

## **Сегментация долей головного мозга**

### **Отчёт**

Приложенные файлы:

<https://drive.google.com/drive/folders/1jfoFSAHDMd55cK-qsOhgETMjPnglzQ66?usp=sharing>

## Задание

Требуется решить задачу сегментации долей мозга (20 классов, перечислены в таблице ниже):

### Left hemisphere

Label 1 C (Caudate)  
Label 2 L (Lentiformis)  
Label 3 IC (Internal capsule)  
Label 4 I (Island)  
Label 5 M4  
Label 6 M5  
Label 7 M6  
Label 8 M1  
Label 9 M2  
Label 10 M3

### Right hemisphere

Label 11 C (Caudate)  
Label 12 L (Lentiformis)  
Label 13 IC (Internal capsule)  
Label 14 I (Island)  
Label 15 M4  
Label 16 M5  
Label 17 M6  
Label 18 M1  
Label 19 M2  
Label 20 M3

Данные: 7 КТ исследований мозга (3D снимков):

- `#.nii.gz` - исходные исследования;
- `#.txt` - соответствующие им углы поворота;
- `#-seg.nii.gz` (или `#-seg.nii`) разметка на 20 долей мозга.

Вследствие того, что исследования изначально были сделаны не под тем углом, под которым обычно врач смотрит на подобные исследования, врач перед разметкой исходные снимки повернул. Соответственно, данные углы поворота определяют, как нужно повернуть снимок мозга, чтобы ему соответствовала разметка.

Несколько полезных ссылок, чтобы начать:

- <http://www.itksnap.org/pmwiki/pmwiki.php> - для просмотра исходных объемов. Для того, чтобы в программе повернуть снимок, нужно сначала загрузить его (open main image), потом add another image - его же, затем tools -> registration -> в нижнем правом углу load transformation from file
- <https://nipy.org/nibabel/>
- <https://pypi.org/project/SimpleITK/>

P.S. Для построения моделей можно использовать любой удобный фреймворк. Цель ТЗ - показать свои скилы в анализе исходных данных, их препроцессинг, обосновать выбор той или иной архитектуры. По итогам необходимо написать отчетик с метриками, визуализацией результатов и всем прочим, что считается необходимым показать, в jupyter notebook, либо отдельным документом.

## I. Описание полученных данных.

Снимки КТ. Всего в наличии семь 3D растровых снимков в формате [Nifti](#). Размеры снимков 512x512x(98...116) пикселей. Цветовая информация сохранена в единственном канале снимка, диапазон цветов (-1024...3071). Служебная информация находится в заголовках файлов: пространственное положение, точки привязки, объёмы трехмерных пикселей (=voxels). В 3D массиве снимка координата Z стоит на первом месте, например, (116x512x512), об этом придётся помнить при работе со снимками.

Файлы разметки. Каждому снимку соответствует файл идентичного размера с сегментацией на 20 классов, как указано в ТЗ.

Текстовые файлы. Из ТЗ известно, что снимки перед сегментацией были повернуты, а параметры поворота хранятся в текстовых файлах. Содержимое текстового файла выглядит так:

```
1 #Insight Transform File V1.0
2 #Transform 0
3 Transform: MatrixOffsetTransformBase_double_3_3
4 Parameters: 0.9822735992397377 -0.18462253124159128 -0.03245145905126205 0.18211102422366576 ....
5 FixedParameters: 0 0 0
6
```

Одной из самых сложных и времязатратных частей задания было понять, что за информация содержится в этом файле. В итоге сложилась такая картина:

- Первые две строки – заголовок, не несущий существенной информации.
- Третья строка определяет тип трансформации снимка, в данном случае это смещение и поворот.
- Строка 4 состоит из двенадцати чисел, которые необходимо собрать в матрицу transform\_data [4x3]. После сборки оказывается, что первые три строки матрицы transform\_data [3,:] представляют собой матрицу Эйлера:

This matrix can be thought of a sequence of three rotations, one about each principle axis. Since matrix multiplication does not commute, the order of the axes which one rotates about will affect the result. For this analysis, we will rotate first about the  $x$ -axis, then the  $y$ -axis, and finally the  $z$ -axis. Such a sequence of rotations can be represented as the matrix product,

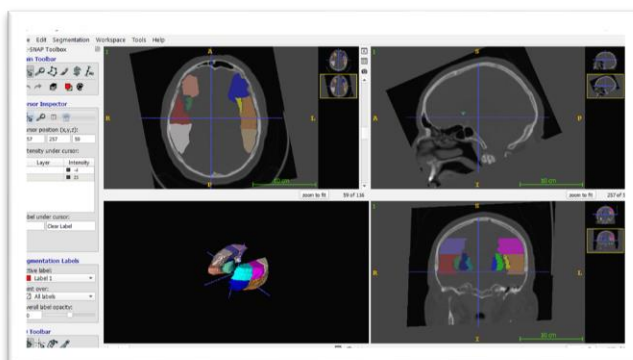
$$R = R_z(\phi)R_y(\theta)R_x(\psi)$$
$$= \begin{bmatrix} \cos \theta \cos \phi & \sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi & \cos \psi \sin \theta \cos \phi + \sin \psi \sin \phi \\ \cos \theta \sin \phi & \sin \psi \sin \theta \sin \phi + \cos \psi \cos \phi & \cos \psi \sin \theta \sin \phi - \sin \psi \cos \phi \\ -\sin \theta & \sin \psi \cos \theta & \cos \psi \cos \theta \end{bmatrix}$$

Given a rotation matrix  $R$ , we can compute the Euler angles,  $\psi$ ,  $\theta$ , and  $\phi$  by equating each element in  $R$  with the corresponding element in the matrix product  $R_z(\phi)R_y(\theta)R_x(\psi)$ . This results in nine equations that can be used to find the Euler angles.

Четвертая строка матрицы (transform\_data [3,:]) – это оффсет, абсолютное смещение снимка в пикселях по трём координатам.

Строка 5: фиксированные параметры равны нулю для всех снимков, не учитываем их.

Просматривать снимки оказалось очень удобно с помощью предложенного инструмента:



## II. Описание структуры решения.

Полный ход работы, которая выполнялась на локальной машине, показан в ноутбуке **main\_notebook.ipynb**.

В отдельные тетради с именами **XZ train in Colab.ipynb** и **YZ train in Colab.ipynb** вынесены те части работы, которые требовали долгих расчётов с GPU и выполнялись в Google Colab параллельно с основной работой.

Модуль **modules.py** – хранилище различных функций, которые оказалось удобнее убрать из основного ноутбука. Там содержатся следующие вещи:

- функции для проверки матрицы Эйлера, для перевода этой матрицы в углы поворота и обратно (использовались для проверки трансформаций);
- функция чтения текстовых файлов;
- функция для загрузки пары «снимок-разметка» и их относительного поворота;
- генератор батчей;
- функции сохранения-загрузки объектов в формате pickle;
- Loss-функция;
- класс с метриками;
- функция создания обучающего датасета из снимка.

Все функции снабжены необходимыми комментариями.

Модуль **baseline.py** – стартовый baseline для подбора гиперпараметров.

Папки:

- **train\_history** – история обучения;
- **unet\_keras** – U-net в фреймворке Keras;
- **source\_data** – исходные данные;
- **dataset** – хранилище подготовленных датасетов.

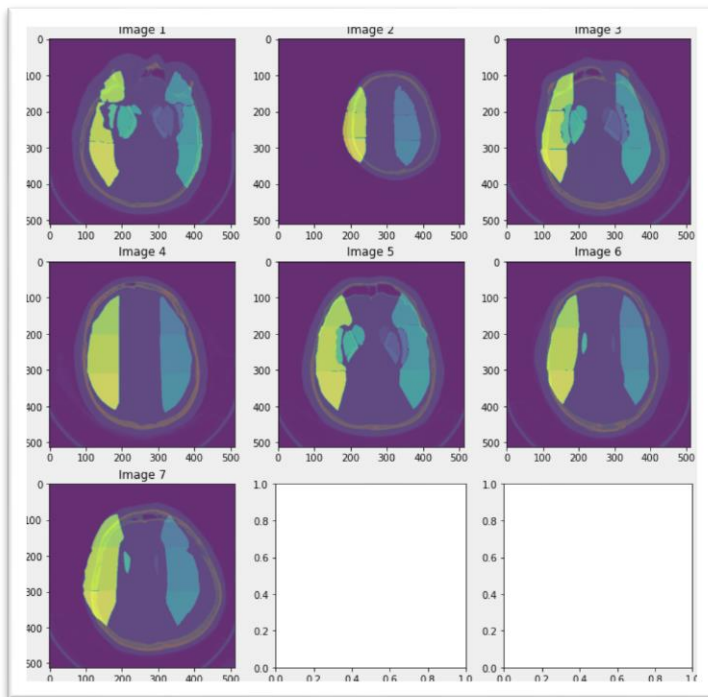
Веса обученной модели и предсказания хранятся в корневой папке для удобства работы.

## III. Взаимосвязь отчёта и ноутбуков

Нумерация разделов в отчёте совпадает с нумерацией разделов в **main\_notebook.ipynb**. Основная информация содержится в отчёте, в ноутбуке даны только краткие комментарии.

## 1. Анализ полученных данных

### 1.1 Пространственное положение снимков, определение направления поворота



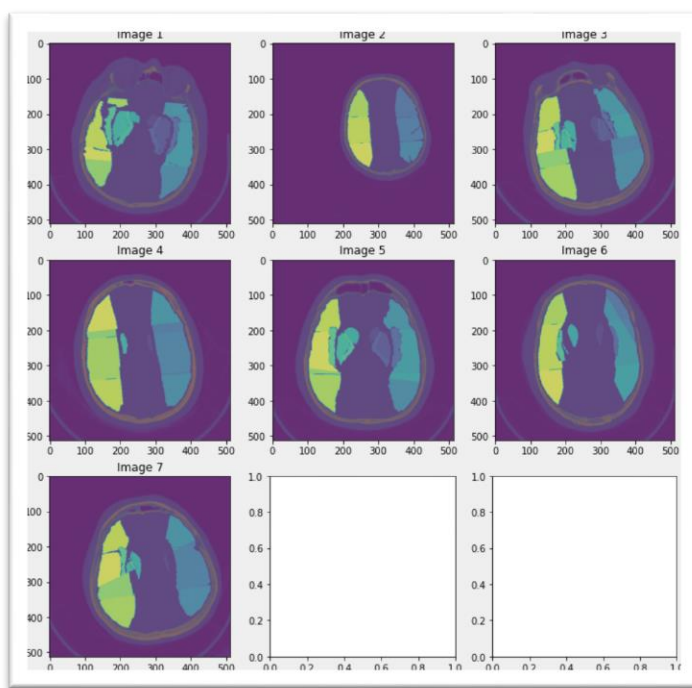
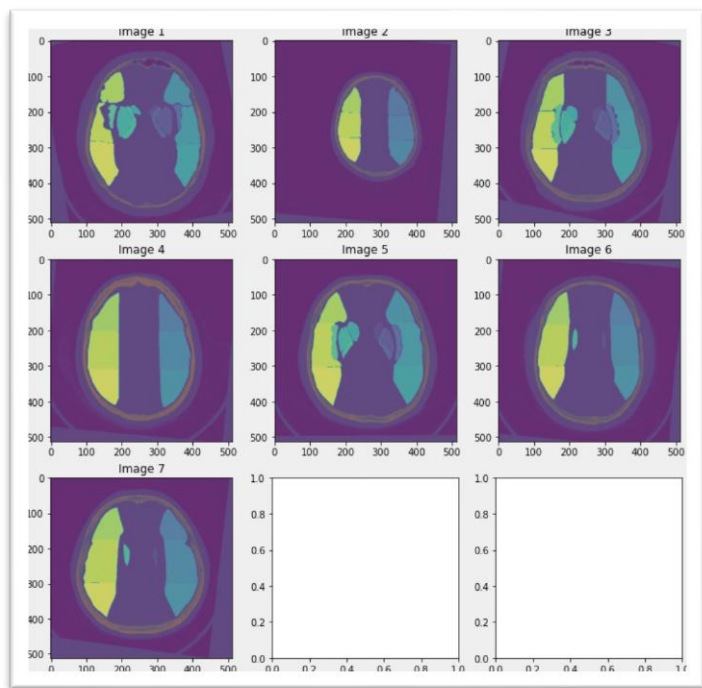
После написания функции загрузки снимков их можно визуализировать вместе с разметкой. Отчетливо видно смещение разметки относительно снимка.

Есть два варианта исправления смещения: довернуть снимок под разметку, или же довернуть разметку.

Первый вариант кажется менее приемлемым: при повороте содержимое снимка искажается, а поскольку в нём каждый пиксель содержит уникальную информацию, это может привести к ухудшенной сегментации.

Второй вариант выглядит более выигрышным, поскольку разметка содержит в себе значительно меньше информации, и небольшое искажение её границ не приведёт к серьёзным ухудшениям.

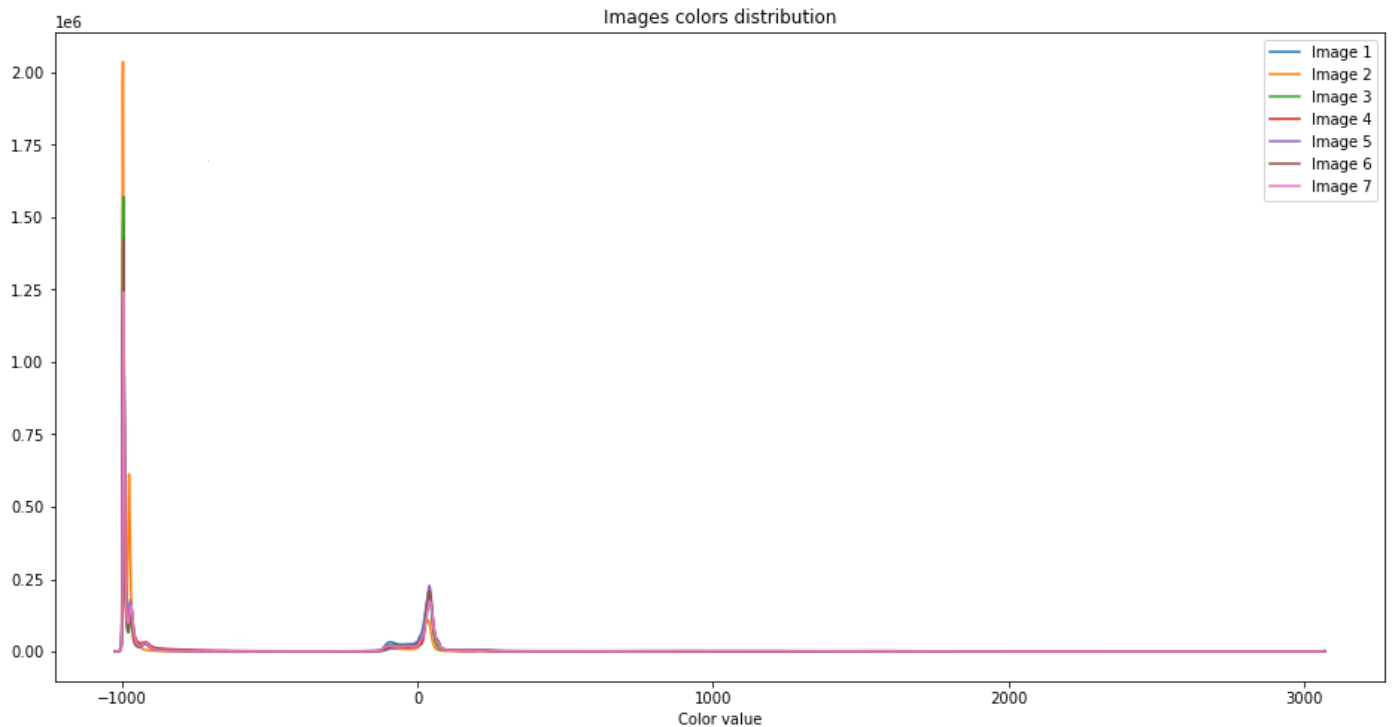
Оба эти варианта показаны на картинках ниже:



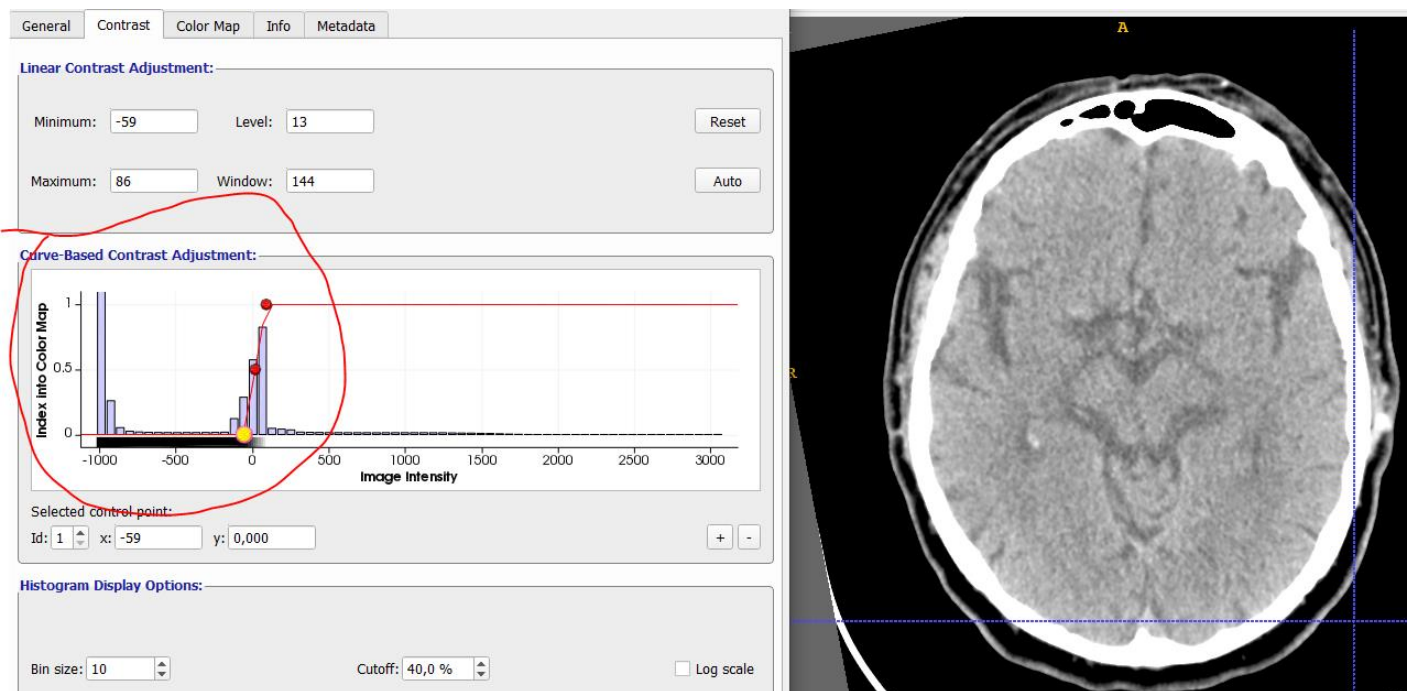
Если же вспомнить, что свёрточные сети инвариантны к размеру и смещению изображений, но чувствительны к их повороту, то решение оказывается очевидным: придётся смириться с частичной потерей информации и довернуть снимки так, чтобы они были ориентированы максимально одинаково. Поэтому *поворачиваем снимки*, а не разметку.

## 1.2 Оценка размерностей и динамического диапазона снимков, определение порядка препроцессинга.

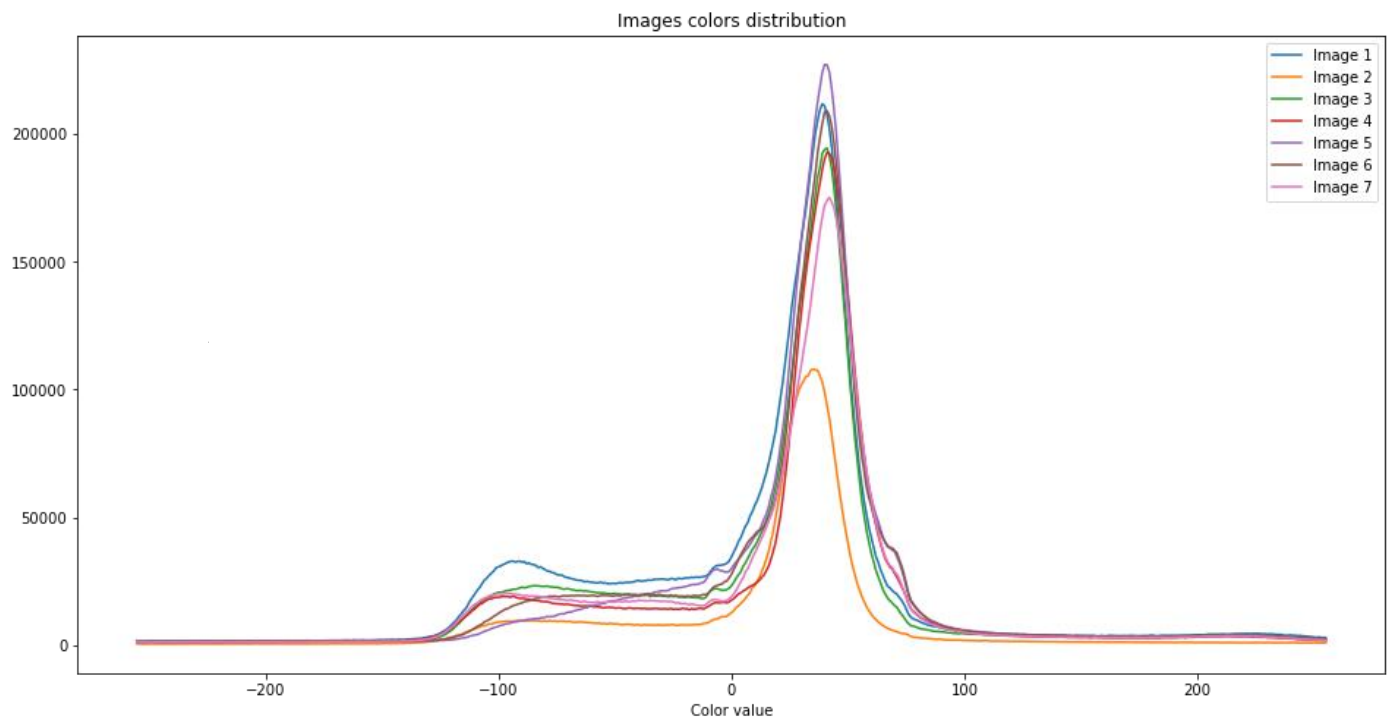
Посчитаем распределение количеств пикселей различных цветов. Цветовой диапазон снимков причудлив:



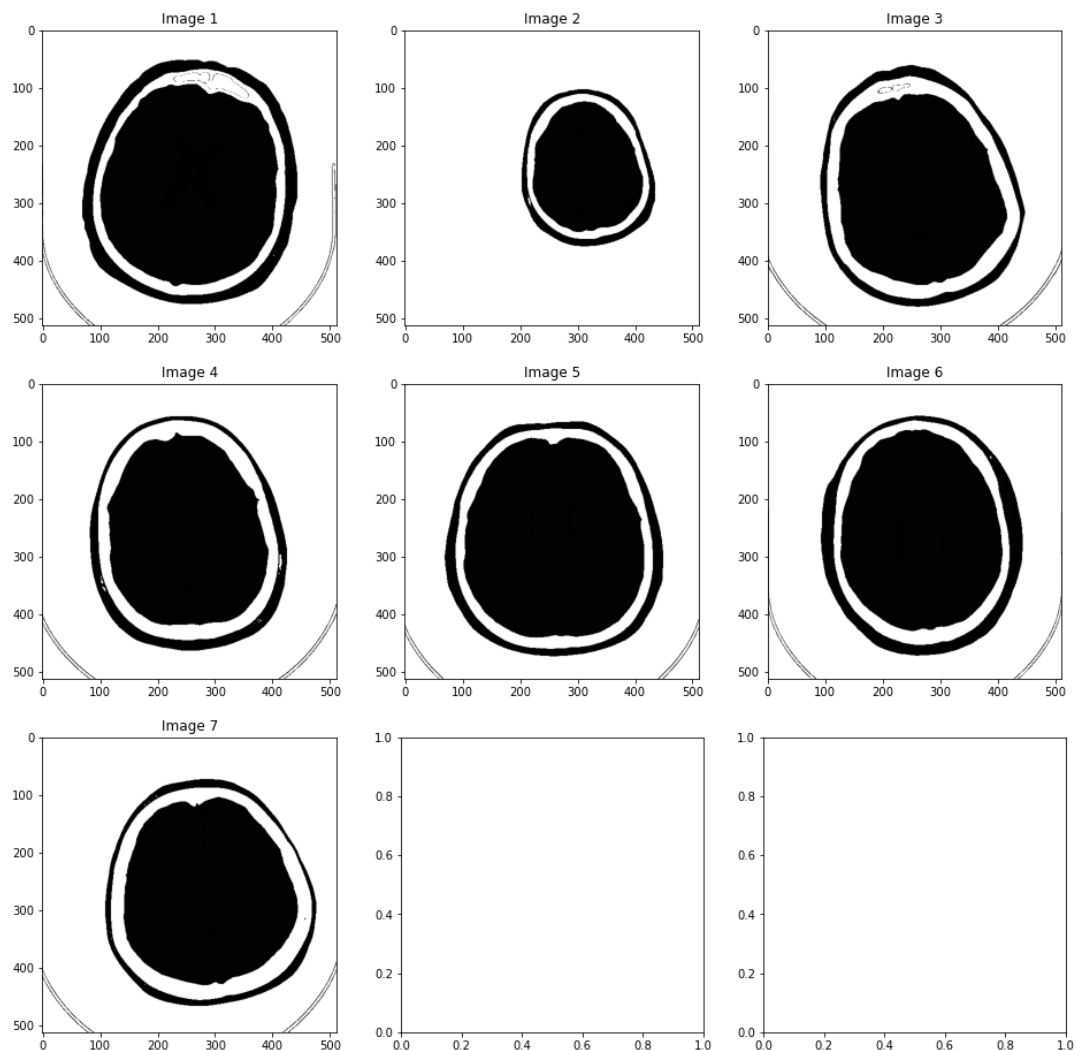
На диаграмме выше явно видны два пика плотности цветов. Визуальный анализ с помощью ITK-SNAP показывает, что интересующая нас информация сосредоточена в узком диапазоне цветов, а резко отрицательные и резко положительные значения цвета относятся к пикселям вне объекта на снимке – это фон:



Если убрать неинформативные цвета, то модели будет проще сконцентрироваться на сути снимка. Отбросим все пиксели вне диапазона(-255, 255) и увеличим масштаб графика.

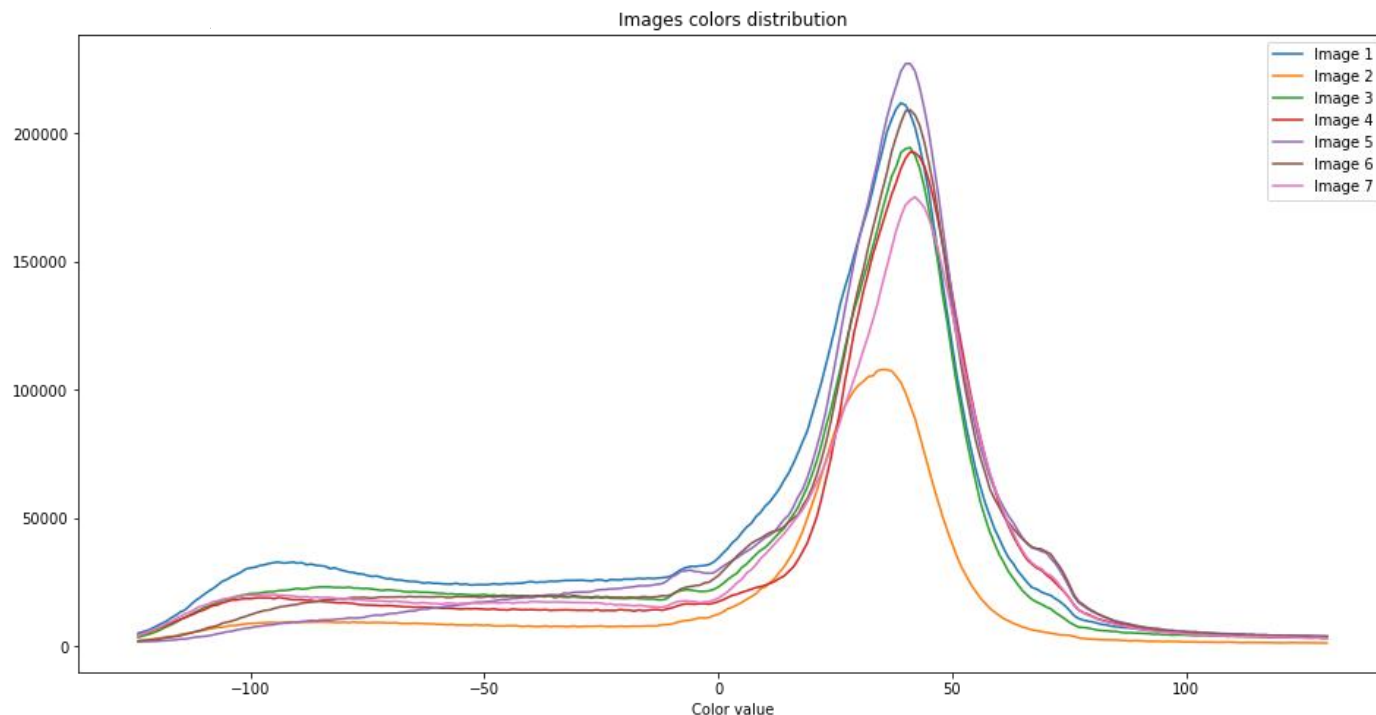


Даже в этом случае видно, что подавляющая часть информации сконцентрирована в ещё более узком окне. Логично бы было тогда уложить диапазон цветов в интервал  $-127 \dots 127$ , или  $0 \dots 255$ . Расчет показывает, что наивыгоднейшее окно – это диапазон цветов  $[-124, 130]$ , при этом потери минимальны и составляют 6% пикселей. Визуально можно грубо оценить, встречаются ли пиксели вне этого диапазона на интересующих нас областях снимков:



Не встречаются. Таким образом, можно ограничить цветовой диапазон окном  $[-124, 130]$ . Так и поступим, сохранив обработанные снимки в папку **dataset**. При этом нужно учесть, что при повороте снимка появятся дополнительные нулевые значения по краям. Каждый снимок будет обработан следующим образом:

- всем пикселям вне окна  $[-124, 130]$  присваиваем значение  $-124$ ;
- смещаем цвета в область положительных значений прибавкой оффсета в  $124$ ;
- сохраняем снимок;
- *нормируем снимок к единице и выполняем поворот только на этапе создания датасета. Если сделать это раньше, то сохраняемые файлы снимков из `int16` перейдут в `float16` и увеличатся примерно в 4 раза.*

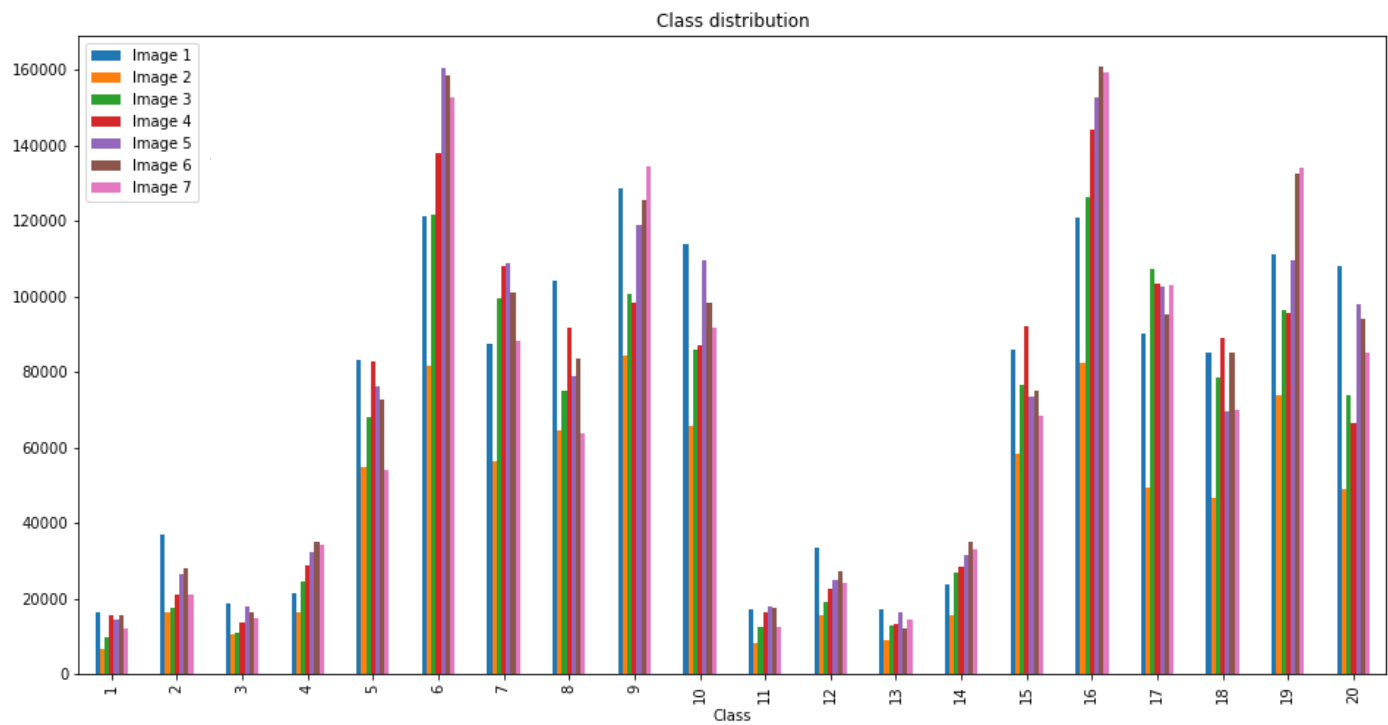


Снимки 2, 5 и 7 могут быть использованы только для обучения модели, поскольку их цветовые распределения в пике выбиваются из общей картины.



1.3 Оценка файлов с разметкой

Баланс классов крайне важен для успешного решения задач сегментации. В имеющихся же снимках есть изрядный дисбаланс, примерно одинаковый от снимка к снимку:



В математическом выражении дисбаланс классов превышает десятикратный:

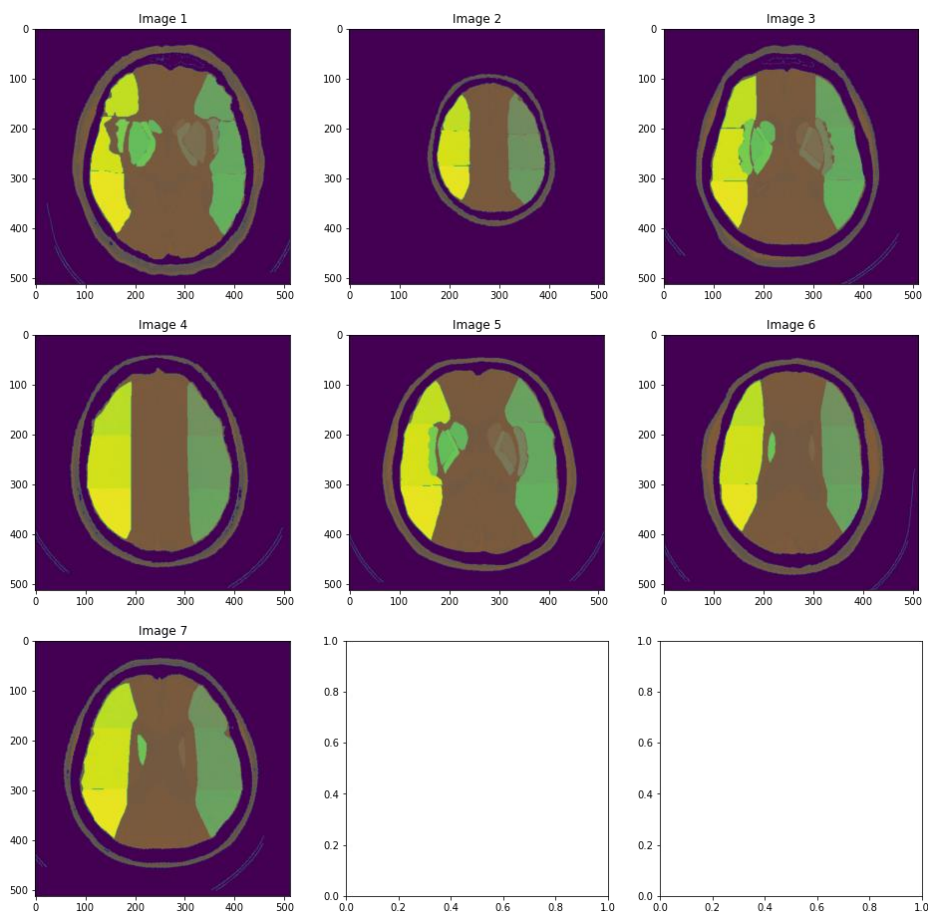
Maximum class disbalance: 10x.										
	Class	Image 1	Image 2	Image 3	Image 4	Image 5	Image 6	Image 7	Total	Class weight, %
1	1	16374	6792	9867	15368	14230	15561	12177	90369	0.99
2	2	37098	16353	17540	21166	26317	27807	20896	167177	1.82
3	3	18500	10418	10937	13568	17724	16367	14786	102300	1.12
4	4	21211	16453	24568	28776	32120	34882	34103	192113	2.10
5	5	83194	54840	68113	82804	76298	72637	54030	491916	5.37
6	6	121085	81470	121371	137754	160413	158263	152661	933017	10.18
7	7	87536	56257	99320	107837	108885	100960	88279	649074	7.08
8	8	104153	64315	74987	91574	78907	83620	63512	561068	6.12
9	9	128622	84392	100595	98314	118719	125368	134229	790239	8.62
10	10	113752	65730	85882	87106	109526	98261	91507	651764	7.11
11	11	17109	8080	12446	16312	17804	17582	12449	101782	1.11
12	12	33260	15700	19137	22733	25033	27376	23948	167187	1.82
13	13	17284	8839	12874	13292	16298	12109	14239	94935	1.04
14	14	23808	15464	26816	28526	31635	35017	33173	194439	2.12
15	15	85673	58420	76428	91896	73597	75044	68385	529443	5.78
16	16	120966	82286	126400	144234	152746	160821	159100	946553	10.33
17	17	90293	49308	107182	103316	102626	95097	102810	650632	7.10
18	18	85097	46702	78428	88908	69344	85213	69870	523562	5.71
19	19	110896	73953	96158	95619	109556	132328	133832	752342	8.21
20	20	107990	48819	73626	66444	97703	93949	85134	573665	6.26

Соответственно, придётся взять специфическую Loss-функцию, учитывающую редко встречающиеся классы.

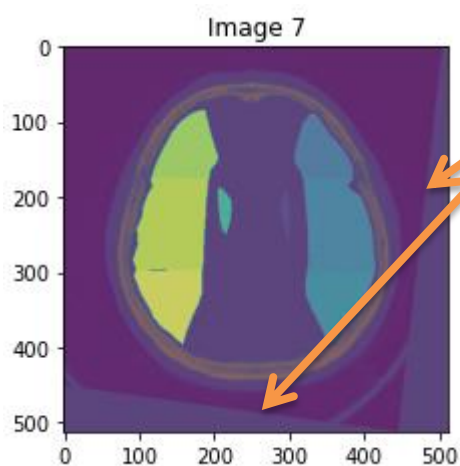
## 2. Препроцессинг данных

### 2.1 Преобразование снимков, сохранение в папку для датасета

Согласно выводам, сделанным в п.1, снимки были ограничены в цветовом диапазоне и сохранены вместе с разметкой в папке **dataset**. Чтобы убедиться, что всё сохранено корректно, можно посмотреть на снимки из этой папки:



Заметно, что теперь отсутствуют кромки повернутых снимков – они имеют нулевые значения и совпадают с фоном снимка:



Этих кромок теперь не видно.

### 3. Выбор Loss-функции и метрик

#### Loss-функция

Базовое решение в задачах мультиклассовой сегментации: CCELoss, но в данном домене велик дисбаланс классов, CCELoss в пробных обучении показал себя плохо, и оказалось разумнее взять относительно недавнюю модификацию Dice loss из <https://arxiv.org/pdf/1706.05721.pdf> под названием Tversky loss, которая довольно популярна в задачах сегментации медицинских изображений. Это модификация Dice Loss, заключающаяся в добавлении в знаменатель значений FP и FN с соответствующими коэффициентами, что позволяет отдельно и по-разному штрафовать модель за ложно-положительные и ложно-отрицательные предсказания:

The output layer in the network consists of  $c$  planes, one per class ( $c = 2$  in lesion detection). We applied softmax along each voxel to form the loss. Let  $P$  and  $G$  be the set of predicted and ground truth binary labels, respectively. The Dice similarity coefficient  $D$  between two binary volumes is defined as:

$$D(P, G) = \frac{2|PG|}{|P| + |G|} \quad (1)$$

If this is used in a loss layer in training [13], it weighs FPs and FNs (precision and recall) equally. In order to weigh FNs more than FPs in training our network for highly imbalanced data, where detecting small lesions is crucial, we propose a loss layer based on the Tversky index [19]. The Tversky index is defined as:

$$S(P, G; \alpha, \beta) = \frac{|PG|}{|PG| + \alpha|P \setminus G| + \beta|G \setminus P|} \quad (2)$$

where  $\alpha$  and  $\beta$  control the magnitude of penalties for FPs and FNs, respectively.

Коэффициенты альфа и бета будем подбирать в процессе обучения модели. Loss-функция находится в файле **modules.py**.

#### Метрики

На этапе тренировки:

- используем тот же **Tversky loss** как метрику процесса обучения сети, и метрики [**CCELoss**, **MSE**] для мониторинга процесса. **CCELoss** и **MSE** нужны для того, чтобы убедиться, что **Tversky loss** работает лучше них. К сожалению, метрику **IoU** на этапе обучения использовать не получилось – штатная имплементация её в Keras очень медленная и тормозит процесс обучения чуть ли не вдвое.

На этапе теста:

в данной [публикации](#) имеется [таблица](#), по которой можно косвенно оценить наиболее популярные метрики в данном домене. Чтобы не усложнять тестовую задачу, используем легко интерпретируемую метрику [Jaccard/Intersection over Union \(IoU\)](#). Для наглядности будем оценивать метрику в целом по снимку и отдельно по каждому классу.

Помимо **IoU**, хотелось бы оценивать поклассово частоту отнесения каждого пикселя к неверному классу. Для этого используем **Misclassification rate**, вычисленный как  $(FP+FN)/(TP+TN+FP+FN)$  отдельно для каждого класса. Источником этих данных будет confusion matrix.

Заключительная метрика, которая нужна: **FP/FN rate**. Поскольку в данном случае нет большой разницы в том, пропустит ли сеть активный пиксель, или интерпретирует ноль как какой-нибудь класс, представляется логичным соблюдать симметрию ложно-положительных и ложно-отрицательных предсказаний (мне видится, что такой ход не сработает, если предсказывать, например, опухоли – там важнее, вероятно, сместить предсказания подальше от ложно-отрицательных?). Поэтому будем стремиться **FP/FN rate** к единице.

Техническая реализация: создан класс Metrics, экземпляр которого при инициализации сразу вычисляет все нужные нам метрики и заодно - confusion matrix. В качестве параметров инициализации экземпляра класса выступают текущие `y_true` и `y_predicted`.

#### 4. Выбор архитектуры сети

Задача: провести сегментацию трехмерного изображения по двадцати классам.

Ограничения:

- 1 (один) локальный PC с 16 Гб оперативной памяти и Geforce RTX 1070 с 8 Гб видеопамати, плюс иногда работающая виртуальная машина в Google Colab.
- ограниченное время для экспериментов с архитектурами: срок сдачи задания ~ 7.02.2021.

Вариант 1: решение «в лоб». Возьмём Unet-подобную архитектуру с трёхмерными свёрточными слоями и будем загружать в модель снимки целиком.

Плюсы решения - простая конфигурация сети, несложная подготовка данных, модель видит трёхмерные взаимосвязи между пикселями и максимально хорошо проведёт сегментацию.

Минусы: даже базовая архитектура Unet с таким количеством параметров не помещается в память.

Итог: решение нереализуемо.

Вариант 2: отмасштабировать снимки до того размера, чтобы сеть для их обработки могла уместиться в памяти.

Плюсы: простая конфигурация сети, несложная подготовка данных, модель видит трёхмерные взаимосвязи между пикселями.

Минусы: потеря информации при сжатии, есть вероятность того, что сеть будет плохо выполнять сегментацию.

Вариант 3: проводить обработку снимков небольшими 3D «кубиками» такого размера, который позволит сети уместиться в памяти.

Плюсы: простая конфигурация сети, модель видит трёхмерные взаимосвязи между пикселями и максимально хорошо проведёт сегментацию.

Минусы: усложненная подготовка данных и генератора батчей; есть риск, что «кубики» могут получиться настолько разнородными, что сеть не сможет найти общие признаки в них; есть небольшой риск того, что при «склейке» предсказанных кубиков классы по границам могут не совпасть (по моему опыту для 2D изображений – это маловероятно, но риск есть.)

Вариант 4: выполнить 2D сегментацию по слоям изображения по оси Z, потом «склеить» эти слои в 3D.

Плюсы: очень простая архитектура сети, сеть гарантированно поместится в память; разумное время на обучение сети.

Минусы: заведомо низкакачественная сегментация, поскольку модель не видит трёхмерных зависимостей; сложная подготовка обучающего датасета; тот же риск несовпадения классов при склейке слоёв.

Вариант 5: скачать какой-либо готовый репозиторий с предобученной моделью, например, [этот](#) или [этот](#), и дообучить модель на своих данных.

Плюсы: вероятно, получится хороший результат; не нужно с нуля реализовывать архитектуру.

Минусы: совершенно неизвестно, заработает ли клонированная модель; непонятны затраты времени на анализ архитектуры проекта и на корректировку кода под мои нужды; решение не совсем удовлетворяет цели тестового задания.

С учётом имеющихся ограничений выбран вариант 4 с модификацией. Суть метода: будем делать 2D сегментацию, но поочерёдно в трёх плоскостях: XY, XZ и YZ. Затем совместим полученные сегментации по простейшему принципу: если в двух из трёх пикселей указан один и тот же класс, то этот класс идёт в итоговую сегментацию, иначе туда идёт 0. Архитектура сети может быть одна и та же для всех трёх плоскостей, а вот обучающие датасеты и веса сети будут разными.

Времени на реализацию тестового задания отведено не очень много, поэтому я выбрал классический U-net в качестве модели. Обоснование: у меня не было никакой уверенности в том, что архитектура со сложным энкодером и большим количеством параметров а) будет успешно работать на моем компьютере и б) обучится в разумные сроки.

В итоге архитектура сети сводится к следующему: на вход мы подаём двумерные срезы трёхмерного снимка, на выходе имеем трехмерный массив с сегментацией - трёхмерный потому, что двумерный срез сегментации с численными метками классов необходимо заменить на one-hot encoding для каждого класса.

Пример: обучаем сеть на 3D снимке размером 128x128x32, и делаем сечения вдоль оси Z. Тогда обучающие двумерные срезы снимка будут иметь размерность 128x128, и будет их 32 штуки. Обучающая сегментация же будет иметь размер 128x128x20, где 20 – число классов. Соответственно, сеть имеет одноканальный вход и двадцатиканальный выход.

Регуляризацию сети выполняем с помощью слоёв GaussianNoise() на входе и Dropout() в конце кодирующей части сети, перед «бутылочным горлышком». Вот полная структура сети:

Layer (type)	Output Shape	Param #	Connected to
input 1 (InputLayer)	(None, 512, 512, 1)	0	
gaussian_noise (GaussianNoise)	(None, 512, 512, 1)	0	input_1[0][0]
conv2d (Conv2D)	(None, 512, 512, 32)	320	gaussian_noise[0][0]
batch_normalization (BatchNormaliza	(None, 512, 512, 32)	128	conv2d[0][0]
activation (Activation)	(None, 512, 512, 32)	0	batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 512, 512, 32)	9248	activation[0][0]
batch_normalization_1 (BatchNor	(None, 512, 512, 32)	128	conv2d_1[0][0]
activation_1 (Activation)	(None, 512, 512, 32)	0	batch_normalization_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 256, 256, 32)	0	activation_1[0][0]
conv2d_2 (Conv2D)	(None, 256, 256, 64)	18496	max_pooling2d[0][0]
batch_normalization_2 (BatchNor	(None, 256, 256, 64)	256	conv2d_2[0][0]
activation_2 (Activation)	(None, 256, 256, 64)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 256, 256, 64)	36928	activation_2[0][0]
batch_normalization_3 (BatchNor	(None, 256, 256, 64)	256	conv2d_3[0][0]
activation_3 (Activation)	(None, 256, 256, 64)	0	batch_normalization_3[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 64)	0	activation_3[0][0]
conv2d_4 (Conv2D)	(None, 128, 128, 128)	73856	max_pooling2d_1[0][0]
batch_normalization_4 (BatchNor	(None, 128, 128, 128)	512	conv2d_4[0][0]
activation_4 (Activation)	(None, 128, 128, 128)	0	batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 128, 128, 128)	147584	activation_4[0][0]
batch_normalization_5 (BatchNor	(None, 128, 128, 128)	512	conv2d_5[0][0]
activation_5 (Activation)	(None, 128, 128, 128)	0	batch_normalization_5[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 128)	0	activation_5[0][0]
conv2d_6 (Conv2D)	(None, 64, 64, 256)	295168	max_pooling2d_2[0][0]
batch_normalization_6 (BatchNor	(None, 64, 64, 256)	1024	conv2d_6[0][0]
activation_6 (Activation)	(None, 64, 64, 256)	0	batch_normalization_6[0][0]
conv2d_7 (Conv2D)	(None, 64, 64, 256)	590080	activation_6[0][0]
batch_normalization_7 (BatchNor	(None, 64, 64, 256)	1024	conv2d_7[0][0]
activation_7 (Activation)	(None, 64, 64, 256)	0	batch_normalization_7[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 32, 32, 256)	0	activation_7[0][0]
conv2d_8 (Conv2D)	(None, 32, 32, 512)	1180160	max_pooling2d_3[0][0]
batch_normalization_8 (BatchNor	(None, 32, 32, 512)	2048	conv2d_8[0][0]
activation_8 (Activation)	(None, 32, 32, 512)	0	batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 32, 32, 512)	2359808	activation_8[0][0]
batch_normalization_9 (BatchNor	(None, 32, 32, 512)	2048	conv2d_9[0][0]
activation_9 (Activation)	(None, 32, 32, 512)	0	batch_normalization_9[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 16, 16, 512)	0	activation_9[0][0]
conv2d_10 (Conv2D)	(None, 16, 16, 1024)	4719616	max_pooling2d_4[0][0]
batch_normalization_10 (BatchNo	(None, 16, 16, 1024)	4096	conv2d_10[0][0]
activation_10 (Activation)	(None, 16, 16, 1024)	0	batch_normalization_10[0][0]
conv2d_11 (Conv2D)	(None, 16, 16, 1024)	9438208	activation_10[0][0]
batch_normalization_11 (BatchNo	(None, 16, 16, 1024)	4096	conv2d_11[0][0]
activation_11 (Activation)	(None, 16, 16, 1024)	0	batch_normalization_11[0][0]
up_sampling2d (UpSampling2D)	(None, 32, 32, 1024)	0	activation_11[0][0]
concatenate (Concatenate)	(None, 32, 32, 1536)	0	up_sampling2d[0][0] activation_9[0][0]
conv2d_12 (Conv2D)	(None, 32, 32, 512)	7078400	concatenate[0][0]
batch_normalization_12 (BatchNo	(None, 32, 32, 512)	2048	conv2d_12[0][0]
activation_12 (Activation)	(None, 32, 32, 512)	0	batch_normalization_12[0][0]
conv2d_13 (Conv2D)	(None, 32, 32, 512)	2359808	activation_12[0][0]
batch_normalization_13 (BatchNo	(None, 32, 32, 512)	2048	conv2d_13[0][0]
activation_13 (Activation)	(None, 32, 32, 512)	0	batch_normalization_13[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 64, 64, 512)	0	activation_13[0][0]
concatenate_1 (Concatenate)	(None, 64, 64, 768)	0	up_sampling2d_1[0][0] activation_7[0][0]
conv2d_14 (Conv2D)	(None, 64, 64, 256)	1769728	concatenate_1[0][0]
batch_normalization_14 (BatchNo	(None, 64, 64, 256)	1024	conv2d_14[0][0]
activation_14 (Activation)	(None, 64, 64, 256)	0	batch_normalization_14[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 256)	590080	activation_14[0][0]
batch_normalization_15 (BatchNo	(None, 64, 64, 256)	1024	conv2d_15[0][0]
activation_15 (Activation)	(None, 64, 64, 256)	0	batch_normalization_15[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 128, 128, 256)	0	activation_15[0][0]
concatenate_2 (Concatenate)	(None, 128, 128, 384)	0	up_sampling2d_2[0][0] activation_5[0][0]
conv2d_16 (Conv2D)	(None, 128, 128, 128)	442496	concatenate_2[0][0]
batch_normalization_16 (BatchNo	(None, 128, 128, 128)	512	conv2d_16[0][0]
activation_16 (Activation)	(None, 128, 128, 128)	0	batch_normalization_16[0][0]
conv2d_17 (Conv2D)	(None, 128, 128, 128)	147584	activation_16[0][0]
batch_normalization_17 (BatchNo	(None, 128, 128, 128)	512	conv2d_17[0][0]
activation_17 (Activation)	(None, 128, 128, 128)	0	batch_normalization_17[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 256, 256, 128)	0	activation_17[0][0]
concatenate_3 (Concatenate)	(None, 256, 256, 192)	0	up_sampling2d_3[0][0] activation_3[0][0]
conv2d_18 (Conv2D)	(None, 256, 256, 64)	110656	concatenate_3[0][0]
batch_normalization_18 (BatchNo	(None, 256, 256, 64)	256	conv2d_18[0][0]
activation_18 (Activation)	(None, 256, 256, 64)	0	batch_normalization_18[0][0]
conv2d_19 (Conv2D)	(None, 256, 256, 64)	36928	activation_18[0][0]
batch_normalization_19 (BatchNo	(None, 256, 256, 64)	256	conv2d_19[0][0]
activation_19 (Activation)	(None, 256, 256, 64)	0	batch_normalization_19[0][0]
up_sampling2d_4 (UpSampling2D)	(None, 512, 512, 64)	0	activation_19[0][0]
concatenate_4 (Concatenate)	(None, 512, 512, 96)	0	up_sampling2d_4[0][0] activation_1[0][0]
conv2d_20 (Conv2D)	(None, 512, 512, 32)	27680	concatenate_4[0][0]
batch_normalization_20 (BatchNo	(None, 512, 512, 32)	128	conv2d_20[0][0]
activation_20 (Activation)	(None, 512, 512, 32)	0	batch_normalization_20[0][0]
conv2d_21 (Conv2D)	(None, 512, 512, 32)	9248	activation_20[0][0]
batch_normalization_21 (BatchNo	(None, 512, 512, 32)	128	conv2d_21[0][0]
activation_21 (Activation)	(None, 512, 512, 32)	0	batch_normalization_21[0][0]
conv2d_22 (Conv2D)	(None, 512, 512, 20)	660	activation_21[0][0]
Total params: 31,466,804			
Trainable params: 31,454,772			
Non-trainable params: 12,032			

## Деление на тренировочные, валидационные и тестовые данные.

Для определенности примем, что обучаем и валидируем сеть на снимках №2...7, а тестовым выступает снимок №1.

При подготовке обучающих данных все двумерные срезы всех обучающих снимков поместим в общий массив, потом сгенерируем случайные индексы и поделим в пропорции 80/20, получив, таким образом, обучающую и валидационную выборки.

При предсказании разметки на тестовом снимке мы точно так же разобьём его на срезы и предскажем каждый срез отдельно (или батчами по несколько штук), а результат соберем обратно в трёхмерный массив.

## Кросс-валидация

Мне не очень нравится схема, когда параметры сети оцениваются по единственному снимку. Идеально было бы применить кросс-валидацию, но обучение большего числа моделей явно не уложится в имеющиеся временные рамки.

## Общий порядок работы



## 5. Подготовка датасетов

### Требования к обучающим данным следующие:

- должно быть три различных датасета, по одному для плоскостей XY, YZ и XZ;
- в каждом из них должно содержаться N двумерных срезов трёхмерного снимка, где N – размерность снимка по оси, вдоль которой производятся сечения, и N срезов сегментации;
- поскольку от слоя к слою сети каждая из размерностей снимков уменьшается вдвое, а затем обратно удваивается, размерности снимков должны быть кратны степеням двойки.
- данные должны храниться в массивах соответствующих размерностей и, по возможности, целиком размещаться в памяти.

### Реализация:

Подготовку датасета выполняет функция **create\_dataset()** в таком порядке:

- поочередно открывает все обучающие пары файлов: предобработанные снимки и их разметку;
- дополняет снимки в размерности Z так, чтобы их размер стал 512x512x128;
- нормирует снимки к единице;
- собирает N срезов снимка в массив **"X"**, N срезов разметки перекодирует в One-hot и собирает в массив **"y"**;
- сохраняет на жёсткий диск отдельными файлами в формате pickle.

### Пример:

Делаем предсказания в плоскости XY, срезы, соответственно, вдоль оси Z. Размеры снимков 512x512x(98...116). После исполнения функции **create\_dataset()** получим массив **"X"** с размерностями:










**"X"** = (число файлов\*число срезов, размерность снимка X, размерность снимка X, 1)=((6\*128), 512, 512, 1),

...и массив **"y"**:

**"y"** = (число файлов\*число срезов, размерность снимка X, размерность снимка X, число классов)=((6\*128), 512, 512, 20).

Массивы будут сохранены в файлы «XY\_X.pkl», «XY\_y1.pkl», «XY\_y2.pkl». Массив при перекодировании в One-hot сильно разрастается в объёме, а формат pickle не предполагает сжатия данных, поэтому массив пришлось разбивать на два файла.

Вот так выглядит папка с подготовленными датасетами:

 XY_X.pkl	PKL File	786 433 KB
 XY_y1.pkl	PKL File	1 966 081 KB
 XY_y2.pkl	PKL File	1 966 081 KB
 XZ_X.pkl	PKL File	786 433 KB
 XZ_y1.pkl	PKL File	1 966 081 KB
 XZ_y2.pkl	PKL File	1 966 081 KB
 YZ_X.pkl	PKL File	786 433 KB
 YZ_y1.pkl	PKL File	1 966 081 KB
 YZ_y2.pkl	PKL File	1 966 081 KB

Для загрузки данных в сеть используем генератор батчей **BatchGenerator()**.

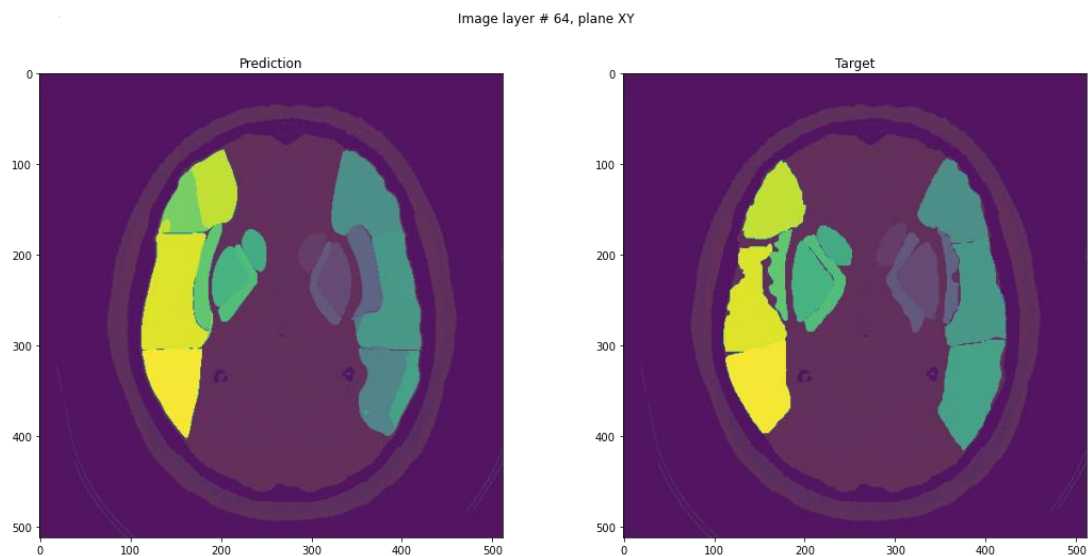
## 6. Обучение сети и сегментация снимков в плоскости XY

Начинаем первое обучение сети с умеренными гиперпараметрами:  $\alpha=\beta=0.5$ ,  $\text{dropout}=0.2$ . Максимальный размер батча, который можно получить на моем компьютере, равен 4.

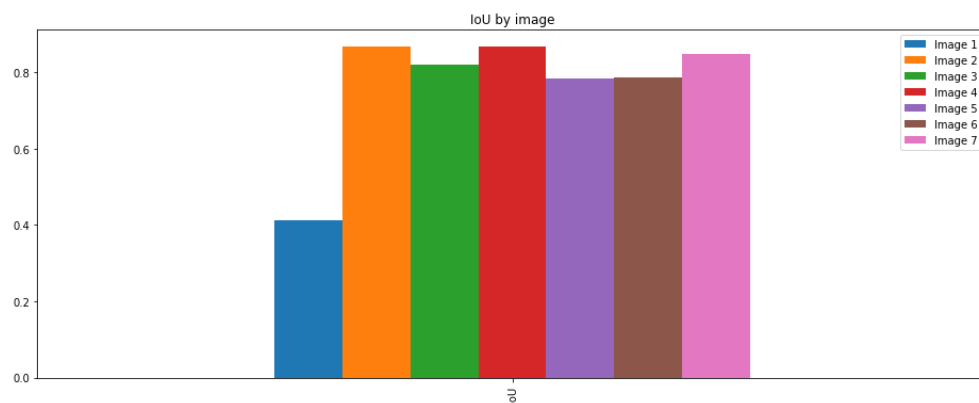
После этого обучения можно будет установить baseline метрики и выяснить, в какую сторону двигать гиперпараметры.

История обучения: раздел 6.3 основного ноутбука.

Результаты: ура, подход работает!



Что с основной метрикой? Смотрим раздел 6.6:



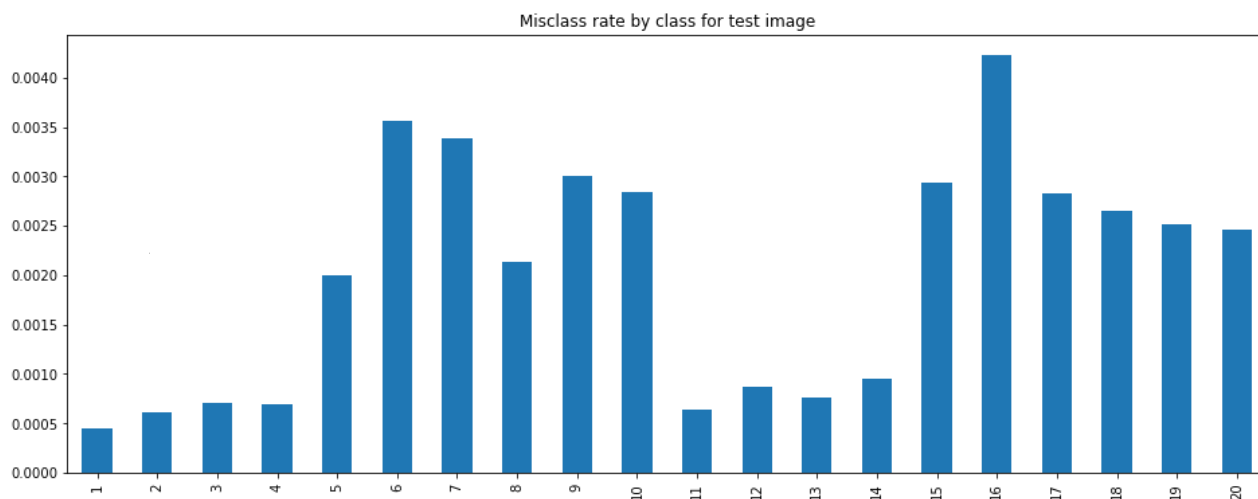
IoU для тестового снимка вдвое ниже, чем для тех, которые сеть уже видела. Это оверфит, значит, можно попробовать добавить dropout.

Можно оценить, как варьируется IoU от класса к классу для всех снимков – примерно одинаковая разница между тестовым и обучающими файлами:

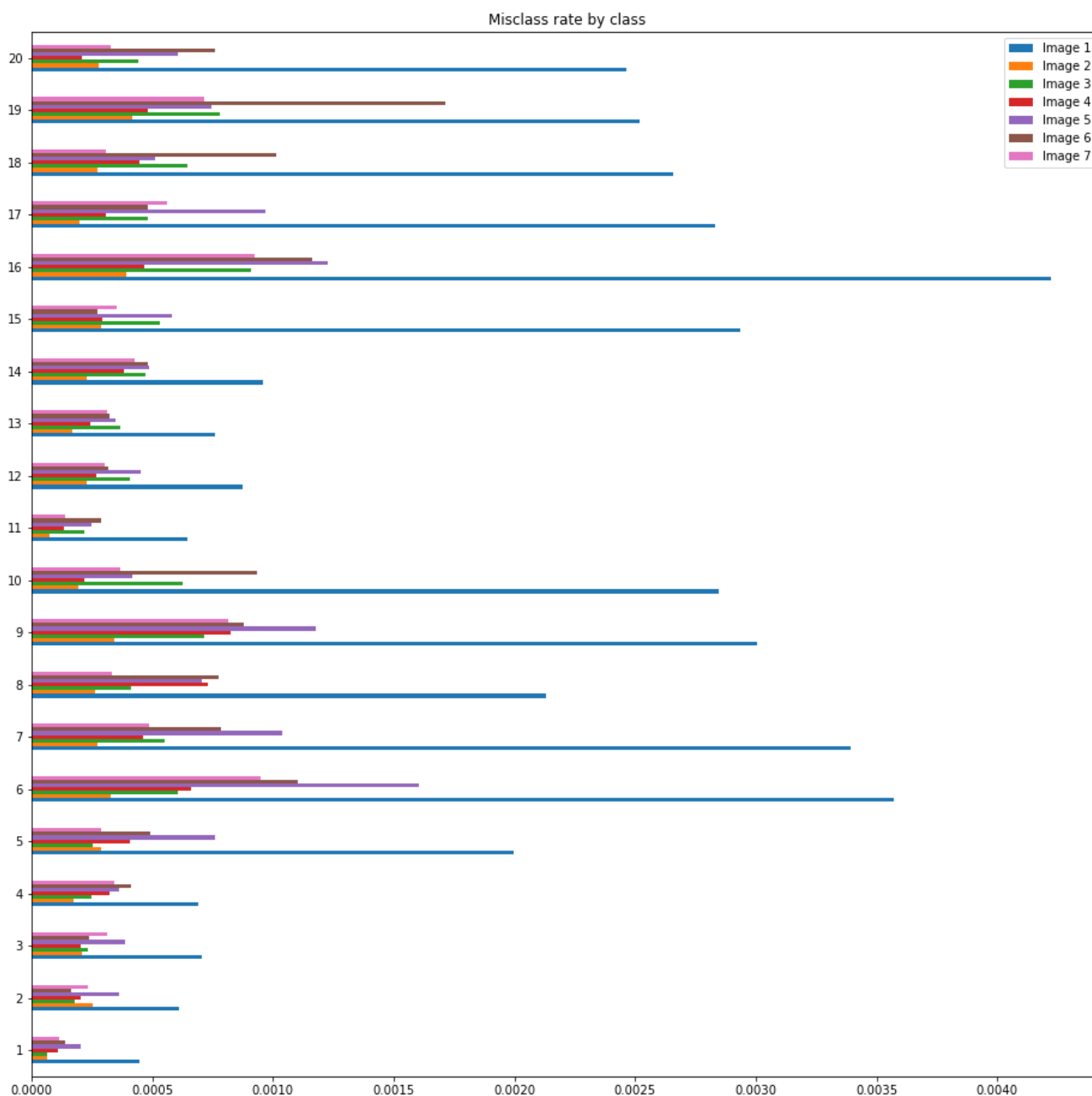




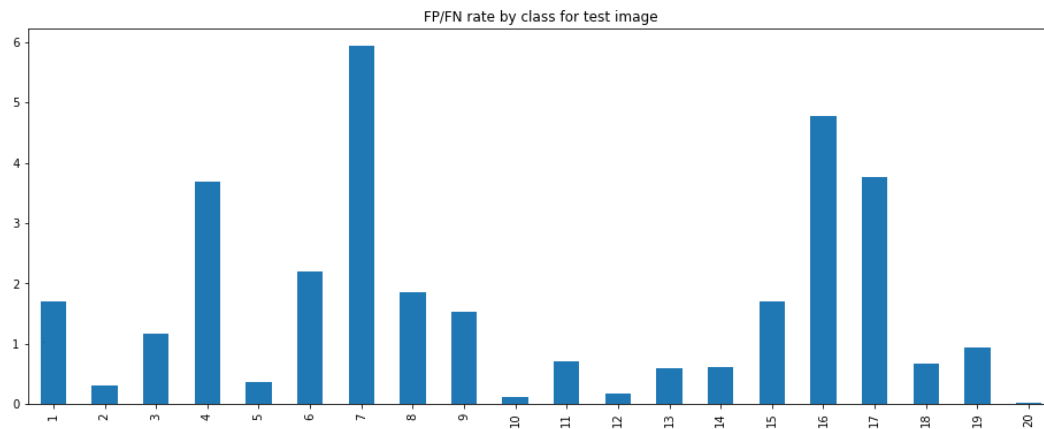
Уровень ошибочно классифицируемых пикселей невелик в абсолютном измерении, но различается в несколько раз от класса к классу:



...и он сильно превышает аналогичную метрику для обучающих снимков:



И, наконец, баланс ложно-положительных и ложно-отрицательных предсказаний по классам имеет сильную дисперсию:

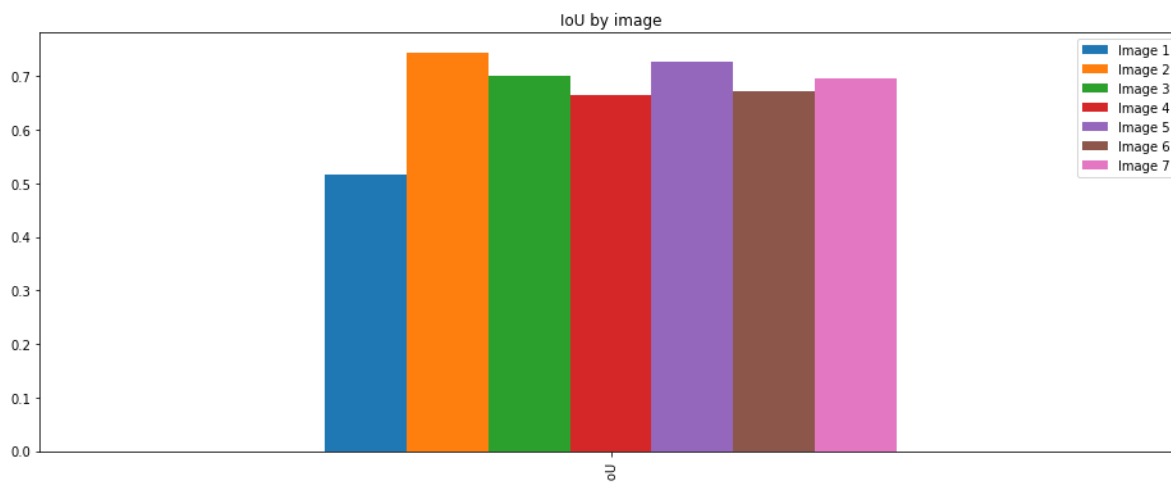


Численные метрики сохраняем как baseline, с которым будем сравнивать все следующие достижения:

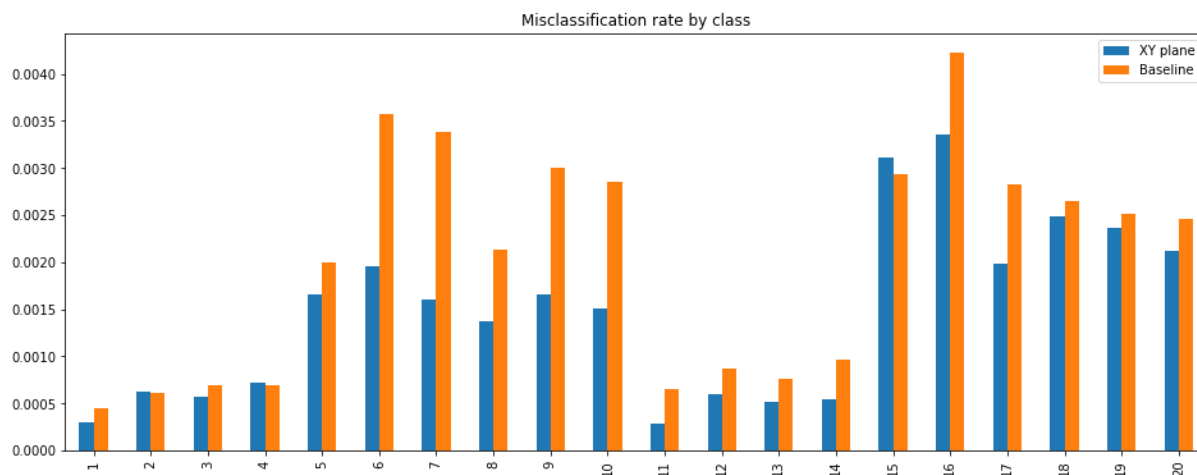
```
Test image baseline:
IoU 0.41,
FP/FN rates [1.70202 0.30919 1.17027 3.68475 0.37258 2.20041 5.93092 1.84582 1.52233
0.11056 0.71488 0.17497 0.598 0.61844 1.69429 4.78197 3.76452 0.66899
0.92979 0.02572]
Test image misclassification rate baseline:
[0.00045 0.00061 0.0007 0.00069 0.002 0.00357 0.00339 0.00213 0.003
0.00285 0.00065 0.00087 0.00076 0.00096 0.00294 0.00422 0.00283 0.00265
0.00252 0.00246]
```

Поскольку предсказания сети имеют смещение в сторону ложно-положительных, для повторного обучения установим **alpha=0.85**, **beta=0.15**, а для уменьшения оверфита поднимем dropout до 0.3. Смотрим **раздел 6.7**:

IoU существенно вырос:



А уровень ошибок классификации – снизился:



В итоге оказывается, что с такими гиперпараметрами:

- IoU улучшилась примерно на 20%,
- дисперсия и среднее FP/FN rate по классам снизились,
- немного упал уровень ошибочной классификации, и стал примерно равен этому же параметру для обучающих снимков.

Оверфит тоже уменьшился, поскольку разница в IoU для обучающих и тестового снимка снизилась:

(В параметры распределения misclassification rate введены множители  $1e3$  и  $1e6$  сугубо для красивого отображения значений)

For test image:

IoU:: baseline 0.41, improved 0.51

Variance of FP/FN rates between classes:: baseline: 2.62, improved: 1.49

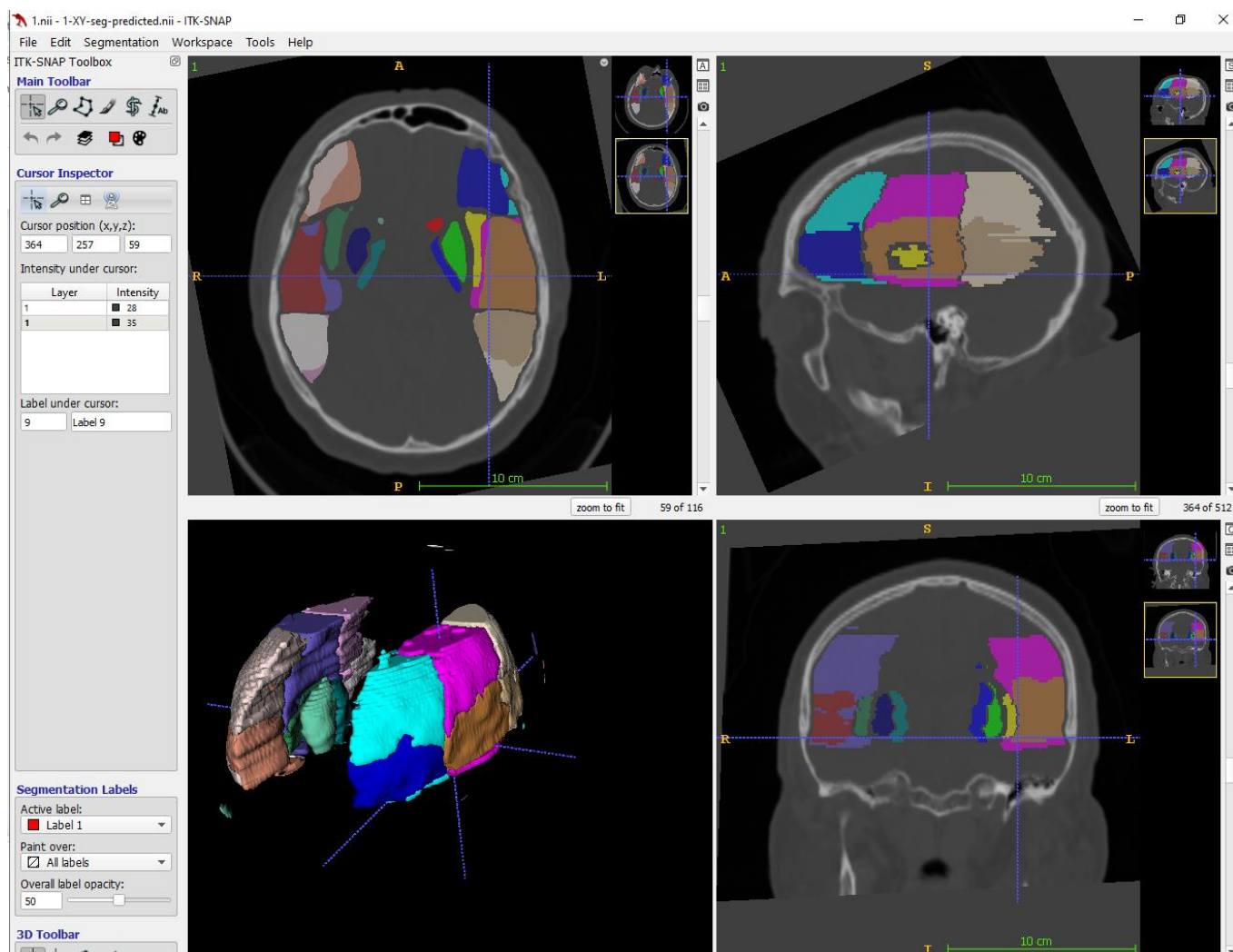
Mean of FP/FN rates between classes:: baseline: 1.64, improved: 1.09

Variance of misclassification rates between classes,  $*1e-6$ :: baseline: 1.35, improved: 0.82

Mean of misclassification rates between classes,  $*1e-3$ :: baseline: 2.01, improved: 1.47

\*\*\*

Результат сегментации вполне себе удовлетворительный, хоть и не очень похож на правду для редко встречающихся классов:



Ввиду ограниченного времени подбор гиперпараметров для плоскости XY на этом закончим.

## 7. Сегментация в плоскостях YZ и XZ

Размер среза в этих плоскостях: 512x128, в то время как в плоскости XY он составлял 512x512.

Обучение проводил в Google Colab, история доступна в ноутбуках **XZ train in Colab.ipynb** и **YZ train in Colab.ipynb**.

### Обучение YZ

Технически оно проводилось раньше, чем повторное обучение XY, на нем проводился первичный подбор гиперпараметров, и в итоге я остановился на:  $\alpha=0.8$ ,  $\beta=0.2$ ,  $\text{dropout}=0.3$ . Не стану загромождать отчёт повторяющимися графиками – они доступны в основном ноутбуке, приведу лишь результат:

```
For test image:
```

```
IoU:: baseline 0.41, YZ plane 0.43
```

```
Variance of FP/FN rates between classes:: baseline: 2.62, YZ plane: 0.29
```

```
Mean of FP/FN rates between classes:: baseline: 1.64, YZ plane: 0.73
```

```
Variance of misclassification rates between classes, *1e-6:: baseline: 1.35, YZ plane: 0.85
```

```
Mean of misclassification rates between classes, *1e-3:: baseline: 2.01, YZ plane: 1.61
```

Качество классификации хуже, чем в плоскости XY, вероятнее всего, просто потому, что площадь снимка в этом сечении в четыре раза меньше.

Для проверки я попробовал  $\alpha=0.7$ ,  $\beta=0.3$  и убедился, что результат хуже, как и следовало ожидать:

```
For test image:
```

```
IoU:: baseline 0.41, YZ 0.31
```

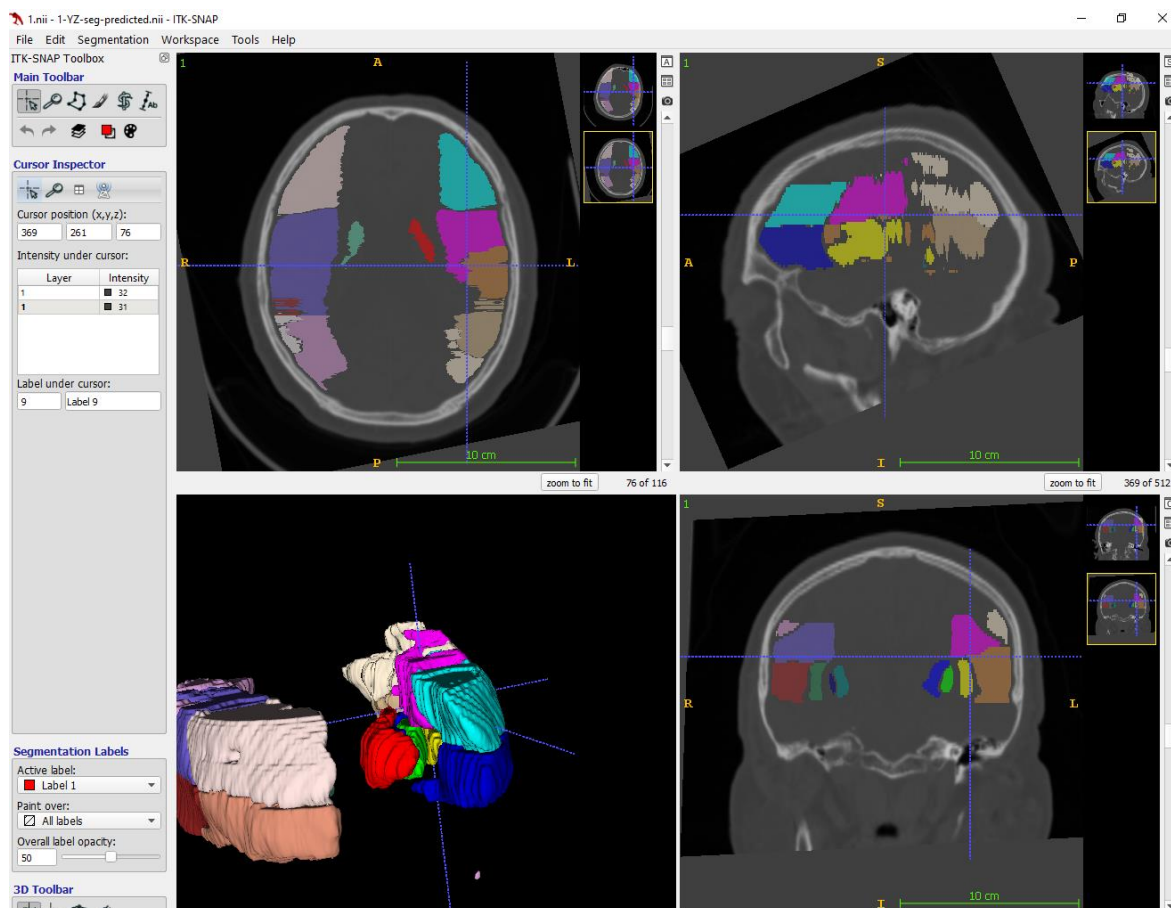
```
Variance of FP/FN rates between classes:: baseline: 2.62, YZ: 0.27
```

```
Mean of FP/FN rates between classes:: baseline: 1.64, YZ: 0.72
```

```
Variance of misclassification rates between classes, *1e-6:: baseline: 1.35, YZ: 0.67
```

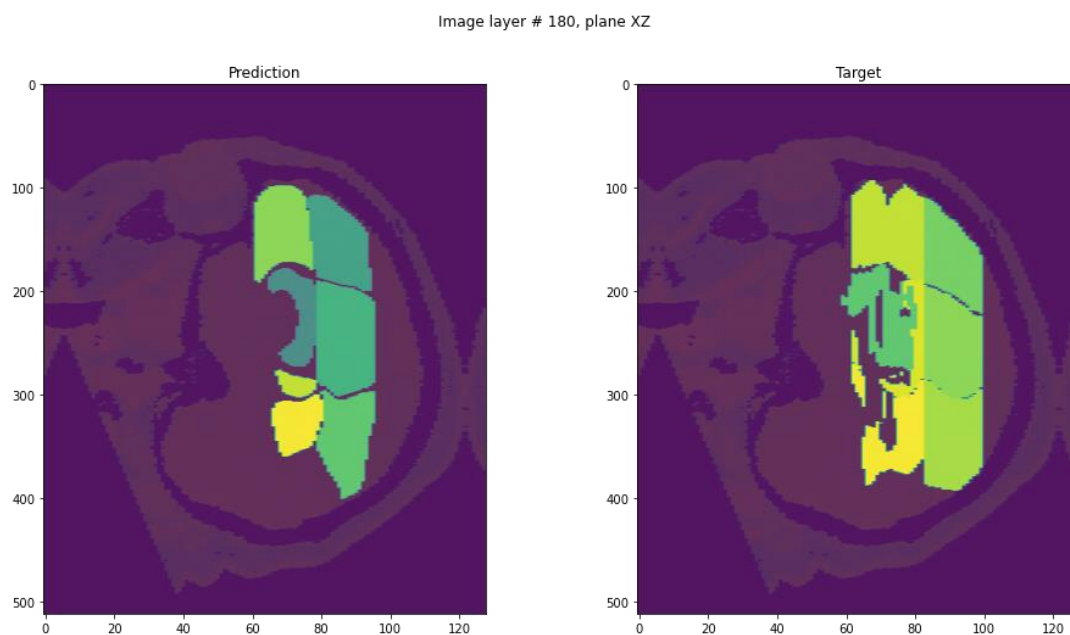
```
Mean of misclassification rates between classes, *1e-3:: baseline: 2.01, YZ: 1.55
```

Теперь готова вторая из трёх частей будущей модели:



## Обучение XZ

Гиперпараметры:  $\alpha=0.85$ ,  $\beta=0.15$ ,  $\text{dropout}=0.3$ . Вот так выглядит результат предсказания обученной модели. Вроде бы всё неплохо, но настораживают несовпадающие цвета сегментации на предсказанной и истинной разметке:



К сожалению, настораживают они меня **теперь**, а на момент обучения сети я был просто удивлён низким качеством модели XZ:

For test image:

IoU:: baseline 0.41, XZ plane 0.13

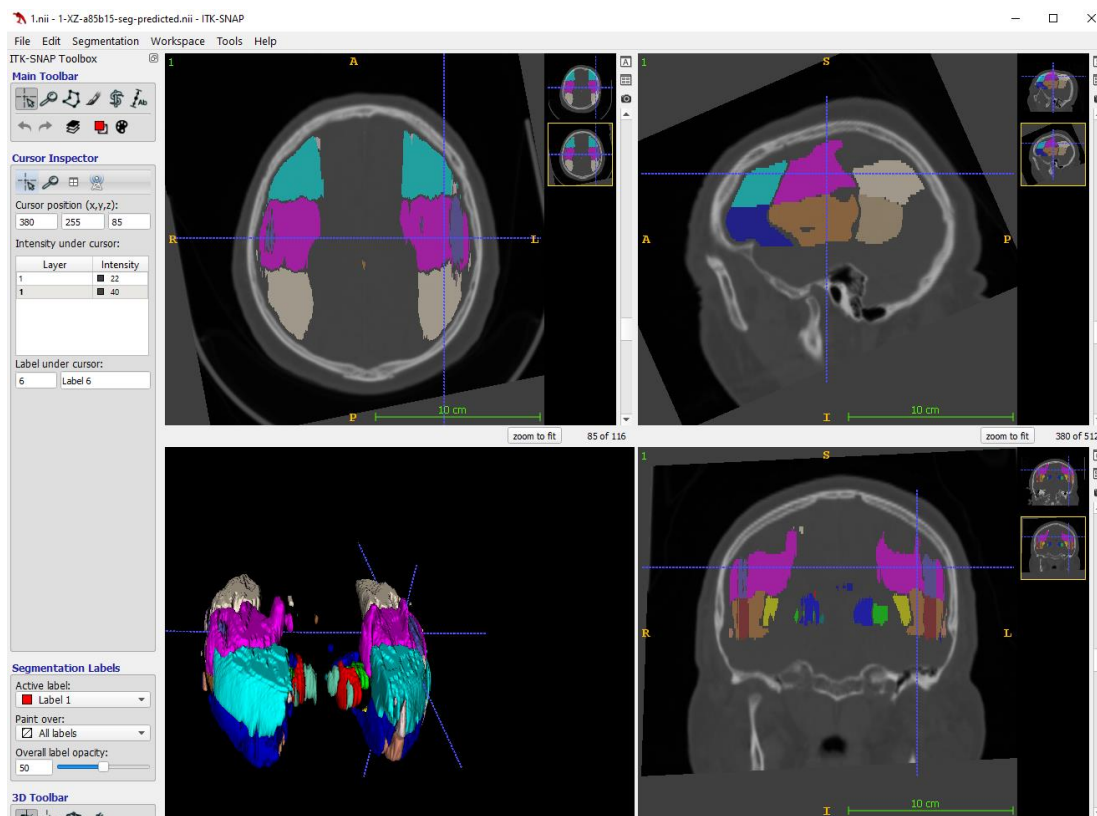
Variance of FP/FN rates between classes:: baseline: 2.62, XZ plane: 1.21

Mean of FP/FN rates between classes:: baseline: 1.64, XZ plane: 0.89

Variance of misclassification rates between classes, \*1e-6:: baseline: 1.35, XZ plane: 2.66

Mean of misclassification rates between classes, \*1e-3:: baseline: 2.01, XZ plane: 2.62

Дошло до меня не сразу, а только после вдумчивого изучения результата предсказаний:



Да, предсказания оказались симметричны потому, что **мозг человека примерно симметричен относительно плоскости XZ, и модель принципиально не умеет отличить левую половину его от правой** при «нарезке» снимка по этой плоскости. Мне следовало бы сообразить это еще на этапе анализа данных и выбора модели, но – не сообразил.

В итоге модель присвоила левым и правым долям один и тот же класс.

Это открытие посетило меня за двое суток до сдачи задания, и самое приличное, что я смог придумать – разделить предсказанную разметку на левую и правую стороны и скорректировать ее так, чтобы в одной стороне были только классы левых долей, а в другой – только классы правых долей. Цель такого действия: довести до конца работу и посмотреть на предсказания ансамбля из трёх моделей, потому что очень интересно, как же в итоге сработает подход.

!!! Я вполне отдаю себе отчёт в том, что таким способом я искусственно улучшил качество предсказаний модели. В реальных условиях, конечно, я предпочёл бы переделать модель, но в данном случае я сместил приоритет в сторону составления качественного отчёта.

#### ▼ Исправление ошибки в плоскости XZ

```
In [193]: # очень стыдно
# исправляем методическую ошибку симметрии в плоскости XZ заменой классов в предсказанной сегментации
wrong_file = sitk.ReadImage('1-XZ_wrong-a85b15-seg-predicted.nii')
prediction = sitk.GetArrayFromImage(wrong_file)
```

```
In [194]: # пусть это ось симметрии головы, там должно быть пусто
div_plane = 274
np.sum(prediction[:, :, div_plane])
```

Out[194]: 0

```
In [195]: # замена разметки
left_array = prediction[:, :, div_plane]
right_array = prediction[:, :, div_plane:]
right_array[right_array>10] -= 10
left_array[left_array==0] -= 10
left_array[left_array<=10] += 10
```

```
In [196]: # сохранение исправленной сегментации
img = sitk.GetImageFromArray(prediction)
img.SetOrigin(wrong_file.GetOrigin())
img.SetSpacing(wrong_file.GetSpacing())
img.SetDirection(wrong_file.GetDirection())
sitk.WriteImage(img, os.path.join('1-XZ-seg-predicted.nii'))
```

Конечно, никаких метрик к фактически сделанной руками разметке применить нельзя. Остается только собрать три предсказания в одно, посчитать его метрики и посмотреть, как это выглядит.



## 8. Объединение полученных сегментаций

IoU, полученная после объединения, хуже, чем IoU для снимка в плоскости XY:

----- Три плоскости -----

For test image, XY-YZ-XZ:

IoU:: baseline 0.41, 3 planes sum 0.46

Variance of FP/FN rates between classes:: baseline: 2.62, 3 planes: 0.09

Mean of FP/FN rates between classes:: baseline: 1.64, 3 planes: 0.37

Variance of misclassification rates between classes, \*1e-6:: baseline: 1.35, 3 planes: 0.49

Mean of misclassification rates between classes, \*1e-3:: baseline: 2.01, 3 planes: 1.31

----- XY -----

For test image:

IoU:: baseline 0.41, improved 0.51

Variance of FP/FN rates between classes:: baseline: 2.62, improved: 1.49

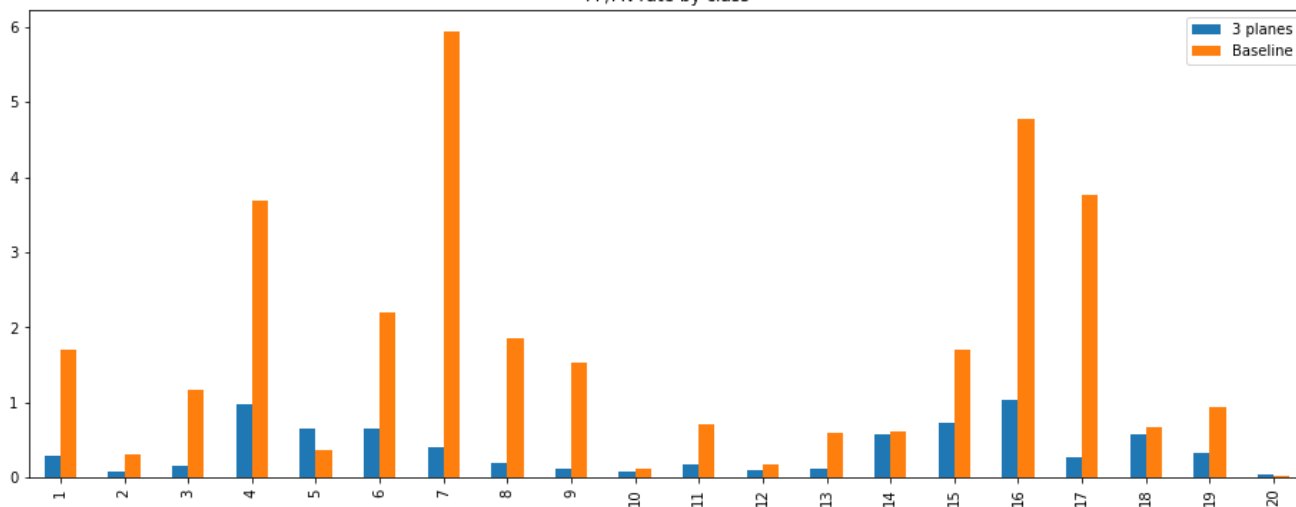
Mean of FP/FN rates between classes:: baseline: 1.64, improved: 1.09

Variance of misclassification rates between classes, \*1e-6:: baseline: 1.35, improved: 0.82

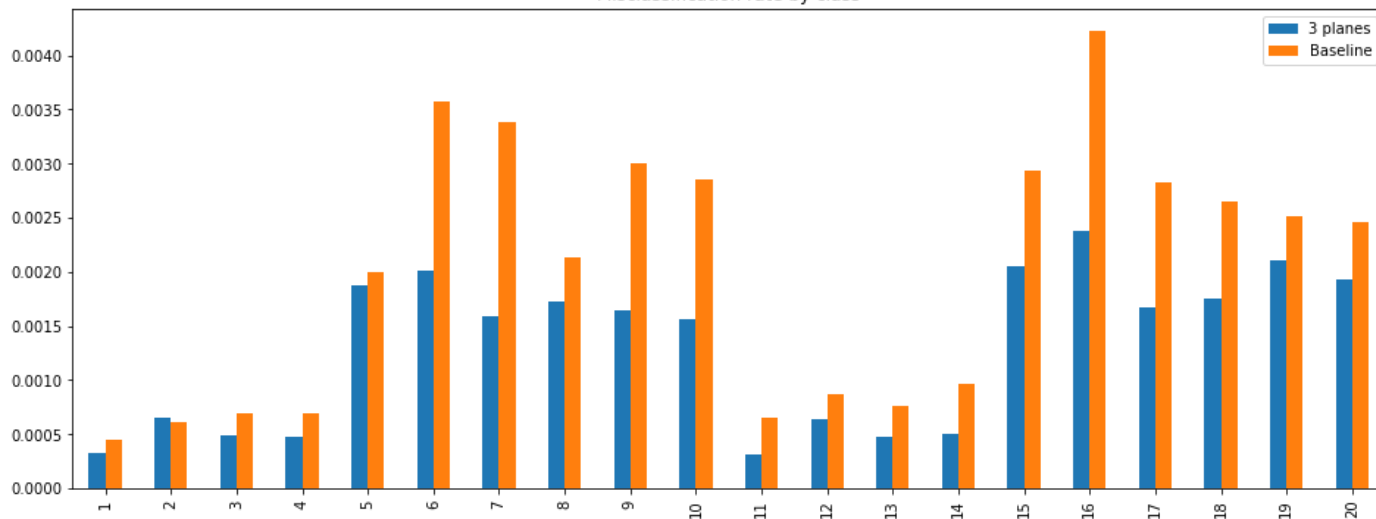
Mean of misclassification rates between classes, \*1e-3:: baseline: 2.01, improved: 1.47

В сравнении с baseline всё выглядит неплохо, сильно упал уровень ложно-положительных предсказаний:

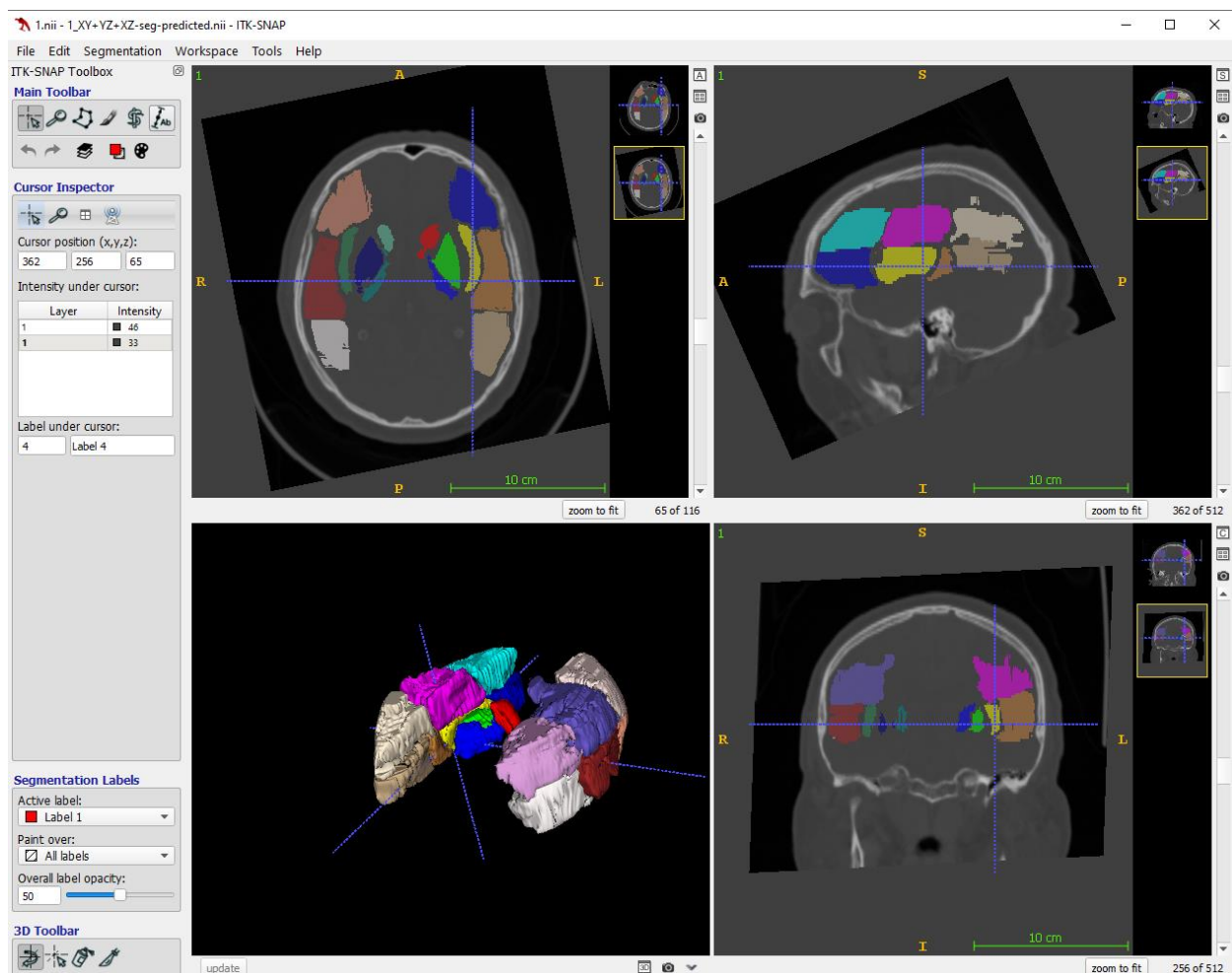
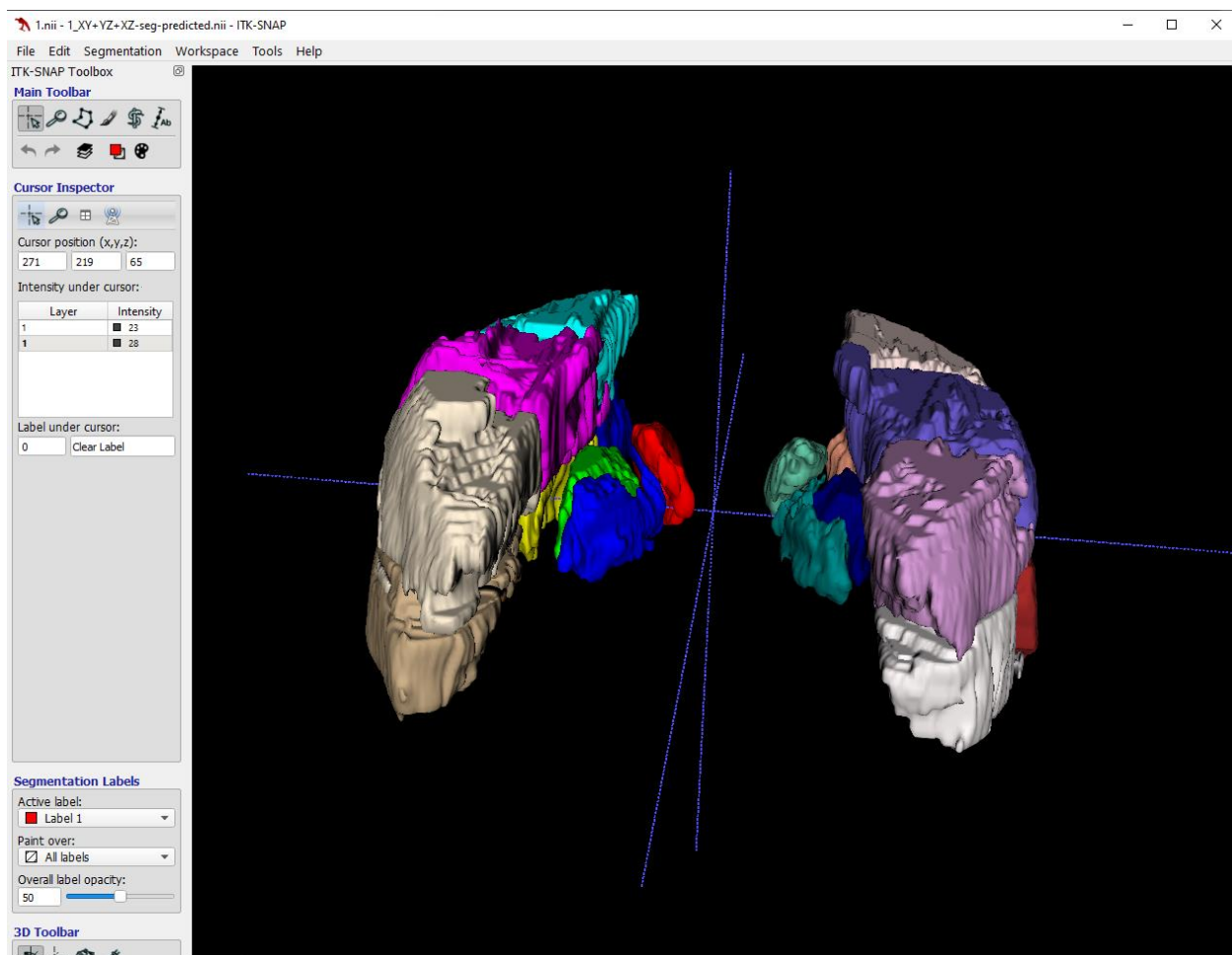
FP/FN rate by class



Misclassification rate by class



Финальный результат работы выглядит так:





## Заключение и выводы

Вполне наглядна работоспособность данной модели, но с моей точки зрения – гораздо перспективнее использование 3D сети для такой сегментации. Впрочем, существуют и решения, аналогичные предложенному мной, но не настолько примитивные – например, здесь <https://github.com/perslev/MultiPlanarUNet>.

Если же продолжать работу с текущим решением – то в первую очередь надо исправить методическую ошибку с плоскостью XZ, затем углубиться в подбор гиперпараметров каждой из плоскостей. Еще один вариант – попробовать искусственно увеличить размеры срезов в плоскостях YZ и XZ, возможно, точность предсказаний улучшится. Хорошо, что я не выбрал вариант с трехмерной сетью и уменьшением размеров снимков – потеря информации оказалось бы существенной.

Стоит рассмотреть возможность обрезки снимков по контуру головного мозга, чтобы еще больше очистить снимки от ненужной информации, но у меня пока нет готового решения, как это можно было бы реализовать.

Вызывает большие сомнения принятая схема оценки качества работы сети на единственном снимке. Но обучающих данных крайне мало, а для выполнения кросс-валидации времени не хватит совсем.

## Что можно было бы сделать лучше в этой работе

Детальность docstrings для вспомогательных функций оставляет желать лучшего.

Подготовить файл predict.py, позволяющий делать сегментацию снимка одной командой. Он был бы логичным завершением работы, но в явном виде этой задачи не стояло.

Поработать с новыми архитектурами сетей для 3D сегментации, но: едва ли не треть времени ушла на изучение формата NifTY и подготовку снимков к анализу, включая написание функции загрузки и поворота.

Подбор гиперпараметров выполнен очень примерно, явно можно улучшить результаты.

Код в ноутбуке тяжело читаемый и объёмный. Множество похожих блоков можно было бы вынести в функции, а оставшийся в ноутбуке код вычистить и привести в порядок, но нехватка времени не позволила это сделать.

Во всех распечатках результатов слово «between» надо заменить на «by». Этот недочет я заметил уже по завершении подготовки отчета и не исправил должным образом.