МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«Санкт–Петербургский государственный университет  
аэрокосмического приборостроения»

Кафедра №43 «Компьютерных технологий и программной инженерии»

ОТЧЁТ ПО ПРАКТИКЕ

ЗАЩИЩЁН С ОЦЕНКОЙ

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ст. преподаватель |  |  |  | С.А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЁТ ПО ПРАКТИКЕ

|  |  |  |
| --- | --- | --- |
| вид практики | производственная | |
| тип практики | технологическая (проектно-технологическая) | |
| на тему индивидуального задания | | Программная реализация алгоритма сегментации |
| изображения с помощью нейронной сети | | | |
|  | | | |

|  |  |
| --- | --- |
| выполнен | Ковалев Даниил Владимирович |
| фамилия, имя, отчество обучающегося в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 09.03.04 |  | Программная инженерия |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности | 02 |  | Проектирование программных систем |
|  | код |  | наименование направленности |
|  | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Обучающийся группы № | 4133 |  | C:\Users\DANYA\Downloads\signature (1).png21.07.23 |  | Ковалев Д.В |
|  | номер |  | подпись, дата |  | инициалы, фамилия |

Санкт–Петербург 2023

Оглавление

[**1. Цель работы в соответствии с вариантом** 3](#_Toc140445153)

[**2. Исходные данные и их описание** 3](#_Toc140445154)

[**3. Теоретический раздел** 4](#_Toc140445155)

[**4. Практический раздел** 13](#_Toc140445158)

[**5.** **Результаты сегментации изображений с помощью нейронной сети** 15](#_Toc140445159)

[**6. Выводы** 16](#_Toc140445160)

[**7. Список литературы** 17](#_Toc140445161)

[**ПРИЛОЖЕНИЕ** 18](#_Toc140445162)

[**Листинг программы** 18](#_Toc140445163)

1. **Цель работы в соответствии с вариантом**

Целью данной работы является создание на языке Python программной реализации алгоритма сегментации наводнений на изображениях с помощью свёрточной нейронной сети.

1. **Исходные данные и их описание**

В данной работе исходными данными являются изображения в формате jpg и png.

**JPEG (*Joint Photographic Experts Group)*** — один из популярных растровых графических форматов, применяемый для хранения фотографий и подобных им изображений. Файлы, содержащие данные JPEG, обычно имеют расширения (суффиксы) **.jpg** (самое популярное), **.jfif**, **.jpe** или **.jpeg**. MIME-тип — image/jpeg.

Алгоритм JPEG позволяет сжимать изображение как с потерями, так и без потерь (режим сжатия lossless JPEG). Поддерживаются изображения с линейным размером не более 65535 × 65535 пикселов.

**PNG (Portable Network Graphics)** — это растровый формат изображений, который широко используется в области иллюстрации и дизайна наряду с JPEG. Формат позволяет хранить графику с практически неограниченным количеством цветов в отличие от, например, GIF, имеющего 8-битный цвет (всего 256 цветов).

Исходные данные содержат в себе два вида изображений: оригинальное фото наводнения, и изображение такого же размера с маской этого наводнения.

**Пример исходных данных:**

Рисунок 1 – Фото наводнения(**.jpg**) Рисунок 2 – Маска наводнения(**.png**)

Всего исходные данные содержат 520 изображения, из них:

* 290 фотографий наводнений для обучения нейронной сети и тестеровки
* 290 масок наводнений для обучений нейронной сети.

1. **Теоретический раздел**

**Сегментация** — это разбиение изображения на множество покрывающих его областей

**Нейронная сеть** — это последовательность нейронов, соединенных между собой синапсами.

**Нейрон** — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной (синий), скрытый (красный) и выходной (зеленый). В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин **слоя**.

Существует входной слой, который получает информацию; n скрытых слоев, которые ее обрабатывают и выходной слой, который выводит результат. У каждого из нейронов есть 2 основных параметра: входные данные (input data) и выходные данные (output data). В случае входного нейрона: input=output. В остальных, в поле input попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации

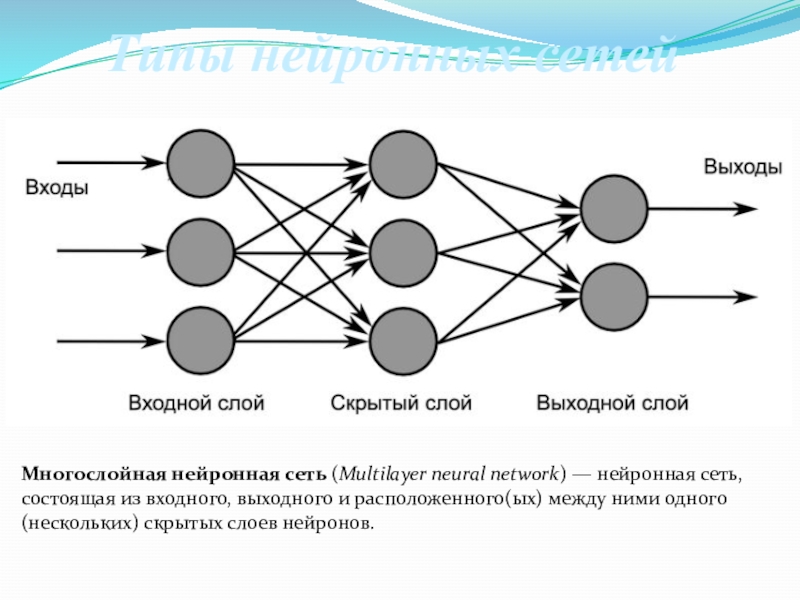


Рисунок 3 – Виды слоев в нейросети

**Синапс -** это связь между двумя нейронами. У синапсов есть 1 параметр — вес. Благодаря ему входная информация изменяется, когда передается от одного нейрона к другому.

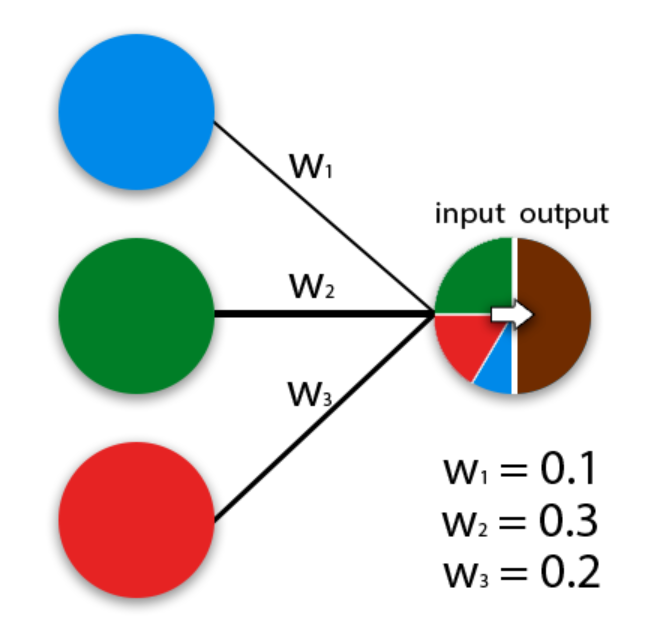


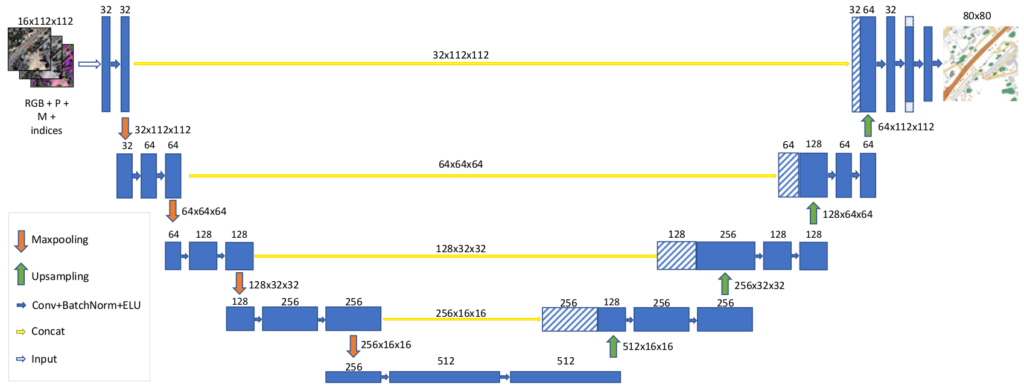
Рисунок 4 – структура синапса

**Функция активации** — это способ нормализации входных данных. Если на входе у вас будет большое число, пропустив его через функцию активации, вы получите выход в нужном вам диапазоне

**Устройство свёрточной нейронной сети**

Сверточные нейронные сети обеспечивают частичную устойчивость к изменениям масштаба, смещениям, поворотам, смене ракурса и прочим искажениям. Сверточные нейронные сети объединяют три архитектурных идеи, для обеспечения инвариантности к изменению масштаба, повороту, повороту, сдвигу и пространственным искажениям:

* локальные рецепторные поля (обеспечивают локальную двумерную связность нейронов);
* общие синаптические коэффициенты (обеспечивают детектирование некоторых черт в любом месте изображения и уменьшают общее число весовых коэффициентов);
* иерархическая организация с пространственными подвыборками.



**Слой свёртки** (англ. convolutional layer) — это основной блок свёрточной нейронной сети. Слой свёртки включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты поэлементного произведения для каждого фрагмента). Весовые коэффициенты ядра свёртки (небольшой матрицы) неизвестны и устанавливаются в процессе обучения.

Результатом свертки изображений является новое изображение, полученное путем применения фильтра к каждому пикселю исходного изображения. Этот процесс позволяет выделить определенные характеристики изображения, такие как границы объектов или текстуры, и уменьшить количество информации, несущественной для анализа. Результат свертки может использоваться для различных задач обработки изображений, таких как распознавание образов, улучшение качества изображения и сегментация.

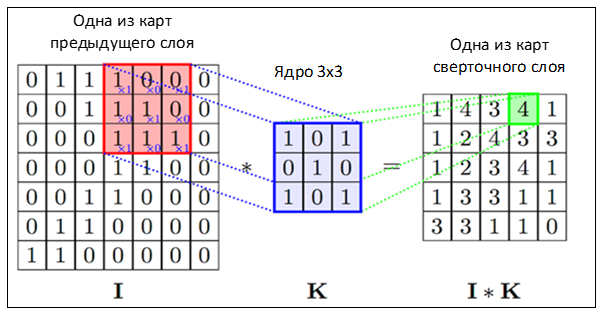


Рисунок 6 – Свертка изображения

**Пулинговый слой** призван снижать размерность изображения. Исходное изображение делится на блоки размером w×h и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума (англ. max pooling). Обучаемых параметров у этого слоя нет. Пулинговый слой в нейронной сети выполняет функцию уменьшения размерности признакового пространства. Он сглаживает данные, удаляет шум, а также позволяет выделить наиболее важные признаки. Кроме того, пулинговый слой улучшает инвариантность к небольшим изменениям входных данных и делает сеть более устойчивой к переобучению. В зависимости от типа пулинга, слой может выполнять различные функции, такие как максимальное или среднее значение, выбор самого большого или самого маленького элемента и т.д.

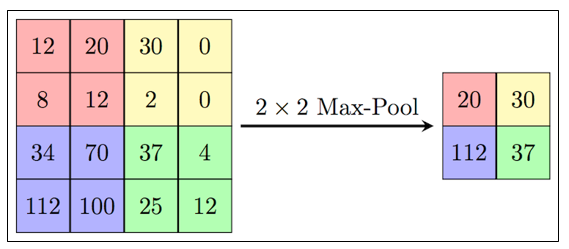


Рисунок 7 – Работа пулингового слоя

**Слой конкатенации**

Слой конкатенации в свёрточной нейронной сети объединяет несколько карт признаков в одну карту признаков большей размерности. Это позволяет сети использовать информацию из нескольких источников и улучшить качество предсказаний. Обычно слой конкатенации используется в конце блока сверток, перед передачей карты признаков на следующий уровень сети.

**Выходной слой**

Слой нейронной сети, на выходах которого формируется результат (отклик сети на входное воздействие). Нейроны выходного слоя так же, как и скрытых слоёв, производят обработку данных. Число составляющих его нейронов определяется количеством зависимых переменных модели.

**Функции активации**

**Сигмоидная функция активации** - это нелинейная функция, которая преобразует входное значение в диапазоне от отрицательной бесконечности до положительной бесконечности в значение от 0 до 1. Эта функция активации часто используется в нейронных сетях для задач бинарной классификации.

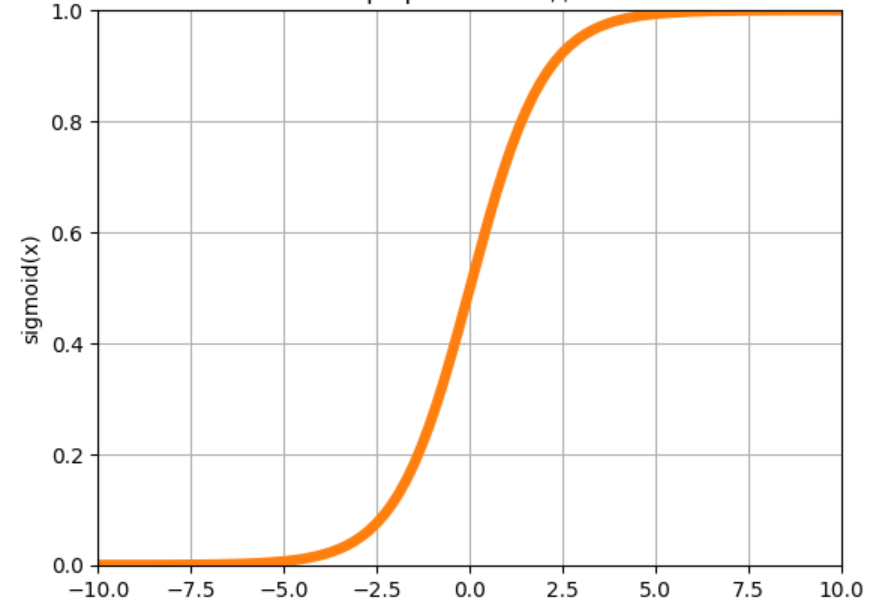


Рисунок 8 – График сигмоиды

Сигмоидная функция активации используется для преобразования выходного значения нейрона в вероятность, т.е. вероятность того, что входное значение относится к классу 1, если мы работаем с задачей бинарной классификации. Если значение сигмоидной функции близко к 1, то вероятность того, что входное значение относится к классу 1, высока. Если значение близко к 0, то вероятность того, что входное значение относится к классу 1, низкая.

**ReLU (Rectified Linear Unit)** - это нелинейная функция активации, которая широко используется в глубоком обучении. Она преобразует входное значение в значение от 0 до положительной бесконечности. Если входное значение меньше или равно нулю, то ReLU выдает ноль, в противном случае - входное значение.

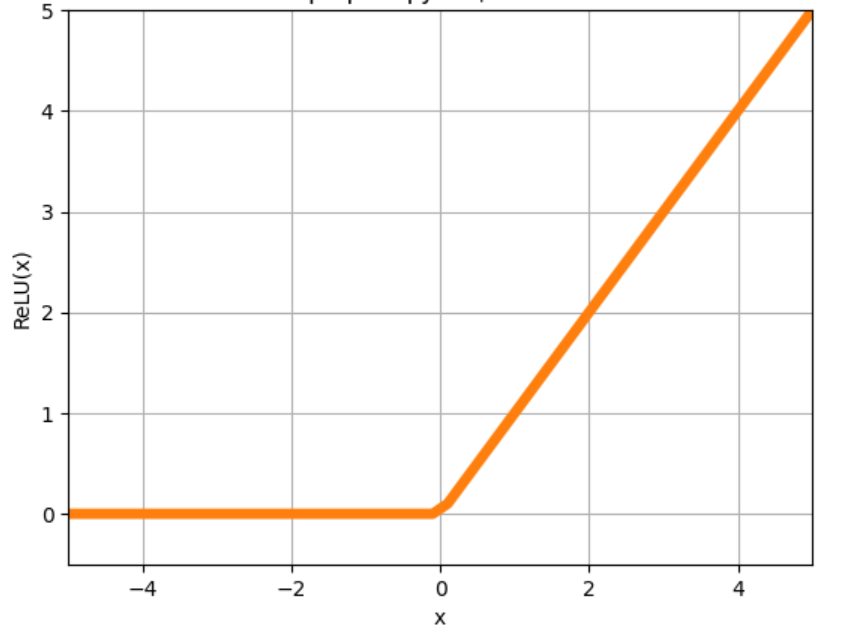


Рисунок 9 – График функции ReLU

ReLU имеет несколько преимуществ по сравнению со сигмоидной функцией активации. Во-первых, ReLU более вычислительно эффективна, поскольку она является простой и быстрой операцией, которая не требует вычисления экспоненты. Во-вторых, ReLU решает проблему затухания градиента, так как она не вызывает затухания градиента при обратном распространении ошибки, как это происходит в случае с сигмоидной функцией активации.

Однако, ReLU имеет некоторые недостатки. Во-первых, при использовании ReLU, некоторые нейроны могут "умереть" (dead neurons), т.е. они могут получить отрицательное значение и оставаться неактивными на всем протяжении обучения. Во-вторых, ReLU несимметрична относительно нуля, поэтому может возникнуть проблема "расслоения" (clustering), когда нейроны могут выдавать только положительные значения.

**Батч нормализация**

Batch Normalization (батч-нормализация) - это метод нормализации входных данных в нейронной сети, который улучшает скорость и качество обучения. Он заключается в нормализации каждого батча входных данных, передаваемых между слоями нейронной сети.

Формула для батч-нормализации:

где - входные данные, и - среднее значение и стандартное отклонение по батчу, e - малое число для стабильности.

Батч-нормализация позволяет ускорить обучение нейронной сети, так как она уменьшает внутреннее смещение (internal covariate shift) - изменение распределения входных данных внутри нейронной сети. Это позволяет использовать более высокие скорости обучения и улучшает обобщающую способность модели.

Батч-нормализация широко используется в глубоком обучении для улучшения производительности нейронных сетей. Она может быть применена к различным типам слоев, включая полносвязные слои, сверточные слои и рекуррентные слои.

**Оптимизатор Adam**

Adam (Adaptive Moment Estimation) - это метод оптимизации градиентного спуска, который используется для обучения нейронных сетей. Он сочетает в себе два других метода оптимизации - Momentum и RMSprop.

Adam подстраивает скорость обучения (learning rate) для каждого параметра на основе оценок первого и второго моментов градиента. Это позволяет ему быстрее сходиться к оптимальному решению и избежать проблемы с затуханием градиента.

Формула для обновления весов в Adam:

где - вектор параметров на шаге , - скорость обучения, и - оценки первого и второго моментов градиента на шаге , - малое число для стабильности.

Adam широко используется в глубоком обучении для обновления весов нейронных сетей. Он показывает хорошие результаты на различных типах задач, включая классификацию изображений, обработку естественного языка и машинный перевод.

**Бинарная кросс-энтропия**

Бинарная кросс-энтропия - это функция потерь, которая широко используется при обучении нейронных сетей для бинарной классификации. Она измеряет расхождение между предсказанными значениями и истинными значениями целевой переменной.

Формула для бинарной кросс-энтропии:

где y - истинное значение целевой переменной (0 или 1), - предсказанное значение (вещественное число между 0 и 1).

В обучении нейронных сетей, бинарная кросс-энтропия используется в качестве функции потерь для задачи бинарной классификации. Она минимизируется в процессе обучения, чтобы уменьшить расхождение между предсказанными значениями и истинными значениями целевой переменной. Оптимизация функции потерь происходит при помощи метода градиентного спуска, который позволяет обновлять веса нейронной сети в соответствии с градиентом функции потерь.

**Точность accuracy**

Accuracy (точность) - это метрика качества, которая используется для оценки производительности нейронной сети в задачах классификации. Она показывает, какая доля правильных ответов была получена на тестовом наборе данных.

Формула для accuracy:

Accuracy = Количество правильных предсказаний / Общее количество предсказаний

Accuracy является одной из наиболее распространенных метрик качества в обучении нейронных сетей. Она позволяет быстро оценить производительность модели, особенно в задачах сбалансированной классификации.

1. **Практический раздел**

**Часть 1. Создание модели и обучение нейронной сети**

1. Импортируются необходимые библиотеки:

* numpy
* pandas
* os
* cv2
* tensorflow
* keras

Библиотека numpy используется для работы с массивами и вычислений.

Pandas используется для работы с данными, в данном случае для чтения CSV-файла.

Ос используется для работы с файловой системой.

Библиотека cv2 используется для обработки изображений.

Библиотеки tensorflow и keras используются для работы с нейронными сетями и обработкой изображений.

2. Задаются пути к папкам с изображениями и масками, а также к файлу metadata.csv. Затем считывается CSV-файл в переменную data.

3. Задается размер изображений img\_dim.

4. Создаются генераторы для обучающего и валидационного наборов данных. Генераторы используются для автоматической подгрузки данных в нейронную сеть во время обучения.

5. Создается функция create\_data, которая будет использоваться для создания списка путей к изображениям и маскам из заданной директории.

6. Создаются две функции для чтения изображений load\_image.

7. Создаются функции вывода изображения и соответствующей маски:

* show\_image – Вывод оригинального изображения
* show\_mask – Вывод соответствующей маски
* show\_images – Совместный вывод фотографий с наложением

8. Создание графика истории обучения, а также инициализирование параметров для оперативного отслеживания хода обучения

9. В качестве базовой модели используется предобученная модель Unet

10.Скомпилируем модель созданной нейронной сети.

С помощью метода compile скомпилируем модель, в качестве оптимизатора градиентного спуска передадим метод “Adam”, бинарную кросс-энтропию в качестве функции потерь и метрику качества accuracy.

1. Обучим модель с помощью метода fit библиотеки keras.

В качестве набора обучающих данных передадим созданную функцию генератора батчей, размер батча установим 40, зададим параметр steps\_per\_epoch = количество изображений для обучений / размер батча, значение количества эпох – 100,

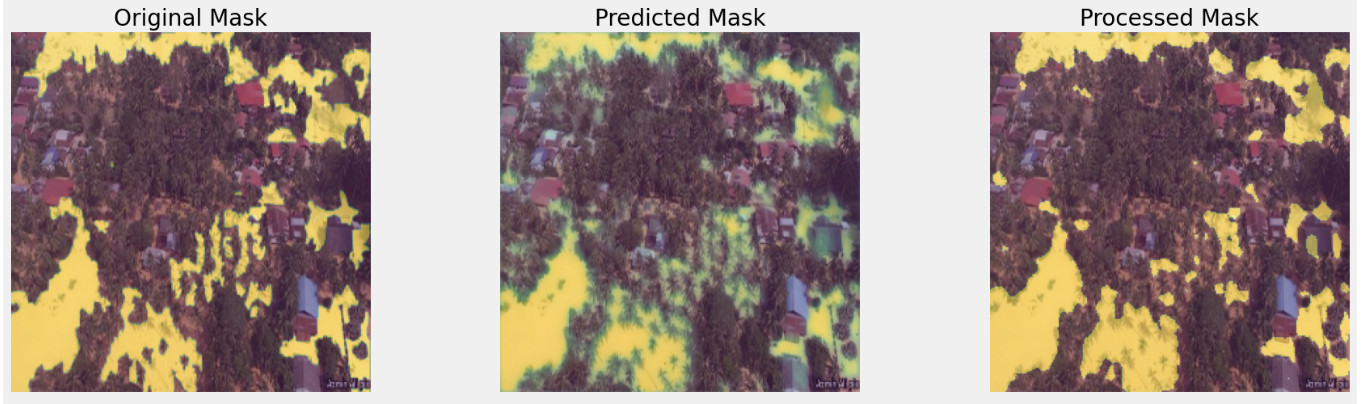
**Часть 2.** **Тестирование предсказаний нейронной сети**

1. Создание функции получений тестовых изображений(не участвующих ни в обучении, ни в валидации) и загрузка изображений в нейронную сеть

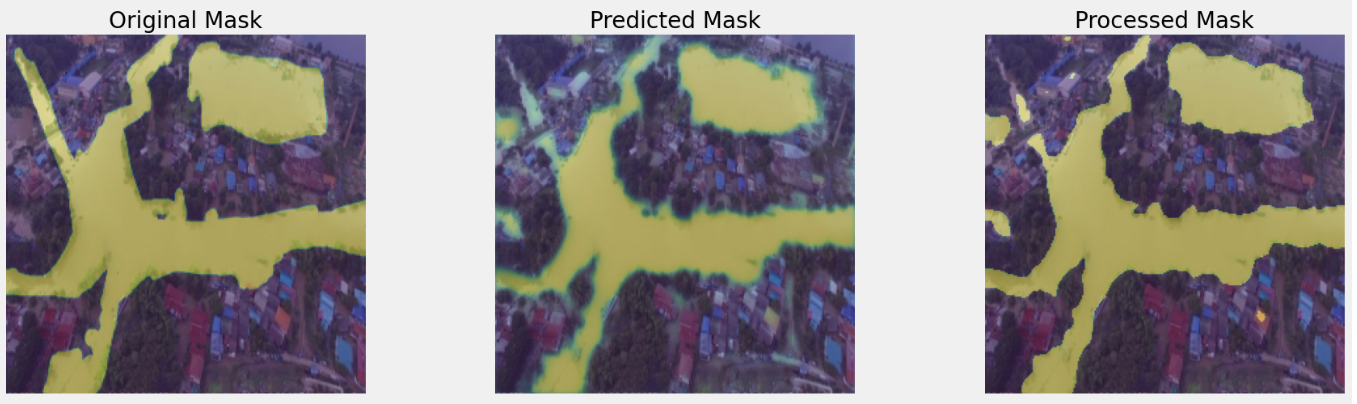
2. Вывод исходных изображений и сегментации изображений с помощью нейронной сети с помощью библиотеки pyplot

1. **Результаты сегментации изображений с помощью нейронной сети**

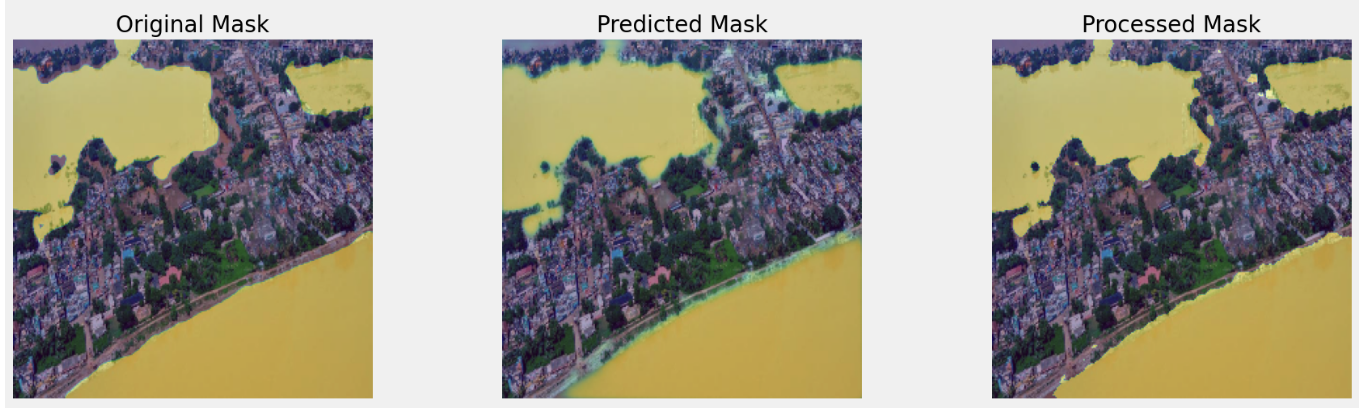
**Примеры успешной сегментации:**



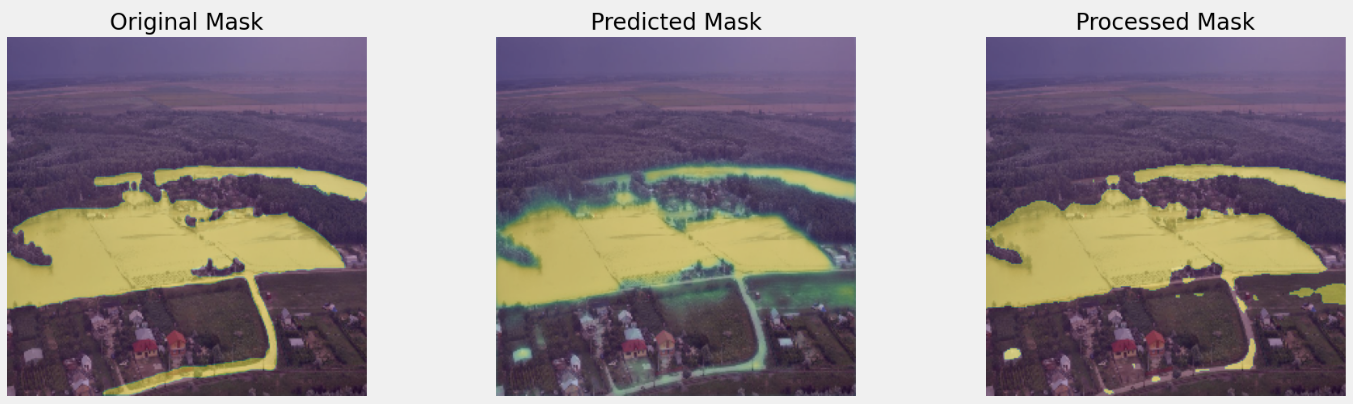
Оригинальное изображение Предполагаемая “ Маска” Выходное изображение



Оригинальное изображение Предполагаемая “ Маска” Выходное изображение



Оригинальное изображение Предполагаемая “ Маска” Выходное изображение



Оригинальное изображение Предполагаемая “ Маска” Выходное изображение

**Примеры неудачной сегментации:**



Оригинальное изображение Предполагаемая “ Маска” Выходное изображение

Данные ошибки нейронной сети можно объяснить тем, что обучающих данных было не достаточно до идеальной аналитики фотографий, однако даже не смотря на это нейросеть показывает достаточно точный результат.

1. **Выводы**

В данной работе я создал на языке Python программную реализацию алгоритма на изображениях затоплений с помощью свёрточной нейронной сети. Я узнал об устройстве нейронных сетей и об их разновидностях. Закрепил знания языка python и научился работать с библиотеками Python.

1. **Список литературы**

1. Ф.М. ГАФАРОВ, А.Ф. ГАЛИМЯНОВ “ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ И ИХ ПРИЛОЖЕНИЯ”  
2. [Интерактивная сегментация](https://habr.com/ru/companies/samsung/articles/508342/) (habr.com)

3. [Сегментация изображений](https://neerc.ifmo.ru/wiki/index.php?title=%D1%E5%E3%EC%E5%ED%F2%E0%F6%E8%FF_%E8%E7%EE%E1%F0%E0%E6%E5%ED%E8%E9OWASP%20API%20Security%20) (neerc.ifmo.ru)

4. [Распознавание изображений на Python с помощью TensorFlow и Keras](https://evileg.com/ru/post/619/)(evileg.com)

5. [Обнаружение объекта на изображении методом цветовой сегментации (Python)](https://waksoft.susu.ru/2019/05/26/obnaruzhenie-obekta-na-izobrazhenii-opirajas-na-cvetovuju-segmentacii-python/)( waksoft.susu.ru)

6. [Нейронные сети на Python](https://proproprogs.ru/neural_network/struktura-i-princip-raboty-polnosvyaznyh-neyronnyh-setey)(proproprogs.ru)

**ПРИЛОЖЕНИЕ A**

**Листинг программы(Jupyter Notebook)**

import numpy as np

import pandas as pd

import os

for dirname, \_, filenames in os.walk('/Zatop'):

for filename in filenames:

print(os.path.join(dirname, filename))

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import random

import cv2

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPool2D, Conv2DTranspose, Concatenate, Input, Dropout

from tensorflow.keras.models import Model

from keras.models import \*

from keras.layers import \*

from keras.optimizers import \*

base\_directory = '/Zatop/Input

images\_folder = os.path.join(base\_directory, 'zatopp')

masks\_folder = os.path.join(base\_directory, 'masks')

csvfilename = r'metadata.csv'

csvfilename = csvfilename.strip('\u202a')

basename=os.path.basename(csvfilename)

filenamex = os.path.splitext(basename)[0]

#Read CSV -файл

data = pd.read\_csv(csvfilename, encoding = 'gbk')

data = pd.read\_csv(csvfilename)

data

img\_dim = 256

image\_datagen = ImageDataGenerator(rescale=1./255,validation\_split=0.15)

mask\_datagen = ImageDataGenerator(rescale=1./255,validation\_split=0.15)

train\_image\_generator = image\_datagen.flow\_from\_directory(

'/Input/zatopp',

target\_size=(img\_dim, img\_dim),

class\_mode = None,

classes = ['images'],

batch\_size = 32,

seed=42,

subset='training')

train\_mask\_generator = mask\_datagen.flow\_from\_directory(

'/Input/Mask',

target\_size=(img\_dim, img\_dim),

class\_mode = None,

classes = ['masks'],

color\_mode = 'grayscale',

batch\_size = 32,

seed=42,

subset='training')

val\_image\_generator = image\_datagen.flow\_from\_directory(

'/Input/zatopp',

target\_size=(img\_dim, img\_dim),

class\_mode = None,

classes = ['images'],

batch\_size = 32,

seed=42,

subset='validation')

val\_mask\_generator = mask\_datagen.flow\_from\_directory(

'/Input/Mask',

target\_size=(img\_dim, img\_dim),

class\_mode = None,

classes = ['masks'],

color\_mode = 'grayscale',

batch\_size = 32,

seed=42,

subset='validation')

train\_generator = zip(train\_image\_generator, train\_mask\_generator)

val\_generator = zip(val\_image\_generator, val\_mask\_generator)

def create\_data(data\_dir):

image\_paths = []

mask\_paths = []

folds = sorted(os.listdir(data\_dir))

for fold in folds:

foldpath = os.path.join(data\_dir, fold)

if fold in ['zatopp']:

images = sorted(os.listdir(foldpath))

for image in images:

fpath = os.path.join(foldpath, image)

image\_paths.append(fpath)

elif fold in ['mask', 'Mask', 'masks', 'Masks', 'MASKS']:

masks = sorted(os.listdir(foldpath))

for mask in masks:

fpath = os.path.join(foldpath, mask)

mask\_paths.append(fpath)

else:

continue

return image\_paths, mask\_paths

def load\_image(image, SIZE):

return np.round(tf.image.resize(img\_to\_array(load\_img(image)) / 255., (SIZE, SIZE)), 4)

def load\_images(image\_paths, SIZE, mask=False, trim=None):

if trim is not None:

image\_paths = image\_paths[:trim]

if mask:

images = np.zeros(shape=(len(image\_paths), SIZE, SIZE, 1))

else:

images = np.zeros(shape=(len(image\_paths), SIZE, SIZE, 3))

for i, image in enumerate(image\_paths):

img = load\_image(image, SIZE)

if mask:

images[i] = img[:, :, :1]

else:

images[i] = img

return images

def show\_image(image, title=None, cmap=None, alpha=1):

plt.imshow(image, cmap=cmap, alpha=alpha)

if title is not None:

plt.title(title)

plt.axis('off')

def show\_mask(image, mask, cmap=None, alpha=0.4):

plt.imshow(image)

plt.imshow(tf.squeeze(mask), cmap=cmap, alpha=alpha)

plt.axis('off')

def show\_images(imgs, msks):

plt.figure(figsize=(13,8))

for i in range(15):

plt.subplot(3,5,i+1)

id = np.random.randint(len(imgs))

show\_mask(imgs[id], msks[id], cmap='binary')

plt.tight\_layout()

plt.show()

def plot\_training(hist):

# Define needed variables

tr\_acc = hist.history['accuracy']

tr\_loss = hist.history['loss']

val\_acc = hist.history['val\_accuracy']

val\_loss = hist.history['val\_loss']

index\_loss = np.argmin(val\_loss)

val\_lowest = val\_loss[index\_loss]

index\_acc = np.argmax(val\_acc)

acc\_highest = val\_acc[index\_acc]

Epochs = [i+1 for i in range(len(tr\_acc))]

loss\_label = f'best epoch= {str(index\_loss + 1)}'

acc\_label = f'best epoch= {str(index\_acc + 1)}'

# Plot training history

plt.figure(figsize= (20, 8))

plt.style.use('fivethirtyeight')

plt.subplot(1, 2, 1)

plt.plot(Epochs, tr\_loss, 'r', label= 'Training loss')

plt.plot(Epochs, val\_loss, 'g', label= 'Validation loss')

plt.scatter(index\_loss + 1, val\_lowest, s= 150, c= 'blue', label= loss\_label)

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(Epochs, tr\_acc, 'r', label= 'Training Accuracy')

plt.plot(Epochs, val\_acc, 'g', label= 'Validation Accuracy')

plt.scatter(index\_acc + 1 , acc\_highest, s= 150, c= 'blue', label= acc\_label)

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout

plt.show()

image\_size = (256, 256, 3)

def unet(sz=image\_size):

x = Input(sz)

inputs = x

#downsampling

f = 8 #number of filters

layers = []

for i in range(6):

x = Conv2D(f, 3, activation='relu',padding='same')(x)

x = Conv2D(f, 3, activation='relu',padding='same')(x)

layers.append(x)

x = MaxPooling2D()(x)

f = f\*2

ff2 = 64

#bottleneck

j = len(layers)-1

x = Conv2D(f, 3, activation='relu',padding='same')(x)

x = Conv2D(f, 3, activation='relu',padding='same')(x)

x = Conv2DTranspose(ff2, 2, strides=(2,2),padding='same')(x)

x = Concatenate(axis=3)([x,layers[j]])

j = j-1

#upsampling

for i in range(5):

ff2 = ff2//2

f = f//2

x = Conv2D(f, 3, activation='relu',padding='same')(x)

x = Conv2D(f, 3, activation='relu',padding='same')(x)

x = Conv2DTranspose(ff2, 2, strides=(2,2),padding='same')(x)

x = Concatenate(axis=3)([x,layers[j]])

j = j-1

#classification

x = Conv2D(f, 3, activation='relu',padding='same')(x)

x = Conv2D(f, 3, activation='relu',padding='same')(x)

outputs = Conv2D(1, 1, activation = 'sigmoid')(x)

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

return model

model = unet()

model.summary()

data\_dir = 'Input'

image\_paths, mask\_paths = create\_data(data\_dir)

len(image\_paths)

len(mask\_paths)

SIZE = 256

imgs = load\_images(image\_paths, SIZE)

msks = load\_images(mask\_paths, SIZE, mask=True)

imgs.shape

msks.shape

show\_images(imgs, msks)

model.compile(optimizer = Adam(lr = 1e-4), loss = 'binary\_crossentropy', metrics = ['accuracy'])

batch\_size = 40

epochs = 100

ask\_epoch = 5

SPE = len(imgs)//batch\_size

history = model.fit(

imgs, msks,

validation\_split=0.2,

epochs=epochs,

verbose=1,

steps\_per\_epoch=SPE,

batch\_size=batch\_size

)

#plot\_training(history)

plt.figure(figsize=(20,25))

n=0

for i in range(1,(5\*3)+1):

plt.subplot(5,3,i)

if n==0:

id = np.random.randint(len(imgs))

image = imgs[id]

mask = msks[id]

pred\_mask = model.predict(image[np.newaxis,...])

plt.title("Original Mask")

show\_mask(image, mask)

n+=1

elif n==1:

plt.title("Predicted Mask")

show\_mask(image, pred\_mask)

n+=1

elif n==2:

pred\_mask = (pred\_mask>0.5).astype('float')

plt.title("Processed Mask")

show\_mask(image, pred\_mask)

n=0

plt.tight\_layout()

plt.show()