Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамента цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №8
дисциплины «Основы нейронных сетей»

Выполнил:
Гайчук Дарья Дмитриевна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А.-доцент департамента цифровых, роботехнических систем и электроники института перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Разработка и исследование моделей семантической сегментации изображений на базе архитектур U-Net и PSPNet.

Цель: изучить методы семантической сегментации изображений с использованием сверточных нейронных сетей. Провести экспериментальное сравнение различных архитектур (SimpleUnet, U-Net, PSPNet) на примере базы изображений строительной тематики, а также выполнить модификацию модели под задачи с уменьшенным числом классов. Оценить качество сегментации и влияние различных параметров архитектуры на точность модели.

Ссылка на git: https://github.com/Ichizuchi/NN_LR10

Ход работы

В первой практике рассматривается базовая задача семантической сегментации, где каждому пикселю изображения необходимо присвоить метку одного из 16 классов. Используется пара изображений: исходное цветное и соответствующее сегментированное. Для загрузки и предобработки данных реализуются функции считывания изображений, преобразования цветных меток в числовые классы и визуализации результата. После этого формируется обучающая и проверочная выборки. Первая модель, построенная на простой линейной архитектуре, демонстрирует базовую точность сегментации. Результаты визуализации показывают, что модель распознает некоторые объекты корректно, но точность невысока.

Далее проводится попытка упростить задачу за счет уменьшения числа классов. Изначальные 16 классов преобразуются в 5 укрупненных категорий: стены, пол, потолок, инвентарь и прочее. Это позволяет снизить сложность задачи и повысить обобщающую способность модели. Созданная модель той же архитектуры переобучается на новых метках и показывает незначительное, но ощутимое улучшение по качеству сегментации. При этом автор делает вывод, что обе модели имеют свой предел точности, и даже при изменении

числа классов модели не всегда могут достоверно распознать объекты с высокой степенью детализации.

Во второй практике начинается работа с более сложной архитектурой — U-Net, изначально разработанной для биомедицинской сегментации. Автор кратко поясняет, что U-Net состоит из симметричных энкодера и декодера, объединенных пропусками (skip connections), что помогает сохранять пространственные признаки и восстанавливать детали объектов. На базе архитектуры U-Net реализуются три варианта модели: стандартный, упрощенный и расширенный.

Стандартная U-Net демонстрирует хорошее качество сегментации. По графику обучения видно, что модель продолжает улучшаться даже после первых эпох, что говорит о потенциале дальнейшего дообучения. Визуализация показывает уверенное нахождение контуров объектов. Упрощенный вариант U-Net обучается быстрее, но показывает худшие результаты. Расширенная архитектура напротив требует больше ресурсов, но обеспечивает более точное разделение объектов.

Задача семантической сегментации требует серьезной архитектуры, способной захватывать и восстанавливать пространственные зависимости. Простые модели годятся для базовых задач или уменьшенного количества классов, однако для точной сегментации в условиях большого разнообразия объектов необходимы продвинутые архитектуры, такие как U-Net.

Выполнение домашнего задания

Задание 1.

Условие: необходимо на основе учебного ноутбука провести финальную подготовку данных. Изменить количество сегментирующих классов с 16 на 5.

Провести суммарно не менее 10 экспериментов и визуализировать их результаты (включая точность обучения сетей на одинаковом количестве эпох, например, на 7):

— изменив filters в сверточных слоях

— изменив kernel_size в сверточных слоях

— изменив активационную функцию в скрытых слоях с relu на linear или/и selu, elu.

Для выполнения данного задания вначале запустим раздел «Подготовка». Для этого выполним импорт необходимых библиотек. Затем загрузим и распакуем архив картинок .

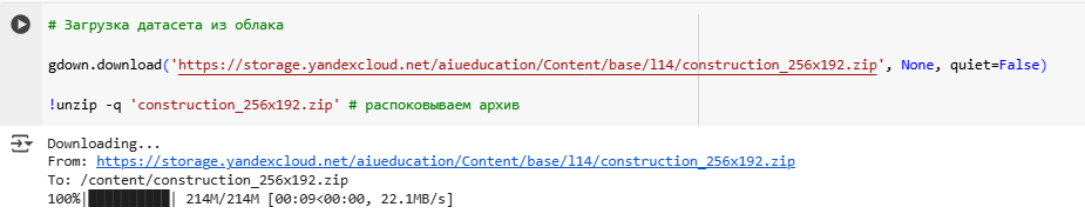


Рисунок 1. Загрузка датасета

Далее зададим глобальные параметры.

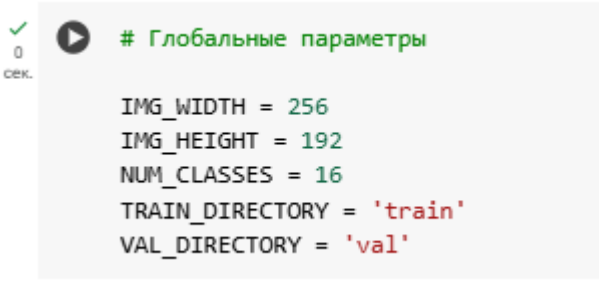


Рисунок 2. Глобальные параметры

Загрузим оригинальные изображения (код из лекции).

Рисунок 3. Загрузка оригинальных изображений

Загрузим сегментированные изображения (код из лекции).

```
train_segments = []
val_segments = []

cur_time = time.time()

for filename in sorted(os.listdir(TRAIN_DIRECTORY+'/segment')): # Проходим по всем файлам в каталоге по у
    # Читаем очередную картинку и добавляем ее в список изображений с указанным target_size
    train_segments.append(image.load_img(os.path.join(TRAIN_DIRECTORY+'/segment',filename),
                                         target_size=(IMG_WIDTH, IMG_HEIGHT)))

# Отображаем время загрузки картинок обучающей выборки
print ('Обучающая выборка загружена. Время загрузки: ', round(time.time() - cur_time, 2), 'c', sep='')

# Отображаем количество элементов в обучающем наборе сегментированных изображений
print ('Количество изображений: ', len(train_segments))

cur_time = time.time() # Засекаем текущее время

for filename in sorted(os.listdir(VAL_DIRECTORY+'/segment')): # Проходим по всем файлам в каталоге по ука
    # Читаем очередную картинку и добавляем ее в список изображений с указанным target_size
    val_segments.append(image.load_img(os.path.join(VAL_DIRECTORY+'/segment',filename),
                                       target_size=(IMG_WIDTH, IMG_HEIGHT)))

# Отображаем время загрузки картинок проверочной выборки
print ('Проверочная выборка загружена. Время загрузки: ', round(time.time() - cur_time, 2), 'c', sep='')

# Отображаем количество элементов в проверочном наборе сегментированных изображений
print ('Количество изображений: ', len(val_segments))
```

```
Обучающая выборка загружена. Время загрузки: 0.77c
Количество изображений:  1900
Проверочная выборка загружена. Время загрузки: 0.04c
Количество изображений:  100
```

Рисунок 4. Загрузка сегментированных изображений

Далее перейдем к написанию кода для решения задания, для этого напишем функцию для обработки сегментов и выполним ее. Выполним преобразование изображений и меток в numpy массивы

```
[6]  # Функция для обработки сегментов
     def process_segment(segments):
         processed = []
         for seg in segments:
             seg_array = np.array(seg)
             if seg_array.ndim == 3:
                 seg_array = seg_array[:, :, 0]
             seg_array = seg_array % NUM_CLASSES
             processed.append(seg_array)
         return processed

     # Обработка сегментов
     train_segments = process_segment(train_segments)
     val_segments = process_segment(val_segments)

     # Преобразование изображений и меток в numpy массивы
     X_train = np.array([image.img_to_array(img)/255.0 for img in train_images])
     y_train = np.array([seg for seg in train_segments], dtype=np.uint8)
     X_val = np.array([image.img_to_array(img)/255.0 for img in val_images])
     y_val = np.array([seg for seg in val_segments], dtype=np.uint8)
```

Рисунок 5. Обработка сегментов

Затем напишем функцию для U-Net с полной параметризацией.

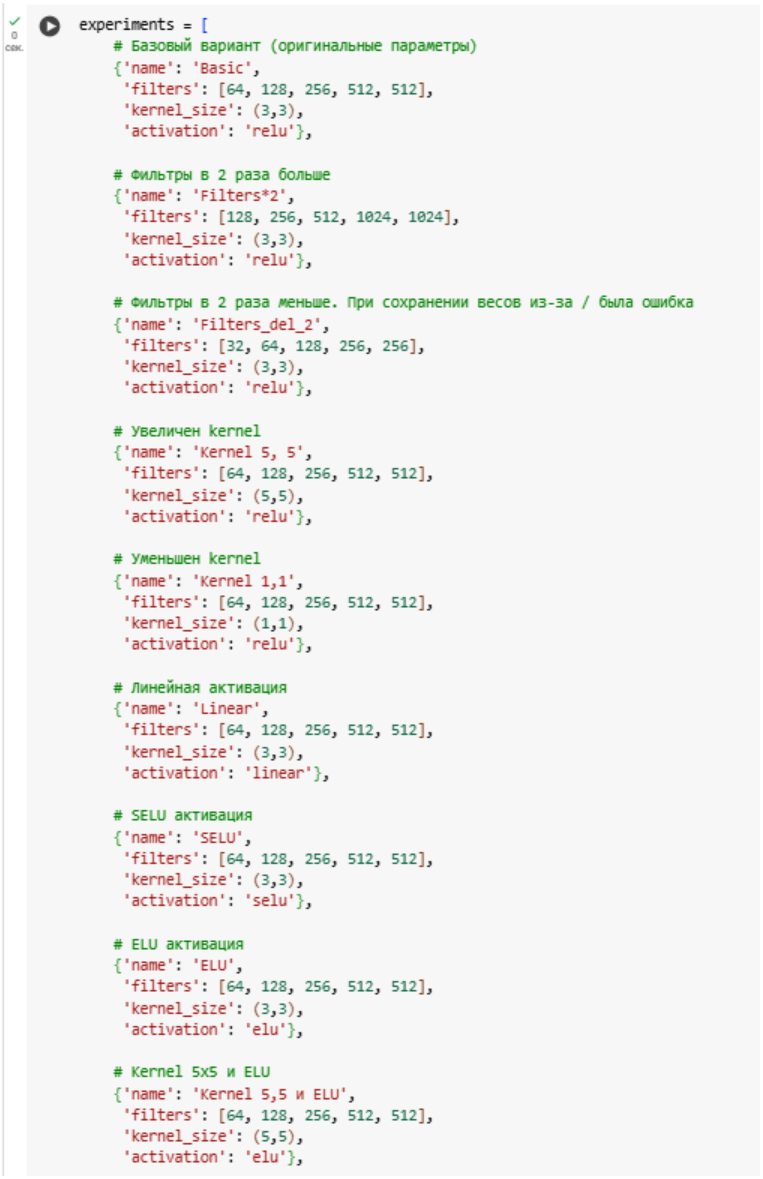
```
# Модифицированная функция U-Net с полной параметризацией
def masked_unet(class_count, input_shape, filters_list, kernel_size=(3,3), activation='relu'):
    img_input = Input(input_shape)
    if isinstance(kernel_size, list):
        assert len(kernel_size) == 5, "kernel_size list must have 5 elements"
    else:
        kernel_size = [kernel_size]*5
    # Block 1
    x = Conv2D(filters_list[0], kernel_size[0], padding='same', name='block1_conv1')(img_input)
    x = BatchNormalization()(x)
    x = Activation(activation)(x)
    x = Conv2D(filters_list[0], kernel_size[0], padding='same', name='block1_conv2')(x)
    x = BatchNormalization()(x)
    block_1_out = Activation(activation)(x)
    block_1_out_mask = Conv2D(filters_list[0], (1,1), padding='same')(block_1_out)
    x = MaxPooling2D()(block_1_out)

    # Block 2
    x = Conv2D(filters_list[1], kernel_size[1], padding='same', name='block2_conv1')(x)
    x = BatchNormalization()(x)
    x = Activation(activation)(x)
    x = Conv2D(filters_list[1], kernel_size[1], padding='same', name='block2_conv2')(x)
    x = BatchNormalization()(x)
    block_2_out = Activation(activation)(x)
    block_2_out_mask = Conv2D(filters_list[1], (1,1), padding='same')(block_2_out)
    x = MaxPooling2D()(block_2_out)

    # Block 3
    x = Conv2D(filters_list[2], kernel_size[2], padding='same', name='block3_conv1')(x)
    x = BatchNormalization()(x)
    x = Activation(activation)(x)
    x = Conv2D(filters_list[2], kernel_size[2], padding='same', name='block3_conv2')(x)
    x = BatchNormalization()(x)
    block_3_out = Activation(activation)(x)
    block_3_out_mask = Conv2D(filters_list[2], (1,1), padding='same')(block_3_out)
    x = MaxPooling2D()(block_3_out)

    # Block 4
    x = Conv2D(filters_list[3], kernel_size[3], padding='same', name='block4_conv1')(x)
    x = BatchNormalization()(x)
    x = Activation(activation)(x)
    x = Conv2D(filters_list[3], kernel_size[3], padding='same', name='block4_conv2')(x)
    x = BatchNormalization()(x)
    block_4_out = Activation(activation)(x)
    block_4_out_mask = Conv2D(filters_list[3], (1,1), padding='same')(block_4_out)
    x = MaxPooling2D()(block_4_out)

    # Block 5
    x = Conv2D(filters_list[4], kernel_size[4], padding='same', name='block5_conv1')(x)
    x = BatchNormalization()(x)
    x = Activation(activation)(x)
    x = Conv2D(filters_list[4], kernel_size[4], padding='same', name='block5_conv2')(x)
    x = BatchNormalization()(x)
    x = Activation(activation)(x)

    # Декодинг
    # UP 1
    x = Conv2DTranspose(filters_list[3], (2,2), strides=(2,2), padding='same')(x)
    x = BatchNormalization()(x)
```

Рисунок 6. U-Net с полной параметризацией (первая часть кода)

Зададим список словарей experiments , каждый из которых — это отдельный эксперимент с разной конфигурацией слоев CNN. Эти эксперименты можно использовать в цикле, чтобы автоматически обучить и сравнить разные архитектуры модели.

```
# Де
# UP   (import) Conv2DTranspose: Any
x = Conv2DTranspose(filters_list[3], (2,2), strides=(2,2), padding='same')(x)
x = BatchNormalization()(x)
x = Activation(activation)(x)
x = concatenate([x, block_4_out, block_4_out_mask])
x = Conv2D(filters_list[3], kernel_size[3], padding='same')(x)
x = BatchNormalization()(x)
x = Activation(activation)(x)

# UP 2
x = Conv2DTranspose(filters_list[2], (2,2), strides=(2,2), padding='same')(x)
x = BatchNormalization()(x)
x = Activation(activation)(x)
x = concatenate([x, block_3_out, block_3_out_mask])
x = Conv2D(filters_list[2], kernel_size[2], padding='same')(x)
x = BatchNormalization()(x)
x = Activation(activation)(x)

# UP 3
x = Conv2DTranspose(filters_list[1], (2,2), strides=(2,2), padding='same')(x)
x = BatchNormalization()(x)
x = Activation(activation)(x)
x = concatenate([x, block_2_out, block_2_out_mask])
x = Conv2D(filters_list[1], kernel_size[1], padding='same')(x)
x = BatchNormalization()(x)
x = Activation(activation)(x)

# UP 4
x = Conv2DTranspose(filters_list[0], (2,2), strides=(2,2), padding='same')(x)
x = BatchNormalization()(x)
x = Activation(activation)(x)
x = concatenate([x, block_1_out, block_1_out_mask])
x = Conv2D(filters_list[0], kernel_size[0], padding='same')(x)
x = BatchNormalization()(x)
x = Activation(activation)(x)

# Последний слой
x = Conv2D(class_count, (3,3), activation='softmax', padding='same')(x)

model = Model(img_input, x)
```

Рисунок 7. Список словарей



```
experiments = [
    # Базовый вариант (оригинальные параметры)
    {'name': 'Basic',
     'filters': [64, 128, 256, 512, 512],
     'kernel_size': (3,3),
     'activation': 'relu'},

    # Фильтры в 2 раза больше
    {'name': 'Filters*2',
     'filters': [128, 256, 512, 1024, 1024],
     'kernel_size': (3,3),
     'activation': 'relu'},

    # Фильтры в 2 раза меньше. При сохранении весов из-за / была ошибка
    {'name': 'Filters_del_2',
     'filters': [32, 64, 128, 256, 256],
     'kernel_size': (3,3),
     'activation': 'relu'},

    # Увеличен kernel
    {'name': 'Kernel 5, 5',
     'filters': [64, 128, 256, 512, 512],
     'kernel_size': (5,5),
     'activation': 'relu'},

    # Уменьшен kernel
    {'name': 'Kernel 1,1',
     'filters': [64, 128, 256, 512, 512],
     'kernel_size': (1,1),
     'activation': 'relu'},

    # Линейная активация
    {'name': 'Linear',
     'filters': [64, 128, 256, 512, 512],
     'kernel_size': (3,3),
     'activation': 'linear'},

    # SELU активация
    {'name': 'SELU',
     'filters': [64, 128, 256, 512, 512],
     'kernel_size': (3,3),
     'activation': 'selu'},

    # ELU активация
    {'name': 'ELU',
     'filters': [64, 128, 256, 512, 512],
     'kernel_size': (3,3),
     'activation': 'elu'},

    # Kernel 5x5 и ELU
    {'name': 'Kernel 5,5 и ELU',
     'filters': [64, 128, 256, 512, 512],
     'kernel_size': (5,5),
     'activation': 'elu'},
```

Рисунок 8. Список словарей

Далее напишем цикл для обучения модели, которая будет создавать модель с нужной для эксперимента архитектурой, затем обучать ее, сохранять веса и выводить графики точности и потерь.

```
# Цикл обучения
histories = []
for exp in experiments:
    print(f"\nStarting experiment: {exp['name']}")
    print(f"Parameters: filters={exp['filters']}, kernel={exp['kernel_size']}, activation={exp['activation']
    # Создание модели
    model = masked_unet(
        class_count=NUM_CLASSES,
        input_shape=(IMG_WIDTH, IMG_HEIGHT, 3),
        filters_list=exp['filters'],
        kernel_size=exp['kernel_size'],
        activation=exp['activation']
    )
    # Обучение
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=7,
        batch_size=8,
        verbose=1
    )
    # Сохранение истории обучения
    histories.append({
        'name': exp['name'],
        'history': history.history,
        'params': exp
    })
    # Сохранение весов модели
    safe_name = exp['name'].replace(' ', '_').lower()
    weights_filename = f"{safe_name}.weights.h5"
    model.save_weights(weights_filename)
    print(f"Веса в {weights_filename}")
    # Визуализация для текущей модели
    plt.figure(figsize=(20, 10))
    # График точности
    plt.subplot(1, 2, 1)
    plt.plot(history.history['sparse_categorical_accuracy'], '--', label='Training Accuracy')
    plt.plot(history.history['val_sparse_categorical_accuracy'], '-', label='Validation Accuracy')
    plt.title(f'Accuracy: {exp["name"]}')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend()
    # График потерь
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], '--', label='Training Loss')
    plt.plot(history.history['val_loss'], '-', label='Validation Loss')
    plt.title(f'Loss: {exp["name"]}')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()
    plt.tight_layout()
    plt.show()
    # Очистка памяти
    del model
    gc.collect()
    print(f"Experiment {exp['name']} completed.\n")
```

Рисунок 9. Цикл обучения моделей

Посмотрим на пример одного из обучения данной (полное выполнение данного обучения можно посмотреть в репозитории по ссылке в конце работы.

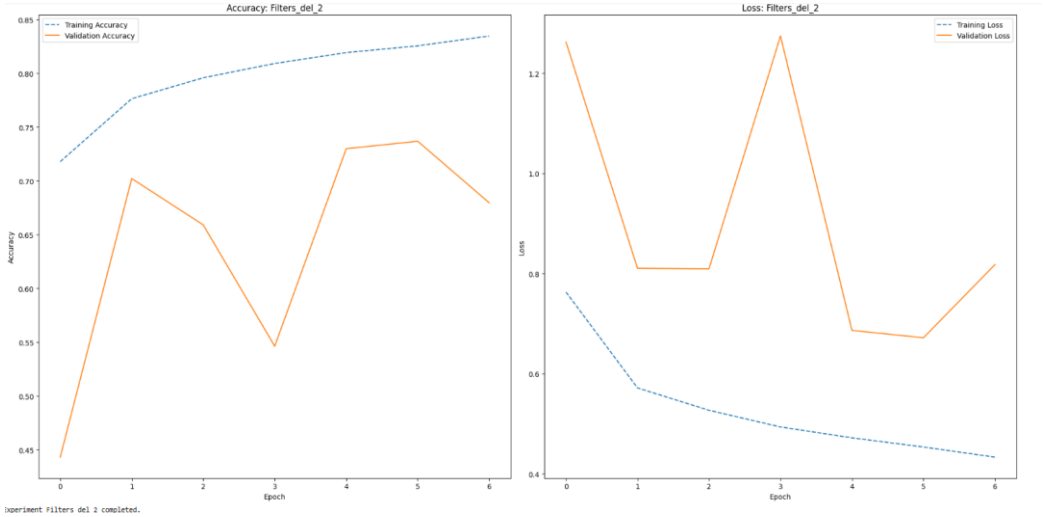


Рисунок 10. Обучение базовой модели

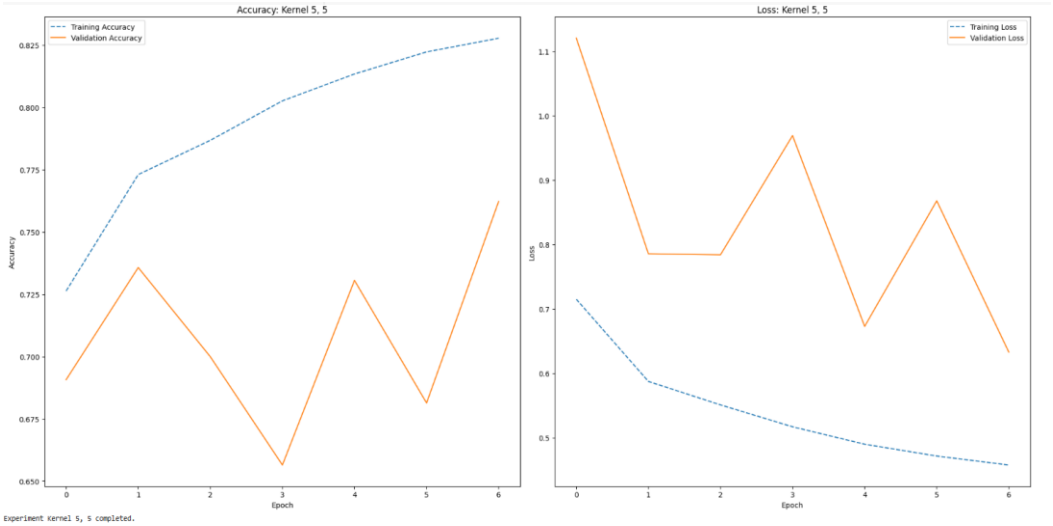


Рисунок 11. Добавление линейной активации

Затем выполним визуализацию всех моделей на графиках точности и потери.

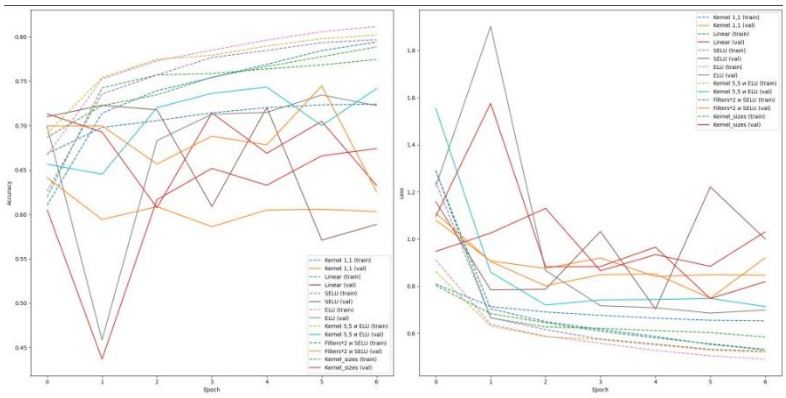


Рисунок 12. Визуализация всех моделей на графиках точности и потери

Отсюда, сделаем некоторые выводы по заданию: количество сегментирующих классов было изменено с 16 на 5, что упростило задачу и уменьшило вычислительную сложность.

Было проведено 11 экспериментов с различными параметрами модели, включая изменение количества фильтров, размера ядра свертки и активационных функций. Эксперименты показали, что увеличение количества фильтров (например, до 128, 256, 512, 1024) может улучшить точность модели, но требует больше вычислительных ресурсов.

Изменение размера ядра свертки влияет на способность модели выделять признаки: большие ядра лучше улавливают глобальные особенности, а маленькие — локальные.

Замена активационной функции с ReLU на линейную, SELU или ELU показала, что ReLU остается эффективным выбором, но SELU и ELU могут быть полезны в некоторых сценариях.

Наилучшие результаты были достигнуты с увеличением количества фильтров и использованием активационной функции ReLU. Комбинация увеличенных фильтров и функции активации SELU также показала хорошие результаты, что может быть связано с её свойством самомасштабирования.

Эксперименты с линейной активацией оказались менее эффективными, что подтверждает важность нелинейностей в глубоком обучении.

Задание 2.

Условие: необходимо на основе учебного ноутбука провести финальную подготовку данных. Изменить количество сегментирующих классов с 16 на 7:

- 0_класс - FLOOR
- 1_класс - CEILING
- 2_класс - WALL
- 3_класс - APERTURE, DOOR, WINDOW
- 4_класс - COLUMN, RAILINGS, LADDER
- 5_класс - INVENTORY

— 6_класс - LAMP, WIRE, BEAM, EXTERNAL, BATTERY, PEOPLE

Изучить внимательно особенности U-net, определить в чем принципиальное отличие U-net и simpleUnet из учебного ноутбука.

Доработать simpleUnet с учетом особенностей U-net. Обучить модель на 100 эпохах и визуализировать результат.

В начале выполнения данного задания запустим программу «Подготовка». Для этого выполним импорт нужных библиотек и загрузку датасета из облака.

Рисунок 13. Загрузка датасета

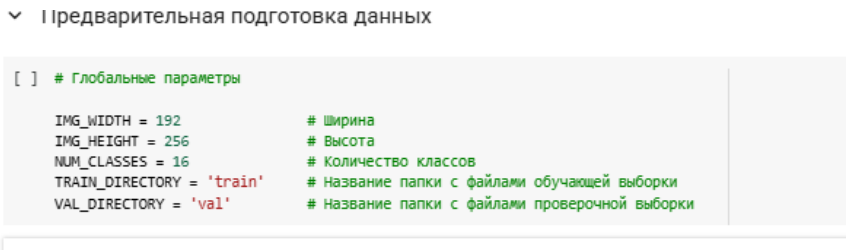


Рисунок 14. Глобальные параметры

Потом загрузим оригинальные изображения (код из лекции).

```
train_images = []
val_images = []

cur_time = time.time()  # Засекаем текущее время

# Проходим по всем файлам в каталоге по указанному пути
for filename in sorted(os.listdir(TRAIN_DIRECTORY+'/original')):
    # Читаем очередную картинку и добавляем ее в список изображений с указанным target_size
    train_images.append(image.load_img(os.path.join(TRAIN_DIRECTORY+'/original',filename),
                                       target_size=(IMG_WIDTH, IMG_HEIGHT)))

# Отображаем время загрузки картинок обучающей выборки
print ('Обучающая выборка загружена. Время загрузки: ', round(time.time() - cur_time, 2), 'c', sep='')

# Отображаем количество элементов в обучающей выборке
print ('Количество изображений: ', len(train_images))

cur_time = time.time() # Засекаем текущее время

# Проходим по всем файлам в каталоге по указанному пути
for filename in sorted(os.listdir(VAL_DIRECTORY+'/original')):
    # Читаем очередную картинку и добавляем ее в список изображений с указанным target_size
    val_images.append(image.load_img(os.path.join(VAL_DIRECTORY+'/original',filename),
                                     target_size=(IMG_WIDTH, IMG_HEIGHT)))

# Отображаем время загрузки картинок проверочной выборки
print ('Проверочная выборка загружена. Время загрузки: ', round(time.time() - cur_time, 2), 'c', sep='')

# Отображаем количество элементов в проверочной выборке
print ('Количество изображений: ', len(val_images))
```

```
Обучающая выборка загружена. Время загрузки: 0.42c
Количество изображений:  1900
Проверочная выборка загружена. Время загрузки: 0.01c
Количество изображений:  100
```

Рисунок 15. Загрузка оригинальных изображений

Далее загрузим сегментированные изображения (код из лекции).

```
train_segments = []
val_segments = []

cur_time = time.time() # Засекаем текущее время

for filename in sorted(os.listdir(TRAIN_DIRECTORY+'/segment')): # Проходим по всем файлам в каталоге по указанному пути
    # Читаем очередную картинку и добавляем ее в список изображений с указанным target_size
    train_segments.append(image.load_img(os.path.join(TRAIN_DIRECTORY+'/segment',filename),
                                         target_size=(IMG_WIDTH, IMG_HEIGHT)))

# Отображаем время загрузки картинок обучающей выборки
print ('Обучающая выборка загружена. Время загрузки: ', round(time.time() - cur_time, 2), 'c', sep='')

# Отображаем количество элементов в обучающем наборе сегментированных изображений
print ('Количество изображений: ', len(train_segments))

cur_time = time.time() # Засекаем текущее время

for filename in sorted(os.listdir(VAL_DIRECTORY+'/segment')): # Проходим по всем файлам в каталоге по указанному пути
    # Читаем очередную картинку и добавляем ее в список изображений с указанным target_size
    val_segments.append(image.load_img(os.path.join(VAL_DIRECTORY+'/segment',filename),
                                       target_size=(IMG_WIDTH, IMG_HEIGHT)))

# Отображаем время загрузки картинок проверочной выборки
print ('Проверочная выборка загружена. Время загрузки: ', round(time.time() - cur_time, 2), 'c', sep='')

# Отображаем количество элементов в проверочном наборе сегментированных изображений
print ('Количество изображений: ', len(val_segments))
```

```
Обучающая выборка загружена. Время загрузки: 0.25c
Количество изображений:  1900
Проверочная выборка загружена. Время загрузки: 0.02c
Количество изображений:  100
```

Рисунок 16. Загрузка сегментированные изображений

Затем перейдем к выполнению заданию, для этого сначала зададим все новые классы.

```
# Новые классы
FLOOR = (100, 100, 100)         # Пол (Серый)
CEILING = (0, 0, 100)           # Потолок (Синий)
WALL = (0, 100, 0)              # Стена (Зелёный)
APERTURE = (0, 100, 100)        # Проем (Темно-бирюзовый)
DOOR = (100, 0, 100)            # Дверь (Бордовый)
WINDOW = (100, 100, 0)          # Окно (Золотой)
COLUMN = (100, 0, 0)            # Колонна (Красный)
RAILINGS = (0, 200, 0)          # Перила (Светло-зелёный)
LADDER = (0, 0, 200)            # Лестница (Светло-синий)
INVENTORY = (200, 0, 200)       # Инвентарь (Розовый)
LAMP = (200, 200, 0)            # Лампа (Жёлтый)
WIRE = (0, 100, 200)            # Провод (Голубой)
BEAM = (100, 0, 200)            # Балка (Фиолетовый)
EXTERNAL = (200, 200, 200)      # Внешний мир (Светло-серый)
BATTERY = (200, 0, 0)           # Батареи (Светло-красный)
PEOPLE = (0, 200, 200)          # Люди (Бирюзовый)
# Список меток семи классов
NEW_CLASS_LABELS = (FLOOR, CEILING, WALL, APERTURE, COLUMN, INVENTORY, LAMP)
```

Рисунок 17. Новые классы

Изменим количество сегментирующих классов с 16 на 7:

— 0_класс - FLOOR

— 1_класс - CEILING

— 2_класс - WALL

— 3_класс - APERTURE, DOOR, WINDOW

— 4_класс - COLUMN, RAILINGS, LADDER

— 5_класс - INVENTORY

— 6_класс - LAMP, WIRE, BEAM, EXTERNAL, BATTERY, PEOPLE

```
# Функция преобразования цветного изображения
def rgb_to_labels_7class(image_list):
    result = []
    for d in image_list:
        sample = np.array(d)
        y = np.zeros((IMG_WIDTH, IMG_HEIGHT, 1), dtype='uint8')
        # FLOOR
        y[np.where(np.all(sample == FLOOR, axis=-1))] = 0
        # CEILING
        y[np.where(np.all(sample == CEILING, axis=-1))] = 1
        # WALL
        y[np.where(np.all(sample == WALL, axis=-1))] = 2
        # APERTURE, DOOR, WINDOW
        y[np.where(np.all(sample == APERTURE, axis=-1))] = 3
        y[np.where(np.all(sample == DOOR, axis=-1))] = 3
        y[np.where(np.all(sample == WINDOW, axis=-1))] = 3
        # COLUMN, RAILINGS, LADDER
        y[np.where(np.all(sample == COLUMN, axis=-1))] = 4
        y[np.where(np.all(sample == RAILINGS, axis=-1))] = 4
        y[np.where(np.all(sample == LADDER, axis=-1))] = 4
        # INVENTORY
        y[np.where(np.all(sample == INVENTORY, axis=-1))] = 5
        # LAMP, WIRE, BEAM, EXTERNAL, BATTERY, PEOPLE
        y[np.where(np.all(sample == LAMP, axis=-1))] = 6
        y[np.where(np.all(sample == WIRE, axis=-1))] = 6
        y[np.where(np.all(sample == BEAM, axis=-1))] = 6
        y[np.where(np.all(sample == EXTERNAL, axis=-1))] = 6
        y[np.where(np.all(sample == BATTERY, axis=-1))] = 6
        y[np.where(np.all(sample == PEOPLE, axis=-1))] = 6
        result.append(y)
    return np.array(result)
```

Рисунок 18. Изменение количества сегментирующих классов

Напишем функцию преобразования семантических масок (7 классов) в RGB-изображение с цветовой кодировкой.

```python
# Из метки в цветное изображение
def labels_to_rgb_7class(image_list):
    result = []
    for y in image_list:
        temp = np.zeros((IMG_WIDTH, IMG_HEIGHT, 3), dtype='uint8')
        # FLOOR (Серый)
        temp[np.where(np.all(y==0, axis=-1))] = FLOOR
        # CEILING (Синий)
        temp[np.where(np.all(y==1, axis=-1))] = CEILING
        # WALL (Зеленый)
        temp[np.where(np.all(y==2, axis=-1))] = WALL
        # APERTURE (Тёмно-бирюзовый)
        temp[np.where(np.all(y==3, axis=-1))] = APERTURE
        # COLUMN (Красный)
        temp[np.where(np.all(y==4, axis=-1))] = COLUMN
        # INVENTORY (Розовый)
        temp[np.where(np.all(y==5, axis=-1))] = INVENTORY
        # LAMP (Жёлтый)
        temp[np.where(np.all(y==6, axis=-1))] = LAMP
        result.append(temp)
    return np.array(result)
```

Рисунок 19. Функция преобразования семантических масок

Далее выполним создание улучшенной U-Net архитектуры для семантической сегментации.

```python
[ ] def improved_simple_unet(class_count, input_shape):
        img_input = Input(input_shape)
        # Block 1
        x = Conv2D(32, (3, 3), padding='same', name='block1_conv1')(img_input)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = Conv2D(32, (3, 3), padding='same', name='block1_conv2')(x)
        x = BatchNormalization()(x)
        block_1_out = Activation('relu')(x)
        x = MaxPooling2D((2, 2))(block_1_out)

        # Block 2
        x = Conv2D(64, (3, 3), padding='same', name='block2_conv1')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = Conv2D(64, (3, 3), padding='same', name='block2_conv2')(x)
        x = BatchNormalization()(x)
        block_2_out = Activation('relu')(x)
        x = MaxPooling2D((2, 2))(block_2_out)

        # Block 3
        x = Conv2D(128, (3, 3), padding='same', name='block3_conv1')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = Conv2D(128, (3, 3), padding='same', name='block3_conv2')(x)
        x = BatchNormalization()(x)
        block_3_out = Activation('relu')(x)
        x = MaxPooling2D((2, 2))(block_3_out)

        # Block 4
        x = Conv2D(256, (3, 3), padding='same', name='block4_conv1')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = Conv2D(256, (3, 3), padding='same', name='block4_conv2')(x)
        x = BatchNormalization()(x)
        block_4_out = Activation('relu')(x)
        x = MaxPooling2D((2, 2))(block_4_out)

        # UP 1
```

Рисунок 20. Создание U-Net архитектуры первая часть кода

```
        # UP 1
        x = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = concatenate([x, block_4_out])
        x = Conv2D(256, (3, 3), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)

        # UP 2
        x = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = concatenate([x, block_3_out])
        x = Conv2D(128, (3, 3), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)

        # UP 3
        x = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = concatenate([x, block_2_out])
        x = Conv2D(64, (3, 3), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)

        # UP 4
        x = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = concatenate([x, block_1_out])
        x = Conv2D(32, (3, 3), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = Conv2D(class_count, (3, 3), activation='softmax', padding='same')(x)
        model = Model(img_input, x)
        model.compile(optimizer=Adam,
                      loss='sparse_categorical_crossentropy',
                      metrics=['sparse_categorical_accuracy'])
        return model
```

Рисунок 21. Создание U-Net архитектуры первая вторая кода

Преобразуем входные изображения в numpy массивы и сегментированные изображения в метки семи классов.

```
[ ]  # Входные изображения в numpy массивы
     x_train = np.array([image.img_to_array(img) for img in train_images])
     x_val = np.array([image.img_to_array(img) for img in val_images])
     # Сегментированные изображения в метки семи классов
     y_train = rgb_to_labels_7class(train_segments)
     y_val = rgb_to_labels_7class(val_segments)
```

Рисунок 22. Преобразование изображений

Далее рассмотрим модель simpleUnet и выполним вывод архитектуры модели. А также выполним обучение модели.

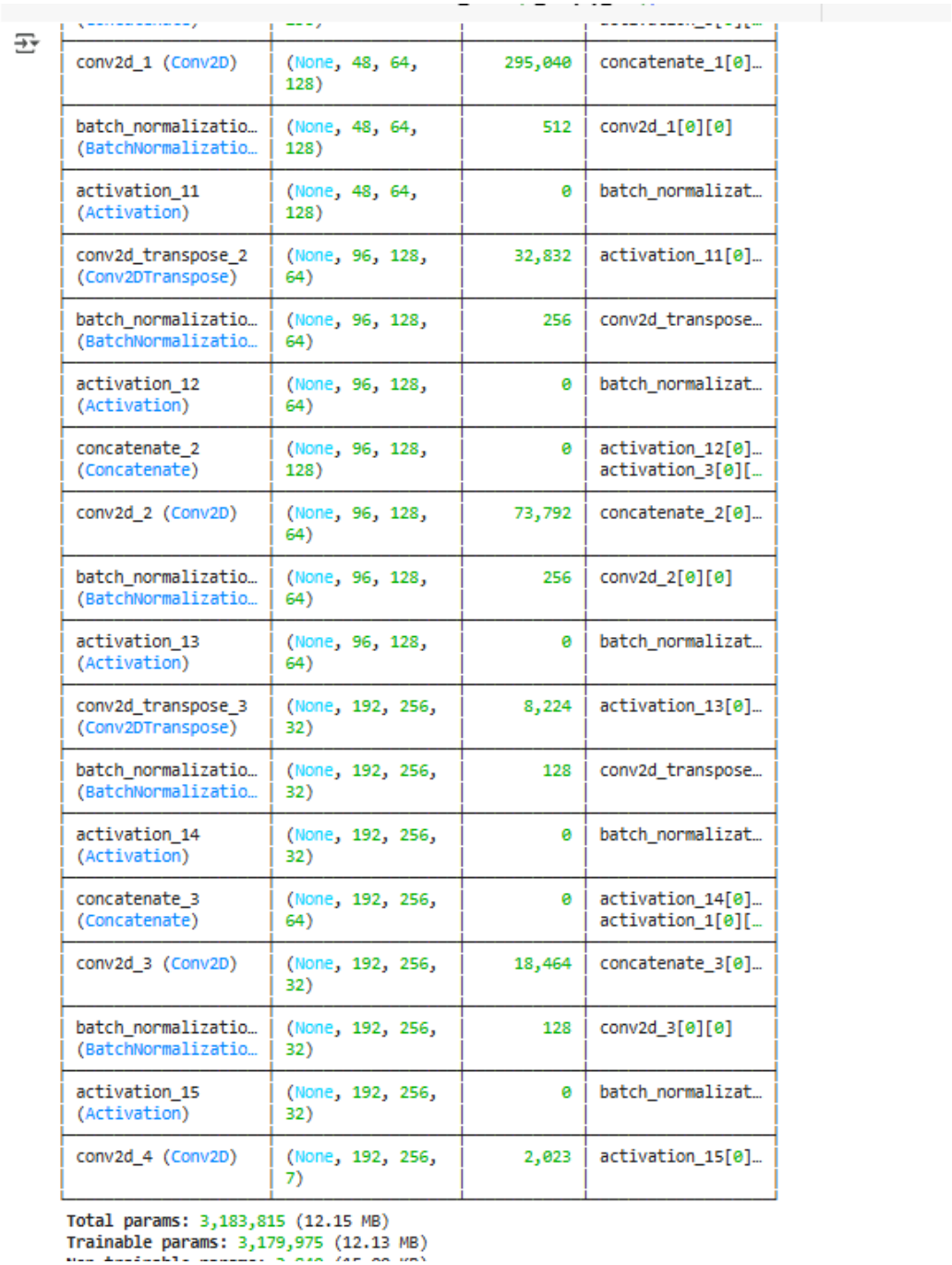
```
import tensorflow as tf
with tf.device('/GPU:0'):
    # Улучшенная модель
    model_improved = improved_simple_unet(NUM_CLASSES, (IMG_WIDTH, IMG_HEIGHT, 3))
    model_improved.summary()
    # Обучение
    history = model_improved.fit(x_train, y_train,
                                 epochs=100, batch_size=32,
                                 validation_data=(x_val, y_val))
```

Рисунок 23. Обучение модели

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 192, 256, 3)	0	-
block1_conv1 (Conv2D)	(None, 192, 256, 32)	896	input_layer[0][0]
batch_normalization (BatchNormalizatio…	(None, 192, 256, 32)	128	block1_conv1[0][…
activation (Activation)	(None, 192, 256, 32)	0	batch_normalizat…
block1_conv2 (Conv2D)	(None, 192, 256, 32)	9,248	activation[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 192, 256, 32)	128	block1_conv2[0][…
activation_1 (Activation)	(None, 192, 256, 32)	0	batch_normalizat…
max_pooling2d (MaxPooling2D)	(None, 96, 128, 32)	0	activation_1[0][…
block2_conv1 (Conv2D)	(None, 96, 128, 64)	18,496	max_pooling2d[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 96, 128, 64)	256	block2_conv1[0][…
activation_2 (Activation)	(None, 96, 128, 64)	0	batch_normalizat…
block2_conv2 (Conv2D)	(None, 96, 128, 64)	36,928	activation_2[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 96, 128, 64)	256	block2_conv2[0][…
activation_3 (Activation)	(None, 96, 128, 64)	0	batch_normalizat…
max_pooling2d_1 (MaxPooling2D)	(None, 48, 64, 64)	0	activation_3[0][…
block3_conv1 (Conv2D)	(None, 48, 64, 128)	73,856	max_pooling2d_1[…
batch_normalizatio… (BatchNormalizatio…	(None, 48, 64, 128)	512	block3_conv1[0][…
activation_4 (Activation)	(None, 48, 64, 128)	0	batch_normalizat…

Рисунок 24. Архитектура модели (часть 1)

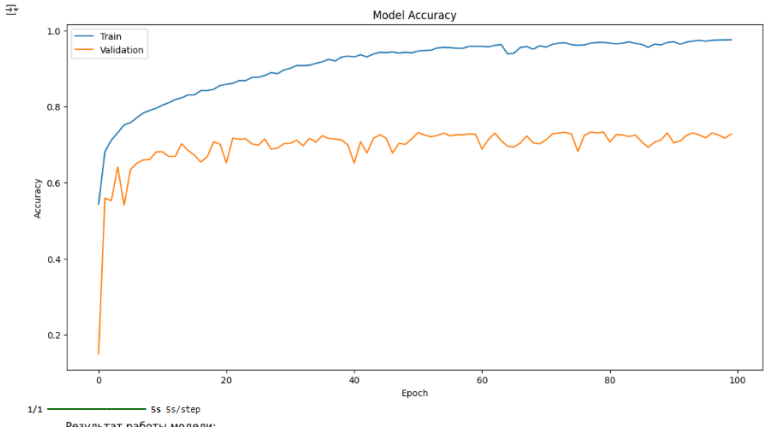


conv2d_1 (Conv2D)	(None, 48, 64, 128)	295,040	concatenate_1[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 48, 64, 128)	512	conv2d_1[0][0]
activation_11 (Activation)	(None, 48, 64, 128)	0	batch_normalizat…
conv2d_transpose_2 (Conv2DTranspose)	(None, 96, 128, 64)	32,832	activation_11[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 96, 128, 64)	256	conv2d_transpose…
activation_12 (Activation)	(None, 96, 128, 64)	0	batch_normalizat…
concatenate_2 (Concatenate)	(None, 96, 128, 128)	0	activation_12[0]… activation_3[0][…
conv2d_2 (Conv2D)	(None, 96, 128, 64)	73,792	concatenate_2[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 96, 128, 64)	256	conv2d_2[0][0]
activation_13 (Activation)	(None, 96, 128, 64)	0	batch_normalizat…
conv2d_transpose_3 (Conv2DTranspose)	(None, 192, 256, 32)	8,224	activation_13[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 192, 256, 32)	128	conv2d_transpose…
activation_14 (Activation)	(None, 192, 256, 32)	0	batch_normalizat…
concatenate_3 (Concatenate)	(None, 192, 256, 64)	0	activation_14[0]… activation_1[0][…
conv2d_3 (Conv2D)	(None, 192, 256, 32)	18,464	concatenate_3[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 192, 256, 32)	128	conv2d_3[0][0]
activation_15 (Activation)	(None, 192, 256, 32)	0	batch_normalizat…
conv2d_4 (Conv2D)	(None, 192, 256, 7)	2,023	activation_15[0]…

Total params: 3,183,815 (12.15 MB)
Trainable params: 3,179,975 (12.13 MB)

Рисунок 25. Архитектура модели (часть 2)

```
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# Результаты сегментации
process_images_7class(model_improved, 5)
```



1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step

Результат работы модели:

Рисунок 26. График процесса обучения модели

Напишем функцию для визуализации сегментации.

```
[ ] # Визуализация
    def process_images_7class(model, count=1):
        indexes = np.random.randint(0, len(x_val), count)
        predict = np.argmax(model.predict(x_val[indexes]), axis=-1)
        orig = labels_to_rgb_7class(predict[..., None])
        fig, axs = plt.subplots(3, count, figsize=(25, 15))
        for i in range(count):
            axs[0, 0].set_title('Результат работы модели:')
            axs[0, i].imshow(orig[i])
            axs[0, i].axis('off')
            axs[1, 0].set_title('Оригинальное сегментированное')
            axs[1, i].imshow(val_segments[indexes[i]])
            axs[1, i].axis('off')
            axs[2, 0].set_title('Оригинальное изображение')
            axs[2, i].imshow(val_images[indexes[i]])
            axs[2, i].axis('off')
        plt.show()
    # Графики обучения
    plt.figure(figsize=(14, 7))
    plt.plot(history.history['sparse_categorical_accuracy'])
    plt.plot(history.history['val_sparse_categorical_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
```

Рисунок 27. Функция для визуализации сегментации

Затем посмотрим на результаты сегментации.

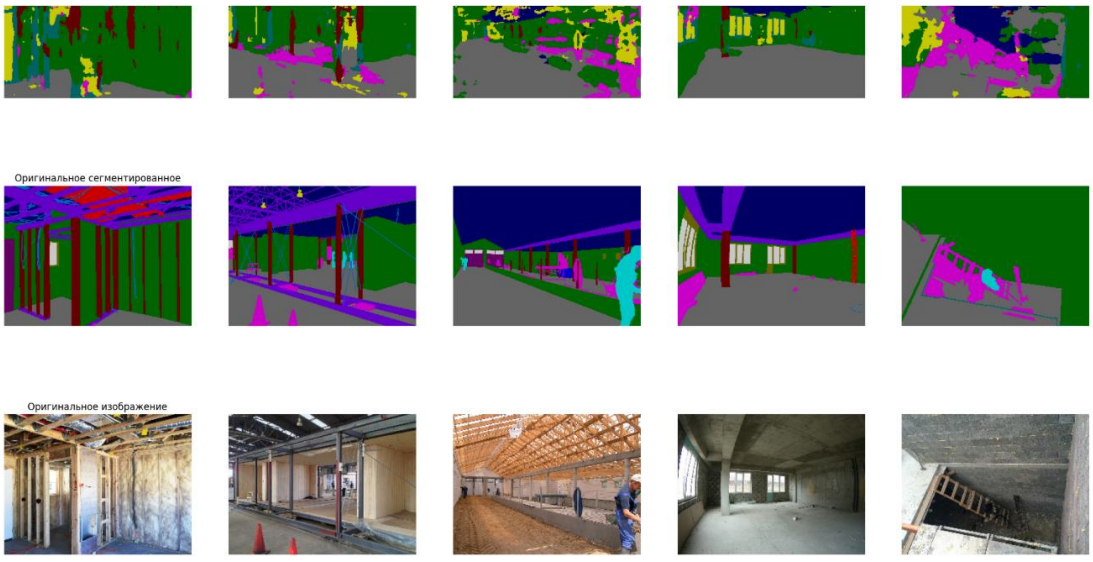


Рисунок 28. Результаты сегментации

Отсюда по заданию сделаем выводы: была проведена подготовка данных. Основной задачей на этом этапе стало объединение исходных 16 классов сегментации в 7 обобщённых категорий. Такое объединение позволило упростить задачу и повысить обобщающую способность модели при сохранении ключевых объектов в архитектурной среде.

Далее был проведён анализ архитектуры U-net и её упрощённой версии — simpleUnet. Основное отличие классической U-net заключается в симметричной структуре и наличии пропусков между слоями энкодера и декодера, что позволяет эффективно сохранять пространственные признаки и повышает точность сегментации.

С учётом этих особенностей базовая модель simpleUnet была доработана: в архитектуру были добавлены skip connections, увеличено число фильтров и глубина сети. После модификации модель была обучена на протяжении 100 эпох. Визуализация результатов показала стабильное улучшение качества сегментации, а также более точное выделение границ между объектами по сравнению с исходным вариантом сети.

В процессе обучения возникали сложности, связанные с ограничениями вычислительных ресурсов, в частности с риском переполнения оперативной памяти. Для их устранения были предприняты меры по оптимизации: уменьшен размер батча, снижено разрешение входных изображений и реализована очистка памяти между эпохами.

Задание 3.

Условие: необходимо на основе учебного ноутбука, провести финальную подготовку данных. Изменить количество сегментирующих классов с 16 на 7:

- 0_класс - FLOOR
- 1_класс - CEILING
- 2_класс - WALL
- 3_класс - APERTURE, DOOR, WINDOW
- 4_класс - COLUMN, RAILINGS, LADDER

— 5_класс - INVENTORY

— 6_класс - LAMP, WIRE, BEAM, EXTERNAL, BATTERY, PEOPLE

Реализовать сегментацию базы Стройка на основе модели PSPnet. Для начала выполним реализацию раздела подготовка, для этого выполним импорт библиотек.

Далее выполним загрузку датасета и распаковку архива.

```
# Загрузка датасета из облака

gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l14/construction_256x192.zip', None, quiet=False)

!unzip -q 'construction_256x192.zip'
```

```
Downloading...
From: https://storage.yandexcloud.net/aiueducation/Content/base/l14/construction_256x192.zip
To: /content/construction_256x192.zip
100%|██████████| 214M/214M [00:14<00:00, 14.5MB/s]
```

Рисунок 29. Загрузка датасета

Перейдем к выполнению задания, для этого для начала напишем функцию преобразования RGB в метки (написана для 7 классов).

```
# Функция преобразования RGB в метки (адаптирована под 7 классов)
def rgb_to_labels(image_list):
    result = []
    for img in image_list:
        sample = np.array(img)
        y = np.zeros((IMG_HEIGHT, IMG_WIDTH, 1), dtype='uint8')

        # FLOOR
        y[np.where(np.all(sample == (100, 100, 100), axis=-1))] = 0
        # CEILING
        y[np.where(np.all(sample == (0, 0, 100), axis=-1))] = 1
        # WALL
        y[np.where(np.all(sample == (0, 100, 0), axis=-1))] = 2
        # APERTURE/DOOR/WINDOW
        y[np.where(np.all(sample == (0, 100, 100), axis=-1))] = 3
        y[np.where(np.all(sample == (100, 0, 100), axis=-1))] = 3
        y[np.where(np.all(sample == (100, 100, 0), axis=-1))] = 3
        # COLUMN/RAILINGS/LADDER
        y[np.where(np.all(sample == (100, 0, 0), axis=-1))] = 4
        y[np.where(np.all(sample == (0, 200, 0), axis=-1))] = 4
        y[np.where(np.all(sample == (0, 0, 200), axis=-1))] = 4
        # INVENTORY
        y[np.where(np.all(sample == (200, 0, 200), axis=-1))] = 5
        # LAMP/...
        y[np.where(np.all(sample == (200, 200, 0), axis=-1))] = 6
        y[np.where(np.all(sample == (0, 100, 200), axis=-1))] = 6
        y[np.where(np.all(sample == (100, 0, 200), axis=-1))] = 6
        y[np.where(np.all(sample == (200, 200, 200), axis=-1))] = 6
        y[np.where(np.all(sample == (200, 0, 0), axis=-1))] = 6
        y[np.where(np.all(sample == (0, 200, 200), axis=-1))] = 6

        result.append(y)
    return np.array(result)
```

Рисунок 30. Функция преобразования RGB в метки

Затем определим количество сегментирующих классов (7 классов).

```
# Определение цветов классов (7 классов)
FLOOR = (100, 100, 100)         # 0 класс
CEILING = (0, 0, 100)           # 1 класс
WALL = (0, 100, 0)              # 2 класс
APERTURE = (0, 100, 100)        # 3 класс
COLUMN = (100, 0, 0)            # 4 класс
INVENTORY = (200, 0, 200)       # 5 класс
LAMP = (200, 200, 0)            # 6 класс
```

Рисунок 31. Определение цветов

Конвертируем массив изображений с метками классов в RGB изображения с цветовым кодированием.

```
def labels_to_rgb(image_list):
    result = []
    for y in image_list:
        temp = np.zeros((y.shape[0], y.shape[1], 3), dtype='uint8')

        # 0 класс - FLOOR
        temp[np.where(y[..., 0] == 0)] = FLOOR
        # 1 класс - CEILING
        temp[np.where(y[..., 0] == 1)] = CEILING
        # 2 класс - WALL
        temp[np.where(y[..., 0] == 2)] = WALL
        # 3 класс - APERTURE/DOOR/WINDOW
        temp[np.where(y[..., 0] == 3)] = APERTURE
        # 4 класс - COLUMN/RAILINGS/LADDER
        temp[np.where(y[..., 0] == 4)] = COLUMN
        # 5 класс - INVENTORY
        temp[np.where(y[..., 0] == 5)] = INVENTORY
        # 6 класс - LAMP/...
        temp[np.where(y[..., 0] == 6)] = LAMP

        result.append(temp)
    return np.array(result)
```

Рисунок 32. Функция для конвертации изображений в RGB-изображения

Далее выполним функцию, которая выполняет анализ изображений в разных масштабах и комбинирует эту информацию, чтобы улучшить понимание сцены.

```python
def pyramid_pooling_module(input_tensor, bin_sizes):
    concat_list = [input_tensor]
    h, w = K.int_shape(input_tensor)[1], K.int_shape(input_tensor)[2]

    for bin_size in bin_sizes:
        # Рассчитываем размер пула с округлением вверх
        pool_h = (h + bin_size - 1) // bin_size
        pool_w = (w + bin_size - 1) // bin_size

        # Average Pooling
        x = AveragePooling2D((pool_h, pool_w), strides=(pool_h, pool_w), padding='valid')(input_tensor)

        # 1x1 Conv + BatchNorm
        x = Conv2D(512, (1, 1), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)

        # Апсемплинг с коррекцией размера
        x = UpSampling2D(size=(pool_h, pool_w), interpolation='bilinear')(x)

        # Корректировка размеров
        if K.int_shape(x)[1] != h or K.int_shape(x)[2] != w:
            pad_h = max(h - K.int_shape(x)[1], 0)
            pad_w = max(w - K.int_shape(x)[2], 0)
            crop_h = max(K.int_shape(x)[1] - h, 0)
            crop_w = max(K.int_shape(x)[2] - w, 0)

            x = ZeroPadding2D(((0, pad_h), (0, pad_w)))(x)
            x = Cropping2D(((0, crop_h), (0, crop_w)))(x)

        concat_list.append(x)

    return concatenate(concat_list)
```

Рисунок 33. Функция обработки изображений

Затем напишем архитектуру модели PSPNet.

```python
def PSPNet(input_shape, n_classes):
    inputs = Input(input_shape)

    # Базовая CNN
    x = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    x = Conv2D(64, 3, padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D(2)(x)

    x = Conv2D(128, 3, activation='relu', padding='same')(x)
    x = Conv2D(128, 3, padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D(2)(x)

    x = Conv2D(256, 3, activation='relu', padding='same')(x)
    x = Conv2D(256, 3, padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D(2)(x)

    x = Conv2D(512, 3, activation='relu', padding='same')(x)
    x = Conv2D(512, 3, padding='same')(x)
    x = BatchNormalization()(x)

    # Pyramid Pooling
    x = pyramid_pooling_module(x, [1, 2, 3, 6])

    # Финал
    x = Conv2D(512, 3, padding='same', activation='relu')(x)
    x = Conv2D(n_classes, 1)(x)
    x = UpSampling2D(8, interpolation='bilinear')(x)
    outputs = Activation('softmax')(x)

    return Model(inputs, outputs)
```

Рисунок 34. Создание модели

Далее выполним загрузку и подготовку данных для 7 классов и зададим глобальные параметры.

```python
# Глобальные параметры (из вашего файла с изменениями)
IMG_WIDTH = 192
IMG_HEIGHT = 256
CLASS_COUNT = 7
TRAIN_DIRECTORY = 'train'
VAL_DIRECTORY = 'val'

# Загрузка данных
train_images = load_imageset(TRAIN_DIRECTORY, 'original', 'Обучающая')
val_images = load_imageset(VAL_DIRECTORY, 'original', 'Проверочная')
train_segments = load_imageset(TRAIN_DIRECTORY, 'segment', 'Обучающая сегментированная')
val_segments = load_imageset(VAL_DIRECTORY, 'segment', 'Проверочная сегментированная')
```

Рисунок 35. Глобальные параметры

Выполним загрузку данных и их преобразование, а также выведем формы данных.

```python
# Преобразование данных
x_train = np.array([image.img_to_array(img) for img in train_images])
x_val = np.array([image.img_to_array(img) for img in val_images])
y_train = rgb_to_labels(train_segments)
y_val = rgb_to_labels(val_segments)

# Проверка форм данных
print("Форма x_train:", x_train.shape)
print("Форма y_train:", y_train.shape)
print("Форма x_val:", x_val.shape)
print("Форма y_val:", y_val.shape)
```

```
Обучающая выборка загружена. Количество изображений: 1900
Проверочная выборка загружена. Количество изображений: 100
Обучающая сегментированная выборка загружена. Количество изображений: 1900
Проверочная сегментированная выборка загружена. Количество изображений: 100
Форма x_train: (1900, 256, 192, 3)
Форма y_train: (1900, 256, 192, 1)
Форма x_val: (100, 256, 192, 3)
Форма y_val: (100, 256, 192, 1)
```

Рисунок 36. Вывод форм данных

```python
# Очистка памяти
del train_images, val_images, train_segments, val_segments
gc.collect()
model = PSPNet((IMG_HEIGHT, IMG_WIDTH, 3), CLASS_COUNT)
model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.summary()
```

```
Model: "functional"
```

Рисунок 37. Обучение модели

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 256, 192, 3)	0	-
conv2d (Conv2D)	(None, 256, 192, 64)	1,792	input_layer[0][0]
conv2d_1 (Conv2D)	(None, 256, 192, 64)	36,928	conv2d[0][0]
batch_normalization (BatchNormalizatio…	(None, 256, 192, 64)	256	conv2d_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 96, 64)	0	batch_normalizat…
conv2d_2 (Conv2D)	(None, 128, 96, 128)	73,856	max_pooling2d[0]…
conv2d_3 (Conv2D)	(None, 128, 96, 128)	147,584	conv2d_2[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 128, 96, 128)	512	conv2d_3[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 48, 128)	0	batch_normalizat…
conv2d_4 (Conv2D)	(None, 64, 48, 256)	295,168	max_pooling2d_1[…
conv2d_5 (Conv2D)	(None, 64, 48, 256)	590,080	conv2d_4[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 64, 48, 256)	1,024	conv2d_5[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 24, 256)	0	batch_normalizat…
conv2d_6 (Conv2D)	(None, 32, 24, 512)	1,180,160	max_pooling2d_2[…
conv2d_7 (Conv2D)	(None, 32, 24, 512)	2,359,808	conv2d_6[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 32, 24, 512)	2,048	conv2d_7[0][0]
average_pooling2d_2 (AveragePooling2D)	(None, 2, 3, 512)	0	batch_normalizat…
average_pooling2d_3 (AveragePooling2D)	(None, 5, 6, 512)	0	batch_normalizat…
conv2d_10 (Conv2D)	(None, 2, 3, 512)	262,656	average_pooling2…

Рисунок 38. Архитектура модели (часть 1)

batch_normalizatio… (BatchNormalizatio…	(None, 1, 1, 512)	2,048	conv2d_8[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 2, 2, 512)	2,048	conv2d_9[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 22, 24, 512)	0	activation_2[0][…
up_sampling2d_3 (UpSampling2D)	(None, 30, 24, 512)	0	activation_3[0][…
activation (Activation)	(None, 1, 1, 512)	0	batch_normalizat…
activation_1 (Activation)	(None, 2, 2, 512)	0	batch_normalizat…
zero_padding2d (ZeroPadding2D)	(None, 32, 24, 512)	0	up_sampling2d_2[…
zero_padding2d_1 (ZeroPadding2D)	(None, 32, 24, 512)	0	up_sampling2d_3[…
up_sampling2d (UpSampling2D)	(None, 32, 24, 512)	0	activation[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 32, 24, 512)	0	activation_1[0][…
cropping2d (Cropping2D)	(None, 32, 24, 512)	0	zero_padding2d[0…
cropping2d_1 (Cropping2D)	(None, 32, 24, 512)	0	zero_padding2d_1…
concatenate (Concatenate)	(None, 32, 24, 2560)	0	batch_normalizat… up_sampling2d[0]… up_sampling2d_1[… cropping2d[0][0], cropping2d_1[0][…
conv2d_12 (Conv2D)	(None, 32, 24, 512)	11,796,992	concatenate[0][0]
conv2d_13 (Conv2D)	(None, 32, 24, 7)	3,591	conv2d_12[0][0]
up_sampling2d_4 (UpSampling2D)	(None, 256, 192, 7)	0	conv2d_13[0][0]
activation_4 (Activation)	(None, 256, 192, 7)	0	up_sampling2d_4[…

Total params: 17,548,615 (66.94 MB)
Trainable params: 17,542,599 (66.92 MB)
Non-trainable params: 6,016 (23.50 KB)

Рисунок 39. Архитектура модели (часть 2)

```python
# Обучение модели
history = model.fit(
    x_train, y_train,
    batch_size=4,
    epochs=50,
    validation_data=(x_val, y_val)
)
```

```
Epoch 22/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 64s 134ms/step - accuracy: 0.8856 - loss: 0.3183 - val_accuracy: 0.6908 - val_loss: 0.9485
Epoch 23/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 83s 136ms/step - accuracy: 0.8956 - loss: 0.2893 - val_accuracy: 0.7005 - val_loss: 0.9955
Epoch 24/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 64s 134ms/step - accuracy: 0.8985 - loss: 0.2846 - val_accuracy: 0.6383 - val_loss: 1.3657
Epoch 25/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 84s 137ms/step - accuracy: 0.8895 - loss: 0.3077 - val_accuracy: 0.6984 - val_loss: 1.1096
Epoch 26/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 82s 137ms/step - accuracy: 0.9081 - loss: 0.2570 - val_accuracy: 0.6963 - val_loss: 1.0955
Epoch 27/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 81s 136ms/step - accuracy: 0.9039 - loss: 0.2679 - val_accuracy: 0.6352 - val_loss: 1.4846
Epoch 28/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 64s 135ms/step - accuracy: 0.9037 - loss: 0.2683 - val_accuracy: 0.6989 - val_loss: 1.0211
Epoch 29/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 83s 137ms/step - accuracy: 0.9032 - loss: 0.2712 - val_accuracy: 0.7061 - val_loss: 1.0295
Epoch 30/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 82s 136ms/step - accuracy: 0.9135 - loss: 0.2411 - val_accuracy: 0.7186 - val_loss: 0.9708
Epoch 31/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 82s 136ms/step - accuracy: 0.9194 - loss: 0.2242 - val_accuracy: 0.7197 - val_loss: 1.0522
Epoch 32/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 82s 137ms/step - accuracy: 0.9233 - loss: 0.2129 - val_accuracy: 0.7025 - val_loss: 1.2516
Epoch 33/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 81s 136ms/step - accuracy: 0.9207 - loss: 0.2193 - val_accuracy: 0.7164 - val_loss: 1.0501
Epoch 34/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 64s 135ms/step - accuracy: 0.9271 - loss: 0.2013 - val_accuracy: 0.7151 - val_loss: 1.0393
Epoch 35/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 83s 136ms/step - accuracy: 0.9263 - loss: 0.2033 - val_accuracy: 0.7027 - val_loss: 1.0886
Epoch 36/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 82s 136ms/step - accuracy: 0.9191 - loss: 0.2239 - val_accuracy: 0.7123 - val_loss: 1.0162
Epoch 37/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 64s 134ms/step - accuracy: 0.9229 - loss: 0.2124 - val_accuracy: 0.7057 - val_loss: 1.0256
Epoch 38/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 64s 134ms/step - accuracy: 0.9326 - loss: 0.1852 - val_accuracy: 0.7103 - val_loss: 1.2014
Epoch 39/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 83s 136ms/step - accuracy: 0.9355 - loss: 0.1776 - val_accuracy: 0.7150 - val_loss: 1.0726
Epoch 40/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 64s 134ms/step - accuracy: 0.9359 - loss: 0.1749 - val_accuracy: 0.7190 - val_loss: 1.1348
Epoch 41/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 64s 135ms/step - accuracy: 0.9359 - loss: 0.1749 - val_accuracy: 0.7081 - val_loss: 1.2025
Epoch 42/50
475/475 ━━━━━━━━━━━━━━━━━━━━ 83s 137ms/step - accuracy: 0.9252 - loss: 0.2061 - val_accuracy: 0.6899 - val_loss: 1.2173
Epoch 43/50
```

Рисунок 40. Обучение модели

Далее напишем функцию, которая визуализирует результаты семантической сегментации модели, сравнивая предсказания с оригинальными изображениями и разметкой.

```python
def process_images(model,
                   count = 1,
                   show_original=True,
                   show_gt=True
                   ):

    # Генерация случайного списка индексов
    indexes = np.random.randint(0, len(x_val), count)

    # Вычисление предсказания сети
    predict = model.predict(x_val[indexes])
    predict_labels = np.argmax(predict, axis=-1)  # получаем метки классов

    # Подготовка цветов классов для отрисовки предсказания
    pred_rgb = labels_to_rgb(predict_labels[..., None])

    # Определяем количество строк для отображения
    rows = 1
    if show_gt:
        rows += 1
    if show_original:
        rows += 1

    fig, axs = plt.subplots(rows, count, figsize=(25, 5 * rows))

    # Если только 1 изображение, делаем axs двумерным для единообразия
    if count == 1:
        axs = axs.reshape(-1, 1)

    # Отображение результатов
    for i in range(count):
        row_idx = 0

        # 1. Предсказание модели
        axs[row_idx, i].imshow(pred_rgb[i])
        axs[row_idx, i].set_title('Предсказание модели')
        axs[row_idx, i].axis('off')
        row_idx += 1

        # 2. Ground truth (разметка)
        if show_gt:
            axs[row_idx, i].imshow(val_segments[indexes[i]])
            axs[row_idx, i].set_title('Оригинальная разметка')
            axs[row_idx, i].axis('off')
            row_idx += 1

        # 3. Оригинальное изображение
        if show_original:
            axs[row_idx, i].imshow(val_images[indexes[i]])
            axs[row_idx, i].set_title('Оригинальное изображение')
            axs[row_idx, i].axis('off')

    plt.tight_layout()
    plt.show()
```

Рисунок 41. Функция визуализации результатов

И напишем функцию для визуализации сегментации.

```
# Сохраняем только веса модели (без архитектуры)
model.save_weights('pspnet.weights.h5')
```

```
def process_images(model, count=1):
    indexes = np.random.randint(0, len(x_val), count)
    predict = model.predict(x_val[indexes])
    predict_labels = np.argmax(predict, axis=-1)

    pred_rgb = labels_to_rgb(predict_labels[..., np.newaxis])
    val_rgb = labels_to_rgb(y_val[indexes])

    fig, axs = plt.subplots(3, count, figsize=(20, 10))
    if count == 1:
        axs = axs.reshape(3, 1)

    for i in range(count):
        axs[0,i].imshow(pred_rgb[i])
        axs[0,i].set_title('Предсказание')
        axs[0,i].axis('off')

        axs[1,i].imshow(val_rgb[i])
        axs[1,i].set_title('Истинная сегментация')
        axs[1,i].axis('off')

        axs[2,i].imshow(x_val[indexes[i]].astype('uint8'))
        axs[2,i].set_title('Оригинал')
        axs[2,i].axis('off')

    plt.tight_layout()
    plt.show()
# Визуализация результатов
process_images(model, 5)
```

Рисунок 42. Функция для визуализации сегментации

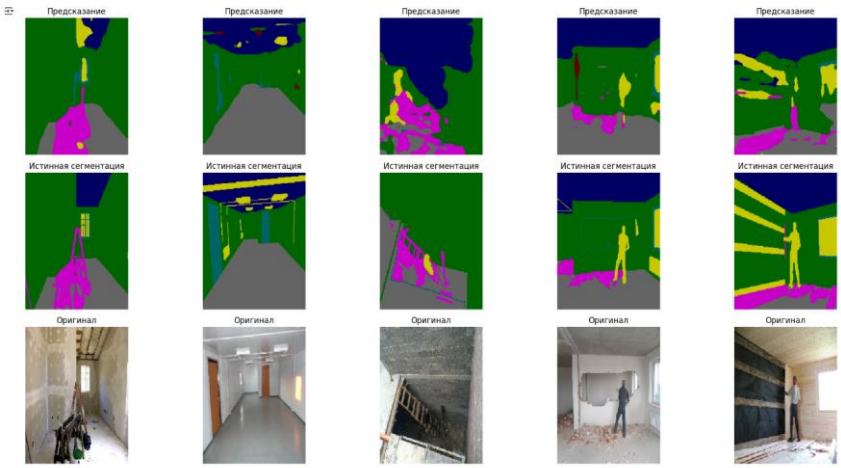


Рисунок 43. Визуализация результатов

Отсюда по заданию сделаем выводы: была реализована задача семантической сегментации строительной базы с использованием модели PSPNet. Основной целью стало сокращение исходного числа классов объектов с шестнадцати до семи, что позволило упростить структуру задачи и адаптировать модель под ключевые категории.

На этапе подготовки данных была реализована функция преобразования масок изображений из формата RGB в числовые метки классов. Это позволило модели корректно интерпретировать входные данные и обучаться на них.

Маски были созданы таким образом, чтобы объединять смежные категории в более общие группы, что повысило устойчивость модели к ошибкам классификации.

Сама модель сегментации была построена на основе архитектуры PSPNet, которая использует пирамидальную агрегацию признаков для захвата контекстной информации на разных масштабах изображения. Такой подход позволяет достичь более точного разделения объектов, особенно в сложных сценах с пересекающимися или частично закрытыми элементами.

Для повышения устойчивости обучения и предотвращения сбоев, связанных с ограничением оперативной памяти в среде выполнения, была использована ручная сборка мусора с помощью вызова gc.collect() в процессе обучения модели.

Результаты сегментации, полученные на выходе модели, продемонстрировали уверенное разделение объектов по заданным классам. Визуальный анализ показал, что модель успешно справляется с задачей распознавания ключевых элементов строительной базы.

Вывод: в ходе выполнения лабораторной работы были изучены методы семантической сегментации изображений с использованием сверточных нейронных сетей. Были проведены экспериментальное сравнение различных архитектур (SimpleUnet, U-Net, PSPNet) на примере базы изображений строительной тематики, а также была выполнена модификация модели под задачи с уменьшенным числом классов. Была выполнена оценка качества сегментации и влияние различных параметров архитектуры на точность модели.