**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc165034597)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7](#_Toc165034598)

[1.1 Задачи распознавания 7](#_Toc165034599)

[1.2 Архитектура U-Net 9](#_Toc165034600)

[1.3 Обучение U-Net 11](#_Toc165034601)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 12](#_Toc165034602)

[2.1 Реализация U-Net 12](#_Toc165034603)

[2.2 Реализация архитектуры U-Net 14](#_Toc165034604)

[ЗАКЛЮЧЕНИЕ 16](#_Toc165034605)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 17](#_Toc165034606)

[ПРИЛОЖЕНИЯ 18](#_Toc165034607)

# ВВЕДЕНИЕ

В современном мире с развитием технологий компьютерного зрения и машинного обучения возникает все больше возможностей для автоматизации процессов распознавания объектов на изображениях и видео. Одной из актуальных задач в этой области является распознавание бытовых вещей на снимках и видео.

Распознавание бытовых вещей имеет широкий спектр применений, начиная от улучшения процесса поиска и классификации объектов на изображениях до создания инновационных систем умного дома и робототехники. При этом, точность и скорость распознавания играют ключевую роль в эффективности таких систем.

Цель данного исследования заключается в разработке алгоритма машинного обучения, способного точно распознавать бытовые вещи на снимках и видео. Для достижения этой цели будут использованы современные методы обработки изображений, а также алгоритмы распознавания объектов.

Исследование по распознаванию бытовых вещей на снимках и видео имеет практическую значимость для различных областей, включая розничную торговлю, безопасность, медицину и другие сферы человеческой деятельности. Результаты данного исследования могут быть использованы для создания инновационных технологических решений, способствующих повышению качества жизни и улучшению рабочих процессов.

Таким образом, разработка алгоритма распознавания бытовых вещей на снимках и видео представляет собой актуальную задачу, которая имеет потенциал для широкого применения в различных областях и секторах экономики.

Задачи исследования:

1. Провести обзор существующих методов распознавания объектов на изображениях и видео.

2. Собрать и подготовить набор данных, содержащий разнообразные изображения бытовых вещей.

3. Проанализировать особенности бытовых вещей, которые могут повлиять на процесс распознавания.

4. Использовать современные методы обработки изображений для предобработки данных.

5. Обучить модель машинного обучения на подготовленном наборе данных для распознавания бытовых вещей.

6. Оценить точность и скорость работы разработанного алгоритма на тестовом наборе данных.

7. Провести сравнительный анализ с другими существующими методами распознавания объектов на изображениях и видео.

8. Предложить возможные пути улучшения алгоритма и его применение в различных областях.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

# 1.1 Задачи распознавания

Задачи распознавания в машинном обучении можно разделить на несколько основных видов, включающих в себя различные типы данных и целей решения задач. Ниже приведены основные виды задач распознавания:

1. Классификация: Задача классификации заключается в присвоении объектам одной или нескольких категорий на основе их характеристик. Например, классификация электронных писем на спам и не-спам, классификация изображений на категории животных и транспорта и т.д.
2. Регрессия: Задача регрессии направлена на предсказание числовых значений на основе входных данных. Например, прогнозирование цены недвижимости на основе характеристик дома, прогнозирование продаж товаров и т.д.
3. Детектирование объектов: Задача детектирования объектов заключается в обнаружении и локализации объектов на изображениях или видео. Эта задача используется, например, для распознавания лиц, автомобилей, пешеходов и других объектов, таких как YOLO (You Only Look Once) или Faster R-CNN.
4. Сегментация: Задача сегментации заключается в разделении изображения на сегменты или области схожих пикселей. Это позволяет выделить объекты на изображении и провести более детальный анализ и обработку.
5. Кластеризация: Задача кластеризации направлена на группировку данных на основе их сходства без предварительно заданных категорий. Это позволяет выявить скрытые паттерны в данных и выделить группы объектов схожих характеристик, таких как K-means или DBSCAN.
6. Обнаружение аномалий: Задача обнаружения аномалий заключается в выявлении необычных или аномальных данных, которые отличаются от обычного поведения. Это может быть полезно для выявления мошеннических операций, дефектов в производстве и других нестандартных событий, таких как One-Class SVM или Isolation Forest.

Эти виды задач распознавания широко используются в различных областях, таких как компьютерное зрение, обработка естественного языка, биомедицинская обработка данных и другие. Каждый вид задач требует специфических подходов и методов для успешного решения.

Я покажу мультиклассовую сегментацию, а это значит мне нужно будет найти и выделить несколько различных объектов на одном изображений.

Для решения задачи мультиклассовой сегментации применяются различные алгоритмы машинного обучения и глубокого обучения, включая сверточные нейронные сети (CNN), такие как U-Net, Mask R-CNN, FCN и другие. Эти алгоритмы позволяют обучать модель на больших наборах данных и делать точные предсказания для каждого пикселя изображения относительно его класса.

Мультиклассовая сегментация широко используется в различных областях, таких как медицинская диагностика, автоматическое распознавание объектов, анализ изображений и тому подобное, где точность и точная сегментация объектов на изображениях являются ключевыми задачами.

# 1.2 Архитектура U-Net

U-Net — это свёрточная нейронная сеть, которая была создана в 2015 году для сегментации биомедицинских изображений в отделении Computer Science Фрайбургского университета. Архитектура сети представляет собой полносвязную свёрточную сеть, модифицированную так, чтобы она могла работать с меньшим количеством примеров (обучающих образов) и делала более точную сегментацию, рисунок 1.

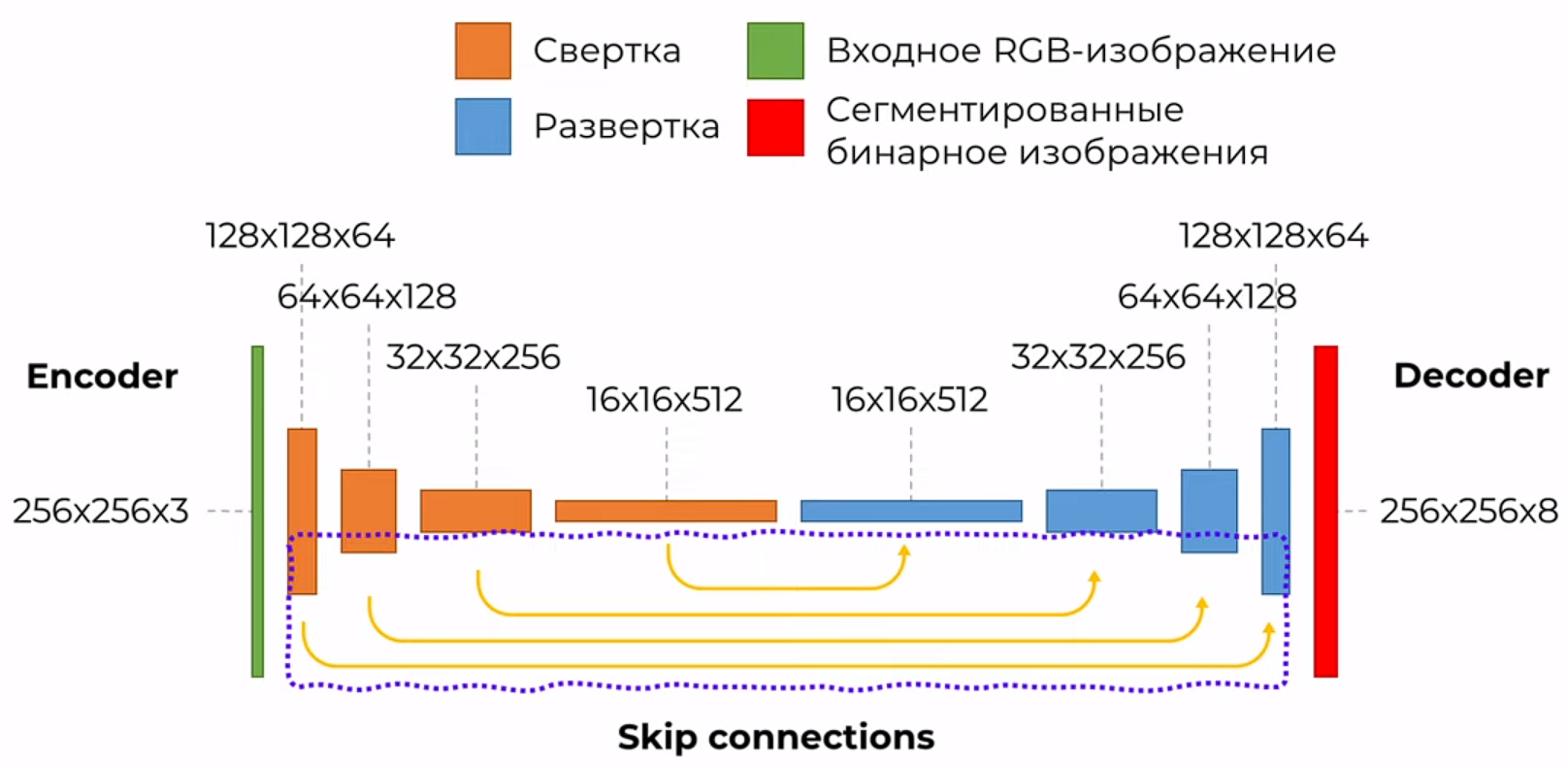


Рисунок 1 – Архитектура U-Net

Она состоит из сокращающего пути (левая сторона – Encoder) и расширяющего пути (правая сторона – Decoder). Сокращающий путь следует типичной архитектуре сверточной сети. Он состоит из последовательного применения двух сверток 3x3 (без заполнения), за каждой из которых следует блок выпрямленного линейного элемента (ReLU) и операция max-pooling 2x2 с шагом 2 для понижения разрешения. На каждом этапе понижения разрешения мы удваиваем количество каналов признаков. Каждый шаг расширяющего пути состоит из повышения разрешения карты признаков, за которым следует свертка 2x2 («верхняя свертка»), которая делит пополам количество каналов признаков, конкатенация с соответствующим образом обрезанной картой признаков из сокращающего пути и две свертки 3x3, за каждой из которых следует ReLU. Обрезка необходима из-за потери граничных пикселей при каждой свертке. На последнем слое используется свертка 1x1 для отображения каждого 64-компонентного вектора признаков в желаемое количество классов. Всего сеть имеет 23 сверточных слоя. Чтобы обеспечить бесшовное черепичное разбиение выходной карты сегментации, важно выбрать размер входного фрагмента таким образом, чтобы все операции max-pooling 2x2 применялись к слою с четными размерами x и y.

Первая часть это сверточная сеть она состоит из комбинации сверточных слоев, слоев пакетной нормализациии и активационной функции LeakyReLU()

Вторая часть состоит из комбинации Transpose сверточных слоев, слоев пакетной нормализации, Dropout слоев и активационной функции ReLU()

Dropout слои являются инструментом для предотвращения переобучения сетей, а также в целом положительно сказываются на процессе её обучения

Слои Batch (пакетной нормализации) применяются для того, чтобы повысить производительность сети и её работу стабильней

Сверточные Transpose слои применяются для того, чтобы провести обратную операцию свертки и увечить размерность выходных данных

Особенность Архитектуры UNet это применение Skip connections сейчас это стандартный инструмент во многих сверточных архитектурах. Это техника часто применяется в глубоких архитектурах и положительно сказываются на сходимость модели. Благодаря, этим межслоевым соединениям информация низкого уровня совместно используется входными и выходными слоями. А в архитектуре UNet эти соединения используются для того, чтобы передать признаки, полученные в Encoder'e в Decoder. Это помогает восстановить простанственную информацию, которая была потерянная в результате операции свертки. В целом, Skip connections позволяют использовать ранние извлеченные признаки в Decoder'е и стабилирует процесс обучения и сходимости нейронной сети.

# 1.3 Обучение U-Net

Сеть обучается методом стохастического градиентного спуска на основе входных изображений и соответствующих им карт сегментации. Из-за сверток выходное изображение меньше входного сигнала на постоянную ширину границы. Применяемая попиксельно, функция soft-max вычисляет энергию по окончательной карте свойств вместе с функцией кросс-энтропии. Кросс-энтропия, вычисляемая в каждой точке, определяется так:

Граница разделения вычисляется с использованием морфологических операций. Затем вычисляется карта весовых коэффициентов:

)

где — карта весов для балансировки частот классов, — расстояние до границы ближайшей ячейки, а — расстояние до границы второй ближайшей ячейки.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

# 2.1 Реализация U-Net

Приведем описание полей и методов:

Поля:

1. CLASSES: Количество классов для сегментации (например, фон, объект 1, объект 2 и т.д.).
2. COLORS: Список цветов, которые будут соответствовать каждому классу для визуализации.
3. SAMPLE\_SIZE: Размер (ширина, высота), до которого изображения будут изменены при обучении.
4. OUTPUT\_SIZE: Желаемый размер (ширина, высота) для предсказаний модели.

Алгоритм def load\_images(image, mask) – Функции загрузки данных:

1. Загрузить изображение.
2. Загрузить маску.
3. Создать индивидуальные маски для каждого класса.
4. Объединить и изменить форму масок
5. Вернуть обработанное изображение и маски

Алгоритм def augmentate\_images(image, masks):

1. Сделать случайную обрезку:
2. Сделать случайное переворачивание (по горизонтали).
3. Изменить размер.

Алгоритм def input\_layer():

1. Определяет входной слой нейронной сети.
2. Устанавливает размер входных данных, используя SAMPLE\_SIZE.
3. Возвращает объект tf.keras.layers.Input.

Алгоритм def downsample\_block(filters, size, batch\_norm=True):

1. Формирует блоки, из которых состоит Encoder нейронной сети.
2. Использует инициализатор GlorotNormal для весовых коэффициентов.
3. Возвращает объект tf.keras.Sequential.

Алгоритм def upsample\_block(filters, size, dropout=False):

1. Формирует блоки, из которых состоит Decoder нейронной сети.
2. Использует инициализатор GlorotNormal для весовых коэффициентов.
3. Возвращает объект tf.keras.Sequential.

Алгоритм def output\_layer(size):

* + 1. Определяет выходной слой нейронной сети.
    2. Принимает аргумент size: Размер ядра транспонированной свертки.
    3. Использует инициализатор GlorotNormal для весовых коэффициентов.
    4. Возвращает объект tf.keras.layers.Conv2DTranspose.

Алгоритм def dice\_mc\_metric(filters, size, batch\_norm=True): – Функция Метрика Dice

1. Разделение по каналам.
2. Расчет Dice для каждого класса.
3. Рассчитать среднее значение Dice.
4. Возвращается avg\_dice, как оценка точности сегментации

Алгоритм def dice\_mc\_loss(filters, size, dropout=False): – Функция потерь Dice

1. Вычисление Dice.
2. Функция потерь.

Алгоритм def dice\_bce\_mc\_loss(filters, size, dropout=False): – Комбинированная функция потерь

* + 1. Вычисление Dice.
    2. Вычисление BCE.
    3. Вычислить комбинированную функцию потерь (0.3 \* Dice + BCE)

# 2.2 Реализация архитектуры U-Net

Перейдем к описанию алгоритма

* + 1. Настройка параметров:
  1. inp\_layer – начальный слой модели, который будет принимать входные изображения.
  2. downsample\_stack – список функций downsample\_block, которые будут последовательно уменьшать размерность признаков изображения.
  3. upsample\_stack – список функций upsample\_block увеличивает размерность признаков и восстанавливает пространственную информацию изображения.
  4. out\_layer – количество классов объектов.
     1. Перебираются блоки энкодера, на каждом шаге сохраняется выходной тензор признаков.
  5. Последний элемент списка отбрасывается, так как он не будет использоваться для соединения.
  6. Результатом становится список тензоров признаков с разных уровней энкодера.
     1. Перебираются декодера и соответствующие им тензоры признаков из энкодера
  7. Применяется блок декодера к текущему тензору признаков
  8. Объединяются тензоры признаков из декодера и энкодера с помощью конкатенации по каналам.
     1. Создается модель Keras, которая принимает входные данные из inp\_layer и возвращает результат сегментации из out\_layer.
     2. Генерируется графическое представление архитектуры модели с указанием размеров тензоров на каждом слое.

На Рисунке 2 изображена модель.

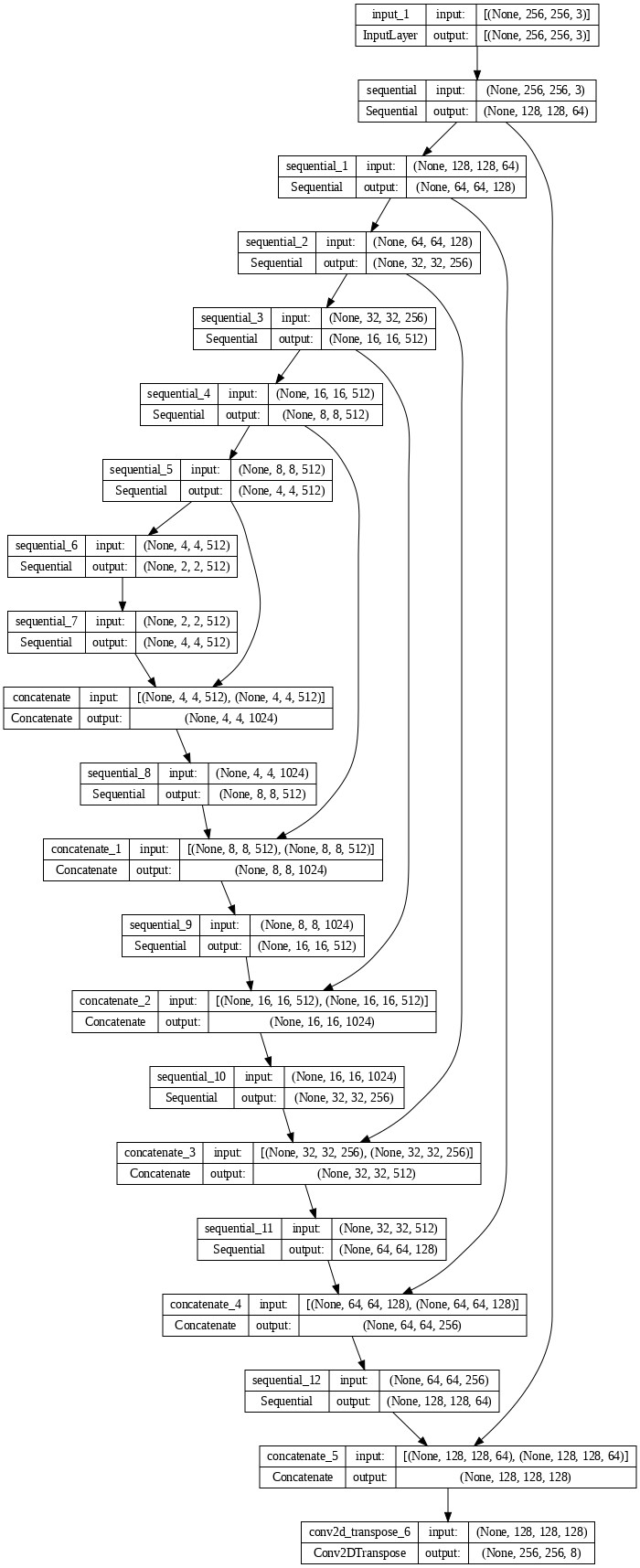


Рисунок 2 – Архитектура U-Net

Рисунок 2 – Модель нейросети

# ЗАКЛЮЧЕНИЕ

В ходе данного исследования были разработаны алгоритмы и создан программный продукт, способный эффективно распознавать объекты на изображениях и видео с использованием методов компьютерного зрения и глубокого обучения.

В процессе работы были рассмотрены основные подходы к распознаванию объектов, проведен анализ существующих методов и выбраны наиболее подходящие для решения поставленных задач. Разработанные алгоритмы были тщательно протестированы на различных наборах данных, что позволило добиться высокой точности и надежности результатов.

Полученные в ходе работы результаты свидетельствуют о возможности успешного применения разработанных методов в различных областях, где требуется автоматическое распознавание объектов на изображениях и видео. Программный продукт, созданный в рамках данного исследования, представляет собой важный инструмент для повышения эффективности и автоматизации процессов в таких областях, как медицина, безопасность, автоматизация производства и другие.

Дальнейшее развитие данного направления исследований позволит улучшить качество распознавания объектов на изображениях и видео, расширить функциональность программного продукта и повысить его применимость в новых областях. Результаты данной работы могут служить основой для будущих исследований в области компьютерного зрения и глубокого обучения, способствуя развитию современных технологий и повышению их практической ценности.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

* 1. https://ru.wikipedia.org/wiki/U-Net
  2. <https://arxiv.org/pdf/1505.04597.pdf>
  3. https://neurohive.io/ru/vidy-nejrosetej/u-net-image-segmentation/

# ПРИЛОЖЕНИЯ

Приложение А — Листинги кода программы.

**Приложение А**

Код класса TPM представлен на листинге 1.

|  |
| --- |
| import os  import cv2  import glob  import numpy as np  import tensorflow as tf  import matplotlib.pyplot as plt  from skimage import measure  from skimage.io import imread, imsave  from skimage.transform import resize  from skimage.morphology import dilation, disk  from skimage.draw import polygon\_perimeter  from moviepy.editor import VideoFileClip  from datetime import timedelta  print(f'Tensorflow version {tf.\_\_version\_\_}')  print(f'GPU is {"ON" if tf.config.list\_physical\_devices("GPU") else "OFF" }')  SAVING\_FRAMES\_PER\_SECOND = 10  def format\_timedelta(td):  result = str(td)  try:  result, ms = result.split(".")  except ValueError:  return result + ".00".replace(":", "-")  ms = round(int(ms) / 10000)  return f"{result}.{ms:02}".replace(":", "-")  def main(video\_file):  video\_clip = VideoFileClip(video\_file)  filename, \_ = os.path.splitext(video\_file)  if not os.path.isdir(filename):  os.mkdir(filename)  saving\_frames\_per\_second = min(video\_clip.fps, SAVING\_FRAMES\_PER\_SECOND)  step = 1 / video\_clip.fps if saving\_frames\_per\_second == 0 else 1 / saving\_frames\_per\_second  for current\_duration in np.arange(0, video\_clip.duration, step):  frame\_duration\_formatted = format\_timedelta(timedelta(seconds=current\_duration)).replace(":", "-")  frame\_filename = os.path.join(filename, f"frame{frame\_duration\_formatted}.jpg")  video\_clip.save\_frame(frame\_filename, current\_duration)  video\_file = '/content/fqw\_nn/video/video.mp4'  main(video\_file)  CLASSES = 8  COLORS = ['black', 'red', 'lime',  'blue', 'orange', 'pink',  'cyan', 'magenta']  SAMPLE\_SIZE = (256, 256)  OUTPUT\_SIZE = (1080, 1920)  def load\_images(image, mask):  image = tf.io.read\_file(image)  image = tf.io.decode\_jpeg(image, channels=3)  image = tf.image.resize(image, OUTPUT\_SIZE)  image = tf.image.convert\_image\_dtype(image, tf.float32)  image = image / 255.0  mask = tf.io.read\_file(mask)  mask = tf.io.decode\_png(mask, channels=3)  mask = tf.image.rgb\_to\_grayscale(mask)  mask = tf.image.resize(mask, OUTPUT\_SIZE)  mask = tf.image.convert\_image\_dtype(mask, tf.float32)  masks = []  for i in range(CLASSES):  masks.append(tf.where(tf.equal(mask, float(i)), 1.0, 0.0))  masks = tf.stack(masks, axis=2)  masks = tf.reshape(masks, OUTPUT\_SIZE + (CLASSES,))  return image, masks  def augmentate\_images(image, masks):  random\_crop = tf.random.uniform((), 0.3, 1)  image = tf.image.central\_crop(image, random\_crop)  masks = tf.image.central\_crop(masks, random\_crop)  random\_flip = tf.random.uniform((), 0, 1)  if random\_flip >= 0.5:  image = tf.image.flip\_left\_right(image)  masks = tf.image.flip\_left\_right(masks)  image = tf.image.resize(image, SAMPLE\_SIZE)  masks = tf.image.resize(masks, SAMPLE\_SIZE)  return image, masks  images = sorted(glob.glob('fqw\_nn/dataset/img/\*.jpg'))  masks = sorted(glob.glob('fqw\_nn/dataset/masks/\*.png'))  images\_dataset = tf.data.Dataset.from\_tensor\_slices(images)  masks\_dataset = tf.data.Dataset.from\_tensor\_slices(masks)  dataset = tf.data.Dataset.zip((images\_dataset, masks\_dataset))  dataset = tf.data.Dataset.zip(images\_dataset, masks\_dataset)  dataset = dataset.map(load\_images, num\_parallel\_calls=tf.data.AUTOTUNE)  dataset = dataset.repeat(50)  dataset = dataset.map(augmentate\_images, num\_parallel\_calls=tf.data.AUTOTUNE)  im = '/content/fqw\_nn/dataset/img/frame0-00-00.00.jpg'  mask = '/content/fqw\_nn/dataset/masks/frame0-00-00.00.png'  im = imread(im)  mask = imread(mask)  mask = resize(mask, (mask.shape[0], mask.shape[1]))  fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize=(10, 3), dpi=125)  ax[0].set\_title('Image')  ax[0].set\_axis\_off()  ax[0].imshow(im)  ax[1].set\_title('Mask')  ax[1].set\_axis\_off()  ax[1].imshow(mask \* 255 / 7)  plt.show()  plt.close()  images\_and\_masks = list(dataset.take(5))  fig, ax = plt.subplots(nrows = 2, ncols = 5, figsize=(16, 6))  for i, (image, masks) in enumerate(images\_and\_masks):  ax[0, i].set\_title('Image')  ax[0, i].set\_axis\_off()  ax[0, i].imshow(image)  ax[1, i].set\_title('Mask')  ax[1, i].set\_axis\_off()  ax[1, i].imshow(image / 1.5)  for channel in range(CLASSES):  contours = measure.find\_contours(np.array(masks[:,:,channel]))  for contour in contours:  ax[1, i].plot(contour[:, 1], contour[:, 0], linewidth=1, color=COLORS[channel])  plt.show()  plt.close()  train\_dataset = dataset.take(2000).cache()  test\_dataset = dataset.skip(2000).take(100).cache()  train\_dataset = train\_dataset.batch(8)  test\_dataset = test\_dataset.batch(8)  def input\_layer(): # задает входной слой нейронной сети и устанавливает размер входных данных  return tf.keras.layers.Input(shape=SAMPLE\_SIZE + (3,))  def downsample\_block(filters, size, batch\_norm=True): # устанавливает блоки формирует Encoder  initializer = tf.keras.initializers.GlorotNormal() # задает метод инициализации весовых коэффициентов  result = tf.keras.Sequential()  # включает сверточный слой  result.add(  tf.keras.layers.Conv2D(filters, size, strides=2, padding='same',  kernel\_initializer=initializer, use\_bias=False))  # добавляет слой пакетной нормализации  if batch\_norm:  result.add(tf.keras.layers.BatchNormalization())  # устанавливает активационную функцию  result.add(tf.keras.layers.LeakyReLU())  return result  def upsample\_block(filters, size, dropout=False): # помогает формировать Decoder нейронной сети  initializer = tf.keras.initializers.GlorotNormal()  result = tf.keras.Sequential()  # Transpose  result.add(  tf.keras.layers.Conv2DTranspose(filters, size, strides=2, padding='same',  kernel\_initializer=initializer, use\_bias=False))  result.add(tf.keras.layers.BatchNormalization())  # Dropout  if dropout:  result.add(tf.keras.layers.Dropout(0.25))  result.add(tf.keras.layers.ReLU())  return result  def output\_layer(size): # задает выходной слой  initializer = tf.keras.initializers.GlorotNormal()  return tf.keras.layers.Conv2DTranspose(CLASSES, size, strides=2, padding='same',  kernel\_initializer=initializer, activation='sigmoid')  inp\_layer = input\_layer()  downsample\_stack = [  downsample\_block(64, 4, batch\_norm=False),  downsample\_block(128, 4),  downsample\_block(256, 4),  downsample\_block(512, 4),  downsample\_block(512, 4),  downsample\_block(512, 4),  downsample\_block(512, 4),  ]  upsample\_stack = [  upsample\_block(512, 4, dropout=True),  upsample\_block(512, 4, dropout=True),  upsample\_block(512, 4, dropout=True),  upsample\_block(256, 4),  upsample\_block(128, 4),  upsample\_block(64, 4)  ]  out\_layer = output\_layer(4)  # Реализуем skip connections  x = inp\_layer  downsample\_skips = []  for block in downsample\_stack:  x = block(x)  downsample\_skips.append(x)  downsample\_skips = reversed(downsample\_skips[:-1])  for up\_block, down\_block in zip(upsample\_stack, downsample\_skips):  x = up\_block(x)  x = tf.keras.layers.Concatenate()([x, down\_block])  out\_layer = out\_layer(x)  unet\_like = tf.keras.Model(inputs=inp\_layer, outputs=out\_layer)  tf.keras.utils.plot\_model(unet\_like, show\_shapes=True, dpi=72)  # Метрика дает оценку точности результатов рабоыты нейронной сети  def dice\_mc\_metric(a, b): # a - ответ нейронной сети, b - результат, который должен получиться на самом деле, то чему старается научиться нейронная сеть  a = tf.unstack(a, axis=3) # Распаковываю многоканальные изображения маски  b = tf.unstack(b, axis=3)  dice\_summ = 0  for i, (aa, bb) in enumerate(zip(a, b)): # ищем среднее значение коэффициента dice  numenator = 2 \* tf.math.reduce\_sum(aa \* bb) + 1  denomerator = tf.math.reduce\_sum(aa + bb) + 1  dice\_summ += numenator / denomerator  avg\_dice = dice\_summ / CLASSES  return avg\_dice  def dice\_mc\_loss(a, b): # функция потери  return 1 - dice\_mc\_metric(a, b)  def dice\_bce\_mc\_loss(a, b): # комбинированная функция  return 0.3 \* dice\_mc\_loss(a, b) + tf.keras.losses.binary\_crossentropy(a, b)  unet\_like.compile(optimizer='adam', loss=[dice\_bce\_mc\_loss], metrics=[dice\_mc\_metric])  history\_dice = unet\_like.fit(train\_dataset, validation\_data=test\_dataset, epochs=25, initial\_epoch=0)  unet\_like.save\_weights('fqw\_nn/networks/unet\_like')  unet\_like.load\_weights('fqw\_nn/networks/unet\_like')  rgb\_colors = [  (0, 0, 0),  (255, 0, 0),  (0, 255, 0),  (0, 0, 255),  (255, 165, 0),  (255, 192, 203),  (0, 255, 255),  (255, 0, 255)  ]  frames = sorted(glob.glob('fqw\_nn/video/frames/\*.jpg'))  for filename in frames:  frame = imread(filename)  sample = resize(frame, SAMPLE\_SIZE)  predict = unet\_like.predict(sample.reshape((1,) + SAMPLE\_SIZE + (3,)))  predict = predict.reshape(SAMPLE\_SIZE + (CLASSES,))  scale = frame.shape[0] / SAMPLE\_SIZE[0], frame.shape[1] / SAMPLE\_SIZE[1]  frame = (frame / 1.5).astype(np.uint8)  for channel in range(1, CLASSES):  contour\_overlay = np.zeros((frame.shape[0], frame.shape[1]))  contours = measure.find\_contours(np.array(predict[:,:,channel]))  try:  for contour in contours:  rr, cc = polygon\_perimeter(contour[:, 0] \* scale[0],  contour[:, 1] \* scale[1],  shape=contour\_overlay.shape)  contour\_overlay[rr, cc] = 1  contour\_overlay = dilation(contour\_overlay, disk(1))  frame[contour\_overlay == 1] = rgb\_colors[channel]  except:  pass  imsave(f'fqw\_nn/video/processed/{os.path.basename(filename)}', frame)  images = glob.glob('/content/fqw\_nn/video/processed/\*.jpg')  images = sorted(images)  fourcc = cv2.VideoWriter\_fourcc(\*"mp4v")  video = cv2.VideoWriter(  filename="output.mp4", fourcc=fourcc, fps=30.0, frameSize=(430, 430)  )  for image in images:  frame = cv2.imread(image)  frame = cv2.resize(frame, dsize=(430, 430))  video.write(frame)  video.release() |