МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по индивидуальному домашнему заданию

по дисциплине «Искусственные нейронные сети»

Тема: Сегментация листов стали и их классификация по видам дефектов

Студент гр. 7382 _____ Гаврилов А.В.

Студент гр. 7382 _____ Гиззатов А.

Студент гр. 7383 _____ Бергалиев М.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Сегментировать и классифицировать листы стали по разным видам дефектов.

Основные теоретические положения.

Архитектура U-net состоит из стягивающего пути для захвата контекста и симметричного расширяющегося пути, который позволяет осуществить точную локализацию. Сеть обучается сквозным способом на небольшом количестве изображений и превосходит предыдущий наилучший метод (сверточную сеть со скользящим окном) на соревновании ISBI по сегментации нейронных структур в электронно-микроскопических стеках**.** Используя ту же сеть, которая была обучена на изображениях световой микроскопии пропускания (фазовый контраст и DIC), U-Net заняла первое место в конкурсе ISBI 2015 года по трекингу клеток в этих категориях с большим отрывом. Кроме того, эта сеть работает быстро. Сегментация изображения 512×512 занимает менее секунды.

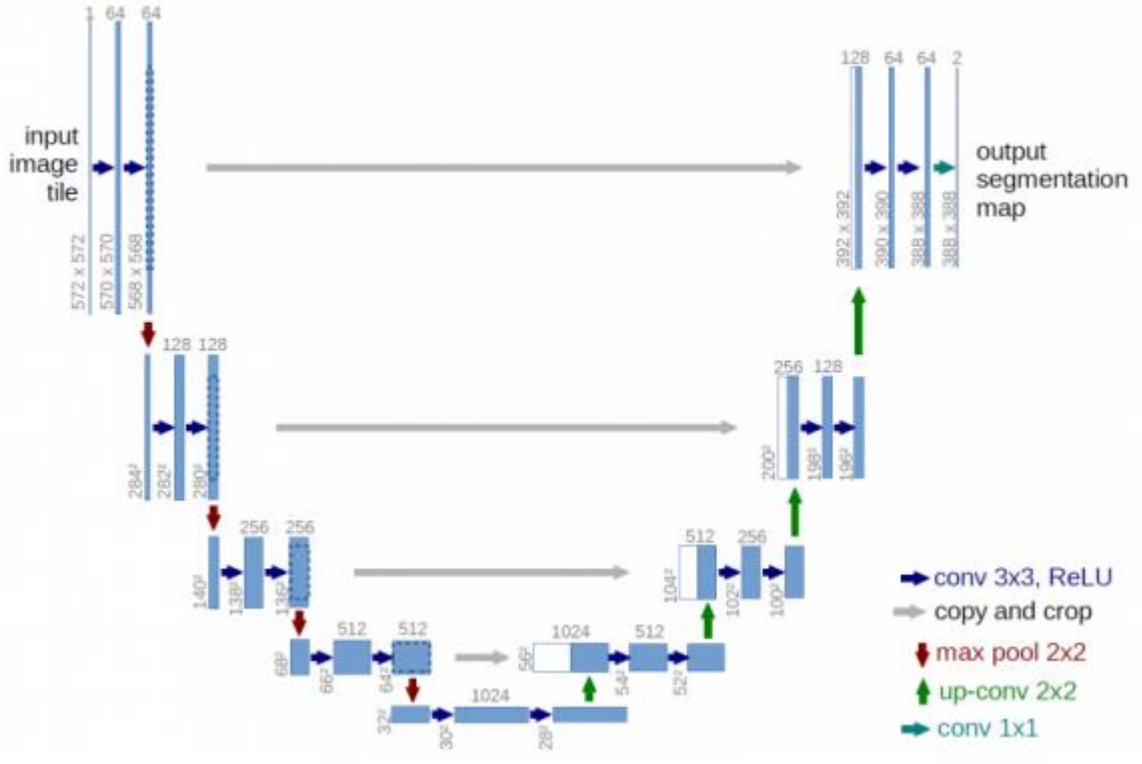

Рис.1 – Архитектура U-Net

Архитектура сети приведенная на рис.1. состоит из сужающегося пути (слева) и расширяющегося пути (справа). Сужающийся путь — типичная архитектура

сверточной нейронной сети. Он состоит из повторного применения двух сверток 3×3, за которыми следуют функция активации ReLU и операция дискретизации по максимальному значению (2×2 степени 2) для понижения разрешения.

На каждом этапе понижающей дискретизации каналы свойств удваиваются. Каждый шаг в расширяющемся пути состоит из операции повышающей дискретизации карты свойств, за которой следуют:

- свертка 2×2, которая уменьшает количество каналов свойств;
- объединение с соответствующим образом обрезанной картой свойств из стягивающегося пути;
- две 3×3 свертки, за которыми следует ReLU.

Обрезка необходима из-за потери граничных пикселей при каждой свертке. На последнем слое используется свертка 1×1 для сопоставления каждого 64-компонентного вектора свойств с желаемым количеством классов. Всего сеть содержит 23 сверточных слоя.

1. Описание датасета.

Данный датасет состоит из более чем 12 тыс. изображений листов стали с соотношением сторон 1600x256. Каждое изображение может не иметь дефектов, дефект одного класса или дефекты нескольких классов. Для каждого изображения нужно сегментировать дефекты каждого класса (ClassId = [1, 2, 3, 4]). Задачей, поставленной на данном датасете, является сегментация дефектов и определение классов этих дефектов.

2. Начальный анализ данных

Так как 1 изображение весит примерно 100 кб, то более 12 тыс. таких изображений будет весить около 5 гигабайт, поэтому было принято решение сократить соотношение сторон до 800x128 и рассмотрено количество памяти для

измененных изображений. В итоге суммарно все изображения весили примерно 1.2 гигабайта, что уже лучше предыдущего результата, но до сих пор являющееся слишком большим с масками классов (каждая маска является таким же изображением). С масками все изображения весили около 6 гигабайт, поэтому было принято решение еще сократить соотношение сторон до 400x64, что было приемлемым для загрузки в ОЗУ для обучения.

Было принято решение нормировать данные считанные с изображений, потому что нейронной сети легче обучаться на данных, которые находятся в небольшом промежутке. Способом нормирования было вычитанием мат.ожидания и делением на СКО, потому что при этой нормировке нейронная сеть выдавала более хорошие значения коэффициента дайса.(Это можно увидеть на рис.2 и рис.3)

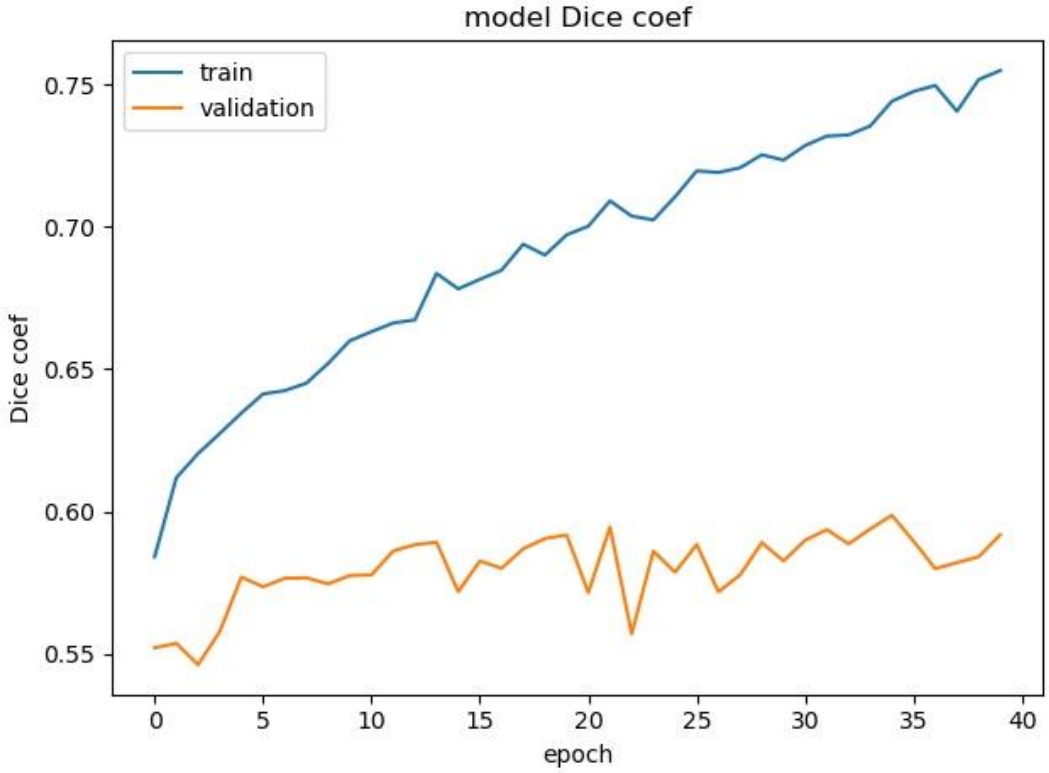

Рис.2 - Коэффициент Дайса на 40 эпохах при нормировании вычитанием мат.ожидания и делением на СКО.

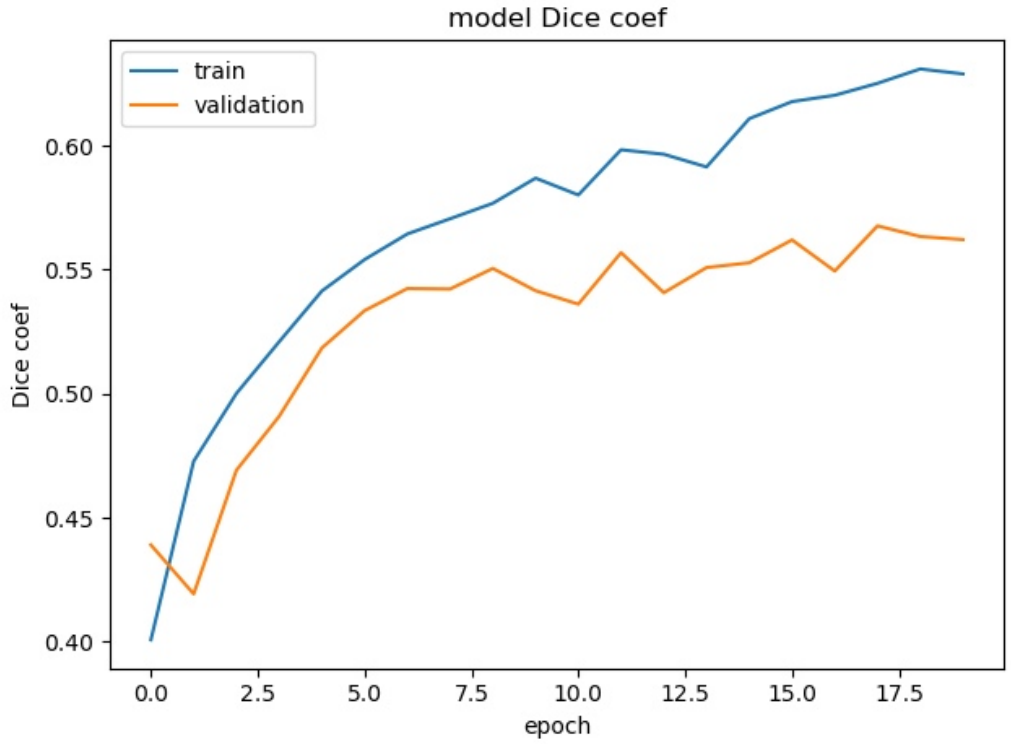

Рис.3 - Коэффициент Дайса на 20 при нормировании делением на 255.

3. Процесс разработки.

В самом начале разработки нейронной сети были рассмотрены несколько нейронных сетей для семантической сегментации:

- U-net
- The One Hundred Layers Tiramisu
- Pspnet

И принято решение использовать U-net так как она использует небольшое количество данных для достижения хороших результатов.

Так как еще не было принято решение о том, какую функцию потерь использовать, какую скорость обучения выставлять и о других параметрах обучения, мы использовали callback, который сохраняет веса модели с наилучшими потерями. Таким образом при одной из итерации обучения мы получили наши стартовые веса, на основе которых мы в дальнейшем и обучали модель для экономии времени.

При выборе параметров обучения мы сначала использовали функцию потерь - бинарную кросентропию, так как по сути задача состоит в

определении принадлежит ли маска данному виду дефекта. Затем мы начали уменьшать скорость обучения, так как по графикам было явно видно, что модель практически не обучалась, так как не получалось найти минимум. И выбрали наименьший параметр при котором сеть относительно неплохо обучалась - 10^{-5}.

Затем попробовали использовать - dice coefficient в качестве функции потерь для минимизации и получили следующие результаты:

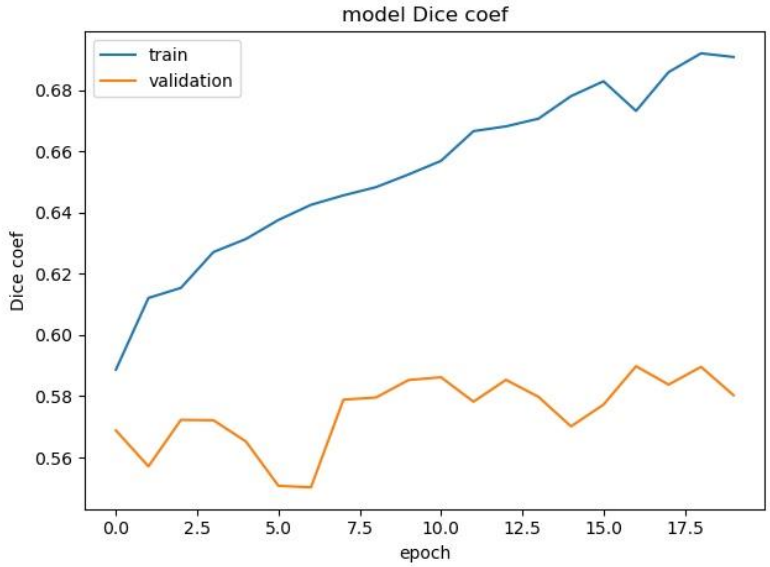

Рисунок 4 - Обучение с коэффициентом дайсона в качестве функции потерь.

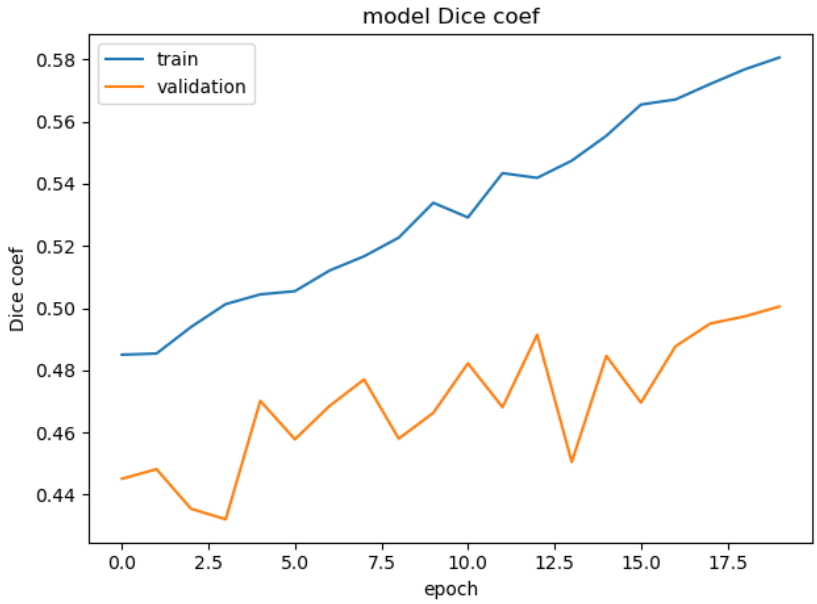

Рисунок 5 - Обучение с бинарной кросентропией в качестве функции потерь.

Выбран отрицательный коэффициент дайсона в качестве функции потерь.

Получившаяся в итоге архитектура:

```
inputs = Input((img_rows, img_cols, 1))
conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)
conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)
conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool3)
conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool4)
conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv5)

up6 = concatenate([Conv2DTranspose(256, (2, 2), strides=(2, 2),
padding='same')(conv5), conv4], axis=3)
conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(up6)
conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv6)

up7 = concatenate([Conv2DTranspose(128, (2, 2), strides=(2, 2),
```

```
padding='same')(conv6), conv3], axis=3)
conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(up7)
conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv7)

up8 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(conv7), conv2], axis=3)
conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(up8)
conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv8)

up9 = concatenate([Conv2DTranspose(32, (2, 2), strides=(2, 2),
padding='same')(conv8), conv1], axis=3)
conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(up9)
conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv9)

conv10 = Conv2D(4, (1, 1), activation='sigmoid')(conv9)

model = Model(inputs=[inputs], outputs=[conv10])
```

Так как модель переобучалась после 20 эпох, было принято решение добавить регуляризацию. Архитектура нейронной сети после добавления регуляризации представлена ниже.

```
inputs = Input((img_rows, img_cols, 1))
conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(inputs)
conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

conv2 = Conv2D(64, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(pool1)
conv2 = Conv2D(64, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

conv3 = Conv2D(128, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(pool2)
conv3 = Conv2D(128, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
conv4 = Conv2D(256, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(pool3)
conv4 = Conv2D(256, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)
conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool4)
conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv5)
up6 = concatenate([Conv2DTranspose(256, (2, 2), strides=(2, 2),
padding='same')(conv5), conv4], axis=3)
conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(up6)
conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv6)
up7 = concatenate([Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(conv6), conv3], axis=3)
conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(up7)
conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv7)
up8 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(conv7), conv2], axis=3)
conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(up8)
conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv8)
up9 = concatenate([Conv2DTranspose(32, (2, 2), strides=(2, 2),
padding='same')(conv8), conv1], axis=3)
```

```
conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(up9)
conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv9)
conv10 = Conv2D(4, (1, 1), activation='sigmoid')(conv9)
model = Model(inputs=[inputs], outputs=[conv10])
```

Получили следующий результат:

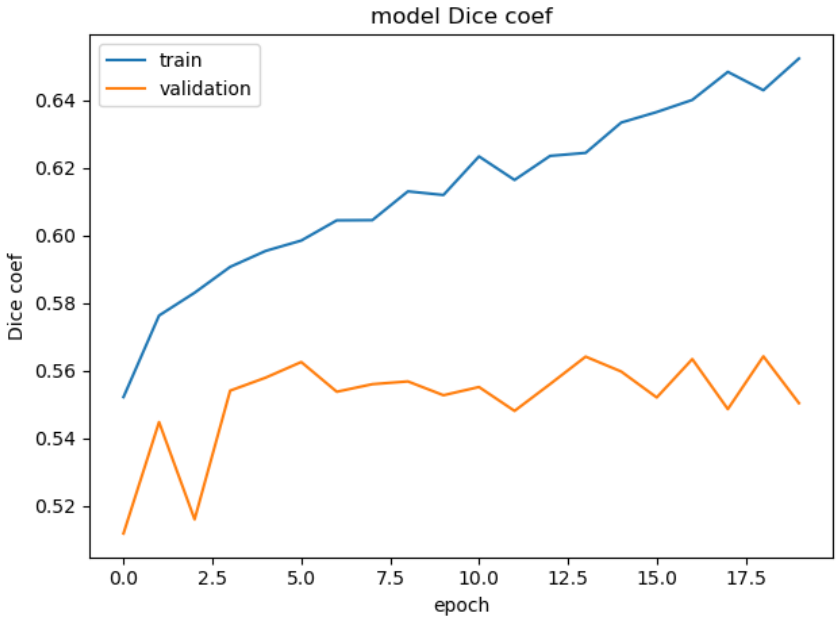

Рисунок 6 - Обучение модели с регуляризацией весов

Регуляризация весов на этих слоях не сильно помогла для достижения хорошего результата.

Примеры выделения дефектов на изображениях:

Рис.7 - Пример сегментирования нейронной сетью.

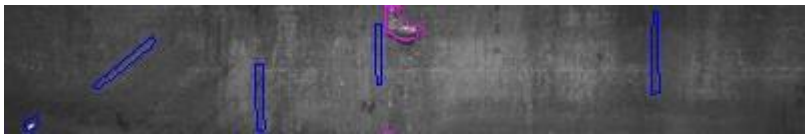

Рис.8. - Правильно сегментированное изображение.

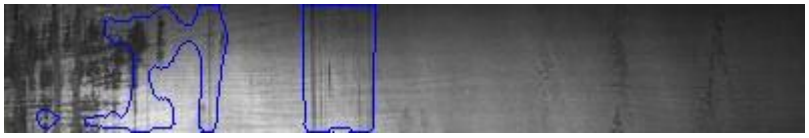

Рис.9 - Пример сегментирования нейронной сетью.

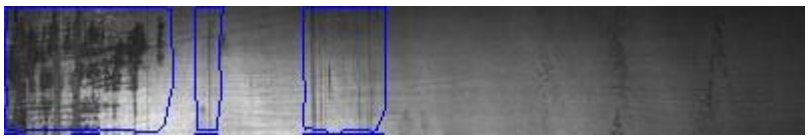

Рис.10 - Правильно сегментированное изображение.

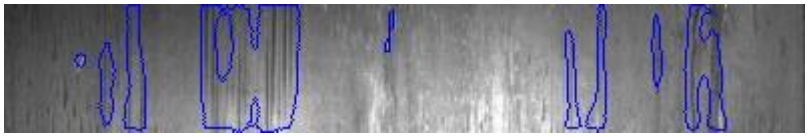

Рис.11 - Пример сегментирования нейронной сетью.

Рис.12 - Правильно сегментированное изображение.

Рис.13 - Пример сегментирования нейронной сетью.

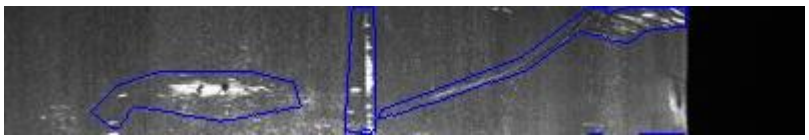

Рис.14 - Правильно сегментированное изображение.

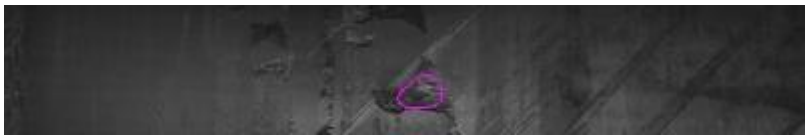

Рис.15 - Пример сегментирования нейронной сетью.

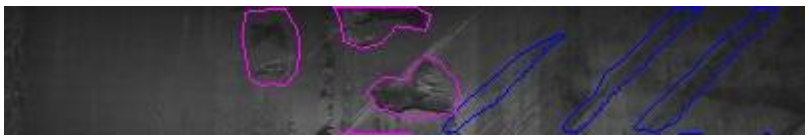

Рис.16 - Правильно сегментированное изображение.

Кто в бригаде за что отвечал:

- Гаврилов А.В. - подготовка данных для обучения нейронной сети, подбор параметров для обучения нейронной сети, модификация выбранной модели.
- Гиззатов А. - преобразовывал изображения для более легкого считывания данных, тестирование функций, выбор модели.

- Бергалиев М. - перекодировал закодированные пиксели маски при уменьшении размеров изображений, декодировал маски из закодированных пикселей в изображения, отрисовка сегментации изображений по маскам, предсказанным сетью.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ.

convert_imgs.py

```
import pandas as pd
import numpy as np
import os
from PIL import Image
from math import floor

def get_mask(enc_pixels, w, h):
    points = []
    mask = np.zeros((h, w))
    for i in range(0, len(enc_pixels)//2):
        start = enc_pixels[2*i]
        run_length = enc_pixels[2*i+1]
        x = (start-1) // h
        y = (start-1) % h
        for j in range(0, run_length):
            mask[y, x] = 1
            x += (y + 1) // h
            y = (y + 1) % h
    return mask

def get_labels(df, w, h, filename):
    data = df.loc[df['ImageId'] == filename]
    masks = np.zeros((h, w, 4))
    for i in range(1, 5):
        if data.values.shape[0] != 0:
            enc_pixels = data.loc[data['ClassId'] ==
i].values
            if enc_pixels.shape[0] != 0:
                enc_pixels = enc_pixels[0][2]
                enc_pixels = np.array(list(map(int,
enc_pixels.split())))
                masks[:, :, i-1] = get_mask(enc_pixels, w, h)
    return masks


def convert(df, directory):
    names = list(map(lambda x: directory + x,
os.listdir(directory)))
    filenames = os.listdir(directory)

    total = len(names)
    im = Image.open(names[0])
    w, h = im.size
    im.close()
    imgs = np.zeros((total, h, w), dtype=np.uint8)
    masks = np.zeros((total, h, w, 4), dtype=np.uint8)
```

```
    for i in range(0, total):
        im = Image.open(names[i])
        im_conv = im.convert('L')
        img = np.array(im_conv, dtype=np.uint8)
        imgs[i] = img
        masks[i] = get_labels(df, w, h, filenames[i])
        if i % 100 == 0:
            print('{}/{}'.format(i, total))
    np.save('imgs.npy', imgs)
    np.save('masks.npy', masks)


def main():
    df = pd.read_csv('new_train.csv', sep=',')
    convert(df, 'dataset\\new_train_images\\')

if __name__ == '__main__':
    main()
```

image_comression.py

```
from PIL import Image
import os

def scale_image(input_image_path,
                output_image_path,
                width=None,
                height=None
                ):
    original_image = Image.open(input_image_path)
    w, h = original_image.size
    '''print('The original image size is {wide} wide x
{height} '
          'high'.format(wide=w, height=h)) '''

    if width and height:
        max_size = (width, height)
    elif width:
        max_size = (width, h)
    elif height:
        max_size = (w, height)
    else:
        # No width or height specified
        raise RuntimeError('Width or height required!')

    original_image.thumbnail(max_size, Image.ANTIALIAS)
    original_image.save(output_image_path)

    scaled_image = Image.open(output_image_path)
    width, height = scaled_image.size
    ''' print('The scaled image size is {wide} wide x
{height} '
          'high'.format(wide=width, height=height)) '''
```

```
if __name__ == '__main__':
    tree = os.listdir('train_images')
    if os.path.exists('./new_train_images'):
        print("already exists!")
    else:
        print("making new_train_images")
        os.mkdir('new_train_images')
        for i in tree:
            scale_image(input_image_path = 'train_images/'+i,
                output_image_path='new_train_images/'+i,
                width=800,height=128)
```

image_marking.py

```
import numpy as np
from PIL import Image, ImageDraw

def get_borders(mask, w, h):
    points = []
    for i in range(0, h):
        for j in range(0, w):
            if round(mask[i, j]) == 1:
                isBorder = False
                isBorder = j == 0
                isBorder |= j-1 >= 0 and round(mask[i][j-1])
== 0
                isBorder |= i == 0
                isBorder |= i-1 >= 0 and round(mask[i-1][j])
== 0
                isBorder |= j == w-1
                isBorder |= j+1 < w and round(mask[i][j+1])
== 0
                isBorder |= i == h-1
                isBorder |= i+1 < h and round(mask[i+1][j])
== 0
                if isBorder:
                    points.append((j, i))
    return points


def draw_masks(image_arr, masks, save_as):
    im = None
    im = Image.fromarray(image_arr)
    im = im.convert('RGB')
    (w, h) = im.size
    draw = ImageDraw.Draw(im)
    fill = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 0,
255)]
    for i in range(0, 4):
        points = get_borders(masks[:, :, i], w, h)
        draw.point(points, fill=fill[i])
```

```
        im.save(save_as, "PNG")
        del draw
        im.close()
```

main.py

```
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, concatenate, Conv2D,
MaxPooling2D, Conv2DTranspose
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras import backend as K
from image_marking import draw_masks
from sklearn.model_selection import train_test_split
from keras.regularizers import l2

l2_lambda = 0.0001

img_rows = 64
img_cols = 400


def load_data():
   data = np.load('imgs.npy')
   target = np.load('masks.npy')
   print(data.shape)
   total, w, h = data.shape
   data = data.reshape((total, w, h, 1))
   train_data, test_data, train_target, test_target =
train_test_split(data, target, test_size=0.1, random_state=42)
   return (train_data, train_target, test_data, test_target)

smooth = 1.


def dice_coef(y_true, y_pred):
   y_true_f = K.flatten(y_true)
   y_pred_f = K.flatten(y_pred)
   intersection = K.sum(y_true_f * y_pred_f)
   return (2. * intersection + smooth) / (K.sum(y_true_f) +
K.sum(y_pred_f) + smooth)


def dice_coef_loss(y_true, y_pred):
   return -dice_coef(y_true, y_pred)


def get_unet():
   inputs = Input((img_rows, img_cols, 1))
   conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(inputs)
```

```
   conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(conv1)
   pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

   conv2 = Conv2D(64, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(pool1)
   conv2 = Conv2D(64, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(conv2)
   pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

   conv3 = Conv2D(128, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(pool2)
   conv3 = Conv2D(128, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(conv3)
   pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

   conv4 = Conv2D(256, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(pool3)
   conv4 = Conv2D(256, (3, 3), activation='relu', padding='same',
activity_regularizer=l2(l2_lambda))(conv4)
   pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

   conv5 = Conv2D(512, (3, 3), activation='relu',
padding='same')(pool4)
   conv5 = Conv2D(512, (3, 3), activation='relu',
padding='same')(conv5)

   up6 = concatenate([Conv2DTranspose(256, (2, 2), strides=(2, 2),
padding='same')(conv5), conv4], axis=3)
   conv6 = Conv2D(256, (3, 3), activation='relu',
padding='same')(up6)
   conv6 = Conv2D(256, (3, 3), activation='relu',
padding='same')(conv6)

   up7 = concatenate([Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(conv6), conv3], axis=3)
   conv7 = Conv2D(128, (3, 3), activation='relu',
padding='same')(up7)
   conv7 = Conv2D(128, (3, 3), activation='relu',
padding='same')(conv7)

   up8 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(conv7), conv2], axis=3)
   conv8 = Conv2D(64, (3, 3), activation='relu',
padding='same')(up8)
   conv8 = Conv2D(64, (3, 3), activation='relu',
padding='same')(conv8)

   up9 = concatenate([Conv2DTranspose(32, (2, 2), strides=(2, 2),
padding='same')(conv8), conv1], axis=3)
   conv9 = Conv2D(32, (3, 3), activation='relu',
padding='same')(up9)
```

```
    conv9 = Conv2D(32, (3, 3), activation='relu',
padding='same')(conv9)

    conv10 = Conv2D(4, (1, 1), activation='sigmoid')(conv9)

    model = Model(inputs=[inputs], outputs=[conv10])

    model.load_weights('start_weights.h5')

    model.compile(optimizer=Adam(lr=1e-5), loss=dice_coef_loss,
metrics=[dice_coef])

    return model

def plot_dice_coef(history):
    dice_coef = history.history['dice_coef']
    val_dice_coef = history.history['val_dice_coef']
    plt.plot(dice_coef)
    plt.plot(val_dice_coef)
    plt.title('model Dice coef')
    plt.ylabel('Dice coef')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()


def train_and_predict():
    imgs_train, imgs_mask_train, imgs_test, imgs_mask_test =
load_data()

    imgs_train = imgs_train.astype('float32')
    mean = np.mean(imgs_train)
    std = np.std(imgs_train)
    imgs_train -= mean
    imgs_train /= std

    imgs_mask_train = imgs_mask_train.astype('float32')

    model = get_unet()
    model_checkpoint = ModelCheckpoint('reserve/weights.h5',
monitor='val_loss', save_best_only=True)

    history = model.fit(imgs_train, imgs_mask_train, batch_size=32,
epochs=20, verbose=1, shuffle=True,
              validation_split=0.2, callbacks=[model_checkpoint])
    model.save('model.h5')

    plot_dice_coef(history)
    imgs_test = imgs_test.astype('float32')

    imgs_test -= mean
    imgs_test /= std
```

```
    imgs_mask_test_res = model.predict(imgs_test, verbose=1)

    indices = np.random.choice(imgs_test.shape[0], 5)

    for n, i in enumerate(indices):
        draw_masks((imgs_test[i, :, :, 0]*std +
mean).astype('uint8'),
                   imgs_mask_test_res[i], '{}.png'.format(n))
        draw_masks((imgs_test[i, :, :, 0]*std +
mean).astype('uint8'),
                   imgs_mask_test[i], '{}_true.png'.format(n))


if __name__ == '__main__':
    train_and_predict()
```