|  |  |  |
| --- | --- | --- |
| 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 3](#_Toc211371213)

[1. Теоретическая часть 5](#_Toc211371214)

[1.1. Семантическая сегментация. 5](#_Toc211371215)

[1.2. Архитектура нейронных сетей для сегментации. 5](#_Toc211371216)

[1.3. Библиотека TenserFlow 9](#_Toc211371217)

[1.4. Библиотека CUDA и CuDNN 10](#_Toc211371218)

[1.5. Библиотека OpenCV. 11](#_Toc211371219)

[2. ПРАКТИЧЕСКАЯ ЧАСТЬ 12](#_Toc211371220)

[2.1 Архитектура и разработка 12](#_Toc211371221)

[2.2 Класс, функции и тренировочная модель. 12](#_Toc211371222)

[2.3 Тестирование 20](#_Toc211371223)

[3. Заключение 23](#_Toc211371224)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 24](#_Toc211371225)

# 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 «Муль-тиНейро-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».

# Теоретическая часть

# 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.

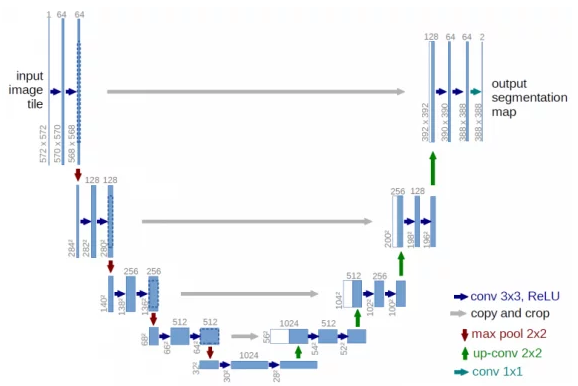
Пороговая сегментация: использует порог яркости или другой характеристики для разделения изображения на сегменты, часто применяется для бинарных изображений.

Для дальнейшей работы, был выбран тип бинарной сегментации.

# Архитектура нейронных сетей для сегментации.

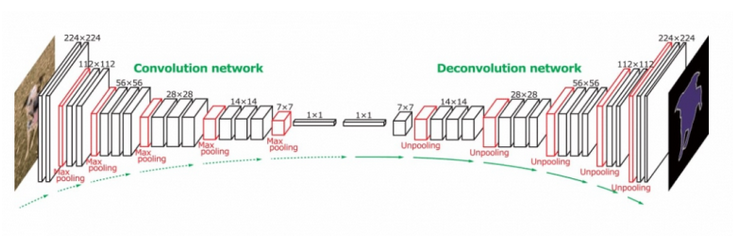
Архитектуры нейросетей для сегментации изображений представляют собой специальные модели глубокого обучения, предназначенные для точного выделения объектов на изображениях путем классификации каждого пикселя. Среди самых известных и эффективных архитектур выделяются:

* 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].



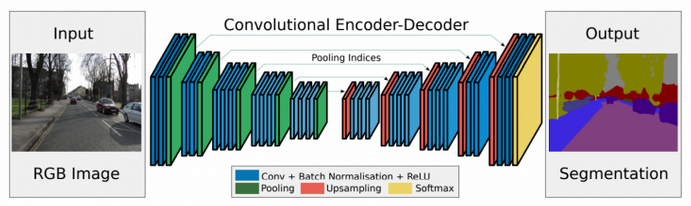
*Рисунок 1 – архитектура U-Net*

* 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 восcтановления размеров изображения и skip connections для объединения признаков разных уровней [2].



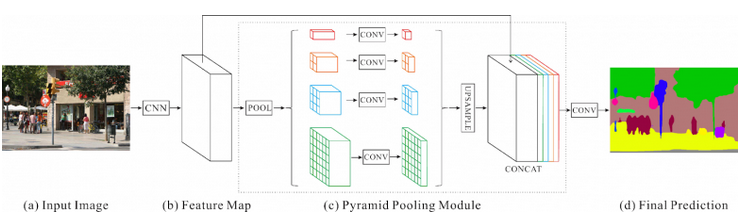
*Рисунок 2 – архитектура FCN*

* 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].



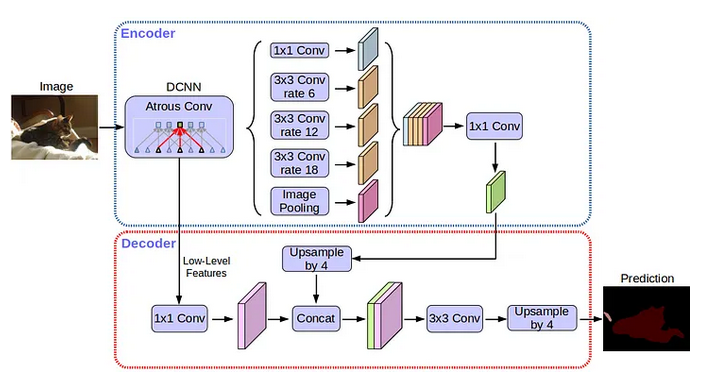
*Рисунок 3 – архитектура SegNet*

* 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.

# 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.

# 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.

# 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.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

# 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».

# 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.

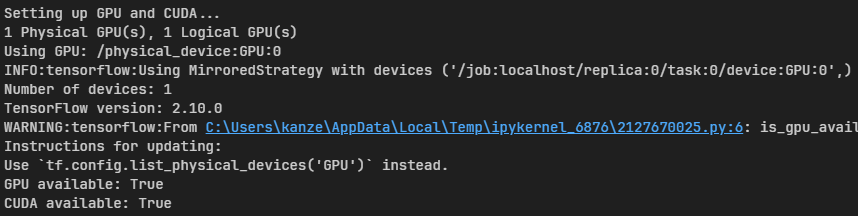
|  |
| --- |
| def mask\_video(file\_name, output\_video\_name):  path\_to\_video\_images = os.path.join(DATABASE\_PATH, LAST\_VIDEO\_IMAGES)  os.makedirs(path\_to\_video\_images, exist\_ok=True)  # Очистка предыдущих файлов  for file in glob(os.path.join(path\_to\_video\_images, '\*')):  try:  os.remove(file)  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.

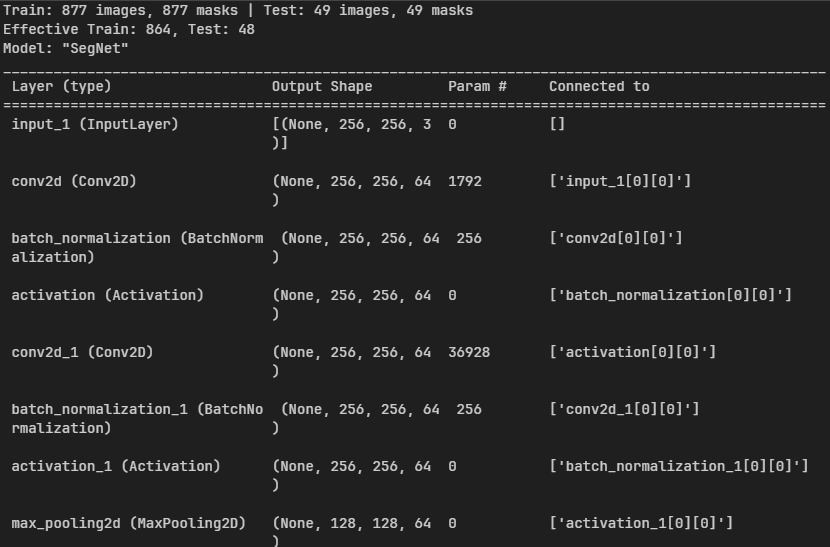
|  |
| --- |
| def train\_and\_fit\_model():  tf.keras.backend.clear\_session()  gc.collect()  if DATABASE\_PATH:  result = train\_model()  if result is None:  return  model, epochs, train\_steps, valid\_steps, callbacks, train\_dataset, valid\_dataset = result  print(model.summary())  physical\_devices = 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") |

# Тестирование

Проверка доступности GPU, TenseFlow и CUDA.



*Рисунок 6 - доступность GPU, TenseFlow и CUDA*

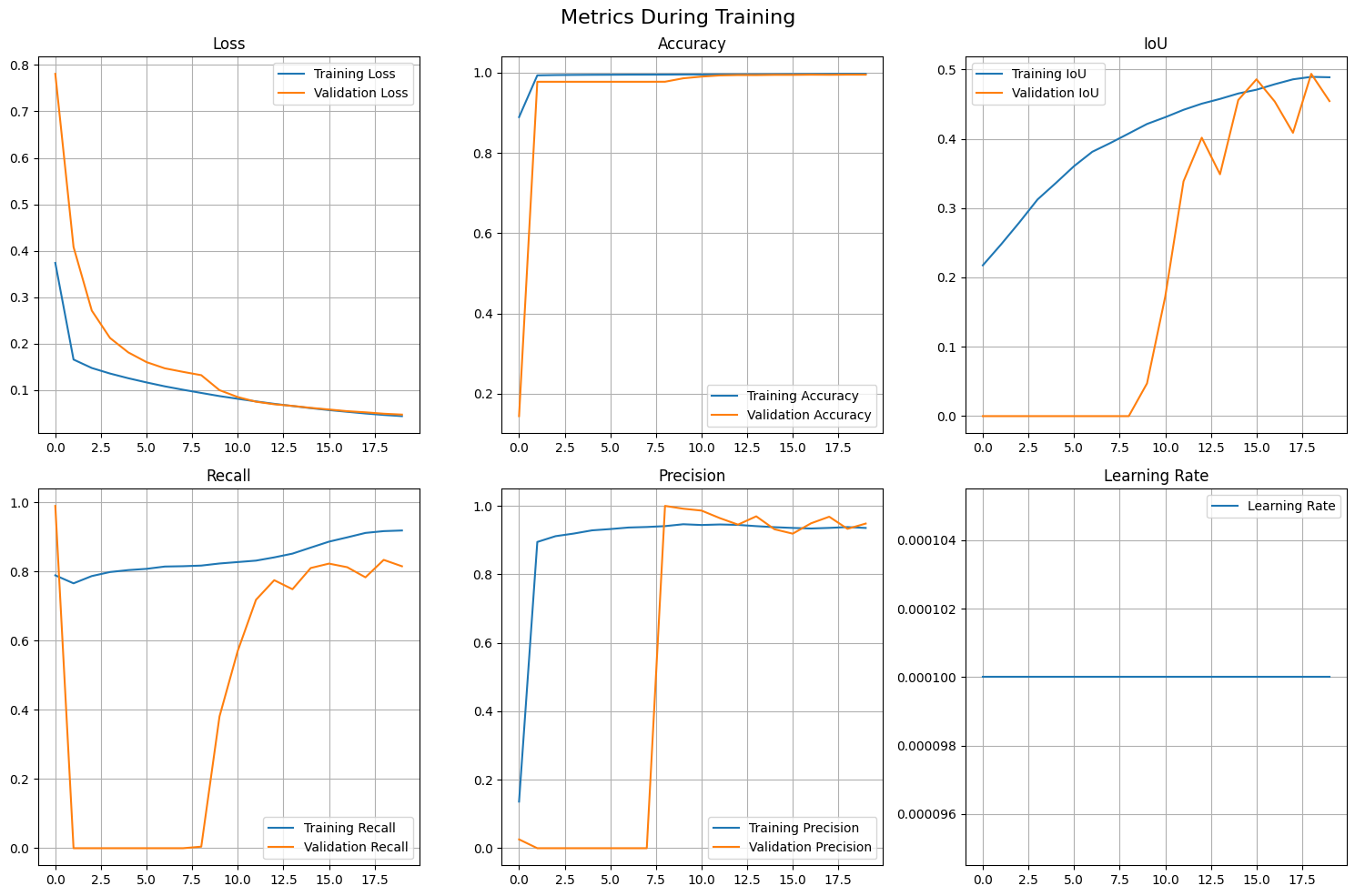
Обучение самой модели SegNet.

*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 | 0.93062561 | | 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.*

# 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).