**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по индивидуальному заданию**
**по дисциплине «Искусственные нейронные сети»**
**Тема: «TGS Salt Challenge»**

Студентка гр. 7382 _____ Петрова А.
Студент гр. 7382 _____ Государкин Я.С.
Преподаватель _____ Жангиров Т.Р

Санкт-Петербург
2020

## Описание датасета

Есть базовый датасет картинок 101x101 сейсмических сканов (данные волновой активности, когда участок земли облучают направленными волнами и ловят отраженные от горных пород волны) горных пород в разных точках земли.

Датасет предварительно разбит на тестовую и тренировочную выборки. Размер тренировочной выборки - 4000 картинок + маски для бинарной классификации (места где находится соль). Размер тестовой - 18000 тысяч картинок, но уже без масок.

## Описание задачи

Спонсорами этого челленджа была поставлена задача вытащить с этих картинок области, где расположены залежи соли. Таким образом можно определить данную задачу как задачу сегментации изображения.

Конечным результатом должны быть изображения с масками соли, для соответствующих изображений.

## Анализ и преобразование датасета

Для того, чтобы подготовить тренировочный и тестовый датасет к обработке нейронной сетью потребовалось сделать следующее:

1. Выкачать из папок соответствующие картинки (и маски) и преобразовать их в numpy массивы. Поскольку исходные снимки сейсмических данных имеют разное количество каналов и они монохромные, то нужно отсечь все, кроме одного (например красного) каналы.
2. Также необходимо сделать апсемплинг картинок до размера 128x128, для того, чтобы можно было без потери кратности проводить сегментацию. Первый и второй пункт реализуются след. функцией:

```
def loadImgsFromFolder(path):
    imgs = []
    valid_images = [".png"]
    for f in os.listdir(path):
        ext = os.path.splitext(f)[1]
        if ext.lower() not in valid_images:
            continue
        img = np.asarray(Image.open(os.path.join(path, f)).resize((128, 128)))
        if img.ndim == 3:
            img = img[:, :, 1]
        imgs.append(img)
    return imgs
```

3. Дальше мы имеем на выходе картинки в виде массивов. Нормализуем снимки по байтовой маске (255), а сегментационные маски (картинки, где белыми пикселями отмечены залежи соли, а черными - остальное) по int32 маске, т.к у этих изображений такая кодировка (т.е пиксель задаётся не байтом, а 4-мя байтами). Ниже выгрузка датасета и нормализация.

```
# load data
train_raw_imgs = loadImgsFromFolder(path_to_train_imgs)
train_raw_masks = loadImgsFromFolder(path_to_train_masks)
test_raw_imgs = loadImgsFromFolder(path_to_test_imgs)

# prepare data
train_imgs = np.expand_dims(np.asarray(train_raw_imgs) / 255.0, axis=3)
test_imgs = np.asarray(test_raw_imgs) / 255.0
train_masks = np.expand_dims(np.asarray(train_raw_masks) / 65535.0, axis=3)
```

## Разработка модели

Разработка модели начинается с подбора подходящей для решаемой задачи архитектуры. В частности для задачи сегментации очень хорошие результаты показывают архитектуры U-Net. Однако, для начала, она довольно сложная, поэтому было принято решение начать с более простой архитектуры

SegNet, которая представляет собой Sequential модель, которую можно условно поделить на 3 блока: pooling, bottleneck, unsampling. Подробнее можно ознакомиться с архитектурой SegNet в данной научной статье.

Также, нужно определиться с метрикой оценки точности сети, лосс-функцией и активацией на последнем слое. Для задач сегментации характерно использовать метрику **MeanIOU**, ей и воспользуемся. Поскольку в условиях нашей задачи классов сегментации всего 2 (бэкграунд и соль), то можно воспользоваться бинарной кросс-энтропией и сигмоидной pixel-wise активацией, которая будет выдавать значения ближе к 1, если считает, что данный пиксель принадлежит классу “соль” и к 0 иначе.

Оптимизатор выберем **Adam** с установленными для keras параметрами (тут нет смысла что-то изобретать, этот оптимизатор один из лучших), размер мини-батча выставим 10, а количество эпох - 20.

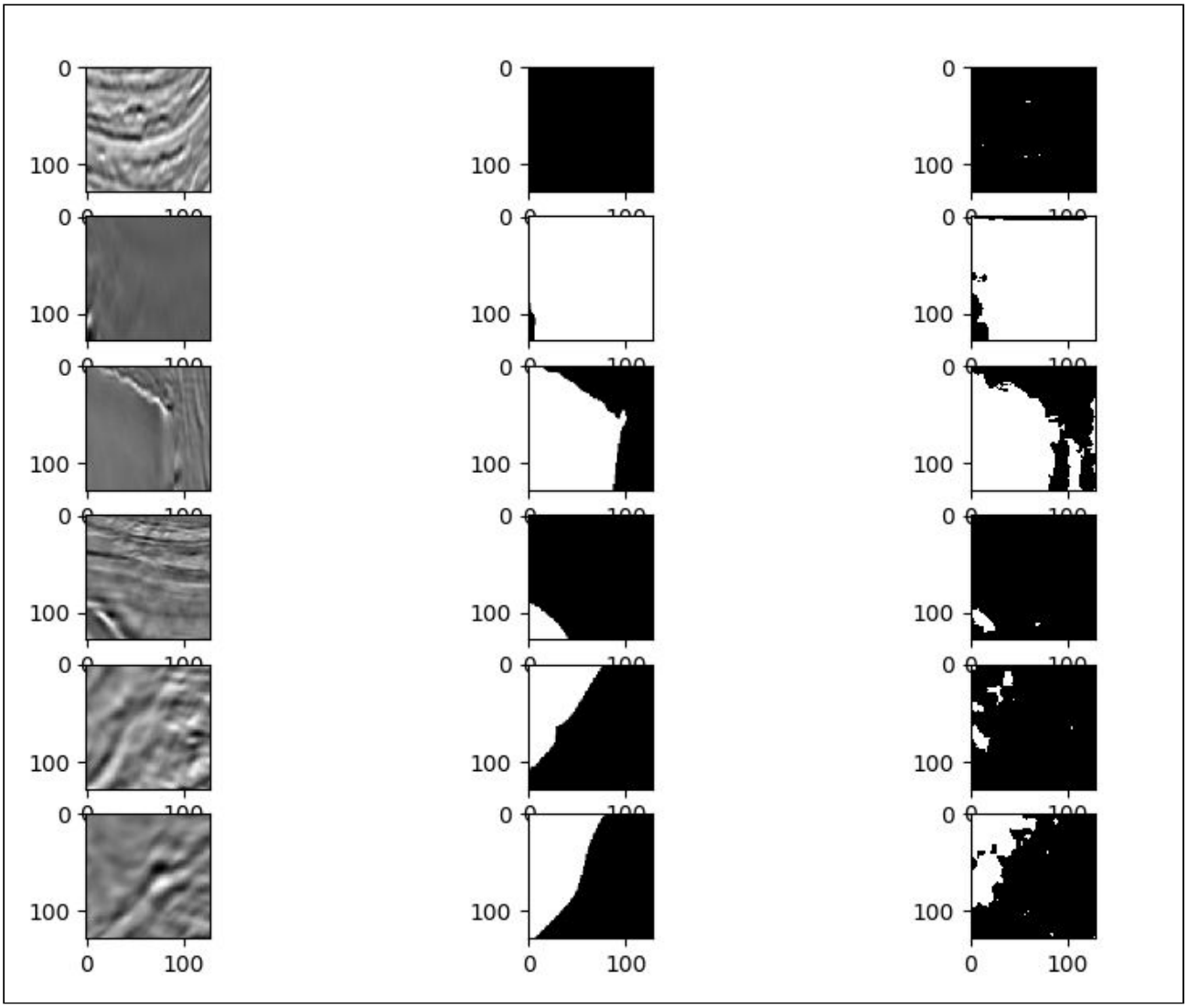
### Прототип сети 1

Следуя архитектуре SegNet соберем простой прототип, для проверки того, что все параметры обучения и архитектуру мы собрали правильно, что сеть действительно обучается:

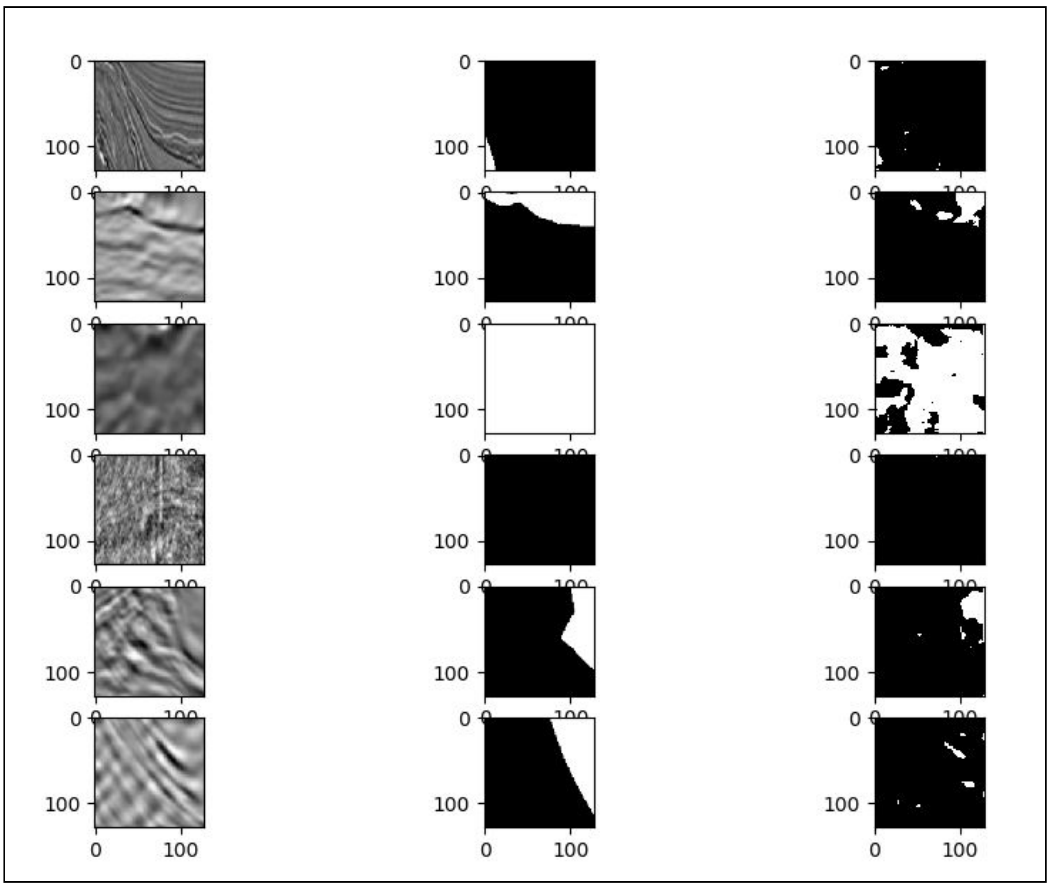
```
self.features = Sequential([
    # pooling
    Conv2D(input_shape=input_shape, strides=(2, 2), filters=8, kernel_size=(3,3), activation='relu', padding='same'),

    Conv2D(filters=16, strides=(2, 2), kernel_size=(3,3), activation='relu', padding='same'),

    # bottleneck
    Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same'),

    # unsampling
    UpSampling2D(size=(2, 2)),
    Conv2D(filters=16, kernel_size=(3,3), activation='relu', padding='same'),

    UpSampling2D(size=(2, 2)),
    Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same'),

    Conv2D(filters=1, kernel_size=(1,1))
])
```

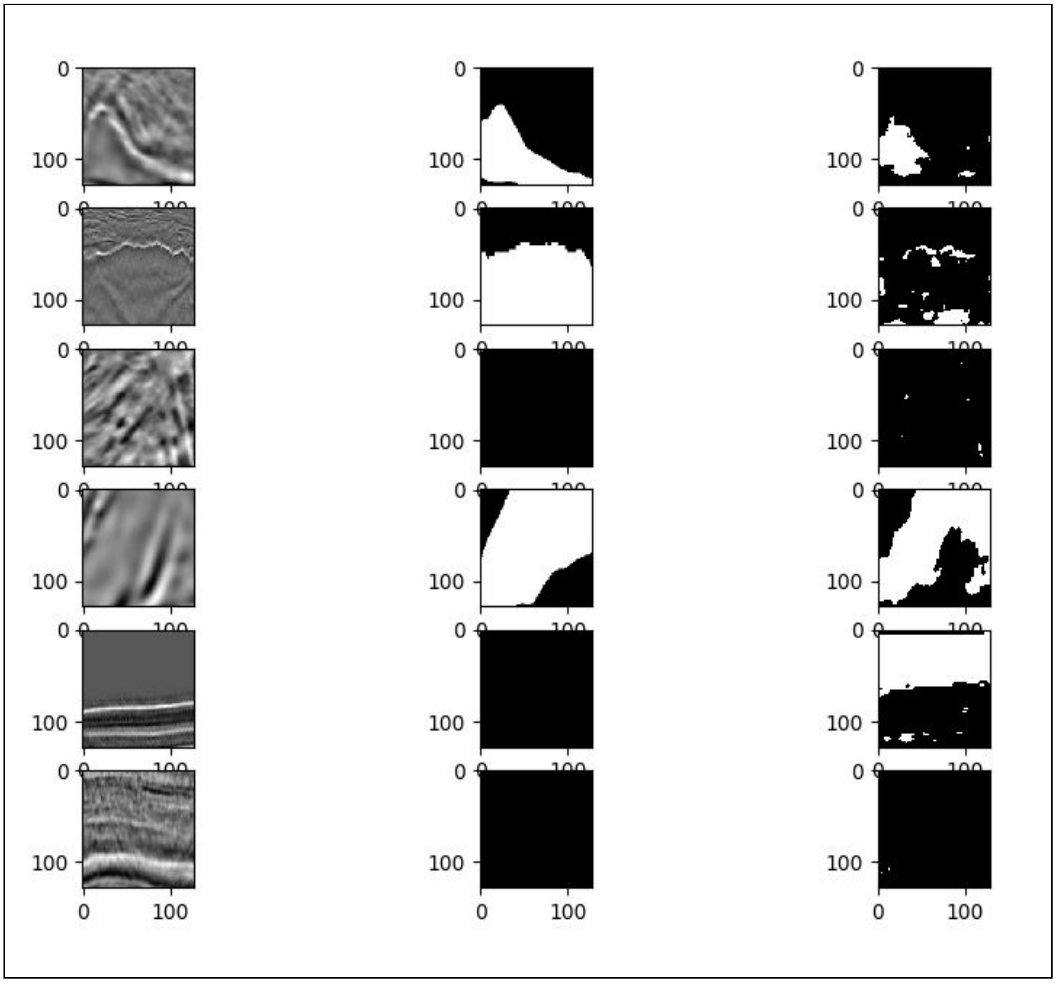
Результат обучения:

На валидационных данных после 20-ти эпох MeanIOU составил 0.52, что уже довольно неплохо. Ниже приведена таблица картинок из тестовой выборки (которая не участвовала в обучении сети), 1 столбик - исходные снимки, 2 столбик - ground truth маска, 3 столбик - prediction сети:




Как можно видеть, результаты довольно правдоподобные, однако сети явно не хватает сложности для того, чтобы лучше определять области с солью. На некоторых предсказаниях много островков на тех местах, где должна быть сплошная область. Ниже представлены другие результаты сети:


0
100
0
100


0
100
0
100

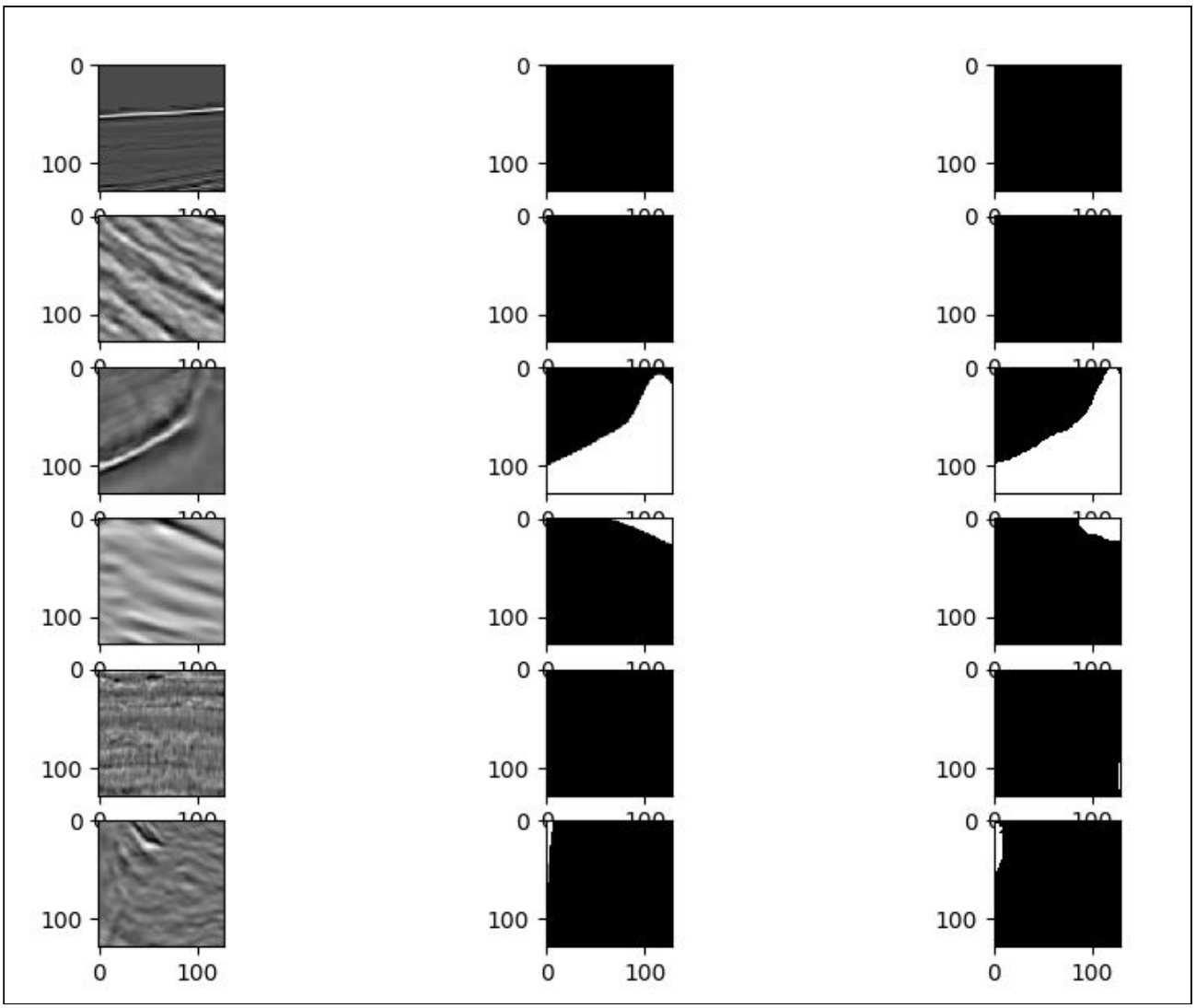
**Прототип сети 2**

В прошлый раз сети недоставало сложности для того, чтобы избежать дробления во многих местах “скопления” соли. Усложним сеть, расширив её и добавим BatchNorm слои, которые упростят обучение сети:

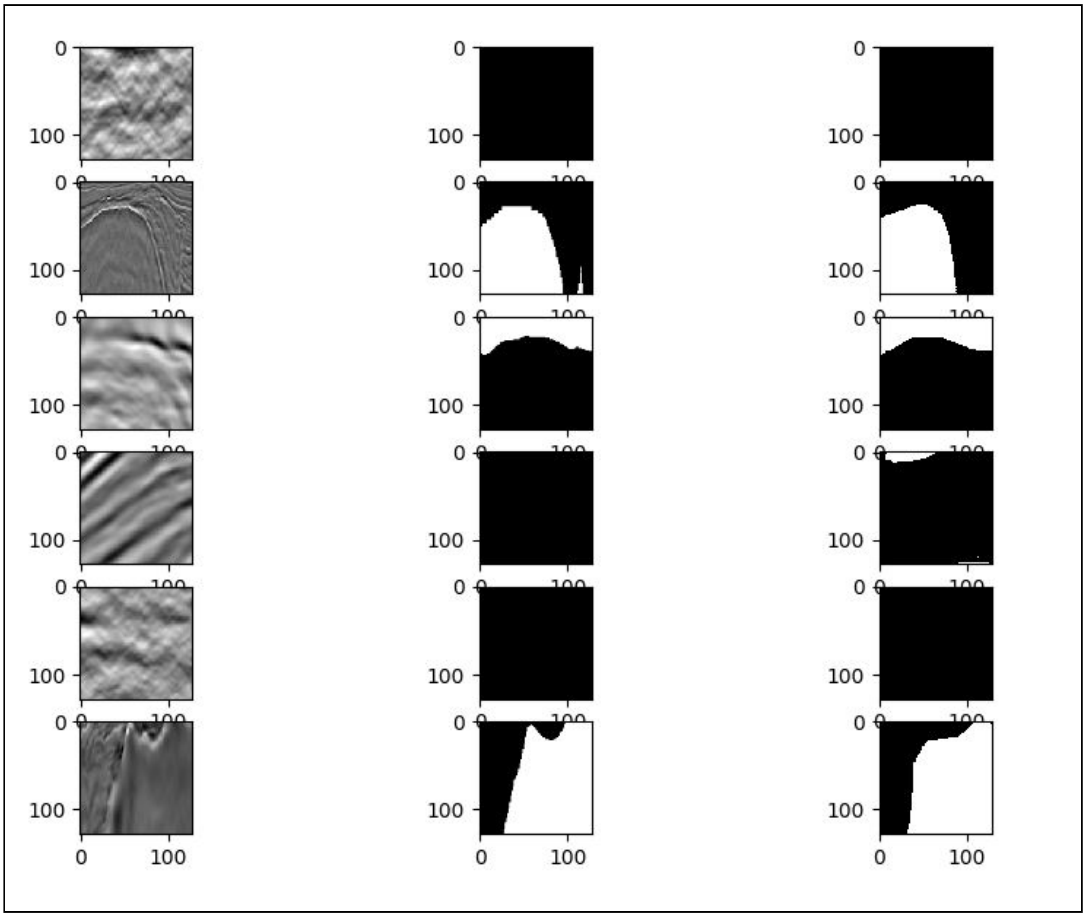
```
self.features = Sequential([
    # pooling
    Conv2D(input_shape=input_shape, strides=(2, 2), filters=8, kernel_size=(3,3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(filters=16, strides=(2, 2), kernel_size=(3,3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(filters=32, strides=(2, 2), kernel_size=(3,3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(filters=64, strides=(2, 2), kernel_size=(3,3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(filters=128, strides=(2, 2), kernel_size=(3,3), activation='relu', padding='same'),
    BatchNormalization(),

    # bottleneck
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    BatchNormalization(),

    # unsampling
    UpSampling2D(size=(2, 2)),
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    UpSampling2D(size=(2, 2)),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    UpSampling2D(size=(2, 2)),
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    UpSampling2D(size=(2, 2)),
    Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    UpSampling2D(size=(2, 2)),
    Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same'),
    BatchNormalization(),

    Conv2D(filters=1, kernel_size=(1,1))
])
```

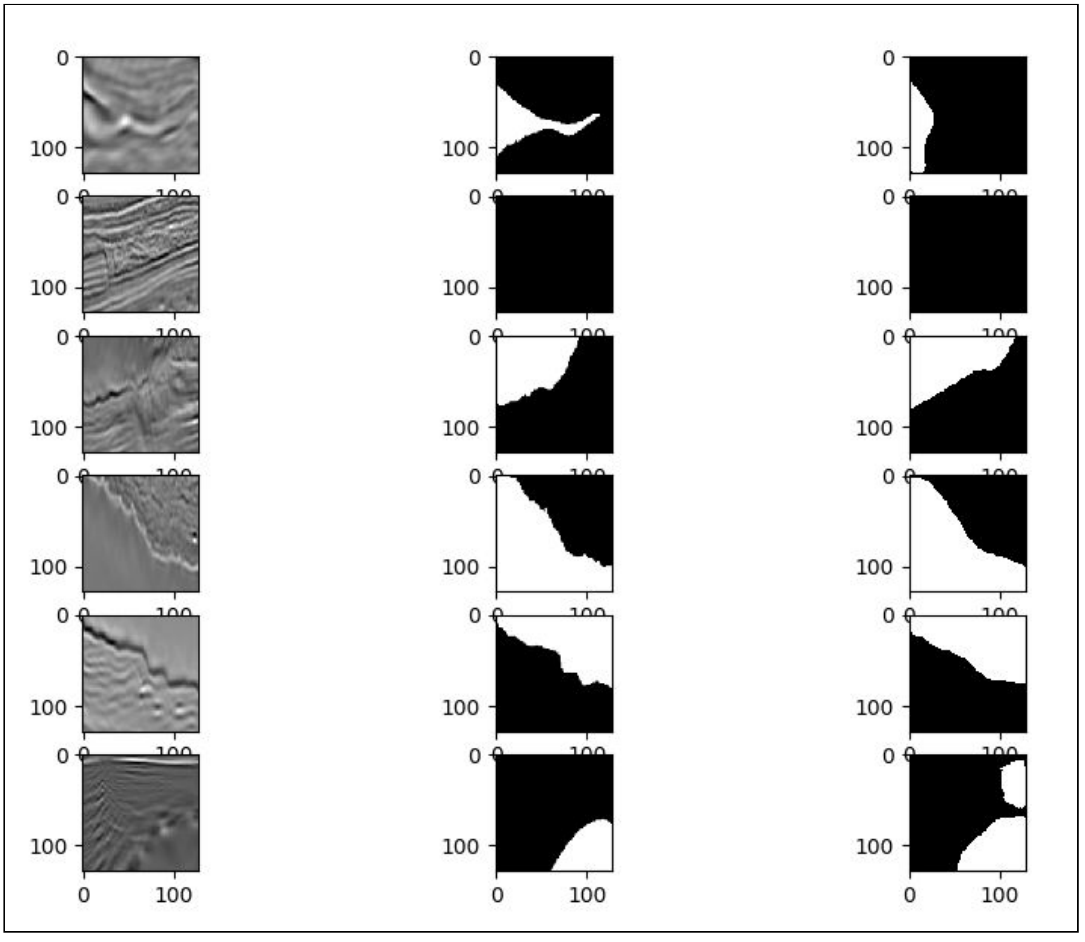
Результат обучения:

На валидационных данных после 20-ти эпох MeanIOU составил 0.77. Это гораздо лучше, чем было в первом прототипе, усложнение пошло сети на пользу. Помимо этого, сеть обучалась гораздо быстрее за счет слоев нормализации. Ниже приведена таблица картинок из тестовой выборки (которая не участвовала в обучении сети), 1 столбик - исходные снимки, 2 столбик - ground truth маска, 3 столбик - prediction сети:




Можно обратить внимание на то, что сеть в некоторых случаях умудряется найти даже небольшие пласты соли на исходной картинке (см последний ряд картинок). Также, можно отметить, что предсказываемые сетью формы стали более сглаженными и больше похожими на ground truth маску соответствующего снимка. Тем не менее, у этого прототипа проявился недостаток в виде небольшого переобучения, которое мы постараемся исправить в дальнейшем. Ниже результаты работы сети на других снимках из тестовой выборки.


0
100
0
100


0
100
0
100

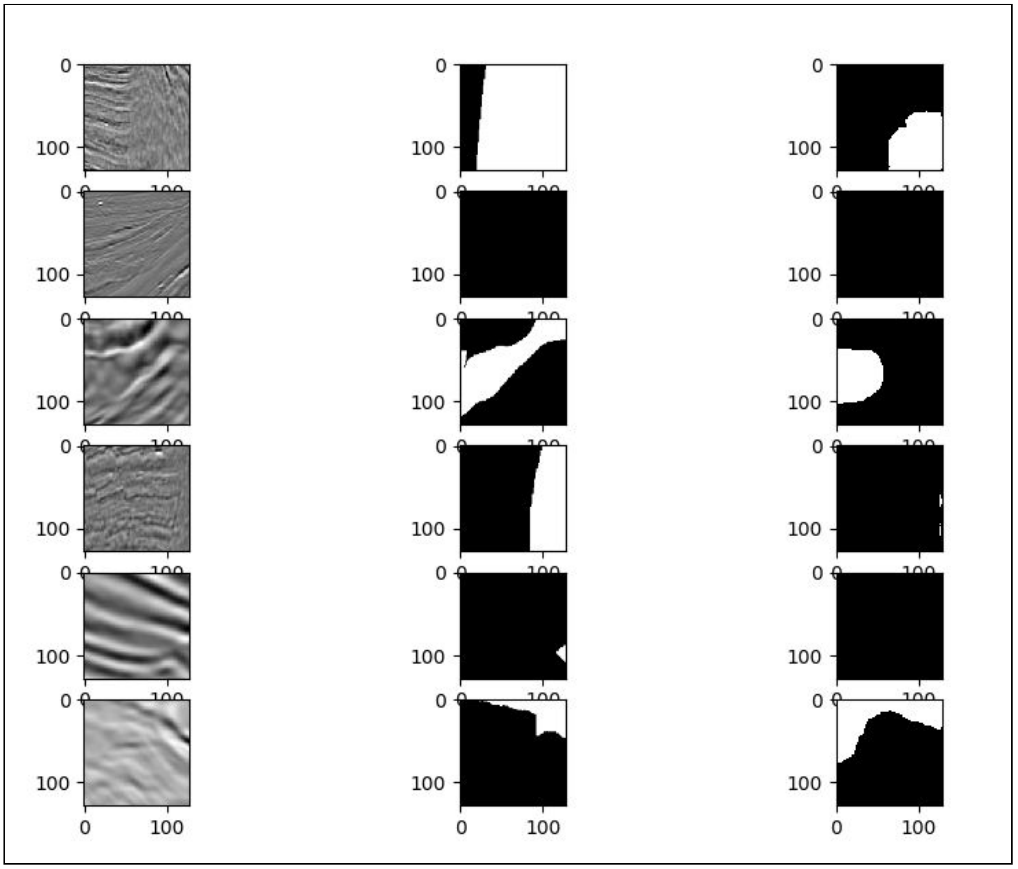
**Прототип сети 3**

Недостатком предыдущей сети было небольшое переобучение, что, возможно, могло привести к детекции соли там, где её на самом деле нет. Чтобы это немного поправить добавим в bottleneck нашей сети Dropout и L2:

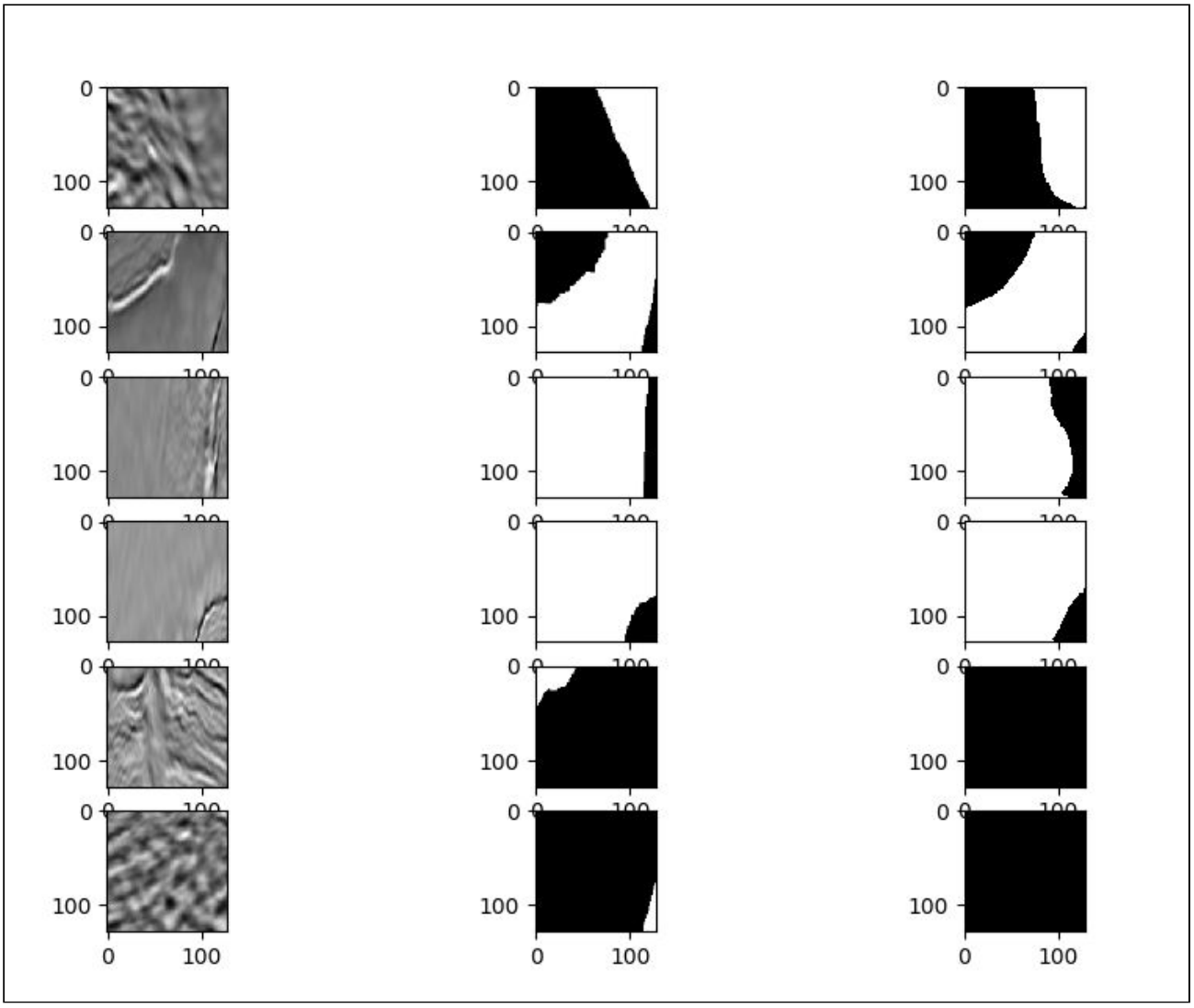
```
# bottleneck
Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(0.01)),
BatchNormalization(),
Dropout(0.1),
Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(0.01)),
BatchNormalization(),
Dropout(0.1),
```

Результат обучения:

На валидационных данных после 20-ти эпох MeanIOU составил 0.68. Это существенно хуже, чем было в предыдущем прототипе, регуляризация уменьшила переобучение но снизила общую точность. Ниже приведена таблица картинок из тестовой выборки (которая не участвовала в обучении сети), 1 столбик - исходные снимки, 2 столбик - ground truth маска, 3 столбик - prediction сети:



Можно обратить внимание, что есть местами существенные различия. Однако, в данном случае сеть не смогла найти соль там где она есть, но и не делала “избыточных предсказаний”, т.е по большей части не указывала не места, где соли не самом деле нет. Тем не менее, результат всё ещё в целом приемлемый.




**Прототип сети 4**

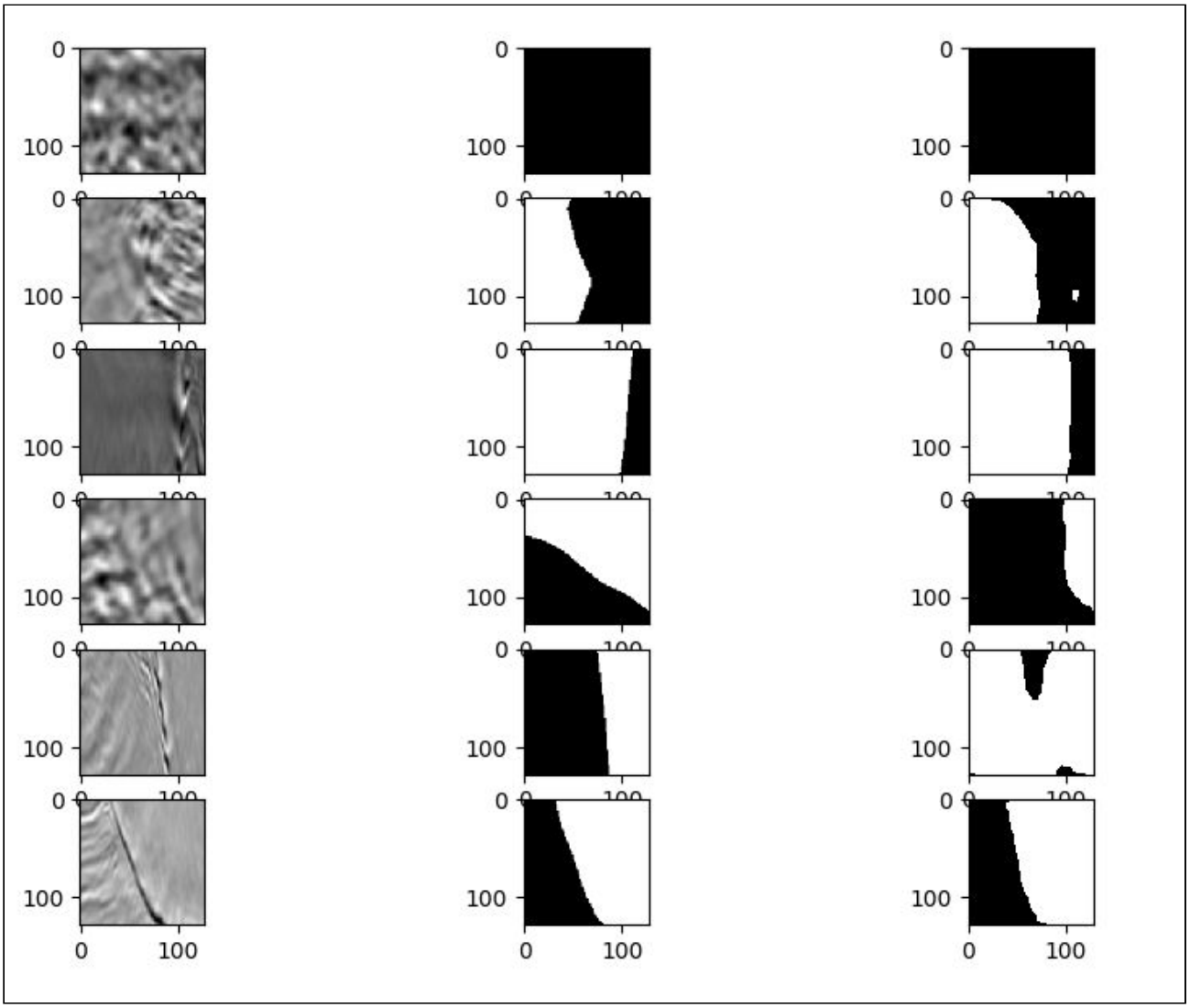
Добавление L2 снизило точность прогнозов, поэтому, чтобы компенсировать снизим коэффициент регуляризации в 10 раз и уберем слои Dropout:

```
# bottleneck
Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(0.001)),
BatchNormalization(),
Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(0.001)),
BatchNormalization(),
```

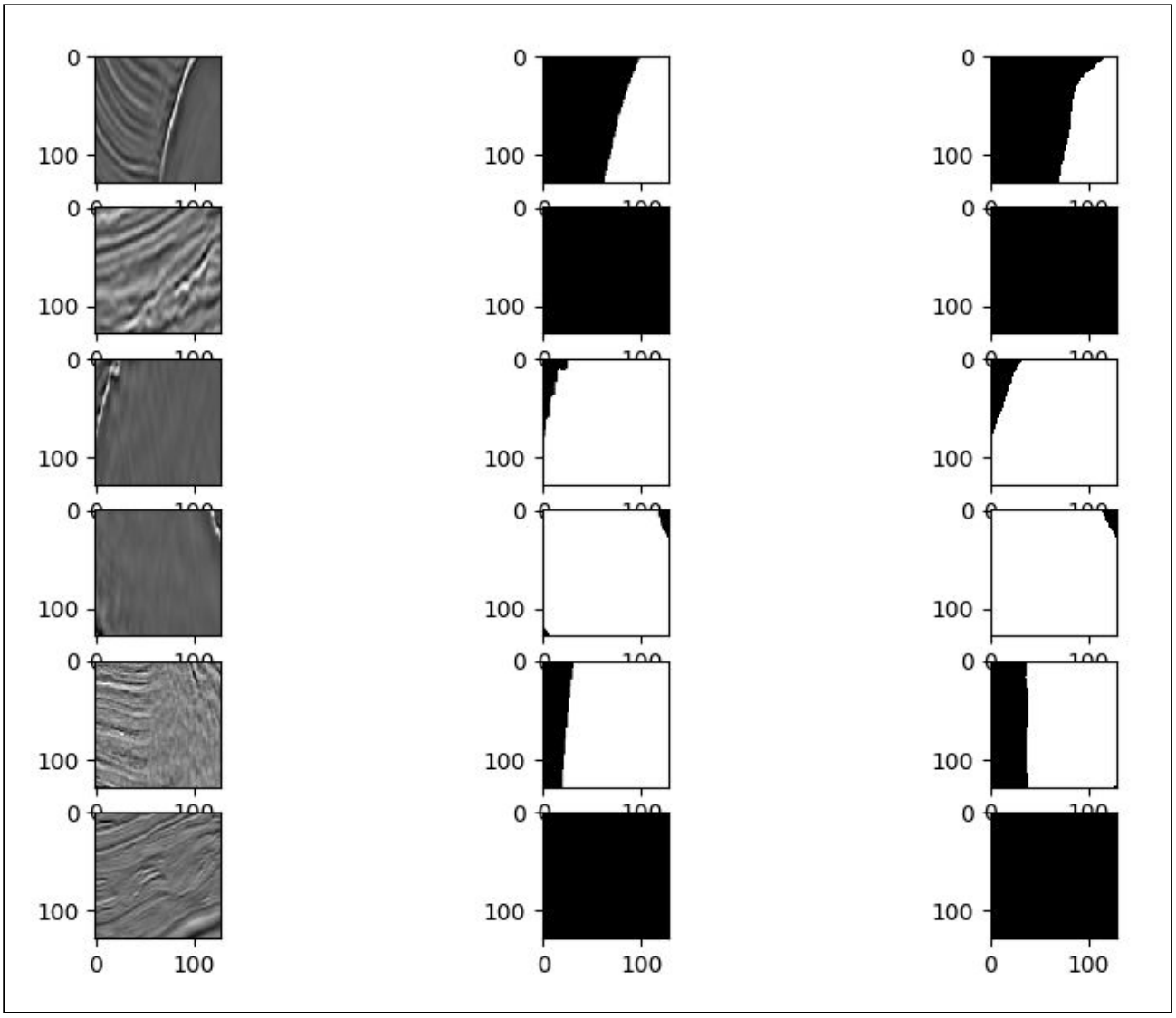
Также, повысим кол-во эпох до 35.

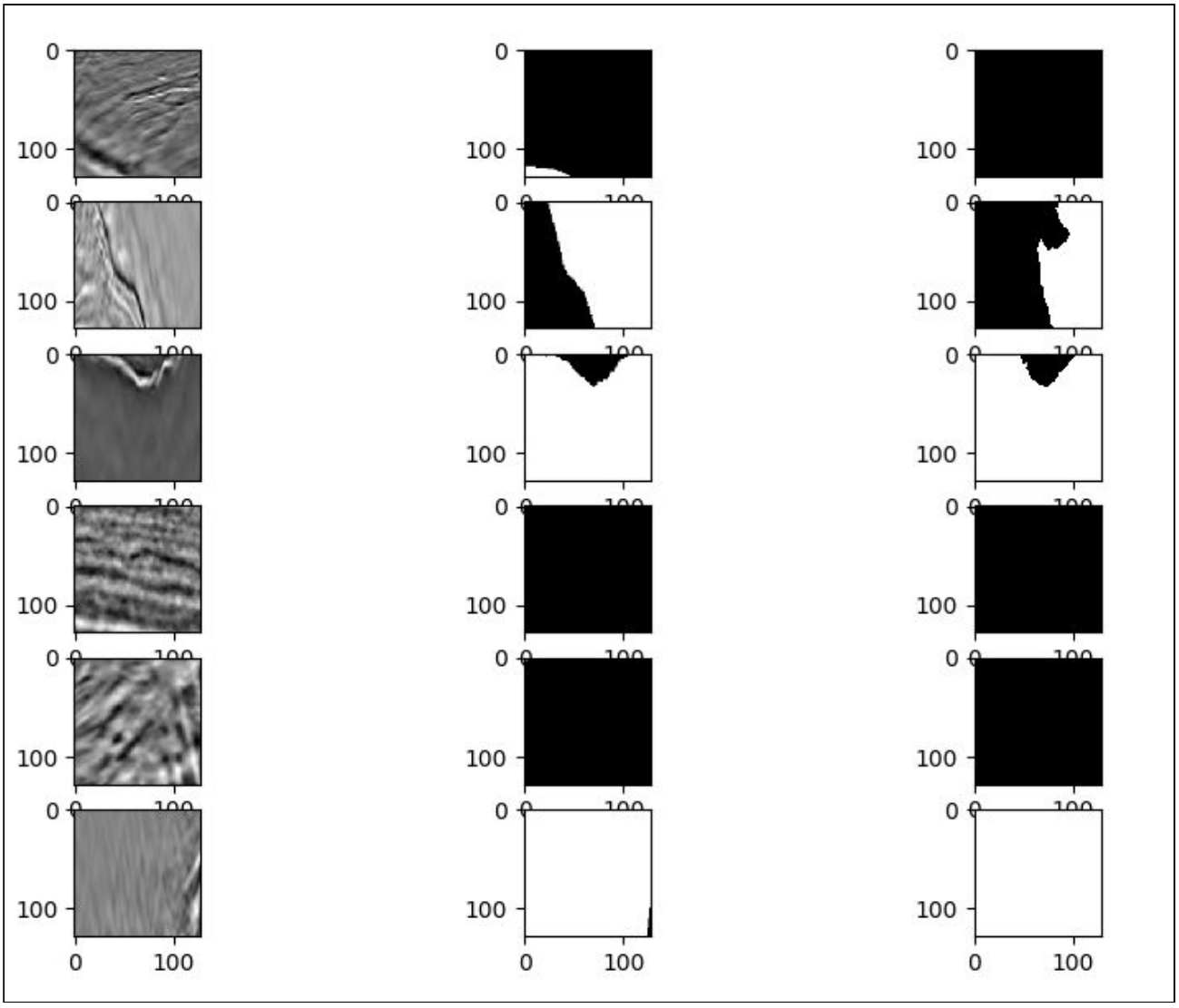
Результат обучения:

На валидационных данных после 35-ти эпох MeanIOU составил 0.80. Это лучше всех предыдущих прототипов. Ниже приведена таблица картинок из тестовой выборки (которая не участвовала в обучении сети), 1 столбик - исходные снимки, 2 столбик - ground truth маска, 3 столбик - prediction сети:



Заметно, что в целом, сеть не дробит сегменты соли как в первом прототипе, меньше предсказывает лишних областей как 2 прототип и имеет хорошую точность в отличае от 3 прототипа. Ниже - другие предсказания сети на тестовой выборке (которая не участвовала в обучении сети):






## Анализ модели

Результирующая модель показывает хорошие результаты, как на валидационных так и на тестовых данных. Метрика MeanIOU лучшего прототипа составила 0.80.

По ходу разработки модели появлялись след. проблемы (по прототипам):

1. Недостаточная сложность сети, а также отсутствие BatchNorm слоёв привела к не очень хорошей точности. Исправлено было масштабированием сети и добавлением BatchNorm слоев

2. В модели появилось переобучение, поэтому стала частой ситуация, когда сеть находит соль там, где её нет. Исправлено было введением Dropout и L2-регуляризации в слои bottleneck-а.
3. Добавление L2 и Dropout серьезно снизило точность прогнозов. Исправлено было небольшим увеличением кол-ва эпох, вырезанием Dropout слоев из сети и уменьшением коэффициентов регуляризации в 10 раз.

## Улучшение модели

Как говорилось выше, SegNet не самая успешная на данный момент архитектура для решения задач сегментации. Для того, чтобы улучшить качество работы сети необходимо преобразовать её в архитектуру U-Net.

U-Net использует идею skip-connection-ов, которые ведут от соответствующих слоев свертки к слоям upsampling-а. В keras есть инструменты для этого, в частности, слои Concatenate, с помощью которых можно организовать такие связи между слоями.

## Области ответственности

- Государкин Ярослав

Отвечал за поиск и выбор темы для ИДЗ. Также, отвечал за поиск и подбор оптимальных архитектур, метрики точности под решение задач сегментации, за разработку последних 3 прототипов в данном ИДЗ.

- Петрова Анна

Отвечала за подготовку датасета, то есть функционал выгрузки в память скрипта, приведение к необходимому виду для обработки нейросетью. Также, отвечала за разработку первого прототипа модели и начального подбора гиперпараметров.

## Выводы

В данном ИДЗ была решена задача сегментации сейсмических сканов горных пород. целью сегментации было найти участки на сканах на которых расположены залежи соли. Задача была решена с помощью нейронной сети архитектуры SegNet. в ходе разработки модели были выявлены проблемы мы связаны с недостаточностью модели, с переобучением модели, я также с падением точности предсказания за счет добавления слоев dropout и L2 регуляризация.

Метрика точности MeanIOU результирующей сети составила 0,8. Предсказания сети на тестовой выборке были продемонстрированы с помощью сравнения Ground Truth масок данных снимков и предсказаний сети. Также, был сделан анализ результирующей сети и предложено, как можно улучшить её качество.