МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

Кафедра компьютерных технологий и программной инженерии

ОТЧЁТ ПО ПРАКТИКЕ
ЗАЩИЩЁН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

Ст. преп.		М.Д. Поляк
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЁТ ПО ПРАКТИКЕ

вид практики производственная

тип практики по получению профессиональных умений и опыта профессиональной деятельности обучающегося направления подготовки

на тему индивидуального задания Определение контуров виноградников на спутниковых снимках, создание набора данных и нейросети для обнаружения контуров виноградников по RGB-изображениям.

выполнен Ковалевым Даниилом Владимировичем
фамилия, имя, отчество обучающегося в творительном падеже

по направлению подготовки 09.03.04 (код) Программная инженерия (наименование направления)

наименование направления

направленности 02 (код) Проектирование программных систем (наименование направленности)

наименование направленности

Обучающийся группы № 4133 (номер) | (подпись, дата) | Д.В. Ковалев (инициалы, фамилия)

Санкт–Петербург 2024

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на прохождение производственной практики по получению профессиональных умений и опыта профессиональной деятельности обучающегося направления подготовки

Фамилия, имя, отчество обучающегося: Ковалев Даниил Владимирович

1. Группа: 4133
2. Тема индивидуального задания: Определение контуров виноградников на спутниковых снимках, создание набора данных и нейросети для обнаружения контуров виноградников по RGB-изображениям.
   Исходные данные:
   https://overpass-turbo.eu/
   https://yandex.ru/maps/
3. Содержание отчетной документации:
   3.1. индивидуальное задание;
   3.2. отчёт, включающий в себя:
   - титульный лист;
   - материалы о выполнении индивидуального задания (содержание определяется кафедрой);
   - выводы по результатам практики;
   - список использованных источников.

   3.3. отзыв руководителя от профильной организации (при прохождении практики в профильной организации).
4. Срок представления отчета на кафедру: « » 2024 г.

Руководитель практики

Ст. преп.		М.Д. Поляк
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

СОГЛАСОВАНО
Руководитель практики от профильной организации

должность	подпись, дата	инициалы, фамилия

Задание принял к исполнению:
Обучающийся

08.07.2024		Д.В. Ковалев
дата	подпись	инициалы, фамилия

1. Цель работы

Создание набора данных и нейросети для обнаружения контуров виноградников по RGB-изображениям.

2. Задачи

1. Подготовить выборку спутниковых снимков виноградников и других типов территорий, вместе с разметкой контуров виноградников. Объем - 100 изображений.
2. Изучить документацию документацию Unet и Python
3. Обучить нейросеть для обнаружения контуров виноградников по RGB-изображениям.

3. Исходные данные.

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 виноградника.

Пример исходных данных:

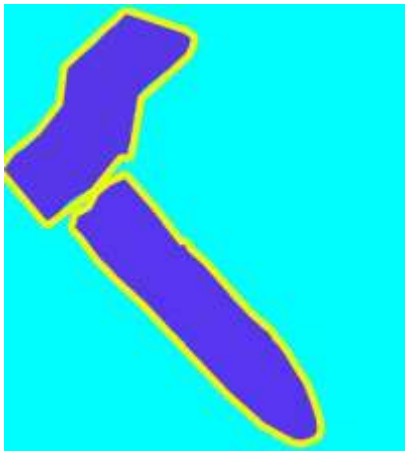


Рисунок 1 – Фото виноградника(**.jpg**)

Рисунок 2 – Маска виноградника(**.png**)

Всего исходные данные содержат 598 изображения, из них:

- 299 фотографий виноградников для обучения нейронной сети и тестеровки
- 299 масок виноградников для обучений нейронной сети.

4. 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

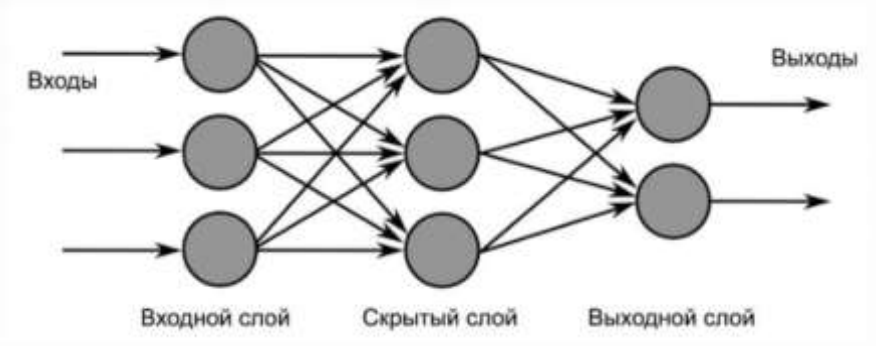


Рисунок 3 – Виды слоев в нейросети

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

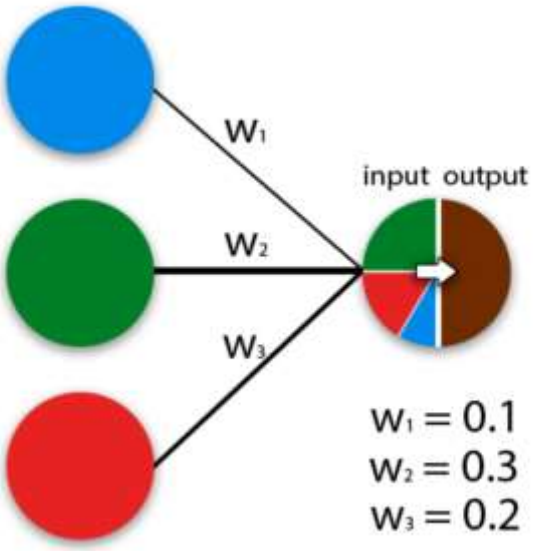


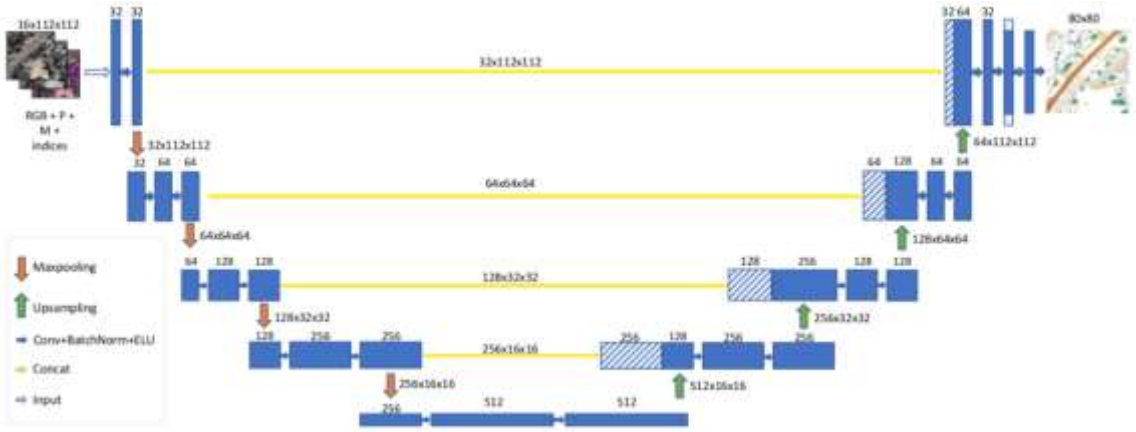
Рисунок 4 – структура синапса

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.

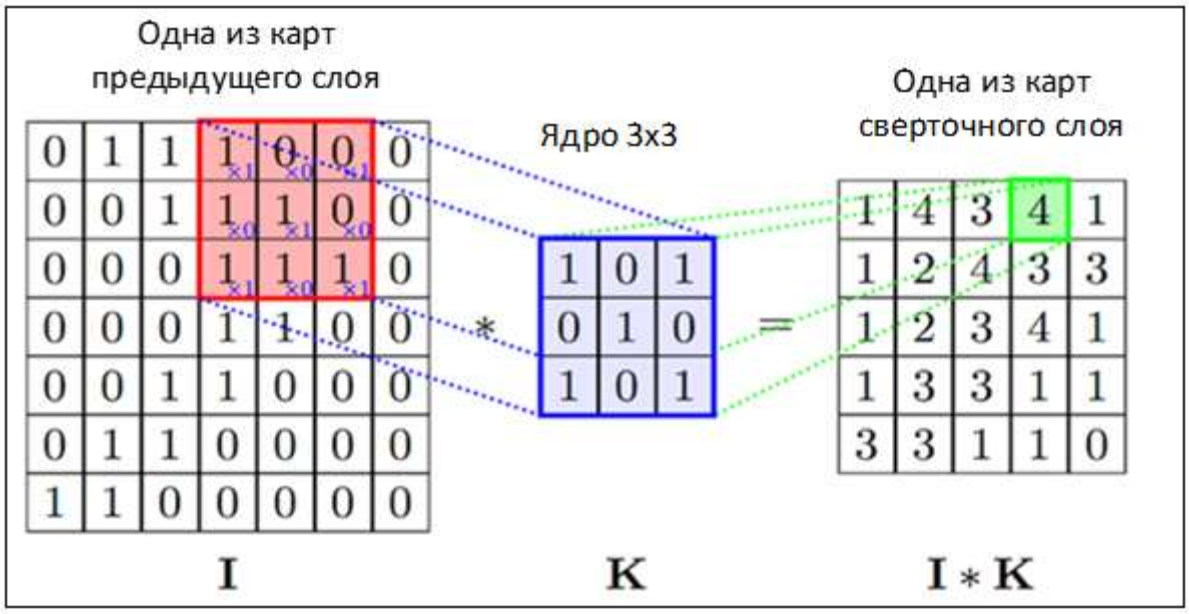


Рисунок 6 – Свертка изображения

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.

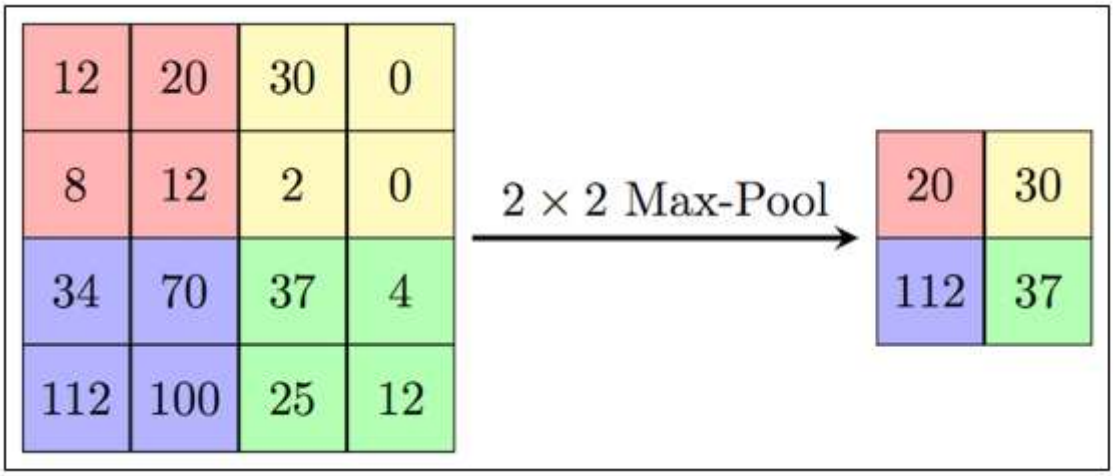


Рисунок 7 – Работа пулингового слоя

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.

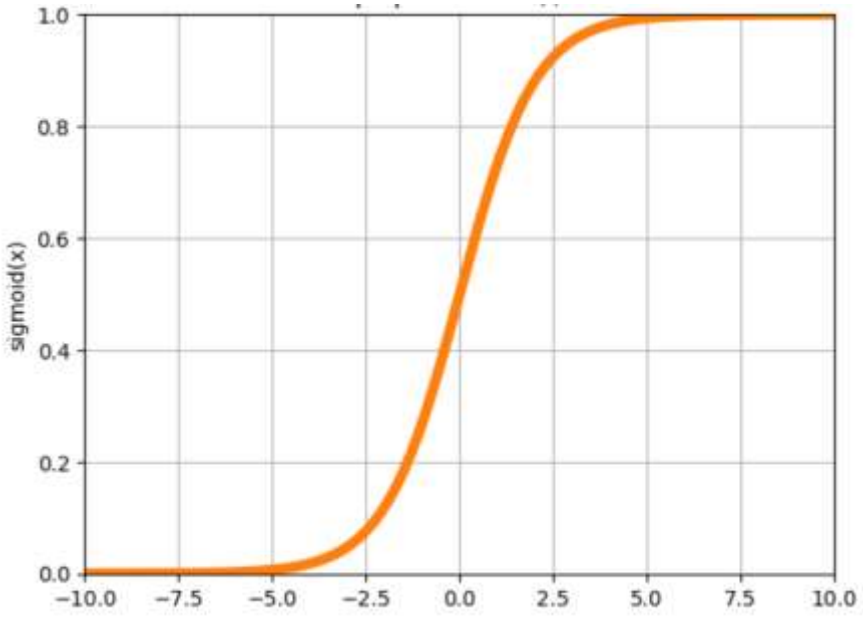


Рисунок 8 – График сигмоиды

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.

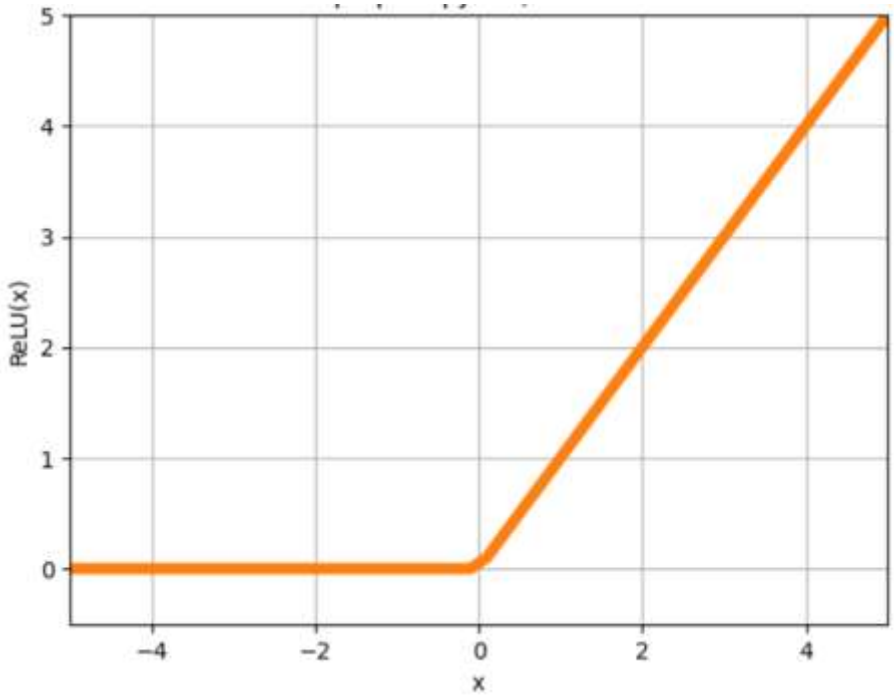


Рисунок 9 – График функции ReLU

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:

$$x_1 = \frac{x - \mu_b}{\sqrt{e + \sigma_b^2}}$$

где x - входные данные, μ_bи σ_b- 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:

$$\omega_{t+1} = \omega_t - \frac{\alpha}{\sqrt{\upsilon_t} + \epsilon} \mathrm{m}_t$$

где ω_t- вектор параметров на шаге t, α - скорость обучения, m_tи υ_t - оценки первого и второго моментов градиента на шаге t , ϵ - 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.

В своей работе, я использую модификацию Adam – Nadam. Nadam (Nesterov-accelerated Adaptive Moment Estimation) использует другую поправку на смещение для вектора первого момента

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:

$$L(y,\ y_1) = \ -y\log(\ y_1) - (1-y)\log(1-\ \ y_1)$$

где y - истинное значение целевой переменной (0 или 1), y_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

$$accuracy = \frac{\text{Количество правильных предсказаний}}{\text{Общее количество предсказаний}}$$

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

5. Практический раздел.

1. Загрузка изображений из файла

def load_img_with_mask(image_path, images_dir: str = 'IMG', masks_dir: str = 'Mask',images_extension: str = 'png', masks_extension: str = 'png')

Принимается на вход путь к изображениям и маскам, их расширения. Возвращает функция кортеж, состоящий из изображений и масок.

1. Изменение размера изображений и масок

def resize_images(images, masks, max_image_size=512)

На вход подаются изображения и маски, а также их максимальный размер. Возвращает функция изображения и маски, приведенные к заданному размеру.

2. Масштабирование значений изображений и масок

def scale_values(images, masks, mask_split_threshold = 128)

На вход подаются изображения и маски, а также пороговое значение. Возвращает измененные масштабированные изображения и маски в виде кортежа.

3. Заполнение изображений и масок до определенного размера

def pad_images(images, masks, pad_mul=16, offset=0)

На вход подаются изображения и маски, а также множитель для определения нового размера. Функция заполняет изображения и масоки до нового размера с использованием функции image.pad_to_bounding_box(), которая добавляет `offset_height` строк “0” сверху, `offset_width` “0” слева, а затем дополняет изображение снизу и справа “0”, пока оно не станет размером `target_height`, `target_width`.

4. Реализация архитектуры U-Net

def get_unet (hidden_activation='relu', initialize = 'he_normal', output_activation = 'sigmoid')

Архитектура U-Net состоит из двух основных частей: энкодера и декодера.

Энкодер состоит из нескольких слоев свертки, каждый из которых уменьшает размерность изображения и извлекает информацию о его признаках. В данном коде энкодер состоит из трех блоков свертки и пулинга. Каждый блок содержит два сверточных слоя с функцией активации hidden_activation и инициализацией весов initializer. После каждого блока выполняется пулинг для уменьшения размерности изображения.

Центральная часть (центр) состоит из двух сверточных слоев, которые помогают сохранить информацию о признаках на более высоком уровне абстракции.

Декодер выполняет обратную операцию энкодера, увеличивая размерность изображения и восстанавливая детали. В данном коде декодер состоит из трех блоков, каждый из которых содержит слой upsampling (увеличение размерности), сверточный слой и операцию конкатенации с соответствующим блоком из энкодера. Затем выполняется два сверточных слоя с функцией активации hidden_activation и инициализацией весов initializer.

В конце декодера применяется слой свертки с одним фильтром и функцией активации output_activation. Этот слой генерирует предсказание сегментации для каждого пикселя изображения.

Возвращается модель U-Net с входом model_input и выходом output.

5. Компиляция модели

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

6. Создание callbacks

Благодаря early_stopping обучение остановится, когда показатель 'val_loss' перестанет уменьшаться. lr_reduce уменьшает скорость обучения в 2-10 раз, когда 'val_loss' перестает улучшаться, что в определенных случаях повышает качество обучения.

7. Обучение модели

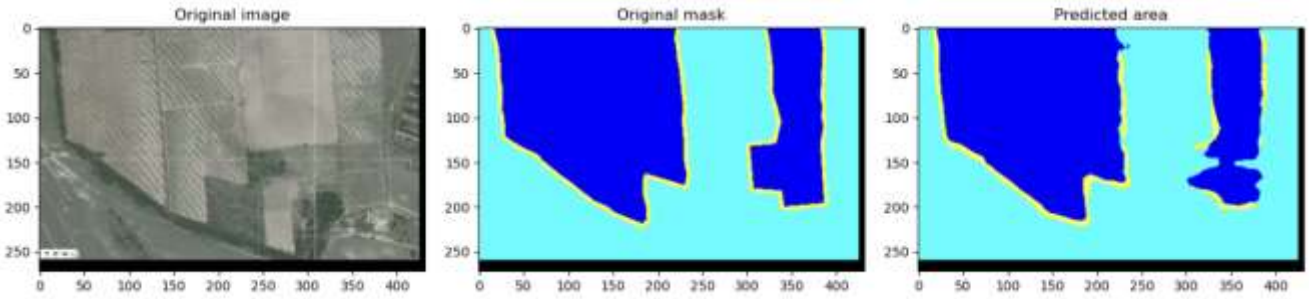
Обучение модели происходит с помощью метода fit библиотеки keras. В качестве параметров передаются: тренировочный датасет, валидационный датасет, количество эпох, callbacks.

8. Тестирование предсказаний нейронной сети

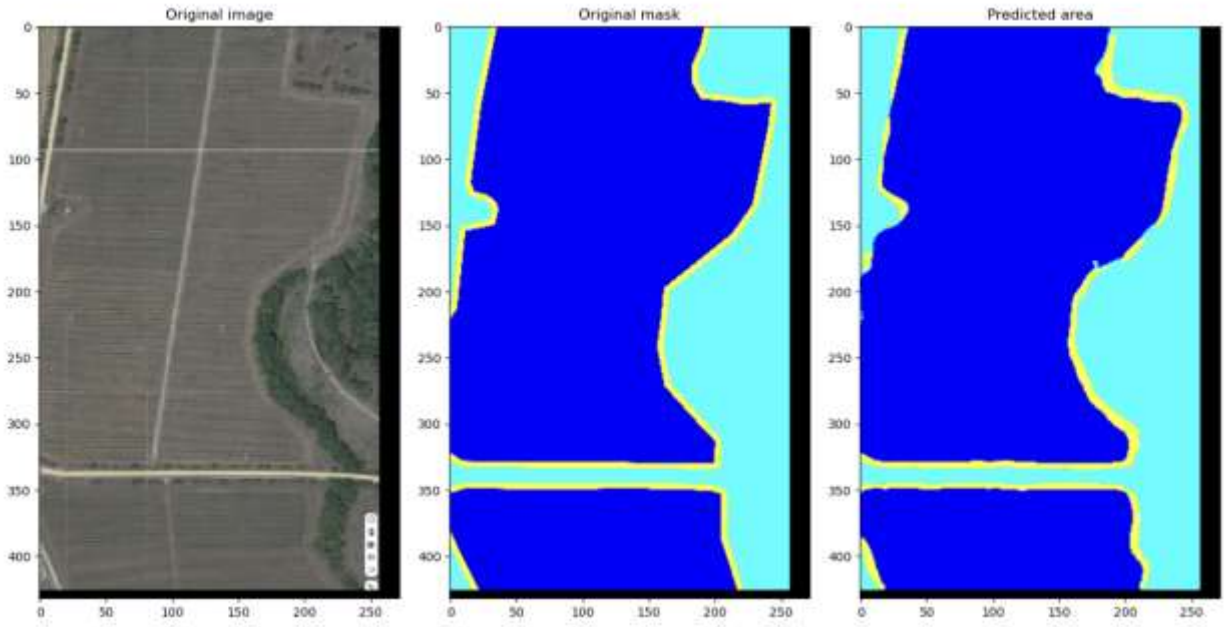
На вход подаются изображения, не участвующие ни в обучении, ни в валидации. Результат работы - графическое окно с n_examples строками и 3 столбцами, в котором в первом столбце отображаются входные изображения, во втором столбце – исходные маски y_true, а в третьем столбце - предсказания модели.

6. Результаты.

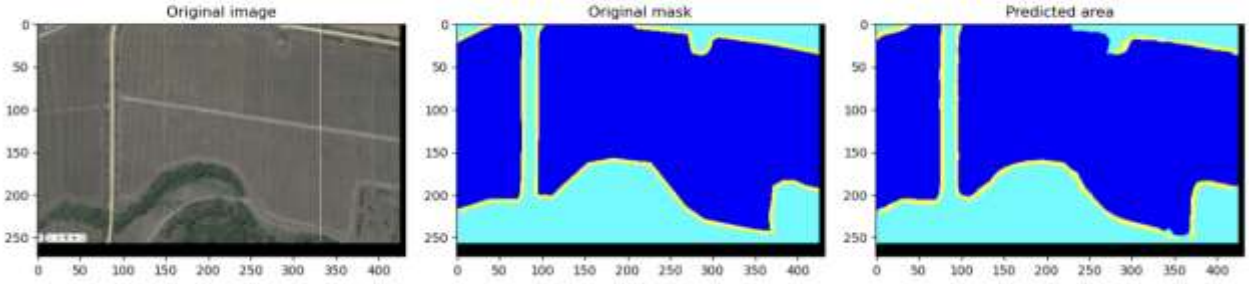
Примеры удачной сегментации:



Оригинальное изображение “ Маска” Выходное изображение

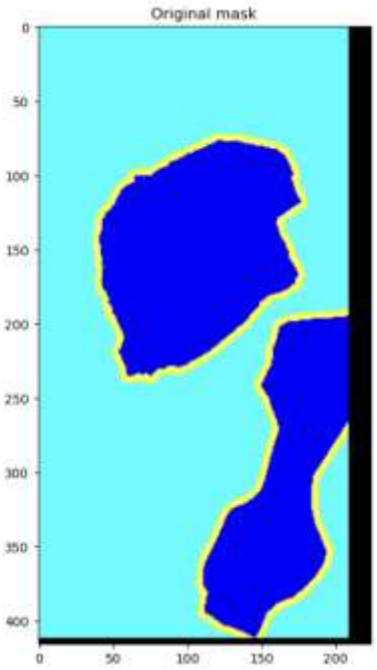


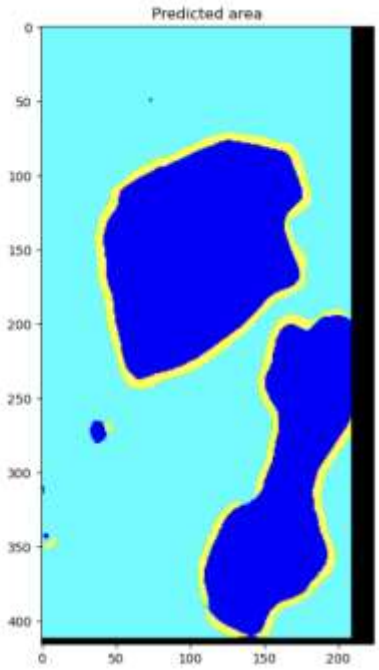
Оригинальное изображение “ Маска” Выходное изображение



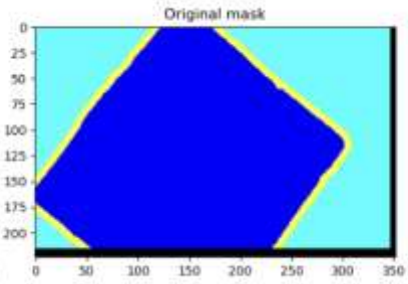
Оригинальное изображение “ Маска” Выходное изображение

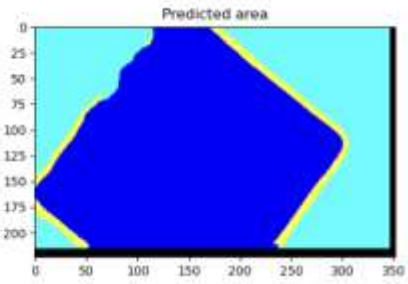
Примеры неудачной сегментации:





Оригинальное изображение “ Маска” Выходное изображение





Оригинальное изображение “ Маска” Выходное изображение

Данные ошибки нейронной сети можно объяснить тем, что обучающих данных было недостаточно до полной аналитики фотографий.

7. Выводы.

В ходе данной работы был успешно составлен набор данных, точно отображающий поверхность виноградных полей и их границы, так же написана и обучена сверточная нейронная сеть. Однако выявлено, что для обучения нейронной сети набора данных из 299 фотографий не достаточно.

8. Список литературы

1. Python [Электронный ресурс]. — Режим доступа: docs.python.org/3.12. Python 3.12.4 Documentation — (Дата обращения: 10.08.2024)

2. Интерактивная сегментация: выделяем кошек, собак и людей / Хабр [Электронный ресурс]. — Режим доступа: https://habr.com/ru/companies/samsung/articles/508342/ — (Дата обращения: 10.08.2024)

3. Распознавание изображений на Python с помощью TensorFlow и Keras [Электронный ресурс]. — Режим доступа: https://evileg.com/ru/post/619/ — (Дата обращения: 10.08.2024)

4. Обнаружение объекта на изображении методом цветовой сегментации (Python) — Режим доступа: https://waksoft.susu.ru/2019/05/26/obnaruzhenie-obekta-na-izobrazhenii-opirajas-na-cvetovuju-segmentacii-python/ — (Дата обращения: 10.08.2024)

5. Структура и принцип работы полносвязных нейронных сетей [Электронный ресурс]. — Режим доступа: https://proproprogs.ru/neural_network/struktura-i-princip-raboty-polnosvyaznyh-neyronnyh-setey — (Дата обращения: 10.08.2024)

ПРИЛОЖЕНИЕ А

Листинг программы(Jupyter Notebook)

```
from functools import partial
import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras

images_dir = 'IMG'
masks_dir = 'Mask'

dirname, _, filenames = next(os.walk(images_dir))

@tf.function
def load_img_with_mask(image_path, images_dir: str = 'IMG', masks_dir: str =
'Mask',images_extension: str = 'png', masks_extension: str = 'png') -> dict:
    image = tf.io.read_file(image_path)
    image = tf.image.decode_png(image, channels=3)

    mask_filename = tf.strings.regex_replace(image_path, images_dir, masks_dir)
    mask_filename = tf.strings.regex_replace(mask_filename, images_extension,
masks_extension)

    mask = tf.io.read_file(mask_filename)
    mask = tf.image.decode_image(mask, channels=3, expand_animations = False)
    return (image, mask)
```

```
def count_images_in_dir(images_dir: str) -> int:
    _, _, filenames = next(os.walk(images_dir))
    return len(filenames)

# Пример использования
images_dir = 'IMG'
masks_dir = 'Mask'

num_images = count_images_in_dir(images_dir)
print(f"Количество фотографий в директории {images_dir}: {num_images}")
num_images = count_images_in_dir(masks_dir)
print(f"Количество фотографий в директории {masks_dir}: {num_images}")

%matplotlib inline
n_examples = 3
examples = [load_img_with_mask(os.path.join(images_dir, filenames[i])) for i in
range(n_examples)]

fig, axs = plt.subplots(n_examples, 2, figsize=(14, n_examples*7),
constrained_layout=True)
for ax, (image, mask) in zip(axs, examples):
    ax[0].imshow(image)
    ax[1].imshow(mask)

fig, axs = plt.subplots(n_examples, 2, figsize=(14, n_examples*7),
constrained_layout=True)
for ax, (image, mask) in zip(axs, examples):
    ax[0].imshow(image)
    ax[1].imshow(mask)

@tf.function
def resize_images(images, masks, max_image_size=512):
    shape = tf.shape(images)
    scale = (tf.reduce_max(shape) // max_image_size) + 1
    target_height, target_width = shape[-3] // scale, shape[-2] // scale
```

```
    images = tf.cast(images, tf.float32)
    masks = tf.cast(masks, tf.float32)
    if scale != 1:
        images = tf.image.resize(images, (target_height, target_width),
method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
        masks = tf.image.resize(masks, (target_height, target_width),
method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
    return (images, masks)

@tf.function
def scale_values(images, masks, mask_split_threshold = 128):
    images = tf.math.divide(images, 255)
    masks = tf.where(masks > mask_split_threshold, 1, 0)
    return (images, masks)

@tf.function
def pad_images(images, masks, pad_mul=16, offset=0):
    shape = tf.shape(images)
    height, width = shape[-3], shape[-2]

    # Вычисляем целевые размеры
    target_height = tf.math.ceil(tf.cast(height, tf.float32) / pad_mul) * pad_mul
    target_width = tf.math.ceil(tf.cast(width, tf.float32) / pad_mul) * pad_mul

    # Убеждаемся, что целевые размеры не меньше исходных
    target_height = tf.maximum(target_height, tf.cast(height, tf.float32))
    target_width = tf.maximum(target_width, tf.cast(width, tf.float32))

    # Преобразуем обратно в int32
    target_height = tf.cast(target_height, tf.int32)
    target_width = tf.cast(target_width, tf.int32)

    # Вычисляем отступы
    pad_height = target_height - height
    pad_width = target_width - width
```

```
    # Применяем падding
    images = tf.image.pad_to_bounding_box(images, offset, offset, target_height, target_width)
    masks = tf.image.pad_to_bounding_box(masks, offset, offset, target_height, target_width)

    return images, masks

batch_size = 8
test_set_size = 5
validation_set_size = 100

dataset = tf.data.Dataset.list_files(images_dir + '/*.png', seed=42)

test_dataset = dataset.take(test_set_size)
dataset = dataset.skip(test_set_size)
test_dataset = test_dataset.map(load_img_with_mask)
test_dataset = test_dataset.map(scale_values)
test_dataset = test_dataset.shuffle(20)
test_dataset = test_dataset.map(lambda img, mask: resize_images(img, mask, max_image_size=512))
test_dataset = test_dataset.map(pad_images)
test_dataset = test_dataset.batch(1).prefetch(5)


validation_dataset = dataset.take(validation_set_size)
train_dataset = dataset.skip(validation_set_size)
validation_dataset = validation_dataset.map(load_img_with_mask)
validation_dataset = validation_dataset.map(scale_values)
validation_dataset = validation_dataset.shuffle(20)
validation_dataset = validation_dataset.map(resize_images)
validation_dataset = validation_dataset.map(pad_images)
validation_dataset = validation_dataset.batch(1).prefetch(5)
```

```
train_dataset = train_dataset.map(load_img_with_mask)
train_dataset = train_dataset.map(scale_values)
train_dataset = train_dataset.shuffle(20)
train_dataset = train_dataset.map(resize_images)
train_dataset = train_dataset.map(pad_images)
train_dataset = train_dataset.batch(1).prefetch(5)

# Подсчет количества изображений в тренировочном наборе # Размер батча
num_batches = tf.data.experimental.cardinality(train_dataset).numpy()
total_images = num_batches * batch_size
print(f"Общее количество изображений в тренировочном наборе:
{total_images}")

def get_unet(hidden_activation='relu', initializer='he_normal',
output_activation='sigmoid'):
    PartialConv = partial(keras.layers.Conv2D,
     activation=hidden_activation,
     kernel_initializer=initializer,
     padding='same')

    # pooling
    model_input = keras.layers.Input(shape=(None, None, 3))
    enc_cov_1 = PartialConv(32, 3)(model_input)
    enc_cov_1 = PartialConv(32, 3)(enc_cov_1)
    enc_pool_1 = keras.layers.MaxPooling2D(pool_size=(2, 2))(enc_cov_1)

    enc_cov_2 = PartialConv(64, 3)(enc_pool_1)
    enc_cov_2 = PartialConv(64, 3)(enc_cov_2)
    enc_pool_2 = keras.layers.MaxPooling2D(pool_size=(2, 2))(enc_cov_2)

    enc_cov_3 = PartialConv(128, 3)(enc_pool_2)
    enc_cov_3 = PartialConv(128, 3)(enc_cov_3)
    enc_pool_3 = keras.layers.MaxPooling2D(pool_size=(2, 2))(enc_cov_3)

    # Center
```

```
    center_cov = PartialConv(256, 3)(enc_pool_3)
    center_cov = PartialConv(256, 3)(center_cov)

    # upsampling
    upsampling1 = keras.layers.UpSampling2D(size=(2, 2))(center_cov)
    dec_up_conv_1 = PartialConv(128, 2)(upsampling1)
    dec_merged_1 = tf.keras.layers.Concatenate(axis=3)([enc_cov_3,
dec_up_conv_1])
    dec_conv_1 = PartialConv(128, 3)(dec_merged_1)
    dec_conv_1 = PartialConv(128, 3)(dec_conv_1)

    upsampling2 = keras.layers.UpSampling2D(size=(2, 2))(dec_conv_1)
    dec_up_conv_2 = PartialConv(64, 2)(upsampling2)
    dec_merged_2 = tf.keras.layers.Concatenate(axis=3)([enc_cov_2,
dec_up_conv_2])
    dec_conv_2 = PartialConv(64, 3)(dec_merged_2)
    dec_conv_2 = PartialConv(64, 3)(dec_conv_2)

    upsampling3 = keras.layers.UpSampling2D(size=(2, 2))(dec_conv_2)
    dec_up_conv_3 = PartialConv(32, 2)(upsampling3)
    dec_merged_3 = tf.keras.layers.Concatenate(axis=3)([enc_cov_1,
dec_up_conv_3])
    dec_conv_3 = PartialConv(32, 3)(dec_merged_3)
    dec_conv_3 =  PartialConv(32, 3)(dec_conv_3)

    output = keras.layers.Conv2D(3, 1, activation=output_activation)(dec_conv_3)

    return tf.keras.Model(inputs=model_input, outputs=output)

def check_dataset_shapes(dataset):
    shapes = set()
    for images, masks in dataset.take(8):  # Проверяем первые 10 батчей
        shapes.add((images.shape[1], images.shape[2]))
    print(f"Unique shapes in dataset: {shapes}")
```

```
# Проверяем датасеты
print("Train dataset:")
check_dataset_shapes(train_dataset)
print("Validation dataset:")
check_dataset_shapes(validation_dataset)
print("Test dataset:")
check_dataset_shapes(test_dataset)

model = get_unet()

optimizer = tf.keras.optimizers.Nadam()
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics =
['accuracy'])

SPE= len(images_dir)//batch_size

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
lr_reduce = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.3,
patience=3, verbose=1)

epochs = 100
history = model.fit(train_dataset, validation_data=validation_dataset,
steps_per_epoch=SPE, epochs=epochs, callbacks=[early_stopping, lr_reduce])

plt.plot(history.history['learning_rate'])
plt.title('Learning Rate')
plt.xlabel('Epoch')
plt.ylabel('Learning Rate')
plt.show()

n_examples = 3

fig, axs = plt.subplots(n_examples, 3, figsize=(14, n_examples*7),
constrained_layout=True)
for ax, ele in zip(axs, test_dataset.take(n_examples)):
```

```
image, y_true = ele
prediction = model.predict(image)[0]
prediction = tf.where(prediction > 0.5, 255, 0)
ax[0].set_title('Original image')
ax[0].imshow(image[0])
ax[1].set_title('Original mask')
ax[1].imshow(y_true[0])
ax[2].set_title('Predicted area')
ax[2].imshow(prediction)
```