Параметыр модели

In [ ]:
```
BATCH_SIZE = 64
EDGE_CROP = 16
GAUSSIAN_NOISE = 0.1
UPSAMPLE_MODE = "SIMPLE"

#Понижение дискритизации внутри сети
NET_SCALING = (1, 1)

#Понижение размерности в предобработке
IMG_SCALING = (3, 3)

#Число изображений для валидации
VALID_IMG_COUNT = 900

#Максимальное число шагов за эпоху при обучении
MAX_TRAIN_STEPS = 9
MAX_TRAIN_EPOCHS = 50
AUGMENT_BRIGHTNESS = False

SEED = 42
```

In [ ]:
```
from skimage.util import montage
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from skimage.io import imread
import matplotlib.pyplot as plt
from matplotlib.cm import get_cmap
from skimage.segmentation import mark_boundaries
from sklearn.model_selection import train_test_split
import keras.backend as K

from keras.preprocessing.image import ImageDataGenerator
from keras import models, layers
import keras.backend as K
from keras.optimizers import Adam
from keras.losses import binary_crossentropy
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, Earl
from tensorflow.keras.optimizers import RMSprop, Adam




from skimage.morphology import binary_opening, disk, label
import gc; gc.enable()

from PIL import Image
```

In [ ]:
```
montage_rgb = lambda x: np.stack([montage(x[:, :, :, i]) for i in range(
ship_dir = "/content/drive/MyDrive/Диплом/Ship_detection/Input/"
train_image_dir = os.path.join(ship_dir, 'train')
```

Определим вспомогательные процедуры для декодирования, кодирования и вывода изображения и маски корабля

In [ ]:
```
def rle_encode(img, min_max_treshold = 1e-3, max_mean_treshold = None):
    '''
    img: numpy array, 1 - mask, 0 - background
    Возвращает бегущую строку как форматированную
    '''
    if (np.max(img) < min_max_treshold):
        return ''
    if (max_mean_treshold and np.mean(img) > max_mean_treshold):
        return ''
    pixels = img.T.flatten()
    pixels = np.concatenate([[0], pixels, [0]])
    runs = np.where(pixels[1:] != pixels[:-1])[0] + 1
    runs[1::2] -= runs[::2]
    return ' '.join(str(x) for x in runs)
```

In [ ]:
```
def multi_rle_encode(img, **kwargs):
    '''
    Кодируем объединенные регионы как разделители масок
    '''
    labels = label(img)
    if img.ndim > 2:
      return [rle_encode(np.sum(labels == k, axis = 2), **kwargs) for k
    else:
        return [rle_encode(labels == k, **kwargs) for k in np.unique(lab
```

In [ ]:
```
def rle_decode(mask_rle, shape = (768, 768)):
    '''
    mask_rle: бегущая - длина как форматированная строка (start length)
    shape: (height,width) массив для возвратного значения
    Возвращаем  numpy array, 1 - mask, 0 - background
    '''
    s = mask_rle.split()
    starts, lengths = [np.asarray(x, dtype = int) for x in (s[0:][::2],
    starts -= 1
    ends = starts + lengths
    img = np.zeros(shape[0] * shape[1], dtype = np.uint8)
    for lo, hi in zip(starts, ends):
        img[lo:hi] = 1
    return img.reshape(shape).T
```

In [ ]:
```
def masks_as_image(in_mask_list):
    #Берем индивидуальную маску корабля и создаем отдельный массив масок
    all_masks = np.zeros((768, 768), dtype = np.uint8)
    for mask in in_mask_list:
        if isinstance(mask, str):
            all_masks |= rle_decode(mask)
    return all_masks
```

In [ ]:
```
def masks_as_color(in_mask_list):
    #Берем индивидуальную маску корабля и создаем цветовую маску для каж
    all_masks = np.zeros((768, 768), dtype = np.float)
    scale = lambda x: (len(in_mask_list) + x + 1) / (len(in_mask_list) *
    for i, mask in enumerate(in_mask_list):
        if isinstance(mask, str):
            all_masks[:, :] += scale(i) * rle_decode(mask)
    return all_masks
```

Продемонстрируем работу

In [ ]:
```
masks = pd.read_csv("/content/drive/MyDrive/Диплом/Ship_detection/Input/
```

```
masks = masks.drop(['Unnamed: 0', 'exist_ship'], axis=1)
masks.head()
```

In [ ]:
```
not_empty = pd.notna(masks.EncodedPixels)
print(not_empty.sum(), "masks in", masks[not_empty].ImageId.nunique(), '
print((~not_empty).sum(), "empty images in", masks.ImageId.nunique(), "t
```

In [ ]:
```
im = Image.open("/content/drive/MyDrive/Диплом/Ship_detection/Input/trai
im
```

In [ ]:
```
fig, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5, figsize = (30, 5))
rle_0 = masks.query('ImageId == "8ce7d933f.jpg"')["EncodedPixels"]
img_0 = masks_as_image(rle_0)
ax0.imshow(im)
ax0.set_title("Оригинальное изображение")

ax1.imshow(img_0)
ax1.set_title("Маска как изображение")

rle_1 = multi_rle_encode(img_0)
img_1 = masks_as_color(rle_0)
ax2.imshow(img_1)
ax2.set_title("Перекодированное")

img_c = masks_as_color(rle_0)
ax3.imshow(img_c)
ax3.set_title("Масква в цвете")

img_c = masks_as_color(rle_1)
ax4.imshow(img_c)
ax4.set_title("Перекодированное в цвета")
print("Проверка Декодирования -> Кодирование", 'RLE_0:', len(rle_0), '->
      'RLE_1:', len(rle_1))
print(np.sum(img_0 - img_1), 'error')
```

Разделим данные на тренировочные и проверочные

In [ ]:
```
#Поле, указывающее, есть ли корабль на картинке:  1 - есть, 0 - нет
masks['ships'] = masks['EncodedPixels'].map(lambda c_row: 1 if isinstanc
unique_img_ids = masks.groupby("ImageId").agg({'ships': 'sum'}).reset_in
unique_img_ids['has_ship'] = unique_img_ids['ships'].map(lambda x: 1.0 i
unique_img_ids['has_ship_vec'] = unique_img_ids['has_ship'].map(lambda x
unique_img_ids['file_size_kb'] = unique_img_ids['ImageId'].map(lambda c_
unique_img_ids['file_size_kb'].hist()
masks.drop(['ships'], axis = 1, inplace = True)
unique_img_ids.sample(10)
```

Построим гистограмму от числа кораблей (копий изображения) для одного файла

In [ ]:
```
unique_img_ids['ships'].hist(bins= unique_img_ids['ships'].max() + 1)
```

========== МЕСТО ДЛЯ ОТСЕИВАНИЯ ДУБЛИКАТОВ ==========

Сбалансируем выборку

In [ ]:
```
train_ids, valid_ids = train_test_split(unique_img_ids, test_size = 0.25
train_df = pd.merge(masks, train_ids)
valid_df = pd.merge(masks, valid_ids)

print(train_df.shape[0], 'training masks')
print(valid_df.shape[0], 'validation masks')
```

Декодируем данные в изображения

In [ ]:
```
def make_image_gen(in_df, batch_size = BATCH_SIZE):
    all_batches = list(in_df.groupby('ImageId'))
    out_rgb = []
    out_mask = []
    while True:
        np.random.shuffle(all_batches)
        for c_img_id, c_masks in all_batches:
            rgb_path = os.path.join(train_image_dir, c_img_id)
            c_img = imread(rgb_path)
            c_mask = np.expand_dims(masks_as_image(c_masks['EncodedPixel
            if IMG_SCALING is not None:
                c_img = c_img[::IMG_SCALING[0], ::IMG_SCALING[1]]
                c_mask = c_mask[::IMG_SCALING[0], ::IMG_SCALING[1]]
            out_rgb += [c_img]
            out_mask += [c_mask]
            if len(out_rgb)>=batch_size:
                yield np.stack(out_rgb, 0)/255.0, np.stack(out_mask, 0)
                out_rgb, out_mask=[], []
```

In [ ]:
```
"""
x (2048, 256, 256, 3) 0.0 1.0
y (2048, 256, 256, 1) 0 1

train_gen = make_image_gen(train_df)
train_x, train_y = next(train_gen)
print('x', train_x.shape, train_x.min(), train_x.max())
print('y', train_y.shape, train_y.min(), train_y.max())
"""
```

In [ ]:
```
'''
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (30, 10))
batch_rgb = montage_rgb(train_x)
batch_seg = montage(train_y[:, :, :, 0])
ax1.imshow(batch_rgb)
ax1.set_title('Images')
ax2.imshow(batch_seg)
ax2.set_title('Segmentations')
ax3.imshow(mark_boundaries(batch_rgb,
                           batch_seg.astype(int)))
ax3.set_title('Outlined Ships')
fig.savefig('overview.png')
'''
```

Сделаем набор для проверки

In [ ]:
```
%%time
valid_x, valid_y = next(make_image_gen(valid_df, VALID_IMG_COUNT))
print(valid_x.shape, valid_y.shape)
```

In [ ]:
```
valid_x[0]
```

In [ ]:
```
s = 10
j = 0
for r in valid_y[-17]:
    k = 10
    i = 0
    for c in r:
        if(i > k):
            print("...")
            break;
        print(c, sep=' ', end='', flush=True)
        i += 1
    print
    j += 1
    if(j > k):
            print("...")
            break;
```

Дополним данные

In [ ]:
```
dg_args = dict(featurewise_center = False,
                   samplewise_center = False,
                   rotation_range = 45,
                   width_shift_range = 0.1,
                   height_shift_range = 0.1,
                   shear_range = 0.01,
                   zoom_range = [0.9, 1.25],
                   horizontal_flip = True,
                   vertical_flip = True,
                   fill_mode = 'reflect',
                    data_format = 'channels_last')
# brightness can be problematic since it seems to change the labels diff
if AUGMENT_BRIGHTNESS:
    dg_args[' brightness_range'] = [0.5, 1.5]
image_gen = ImageDataGenerator(**dg_args)

if AUGMENT_BRIGHTNESS:
    dg_args.pop('brightness_range')
label_gen = ImageDataGenerator(**dg_args)

def create_aug_gen(in_gen, seed = None):
    np.random.seed(seed if seed is not None else np.random.choice(range(
    for in_x, in_y in in_gen:
        seed = np.random.choice(range(9999))
        # keep the seeds syncronized otherwise the augmentation to the i
        g_x = image_gen.flow(255*in_x,
                             batch_size = in_x.shape[0],
                             seed = seed,
                             shuffle=True)
        g_y = label_gen.flow(in_y,
                             batch_size = in_x.shape[0],
                             seed = seed,
                             shuffle=True)

        yield next(g_x)/255.0, next(g_y)
```

In [ ]:
```
"""
x (64, 256, 256, 3) float32 0.0 1.0
y (64, 256, 256, 1) float32 0.0 1.0
cur_gen = create_aug_gen(train_gen)
```

```
t_x, t_y = next(cur_gen)
print('x', t_x.shape, t_x.dtype, t_x.min(), t_x.max())
print('y', t_y.shape, t_y.dtype, t_y.min(), t_y.max())
# only keep first 9 samples to examine in detail
t_x = t_x[:9]
t_y = t_y[:9]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (20, 10))
ax1.imshow(montage_rgb(t_x), cmap='gray')
ax1.set_title('images')
ax2.imshow(montage(t_y[:, :, :, 0]), cmap='gray_r')
ax2.set_title('ships')
"""
```

In [ ]:
```
gc.collect()
```

Соберем модель

In [ ]:
```
# Build U-Net model
def upsample_conv(filters, kernel_size, strides, padding):
    return layers.Conv2DTranspose(filters, kernel_size, strides=strides,
def upsample_simple(filters, kernel_size, strides, padding):
    return layers.UpSampling2D(strides)

if UPSAMPLE_MODE=='DECONV':
    upsample=upsample_conv
else:
    upsample=upsample_simple

input_img = layers.Input([256, 256, 3], name = 'RGB_Input')
pp_in_layer = input_img

if NET_SCALING is not None:
    pp_in_layer = layers.AvgPool2D(NET_SCALING)(pp_in_layer)

pp_in_layer = layers.GaussianNoise(GAUSSIAN_NOISE)(pp_in_layer)
pp_in_layer = layers.BatchNormalization()(pp_in_layer)

c1 = layers.Conv2D(8, (3, 3), activation='relu', padding='same') (pp_in_
c1 = layers.Conv2D(8, (3, 3), activation='relu', padding='same') (c1)
p1 = layers.MaxPooling2D((2, 2)) (c1)

c2 = layers.Conv2D(16, (3, 3), activation='relu', padding='same') (p1)
c2 = layers.Conv2D(16, (3, 3), activation='relu', padding='same') (c2)
p2 = layers.MaxPooling2D((2, 2)) (c2)

c3 = layers.Conv2D(32, (3, 3), activation='relu', padding='same') (p2)
c3 = layers.Conv2D(32, (3, 3), activation='relu', padding='same') (c3)
p3 = layers.MaxPooling2D((2, 2)) (c3)

c4 = layers.Conv2D(64, (3, 3), activation='relu', padding='same') (p3)
c4 = layers.Conv2D(64, (3, 3), activation='relu', padding='same') (c4)
p4 = layers.MaxPooling2D(pool_size=(2, 2)) (c4)


c5 = layers.Conv2D(128, (3, 3), activation='relu', padding='same') (p4)
c5 = layers.Conv2D(128, (3, 3), activation='relu', padding='same') (c5)

u6 = upsample(64, (2, 2), strides=(2, 2), padding='same') (c5)
u6 = layers.concatenate([u6, c4])
c6 = layers.Conv2D(64, (3, 3), activation='relu', padding='same') (u6)
c6 = layers.Conv2D(64, (3, 3), activation='relu', padding='same') (c6)
```

```
u7 = upsample(32, (2, 2), strides=(2, 2), padding='same') (c6)
u7 = layers.concatenate([u7, c3])
c7 = layers.Conv2D(32, (3, 3), activation='relu', padding='same') (u7)
c7 = layers.Conv2D(32, (3, 3), activation='relu', padding='same') (c7)

u8 = upsample(16, (2, 2), strides=(2, 2), padding='same') (c7)
u8 = layers.concatenate([u8, c2])
c8 = layers.Conv2D(16, (3, 3), activation='relu', padding='same') (u8)
c8 = layers.Conv2D(16, (3, 3), activation='relu', padding='same') (c8)

u9 = upsample(8, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = layers.concatenate([u9, c1], axis=3)
c9 = layers.Conv2D(8, (3, 3), activation='relu', padding='same') (u9)
c9 = layers.Conv2D(8, (3, 3), activation='relu', padding='same') (c9)

d = layers.Conv2D(1, (1, 1), activation='sigmoid') (c9)
# d = layers.Cropping2D((EDGE_CROP, EDGE_CROP))(d)
# d = layers.ZeroPadding2D((EDGE_CROP, EDGE_CROP))(d)
if NET_SCALING is not None:
    d = layers.UpSampling2D(NET_SCALING)(d)

seg_model = models.Model(inputs=[input_img], outputs=[d])
seg_model.summary()
```

In [ ]:
```
image_file = 'model_1.png'
tf.keras.utils.plot_model(seg_model, to_file = image_file, show_shapes =
```

In [ ]:
```
#https://lars76.github.io/2018/09/27/loss-functions-for-segmentation.htm
def dice_coef(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.math.sigmoid(y_pred)
    numerator = 2 * tf.reduce_sum(y_true * y_pred)
    denominator = tf.reduce_sum(y_true + y_pred)
    return numerator / denominator

def dice_loss(y_true, y_pred):
  return 1 - dice_coef(y_true, y_pred)

def true_positive_rate(y_true, y_pred):
    return K.sum(K.flatten(y_true)*K.flatten(K.round(y_pred)))/K.sum(y_t

#Cross entropy + DICE loss
def comb_loss(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    o = tf.nn.sigmoid_cross_entropy_with_logits(y_true, y_pred) + dice_l
    return tf.reduce_mean(o)


def balanced_cross_entropy(y_true, y_pred):
    weight_a = beta * tf.cast(y_true, tf.float32)
    weight_b = (1 - beta) * tf.cast(1 - y_true, tf.float32)

    o = (tf.math.log1p(tf.exp(-tf.abs(y_pred))) + tf.nn.relu(-y_pred)) *
    return tf.reduce_mean(o)

beta = 0.7
def tversky_loss(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.math.sigmoid(y_pred)
    numerator = y_true * y_pred
    denominator = y_true * y_pred + beta * (1 - y_true) * y_pred + (1 - 
```

```
    return 1 - tf.reduce_sum(numerator) / tf.reduce_sum(denominator)
```

In [ ]:
```
weight_path="/content/drive/MyDrive/Диплом/Ship_detection/weights/u_net/

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,

reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.33,
                                   patience=1, verbose=1, mode='min',
                                   min_delta=0.0001, cooldown=0, min_lr=

early = EarlyStopping(monitor="val_loss", mode="min", verbose=2,
                      patience=20) # probably needs to be more patient,

callbacks_list = [checkpoint, early, reduceLROnPlat]
```

In [ ]:
```
gc.collect()
```

In [ ]:
```
RMS = RMSprop( learning_rate=0.001,
                    rho=0.9,
                    momentum=0.0,
                    epsilon=1e-07,
                    centered=False,
                    name="RMSprop")

adam = Adam(learning_rate=0.001,
                beta_1=0.9,
                beta_2=0.999,
                epsilon=1e-07,
                amsgrad=False,
                name="Adam")
seg_model.compile(optimizer = adam, loss= tversky_loss, metrics=['binary
#weight_path1="/content/drive/MyDrive/Диплом/Ship_detection/weights/u_ne
#seg_model.load_weights(weight_path1)
```

In [ ]:
```
def fit(seq_model):
    step_count = MAX_TRAIN_STEPS
    #step_count = train_df.shape[0]//BATCH_SIZE
    aug_gen = create_aug_gen(make_image_gen(train_df))
    loss_history = [seg_model.fit(aug_gen,
                                  steps_per_epoch=step_count,
                                 # batch_size = BATCH_SIZE,
                                  epochs=MAX_TRAIN_EPOCHS,
                                  validation_data=(valid_x, valid_y),
                                  callbacks=callbacks_list
                                 )]
    return loss_history

loss_history = fit(seg_model)
```

In [ ]:
```
def show_loss(loss_history):
    epochs = np.concatenate([mh.epoch for mh in loss_history])
    fig, (ax1, ax2, ax3, ax, ax5) = plt.subplots(1, 5, figsize=(22, 10))

    _ = ax1.plot(epochs, np.concatenate([mh.history['loss'] for mh in lo
                 epochs, np.concatenate([mh.history['val_loss'] for mh i
    ax1.legend(['Training', 'Validation'])
    ax1.set_title('Loss')

    _ = ax2.plot(epochs, np.concatenate([mh.history['binary_accuracy'] f
                 epochs, np.concatenate([mh.history['val_binary_accuracy
    ax2.legend(['Training', 'Validation'])
```

```
    ax2.set_title('Binary Accuracy (%)')

    _ = ax3.plot(epochs, np.concatenate([mh.history['dice_coef'] for mh
                 epochs, np.concatenate([mh.history['val_dice_coef'] for
    ax3.legend(['Training', 'Validation'])
    ax3.set_title('DICE Coefficient (%)')

    _ = ax4.plot(epochs, np.concatenate([mh.history['true_positive_rate'
                 epochs, np.concatenate([mh.history['val_true_positive_r
    ax4.legend(['Training', 'Validation'])
    ax4.set_title('TFP')

      _ = ax5.plot(epochs, np.concatenate([mh.history['false_positives']
                 epochs, np.concatenate([mh.history['val_false_positives
    ax5.legend(['Training', 'Validation'])
    ax5.set_title('FPR')


    fig.savefig('/content/drive/MyDrive/Диплом/Ship_detection/RMS_TSKY_B

show_loss(loss_history)
```

In [ ]:
```
gc.collect()
```

In [ ]:
```
seg_model.load_weights(weight_path)
seg_model.save('/content/drive/MyDrive/Диплом/Ship_detection/weights/u_n
```

In [ ]:
```
pred_y = seg_model.predict(valid_x)
print(pred_y.shape, pred_y.min(axis=0).max(), pred_y.max(axis=0).min(),
```

In [ ]:
```
fig, ax = plt.subplots(1, 1, figsize = (6, 6))
ax.hist(pred_y.ravel(), np.linspace(0, 1, 20))
ax.set_xlim(0, 1)
ax.set_yscale('log', nonposy='clip')
```

Подготовка для полноразмерной модели

In [ ]:
```
if IMG_SCALING is not None:
    fullres_model = models.Sequential()
    fullres_model.add(layers.AvgPool2D(IMG_SCALING, input_shape = (None,
    fullres_model.add(seg_model)
    fullres_model.add(layers.UpSampling2D(IMG_SCALING))
else:
    fullres_model = seg_model
fullres_model.save('/content/drive/MyDrive/Диплом/Ship_detection/weights
```

In [ ]:
```
gc.collect()
```

Визуализируем предсказание

In [ ]:
```
def raw_prediction(img, path=train_image_dir):
    c_img = imread(os.path.join(path, c_img_name))
    c_img = np.expand_dims(c_img, 0)/255.0
    cur_seg = fullres_model.predict(c_img)[0]
    return cur_seg, c_img[0]

def smooth(cur_seg):
    return binary_opening(cur_seg>0.99, np.expand_dims(disk(2), -1))
```

```
def predict(img, path=train_image_dir):
    cur_seg, c_img = raw_prediction(img, path=path)
    return smooth(cur_seg), c_img


## Get a sample of each group of ship count
n_samples = 100
samples = valid_df.groupby('ships').apply(lambda x: x.sample(random_state
fig, m_axs = plt.subplots(samples.shape[0], 4, figsize = (15, samples.sh
[c_ax.axis('off') for c_ax in m_axs.flatten()]

for (ax1, ax2, ax3, ax4), c_img_name in zip(m_axs, samples.ImageId.value
    first_seg, first_img = raw_prediction(c_img_name, train_image_dir)
    ax1.imshow(first_img)
    ax1.set_title('Image: ' + c_img_name)
    ax2.imshow(first_seg[:, :, 0], cmap=get_cmap('jet'))
    ax2.set_title('Model Prediction')
    reencoded = masks_as_color(multi_rle_encode(smooth(first_seg)[:, :,
    ax3.imshow(reencoded)
    ax3.set_title('Prediction Masks')
    ground_truth = masks_as_color(masks.query('ImageId=="{}"'.format(c_i
    ax4.imshow(ground_truth)
    ax4.set_title('Ground Truth')

fig.savefig('/content/drive/MyDrive/Диплом/Ship_detection/RMS_50_TSKY_BE
plt.show()
```

In [ ]: