МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Пермский государственный национальный
исследовательский университет»

Физико-математический институт

Научно-исследовательская работа

«Сегментация видеопотока: Изменяющийся и прячущийся объект. Решение задачи выявления лабораторного животного в сложной экспериментальной среде, используя нейронные сети»

Работу выполнил студент
группы ПМИ 1 курса
магистратуры физико-
математического факультета
Канзепаров Руслан Маратович
«___» _____2025 г.
_____
Научный руководитель:
Директор института, доктор
физико-математических наук,
доцент Марина Александровна
Барулина
«___» _____2025 г.
_____

Пермь 2025

Содержание

Введение

Данная работа была выдана лабораторией экспериментальной фармакологии пермского государственного научно-исследовательского университета.

В данном проекте методами машинного обучения решается задача сегментации тела лабораторного животного (черной мыши) в клетке установки «МультиНейро-СДА», имитирующей домашнюю клетку. Решение этой задачи возможно на основе видеозаписей экспериментов, и ряда преобразований этих видеозаписей, в том числе:

1. Препроцессинга, связанного с подготовкой видеозаписи к работе – обрезки, стандартизация, изменения размера, коррекции геометрических искажений.

2. Подготовка обучающей и тестовой выборки, путем ручной сегментации тела животного.

3. Дальнейшего их преобразования для использования в процессе обучения нейрсети.

Далее, полученные изображения, содержащие оригинальные изображения и сегментационные маски тел животных, подаются на вход сегментационной сети – для обучения и тестирования нейросети, или подается «сырой» видеопоток для его обработки, в зависимости от фазы выполнения проекта.

Задача сегментации тела животного осложняется множеством факторов, например, таких как неоднородность изображений, малая контрастность или изменяемость фона, периодические возникновения отражений животного в стенках установки, большая степень изменяемости тела самого животного, его быстрые перемещения, а также сегментация может осложняться самим качеством видеоматериалов. Решение комплекса этих проблем в ходе обучения сегментационной модели поможет получить хорошие показатели сегментации тела животного и успешно создать инструмент для решения задач следующего уровня.

На уровне продукта задача проекта заключается в построении, на основе приложенных материалов, программного комплекса, состоящего как минимум, из следующих компонентов:

1. Программы для обучения сегментационной модели,

2. Самой модели, которая была бы способна сегментировать тело животного на видеоматериалах с записью эксперимента в установке «МультиНейро-СДА»

3. Программы, способной использовать данную модель для создания видео массива сегментированного тела животного, определённого на видеозаписях.

Решение задачи этого проекта необходимо для перехода к количественной оценки встречаемости и оценки характеристик последовательностей поведенческих паттернов, что позволит интерпретировать состояние животного в эксперименте. Создание подобной системы избавит экспертов от необходимости тратить многие человеко-часы на просмотр и дешифровку больших массивов видеоматериалов.

Используемый стек технологий

1. Языки программирования Python и/или C++;

2. Программную библиотеку для машинного обучения «TensorFlow» с, или же без, высокоуровнего API «Keras»;

3. Библиотеку для работы с компьютерным зрением «OpenCV»;

4. Программное обеспечение для проведения параллельных вычислений «CUDA»;

5. Библиотеку с поддержкой GPU примитивов для глубоких нейронных сетей «CuDNN»;

6. Среду разработки «JupyterLab».

1. Теоретическая часть

1.1. Семантическая сегментация.

Семантическая сегментация - это метод компьютерного зрения, который позволяет не только обнаруживать объекты на изображениях, но и определяет их точное пространственное местоположение, классифицируя каждый пиксель.

Типы сегментации:

Бинарная сегментация – алгоритм, который рекурсивно делит последовательность данных на части, выявляя точки изменений, где статистические свойства данных существенно изменяются. Используется для выделения одного объекта.

Мультиклассовая сегментация — это расширение бинарной сегментации на более сложные сценарии с многочисленными классами объектов. каждый элемент данных (например, пиксель изображения) классифицируется в один из нескольких взаимоисключающих классов. В отличие от бинарной сегментации, где всего два класса (например, объект и фон), в мультиклассовой сегментации классов несколько, и модель должна точно определить к какому из них принадлежит каждый пиксель или элемент.

Инстанс-сегментация (экземплярная сегментация): кроме классификации пикселей, отдельно выделяются разные объекты одного класса.

Паноптическая сегментация: объединяет семантическую и инстанс-сегментацию, классифицируя все пиксели и отдельно выделяя экземпляры объектов. Это позволяет получить полное представление о сцене.

Пороговая сегментация: использует порог яркости или другой характеристики для разделения изображения на сегменты, часто применяется для бинарных изображений.

Для дальнейшей работы, был выбран тип бинарной сегментации.

1.2. Архитектура нейронных сетей для сегментации.

Архитектуры нейросетей для сегментации изображений представляют собой специальные модели глубокого обучения, предназначенные для

точного выделения объектов на изображениях путем классификации каждого пикселя. Среди самых известных и эффективных архитектур выделяются:

- U-Net: Это одна из наиболее популярных архитектур для задач сегментации, особенно в биомедицинских приложениях. U-Net состоит из двух частей — сжимающего (энкодера) пути для выделения признаков и расширяющего (декодера) пути для восстановления пространственного разрешения и точной локализации сегментов. Особенность U-Net — соединения пропуска (skip connections), которые передают информацию с уровней энкодера в декодер, улучшая качество сегментации. Эта архитектура хорошо работает на небольшом количестве данных и быстро выполняет сегментацию [1].

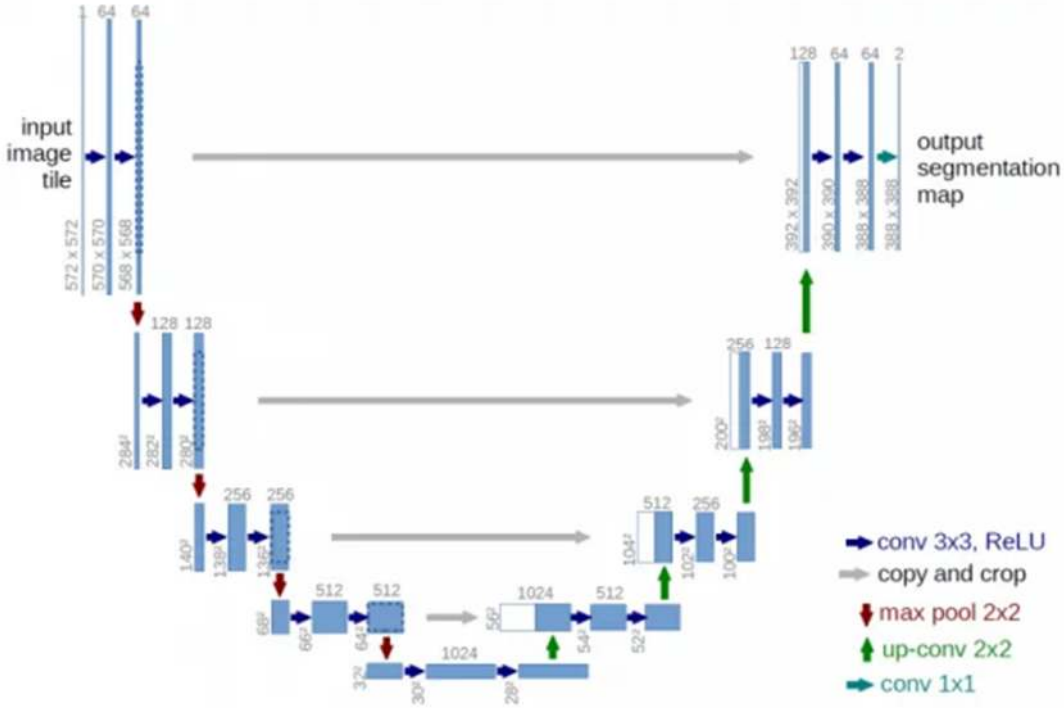

Рисунок 1 – архитектура U-Net

- FCN (Fully Convolutional Network): Это полностью сверточная сеть, заменяющая полносвязные слои свертками, что позволяет обрабатывать изображения любого размера и создавать карты сегментации с пространственным разрешением. FCN использует слои деконволюции для восстановления размеров изображения и skip connections для объединения признаков разных уровней [2].

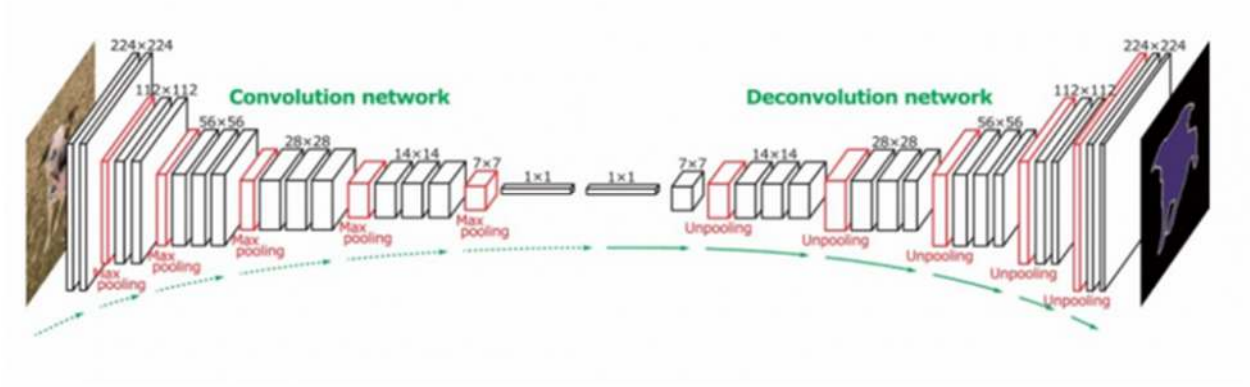

Рисунок 2 – архитектура FCN

- SegNet: Архитектура, разработанная для эффективной семантической сегментации с низкими вычислительными затратами. Особенность SegNet — восстановление пространственного разрешения с помощью индексов пулинга из энкодера, что улучшает качество сегментации по сравнению с другими методами [3].

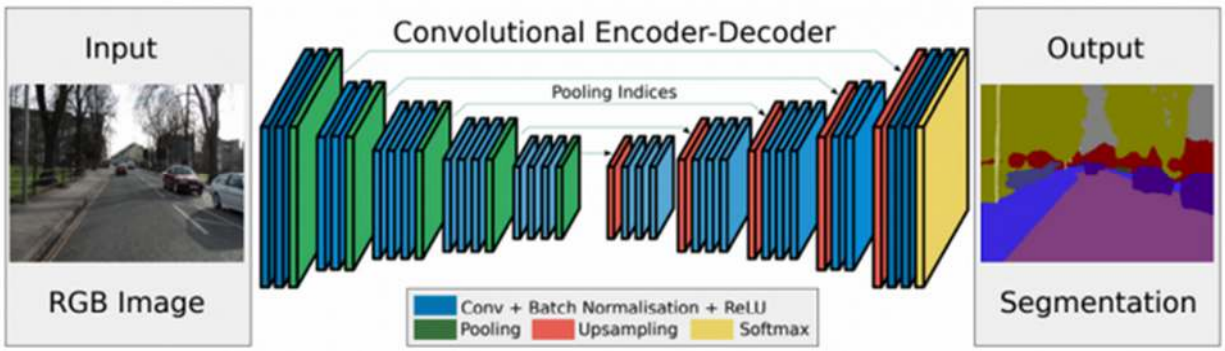

Рисунок 3 – архитектура SegNet

- PSPNet (Pyramid Scene Parsing Network): Архитектура, которая вводит пирамидальный парсинг сцены для улучшения контекста и понимания глобальных сцен изображения, что повышает точность семантической сегментации [4].

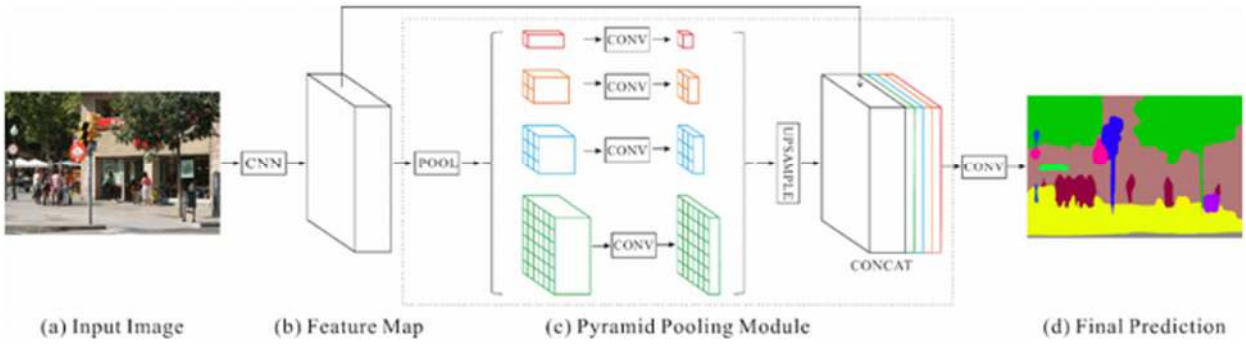

Рисунок 4 – архитектура PSPNet

- DeepLab: Архитектура нейронных сетей для семантической сегментации изображений, разработанное командой Google Research, которое значительно улучшило качество сегментации объектов на уровне пикселей [5].

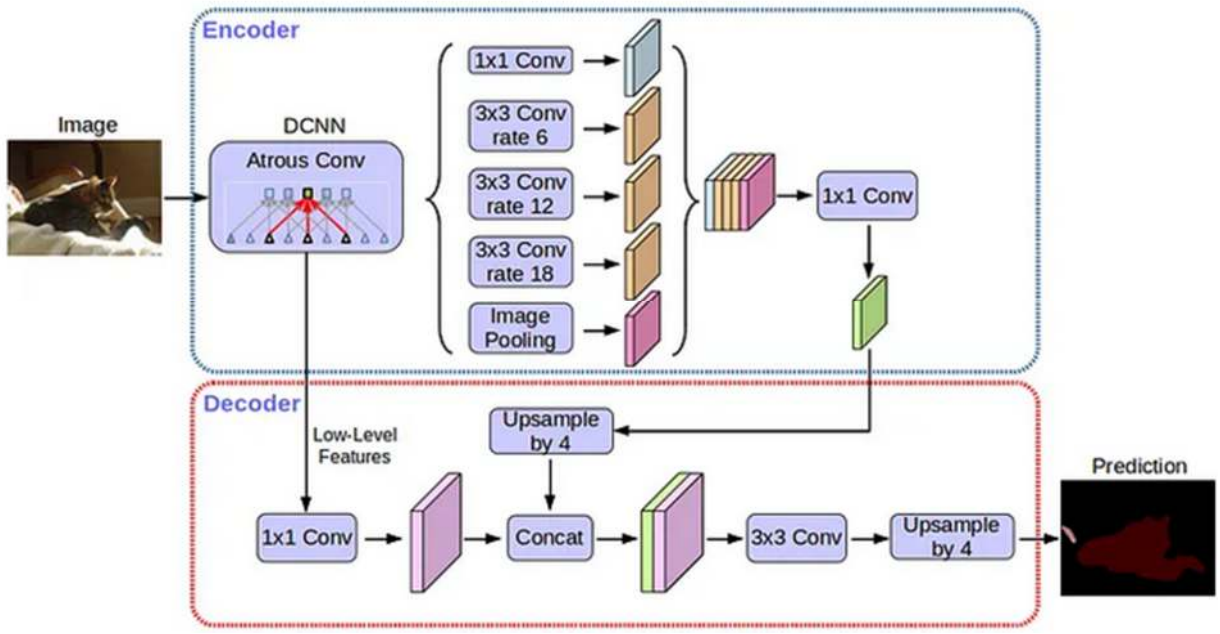

Рисунок 5 – архитектура DeepLab

Для данной задачи была выбрана модель SegNet. Ее преимущества по сравнению с другими моделями:

1. Эффективное восстановление разрешения: SegNet использует индексы пула (pooling indices), которые сохраняются на этапе энкодера и применяются на декодере для точного восстановления положения значимых

признаков. Это уменьшает потери информации при увеличении разрешения и повышает качество сегментации.

2. Низкие вычислительные затраты и экономия памяти.

3. Простота и модульность архитектуры: строится на базе известного энкодера VGG16, что облегчает понимание и адаптацию сети под разные задачи без сложных модификаций.

Сравнение архитектур можно наблюдать в таблице 1.

Архитектура модели	U-Net	FCN	SegNet	PSPNet	DeepLab
Производительность	2	1	1	2	2
Вычислительные затраты	1	1	2	0	0
Обработка изображений любого размера	1	2	2	2	2
Маленькая обучающаяся выборка	2	1	2	0	0
Точность	2	1	1	2	2
Реализация	2	1	2	1	1

Таблица 1 – сравнение архитектур.

Оценки: 0 – плохо, 1 – удовлетворительно, 2 – отлично

Из вышеперечисленных результатов в таблице, можно сделать вывод, что самыми приоритетными для моей работы являются U-Net и SegNet. Возможно U-Net был бы предпочтительнее для данного проекта, но выбрал SegNet из-за простоты и меньших вычислительных затрат.

1.3. Библиотека TenserFlow

TensorFlow — это открытая библиотека для машинного обучения и глубокого обучения, созданная компанией Google в 2015 году. Она служит для построения, обучения и развёртывания моделей машинного обучения, в том числе нейронных сетей. TensorFlow помогает обрабатывать и анализировать данные, обучать алгоритмы, которые могут распознавать изображения, анализировать текст, прогнозировать события и выполнять многие другие задачи искусственного интеллекта [6].

Основная идея TensorFlow заключается в использовании тензоров (многомерных массивов данных) и представлении вычислений в виде графа,

где вершины — это операции, а ребра — данные, которые передаются между ними. Такой подход позволяет эффективно распределять вычисления на процессоры CPU, графические процессоры GPU и специализированные тензорные процессоры TPU. TensorFlow широко применяется для создания и тренировки нейронных сетей разной сложности — от простых моделей до сложных глубоких сетей для распознавания речи, обработки естественного языка, компьютерного зрения и других направлений.

В проекте была использована версия tenserflow-gpu-2.10.0. Именно это последняя версия tenserflow, которая поддерживает видеокарты (GPU) на Windows.

1.4. Библиотека CUDA и CuDNN

CUDA (Compute Unified Device Architecture) — это платформа параллельных вычислений, которая позволяет программам использовать возможности GPU компании NVIDIA для обработки данных общего назначения. Она реализует технику GPGPU (General-Purpose computing on Graphics Processing Units), позволяющую делить задачи на множество мелких потоков и выполнять их параллельно [7].

Основные функции CUDA:

- Обеспечивает программное управление потоками и памятью GPU.
- Позволяет ускорять научные, инженерные и аналитические задачи, увеличивая производительность за счет параллелизма.
- Поддерживает различные языки программирования, такие как C++, Python и другие.
- Идеальна для обработки больших массивов данных, обучения нейросетей и симуляций.

CuDNN (CUDA Deep Neural Network), это дополнительная библиотека от NVIDIA, предназначенная для оптимизации и ускорения выполнения операций глубокого обучения на GPU. Она использует возможности CUDA и специально разработана для работы с нейросетями, значительно сокращая время обучения и эффективности моделей [8].

Задачи cuDNN:

- Оптимизация операций сверточных нейросетей, таких как свертки, нормализация и активации.

- Использование при создании моделей глубокого обучения — от обработки изображений до естественного языка.
- Обеспечивает максимально возможную производительность на совместимых GPU за счет низкоуровневой оптимизации.

Для проекта была выбрана версия CUDA 12.3 и cuDNN 9.0.0 для стабильной работы с tenserflow-gpu-2.10.0.

1.5. Библиотека OpenCV.

OpenCV (Open Source Computer Vision Library) — это библиотека с открытым исходным кодом, предназначенная для решения задач компьютерного зрения, обработки изображений и машинного обучения. Она содержит более 2500 алгоритмов, которые позволяют обрабатывать и анализировать изображения и видео, распознавать объекты, лица, жесты, а также выполнять задачи калибровки камер и многое другое [9].

OpenCV нужна для того, чтобы научить компьютер «видеть» и понимать визуальную информацию. Библиотека работает на разных платформах и поддерживает множество языков программирования, включая C++, Python и Java, что делает её удобной и популярной для разработки программ с функциями компьютерного зрения.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Архитектура и разработка

Для разработки программы необходимо следовать некоторым требованиям:

- Язык программирования Python.
- Библиотека для машинного обучения «TenserFlow».
- Библиотека для работы с компьютерным зрением «OpenCV».
- ПО для проведения параллельных вычислений «CUDA».
- Библиотека с GPU для глубоких нейронных сетей «CuDNN».
- Среда разработки «JupyterLab».

2.2 Класс, функции и тренировочная модель.

Конфигурация

```
# ========== КОНФИГУРАЦИЯ ==========
GLOBAL_SHAPE_SIZE = 256
MODEL_NAME = "SegNet"
FILE_EXT = ".keras"
DATABASE_PATH = 'mouse_segmentation/'
SAVED_IMAGES_PATH = os.path.join(DATABASE_PATH, 'save_images')
LAST_VIDEO_IMAGES = 'videoImg'

os.makedirs(SAVED_IMAGES_PATH, exist_ok=True)
```

Настройка работы GPU и CUDA

```
# ========== НАСТРОЙКА GPU И CUDA ==========
def setup_gpu():
    """Настройка GPU для использования CUDA"""
    # Проверяем доступные физические устройства
    gpus = tf.config.list_physical_devices('GPU')

    if gpus:
        try:
            # Разрешаем рост памяти GPU
            for gpu in gpus:
                tf.config.experimental.set_memory_growth(gpu, True)

            # Логическая устройство должно быть создано после настройки GPU
            logical_gpus = tf.config.list_logical_devices('GPU')
            print(f"{len(gpus)} Physical GPU(s), {len(logical_gpus)} Logical GPU(s)")
```

```
            print(f"Using GPU: {gpus[0].name}")

            # Устанавливаем стратегию распределения
            strategy = tf.distribute.MirroredStrategy()
            print(f'Number of devices: {strategy.num_replicas_in_sync}')
            return strategy

        except RuntimeError as e:
            # Ошибка настройки GPU
            print(f"GPU setup error: {e}")
            print("Falling back to CPU")
            return None
    else:
        print("No GPU devices found. Using CPU")
        return None
```

Класс для реализации обратного макс пулинга MaxUnpooling2D. В классе реализовано простое масштабирование и работа с тензорами.

```
class MaxUnpooling2D(layers.Layer):
    def __init__(self, pool_size=(2, 2), **kwargs):
        super(MaxUnpooling2D, self).__init__(**kwargs)
        self.pool_size = pool_size

    def call(self, inputs, output_shape):
        return tf.image.resize(inputs, output_shape[1:3], method='nearest')

    def get_config(self):
        config = super(MaxUnpooling2D, self).get_config()
        config.update({"pool_size": self.pool_size})
        return config

    @classmethod
    def from_config(cls, config):
        return cls(**config)
```

Функция conv_block реализует сверточный блок

```
def conv_block(inputs, filters, pool=True):
    x = Conv2D(filters, 3, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = Activation('ReLU')(x)
    x = Conv2D(filters, 3, padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('ReLU')(x)
    if pool:
```

```
        p = MaxPool2D((2, 2))(x)
        return x, p
    return x
```

Функция build_segnet реализует архитектуру SegNet. Имеет симметричную архитектуру – encoder и decoder, использует обратный пулинг для увеличения изображения и бинарная сегментация.

```
def build_segnet(input_shape):
    tf.keras.backend.clear_session()
    inputs = Input(input_shape)

    # Encoder
    x1, p1 = conv_block(inputs, 64)
    x2, p2 = conv_block(p1, 128)
    x3, p3 = conv_block(p2, 256)
    x4, p4 = conv_block(p3, 512)

    # Bridge
    b = conv_block(p4, 512, pool=False)

    # Decoder
    d1 = MaxUnpooling2D((2, 2))(b, tf.shape(x4))
    d1 = conv_block(d1, 512, pool=False)
    d2 = MaxUnpooling2D((2, 2))(d1, tf.shape(x3))
    d2 = conv_block(d2, 256, pool=False)
    d3 = MaxUnpooling2D((2, 2))(d2, tf.shape(x2))
    d3 = conv_block(d3, 128, pool=False)
    d4 = MaxUnpooling2D((2, 2))(d3, tf.shape(x1))
    d4 = conv_block(d4, 64, pool=False)

    # Output
    outputs = Conv2D(1, 1, padding='same', activation='sigmoid')(d4)
    return Model(inputs, outputs, name=MODEL_NAME)
```

Функция load_data загружает данные и подготавливает их для обучения модели сегментации. В функции реализована сортировка файлов, выравнивание по батчу, перемещение и фиксированный random_state.

Функция convert_to_gray_img загружает изображение и преобразует его в оттенок серого.

Функция mod_img выполняет препроцессинг изображения для подготовки в тренировочную модель.

Функция read_image загружает и подготавливает цветное изображение для обработки.

Функция read_mask загружает и подготавливает бинарную маску для задач сегментации.

Функция preprocess_model_images создает конвейер предобработки данных для TensorFlow, преобразуя пути к файлам в готовые тензоры для модели.

```
def preprocess_model_images(image_path, mask_path):
    def f(image_path, mask_path):
        image_path = image_path.decode()
        mask_path = mask_path.decode()
        x = read_image(image_path)
        y = read_mask(mask_path)
        return x, y

    image, mask = tf.numpy_function(f, [image_path, mask_path], [tf.float32,
tf.float32])
    image.set_shape([GLOBAL_SHAPE_SIZE, GLOBAL_SHAPE_SIZE, 3])
    mask.set_shape([GLOBAL_SHAPE_SIZE, GLOBAL_SHAPE_SIZE, 1])
    return image, mask
```

Функция train_model() организует полный процесс обучения модели сегментации SegNet. В ней задается размер батча, эпохи, скорость обучения, обучающий и тренировочный датасет. Создается архитектура SegNet и производит процесс обучения.

```
def train_model():
    batch_size = 16
    epochs = 20
    lr = 1e-4
    model_path = os.path.join(DATABASE_PATH, MODEL_NAME + FILE_EXT)
    csv_path = os.path.join(DATABASE_PATH, MODEL_NAME + "_data.csv")

    (train_x, train_y), (test_x, test_y) = load_data(batch_size)
    train_dataset = tf_dataset(train_x, train_y, batch=batch_size)  # Исправлено здесь
    valid_dataset = tf_dataset(test_x, test_y, batch=batch_size)    # Исправлено здесь

    input_shape = (GLOBAL_SHAPE_SIZE, GLOBAL_SHAPE_SIZE, 3)
    model = build_segnet(input_shape)
```

```
    model.compile(
        loss="binary_crossentropy",
        optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
        metrics=[
            tf.keras.metrics.IoU(num_classes=2, target_class_ids=[1]),
            tf.keras.metrics.Recall(),
            tf.keras.metrics.Precision(),
            'accuracy'
        ]
    )

    callbacks = [
        ModelCheckpoint(model_path, monitor="val_loss", save_best_only=True,
verbose=1),
        ReduceLROnPlateau(monitor="val_loss", patience=5, factor=0.1, verbose=1),
        CSVLogger(csv_path),
        EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True),
    ]

    train_steps = len(train_x) // batch_size
    valid_steps = len(test_x) // batch_size

    return model, epochs, train_steps, valid_steps, callbacks, train_dataset, valid_dataset
```

Функция fit_model является оберткой для процесса обучения модели, которая делегирует вызов стандартному методу fit() Keras.

```
def fit_model(model, epochs, train_steps, valid_steps, callbacks, train_dataset,
valid_dataset):
    model.fit(
        train_dataset,
        validation_data=valid_dataset,
        epochs=epochs,
        steps_per_epoch=train_steps,
        validation_steps=valid_steps,
        callbacks=callbacks
    )
```

Функция convert_to_vid создает видеофайл из последовательности изображений с помощью библиотеки OpenCV.

```
def convert_to_vid(video_name='miceSeg.avi', fps=30):
    path_to_video_images = os.path.join(DATABASE_PATH, LAST_VIDEO_IMAGES)
    if not os.path.exists(path_to_video_images):
```

```
        print("Run mask_video first")
        return

    images = sorted([img for img in os.listdir(path_to_video_images) if img.endswith((".jpg",
".jpeg", ".png"))])
    if not images:
        print("No images found")
        return

    frame = cv2.imread(os.path.join(path_to_video_images, images[0]))
    if frame is None:
        print("Failed to read first frame")
        return

    height, width, _ = frame.shape

    path_to_video = os.path.join(DATABASE_PATH, video_name)
    video = cv2.VideoWriter(path_to_video, cv2.VideoWriter_fourcc(*'DIVX'), fps, (width,
height))

    for image in tqdm(images, desc="Creating video"):
        img = cv2.imread(os.path.join(path_to_video_images, image))
        if img is not None:
            video.write(img)
        else:
            print(f"Failed to load image: {image}")

    video.release()
    print(f"Video generated successfully: {path_to_video}")
```

Функция mask_video выполняет обработку видео с применением сегментационной модели - накладывает маски сегментации на каждый кадр видео и сохраняет результат, тоже с помощью библиотеки OpenCV.

```
def mask_video(file_name, output_video_name):
    path_to_video_images = os.path.join(DATABASE_PATH, LAST_VIDEO_IMAGES)
    os.makedirs(path_to_video_images, exist_ok=True)

    # Очистка предыдущих файлов
    for file in glob(os.path.join(path_to_video_images, '*')):
        try:
            os.remove(file)
        except:
            print(f"Error deleting file: {file}")
```

```
    model_path = os.path.join(DATABASE_PATH, MODEL_NAME + FILE_EXT)
    if not os.path.exists(model_path):
        print(f"Model not found at {model_path}")
        return

    model = tf.keras.models.load_model(model_path, custom_objects={'MaxUnpooling2D':
MaxUnpooling2D})
    cap = cv2.VideoCapture(file_name)
    if not cap.isOpened():
        print(f"Failed to open video: {file_name}")
        return

    fps = cap.get(cv2.CAP_PROP_FPS)
    cur_im = 0

    print(f"Starting video processing: {file_name}")

    while True:
        success, img = cap.read()
        if not success:
            break

        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        h_orig, w_orig = img_rgb.shape[:2]
        x = mod_img(img_rgb)

        x = np.expand_dims(x, axis=0)
        pred_mask = model.predict(x, verbose=0)[0]

        pred_mask = np.concatenate([pred_mask] * 3, axis=2)
        pred_mask = (pred_mask > 0.5) * 255
        pred_mask = pred_mask.astype(np.uint8)

        blur = cv2.GaussianBlur(pred_mask, (13, 13), 0)
        _, thresh = cv2.threshold(blur, 100, 255, cv2.THRESH_BINARY)

        pred_mask = cv2.resize(thresh, (w_orig, h_orig), interpolation=cv2.INTER_NEAREST)
        original_image = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR)
        original_image = cv2.resize(original_image, (w_orig, h_orig),
interpolation=cv2.INTER_LINEAR)
        original_image = original_image.astype(np.uint8)

        alpha = 0.6
        try:
```

```
            output_image = cv2.addWeighted(pred_mask, alpha, original_image, 1 - alpha, 0)
        except Exception as e:
            print(f"Error in cv2.addWeighted for frame {cur_im}: {e}")
            continue

        name = os.path.join(path_to_video_images, f"{cur_im:06d}.jpg")
        cv2.imwrite(name, output_image)
        cur_im += 1
        if cur_im % 30 == 0:
            print(f"Processed {cur_im} frames")

    cap.release()
    print(f"Video processing completed. Total frames: {cur_im}")

    # Создание видео сразу после обработки
    convert_to_vid(output_video_name, fps)
    return fps
```

Функция train_and_fit_model() является главной координационной функцией для всего процесса обучения модели.

```
def train_and_fit_model():
    tf.keras.backend.clear_session()
    gc.collect()

    if DATABASE_PATH:
        result = train_model()
        if result is None:
            return
        model, epochs, train_steps, valid_steps, callbacks, train_dataset, valid_dataset = result
        print(model.summary())

        physical_devices = tf.config.list_physical_devices('GPU')
        if physical_devices:
            tf.config.experimental.set_memory_growth(physical_devices[0], True)

        fit_model(model, epochs, train_steps, valid_steps, callbacks, train_dataset, valid_dataset)

        tf.keras.backend.clear_session()
        gc.collect()

        print("Training finished")
    else:
        print("Choose correct folder with Train/Test data")
```

2.3 Тестирование

Проверка доступности GPU, TenseFlow и CUDA.

```
Setting up GPU and CUDA...
1 Physical GPU(s), 1 Logical GPU(s)
Using GPU: /physical_device:GPU:0
INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:GPU:0',)
Number of devices: 1
TensorFlow version: 2.10.0
WARNING:tensorflow:From C:\Users\kanze\AppData\Local\Temp\ipykernel_6876\2127670025.py:6: is_gpu_avail
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
GPU available: True
CUDA available: True
```

Рисунок 6 - доступность GPU, TenseFlow и CUDA

Обучение самой модели SegNet.

```
Train: 877 images, 877 masks | Test: 49 images, 49 masks
Effective Train: 864, Test: 48
Model: "SegNet"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3  0          []
                             )]

 conv2d (Conv2D)             (None, 256, 256, 64  1792       ['input_1[0][0]']
                             )

 batch_normalization (BatchNorm  (None, 256, 256, 64  256        ['conv2d[0][0]']
 alization)                  )

 activation (Activation)     (None, 256, 256, 64  0          ['batch_normalization[0][0]']
                             )

 conv2d_1 (Conv2D)           (None, 256, 256, 64  36928      ['activation[0][0]']
                             )

 batch_normalization_1 (BatchNo  (None, 256, 256, 64  256        ['conv2d_1[0][0]']
 rmalization)                )

 activation_1 (Activation)   (None, 256, 256, 64  0          ['batch_normalization_1[0][0]']
                             )

 max_pooling2d (MaxPooling2D)  (None, 128, 128, 64  0          ['activation_1[0][0]']
                             )
```

Рисунок 7 – обучение модели.

Результаты метрик, собранные во время обучения SegNet представелны в таблице 2.

epoch	acc	loss	Learning _rate	precision	recall	val_acc	val_loss	val_pr	val_recall
1	0.966917	0.195911	1,00E-04	0.364419	0.748397	0.977161	0.457486	0.0	0.0
2	0.993396	0.080622	1,00E-04	0.911377	0.763470	0.977161	0.289378	0.0	0.0

3	0.993982	0.067492	1,00E-04	0.927492	0.777615	0.977161	0.194976	0.0	0.0
4	0.994535	0.060782	1,00E-04	0.938268	0.795164	0.977161	0.149210	0.0	0.0
5	0.994810	0.056287	1,00E-04	0.941941	0.805403	0.977161	0.128967	0.0	0.0
6	0.995006	0.052511	1,00E-04	0.946142	0.811167	0.977161	0.118491	0.0	0.0
7	0.995077	0.049216	1,00E-04	0.949218	0.811782	0.977161	0.114949	0.0	0.0
8	0.995209	0.045808	1,00E-04	0.949601	0.817951	0.977161	0.109463	0.0	0.0
9	0.995267	0.042578	1,00E-04	0.951312	0.819230	0.978261	0.097580	1.0	0.048187
10	0.995430	0.039401	1,00E-04	0.952334	0.826325	0.986533	0.067388	0.996263	0.411914
11	0.995633	0.036439	1,00E-04	0.949847	0.838816	0.991578	0.050688	0.975747	0.647365
12	0.995887	0.033694	1,00E-04	0.943099	0.858226	0.991978	0.046359	0.987430	0.657150
13	0.996243	0.031167	1,00E-04	0.938170	0.881305	0.993361	0.040325	0.985852	0.719674
14	0.996579	0.028849	1,00E-04	0.937373	0.899114	0.994023	0.036337	0.977701	0.755543
15	0.996695	0.027072	1,00E-04	0.935892	0.906567	0.994492	0.033662	0.959960	0.791899
16	0.996898	0.025240	1,00E-04	0.937714	0.914801	0.994692	0.031307	0.960088	0.800890
17	0.997002	0.023767	1,00E-04	0.937881	0.919875	0.995170	0.029053	0.959047	0.823703
18	0.997162	0.022291	1,00E-04	0.940168	0.925340	0.995020	0.028627	0.953430	0.822130
19	0.997296	0.0209481	1,00E-04	0.9414778	0.9306256 1	0.995425	0.026609	0.947265	0.846836

Таблица 2 - Результаты метрик таблицы SegNet.

Так же были нарисованы графики для наглядности результатов метрик на рисунке 8.

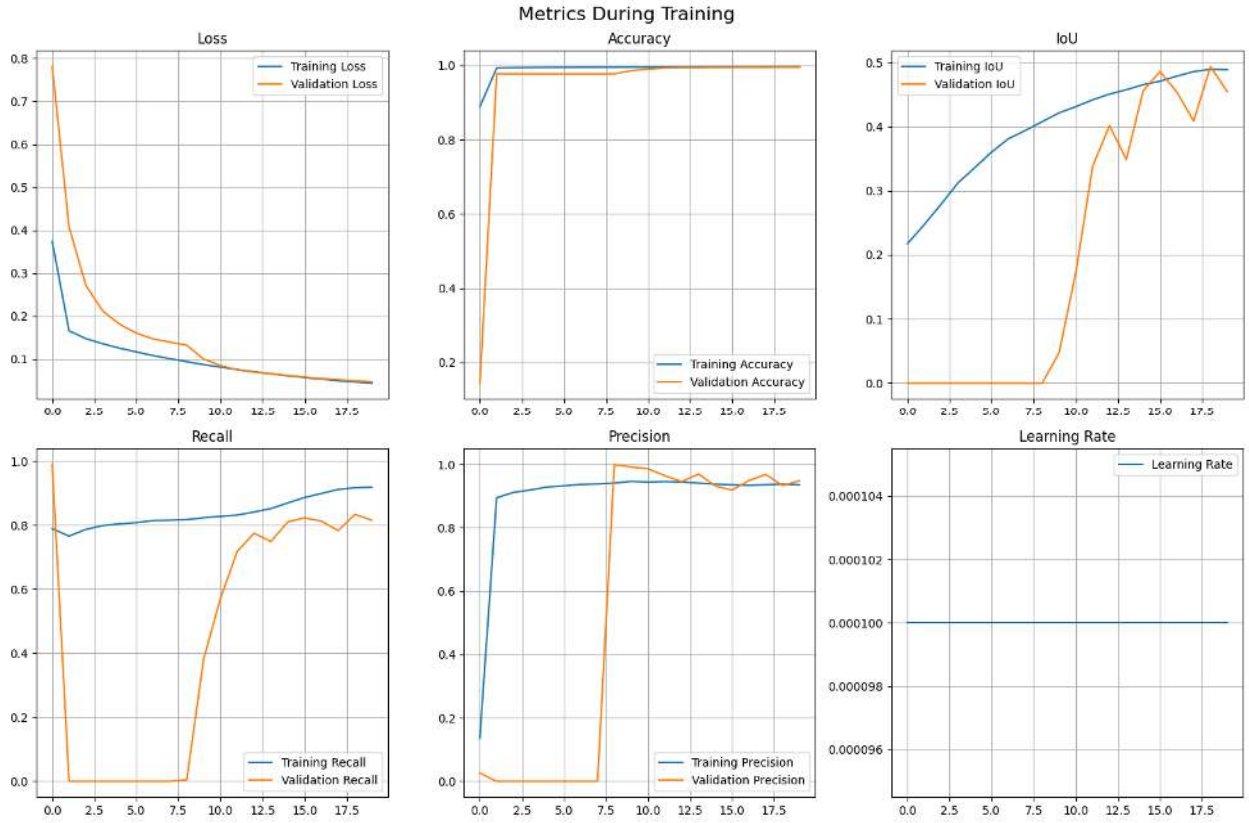

Рисунок 8 – графики с метриками.

Работоспособность программы, можно увидеть на рисунке 9, взять скриншот из видео, которое скомпилировалось.

Рисунок 9 – фрагмент из сегментированного видео.

3. Заключение

В результате научно-исследовательской работы был изучен материал по построению архитектуры модели. Написана программа, для сегментации видео с помощью архитектурной модели SegNet.

В заключении можно сказать, что архитектурная модель SegNet была протестирована и реализована. В итоге показывает нормальный результат сегментации видео.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. U-Net: нейросеть для сегментации изображений [Электронный ресурс] URL: https://neurohive.io/ru/vidy-nejrosetej/u-net-image-segmentation/ (дата обращения 1.09.2025).
2. Fully Convolutional Network [Электронный ресурс] URL: https://deepmachinelearning.ru/docs/Neural-networks/Semantic-segmentation/FCN (дата обращения 1.09.2025).
3. Эволюция архитектур нейросетей в компьютерном зрении: сегментация изображений [Электронный ресурс] URL: https://habr.com/ru/companies/slsoft/articles/864994/ (дата обращения 1.09.2025).
4. Эволюция архитектур нейросетей в компьютерном зрении: сегментация изображений [Электронный ресурс] URL: https://habr.com/ru/companies/slsoft/articles/864994/ (дата обращения 1.09.2025).
5. Эволюция архитектур нейросетей в компьютерном зрении: сегментация изображений [Электронный ресурс] URL: https://habr.com/ru/companies/slsoft/articles/864994/ (дата обращения 1.09.2025).
6. TensorFlow: обзор программы [Электронный ресурс] URL: https://practicum.yandex.ru/blog/pro-chto-takoe-tensorflow/ (дата обращения 1.09.2025).
7. CUDA [Электронный ресурс] URL: https://itglobal.com/ru-ru/company/glossary/cuda/ (дата обращения 1.09.2025).
8. Общая информация по CUDA [Электронный ресурс] URL: https://www.fastvideo.ru/info/cuda/cuda-info.htm (дата обращения 1.09.2025).
9. OpenCV [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/OpenCV (дата обращения 1.09.2025).