currently, only a handful of very experienced researchers can identify individual whales on sight while out on the water. We aim to build a classifier that use aerial images of whales to predict the individual whales.

In [4]:
```
import os
import glob
import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from skimage.feature import hog
from numpy import linalg
import numpy.matlib
from IPython.display import clear_output
from time import sleep
```

## Dataset Preparation

In [2]:
```
all_whales_df = pd.read_csv("train.csv")
```

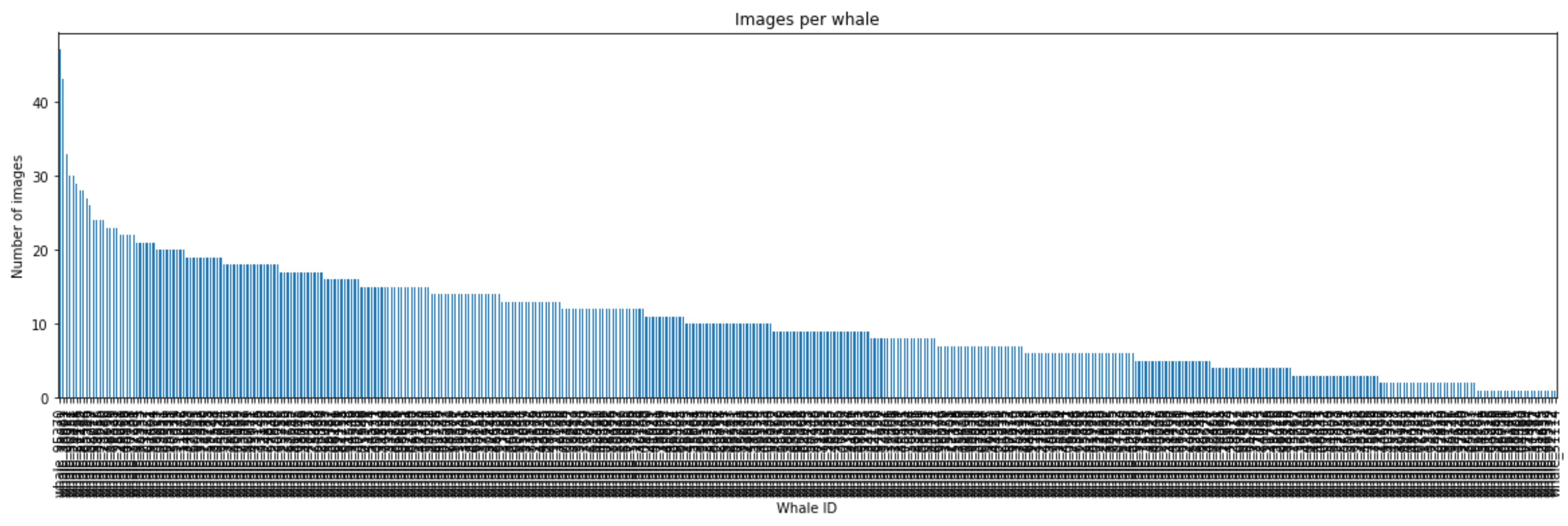
In [3]:
```
all_whales_df.head(3)
```

Out[3]:

| | Image | whaleID |
|---|---|---|
| **0** | w_7812.jpg | whale_48813 |
| **1** | w_4598.jpg | whale_09913 |
| **2** | w_3828.jpg | whale_45062 |

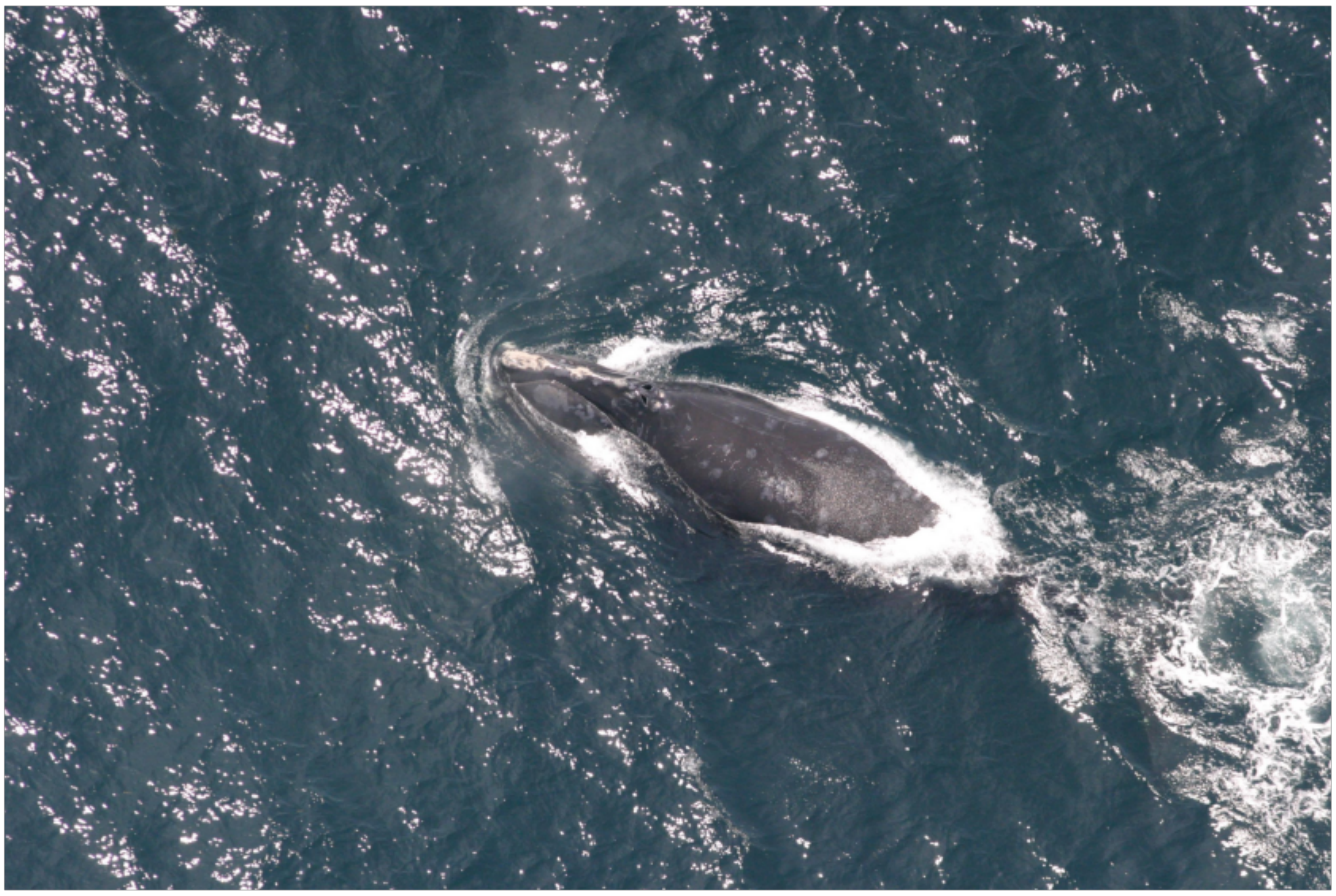
In [31]:
```
from matplotlib.pyplot import figure
import seaborn as sns

plt.figure(figsize=(20, 5))
plt.title('Images per whale')
plt.ylabel('Number of images')
plt.xlabel('Whale ID')

# print(list(all_whales_df['whaleID'].value_counts()))
all_whales_df['whaleID'].value_counts().plot.bar()
```

Out[31]:
```
<AxesSubplot:title={'center':'Images per whale'}, xlabel='Whale ID', ylabel='Number of images'>
```



There are a of whales which less than 10 images in the dataset. So we decide to focus of just five whales where each whale has more than 30 images in the dataset. This gives us enough data for each class of the classifier.

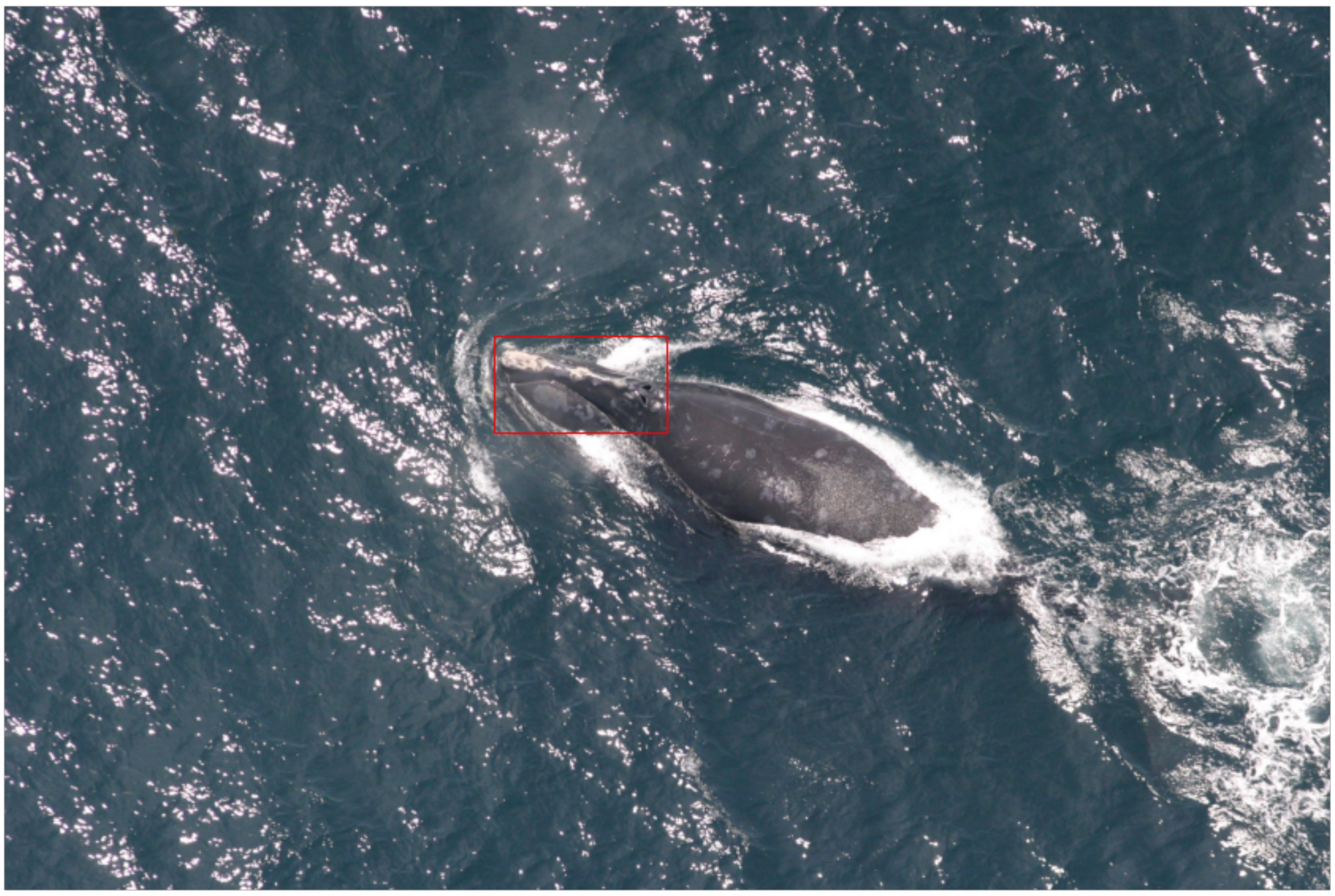
In [12]:
```python
# sample image from dataset
im = cv2.imread('imgs/whale_08017/w_1194.jpg')
plt.figure(figsize=(15,15))
plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```



The feature that I am most interested in is the callosity pattern which includes the facial markings on top of the head of the whale and the white markings above the blowholes. These features are unique to each whale. According to expert 'Christin B. Khan' on Kaggle(https://www.kaggle.com/competitions/noaa-right-whale-recognition/discussion/16172 (https://www.kaggle.com/competitions/noaa-right-whale-recognition/discussion/16172)), "The white markings on top of the whale's head are called callosities and they change very slightly over time". The dorsal fins, side flippers, and tail shape were other characteristics I chose to disregard even though they would have been informative but would have made detection more difficult.

For this project, we only focus on the marks on the top of the head.

In [14]:
```
# The bounding box shows the area we would be focusing on
plt.figure(figsize=(15,15))
plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB))
plt.axis('off')
rect = patches.Rectangle((1131.3410981697173, 763.3144758735441), 402.1031613976704, 224.905, linewidt
plt.gca().add_patch(rect)
plt.show()
```



There are annotations available online for the whale heads. We use one of those files with annotations.

In [15]:
```
annotations_df = pd.read_json('/Users/pratikaher/FALL22/CV/Whale/right_whale_hunt/annotations/whale_fa
```

In [16]:
```
annotations_df.head(2)
```

Out[16]:

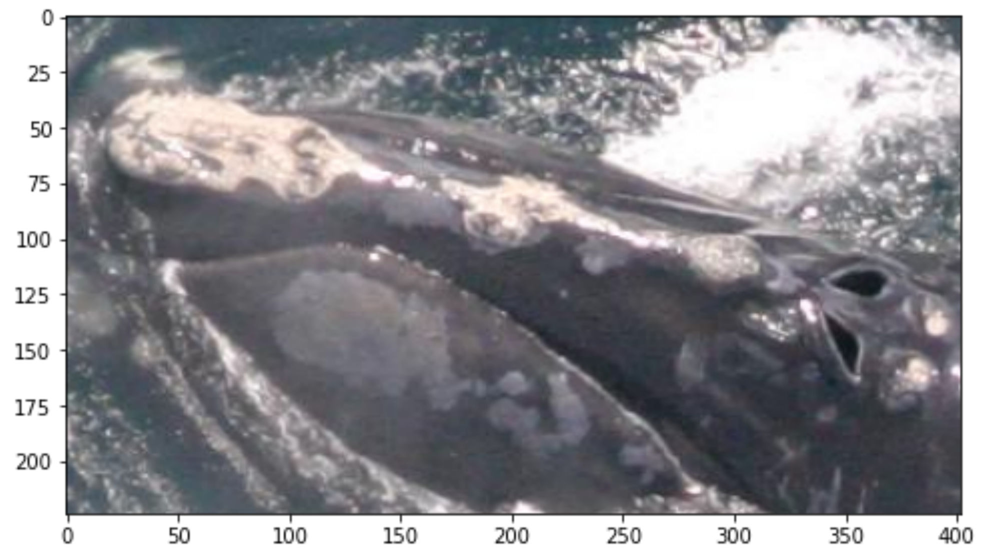
| | annotations | class | filename |
|---|---|---|---|
| **0** | [{'class': 'Head', 'height': 512.0, 'type': 'r... | image | imgs/whale_08017/w_1106.jpg |
| **1** | [{'class': 'Head', 'height': 224.9051580698835... | image | imgs/whale_08017/w_1194.jpg |

In [17]:
```
annotations_df['annotations'][1][0]
```

Out[17]:
```
{'class': 'Head',
 'height': 224.90515806988356,
 'type': 'rect',
 'width': 402.1031613976704,
 'x': 1131.3410981697173,
 'y': 763.3144758735441}
```

In [18]:
```
def get_cropped_image(im, annotations):
    y = int(annotations['y'])
    x = int(annotations['x'])
    h = int(annotations['height'])
    w = int(annotations['width'])

    crop_img = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)[y:y+h, x:x+w]
    return crop_img
```

In [19]:
```python
plt.figure(figsize=(8,8))
crop_img = get_cropped_image(im, annotations_df['annotations'][1][0])
plt.imshow(crop_img)
plt.show()
```



The above example image is what we will focus on for every whale. For this project, we take the five whales with the most number of images and try to explore various algorithms and techniques to classify those individuals. For those five whales, we will extract the coordinates of the head and crop the images and store them seperately. Below is the code that does that.

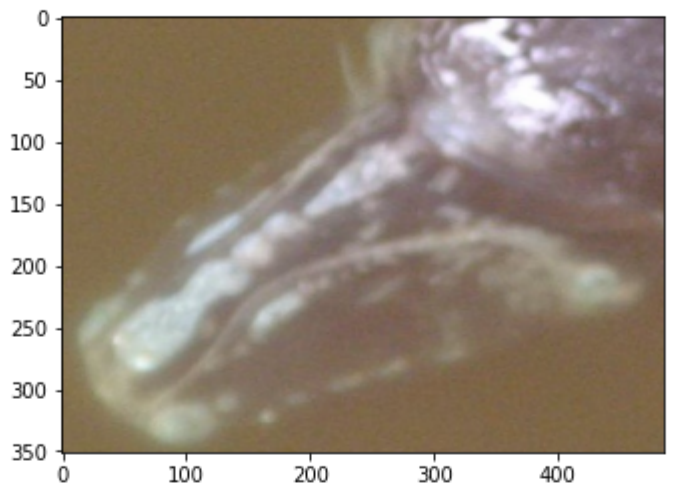
In [66]:
```python
path = '/Users/pratikaher/FALL22/CV/Whale/whale_samples/'
write_path = '/Users/pratikaher/FALL22/CV/Whale/whale_samples/cropped/'
```

In [69]:
```python
for file in os.listdir(path):
    if file != '.DS_Store':

#         print(file, annotations_df[annotations_df['filename'].str.contains(file)])
        for index, row in annotations_df[annotations_df['filename'].str.contains(file)].iterrows():

            annotations_list = row['annotations']

            if not annotations_list:
                continue

            for r in annotations_list:
                if r['class'] == 'Head':
                    annotations = r

            im = cv2.imread(path+file+ '/' +row['filename'].split('/')[-1])

            cropped_image = get_cropped_image(im, annotations)

#             rect = patches.Rectangle((annotations['x'], annotations['y']), annotations['width'], ann
#             plt.gca().add_patch(rect)
#             plt.show()
#             plt.figure(figsize=(15,15))
#             plt.imshow(cropped_image)

            newpath = write_path + file
            if not os.path.exists(newpath):
                os.makedirs(newpath)

            cv2.imwrite(newpath + '/' +row['filename'].split('/')[-1],  cv2.cvtColor(cropped_image, cv
#             plt.imshow(cv2.cvtColor(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB), cv2.COLOR_BGR2RG
#             plt.show()
```

Orientation seems to be quite important for the classifier to be able to detect images. So we rotate the image so that all images are oriented from top to bottom. We use cv2.findcontours and then get the biggest contour and rotate the image around that contour. We assume the biggest contour is the whale head. This helps all images to be aligned from the top-bottom direction. This seems to work in about 80% of images. For the rest, I manually flip the images so that they are aligned.

In [70]:
```python
countour_example = cv2.imread('/Users/pratikaher/FALL22/CV/Whale/whale_samples/cropped/whale_48813/w_3
countour_example_hsvImage = cv2.cvtColor(countour_example, cv2.COLOR_BGR2HSV)
countour_example_gray = cv2.cvtColor(countour_example_hsvImage, cv2.COLOR_BGR2GRAY)

contours, hierarchy = cv2.findContours(countour_example_gray,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)

best = None
best_size = 0
for cnt in contours:
#     print(cnt)
    x,y,w,h = cv2.boundingRect(cnt)
    size = w * h
    if size > best_size:
        best_size = size
        best = cnt
```

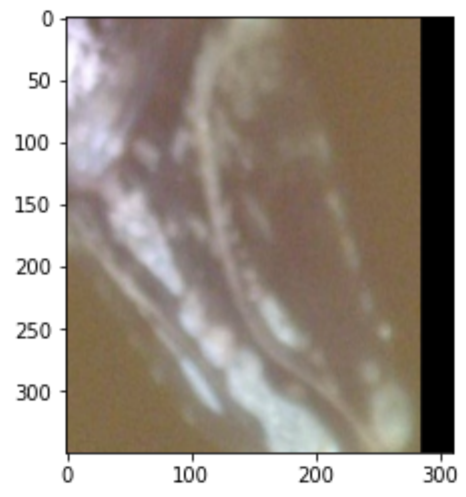
In [71]:
```python
plt.imshow(countour_example)
plt.show()
```



In [72]:
```python
ellipse = cv2.fitEllipse(best)
```

In [73]:
```python
def intround(floatNum):

    # print(type(floatNum), isinstance(floatNum, np.ndarray))
    if isinstance(floatNum, np.ndarray):
        return int(round(floatNum[0]))

    return int(round(floatNum))

def rotateAndCrop(img, ellipse):
    # Center of rotation is the center of the image
    center=tuple(np.array(img.shape[0:2])/2)

    # Affine transform - rotation about the center
    rotMat = cv2.getRotationMatrix2D(center, ellipse[2], 1.0)

    # Recalculate the size of the rotated image
    # so the corners do not get cut off
    (h, w) = img.shape[0:2]
    r = np.deg2rad(ellipse[2])
    w, h = (abs(np.sin(r)*h) + abs(np.cos(r)*w),abs(np.sin(r)*w) + abs(np.cos(r)*h))

    rotated = cv2.warpAffine(img,rotMat,dsize=(int(w),int(h)),flags=cv2.INTER_LINEAR)

    # Get the center of the ellipse and rotate it with the affine transform
    centEll = np.ones((3,1))
    centEll[0] = ellipse[0][0]
    centEll[1] = ellipse[0][1]
    centEll = np.dot(rotMat, centEll)

    # Get the half lengths of the two axes
    mjAxis = max(ellipse[1]) / 2.0
    miAxis = min(ellipse[1]) / 2.0

    p1x = intround(centEll[0] - miAxis)
    p1y = intround(centEll[1] - mjAxis)
    p2x = intround(centEll[0] + miAxis)
    p2y = intround(centEll[1] + mjAxis)

    if p1x < 0:
        p1x = 0
    if p1y < 0:
        p1y = 0
    if p2x >= img.shape[1]:
        p2x = img.shape[1]-1
    if p2y >= img.shape[0]:
        p2y = img.shape[0]-1

    return rotated[p1y:p2y, p1x:p2x]

```

In [74]:
```
plt.imshow(rotateAndCrop(countour_example, ellipse))
plt.show()
```



## Divide data into train-test-validation split

In [4]:
```
cropped_path = '/Users/pratikaher/FALL22/CV/Whale/whale_samples/cropped/'

image_names = []
target_values = []
target_index = 0

for more_dirs in os.listdir(cropped_path):

    if more_dirs == '.DS_Store':
        continue

    list_images_in_folder = list(glob.glob(cropped_path+ more_dirs +'/*.jpg'))
    image_names.extend(list_images_in_folder)
    target_values.extend([target_index]*len(list_images_in_folder))
    target_index += 1
```

In [6]:
```
image_data = list(zip(image_names, target_values))
```

In [7]:
```
import random
random.shuffle(image_data)

test_size = int(len(image_data) * 0.2)
valid_size = int(len(image_data) * 0.1)
train_size = len(image_data) - test_size - valid_size

train_data = image_data[:train_size]
valid_data = image_data[train_size:train_size+valid_size]
test_data = image_data[-test_size:]
```

# Feature Exploration

## Histogram of oriented gradients feature

In [20]:
```
fd, hog_image = hog(cv2.cvtColor(crop_img, cv2.COLOR_BGR2RGB), orientations=8, pixels_per_cell=(8, 8),
                    cells_per_block=(1, 1), visualize=True, channel_axis=-1)

plt.figure(figsize=(6,6))
plt.axis("off")
plt.imshow(hog_image, cmap="gray")
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x7fb12320a860>
```

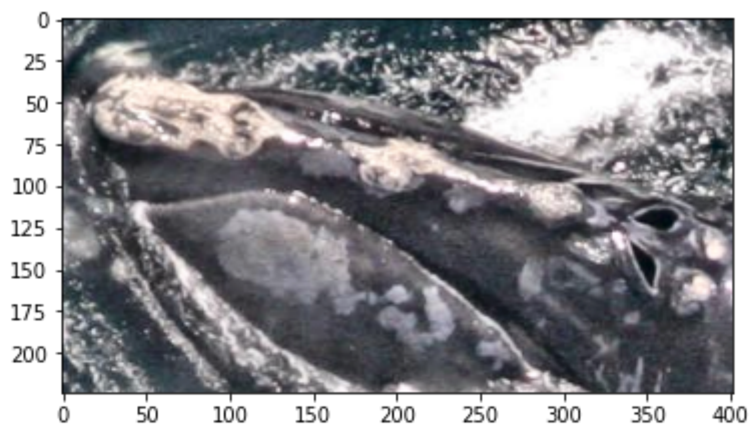
The histogram of gradients was not able to capture the callosity pattern at the top of the heas of thw whale. This is probably because of a lot of noise at the head region of the whale due to glare of sun and water splashes.

## Increase Contrast

I thought increasing the contrast of the image will give us better results as it will make the dark regions darker and bright regions brighter. So we can better see the head.

In [21]:
```
clahe = cv2.createCLAHE(clipLimit=2., tileGridSize=(8,8))

lab = cv2.cvtColor(crop_img, cv2.COLOR_BGR2LAB)  # convert from BGR to LAB color space
l, a, b = cv2.split(lab)  # split on 3 different channels

l2 = clahe.apply(l)  # apply CLAHE to the L-channel

lab = cv2.merge((l2,a,b))  # merge channels
contrast_image = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
```

In [22]:
```
plt.imshow(contrast_image)
plt.show()
```

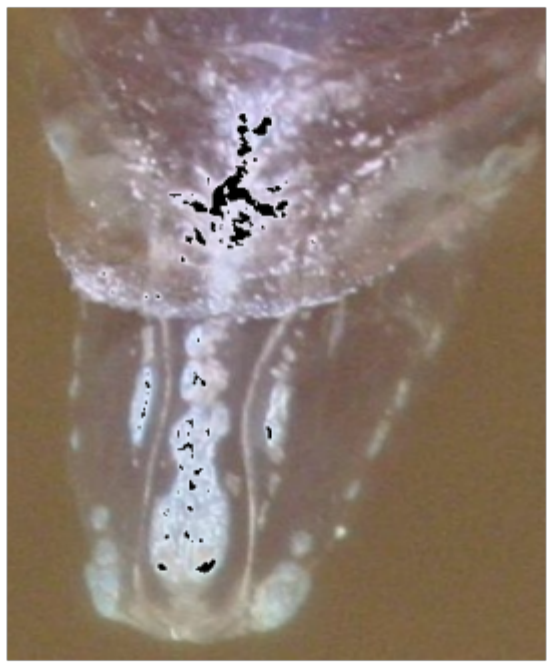


Visually it did not seem to have that much of a difference.

## Changing white pixels to eliminate glare

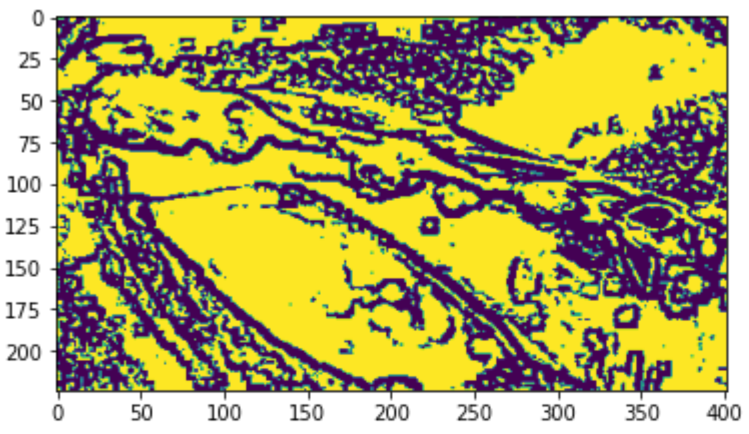
I thought that changing white pixels to black in the image would result in reducing the glare and waves, but it turns out that a lot of white pixels in the trunk are as white or whiter than the glare. So this method would not work as expected.

In [257]:
```
gray = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2GRAY)

plt.figure(figsize=(6,6))
ret, thresh = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY)

cropped_image[thresh == 255] = 0
cropped_image[thresh == 254] = 0
cropped_image[thresh == 253] = 0

plt.imshow(cropped_image)
plt.axis("off")

plt.show()
```



## Morphological transformation to remove noise

In [268]:
```
image=cv2.cvtColor(crop_img,cv2.COLOR_BGR2GRAY)
se=cv2.getStructuringElement(cv2.MORPH_RECT , (8,8))
bg=cv2.morphologyEx(image, cv2.MORPH_DILATE, se)
out_binary=cv2.threshold(image, 0, 255, cv2.THRESH_OTSU )[1]

plt.imshow(out_binary)

plt.show()
```



It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. It did not work quite well as there was still noise in the image after morphological transformation

## SIFT Features

In [275]:
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

MIN_MATCH_COUNT = 4

img1 = cv.imread('/Users/pratikaher/FALL22/CV/Whale/investigation/24458/cropped2/w_4611.jpg', 0)  # qu
img2 = cv.imread('/Users/pratikaher/FALL22/CV/Whale/investigation/24458/cropped2/w_5271.jpg', 0)  # tr

# Initiate SIFT detector
sift = cv.SIFT_create()
# find the key points and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
```

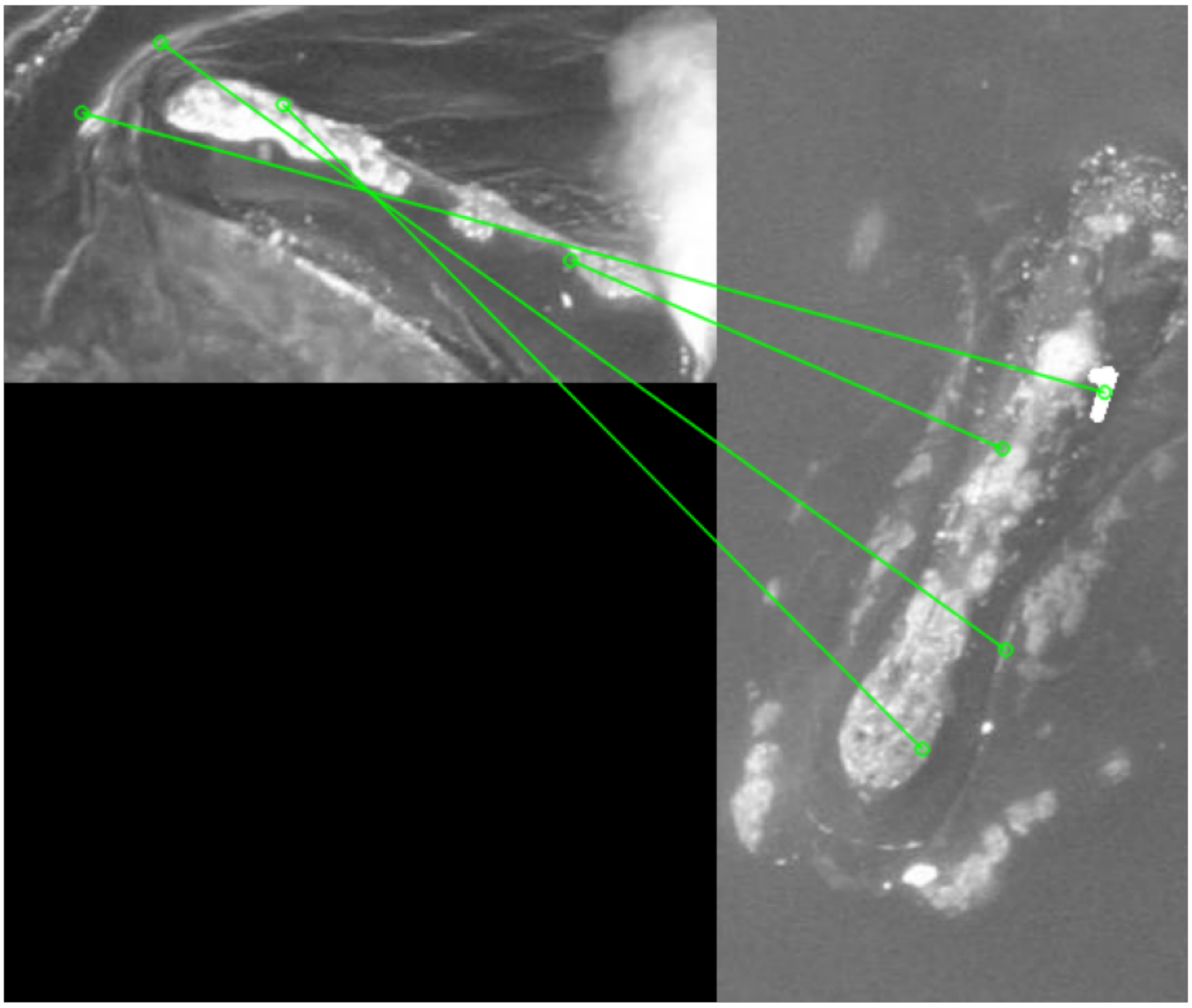
```
(181, 342) (478, 226)
```

In [276]:
```
index_params = dict(algorithm=1, trees=5)
search_params = dict(checks=50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1, des2, k=2)
```

In [277]:
```
# store all the good matches as per Lowe's ratio test.
good = []
for m, n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
    if len(good) >= MIN_MATCH_COUNT:
        src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)
        M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
        matchesMask = mask.ravel().tolist()
        h, w = img1.shape
        pts = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
        dst = cv.perspectiveTransform(pts, M)
        img2 = cv.polylines(img2, [np.int32(dst)], True, 255, 3, cv.LINE_AA)

```

In [278]:
```
draw_params = dict(matchColor=(0, 255, 0),  # draw matches in green color
                   singlePointColor=None,
                   matchesMask=matchesMask,  # draw only in-liers
                   flags=2)
img3 = cv.drawMatches(img1, kp1, img2, kp2, good, None, **draw_params)
```

In [279]:
```python
plt.figure(figsize=(15,15))
plt.imshow(img3, 'gray')
plt.axis('off')
plt.show()
```



I found that sift features do not match well between two images of the same whale.

## Experiment with saturation values

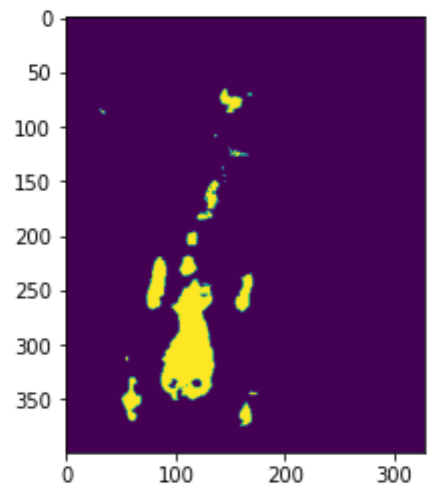
By experimening with Hue-Saturation Value (HSV), value of the images, I found a combination of values that helps detect the shape of the pattern on the head of the whale in a lot of images.

In [156]:
```python
cropped_image = cv2.imread('/Users/pratikaher/FALL22/CV/Whale/whale_samples/cropped/whale_48813/w_934.
```

In [151]:
```python
hsv = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2HSV)
```

In [154]:
```python
h = hsv[:,:,0]
s = hsv[:,:,1]
v = hsv[:,:,2]

HSV_MIN = np.array([1, 1, 1],np.uint8)
HSV_MAX = np.array([50, 255, 255],np.uint8)

frame_threshold = cv2.inRange(hsv, HSV_MIN, HSV_MAX)
```

In [155]:
```python
plt.imshow(frame_threshold)
plt.show()
```



I could see a that HSV images were quite helpful to detect the shape of callousity patterns in a lot images. So I decide to use the HSV images for classifier to classify images to classes. So we convert all the cropped images to HSV images.

In [2]:
```python
cropped_path = '/Users/pratikaher/FALL22/CV/Whale/whale_samples/cropped/'
```

In [14]:
```python
# convert cropped images to HSV images
for more_dirs in os.listdir(cropped_path):

    if more_dirs == '.DS_Store':
        continue

    for file in os.listdir(cropped_path+more_dirs):

        if file == '.DS_Store' or not file.endswith('.jpg'):
            continue

        final_file = cropped_path+more_dirs+'/'+file


#         print(final_file)
        # read image
        read_cropped_image = cv2.imread(final_file)

        # convert img to hsv
        hsv = cv2.cvtColor(read_cropped_image, cv2.COLOR_BGR2HSV)
        h = hsv[:,:,0]
        s = hsv[:,:,1]
        v = hsv[:,:,2]

        HSV_MIN = np.array([1, 5, 1],np.uint8)
        HSV_MAX = np.array([50, 255, 255],np.uint8)

        # hsv_img = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

        frame_threshed = cv2.inRange(hsv, HSV_MIN, HSV_MAX)

        newpath = cropped_path+more_dirs+'/saturation/'

        if not os.path.exists(newpath):
            os.makedirs(newpath)

        cv2.imwrite(newpath+file, frame_threshed)


#         f, axarr = plt.subplots(1,2, figsize=(10,10))
#         plt.figure(figsize=(8,8))
#         plt.imshow(read_cropped_image)
#         plt.imshow(frame_threshed)

#         axarr[0].imshow(read_cropped_image)
#         axarr[1].imshow(frame_threshed)

#         # plt.imshow(h)
#         # plt.imshow(cropped_image)

#         plt.axis('off')
#         plt.show()
```

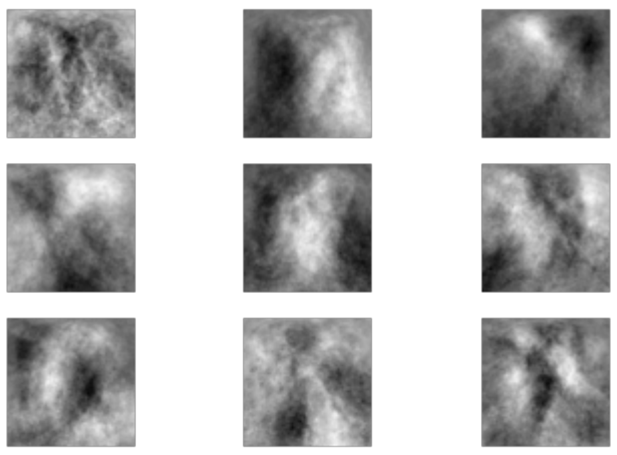
## PCA Decomposition

In [7]:
```python

PATH = '/Users/pratikaher/FALL22/CV/Whale/whale_samples/cropped/'
N = 0
last_N = 0
xdim = 256
ydim = 256
D = np.zeros((140,xdim*ydim))
y_val = 0
y = []

whale_HM = {}

for foldername in os.listdir(PATH):

    for file in os.listdir(PATH+foldername):

        if( file.endswith(".jpg") ):

            img = cv2.imread(PATH + foldername + "/" + file)
            img = cv2.resize(img, (256, 256))

            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            sz = img.shape
#             print(sz)
            D[N,:] = img.flatten() # flatten image and pack into data matrix rows

            if(False):
                plt.figure(figsize=(xdim/72,ydim/72))
                plt.imshow(img, cmap='gray') # display the image
                plt.axis('off')
                plt.show()
                sleep(0.01)
                clear_output(wait=True)
            N = N + 1

            y.append(y_val)


    whale_HM[range(last_N, N+1)] = foldername
    last_N = N
    y_val += 1

    print(foldername, N)

D = D.T # make images (num_images x pixels) in size
print( "loaded " + str(N) + " images" )
```

```
whale_48813 18
whale_65586 41
whale_36851 67
whale_38681 108
whale_28892 140
loaded 140 images
```

In [9]:
```python
mu = np.mean(D, axis = 1)
mu = np.matlib.repmat(mu, N, 1)
D  = D - mu.T
C2 = D.T@D;
val,vec = linalg.eig(C2);
```

In [10]:
```python
for k in range(9):
    e = D@vec[:,k]
    plt.subplot(3,3,k+1);
    plt.imshow( e.reshape(xdim,ydim), cmap='gray');
    plt.axis('off')
```



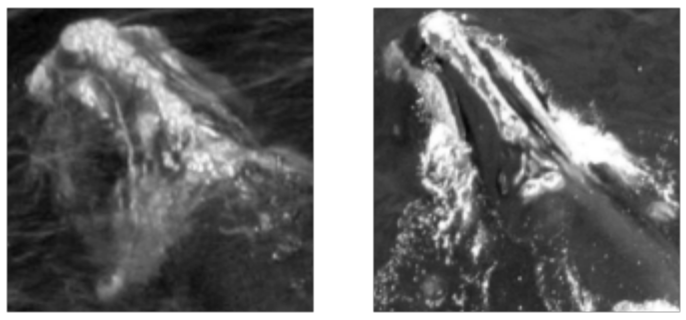
In [11]:
```python
d = 3
P = np.zeros((d,N))
for k in range(d):
    e = D@vec[:,k]
    P[k,:] = e.T@D
```

In [23]:
```python
def get_whale_from_photo_number(photo_num):

    return [whale_HM[key] for key in whale_HM if photo_num in key][0]
```

In [13]:
```python
Fnew  = D + mu.T

j = 10
mindist = 1e10
for k in range(N):
    if( j != k ):
        dist = linalg.norm(P[:,k]-P[:,j])
        if( dist < mindist ):
            mindist = dist
            minind = k

print(j, minind)
print(get_whale_from_photo_number(j), get_whale_from_photo_number(minind))

plt.subplot(121);
plt.imshow( Fnew[:,j].reshape(xdim,ydim), cmap='gray'); plt.axis('off');
plt.subplot(122);
plt.imshow( Fnew[:,minind].reshape(xdim,ydim), cmap='gray'); plt.axis('off')
```

```
10 27
whale_48813 whale_65586
```

Out[13]:
```
(-0.5, 255.5, 255.5, -0.5)
```



### PCA to classify images into categories

We use PCA projected images to find the closest image to a given image.

In [14]:
```python
from collections import defaultdict
```

In [15]:
```python
Fnew  = D + mu.T

success = 0
HM = defaultdict(int)

for j in range(N):

    mindist = 1e10
    for k in range(N):
        if( j != k ):
            dist = linalg.norm(P[:,k]-P[:,j])
            if( dist < mindist ):
                mindist = dist
                minind = k

    if get_whale_from_photo_number(j) == get_whale_from_photo_number(minind):
        success += 1

print("Accuracy",success/N)
```

```
Accuracy 0.2642857142857143
```

PCA does not give very good accuracy on the saturation images as the images are not will algined and the whale faces are at a different position in the frame.

## TSNE

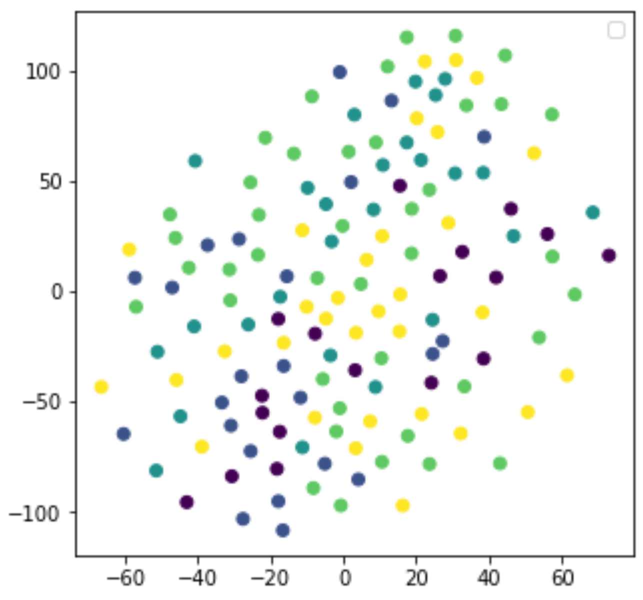
In [80]:
```python
from matplotlib import cm
# import numpy as np
from sklearn import datasets
# from sklearn.manifold import TSNE
```

In [12]:
```python
# TSNE (dimensionality reduction)
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=0)
X    = tsne.fit_transform(D.T) # 2D representation
# y    = range(len(digits.target_names)) # labels for visual
```

```
/Users/pratikaher/opt/anaconda3/envs/cv/lib/python3.10/site-packages/sklearn/manifold/_t_sne.py:800: Futu
reWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
/Users/pratikaher/opt/anaconda3/envs/cv/lib/python3.10/site-packages/sklearn/manifold/_t_sne.py:810: Futu
reWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  warnings.warn(
```

In [13]:
```python
plt.figure(figsize=(5, 5))

plt.scatter(X[:, 0], X[: ,1], c=y)

plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore ar
e ignored when legend() is called with no argument.
```



We can see some association in the graph. But overall, we can see that scatter plot is all over the place for TSNE. This is because, as with PCA, the faces of the whales are not perfectly aligned.

## Classification

### SVM classifier

We use SVM on the pixel value of HSV images to classify images into respective categories.

In [1]:
```python
from skimage.transform import resize
from sklearn.metrics import accuracy_score
from skimage.io import imread
```

In [2]:
```python
cropped_path = '/Users/pratikaher/FALL22/CV/Whale/whale_samples/cropped/'
```

In [5]:
```python
flat_data_arr = []
target_arr = []

category_index = 0

for more_dirs in os.listdir(cropped_path):

    for file in os.listdir(cropped_path+more_dirs+"/saturation"):

        newpath = cropped_path+more_dirs + "/" + file
        img_array = imread(newpath)
        img_resized=resize(img_array,(256,256,3))
        flat_data_arr.append(img_resized.flatten())
        target_arr.append(category_index)

    category_index += 1
```

In [6]:
```
flat_data=np.array(flat_data_arr)
target=np.array(target_arr)
df=pd.DataFrame(flat_data)
df['Target']=target
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

In [7]:
```
from sklearn import svm
from sklearn.model_selection import GridSearchCV
param_grid={'C':[0.1,1,10,100],'gamma':[0.0001,0.001,0.1,1],'kernel':['rbf','poly']}
svc = svm.SVC(probability=True)
model = GridSearchCV(svc,param_grid)
```

In [8]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=77,stratify=y, shuffle=
```

I am using grid seach to identify the best parameters for the SVM model.

In [9]:
```
model.fit(x_train,y_train)
```

Out[9]:
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [10]:
```
# Accuracy on the train set
accuracy_score(model.predict(x_train), y_train)
```

Out[10]:
```
1.0
```

We see that the model is overfitting beacuse the accuracy is 100% on the training set.

In [11]:
```
y_pred=model.predict(x_test)
```

In [12]:
```
print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
```

```
The model is 28.57142857142857% accurate
```

Let us try to see how the accuracy is with just two classes. We take images of two whales and name the whales alex and julia. We create a classifier that trains on the HSV values of two whales and use SVM classifer for testing.

In [25]:
```
import pandas as pd
import os
from skimage.transform import resize
from sklearn.metrics import accuracy_score
from skimage.io import imread
import numpy as np
import matplotlib.pyplot as plt

Categories = ['alex','julia']

datadir = '/Users/pratikaher/FALL22/CV/Whale/investigation/24458/outputs'

flat_data_arr = []
target_arr = []

for category in Categories:
    print(f'loading... category : {category}')
    path=os.path.join(datadir,category)
    for img in os.listdir(path):
        img_array = imread(os.path.join(path,img))
        img_resized=resize(img_array,(256,256,3))
        flat_data_arr.append(img_resized.flatten())
        target_arr.append(Categories.index(category))
```

```
loading... category : alex
loading... category : julia
```

In [26]:
```
flat_data=np.array(flat_data_arr)
target=np.array(target_arr)
df=pd.DataFrame(flat_data)
df['Target']=target
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

In [27]:
```python
from sklearn import svm
from sklearn.model_selection import GridSearchCV
param_grid={'C':[0.1,1,10,100],'gamma':[0.0001,0.001,0.1,1],'kernel':['rbf','poly']}
svc = svm.SVC(probability=True)
model = GridSearchCV(svc,param_grid)
```

In [28]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=77,stratify=y)
```

In [29]:
```python
model.fit(x_train,y_train)
```

Out[29]:
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [7]:
```python
y_pred=model.predict(x_test)
```

In [8]:
```python
print("The predicted Data is :")
print(y_pred)
print("The actual data is:")
print(np.array(y_test))
print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
```

```
The predicted Data is :
[1 1 1 1 1 1 1]
The actual data is:
[1 1 0 1 1 0 1]
The model is 71.42857142857143% accurate
```

## Image Transformers

CLIP stands for Contrastive Language-Image Pre-Training is trained on ImageNet dataset and can be used to build a classifier by training it with the set of our images. It is preferred for zero shot learning, as it can classify the things it has never seen before.

In [ ]:
```python
from sentence_transformers import SentenceTransformer, util
from PIL import Image
import glob
import os

# Load the OpenAI CLIP Model
print('Loading CLIP Model...')
model = SentenceTransformer('clip-ViT-B-32')
```

```
/Users/pratikaher/opt/anaconda3/envs/cv/lib/python3.10/site-packages/tqdm/auto.py:22: TqdmWarning: IProgr
ess not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user
_install.html (https://ipywidgets.readthedocs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm

Loading CLIP Model...
```

In [ ]:
```python
cropped_path = '/Users/pratikaher/FALL22/CV/Whale/whale_samples/cropped/'

image_names = []

for more_dirs in os.listdir(cropped_path):

    if more_dirs == '.DS_Store':
        continue

    image_names.extend(list(glob.glob(cropped_path+ more_dirs +'/*.jpg')))
```

In [ ]:
```python
image_data = list(zip(image_names, y))
```

In [47]:
```python
import random
random.shuffle(image_data)

test_size = int(len(image_data) * 0.2)
valid_size = int(len(image_data) * 0.1)
train_size = len(image_data) - test_size - valid_size


train_data = image_data[:train_size]
valid_data = image_data[train_size:train_size+valid_size]
test_data = image_data[-test_size:]
```

In [48]:
```
print("Images:", len(train_data))
encoded_images_train = model.encode([Image.open(filepath[0]) for filepath in train_data], batch_size=1
```

```
Images: 98

Batches: 100%|████████████████████████████████████████████████████████| 1/1
[00:06<00:00,  6.30s/it]
```

In [49]:
```
encoded_images_test = model.encode([Image.open(filepath[0]) for filepath in test_data], batch_size=128
```

```
Batches: 100%|████████████████████████████████████████████████████████| 1/1
[00:01<00:00,  1.97s/it]
```

Once we have the embeddings from CLIP model, we can use K-nearest neighbors classifier to classify the image embeddings obtained from the transformer.

In [50]:
```
from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(encoded_images_train, [e[1] for e in train_data])
```

Out[50]:
```
▾      KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

I experimented with various k values but 3 seems to give best results.

In [51]:
```
success = 0
for i in range(len(test_data)):

    test_prediction = neigh.predict(encoded_images_test[i].unsqueeze(0))

    if test_prediction[0] == test_data[i][1]:

        success += 1

print("Accuracy", success/len(test_data))
```

```
Accuracy 0.5357142857142857
```

## Limitations

Neural network based algorithms like CNN should work well to detect the callosity patterns of the whales. We can also experiment with different filters. It is really difficult to separate the features of the head of the whale from the water and the glare of the sun. HSV images are the closest I could come to getting the head patterns. With better object localization, it is possible that we can get the features of the head more clearly. I could also have expanded the labeled image set using some augmentation techniques. For every image, we would run it through some filters which are meant to signify the variance between the images of the set, and then use filter outputs as additional training data. Some of the filters we could use low pass filter (smoothing), high pass filter (edge detection or sharpening), and various affine transform such as rotation, scaling, and pixel-wise shift. It would be also interesting to see if we can use ensemble models each focusing on different features from pre-processing (e.g edge detection and HOG values).

## Conclusion

I can see good accuracy with just two categories. When I include more categories their accuracy goes down as the difference between categories is very little because there are only ~30 examples to train from. This is in line with the existing methods, where others have also could achieve accuracy in the range of 50-65 percent. With classification methods like SVM, I can see that the model is overfitting to the training data. I conclude that, while I got reasonable results for two classes. It would be really useful to train a model on whale images with sufficient pre-processing techniques.