CS444-Whale and Dolphin Identification

In [16]:
```
!pip install -q bbox-utility
```

```
WARNING: Running pip as the 'root' user can result in broken permissions and confl
icting behaviour with the system package manager. It is recommended to use a virtu
al environment instead: https://pip.pypa.io/warnings/venv
```

Import Libraries

In [17]:
```
import numpy as np
import pandas as pd
import os
import glob
import shutil
from tqdm.notebook import tqdm
from joblib import Parallel, delayed

from bbox.utils import yolo2voc, draw_bboxes

import cv2
import matplotlib.pyplot as plt
```

Config

In [18]:
```
BASE_PATH = '../input/happywhale-boundingbox-yolov5-dataset'
IMG_SIZE = (768, 768) # new image resolution
CONF = 0.6 # confidence threshold for bbox
```

Meta Data

In [19]:
```
df = pd.read_csv(BASE_PATH+'/train.csv')
df['label_path'] = df['label_path'].map(lambda x: x.replace('/kaggle/working',BASE_

test_df = pd.read_csv(BASE_PATH+'/test.csv')
test_df['label_path'] = test_df['label_path'].map(lambda x: x.replace('/kaggle/wor
```

Pseudo Labelling

In [20]:
```
df = df.fillna('[]')
test_df = test_df.fillna('[]')
```

In [21]:
```
df['bbox'] = df['bbox'].map(eval)
test_df['bbox'] = test_df['bbox'].map(eval)

df['conf'] = df['conf'].map(eval)
test_df['conf'] = test_df['conf'].map(eval)
```

Crop Function

In [22]:
```
def load_image(path):
    return cv2.imread(path)[...,::-1]

np.random.seed(32)
colors = [(np.random.randint(255), np.random.randint(255), np.random.randint(255))
          for idx in range(1)]

def crop_image(row):
    image_path = row['image_path']
    if 'train' in image_path:
        save_dir = '/tmp/train_images'
    else:
        save_dir = '/tmp/test_images'
    img = load_image(image_path)
    if len(row['bbox']): # if there is no bbox
        bbox = row['bbox'][0]
        conf = row['conf'][0]
        if conf>=CONF: # don't crop for poor confident bboxes
            xmin, ymin, xmax, ymax = bbox
            img = img[ymin:ymax, xmin:xmax] # crop image
    img = cv2.resize(img[...,::-1], dsize=IMG_SIZE, interpolation=cv2.INTER_AREA)
    cv2.imwrite(f'{save_dir}/{row.image_id}', img) # save image in the new directo
    return
```

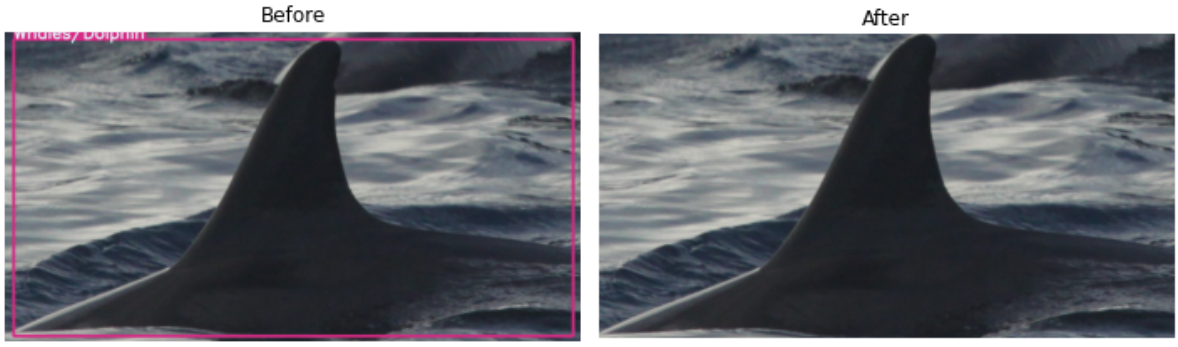
Visualization

Train Images

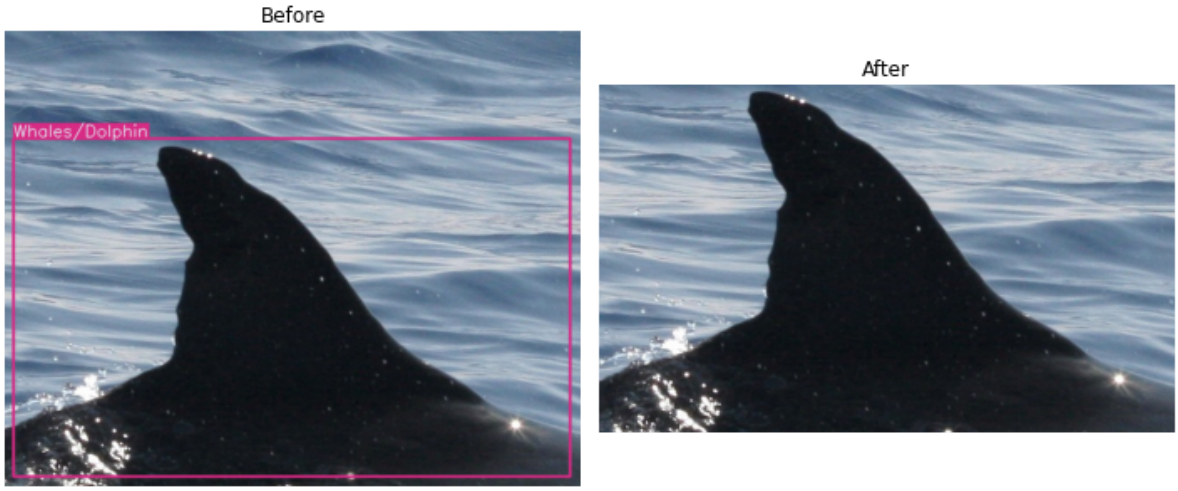
In [23]:
```
for i in range(10):
    row = df.sample(frac=1.0).iloc[i]
    img = load_image(row.image_path)
    bbox = row['bbox'][0]
    xmin, ymin, xmax, ymax = bbox

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
#     plt.imshow(img)
    dim = np.sqrt(np.prod(img.shape[:2]))
    line_thickness = int(2/512*dim)
    plt.imshow(
            draw_bboxes(
                img=img,
                bboxes=np.array(row['bbox']),
                classes=['Whales/Dolphin'],
                class_ids=[0],
                class_name=True,
                colors=colors,
                bbox_format="voc",
                line_thickness=line_thickness,
            ))
    plt.title('Before')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(img[ymin:ymax, xmin:xmax])
    plt.title('After')
    plt.axis('off')
```

```
    plt.suptitle(f'id: {row.image_id}', y=0.94)
    plt.tight_layout()
    plt.show()
```

id: 85b9ce1ec42e7e.jpg



id: b7ed326f220915.jpg



id: fc0a61b545d2cf.jpg

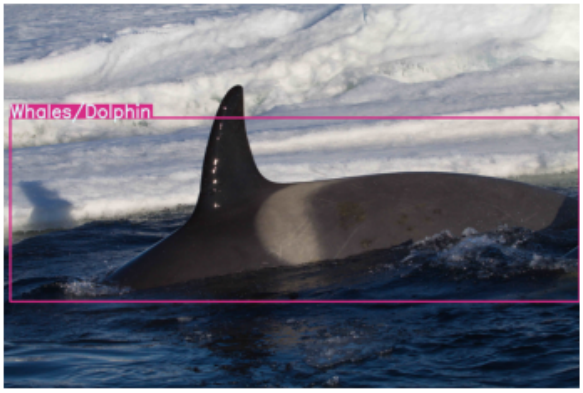
id: dbf8332c5ad1ad.jpg

Before

After

id: db571a3686e9ae.jpg

Before



After

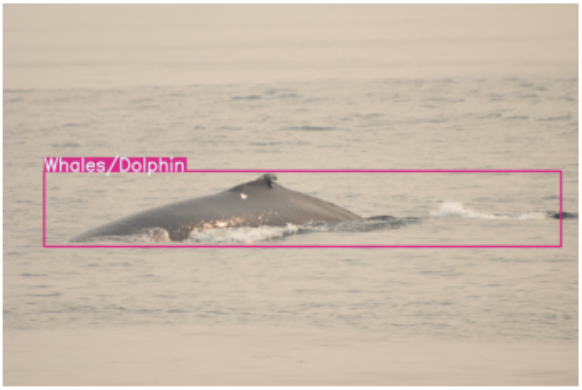
id: dbc1998b018654.jpg

Before

After

id: 3807c71bbe7a3f.jpg

Before



After

id: 6bd37d38afaeba.jpg

Before

After

id: 0e1d3a38b07c50.jpg

Before

After

id: 74c5afd86fb9f0.jpg

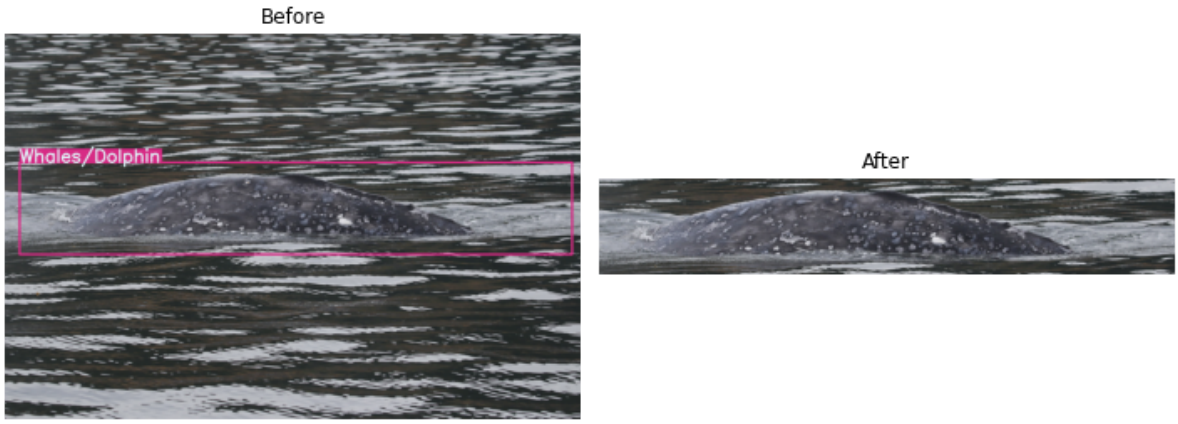
Before

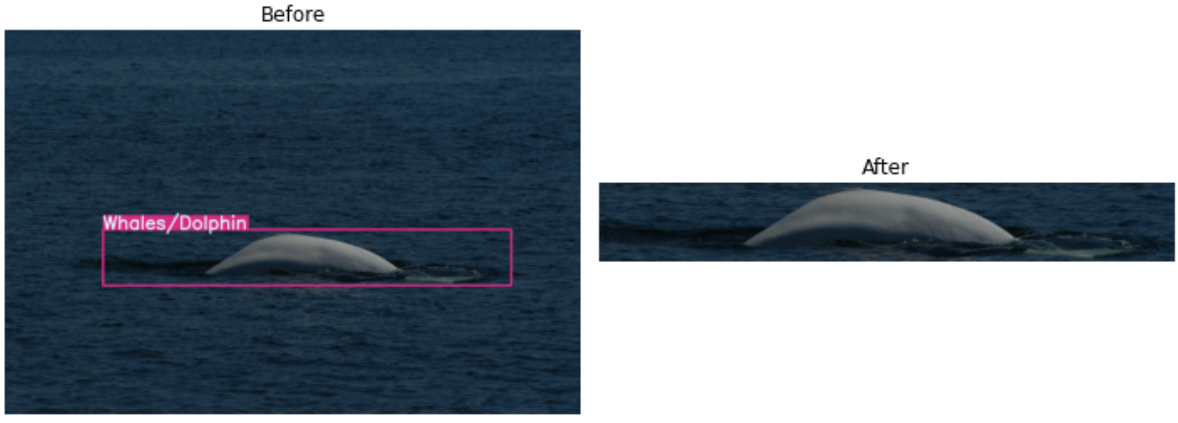
After

Test

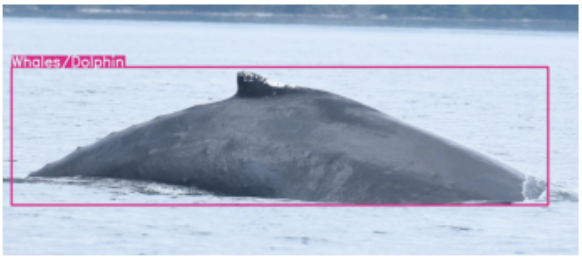
In [24]:
```
for i in range(10):
    row = test_df.iloc[i]
    img = load_image(row.image_path)
    bbox = row['bbox'][0]
    xmin, ymin, xmax, ymax = bbox

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
#     plt.imshow(img)
    dim = np.sqrt(np.prod(img.shape[:2]))
    line_thickness = int(2/512*dim)
    plt.imshow(
            draw_bboxes(
                img=img,
                bboxes=np.array(row['bbox']),
                classes=['Whales/Dolphin'],
```

```
                class_ids=[0],
                class_name=True,
                colors=colors,
                bbox_format="voc",
                line_thickness=line_thickness,
            ))
    plt.title('Before')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(img[ymin:ymax, xmin:xmax])
    plt.title('After')
    plt.axis('off')

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





Before



After

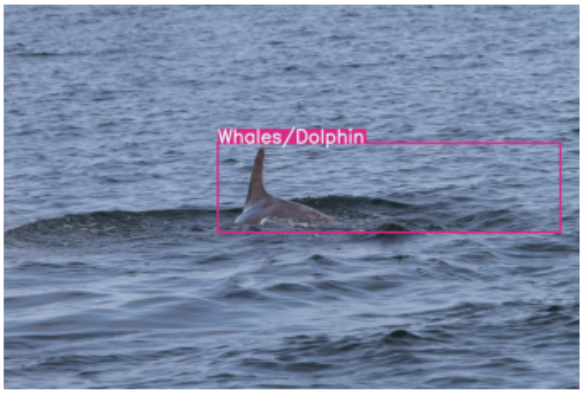
Before

After

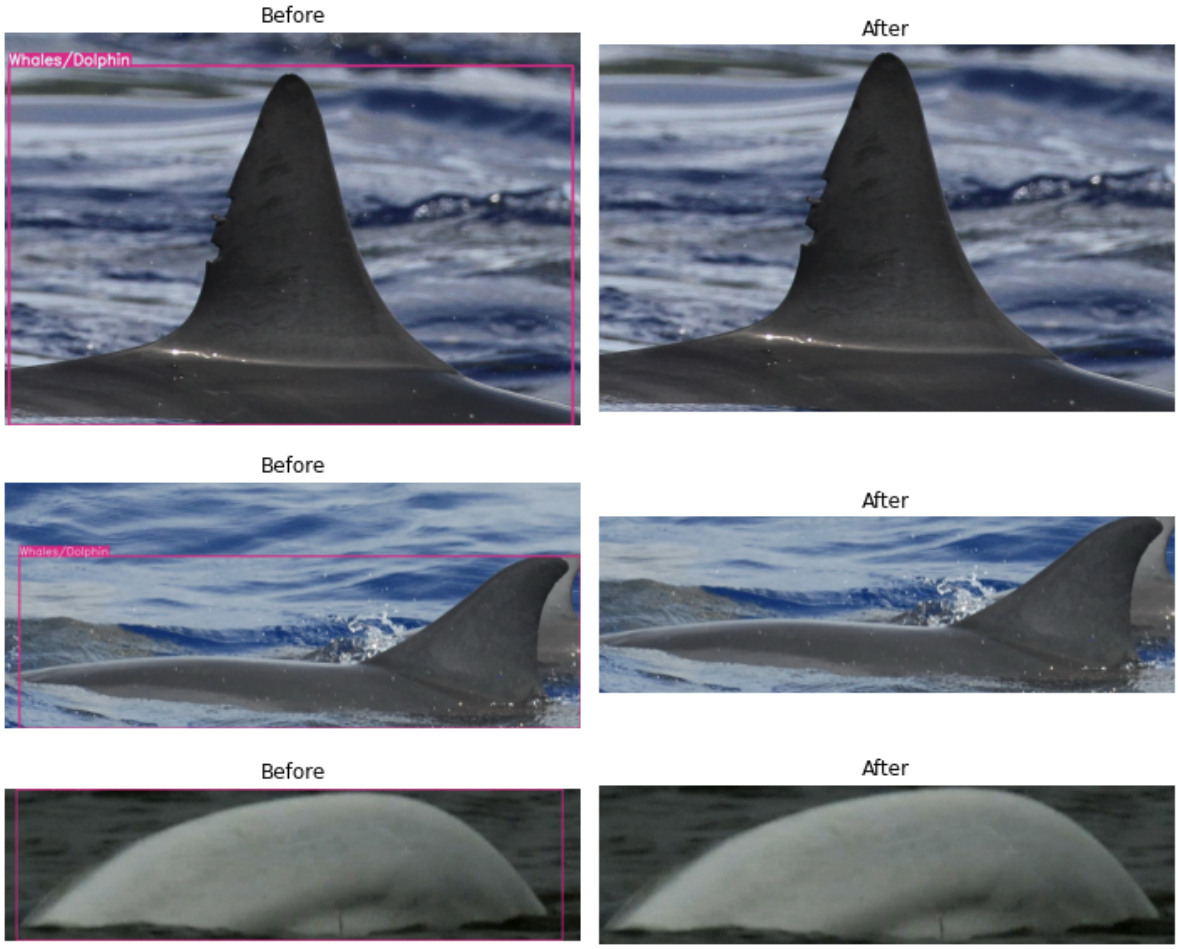
Before

After

Before



After



Create Save Directory

In [25]:
```
!mkdir -p /tmp/train_images && mkdir -p /tmp/test_images
```

Crop

In [26]:
```
# Train
_ = Parallel(n_jobs=-1, backend='threading')(delayed(crop_image)(row)\
                                             for _, row in tqdm(df.iterrows(), total=l

# Test
_ = Parallel(n_jobs=-1, backend='threading')(delayed(crop_image)(row)\
                                             for _, row in tqdm(test_df.iterrows(), to
```

```
train :   0%|          | 0/51033 [00:00<?, ?it/s]
```

Save Meta Data

In [27]:
```
df.to_csv('train.csv',index=False)
test_df.to_csv('test.csv',index=False)
```

Archive Files

In [ ]:
```
# Train
shutil.make_archive(base_name='/kaggle/working/train_images',
                    format='zip',
                    root_dir='/tmp/',
                    base_dir='train_images')
# Test
shutil.make_archive(base_name='/kaggle/working/test_images',
                    format='zip',
                    root_dir='/tmp/',
                    base_dir='test_images')
```

In [47]:
```
import os
IS_COLAB = not os.path.exists('/kaggle/input')
print(IS_COLAB)
```

```
False
```

In [48]:
```
import tensorflow as tf
try:
    # TPU detection. No parameters necessary if TPU_NAME environment variable is
    # set: this is always the case on Kaggle.
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.TPUStrategy(tpu)
else:
    # Default distribution strategy in Tensorflow. Works on CPU and single GPU.
    strategy = tf.distribute.get_strategy()

AUTO = tf.data.experimental.AUTOTUNE
print("REPLICAS: ", strategy.num_replicas_in_sync)
```

```
Running on TPU  grpc://10.0.0.2:8470
REPLICAS:  8
```

In [49]:
```
if IS_COLAB:
  from google.colab import drive
  drive.mount('/content/drive')
else:
  from kaggle_datasets import KaggleDatasets
```

In [50]:
```
!pip install -q efficientnet
!pip install tensorflow_addons
import re
import os
import numpy as np
import pandas as pd
import random
import math
import tensorflow as tf
import efficientnet.tfkeras as efn
from sklearn import metrics
from sklearn.model_selection import KFold, train_test_split
from tensorflow.keras import backend as K
import tensorflow_addons as tfa
from tqdm.auto import tqdm
import matplotlib.pyplot as plt
import pickle
import json
import tensorflow_hub as tfhub
from datetime import datetime
```

```
WARNING: Running pip as the 'root' user can result in broken permissions and confl
icting behaviour with the system package manager. It is recommended to use a virtu
al environment instead: https://pip.pypa.io/warnings/venv
Requirement already satisfied: tensorflow_addons in /opt/conda/lib/python3.10/site
-packages (0.20.0)
Requirement already satisfied: packaging in /opt/conda/lib/python3.10/site-package
s (from tensorflow_addons) (21.3)
Requirement already satisfied: typeguard<3.0.0,>=2.7 in /opt/conda/lib/python3.10/
site-packages (from tensorflow_addons) (2.13.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/python3.
10/site-packages (from packaging->tensorflow_addons) (3.0.9)
WARNING: Running pip as the 'root' user can result in broken permissions and confl
icting behaviour with the system package manager. It is recommended to use a virtu
al environment instead: https://pip.pypa.io/warnings/venv
```

Image Serialisation and Config

In [ ]:

```python
import os
import numpy as np
import tensorflow as tf
from PIL import Image

# Define the directory containing the PNG images
png_dir = '/kaggle/input/boxed-whales/train_images/train_images'

# Define the output directory for the TFRecord files
tfrec_dir = '/kaggle/working/'

# Define the number of folds
num_folds = 10

# Divide the dataset into folds
png_files = [os.path.join(png_dir, f) for f in os.listdir(png_dir) if f.endswith('
num_images = len(png_files)
fold_size = num_images // num_folds
image_folds = [png_files[i:i+fold_size] for i in range(0, num_images, fold_size)]

# Define a function to load and preprocess each image
def load_and_preprocess_image(filename):
    image = Image.open(filename)
    image = image.resize((224, 224)) # Resize the image to a fixed size
    image = np.asarray(image) # Convert the image to a numpy array
    return image

# Loop over each fold
for fold_idx in range(num_folds):
    print(f"Processing fold {fold_idx+1}...")

    # Define the output TFRecord file for this fold
    tfrec_file = os.path.join(tfrec_dir, f"happywhale-2022-train-{fold_idx+1}.tfre

    # Serialize each image in the fold and write it to the TFRecord file
    with tf.io.TFRecordWriter(tfrec_file) as writer:
        for png_file in image_folds[fold_idx]:
            image = load_and_preprocess_image(png_file)
            serialized_image = tf.io.serialize_tensor(image)
            feature = {'image': tf.train.Feature(bytes_list=tf.train.BytesList(valu
            example = tf.train.Example(features=tf.train.Features(feature=feature)
            writer.write(example.SerializeToString())

print("Done!")
```

In [ ]:
```
import os
import numpy as np
import tensorflow as tf
from PIL import Image

# Define the directory containing the PNG images
png_dir = '/kaggle/input/boxed-whales/test_images/test_images'

# Define the output directory for the TFRecord files
tfrec_dir = '/kaggle/working/'

# Define the number of folds
num_folds = 10

# Divide the dataset into folds
png_files = [os.path.join(png_dir, f) for f in os.listdir(png_dir) if f.endswith('
num_images = len(png_files)
fold_size = num_images // num_folds
image_folds = [png_files[i:i+fold_size] for i in range(0, num_images, fold_size)]

# Define a function to load and preprocess each image
def load_and_preprocess_image(filename):
    image = Image.open(filename)
    image = image.resize((224, 224)) # Resize the image to a fixed size
    image = np.asarray(image) # Convert the image to a numpy array
    return image

# Loop over each fold
for fold_idx in range(num_folds):
    print(f"Processing fold {fold_idx+1}...")

    # Define the output TFRecord file for this fold
    tfrec_file = os.path.join(tfrec_dir, f"happywhale-2022-test-{fold_idx+1}.tfrec

    # Serialize each image in the fold and write it to the TFRecord file
    with tf.io.TFRecordWriter(tfrec_file) as writer:
        for png_file in image_folds[fold_idx]:
            image = load_and_preprocess_image(png_file)
            serialized_image = tf.io.serialize_tensor(image)
            feature = {'image': tf.train.Feature(bytes_list=tf.train.BytesList(valu
            example = tf.train.Example(features=tf.train.Features(feature=feature)
            writer.write(example.SerializeToString())

print("Done!")
```

In [51]:
```
save_dir = '.'
EXPERIMENT = 0
run_ts = datetime.now().strftime('%Y%m%d-%H%M%S')
print(run_ts)
if IS_COLAB:
    save_dir = f'/content/drive/MyDrive/Kaggle/HappyWhale-2022/experiments-{EXPERI
    !mkdir -p {save_dir}
```

```
20230512-021129
```

In [52]:
```
class config:


    SEED = 42
    FOLD_TO_RUN = 0#In this notebook, we do not train models
    FOLDS = 5
    DEBUG = False
    EVALUATE = True
```

```
    RESUME = False
    RESUME_EPOCH = None


    ### Dataset
    BATCH_SIZE = 32 * strategy.num_replicas_in_sync
    IMAGE_SIZE = 768
    N_CLASSES = 15587

    ### Model
    model_type = 'effnetv1'
    EFF_NET = 6
    EFF_NETV2 = 's-21k-ft1k'
    FREEZE_BATCH_NORM = False
    head = 'arcface'
    EPOCHS = 20
    LR = 0.001
    message='baseline'

    ### Augmentations
    CUTOUT = False

    ### Save-Directory
    save_dir = save_dir

    ### Inference
    KNN = 100

def count_data_items(filenames):
    n = [int(re.compile(r"-([0-9]*)\.").search(filename).group(1))
         for filename in filenames]
    return np.sum(n)

# Function to seed everything
def seed_everything(seed):
    random.seed(seed)
    np.random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    tf.random.set_seed(seed)

def is_interactive():
    return 'runtime'    in get_ipython().config.IPKernelApp.connection_file
IS_INTERACTIVE = is_interactive()
print(IS_INTERACTIVE)
```

```
True
```

In [53]:
```
MODEL_NAME = None
if config.model_type == 'effnetv1':
    MODEL_NAME = f'effnetv1_b{config.EFF_NET}'
elif config.model_type == 'effnetv2':
    MODEL_NAME = f'effnetv2_{config.EFF_NETV2}'

config.MODEL_NAME = MODEL_NAME
print(MODEL_NAME)
```

```
effnetv1_b6
```

In [ ]:
```
```

In [54]:
```
with open(config.save_dir+'/config.json', 'w') as fp:
    json.dump({x:dict(config.__dict__)[x] for x in dict(config.__dict__) if not x.
```

In [55]:
```
GCS_PATH = 'gs://kds-d916c3252bf3bc5b3500b904f05f51ce57c8df85221d11b7711bcda9'  # (
if not IS_COLAB:
    GCS_PATH = KaggleDatasets().get_gcs_path('happywhale-tfrecords-v1')

train_files = np.sort(np.array(tf.io.gfile.glob(GCS_PATH + '/happywhale-2022-train
test_files = np.sort(np.array(tf.io.gfile.glob(GCS_PATH + '/happywhale-2022-test*.
print(GCS_PATH)
print(len(train_files),len(test_files),count_data_items(train_files),count_data_it
```

```
gs://kds-c09fe7f72571cea8f04365f1d5565b3534972f0d69d72e43c05dab9e
10 10 51033 27956
```

Data

In [56]:
```
def arcface_format(posting_id, image, label_group, matches):
    return posting_id, {'inp1': image, 'inp2': label_group}, label_group, matches

def arcface_inference_format(posting_id, image, label_group, matches):
    return image,posting_id

def arcface_eval_format(posting_id, image, label_group, matches):
    return image,label_group

# Data augmentation function
def data_augment(posting_id, image, label_group, matches):

    ### CUTOUT
    if tf.random.uniform([])>0.5 and config.CUTOUT:
      N_CUTOUT = 6
      for cutouts in range(N_CUTOUT):
        if tf.random.uniform([])>0.5:
           DIM = config.IMAGE_SIZE
           CUTOUT_LENGTH = DIM//8
           x1 = tf.cast( tf.random.uniform([],0,DIM-CUTOUT_LENGTH),tf.int32)
           x2 = tf.cast( tf.random.uniform([],0,DIM-CUTOUT_LENGTH),tf.int32)
           filter_ = tf.concat([tf.zeros((x1,CUTOUT_LENGTH)),tf.ones((CUTOUT_LENGTH
           filter_ = tf.concat([tf.zeros((DIM,x2)),filter_,tf.zeros((DIM,DIM-x2-CU
           cutout = tf.reshape(1-filter_,(DIM,DIM,1))
           image = cutout*image

    image = tf.image.random_flip_left_right(image)
    # image = tf.image.random_flip_up_down(image)
    image = tf.image.random_hue(image, 0.01)
    image = tf.image.random_saturation(image, 0.70, 1.30)
    image = tf.image.random_contrast(image, 0.80, 1.20)
    image = tf.image.random_brightness(image, 0.10)
    return posting_id, image, label_group, matches

# Function to decode our images
def decode_image(image_data):
    image = tf.image.decode_jpeg(image_data, channels = 3)
    image = tf.image.resize(image, [config.IMAGE_SIZE,config.IMAGE_SIZE])
    image = tf.cast(image, tf.float32) / 255.0
    return image

# This function parse our images and also get the target variable
def read_labeled_tfrecord(example):
    LABELED_TFREC_FORMAT = {
        "image_name": tf.io.FixedLenFeature([], tf.string),
        "image": tf.io.FixedLenFeature([], tf.string),
        "target": tf.io.FixedLenFeature([], tf.int64),
#         "matches": tf.io.FixedLenFeature([], tf.string)
```

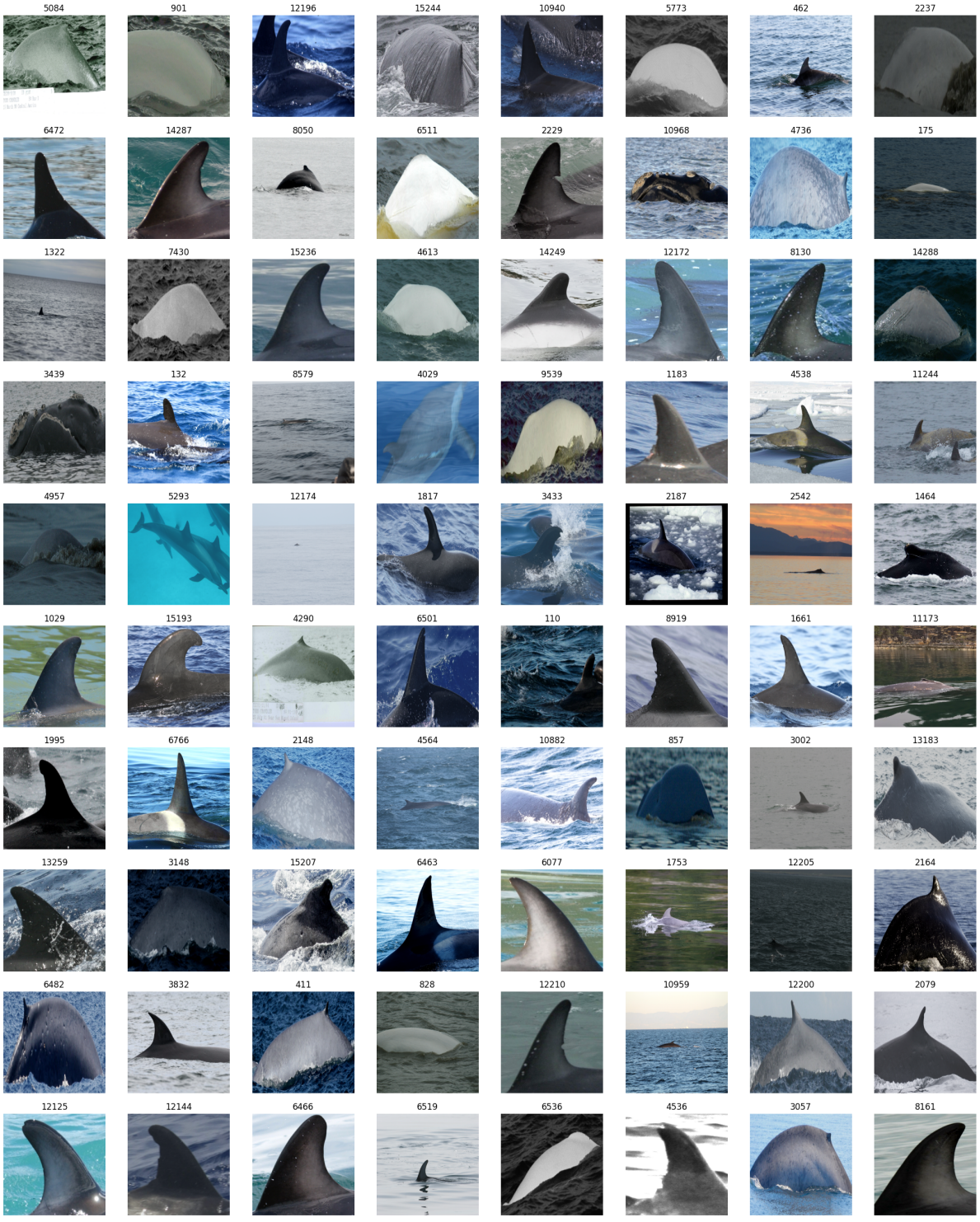
```
    }

    example = tf.io.parse_single_example(example, LABELED_TFREC_FORMAT)
    posting_id = example['image_name']
    image = decode_image(example['image'])
#     label_group = tf.one_hot(tf.cast(example['label_group'], tf.int32), depth = 
    label_group = tf.cast(example['target'], tf.int32)
#     matches = example['matches']
    matches = 1
    return posting_id, image, label_group, matches

# This function loads TF Records and parse them into tensors
def load_dataset(filenames, ordered = False):

    ignore_order = tf.data.Options()
    if not ordered:
        ignore_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads = AUTO)
#     dataset = dataset.cache()
    dataset = dataset.with_options(ignore_order)
    dataset = dataset.map(read_labeled_tfrecord, num_parallel_calls = AUTO)
    return dataset

# This function is to get our training tensors
def get_training_dataset(filenames):
    dataset = load_dataset(filenames, ordered = False)
    dataset = dataset.map(data_augment, num_parallel_calls = AUTO)
    dataset = dataset.map(arcface_format, num_parallel_calls = AUTO)
    dataset = dataset.map(lambda posting_id, image, label_group, matches: (image, 
    dataset = dataset.repeat()
    dataset = dataset.shuffle(2048)
    dataset = dataset.batch(config.BATCH_SIZE)
    dataset = dataset.prefetch(AUTO)
    return dataset

# This function is to get our training tensors
def get_val_dataset(filenames):
    dataset = load_dataset(filenames, ordered = True)
    dataset = dataset.map(data_augment, num_parallel_calls = AUTO)
    dataset = dataset.map(arcface_format, num_parallel_calls = AUTO)
    dataset = dataset.map(lambda posting_id, image, label_group, matches: (image, 
    dataset = dataset.batch(config.BATCH_SIZE)
    dataset = dataset.prefetch(AUTO)
    return dataset

# This function is to get our training tensors
def get_eval_dataset(filenames, get_targets = True):
    dataset = load_dataset(filenames, ordered = True)
    dataset = dataset.map(data_augment, num_parallel_calls = AUTO)
    dataset = dataset.map(arcface_eval_format, num_parallel_calls = AUTO)
    if not get_targets:
        dataset = dataset.map(lambda image, target: image)
    dataset = dataset.batch(config.BATCH_SIZE)
    dataset = dataset.prefetch(AUTO)
    return dataset

# This function is to get our training tensors
def get_test_dataset(filenames, get_names = True):
    dataset = load_dataset(filenames, ordered = True)
    dataset = dataset.map(data_augment, num_parallel_calls = AUTO)
    dataset = dataset.map(arcface_inference_format, num_parallel_calls = AUTO)
    if not get_names:
        dataset = dataset.map(lambda image, posting_id: image)
```

```
    dataset = dataset.batch(config.BATCH_SIZE)
    dataset = dataset.prefetch(AUTO)
    return dataset
```

In [57]:

```
row = 10; col = 8;
row = min(row,config.BATCH_SIZE//col)
N_TRAIN = count_data_items(train_files)
print(N_TRAIN)
ds = get_training_dataset(train_files)

for (sample,label) in ds:
    img = sample['inp1']
    plt.figure(figsize=(25,int(25*row/col)))
    for j in range(row*col):
        plt.subplot(row,col,j+1)
        plt.title(label[j].numpy())
        plt.axis('off')
        plt.imshow(img[j,])
    plt.show()
    break
print(img.shape)
```

```
51033
```

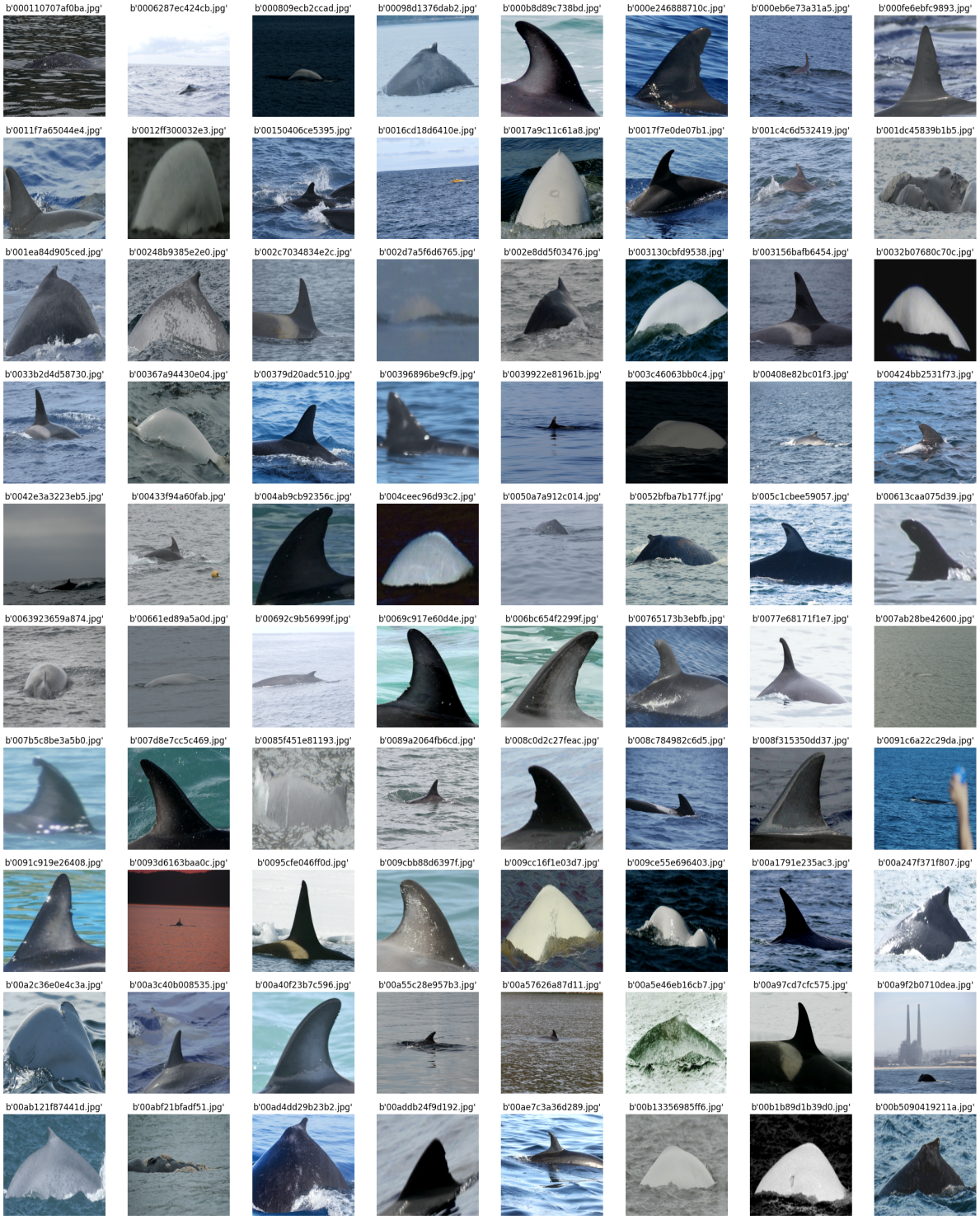


(256, 768, 768, 3)

In [58]:

```
row = 10; col = 8;
row = min(row,config.BATCH_SIZE//col)
N_TEST = count_data_items(test_files)
print(N_TEST)
ds = get_test_dataset(test_files)

for (img,label) in ds:
    plt.figure(figsize=(25,int(25*row/col)))
    for j in range(row*col):
        plt.subplot(row,col,j+1)
        plt.title(label[j].numpy())
        plt.axis('off')
        plt.imshow(img[j,])
    plt.show()
    break
print(img.shape)
```

27956



(256, 768, 768, 3)

Model

In [59]:

```
class ArcMarginProduct(tf.keras.layers.Layer):

    def __init__(self, n_classes, s=30, m=0.50, easy_margin=False,
                 ls_eps=0.0, **kwargs):

        super(ArcMarginProduct, self).__init__(**kwargs)

        self.n_classes = n_classes
        self.s = s
        self.m = m
        self.ls_eps = ls_eps
```

```
        self.easy_margin = easy_margin
        self.cos_m = tf.math.cos(m)
        self.sin_m = tf.math.sin(m)
        self.th = tf.math.cos(math.pi - m)
        self.mm = tf.math.sin(math.pi - m) * m

    def get_config(self):

        config = super().get_config().copy()
        config.update({
            'n_classes': self.n_classes,
            's': self.s,
            'm': self.m,
            'ls_eps': self.ls_eps,
            'easy_margin': self.easy_margin,
        })
        return config

    def build(self, input_shape):
        super(ArcMarginProduct, self).build(input_shape[0])

        self.W = self.add_weight(
            name='W',
            shape=(int(input_shape[0][-1]), self.n_classes),
            initializer='glorot_uniform',
            dtype='float32',
            trainable=True,
            regularizer=None)

    def call(self, inputs):
        X, y = inputs
        y = tf.cast(y, dtype=tf.int32)
        cosine = tf.matmul(
            tf.math.l2_normalize(X, axis=1),
            tf.math.l2_normalize(self.W, axis=0)
        )
        sine = tf.math.sqrt(1.0 - tf.math.pow(cosine, 2))
        phi = cosine * self.cos_m - sine * self.sin_m
        if self.easy_margin:
            phi = tf.where(cosine > 0, phi, cosine)
        else:
            phi = tf.where(cosine > self.th, phi, cosine - self.mm)
        one_hot = tf.cast(
            tf.one_hot(y, depth=self.n_classes),
            dtype=cosine.dtype
        )
        if self.ls_eps > 0:
            one_hot = (1 - self.ls_eps) * one_hot + self.ls_eps / self.n_classes

        output = (one_hot * phi) + ((1.0 - one_hot) * cosine)
        output *= self.s
        return output
```

In [60]:

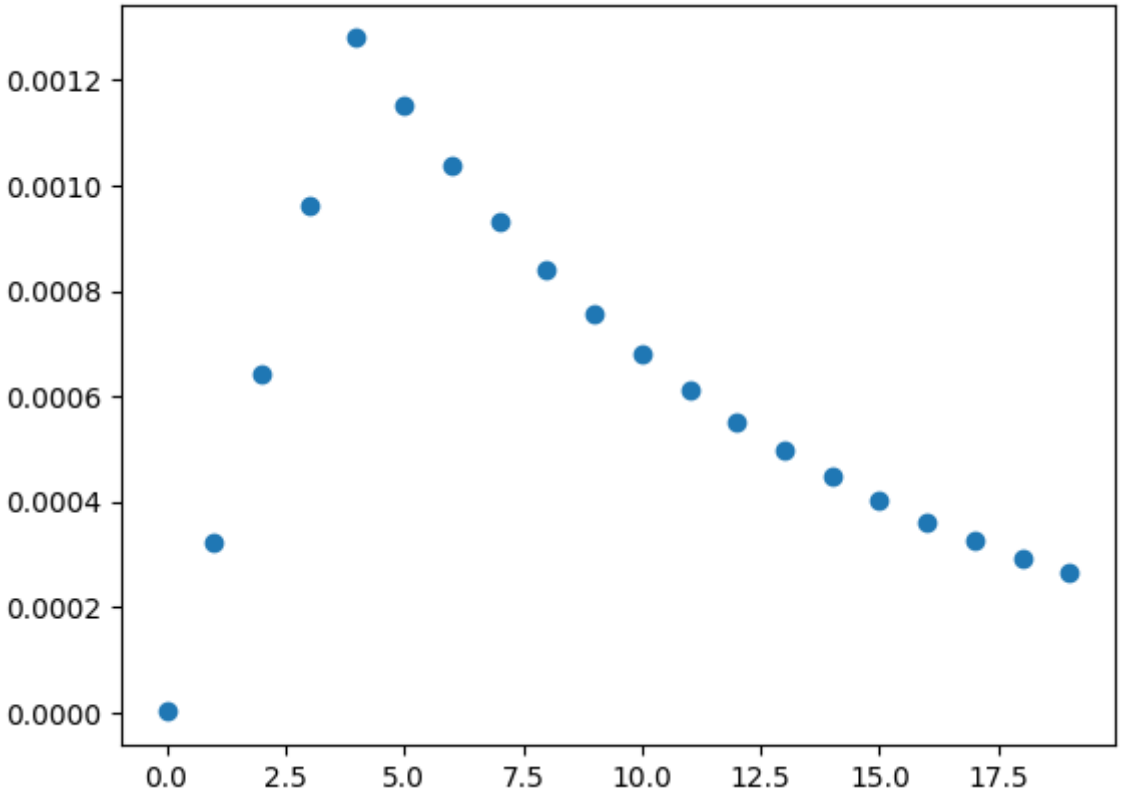
```
EFNS = [efn.EfficientNetB0, efn.EfficientNetB1, efn.EfficientNetB2, efn.EfficientN
        efn.EfficientNetB4, efn.EfficientNetB5, efn.EfficientNetB6, efn.EfficientN

def freeze_BN(model):
    for layer in model.layers:
        if not isinstance(layer, tf.keras.layers.BatchNormalization):
            layer.trainable = True
        else:
            layer.trainable = False
```

```
def get_model():

    if config.head=='arcface':
        head = ArcMarginProduct
    else:
        assert 1==2, "INVALID HEAD"

    with strategy.scope():

        margin = head(
            n_classes = config.N_CLASSES,
            s = 30,
            m = 0.3,
            name=f'head/{config.head}',
            dtype='float32'
            )

        inp = tf.keras.layers.Input(shape = [config.IMAGE_SIZE, config.IMAGE_SIZE,
        label = tf.keras.layers.Input(shape = (), name = 'inp2')

        if config.model_type == 'effnetv1':
            x = EFNS[config.EFF_NET](weights = 'noisy-student', include_top = Fals
            embed = tf.keras.layers.GlobalAveragePooling2D()(x)
        elif config.model_type == 'effnetv2':
            FEATURE_VECTOR = f'{EFFNETV2_ROOT}/tfhub_models/efficientnetv2-{config
            embed = tfhub.KerasLayer(FEATURE_VECTOR, trainable=True)(inp)

        embed = tf.keras.layers.Dropout(0.2)(embed)
        embed = tf.keras.layers.Dense(512)(embed)
        x = margin([embed, label])

        output = tf.keras.layers.Softmax(dtype='float32')(x)

        model = tf.keras.models.Model(inputs = [inp, label], outputs = [output])
        embed_model = tf.keras.models.Model(inputs = inp, outputs = embed)

        opt = tf.keras.optimizers.Adam(learning_rate = config.LR)
        if config.FREEZE_BATCH_NORM:
            freeze_BN(model)

        model.compile(
            optimizer = opt,
            loss = [tf.keras.losses.SparseCategoricalCrossentropy()],
            metrics = [tf.keras.metrics.SparseCategoricalAccuracy(),tf.keras.metri
            )
        print(model.summary())
        return model,embed_model
```

In [61]:

```
def get_lr_callback(plot=False):
    lr_start   = 0.000001
    lr_max     = 0.000005 * config.BATCH_SIZE
    lr_min     = 0.000001
    lr_ramp_ep = 4
    lr_sus_ep  = 0
    lr_decay   = 0.9

    def lrfn(epoch):
        if config.RESUME:
            epoch = epoch + config.RESUME_EPOCH
        if epoch < lr_ramp_ep:
            lr = (lr_max - lr_start) / lr_ramp_ep * epoch + lr_start

        elif epoch < lr_ramp_ep + lr_sus_ep:
```

```
            lr = lr_max

        else:
            lr = (lr_max - lr_min) * lr_decay**(epoch - lr_ramp_ep - lr_sus_ep) +

        return lr

    if plot:
        epochs = list(range(config.EPOCHS))
        learning_rates = [lrfn(x) for x in epochs]
        plt.scatter(epochs,learning_rates)
        plt.show()

    lr_callback = tf.keras.callbacks.LearningRateScheduler(lrfn, verbose=False)
    return lr_callback

get_lr_callback(plot=True)
```



Out[61]: <keras.callbacks.LearningRateScheduler at 0x79182cfbf2b0>

In [62]:
```
class Snapshot(tf.keras.callbacks.Callback):

    def __init__(self,fold,snapshot_epochs=[]):
        super(Snapshot, self).__init__()
        self.snapshot_epochs = snapshot_epochs
        self.fold = fold


    def on_epoch_end(self, epoch, logs=None):
        if epoch in self.snapshot_epochs: # your custom condition
            self.model.save_weights(config.save_dir+f"/EF{config.MODEL_NAME}_epoch
        self.model.save_weights(config.save_dir+f"/{config.MODEL_NAME}_last.h5")
```

In [ ]:

Train

In [63]:
```
TRAINING_FILENAMES = [x for i,x in enumerate(train_files) if i%config.FOLDS!=confi
VALIDATION_FILENAMES = [x for i,x in enumerate(train_files) if i%config.FOLDS==con
print(len(TRAINING_FILENAMES),len(VALIDATION_FILENAMES),count_data_items(TRAINING_
```

```
8 2 40826 10207
```

In [64]:
```
if config.DEBUG:
    TRAINING_FILENAMES = [TRAINING_FILENAMES[0]]
    VALIDATION_FILENAMES = [VALIDATION_FILENAMES[0]]
    print(len(TRAINING_FILENAMES),len(VALIDATION_FILENAMES),count_data_items(TRAIN
    test_files = [test_files[0]]
```

In [65]:
```
seed_everything(config.SEED)
VERBOSE = 1
train_dataset = get_training_dataset(TRAINING_FILENAMES)
val_dataset = get_val_dataset(VALIDATION_FILENAMES)
STEPS_PER_EPOCH = count_data_items(TRAINING_FILENAMES) // config.BATCH_SIZE
train_logger = tf.keras.callbacks.CSVLogger(config.save_dir+'/training-log-fold-%i
# SAVE BEST MODEL EACH FOLD
sv_loss = tf.keras.callbacks.ModelCheckpoint(
    config.save_dir+f"/{config.MODEL_NAME}_loss_{config.FOLD_TO_RUN}.h5", monitor=
    save_weights_only=True, mode='min', save_freq='epoch')
# BUILD MODEL
K.clear_session()
model,embed_model = get_model()
snap = Snapshot(fold=config.FOLD_TO_RUN,snapshot_epochs=[5,8])
print("#########################################")
model.summary()

if config.RESUME:
    model.load_weights(config.resume_model_wts)
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 inp1 (InputLayer)              [(None, 768, 768, 3  0           []
                                )]

 efficientnet-b6 (Functional)   (None, None, None,   40960136    ['inp1[0][0]']
                                2304)

 global_average_pooling2d (Glob  (None, 2304)        0           ['efficientnet-b6
[0][0]']
 alAveragePooling2D)

 dropout (Dropout)              (None, 2304)         0           ['global_average_
pooling2d[0][0]'
                                                                 ]

 dense (Dense)                  (None, 512)          1180160     ['dropout[0][0]']

 inp2 (InputLayer)              [(None,)]            0           []

 head/arcface (ArcMarginProduct  (None, 15587)       7980544     ['dense[0][0]',
 )                                                                'inp2[0][0]']

 softmax (Softmax)              (None, 15587)        0           ['head/arcface[0]
[0]']

==================================================================================================
Total params: 50,120,840
Trainable params: 49,896,408
Non-trainable params: 224,432
__________________________________________________________________________________________________
None
##########################################
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 inp1 (InputLayer)              [(None, 768, 768, 3  0           []
                                )]

 efficientnet-b6 (Functional)   (None, None, None,   40960136    ['inp1[0][0]']
                                2304)

 global_average_pooling2d (Glob  (None, 2304)        0           ['efficientnet-b6
[0][0]']
 alAveragePooling2D)

 dropout (Dropout)              (None, 2304)         0           ['global_average_
pooling2d[0][0]'
                                                                 ]

 dense (Dense)                  (None, 512)          1180160     ['dropout[0][0]']

 inp2 (InputLayer)              [(None,)]            0           []

 head/arcface (ArcMarginProduct  (None, 15587)       7980544     ['dense[0][0]',
```

```
 )                                                                 'inp2[0][0]']

 softmax (Softmax)              (None, 15587)        0           ['head/arcface[0]
[0]']

==================================================================================
================
Total params: 50,120,840
Trainable params: 49,896,408
Non-trainable params: 224,432
__________________________________________________________________________________
________________
```

In [66]:
```
print('#### Image Size %i with EfficientNet B%i and batch_size %i'%
      (config.IMAGE_SIZE,config.EFF_NET,config.BATCH_SIZE))
"""#In this notebook, we do not train models
history = model.fit(train_dataset,
                validation_data = val_dataset,
                steps_per_epoch = STEPS_PER_EPOCH,
                epochs = config.EPOCHS,
                callbacks = [snap,get_lr_callback(),train_logger,sv_loss],
                verbose = VERBOSE)

"""
```

Out[66]:
```
#### Image Size 768 with EfficientNet B6 and batch_size 256
'#In this notebook, we do not train models \nhistory = model.fit(train_dataset,\n
validation_data = val_dataset,\n                steps_per_epoch = STEPS_PER_EPOC
H,\n                epochs = config.EPOCHS,\n                callbacks = [snap,get
_lr_callback(),train_logger,sv_loss], \n                verbose = VERBOSE)\n
\n'
```

In [67]:
```
#model.load_weights(config.save_dir+f"/{config.MODEL_NAME}_loss.h5")
embed_models=[]
for i in range(4):
    model,embed_model = get_model()
    embed_models.append((model.load_weights(f"../input/happywhale-arcface-eff6/effi
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 inp1 (InputLayer)           [(None, 768, 768, 3)]     0         
                                                                 
 efficientnet-b6 (Functional (None, None, None, 2304   40960136  
 )                           )                                   
                                                                 
 global_average_pooling2d_1 (GlobalAveragePooling2D  (None, 2304)              0         
 )                                                               
                                                                 
 dropout_1 (Dropout)         (None, 2304)              0         
                                                                 
 dense_1 (Dense)             (None, 512)               1180160   
                                                                 
 inp2 (InputLayer)           [(None,)]                 0         
                                                                 
 head/arcface (ArcMarginProduct)  (None, 15587)             7980544   
                                                                 
 softmax_1 (Softmax)         (None, 15587)             0         
                                                                 
=================================================================
Total params: 50,120,840
Trainable params: 49,896,408
Non-trainable params: 224,432
_________________________________________________________________
None
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 inp1 (InputLayer)           [(None, 768, 768, 3)]     0         
                                                                 
 efficientnet-b6 (Functional (None, None, None, 2304   40960136  
 )                           )                                   
                                                                 
 global_average_pooling2d_2 (GlobalAveragePooling2D  (None, 2304)              0         
 )                                                               
                                                                 
 dropout_2 (Dropout)         (None, 2304)              0         
                                                                 
 dense_2 (Dense)             (None, 512)               1180160   
                                                                 
 inp2 (InputLayer)           [(None,)]                 0         
```

```
 head/arcface (ArcMarginProduct  (None, 15587)       7980544     ['dense_2[0][0]',
 )                                                                'inp2[0][0]']

 softmax_2 (Softmax)             (None, 15587)       0           ['head/arcface[0]
[0]']

==================================================================================
================
Total params: 50,120,840
Trainable params: 49,896,408
Non-trainable params: 224,432
__________________________________________________________________________________
________________
None
Model: "model_6"
__________________________________________________________________________________
________________
 Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
================
 inp1 (InputLayer)               [(None, 768, 768, 3  0           []
                                 )]

 efficientnet-b6 (Functional)    (None, None, None,   40960136    ['inp1[0][0]']
                                 2304)

 global_average_pooling2d_3 (Gl  (None, 2304)        0           ['efficientnet-b6
[0][0]']
 obalAveragePooling2D)

 dropout_3 (Dropout)             (None, 2304)        0           ['global_average_
pooling2d_3[0][0
                                                                 ]']

 dense_3 (Dense)                 (None, 512)         1180160     ['dropout_3[0]
[0]']

 inp2 (InputLayer)               [(None,)]           0           []

 head/arcface (ArcMarginProduct  (None, 15587)       7980544     ['dense_3[0][0]',
 )                                                                'inp2[0][0]']

 softmax_3 (Softmax)             (None, 15587)       0           ['head/arcface[0]
[0]']

==================================================================================
================
Total params: 50,120,840
Trainable params: 49,896,408
Non-trainable params: 224,432
__________________________________________________________________________________
________________
None
Model: "model_8"
__________________________________________________________________________________
________________
 Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
================
 inp1 (InputLayer)               [(None, 768, 768, 3  0           []
                                 )]

 efficientnet-b6 (Functional)    (None, None, None,   40960136    ['inp1[0][0]']
                                 2304)
```

```
 global_average_pooling2d_4 (Gl  (None, 2304)        0           ['efficientnet-b6
[0][0]']
 obalAveragePooling2D)

 dropout_4 (Dropout)            (None, 2304)         0           ['global_average_
pooling2d_4[0][0
                                                                ]']

 dense_4 (Dense)                (None, 512)          1180160     ['dropout_4[0]
[0]']

 inp2 (InputLayer)              [(None,)]            0           []

 head/arcface (ArcMarginProduct  (None, 15587)       7980544     ['dense_4[0][0]',
 )                                                               'inp2[0][0]']

 softmax_4 (Softmax)            (None, 15587)        0           ['head/arcface[0]
[0]']

==================================================================================
================
Total params: 50,120,840
Trainable params: 49,896,408
Non-trainable params: 224,432
__________________________________________________________________________________
________________
None
```

In [68]:
```
len(embed_models)
```

Out[68]:
```
4
```

In [69]:
```
print(embed_models)
```

```
[(None, <keras.engine.functional.Functional object at 0x79182cfcfe80>), (None, <ke
ras.engine.functional.Functional object at 0x7918304122f0>), (None, <keras.engine.
functional.Functional object at 0x79182b339060>), (None, <keras.engine.functional.
Functional object at 0x79182eb024a0>)]
```

Evaluation

In [70]:
```
def get_ids(filename):
    ds = get_test_dataset([filename],get_names=True).map(lambda image, image_name:
    NUM_IMAGES = count_data_items([filename])
    ids = next(iter(ds.batch(NUM_IMAGES))).numpy().astype('U')
    return ids

def get_targets(filename):
    ds = get_eval_dataset([filename],get_targets=True).map(lambda image, target: ta
    NUM_IMAGES = count_data_items([filename])
    ids = next(iter(ds.batch(NUM_IMAGES))).numpy()
    return ids

def get_embeddings(filename):
    ds = get_test_dataset([filename],get_names=False)
    embeddings = np.mean(np.stack([embed_models[x][1].predict(ds,verbose=0) for x
    #print (embeddings.shape)
    return embeddings

def get_predictions(test_df,threshold=0.2):
    predictions = {}
```

```
    for i,row in tqdm(test_df.iterrows()):
        if row.image in predictions:
            if len(predictions[row.image])==5:
                continue
            predictions[row.image].append(row.target)
        elif row.confidence>threshold:
            predictions[row.image] = [row.target,'new_individual']
        else:
            predictions[row.image] = ['new_individual',row.target]

    for x in tqdm(predictions):
        if len(predictions[x])<5:
            remaining = [y for y in sample_list if y not in predictions]
            predictions[x] = predictions[x]+remaining
            predictions[x] = predictions[x][:5]

    return predictions

def map_per_image(label, predictions):
    """Computes the precision score of one image.

    Parameters
    ----------
    label : string
            The true label of the image
    predictions : list
            A list of predicted elements (order does matter, 5 predictions allowed

    Returns
    -------
    score : double
    """
    try:
        return 1 / (predictions[:5].index(label) + 1)
    except ValueError:
        return 0.0

f = open ('../input/happywhale-splits/individual_ids.json', "r")
target_encodings = json.loads(f.read())
target_encodings = {target_encodings[x]:x for x in target_encodings}
sample_list = ['938b7e931166', '5bf17305f073', '7593d2aee842', '7362d7a01d00','956
```

In [71]:
```
train_targets = []
train_embeddings = []
for filename in tqdm(TRAINING_FILENAMES):
    embeddings = get_embeddings(filename)
    targets = get_targets(filename)
    train_embeddings.append(embeddings)
    train_targets.append(targets)
train_embeddings = np.concatenate(train_embeddings)
train_targets = np.concatenate(train_targets)
```

```
  0%|          | 0/8 [00:00<?, ?it/s]
```

In [72]:
```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=config.KNN,metric='cosine')
neigh.fit(train_embeddings)
```

Out[72]:

NearestNeighbors

```
NearestNeighbors(metric='cosine', n_neighbors=100)
```

In [73]:
```
test_ids = []
test_nn_distances = []
test_nn_idxs = []
val_targets = []
val_embeddings = []
for filename in tqdm(VALIDATION_FILENAMES):
    embeddings = get_embeddings(filename)
    targets = get_targets(filename)
    ids = get_ids(filename)
    distances,idxs = neigh.kneighbors(embeddings, config.KNN, return_distance=True
    test_ids.append(ids)
    test_nn_idxs.append(idxs)
    test_nn_distances.append(distances)
    val_embeddings.append(embeddings)
    val_targets.append(targets)
test_nn_distances = np.concatenate(test_nn_distances)
test_nn_idxs = np.concatenate(test_nn_idxs)
test_ids = np.concatenate(test_ids)
val_embeddings = np.concatenate(val_embeddings)
val_targets = np.concatenate(val_targets)
```

```
  0%|          | 0/2 [00:00<?, ?it/s]
```

In [74]:
```
allowed_targets = set([target_encodings[x] for x in np.unique(train_targets)])
val_targets_df = pd.DataFrame(np.stack([test_ids,val_targets],axis=1),columns=['ima
val_targets_df['target'] = val_targets_df['target'].astype(int).map(target_encodin
val_targets_df.loc[~val_targets_df.target.isin(allowed_targets),'target'] = 'new_i
val_targets_df.target.value_counts()
```

Out[74]:
```
new_individual    2177
37c7aba965a5        79
114207cab555        40
e69d5f9f8d1e        33
ffbb4e585ff2        32
                  ...
80f4e0d635a7         1
c3dad9e679f3         1
179af04b755d         1
b7065da154c5         1
5c0572fe9fc1         1
Name: target, Length: 3255, dtype: int64
```

In [75]:
```
test_df = []
for i in tqdm(range(len(test_ids))):
    id_ = test_ids[i]
    targets = train_targets[test_nn_idxs[i]]
    distances = test_nn_distances[i]
    subset_preds = pd.DataFrame(np.stack([targets,distances],axis=1),columns=['targ
    subset_preds['image'] = id_
    test_df.append(subset_preds)
test_df = pd.concat(test_df).reset_index(drop=True)
test_df['confidence'] = 1-test_df['distances']
test_df = test_df.groupby(['image','target']).confidence.max().reset_index()
test_df = test_df.sort_values('confidence',ascending=False).reset_index(drop=True)
test_df['target'] = test_df['target'].map(target_encodings)
test_df.to_csv('val_neighbors.csv')
test_df.image.value_counts().value_counts()
```

```
  0%|          | 0/10207 [00:00<?, ?it/s]
```

Out[75]:
```
1      239
22     178
23     169
21     164
26     161
      ...
6       55
97      54
98      49
99      42
100     32
Name: image, Length: 100, dtype: int64
```

In [76]:
```
## Compute CV
best_th = 0
best_cv = 0
for th in [0.1*x for x in range(11)]:
    all_preds = get_predictions(test_df,threshold=th)
    cv = 0
    for i,row in val_targets_df.iterrows():
        target = row.target
        preds = all_preds[row.image]
        val_targets_df.loc[i,th] = map_per_image(target,preds)
    cv = val_targets_df[th].mean()
    print(f"CV at threshold {th}: {cv}")
    if cv>best_cv:
        best_th = th
        best_cv = cv
```

```
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.0: 0.7544756866202933
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.1: 0.7544756866202933
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.2: 0.7544756866202933
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.30000000000000004: 0.7544756866202933
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.4: 0.7545246726103001
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.5: 0.7662323242219392
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.6000000000000001: 0.809829855328043
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.7000000000000001: 0.7630482348714934
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.8: 0.6670846804480586
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.9: 0.5819470298161392
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 1.0: 0.547852780771366
```

In [82]:
```
## Compute CV
best_th = 0
best_cv = 0
for th in [0.6101+0.000001*x for x in range(11)]:
    all_preds = get_predictions(test_df,threshold=th)
    cv = 0
    for i,row in val_targets_df.iterrows():
        target = row.target
        preds = all_preds[row.image]
        val_targets_df.loc[i,th] = map_per_image(target,preds)
    cv = val_targets_df[th].mean()
    print(f"CV at threshold {th}: {cv}")
    if cv>best_cv:
        best_th = th
        best_cv = cv
```

```
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.6101: 0.8100257992880704
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.610101: 0.8100257992880704
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.6101019999999999: 0.8100257992880704
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.610103: 0.8100257992880704
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.610104: 0.8100257992880704
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.610105: 0.8100257992880704
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.6101059999999999: 0.8100257992880704
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.610107: 0.8100257992880704
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.610108: 0.8099768132980635
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.610109: 0.8099768132980635
0it [00:00, ?it/s]
  0%|          | 0/10207 [00:00<?, ?it/s]
CV at threshold 0.6101099999999999: 0.8099768132980635
```

In [84]:
```
print("Best threshold",best_th)
print("Best cv",best_cv)
val_targets_df.describe()
```

```
Best threshold 0.6101
Best cv 0.8100257992880704
```

Out[84]:

	0.000000	0.100000	0.200000	0.300000	0.400000	0.500000	0.
count	10207.000000	10207.000000	10207.000000	10207.000000	10207.000000	10207.000000	10207
mean	0.754476	0.754476	0.754476	0.754476	0.754525	0.766232	0
std	0.342060	0.342060	0.342060	0.342060	0.342060	0.341704	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0
50%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1

8 rows × 58 columns

In [85]:
```
## Adjustment: Since Public lb has nearly 10% 'new_individual' (Be Careful for priv
val_targets_df['is_new_individual'] = val_targets_df.target=='new_individual'
print(val_targets_df.is_new_individual.value_counts().to_dict())
val_scores = val_targets_df.groupby('is_new_individual').mean().T
val_scores['adjusted_cv'] = val_scores[True]*0.1+val_scores[False]*0.9
best_threshold_adjusted = val_scores['adjusted_cv'].idxmax()
print("best_threshold",best_threshold_adjusted)
val_scores
```

```
{False: 8030, True: 2177}
best_threshold 0.6000000000000001
/tmp/ipykernel_34/4292826914.py:4: FutureWarning: The default value of numeric_onl
y in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will d
efault to False. Either specify numeric_only or select only columns which should b
e valid for the function.
  val_scores = val_targets_df.groupby('is_new_individual').mean().T
```

Out[85]:

is_new_individual	False	True	adjusted_cv
0.0	0.823466	0.500000	0.791120
0.1	0.823466	0.500000	0.791120
0.2	0.823466	0.500000	0.791120
0.30000000000000004	0.823466	0.500000	0.791120
0.4	0.823466	0.500230	0.791143
0.5	0.821847	0.561093	0.795772
0.6000000000000001	0.792831	0.872531	0.800801
0.7000000000000001	0.700490	0.993799	0.729821
0.8	0.577015	0.999311	0.619245
0.9	0.468609	1.000000	0.521748
1.0	0.425272	1.000000	0.482745
0.6	0.792831	0.872531	0.800801
0.61	0.785484	0.900322	0.796967
0.62	0.778883	0.922370	0.793232
0.63	0.770913	0.942352	0.788057
0.64	0.763192	0.957051	0.782578
0.65	0.754537	0.969453	0.776029
0.6599999999999999	0.745509	0.977722	0.768730
0.6699999999999999	0.733429	0.985301	0.758616
0.6799999999999999	0.723093	0.988516	0.749635
0.69	0.712383	0.991502	0.740295
0.7	0.700490	0.993799	0.729821
0.611	0.784674	0.902848	0.796492
0.612	0.784114	0.904226	0.796125
0.613	0.783367	0.906523	0.795682
0.614	0.782557	0.909968	0.795298
0.615	0.782183	0.911805	0.795146
0.616	0.781312	0.913413	0.794522
0.617	0.780814	0.915480	0.794280
0.618	0.780378	0.917777	0.794118
0.619	0.779755	0.920073	0.793787
0.6101	0.785484	0.900551	0.796990
0.6102	0.785421	0.900551	0.796934
0.6103	0.785297	0.900781	0.796845
0.6103999999999999	0.785172	0.901240	0.796779
0.6104999999999999	0.785172	0.901240	0.796779

is_new_individual	False	True	adjusted_cv
0.6106	0.784985	0.901240	0.796611
0.6107	0.784861	0.901929	0.796568
0.6108	0.784799	0.902159	0.796535
0.6109	0.784736	0.902848	0.796548
0.6101099999999999	0.785421	0.900551	0.796934
0.61012	0.785421	0.900551	0.796934
0.61013	0.785421	0.900551	0.796934
0.61014	0.785421	0.900551	0.796934
0.61015	0.785421	0.900551	0.796934
0.6101599999999999	0.785421	0.900551	0.796934
0.61017	0.785421	0.900551	0.796934
0.61018	0.785421	0.900551	0.796934
0.61019	0.785421	0.900551	0.796934
0.610101	0.785484	0.900551	0.796990
0.6101019999999999	0.785484	0.900551	0.796990
0.610103	0.785484	0.900551	0.796990
0.610104	0.785484	0.900551	0.796990
0.610105	0.785484	0.900551	0.796990
0.6101059999999999	0.785484	0.900551	0.796990
0.610107	0.785484	0.900551	0.796990
0.610108	0.785421	0.900551	0.796934
0.610109	0.785421	0.900551	0.796934

Inference

In [86]:
```
train_embeddings = np.concatenate([train_embeddings,val_embeddings])
train_targets = np.concatenate([train_targets,val_targets])
print(train_embeddings.shape,train_targets.shape)
```

```
(51033, 512) (51033,)
```

In [87]:
```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=config.KNN,metric='cosine')
neigh.fit(train_embeddings)
```

Out[87]:
```
NearestNeighbors(metric='cosine', n_neighbors=100)
```

In [88]:
```
test_ids = []
test_nn_distances = []
test_nn_idxs = []
for filename in tqdm(test_files):
```

```
    embeddings = get_embeddings(filename)
    ids = get_ids(filename)
    distances,idxs = neigh.kneighbors(embeddings, config.KNN, return_distance=True
    test_ids.append(ids)
    test_nn_idxs.append(idxs)
    test_nn_distances.append(distances)
test_nn_distances = np.concatenate(test_nn_distances)
test_nn_idxs = np.concatenate(test_nn_idxs)
test_ids = np.concatenate(test_ids)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
```

In [89]:
```
sample_submission = pd.read_csv('../input/happy-whale-and-dolphin/sample_submissio
print(len(test_ids),len(sample_submission))
test_df = []
for i in tqdm(range(len(test_ids))):
    id_ = test_ids[i]
    targets = train_targets[test_nn_idxs[i]]
    distances = test_nn_distances[i]
    subset_preds = pd.DataFrame(np.stack([targets,distances],axis=1),columns=['tar
    subset_preds['image'] = id_
    test_df.append(subset_preds)
test_df = pd.concat(test_df).reset_index(drop=True)
test_df['confidence'] = 1-test_df['distances']
test_df = test_df.groupby(['image','target']).confidence.max().reset_index()
test_df = test_df.sort_values('confidence',ascending=False).reset_index(drop=True)
test_df['target'] = test_df['target'].map(target_encodings)
test_df.to_csv('test_neighbors.csv')
test_df.image.value_counts().value_counts()
```

```
27956 27956
  0%|          | 0/27956 [00:00<?, ?it/s]
```

Out[89]:
```
20     378
64     373
55     369
21     366
51     362
      ...
100    119
5      110
2       94
4       79
3       77
Name: image, Length: 100, dtype: int64
```

In [90]:
```
sample_list = ['938b7e931166', '5bf17305f073', '7593d2aee842', '7362d7a01d00','956
```

In [91]:
```
predictions = {}
for i,row in tqdm(test_df.iterrows()):
    if row.image in predictions:
        if len(predictions[row.image])==5:
            continue
        predictions[row.image].append(row.target)
    elif row.confidence>best_threshold_adjusted:
        predictions[row.image] = [row.target,'new_individual']
    else:
        predictions[row.image] = ['new_individual',row.target]

for x in tqdm(predictions):
    if len(predictions[x])<5:
        remaining = [y for y in sample_list if y not in predictions]
        predictions[x] = predictions[x]+remaining
        predictions[x] = predictions[x][:5]
    predictions[x] = ' '.join(predictions[x])
```

```
predictions = pd.Series(predictions).reset_index()
predictions.columns = ['image','predictions']
predictions.to_csv('cs444_submission_1.csv',index=False)
predictions.head()
```

```
0it [00:00, ?it/s]
  0%|          | 0/27956 [00:00<?, ?it/s]
```

Out[91]:

	image	predictions
0	3c52966f74d2ad.jpg	978520860ceb new_individual 80274941b298 d4dfc...
1	d846a86edded63.jpg	03a3bbaeed84 new_individual 51f0440962bb 65822...
2	59bc2d264c9731.jpg	47a2f9918aa2 new_individual 73c68d52e748 82fa8...
3	50df0a954eb94c.jpg	713eb1a00c3d new_individual 9633e4159c19 36093...
4	0cfc88bad51a18.jpg	e4a55c745bd9 new_individual 2ca0b9b39092 5c8d7...