Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
на тему

МЕТОДЫ И АЛГОРИТМЫ РАСПОЗНАВАНИЯ ДОРОЖНЫХ ОБЪЕКТОВ С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ НА ПРИМЕРЕ ПЕШЕХОДНЫХ ПЕРЕХОДОВ И ДОРОЖНЫХ ЗНАКОВ

Студент Д.С. Иванчук

Руководитель А.В. Леченко

Минск 2021

РЕФЕРАТ

Объектом исследования является архитектура нейронных сетей Unet и её модификация Unet-X, используемые для семантической сегментации изображений.

Цель работы — составление размеченного корпуса данных для задачи семантической сегментации изображений для трёх классов: пешеходных переходов, дорожных знаков и нулевого класса; построение и обучение моделей Unet и Unet-X на нём; исследование полученных результатов и сравнительный анализ эффективности моделей.

Результаты работы могут использоваться исследователями, заинтересованными в решении задачи семантической сегментации. Работу можно брать за основу для дальнейшего изучения архитектур нейронных сетей, а также их оптимизаций.

ПЕРЕЧЕНЬ ОПРЕДЕЛЕНИЙ И СОКРАЩЕНИЙ

В настоящей пояснительной записке применяются следующие определения, сокращения.

Машинное обучение (англ. machine learning, ML) — класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач. Для построения таких методов используются средства математической статистики, численных методов, математического анализа, методов оптимизации, теории вероятностей, теории графов, различные техники работы с данными в цифровой форме.

Иску́сственный интелле́кт (ИИ; англ. artificial intelligence, AI) — свойство интеллектуальных систем выполнять творческие функции, которые традиционно считаются прерогативой человека (не следует путать с искусственным сознанием, ИС); наука и технология создания интеллектуальных машин, особенно интеллектуальных компьютерных программ.

True Positive результат — это результат, при котором модель правильно предсказывает наличие класса. Точно так же true negative результат — это результат, когда модель правильно предсказывает отсутствие класса. False postive результат — это результат, при котором модель неверно предсказывает наличие класса. A false negative результат — это результат, когда модель неверно предсказывает отсутствие класса.

Функция активации нейрона — функция, которая определяет выходное значение нейрона в зависимости от результата взвешенной суммы входов и порогового значения.

Метод обратного распространения ошибки (англ. *backpropagation*) — метод вычисления градиента, который используется при обновлении весов многослойного перцептрона.

Сигмоид — это гладкая монотонная возрастающая нелинейная функция, имеющая форму буквы «S», применяющаяся в качестве нелинейности слоя нейронной сети, а так же для образования бинарного признака.

ReLU (Rectified Linear Unit) — функция активации, определяемая как положительная часть ее аргументов.
Свёрточная нейронная сеть (англ. convolutional neural network, CNN) — специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание образов, входит в состав технологий глубокого обучения (англ. deep

learning).

Стохастический градиентный спуск (англ. Stochastic gradient descent, SGD) — это итерационный метод для оптимизации целевой функции с подходящими свойствами гладкости (например, дифференцируемость или субдифференцируемость). Его можно расценивать как стохастическую аппроксимацию оптимизации методом градиентного спуска, поскольку он заменяет реальный градиент, вычисленный из полного набора данных его оценкой, вычисленной из случайно выбранного подмножества данных. Это сокращает задействованные вычислительные ресурсы и помогает достичь более высокой скорости итераций в обмен на более низкую скорость сходимости. Особенно большой эффект достигается в приложениях связанных с обработкой больших данных.

Jupyter-ноутбук — это среда разработки, где сразу можно видеть результат выполнения кода и его отдельных фрагментов. В такой среде разработки можно, например, написать функцию и сразу проверить её работу, без запуска программы целиком.

ВВЕДЕНИЕ

Машинное обучение без преувеличения является одним из самых перспективных направлений в области информационных технологий. Сфера применения данной отрасли с каждым годом становится всё шире, а последние достижения будоражат умы людей во всём мире. Искусственный интеллект, основанный на ML алгоритмах всё чаще превосходит людей в играх, в управлении автомобилем, распознавании изображений и многом другом.

Благодаря машинному обучению программисты стали способны решать задачи, ранее казавшиеся невозможными. Ведь не всегда человек заранее знает точный алгоритм для решения задачи, а с машинным обучением он обрёл инструмент, позволяющий делегировать поиск решения компьютеру, указав ему лишь определённый инструментарий.

Существуют несколько основных задач алгоритмов машинного обучения. В данной работе мы рассмотрим задачу обучения с учителем (Supervised learning), заключающуюся принудительнov обучении системы с помощью примеров «стимул-реакция». Человек заранее определяет желаемый результат работы алгоритма. Алгоритму остаётся проанализировать исходное признаковое пространство и настроить себя на оптимальный результат.

Технология машинного обучения на основе анализа данных берёт начало в 1950 году, когда начали разрабатывать первые программы для игры в шашки. За прошедшие десятилетия общий принцип не изменился. Зато благодаря взрывному росту вычислительных мощностей компьютеров многократно усложнились закономерности и прогнозы, создаваемые ими, и расширился круг проблем и задач, решаемых с использованием машинного обучения.

Методы машинного обучения (например, deep learning), активно применяются при разработке беспилотных автомобилей для локализации объектов вокруг, предсказания их поведения, планирования собственных действий.

Глубокое обучение (Deep learning) — это направление в области Искусственного Интеллекта (Artificial Intelligence) и Машинного Обучения (Machine Learning), основанное на применении глубоких нейросетевых моделей, способным формировать в процессе обучения многоуровневые, иерархические представления об окружающем мире, в которых понятия более высокого уровня определяются на основе понятий более низкого уровня.

Одной из типовых задач глубокого обучения можно назвать задачу семантической сегментации.

В данной работе будет рассмотрена подмножество задачи семантической сегментации — много классовая семантическая

сегментация. Будут рассмотрены отличия решения задачи много классовой сегметанции от решения задачи двухклассовой семантической сегментации объектов. Будет проведен анализ архитектуры нейронных сетей Unet и Unet-X для сегментации дорожных объектов на изображениях на примере пешеходных переходов и дорожных знаков.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Семантическая сегментация как частная задача компьютерного зрения

Сегметация является ярким представителем задачи компьютерного зрения. Главное её отличие от ближайшего родственника (классификации) — прогнозирование не одной метки принадлежности к классу по изображению, а трансформация исходного изображения в множество каналов, определяющих принадлежность отдельных пикселей к определённому классу.

Сегментация находит применение в самых разных областях реальной жизни, таких как:

- Медицина: обнаружение аномалий в автоматическом режиме помогает людям в процессе их работы. Статистический анализ полученных результатов уменьшает время обработки данных.

- Навигация беспилотных средств: на изображениях, получаемых от датчиков подобных систем, определются объекты, которые в дальнейшем необходимы для принятия решения.

- Обработка документов: современные средства оцифровки не могут обойтись без неё. Кроме этого, анализ документов, основанных на данном подходе позволяют в разы ускорить документооборот, что немаловажно.

В данной работе будет рассмотрена задача трёх классовой семантической сегментации пешеходных переходов на изображении, то есть классификация каждого пикселя изображения на 3 класса: пешеходный переход, дорожный знак, фон.

Не существует единого решения задачи семантической сегментации. В работе будет рассмотрен нейросетевой подход, основанный на использовании нейронной сети UNet и её модификации.

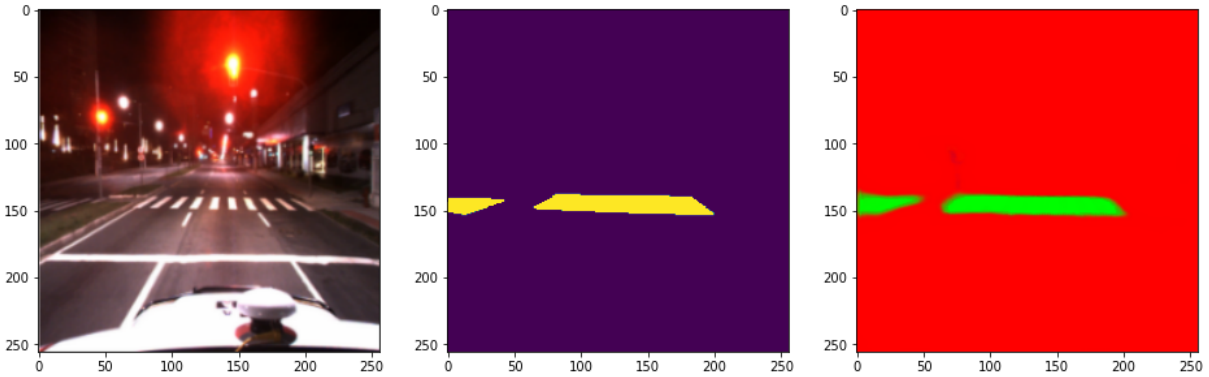
1.2 Математическая постановка задачи

На вход алгоритм получет тензор **H**x**W**x**C**, где **H, W** — высота, ширина изображения, **C —** число каналов изображения (при использовании RGB схемы **C** = 3). Результатом является тензор **H**x**W**x**N**, где **N** — число классов, необходимое определить. В нашем случае используется 3 класса:

0. Нулевой класс — всё кроме пешеходных переходов и дорожных знаков.
1. Пешеходный переход.
2. Дорожный знак.

Результатом работы алгоритма является матрица разметки, где

каждый пиксель имеет метку от 0 до 2, в соответствии с классификацией, данной выше.



Пример семантической сегментации пешеходного перехода (центральный элемент — маска, размеченная человеком, слева нейросетью)

Процесс машинного обучения нельзя построить без адекватной оценки качества,потому рассмотрим несколько метрик, используемых для определения точности:

1) Accuracy

Интуитивно понятной, очевидной и почти неиспользуемой метрикой является accuracy — доля правильных ответов алгоритма:

$$Accuracy = \frac{\sum_{Class_i} TruePositive_{Class_i}}{TotalObjects}$$

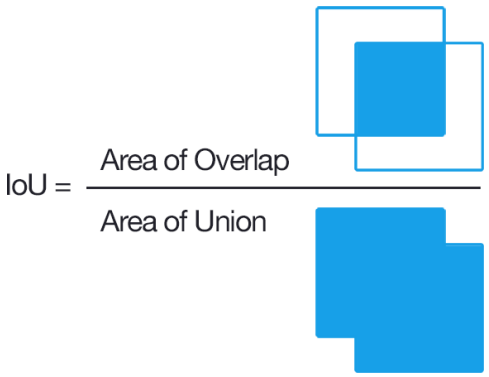
Эта метрика бесполезна в задачах с несбалансированными классами. Действительно, если т.к. нас интересует общая точность, модель может просто предсказывать всё как самый большой класс и будет права в рамках данной метрики.

2) Коэффициент Жаккара (IoU)

Коэффициент Жаккара, представляет собой статистику, чуть более точную с точки зрения нашей задачи. Измерение формально определяется как доля True Positive среди всего пространства исходов, которое отнесло примеры к заданному классу (True Positive + ошибки 1-ого и 2-ого рода). Математическое представление индекса записывается как:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Эта метрика также называется метрикой Intersection-Over-Union (IoU), тизуальное представление метрики IoU представлено ниже.



Визуализация коэффициента Жаккара

1.3 Общая постановка задачи

Основная задача данной работы состоит в исследовании поведения и эффективности таких моделей как Unet и Unet-X. Другой, побочной, но не менее важной задачей является попытка решения задачи наилучшим путём.

В ходе работы также должны быть выявлены векторы дальнейшего развития в рамках рассмотренных подходов.

2 ОБЗОР НЕЙРОСЕТЕВОГО ПОДХОДА К РЕШЕНИЮ ЗАДАЧИ СЕМАНТИЧЕСКОЙ СЕГМЕНТАЦИИ

2.1 Сверточная нейронная сеть

Нейро́нная сеть (искусственная нейронная сеть, ИНС) — математическая модель, а также её реализация, названная так по той причине, что создавалась она на основе человеческих знаний о нервной системе животных. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы. Первой такой попыткой были нейронные сети У. Маккалока и У. Питтса.

Первоначально в основе ИНС лежало понятие искуственного нейрона (аналог нашей нейронной клетки). Нейроном можно назвать математический объект, способный получать сигналы от других нейронов и посылающий на основе полученных сигналов новый сигнал в другие нейроны. Связывая множество подобных нейронов вместе мы получим сложную сеть, имеющую мощный аппроксимационный потенциал.

Как и любой другой алгоритм машинного обучения, нейронные сети способны обучаться. Процесс обучения ИНС заключается в настройке внутренних параметров искусственных нейронов определяющих их дальнейшее поведение. Именно большое количество своеобразных степеней свободы в нейронных сетях делает их крайне мощным аппроксимационным алгоритмом, способным выполнять различные задачи. Так, к примеру было доказано, что полносвязная нейронная сеть с любой нелинейностью способны аппроксимировать любую непрерывную функцию с наперёд заданной точностью.

Глубинная нейронная сеть (ГНС, англ. DNN — Deep neural network) — это искусственная нейронная сеть (ИНС) с несколькими слоями между входным и выходным слоями. Свою популярность они обрели по той причине, что в отличие от однослойных нейронных сетей им требуется гораздо меньшее число обучающих параметров, для решения одинаковой задачи.

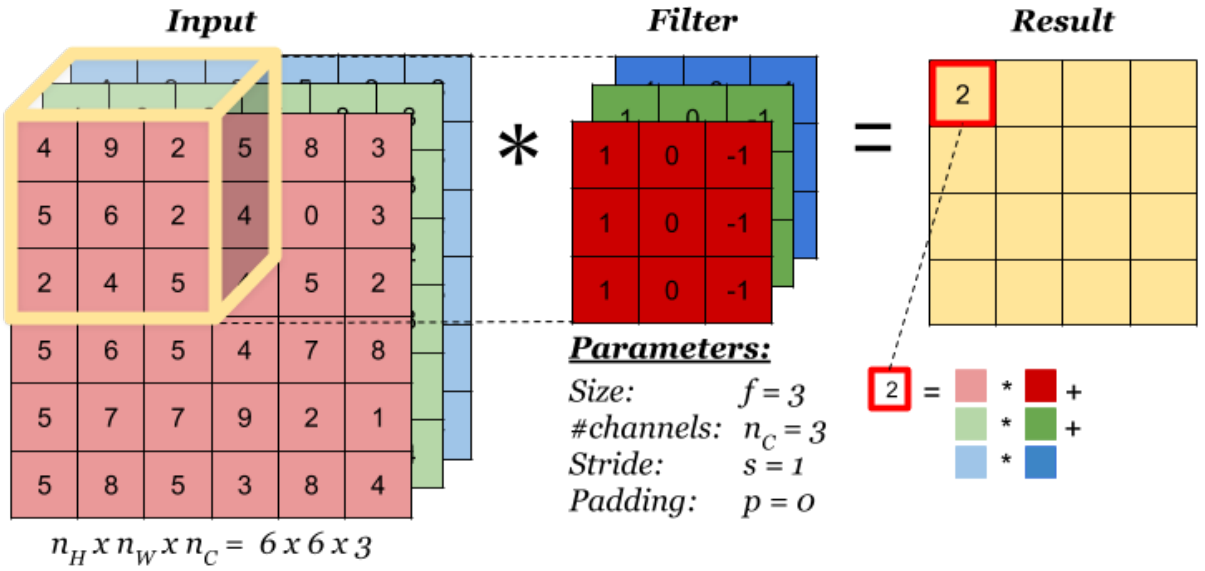
ГНС могут моделировать сложные нелинейные отношения. По своей сути каждый слой трансформирует пространство предыдущего наилучшим образом, без необходимости запоминания других сложных зависимостей.

Свёрточная нейронная сеть (англ. convolutional neural network, CNN) — специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание образов, входит в состав технологий глубокого обучения (англ. deep learning). Использует некоторые особенности зрительной коры,

в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток. Свёрточные нейронные сети отличаются от полносвязных сетей тем, что её слои делают аггрегации по локальной группе нейронов в пределах так называемого «окна», позволяя при меньшем числе параметров лучше определять взаимодействие соседних нейронов. Именно это свойство и определило популярность CNN в задачах компьютерного зрения, звука и задачах, где данное допущение является оправданным.

Работа сверточной нейронной сети обычно интерпретируется как переход от конкретных особенностей изображения к более абстрактным деталям, и далее к ещё более абстрактным деталям вплоть до выделения понятий высокого уровня. При этом сеть самонастраивается и вырабатывает сама необходимую иерархию абстрактных признаков, фильтруя маловажные детали и выделяя существенное.

2.2 Архитектура и принцип работы CNN



Типовой слой сверточной нейронной сети

В обычной полносвязной нейронной сети, каждый нейрон связан со всеми нейронами предыдущего слоя, причём каждая связь имеет свой независимый весовой коэффициент. В свёрточной нейронной сети *операция свёртки* оперирует ограниченной матрицей весов небольшого размера (ядром), которую «двигают» по всему обрабатываемому слою, формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. Её интерпретируют как

графическое кодирование какого-либо признака, например, наличие наклонной линии под определённым углом. Каждое отдельное ядро переводит свой слой в карту признаков, определяемым им.

Операция субдискретизации (англ. *subsampling*, англ. *pooling*, также переводимая как «операция подвыборки» или операция объединения), выполняет уменьшение размерности сформированных карт признаков. В данной архитектуре сети считается, что информация о факте наличия искомого признака важнее точного знания его координат, поэтому из нескольких соседних нейронов карты признаков выбирается максимальный и принимается за один нейрон уплотнённой карты признаков меньшей размерности. За счёт данной операции, помимо ускорения дальнейших вычислений, сеть становится более инвариантной к масштабу входного изображения.

Рассмотрим типовую структуру свёрточной нейронной сети более подробно. Сеть состоит из большого количества слоёв. После начального слоя (входного изображения) сигнал проходит серию свёрточных слоёв, в которых чередуется собственно свёртка и субдискретизация (пулинг). Чередование слоёв позволяет составлять «карты признаков» из карт признаков, на каждом следующем слое карта уменьшается в размере, но увеличивается количество каналов. На практике это означает способность распознавания сложных иерархий признаков. Обычно после прохождения нескольких слоёв карта признаков вырождается в вектор или даже скаляр, но таких карт признаков становятся сотни. На выходе свёрточных слоёв сети дополнительно устанавливают несколько слоёв полносвязной нейронной сети (перцептрон), на вход которому подаются оконечные карты признаков.

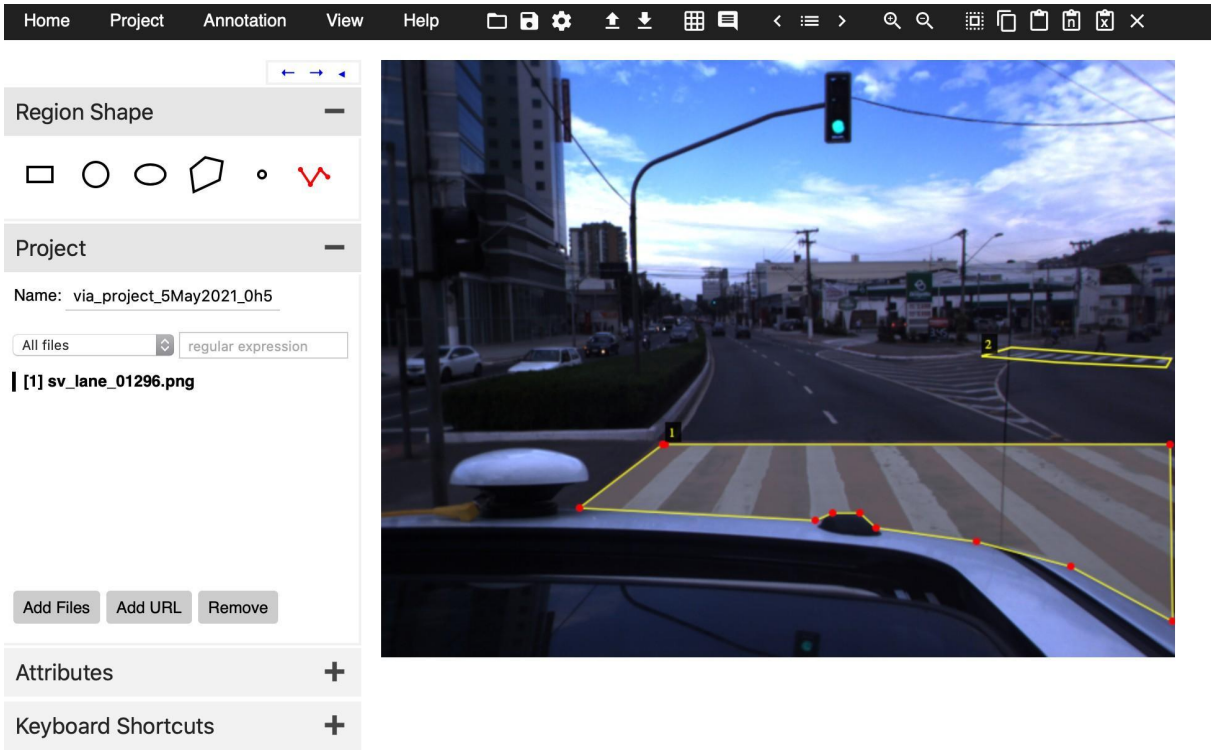
3 ПРОВЕДЕНИЕ ИССЛЕДОВАНИЯ

3.1 Подготовка данных

Так как для семантической сегментации мы планируем использовать нейронную сеть, нам необходимо обучить ее на размеченных данных. Разметка для семантической сегментации на 3 класса представляет собой составление трёх отдельных пикслеьных карт, определяющих принадлежность пикселя к отдельному классу. Процесс получения разметки на заданных изображениях называется аннотированием.

В качестве исходных данных был выбран датасет из открытого GitHub-репозитория Эдуарда Адасько. В сумме было выбрано 3083 фотографий. Полученный датасет имеет ярко выраженный дисбаланс классов, т. к. отношение числей по трём классам составляет приблизительно 100:10:1, что накладывает некоторые трудности на дальнейший процесс обучения.

Для аннотирования изображений был использован инструмент VGG Image Annotator. Он позволяет выделить область искомого объекта на множестве фотографий и получить файл в формате JSON для всех масок, содержимое которого в дальнейшем можно преобразовать в необходимый формат.

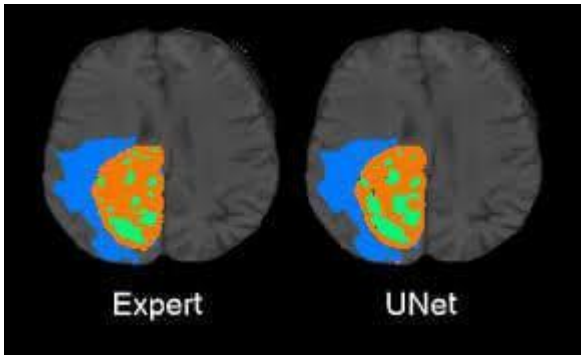


После генерации JSON-файла необходимо преобразовать полученные данные в изображения масок. Для этого был написан скрипт на языке Python (Приложение А — Генерация масок из формата JSON),

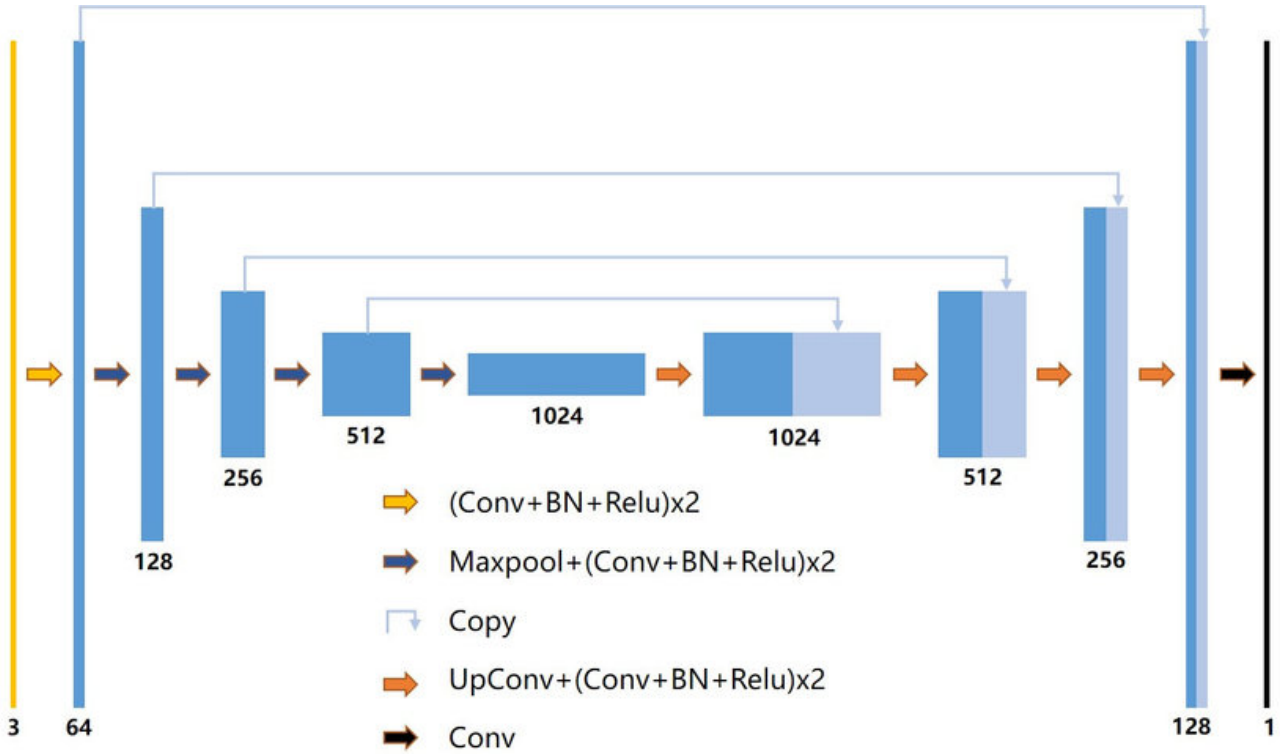
3.2 Обзор и построение выбранной архитектуры и её модификации

3.2.1 UNet

U-Net считается одной из стандартных архитектур CNN для задач сегментации изображений. Архитектура состоит из стягивающего пути для захвата контекста и симметричного расширяющегося пути, который позволяет осуществить точную локализацию.

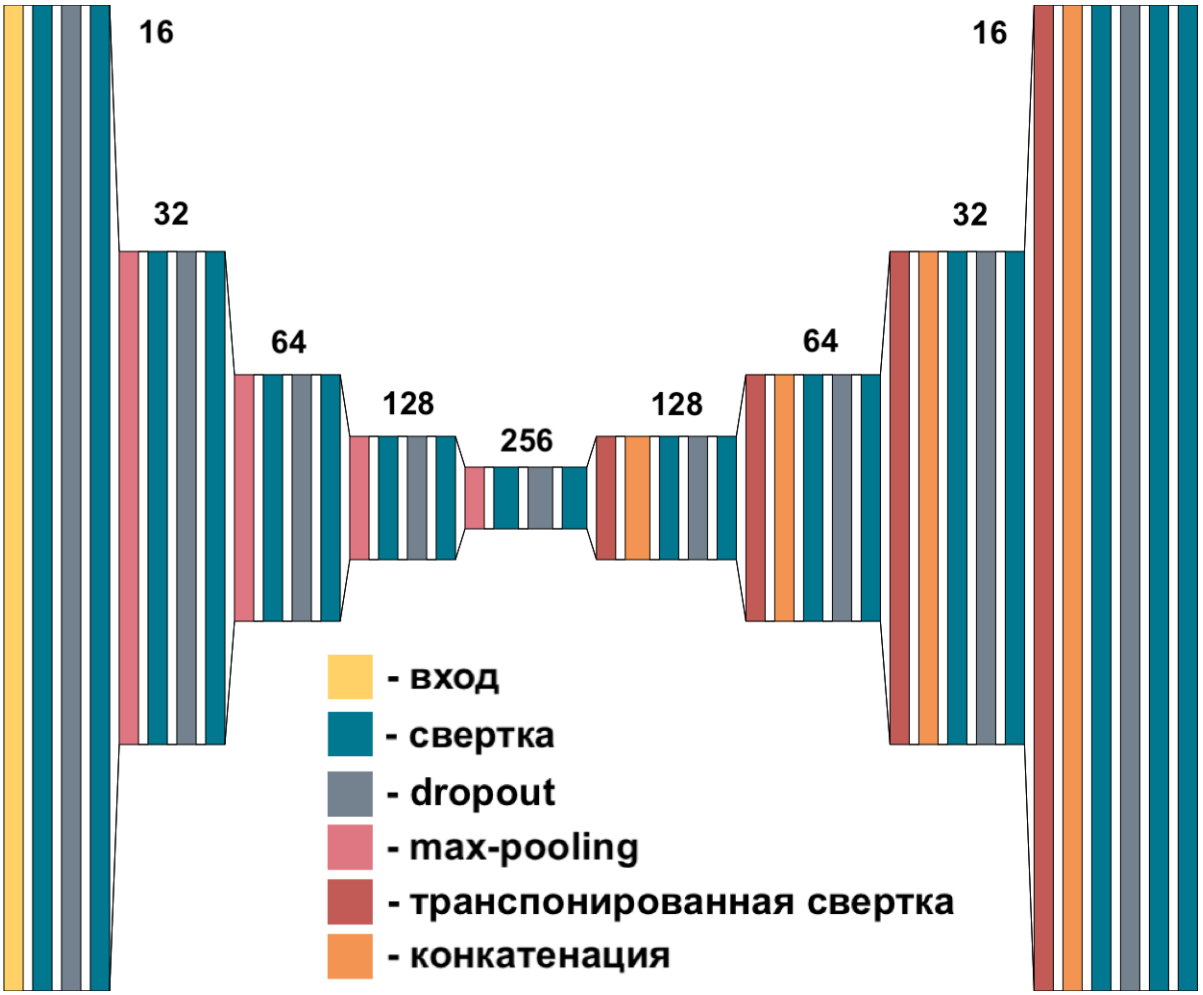


Пример сегментации мозга Unet моделью



Пример сети Unet.

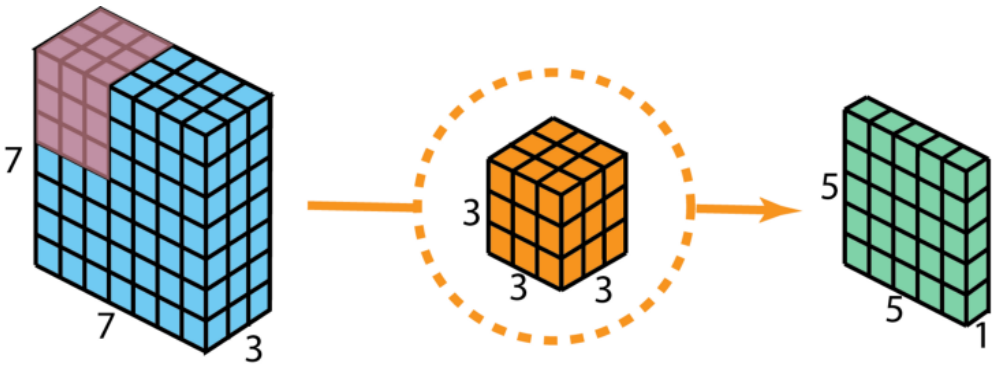
Архитектура U-net. Ключевой особенностью данной архитектуры является наличие сквозных соединений между уровнями вложенности (residual layers). Она позволяет сохранять ранее вычисленную информацию, что потенциально повышает точность модели при развертывании сети.



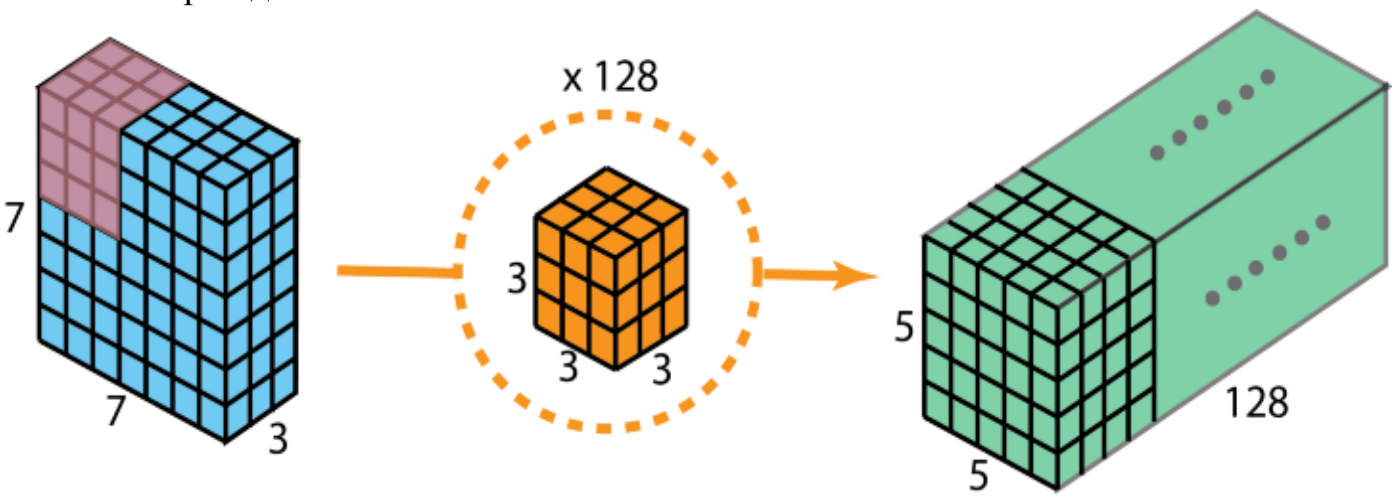
Визуализация полученной модели UNet

3.2.2 Разделимые по глубине свёртки

Разделимые по глубине свертки состоят из двух этапов: глубинных сверток и сверток Стандартная двумерная свертка для создания вывода с 1 слоем с использованием 1 фильтра.

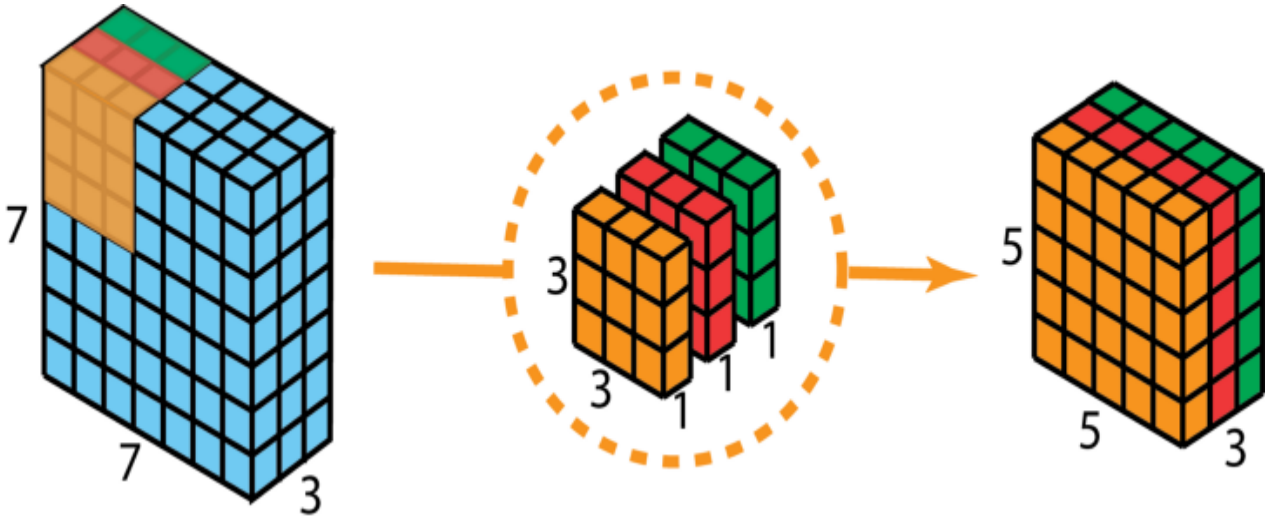


Обычно между в свёрточном слое применяется множество фильтров. Если для хранения одного ядра неоюходимо держать в памяти **W*H*D** значений весов, то при использовании, к примеру, 128 фильтров число обучаемых значений в 128 раз. В случае очень больших и глубоких сетей, данное число параметров может сделать процесс обучения и предсказания неимоверно долгим и вычислительно сложным.

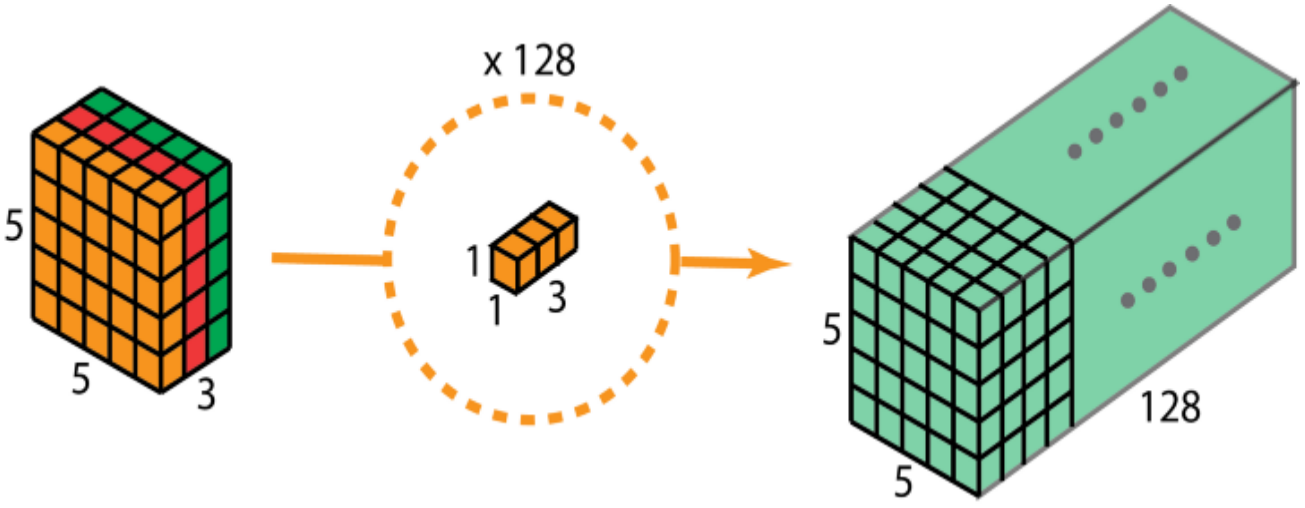


Стандартная двумерная свертка для создания вывода со 128 слоями с использованием 128 фильтров.

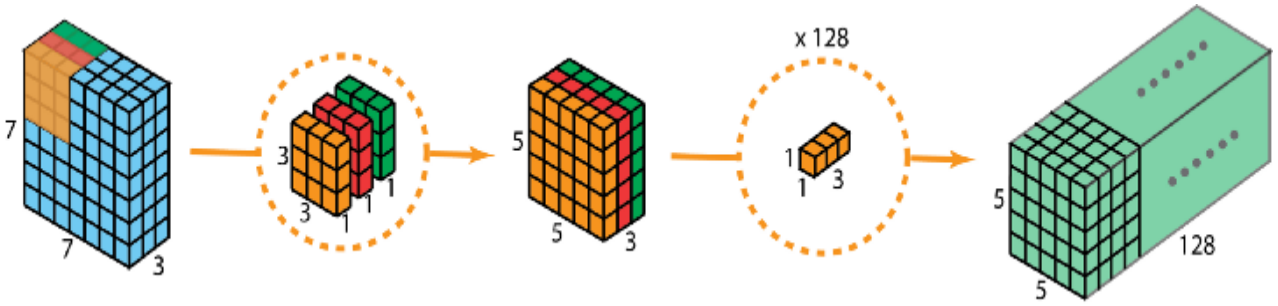
Разделяемая по глубине свертка, способна добиться того же преобразования с меньшим число параметров.



Идея здесь следующая: применять фиксированные поканальные свёркти, а преобразованные каналы разворачивать набором одномерных фильтров. Результат преобразования является более связным и имеет меньшее число степеней свободы, однако имеет много меньшую константу при множителе числа выходных каналов.



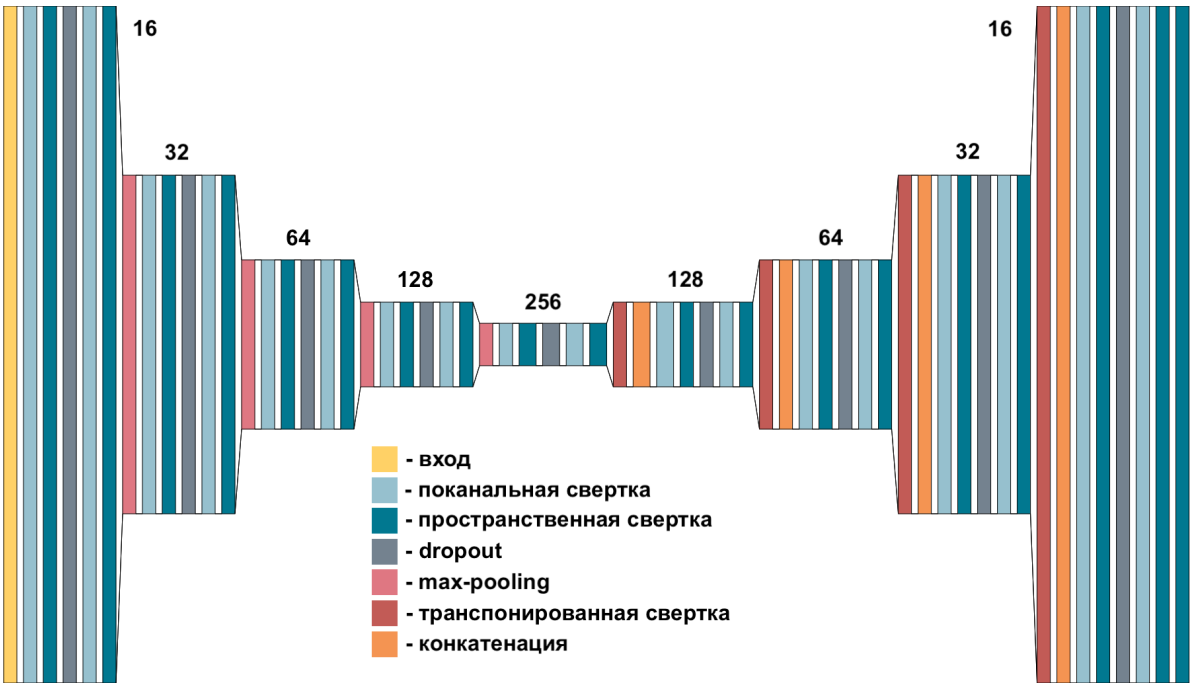
Общий процесс разделимой по глубине свертки показан на рисунке ниже.



Общий процесс разделимой по глубине свертки.

3.2.3 Оптимизация сети UNet — Unet-X

Архитектуру Unet можно дополнительно оптимизировать. Для этого используем идею поканальной раздельной свертки. Она позволит уменьшить количество обучаемых параметров, при этом увеличив количество слоев нейронной сети. Каждый сверточный слой при этом разобьется на два слоя: поканальный сверточный слой и пространственный сверточный слой. Визуализация полученной архитектуры представлена на рисунке ниже.



Оптимизированная сеть UNet

3.3 Процесс обучения

Для написания практической части выбор пал на фреймворк tensorflow на языке Python. Причина тому — гибкая конфигурация моделей и компиляция модели перед началом работы, обеспечивающая высокую производительность.

Так как архитектура нейронной сети была заранее определена, единственное, чем я мог влиять на процесс и результат обучения – функцией потерь, т. к. они определяют конкретные штрафы, используемые для настройки модели. Для основы я решил обучить модель на популярной в многоклассовой классификации связке (softmax + categorical cross entropy)

$$Softmax = \frac{\exp p_i}{\sum_{p_j} \exp p_j}$$

$$CrossEntropy_{Categorical} = -\sum_{i=1}^{N} y_i \log p_i$$

В качестве оптимизатора был выбран хорошо зарекомендовавший себя алгоритм Adam. Причины тому:

1) Частичная инвариативность гиперпараметров относительно задач,
2) Использование инерционной эвристики,
3) Низкое влияние накапливающего эффекта.

Для проверки обобщающей способности модели был составлен валидационный корпус данных размером 10% от первончачльного набора. Для оценки качества модели я использовал следующие метрики:

1) IoU (Для 2ух классов объектов и общая)
2) Точность (Accuracy)

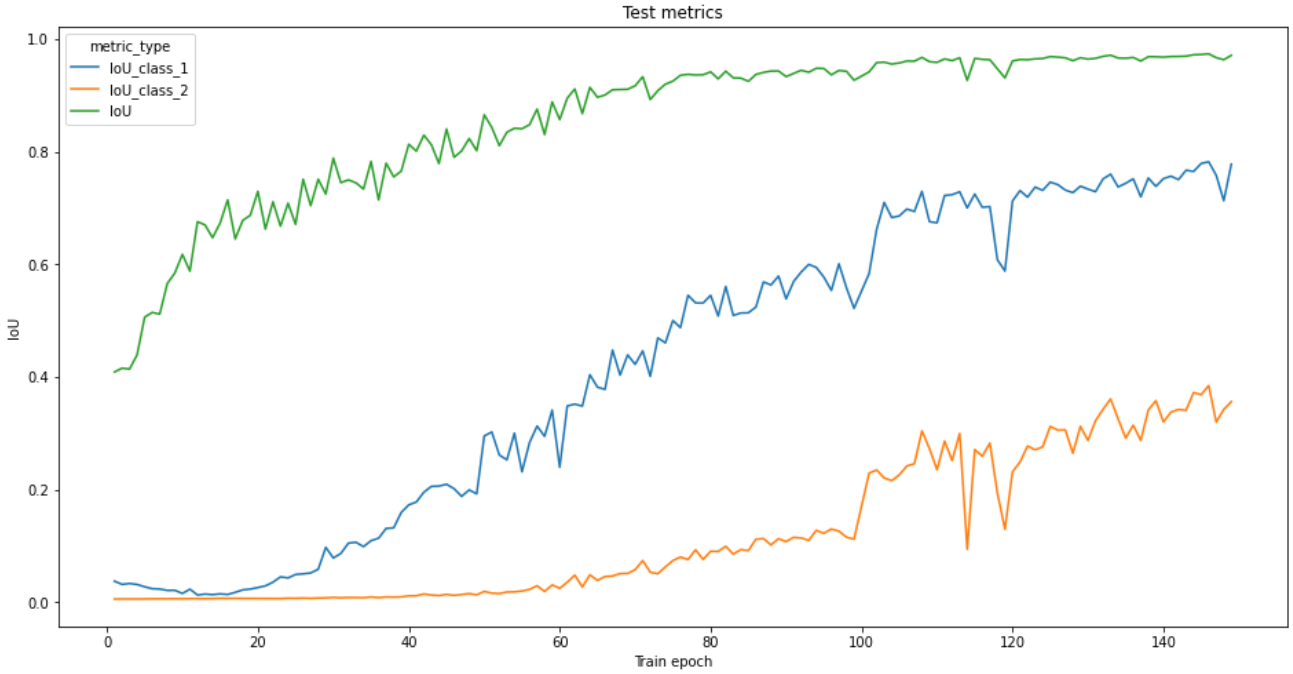
3.4 Сравнение результатов обучения сетей

Эксперимент 1

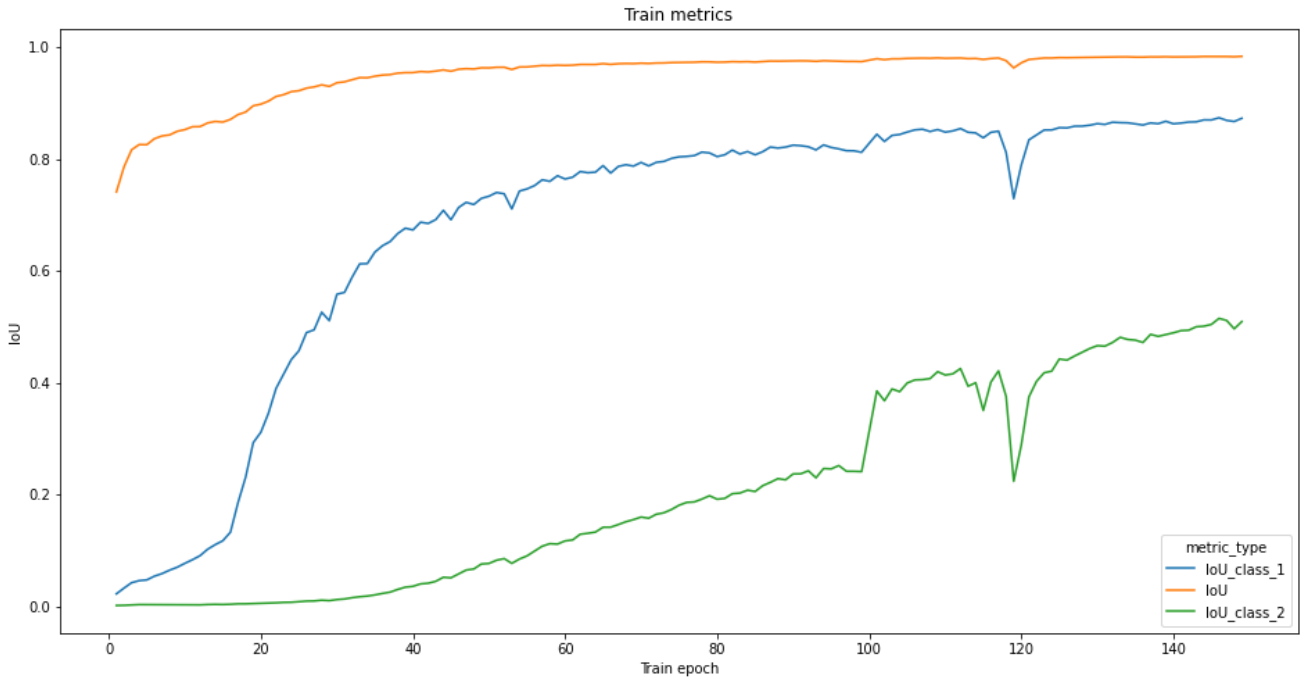
(Softmax + categorical cross entropy | Unet)

Результаты работы:

1) Тестовый набор:



2) Тренировочный набор:



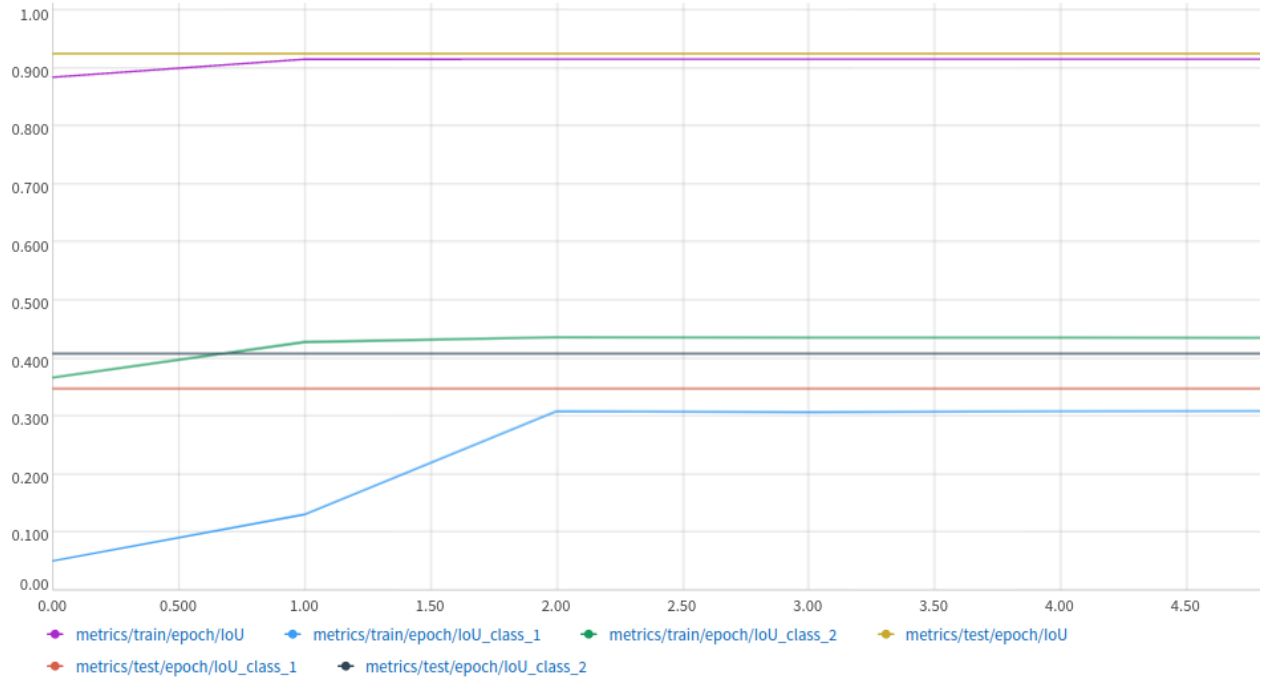
3) Основные выводы:

Модель в процессе обучения смогла добиться определенных результатов. Как мы можем видеть, из-за дисбаланса классов сеть долгое время не пыталась предугадывать наименее проявленный класс. Причина тому — слабый размер ошибки, превносимый данным классом. В целом процесс обучения получился весьма занимательным: первоначально модель пыталась как можно обучиться на нулевом классе, предсказывая всё как нулевой класс, затем, убрав ошибку на этом классе, она принялась распознавать второй класс.

Имея данное поведение, я сделал следующее предположение: возможно нужно дать равный штраф на каждый класс пикселей, чтобы получить равномерное увеличение IoU на каждом классе, вместо ступенчатого. Я видоизменил формулу, добавив вес каждому классу:

$$CrossEntropy_{Categorical} = -\sum_{i=1}^{N} w_i * y_i \log p_i$$

Результаты следущие:



Модель начала сходиться к одному из двух минимумов:

1) Предсказывание всех объектов как нулевой класс

2) Равномерная распределение для каждого класса (вероятность каждого класса стремилась к 0.3333)

В дальнейшем видоизмененую categorical cross entropy решено не использовать.

В чем причина ступенчатого поведения при обучении? Причина может крыться не столько в размере штрафа класса, сколько в формуле штрафа. Модель получает штраф только от True Positive пикселей, тем самым она смелее выдаёт другим классам (которых по нашей задаче меньше) False Positive результаты. Как это можно попытаться побороть? Возможным решением может оказаться переход к binary cross entropy и сигмоидальной активации. Да, мы потеряем некоторую часть интерпретируемости предсказаний, т.к. мы выбираем не один класс из множества с максимальной вероятность, а просто берём класс с максимальной проявленностью признака принадлежности. Каждый класс обучается против всех остальных, но вместе с этим мы получаем штраф за False Positive у каждого пикселя.

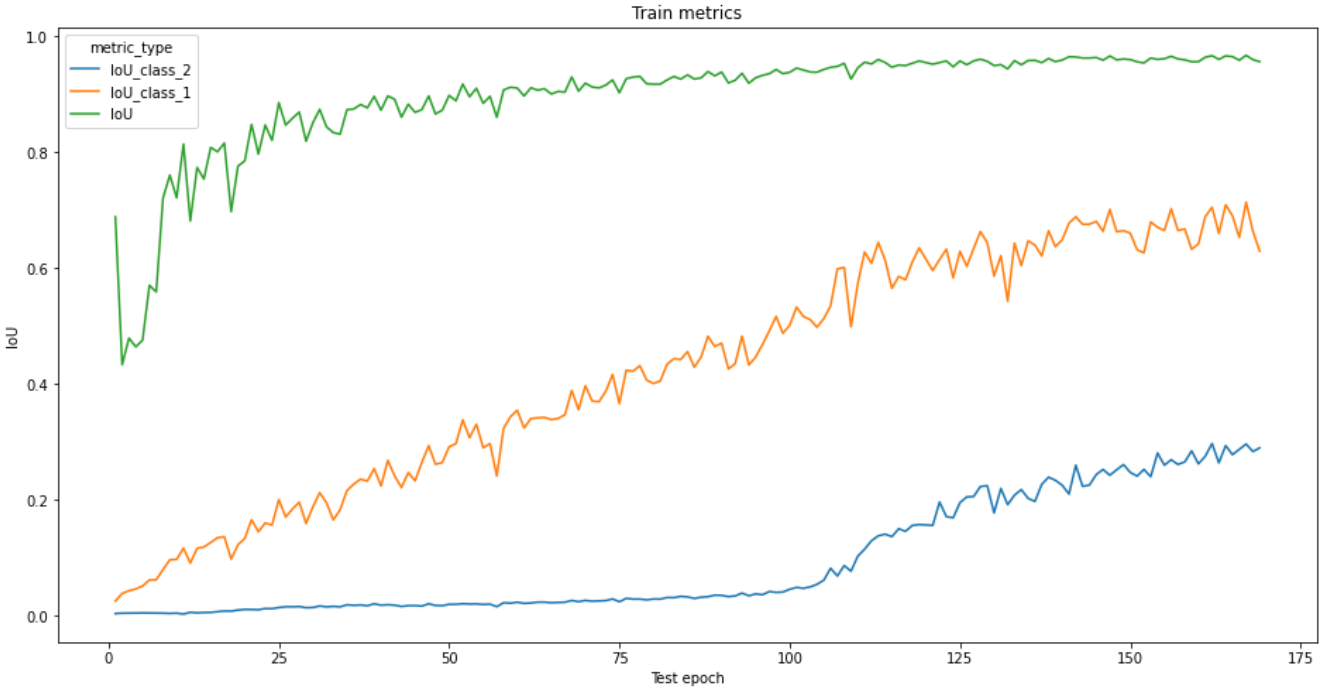
$$CrossEntropy_{Binary} = -\sum_{i=1}^{N} y_i \log p_i - (1 - y_i) \log (1 - p_i)$$

Эксперимент 2

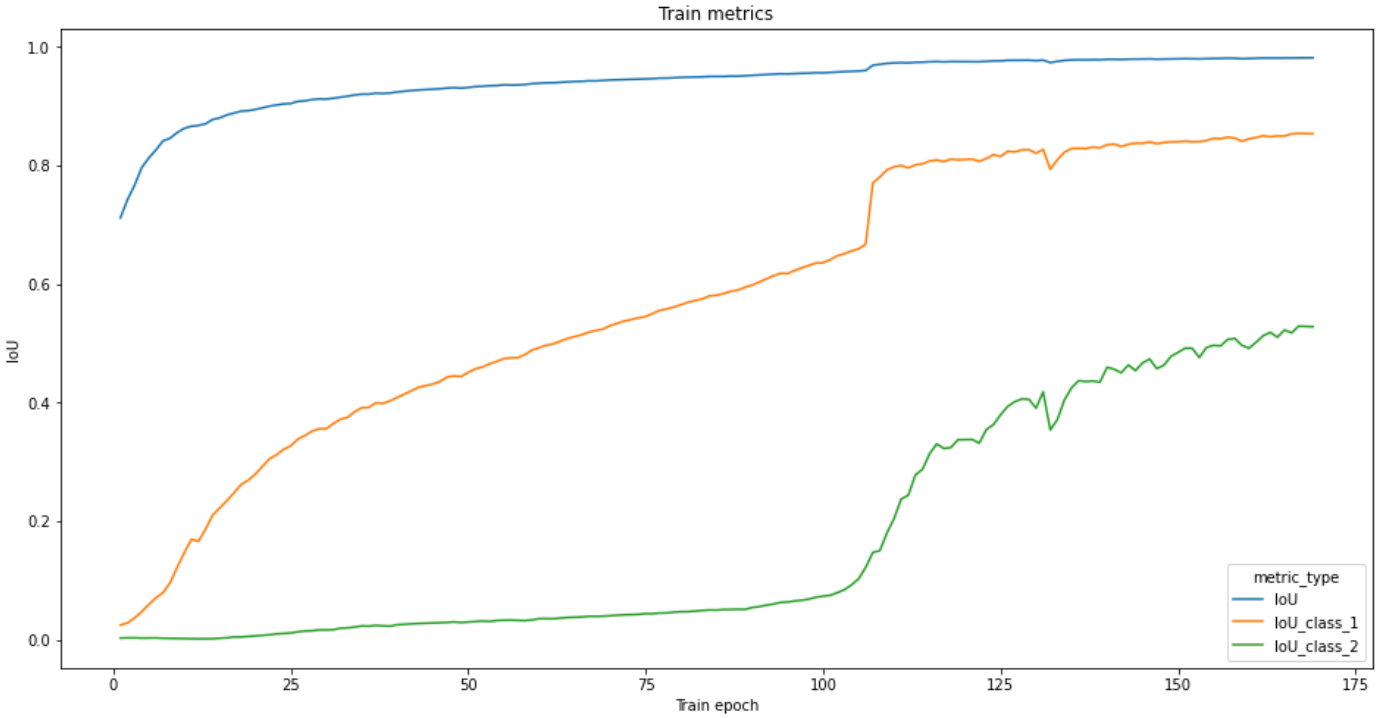
(Sigmoid + binary cross entropy | Unet)

Результаты работы:

1) Тестовый набор:



2) Тренировочный набор:



3) Основные выводы:

К сожалению, модель не отработала так, как предполагалось, однако конечный результат не стал заметно хуже, лучше. Применение весов к binary CE делает модель нестабильной из-за случайных False Positive (возможно tensorflow не всегда использует символьное дифференцирование).

Как итог, categorical cross entropy в среднем превосходит по метрикам binary на 4-5% на тестовом наборе и имеет сравнительно одинаковые показатели на тренировочном.

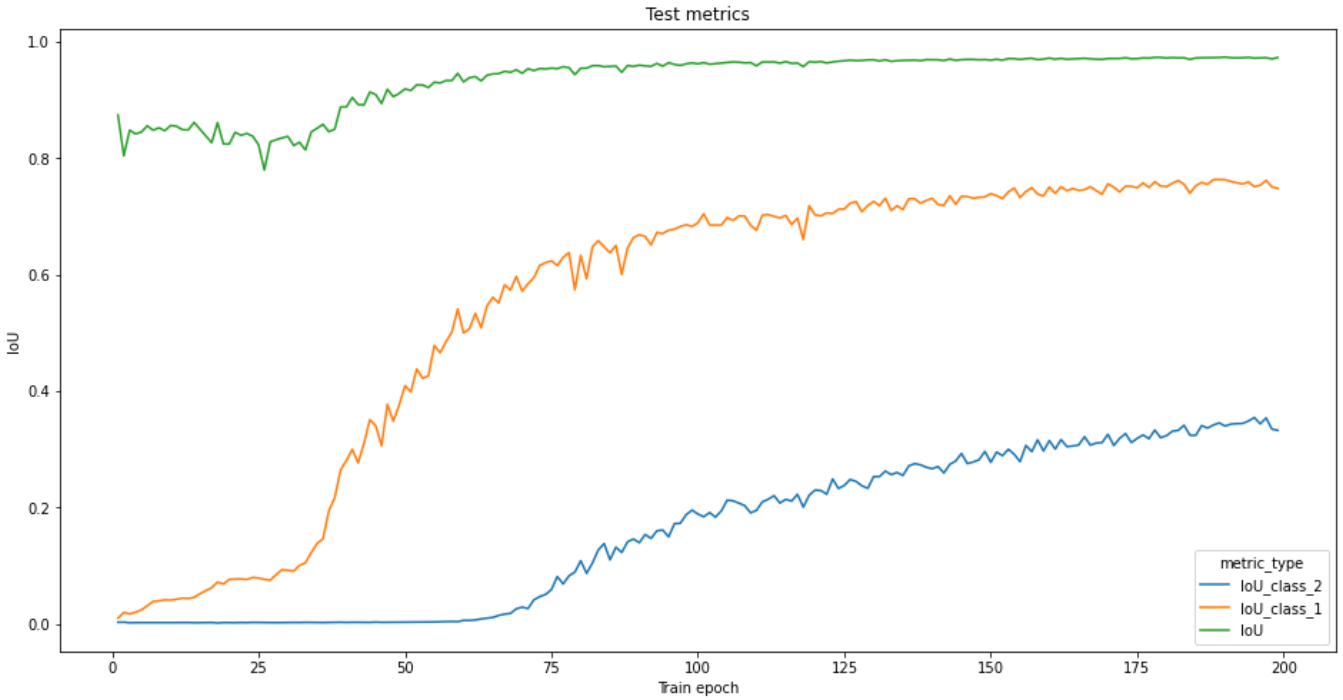
Перейдём непосредственно к обучению Unet-X, и посмотрим на её результаты.

Эксперимент 3

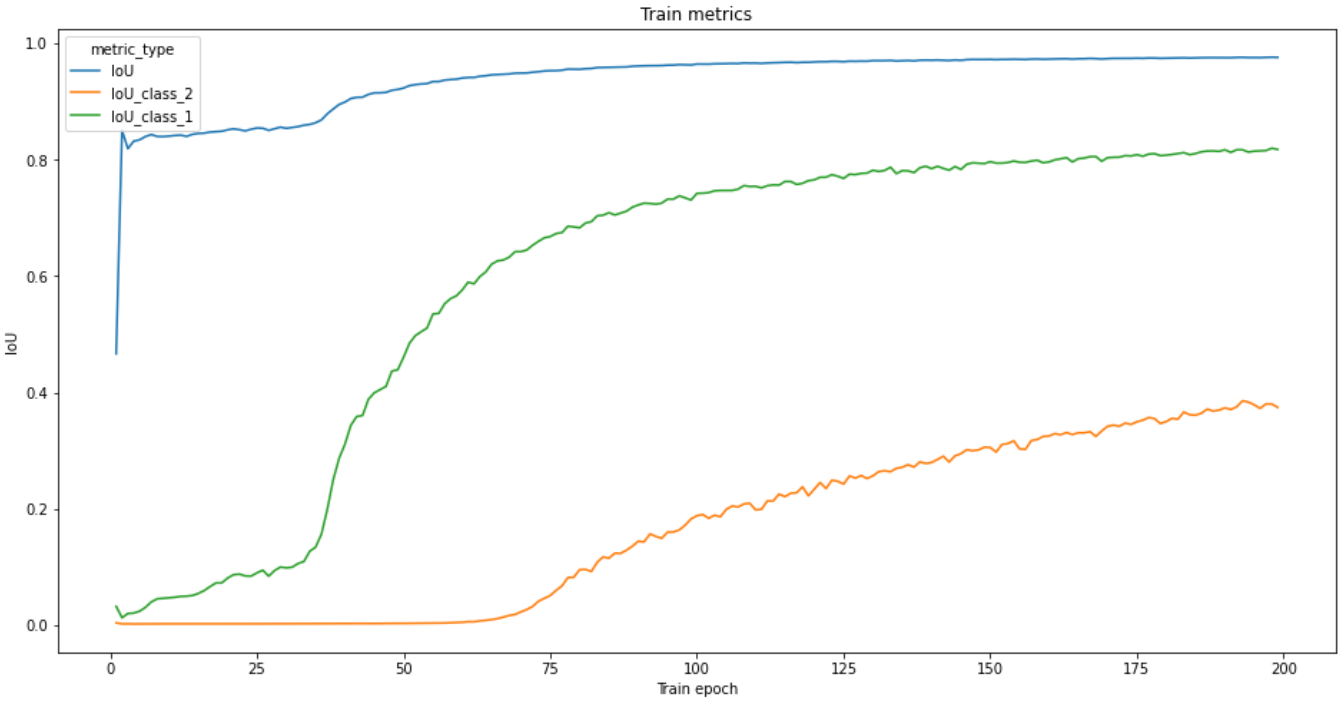
(Softmax + categorical cross entropy | Unet-X)

Результаты работы:

1) Тестовый набор:



2) Тренировочный набор:



3) Основные выводы:

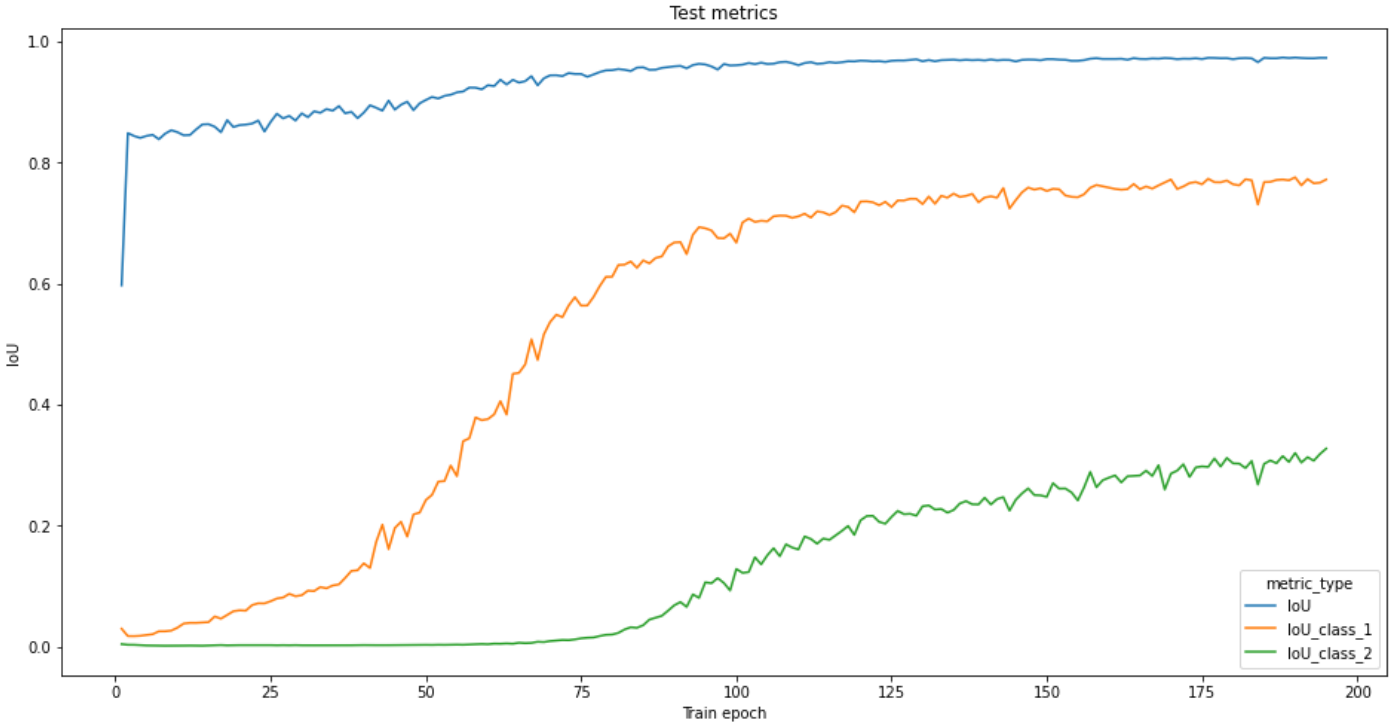
Первое, что хочется отметить: Unet-X обучается в 2 раза быстрее своего предшественника и имеет в 5 раз меньше параметров. Среднее время обучения 1 эпохи на Unet составляет 430 секунд, а на Unet-X – 209. Что касается качества полученной модели: тернировочные метрики у Unet-X немного ниже, чем у Unet, чего нельзя сказать о тестовых метриках. Unet-X имеет лучшую обощающую способность, что в рамках машинного обучения является основополагающим фактором для сравнения моделей. Причина может крыться в меньшей ёмкости полученной модели, потому модель в меньшей степени пытается запоминать конкретные картинки из тренировочного набора, она просто ищет способ обобщить имеющиеся данные.

Эксперимент 4

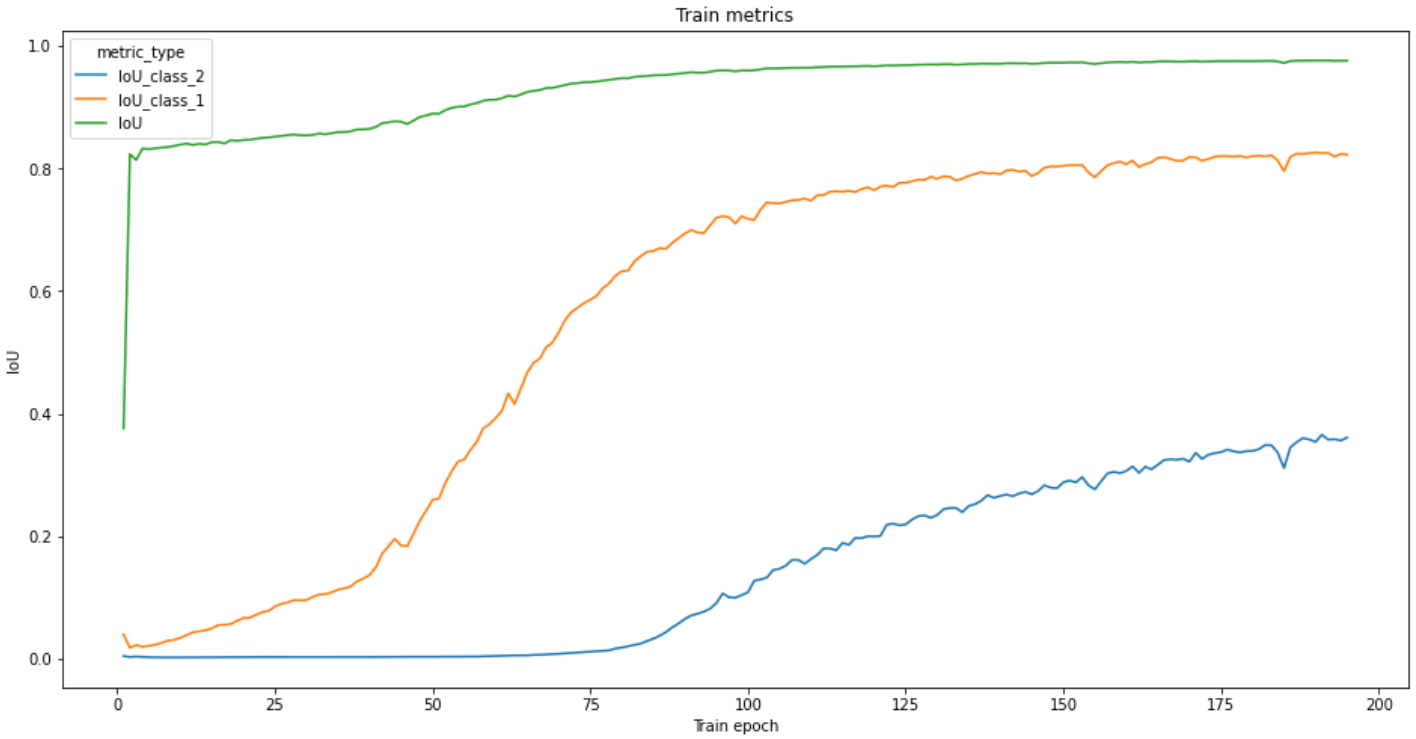
(Sigmoid + binary cross entropy | Unet-X)

Результаты работы:

1) Тестовый набор:



2) Тренировочный набор:



3) Основные выводы

Аналогично Unet, при данном способе обучения модель имеет чуть меньше метрики относительно categorical cross entropy, но сохраняет основные тенденции предыдущего эксперимента.

3.5 Результаты сравнения

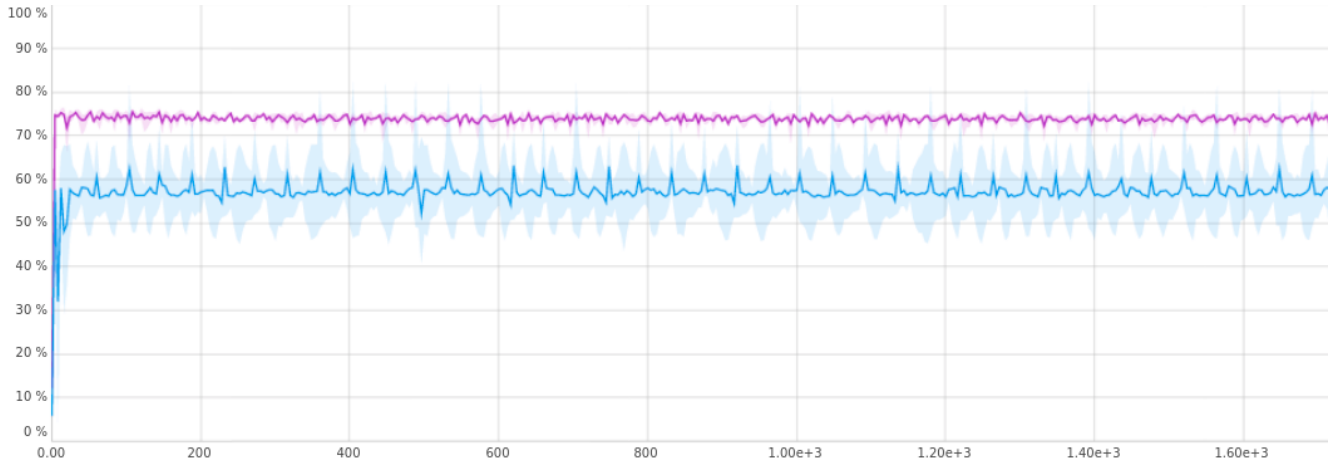
В ходе проведенных экспериментов мы смогли решить задачу многоклассовой сегментации двумя моделями: Unet и Unet-X. Результаты работы таковы: модели с большой точностью смогли распознать пешеходные переходы, однако с дорожными занаками возникли трудности, хотя в целом модель справилась и с этим.

Что касается Unet-X, модель оказалась более устойчивой к переобучению. Помимо этого, меньшее число параметров позволяет быстрее обрабатывать изображения, что в случае применения данной модели в реальных продуктах делает её крайне эффективной.

В дальнейшем обе модели процесс обучения можно было бы усовершенствовать следующим образом: добавить в набор большее количество изображений знаков, так как класс сам по себе имеет менее очевидную структуру, нежели пешеходные переходы, рассмотреть разное число уровней вложенности в данных архитектура, улучшить качество разметки исходных данных (в особенности для дорожных знаков).

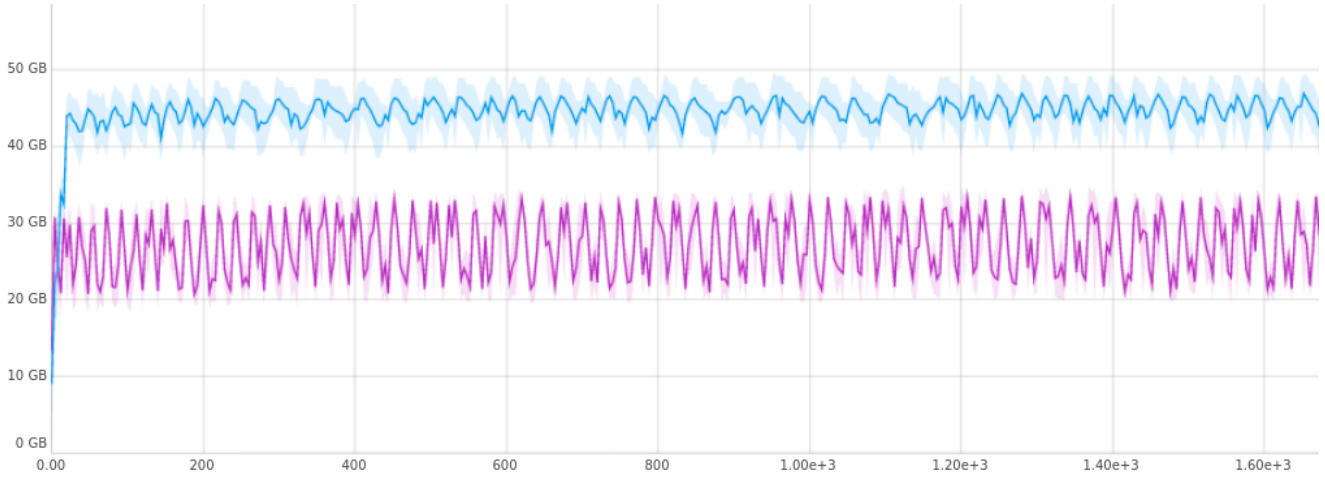
Рассмотрим потребление ресурсов системы в процессе обучения. Для этого я арендовал Google Cloud Server на 16 ядер и 60 гигабайт оперативной памяти. Фиолетовый – Unet-X, Голубой – Unet. Размер батча – 128 картинок.

1) Потребление ресурсов процессора



Из интересного могу отметить чуть меньшую дисперсию потребления у Unet-X, но не я буду слишком смелым в своих предположениях.

2) Потребление памяти



Как мы видим Unet-X требует меньше памяти в процессе обучения, что предполагалось изначально. Однако в отличие от потребления процессорных ресурсов, дисперсия потребления выше у Unet-X.

3) Итоговая таблица потребления ресурсов:

Характеристика\Модель	Unet	Unet-X
Время обучения эпохи	430 сек.	209 сек.
Среднее потребление памяти	43 Gb	25 Gb
Потребление процессора	75%	58%

ЗАКЛЮЧЕНИЕ

В ходе написания работы был изучен теоретический материал о использовании нейросетевых подходов в задаче многоклассовой сегментации и рассмотрены такие архитектуры свёрточных сетей как Unet и Unet-X. На основе изученного материала данные модели были реализованы и протестированы в контексте задачи распознавания пешеходных переходов и дорожных знаков. В ходе сравнительного анализа были определены сильные и слабые стороны каждой модели и изучены области применимости каждой из архитектур.

Перечисленные модели не являются единственными представителями, способными решать данную задачу. Полученные результаты можно использовать для сравнительного анализа с другими моделями для данного класса задач.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Что такое машинное обучение? [Электронный ресурс]. – 2021. – Режим доступа: https://www.oracle.com/ru/data-science/machine-learning/what-is-machine-learning/ Дата доступа: 02.11.2021.

2 A 2021 guide to Semantic Segmentation [Электронный ресурс]. – 2021. – Режим доступа: https://nanonets.com/blog/semantic-image-segmentation-2020/ Дата доступа: 02.11.2021

3 Введение в машинное обучение [Электронный ресурс]. – 2019. – Режим доступа: https://habr.com/ru/post/448892/ Дата доступа: 02.11.2021.

4 Специалист по машинному обучению в команду беспилотных автомобилей [Электронный ресурс]. – 2021. – Режим доступа: https://yandex.com/jobs/vacancies/%D1%81%D0%BF%D0%B5%D1%86%D0%B8%D0%B0%D0%BB%D0%B8%D1%81%D1%82-%D0%BF%D0%BE-%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%BC%D1%83-%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D1%8E-%D0%B2-%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D1%83-%D0%B1%D0%B5%D1%81%D0%BF%D0%B8%D0%BB%D0%BE%D1%82%D0%BD%D1%8B%D1%85-%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%BE%D0%B1%D0%B8%D0%BB%D0%B5%D0%B9-3741 Дата доступа: 02.11.2021.

5 Автоматическая сегментация изображений рукописных документов [Электронный ресурс]. – 2014. – Режим доступа: http://www.machinelearning.ru/wiki/images/0/06/2014_517_MalyshevaEK.pdf Дата доступа: 02.11.2021

6 Метрики в задачах машинного обучения [Электронный ресурс]. – 2017. – Режим доступа: https://habr.com/ru/company/ods/blog/328372/ Дата доступа: 02.11.2021

7 Нейронная сеть [Электронный ресурс]. – 2021. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C Дата доступа: 07.11.2021

8 Глубокое обучение [Электронный ресурс]. – 2021. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D1%83%D0%B1%D0%BE%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5 Дата доступа: 07.11.2021

9 Свёрточная нейронная сеть [Электронный ресурс]. – 2021. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%91%D1%80%D1%82%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C Дата доступа: 07.11.2021

10 Введение в нейронные сети [Электронный ресурс]. – 2021. – Режим доступа: https://neuralnet.info/chapter/%D0%B2%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5/ Дата доступа: 07.11.2021

11 Streetview crosswalk classification dataset [Электронный ресурс]. – 2021. – Режим доступа: https://github.com/edadasko/crosswalk_segmentation/ Дата доступа: 07.11.2021

12 U-Net: нейросеть для сегментации изображений [Электронный ресурс]. – 2018. – Режим доступа: https://neurohive.io/ru/vidy-nejrosetej/u-net-image-segmentation/ Дата доступа: 07.11.2021

13 Функция потерь [Электронный ресурс]. – 2021. – Режим доступа: https://www.helenkapatsa.ru/funktsiia-potieri/ Дата доступа: 07.11.2021

14 Машинное обучение [Электронный ресурс]. – 2021. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5/ Дата доступа: 07.11.2021

15 Искусственный интеллект [Электронный ресурс]. – 2021. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D0%BB%D0%BB%D0%B5%D0%BA%D1%82 Дата доступа: 07.11.2021

16 VGG Image Annotator [Электронный ресурс]. – 2021. – Режим доступа: https://www.robots.ox.ac.uk/~vgg/software/via/ Дата доступа: 07.11.2021

17 Classification: True vs. False and Positive vs. Negative [Электронный ресурс]. – 2021. – Режим доступа: https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative Дата доступа: 07.11.2021

18 Метод обратного распространения ошибки [Электронный ресурс]. – 2021. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B3%D0%BE_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B8 Дата доступа: 07.11.2021

19 Сигмоида [Электронный ресурс]. – 2021. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BC%D0%BE%D0%B8%D0%B4%D0%B0 Дата доступа: 07.11.2021

20 Функции активации нейросети [Электронный ресурс]. – 2018. – Режим доступа: https://neurohive.io/ru/osnovy-data-science/activation-functions/ Дата доступа: 07.11.2021

21 Rectifier (neural networks) [Электронный ресурс]. – 2018. – Режим

доступа: https://en.wikipedia.org/wiki/Rectifier_(neural_networks) Дата доступа: 07.11.2021

22 Глубокое обучение [Электронный ресурс]. – 2021. – Режим доступа: https://library.kre.dp.ua/Books/2-4%20kurs/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%20+%20%D0%BC%D0%BE%D0%B2%D0%B8%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B8%D0%B9%20%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82/Machineobuchenie@bzd_channel.pdf Дата доступа: 07.11.2021

23 Стохастический градиентный спуск [Электронный ресурс]. – 2021. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%BE%D1%85%D0%B0%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B3%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D1%83%D1%81%D0%BA Дата доступа: 07.11.2021

24 Что такое jupyter-ноутбук [Электронный ресурс]. – 2021. – Режим доступа: https://thecode.media/jupyter/#:~:text=Jupyter%2D%D0%BD%D0%BE%D1%83%D1%82%D0%B1%D1%83%D0%BA%20%E2%80%94%20%D1%8D%D1%82%D0%BE%20%D1%81%D1%80%D0%B5%D0%B4%D0%B0%20%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8,%D0%BA%D0%BE%D0%B4%D0%B0%20%D0%B8%20%D0%B5%D0%B3%D0%BE%20%D0%BE%D1%82%D0%B4%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D1%85%20%D1%84%D1%80%D0%B0%D0%B3%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%B2.&text=%D0%92%20%D1%82%D0%B0%D0%BA%D0%BE%D0%B9%20%D1%81%D1%80%D0%B5%D0%B4%D0%B5%20%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8%20%D0%BC%D0%BE%D0%B6%D0%BD%D0%BE,%D0%BC%D0%BE%D0%B6%D0%BD%D0%BE%20%D0%BF%D0%BE%D0%BC%D0%B5%D0%BD%D1%8F%D1%82%D1%8C%20%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA%20%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BA%D0%BE%D0%B4%D0%B0. Дата доступа: 07.11.2021

25 Кластеризация: метод k-средних [Электронный ресурс]. – 2021. – Режим доступа: http://statistica.ru/theory/klasterizatsiya-metod-k-srednikh/ Дата доступа: 07.11.2021

26 Multiclass semantic segmentation using U-Net [Электронный ресурс]. – 2021. – Режим доступа: https://www.youtube.com/watch?v=XyX5HNuv-xE Дата доступа: 07.11.2021

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

Модель UNet

```
def multi_unet_model(n_classes=4, IMG_HEIGHT=256, IMG_WIDTH=256, IMG_CHANNELS=1):

    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    s = Lambda(lambda x: x / 255)(inputs)
    s = inputs

    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)
    c1 = Dropout(0.2)(c1)
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c1)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p1)
    c2 = Dropout(0.2)(c2)
    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c3)
    p3 = MaxPooling2D((2, 2))(c3)

    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c4)
    p4 = MaxPooling2D(pool_size=(2, 2))(c4)

    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c5)


    u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c6)
```

```
    u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c7)

    u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u8)
    c8 = Dropout(0.2)(c8)  # Original 0.1
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c8)
```



```
    u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u9)
    c9 = Dropout(0.2)(c9)  # Original 0.1
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c9)

    outputs = Conv2D(n_classes, (1, 1), activation='softmax')(c9)

    model = Model(inputs=[inputs], outputs=[outputs])


    return model
```

Модель UNet-X

```
def separableConv(x, channels):
    x=DepthwiseConv2D((3,3),padding="same")(x)
    x=Conv2D(channels, (1,1),padding="same", activation="relu")(x)
    return x

def multi_unetx_model(n_classes=4, IMG_HEIGHT=256, IMG_WIDTH=256, IMG_CHANNELS=1):
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    s = Lambda(lambda x: x / 255)(inputs)
    s = inputs

    conv_1_1 = separableConv(s, 16)
    conv_1_1 = Dropout(0.1) (conv_1_1)
    conv_1_2 = separableConv(conv_1_1, 16)
    pool_1 = MaxPooling2D(2)(conv_1_2)

    conv_2_1 = separableConv(pool_1, 32)
    conv_2_1 = Dropout(0.1) (conv_2_1)
    conv_2_2 = separableConv(conv_2_1, 32)
    pool_2 = MaxPooling2D(2)(conv_2_2)

    conv_3_1 = separableConv(pool_2, 64)
    conv_3_1 = Dropout(0.2) (conv_3_1)
    conv_3_2 = separableConv(conv_3_1, 64)
    pool_3 = MaxPooling2D(2)(conv_3_2)
```

```
    conv_4_1 = separableConv(pool_3, 128)
    conv_4_1 = Dropout(0.2) (conv_4_1)
    conv_4_2 = separableConv(conv_4_1, 128)
    pool_4 = MaxPooling2D(2)(conv_4_2)

    conv_5_1 = separableConv(pool_4, 256)
    conv_5_1 = Dropout(0.3) (conv_5_1)
    conv_5_2 = separableConv(conv_5_1, 256)

    up_1 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same') (conv_5_2)
    conc_1 = concatenate([conv_4_2, up_1])
    conv_up_1_1 = separableConv(conc_1, 128)
    conv_up_1_1 = Dropout(0.2) (conv_up_1_1)
    conv_up_1_2 = separableConv(conv_up_1_1, 128)
```



```
    up_2 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (conv_up_1_2)
    conc_2 = concatenate([conv_3_2, up_2])
    conv_up_2_1 = separableConv(conc_2, 64)
    conv_up_2_1 = Dropout(0.2) (conv_up_2_1)
    conv_up_2_2 = separableConv(conv_up_2_1, 64)

    up_3 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (conv_up_2_2)
    conc_3 = concatenate([conv_2_2, up_3])
    conv_up_3_1 = separableConv(conc_3, 32)
    conv_up_3_1 = Dropout(0.1) (conv_up_3_1)
    conv_up_3_2 = separableConv(conv_up_3_1, 32)

    up_4 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (conv_up_3_2)
    conc_4 = concatenate([conv_1_2, up_4])
    conv_up_4_1 = separableConv(conc_4, 16)
    conv_up_4_1 = Dropout(0.1) (conv_up_4_1)
    conv_up_4_2 = separableConv(conv_up_4_1, 16)
    outputs = Conv2D(n_classes, (1, 1), activation='softmax') (conv_up_4_2)

    model = Model(inputs=[inputs], outputs=[outputs])
    return model
```

Генерация масок из формата JSON

```
IMG_WIDTH = 256
IMG_HEIGHT = 256
IMG_CHANNELS = 3
TRAIN_PATH = 'images/'
      souce_folder = "images"
json_path = "via_project_4Oct2021_9h22m (37).json"
file_polygons = {}
      with open(json_path) as f:
    data = json.load(f)["_via_img_metadata"]

      def add_to_dict(data, itr, key, count):
    x_points = []
    try:
```

```
            x_points = data[itr]["regions"][count]["shape_attributes"]["all_points_x"]
            y_points = data[itr]["regions"][count]["shape_attributes"]["all_points_y"]
        except:
            pass

        all_points = []

for i, x in enumerate(x_points):
            all_points.append([x, y_points[i]])

        file_polygons[key] = all_points

    for itr in data:
        file_name_json = data[itr]["filename"]
        sub_count = 0
```

```
    if len(data[itr]["regions"]) > 1:
        for _ in range(len(data[itr]["regions"])):
            key = file_name_json[:-4] + "*" + str(sub_count+1)
            add_to_dict(data, itr, key, sub_count)
            sub_count += 1
    else:
        add_to_dict(data, itr, file_name_json[:-4], sub_count)

 for file_name in os.listdir(source_folder):
     to_save_folder = os.path.join(source_folder, file_name[:-4])
     image_folder = os.path.join(to_save_folder, "images")
    mask_folder = os.path.join(to_save_folder, "masks")
     curr_img = os.path.join(source_folder, file_name)

    try:
         os.mkdir(to_save_folder)
        os.mkdir(image_folder)
        os.mkdir(mask_folder)
        os.rename(curr_img, os.path.join(image_folder, file_name))
     except:
         pass


from PIL import Image, ImageChops
from matplotlib import pyplot as plt

def crop_center(pil_img, crop_width: int, crop_height: int) -> Image:
    """
    Функция для обрезки изображения по центру.
    """
    img_width, img_height = pil_img.size
    return pil_img.crop(((img_width - crop_width) // 2,
                         (img_height - crop_height) // 2,
                         (img_width + crop_width) // 2,
                         (img_height + crop_height) // 2))



for polygone in file_polygons:
    num_masks = polygone.split("*")
    to_save_folder = os.path.join(source_folder, num_masks[0])
    mask_folder = os.path.join(to_save_folder, "masks")

    im = cv2.imread(os.path.join(to_save_folder, "images", num_masks[0] + ".png"))
    mask = np.zeros((im.shape[0], im.shape[1]))
    arr = np.array(file_polygons[polygone])

    im = Image.open(os.path.join(to_save_folder, "images", num_masks[0] + ".png"))
    im_new = crop_center(im, 208, 217)
    im_new.save(os.path.join(to_save_folder, "images", num_masks[0] + ".png"), quality=1000)

    try:
        cv2.fillPoly(mask, [arr], color=(255))
    except:
        pass
```

```
    if len(num_masks) > 1:
        cv2.imwrite(os.path.join(mask_folder, polygone.replace("*", "_") + ".png"), mask)
        im = Image.open(os.path.join(mask_folder, polygone.replace("*", "_") + ".png"))
        im_new = crop_center(im, 208, 217)
        im_new.save(os.path.join(mask_folder, polygone.replace("*", "_") + ".png"), quality=1000)
    else:
        cv2.imwrite(os.path.join(mask_folder, polygone + ".png"), mask)
        im = Image.open(os.path.join(mask_folder, polygone + ".png"))
        im_new = crop_center(im, 208, 217)
        im_new.save(os.path.join(mask_folder, polygone + ".png"), quality=1000)
```

Создание масок для дорожных знаков

```
import sys
from tqdm import tqdm
from skimage.io import imread, imshow, imread_collection, concatenate_images
from skimage.transform import resize
train_ids = [0] * 3083
for i in range(3083):
    train_ids[i] = 'lane_' + str(i)

X_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS), dtype=np.uint8)
Y_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, 1), dtype=np.bool)

print("X_train",X_train.shape)
print("Y_train",Y_train.shape)
print('Getting and resizing train images and masks ... ')
sys.stdout.flush()
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):
    path = TRAIN_PATH + id_
    if id_[0] == '.':
        continue
#     print(path + '/images/' + id_ + '.png')
    img = imread(path + '/images/' + id_ + '.png')[:,:,:IMG_CHANNELS]
    img = resize(img, (IMG_HEIGHT, IMG_WIDTH), mode='constant', preserve_range=True)
    X_train[n] = img
    mask = np.zeros((IMG_HEIGHT, IMG_WIDTH, 1), dtype=np.bool)
    for mask_file in next(os.walk(path + '/masks/'))[2]:
        if mask_file[0] == '.':
            continue
        mask_ = imread(path + '/masks/' + mask_file)
        mask_ = np.expand_dims(resize(mask_, (IMG_HEIGHT, IMG_WIDTH),
mode='constant',preserve_range=True), axis=-1)
        mask = np.maximum(mask, mask_)
    Y_train[n] = mask
print('Done!')
```

Сохранение масок дорожных знаков и наложение масок друг на друга

```
np.save('x_train.npy', X_train)
np.save('y_train.npy', Y_train)
multi_Y_train = np.zeros((3083, 256, 256, 1), dtype=int)
X_train1 = np.load('dataset/x_train.npy')
Y_train1 = np.load('dataset/y_train.npy')
ind = 0
for i in range(3083):
    for j in range(256):
        for k in range(256):
            ind+=1
            if ind == 1000000:
                print('1000000')
            if ind == 10000000:
                print('10000000')
            if ind == 100000000:
                print('100000000')
            if Y_train[i][j][k] == True:
                multi_Y_train[i][j][k] = 2
            if Y_train1[i][j][k] == True:
                multi_Y_train[i][j][k] = 1
np.save('multi_y_train.npy', multi_Y_train)
```

Подготовка данных для много классовой сегментации

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
n, h, w, b = multi_Y_train.shape
train_masks_reshaped = multi_Y_train.reshape(-1,1)
train_masks_reshaped_encoded = labelencoder.fit_transform(train_masks_reshaped)
train_masks_encoded_original_shape = train_masks_reshaped_encoded.reshape(n, h, w)
train_masks_input = np.expand_dims(train_masks_encoded_original_shape, axis=3)
train_masks_input.shape
from tensorflow.keras.utils import to_categorical
train_masks_cat = to_categorical(train_masks_input, num_classes=3)
y_train_cat = train_masks_cat.reshape((multi_Y_train.shape[0], multi_Y_train.shape[1],
multi_Y_train.shape[2], 3))
from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight('balanced',
                                        np.unique(train_masks_reshaped_encoded),
                                        train_masks_reshaped_encoded)
print("Class weights are...:", class_weights)
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Модели нейронных сетей

UNet

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)	0	
conv2d (Conv2D)	(None, 256, 256, 16)	448	input_1[0][0]
dropout (Dropout)	(None, 256, 256, 16)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 16)	2320	dropout[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 32)	4640	max_pooling2d[0][0]
dropout_1 (Dropout)	(None, 128, 128, 32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 32)	9248	dropout_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 64)	18496	max_pooling2d_1[0][0]
dropout_2 (Dropout)	(None, 64, 64, 64)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 64)	36928	dropout_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 128)	73856	max_pooling2d_2[0][0]
dropout_3 (Dropout)	(None, 32, 32, 128)	0	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 128)	147584	dropout_3[0][0]

max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 16, 16, 256)	295168	max_pooling2d_3[0][0]
dropout_4 (Dropout)	(None, 16, 16, 256)	0	conv2d_8[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 256)	590080	dropout_4[0][0]

conv2d_transpose (Conv2DTranspo	(None, 32, 32, 128)	131200	conv2d_9[0][0]
concatenate (Concatenate)	(None, 32, 32, 256)	0	conv2d_transpose[0][0] conv2d_7[0][0]
conv2d_10 (Conv2D)	(None, 32, 32, 128)	295040	concatenate[0][0]
dropout_5 (Dropout)	(None, 32, 32, 128)	0	conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 32, 32, 128)	147584	dropout_5[0][0]
conv2d_transpose_1 (Conv2DTrans	(None, 64, 64, 64)	32832	conv2d_11[0][0]
concatenate_1 (Concatenate)	(None, 64, 64, 128)	0	conv2d_transpose_1[0][0] conv2d_5[0][0]
conv2d_12 (Conv2D)	(None, 64, 64, 64)	73792	concatenate_1[0][0]
dropout_6 (Dropout)	(None, 64, 64, 64)	0	conv2d_12[0][0]
conv2d_13 (Conv2D)	(None, 64, 64, 64)	36928	dropout_6[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 128, 128, 32)	8224	conv2d_13[0][0]
concatenate_2 (Concatenate)	(None, 128, 128, 64)	0	conv2d_transpose_2[0][0] conv2d_3[0][0]
conv2d_14 (Conv2D)	(None, 128, 128, 32)	18464	concatenate_2[0][0]
dropout_7 (Dropout)	(None, 128, 128, 32)	0	conv2d_14[0][0]
conv2d_15 (Conv2D)	(None, 128, 128, 32)	9248	dropout_7[0][0]
conv2d_transpose_3 (Conv2DTrans	(None, 256, 256, 16)	2064	conv2d_15[0][0]
concatenate_3 (Concatenate)	(None, 256, 256, 32)	0	conv2d_transpose_3[0][0] conv2d_1[0][0]
conv2d_16 (Conv2D)	(None, 256, 256, 16)	4624	concatenate_3[0][0]

dropout_8 (Dropout)	(None, 256, 256, 16)	0	conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 256, 256, 16)	2320	dropout_8[0][0]
conv2d_18 (Conv2D)	(None, 256, 256, 3)	51	conv2d_17[0][0]

Total params: 1,941,139

Trainable params: 1,941,139

Non-trainable params: 0

UNet-X

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)	0	
depthwise_conv2d (DepthwiseConv	(None, 256, 256, 3)	30	input_1[0][0]
conv2d (Conv2D)	(None, 256, 256, 16)	64	depthwise_conv2d[0][0]
dropout (Dropout)	(None, 256, 256, 16)	0	conv2d[0][0]
depthwise_conv2d_1 (DepthwiseCo	(None, 256, 256, 16)	160	dropout[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 16)	272	depthwise_conv2d_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0	conv2d_1[0][0]
depthwise_conv2d_2 (DepthwiseCo	(None, 128, 128, 16)	160	max_pooling2d[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 32)	544	depthwise_conv2d_2[0][0]
dropout_1 (Dropout)	(None, 128, 128, 32)	0	conv2d_2[0][0]
depthwise_conv2d_3 (DepthwiseCo	(None, 128, 128, 32)	320	dropout_1[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 32)	1056	depthwise_conv2d_3[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d_3[0][0]
depthwise_conv2d_4 (DepthwiseCo	(None, 64, 64, 32)	320	max_pooling2d_1[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 64)	2112	depthwise_conv2d_4[0][0]
dropout_2 (Dropout)	(None, 64, 64, 64)	0	conv2d_4[0][0]
depthwise_conv2d_5 (DepthwiseCo	(None, 64, 64, 64)	640	dropout_2[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 64)	4160	depthwise_conv2d_5[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0	conv2d_5[0][0]
depthwise_conv2d_6 (DepthwiseCo	(None, 32, 32, 64)	640	max_pooling2d_2[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 128)	8320	depthwise_conv2d_6[0][0]
dropout_3 (Dropout)	(None, 32, 32, 128)	0	conv2d_6[0][0]
depthwise_conv2d_7 (DepthwiseCo	(None, 32, 32, 128)	1280	dropout_3[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 128)	16512	depthwise_conv2d_7[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0	conv2d_7[0][0]
depthwise_conv2d_8 (DepthwiseCo	(None, 16, 16, 128)	1280	max_pooling2d_3[0][0]

conv2d_8 (Conv2D)	(None, 16, 16, 256)	33024	depthwise_conv2d_8[0][0]
dropout_4 (Dropout)	(None, 16, 16, 256)	0	conv2d_8[0][0]
depthwise_conv2d_9 (DepthwiseCo	(None, 16, 16, 256)	2560	dropout_4[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 256)	65792	depthwise_conv2d_9[0][0]
conv2d_transpose (Conv2DTranspo	(None, 32, 32, 128)	131200	conv2d_9[0][0]
concatenate (Concatenate)	(None, 32, 32, 256)	0	conv2d_7[0][0] conv2d_transpose[0][0]
depthwise_conv2d_10 (DepthwiseC	(None, 32, 32, 256)	2560	concatenate[0][0]
conv2d_10 (Conv2D)	(None, 32, 32, 128)	32896	depthwise_conv2d_10[0][0]
dropout_5 (Dropout)	(None, 32, 32, 128)	0	conv2d_10[0][0]

depthwise_conv2d_11 (DepthwiseC	(None, 32, 32, 128)	1280	dropout_5[0][0]
conv2d_11 (Conv2D)	(None, 32, 32, 128)	16512	depthwise_conv2d_11[0][0]
conv2d_transpose_1 (Conv2DTrans	(None, 64, 64, 64)	32832	conv2d_11[0][0]
concatenate_1 (Concatenate)	(None, 64, 64, 128)	0	conv2d_5[0][0] conv2d_transpose_1[0][0]
depthwise_conv2d_12 (DepthwiseC	(None, 64, 64, 128)	1280	concatenate_1[0][0]
conv2d_12 (Conv2D)	(None, 64, 64, 64)	8256	depthwise_conv2d_12[0][0]
dropout_6 (Dropout)	(None, 64, 64, 64)	0	conv2d_12[0][0]
depthwise_conv2d_13 (DepthwiseC	(None, 64, 64, 64)	640	dropout_6[0][0]
conv2d_13 (Conv2D)	(None, 64, 64, 64)	4160	depthwise_conv2d_13[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 128, 128, 32)	8224	conv2d_13[0][0]
concatenate_2 (Concatenate)	(None, 128, 128, 64)	0	conv2d_3[0][0] conv2d_transpose_2[0][0]
depthwise_conv2d_14 (DepthwiseC	(None, 128, 128, 64)	640	concatenate_2[0][0]
conv2d_14 (Conv2D)	(None, 128, 128, 32)	2080	depthwise_conv2d_14[0][0]
dropout_7 (Dropout)	(None, 128, 128, 32)	0	conv2d_14[0][0]
depthwise_conv2d_15 (DepthwiseC	(None, 128, 128, 32)	320	dropout_7[0][0]
conv2d_15 (Conv2D)	(None, 128, 128, 32)	1056	depthwise_conv2d_15[0][0]
conv2d_transpose_3 (Conv2DTrans	(None, 256, 256, 16)	2064	conv2d_15[0][0]
concatenate_3 (Concatenate)	(None, 256, 256, 32)	0	conv2d_1[0][0] conv2d_transpose_3[0][0]
depthwise_conv2d_16 (DepthwiseC	(None, 256, 256, 32)	320	concatenate_3[0][0]
conv2d_16 (Conv2D)	(None, 256, 256, 16)	528	depthwise_conv2d_16[0][0]
dropout_8 (Dropout)	(None, 256, 256, 16)	0	conv2d_16[0][0]
depthwise_conv2d_17 (DepthwiseC	(None, 256, 256, 16)	160	dropout_8[0][0]
conv2d_17 (Conv2D)	(None, 256, 256, 16)	272	depthwise_conv2d_17[0][0]
conv2d_18 (Conv2D)	(None, 256, 256, 3)	51	conv2d_17[0][0]

Total params: 386,577
Trainable params: 386,577
Non-trainable params: 0