**Тестовое задание**

***1. Реализовать модуль генерации данных***

Реализовать на языке Python класс, один из методов которого будет создавать изображение и описание фигур, находящихся на этом изображении. Сгенерированное изображение должно быть размером 256х256 пикселей и представлять собой однотонный фон случайного цвета, на котором размещены примитивные геометрические фигуры случайного размера, формы, положения и поворота (без выхода за границы изображения и наложения друг на друга) и случайного цвета, отличного от цвета фона. На изображении могут находиться одновременно от 1 до 5 примитивных фигур. В качестве примитивных фигур выступают следующие фигуры: ромб, треугольник, круг, гексагон (правильный шестиугольник). Описанный прямоугольник вокруг сгенерированной фигуры должен иметь размеры по наименьшей стороне не менее 25 пикселей, а по наибольшей не более 150 пикселей.

Выход метода класса:

1. сгенерированное изображение (см. пример на рисунке 1);
2. список параметров *описывающих* прямоугольников сгенерированных примитивных фигур (уникальный идентификатор (id), название фигуры (name), координаты *x*, *y* верхнего левого угла, ширина (*w*) и высота (*h*));

В результате привести листинг кода и папку, содержащую 100 примеров сгенерированных изображений (формат png без сжатия) каждому из которых в соответствие сгенерирован json файл описания (например, изображению 001.png соответствует файл описание 001.json). Примеры сгенерированного изображения и пример файла описания находятся в приложении к этому заданию.

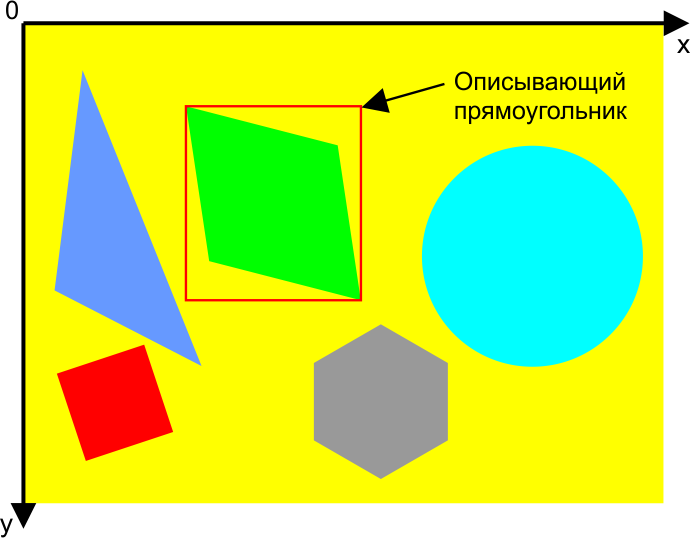


Рисунок 1 – Пример сгенерированного изображения с указанием направления осей и иллюстрацией, что такое описывающий прямоугольник

***2. Детекция примитивных геометрических фигур на изображении***

Реализовать нейросетевую модель (нейронную сеть), которая осуществляет детекцию примитивных геометрических фигур на изображении. В качестве обучающих и тестовых данных использовать изображения, сгенерированные при помощи вашего класс из п.1. Выходом обученной нейронной сети являются параметры *описывающего* прямоугольника. Провести обучение предложенной нейросетевой модели. Привести график функции потерь (loss function) в ходе обучения. В процессе обучения фиксировать параметры обучающей выборки: общее количество изображений и количество изображений на каждый тип фигуры. При обучении использовать аугментацию. Для тестовой выборки из 1000 примеров для 2 разных checkpoint-ов обучения (промежуточного и финального) привести максимальную, минимальную (с приложением примеров изображений и результатов детеции для этих двух случаев) и среднюю оценку IoU, а также посчитать показатели precision и recall для IoU > 0.5. Результаты выполнения задания оформить в виде отчета с описанием архитектуры выбранной сети, обоснованием этого выбора, иллюстрациями результатов, описанием применяемых аугментаций, графиками обучения, комментариями, результатами тестирования и выводами (шрифт Times New Roman, размер шрифта 14, полуторный интервал).

P.S.: реализация в качестве обучающей выборки изображений, сгенерированных на лету, путем вызова соответствующей функции класса, а не сгенерированных предварительно не является обязательным требованием, но дает большой плюс. В этом случае под эпохой обучения принять реализацию обучения на 1 тысячи изображений. В случае использования обучающей выборки из предварительно сгенерированных изображений объем обучающей выборки должен составлять не менее 5 тысяч изображений.

***3. Исследования по дообучению нейронной сети***

Подготовить сбалансированную (примерно одинаковое количество изображений на каждый тип фигуры) обучающую (12000 изображений) и тестовую №1 (3000 изображений) выборку, используя результаты п.1 *без фигуры гексагон*! Заново обучить предложенную в п.2 нейросетевую модель (количество выходов сразу должно быть 5) на сгенерированной обучающей выборке (перед обучением задать весовые коэффициенты сети случайными значениями) и зафиксировать этот checkpoint (назовем его start\_learning). Подготовить сбалансированную тестовую выборку №2 из 3000 изображений, но *содержащую фигуру гексагон*! После этого совершить следующий порядок действий 20 раз:

1. Провести отдельно тестирование нейросетевой модели на тестовой выборке №1 и №2. Для каждой выборки зафиксировать значения precision и recall для IoU > 0.5, а также количество детектированных фигур и общее количество для каждого типа фигуры.
2. Сгенерировать выборку из 400 изображений, содержащую обязательно гексагон (но не только), и добавить ее в обучающую выборку (на первой итерации обучающая выборка станет 12400 изображений, на второй 12800 и т.д.).
3. Дообучить нейросетевую модель c позиции start\_learning.

Результаты исследования привести в форме отчета с таблицами, графиками, гипотезами, своими мыслями и выводами.

Для выполнения данного задания были выполнены следующие шаги:

1. Написан классPictureDrawer, который генерирует фон рандомного цвета размером 256х256 пикслей.

Его возможности:

- генерирует от 1 до 5 областей прямоугольной формы с размерами сторон от 25 до 150 пикселей в произвольном месте, которые в свою очередь будут задавать границы областей построения будущих финур;

- при генерации областей проверяет новую генерируемую область на предмет наложения на уже сгененированные области ранее, если наложение произошло, то процесс повторяется до выполнения условия не пересечения;

- если сцена с ограничивающими прямоугольными областями построена, переходим в построению фигур;

- количество фигур задается параметром num\_rectangles от 1 до 5;

- будет ли генерироваться гексагон, задается параметром hexagon\_status;

- произвольная генерация фигур задается параметром random\_status.

2. Написан класс My\_Dataset для генерации датасета:

- генерирует torch.Dataset;

- трансформация входящего изображения задается параметром transform;

- аугментация задается параметром aug;

- динамическая генерация задается параметром fly;

- путь для сохранения наших данных задается параметром folder.

3. Написан класс UNet:

- создает нейронную сеть по архитектуре UNet;

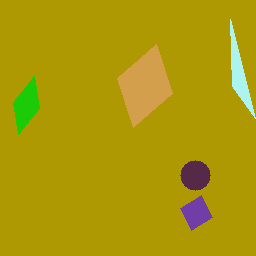
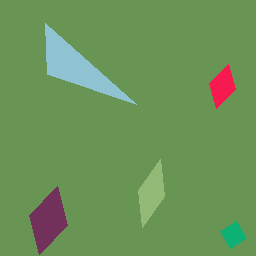
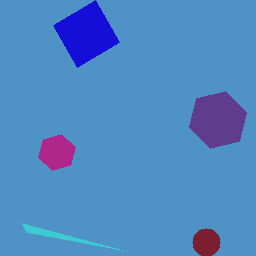
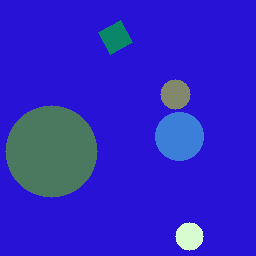
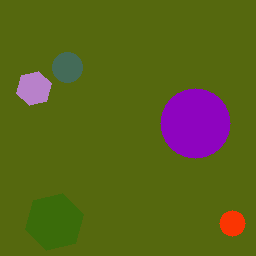
- первоначально она была задумана для семантической сегментации (ранее использовал ее именно для этого). Получая на вход изображение, она выводит изображение в оттенках серого, где интенсивность каждого пикселя зависит от вероятности того, что этот пиксель принадлежит интересующей нас области, но с заменой последних слоев, она нам теперь выдает массив размерностью (5, 4), т. к. описываемые прямоугольные области фигур имеют именно такой размер.

4 . Написан класс ShapeDetector, который создает простую нейронную сеть с фильтрами Conv2D и паддингом 1. Перед разворачиванием сети в линейный слой использовал макспуллинг. Перед последним линейным слоем делал активацию Relu, которая принимает значения от 0 до х, сглаживая отрицательные значения функции до 0. На выходе преобразуем выход линейного слоя в массив размерностью (5, 4).

***Реализация модуля генерации данных***

На данном шаге была выполнена данная реализация, результаты сохранены в папку «first\_stage».

Примеры генерации:



Пример файла-описания:

[

{

"id": 1,

"name": "circle",

"region": {

"origin": {"x": 160.0, "y": 88.0},

"size": {"width": 70.0, "height": 70.0}

}

},

{

"id": 2,

"name": "circle",

"region": {

"origin": {"x": 219.0, "y": 210.5},

"size": {"width": 26.0, "height": 26.0}

}

},

{

"id": 3,

"name": "hexagon",

"region": {

"origin": {"x": 16.61333512679677, "y": 71.11333512679677},

"size": {"width": 34.77332974640646, "height": 34.77332974640646}

}

},

{

"id": 4,

"name": "hexagon",

"region": {

"origin": {"x": 25.52222521132795, "y": 193.02222521132796}, "size": {"width": 57.95554957734409, "height": 57.95554957734407}

}

},

{

"id": 5,

"name": "circle",

"region": {

"origin": {"x": 52.0, "y": 52.0},

"size": {"width": 31.0, "height": 31.0}

}

}

]

***Детекция примитивных геометрических фигур на изображении***

*На данном были выполнены следующие шаги:*

*1. Сгенерированы по 1000 картинок по 5 рандомных фигур, содержащие фигуру гексагон, на каждой для двух тестовых датасетов и помещены в папки first\_test и second\_test.*

*2. Сделаны Dataloader’ы с тестовыми данными с размером batch\_size=32.*

*3. Сделан Dataloader с динамической генерацией картинок.*

*4. Загружен класс* ShapeDetector, обучение будет проводиться на Cuda.

5. При обучении модели использовались:

- оптимизатор Adam

- за минимизацию ошибок и последующего обратного пересчета весов отвечает метрика MSE.

6. Обучение модели проводилось на с эпохой, равной 1000 картинок 20 эпох.

7. В процесс обучения велся подсчет детектируемых фигур, общее количество фигур, а также подсчет loss.

Количество circle в эпохе обучения: 1005

Количество hexagon в эпохе обучения: 1025

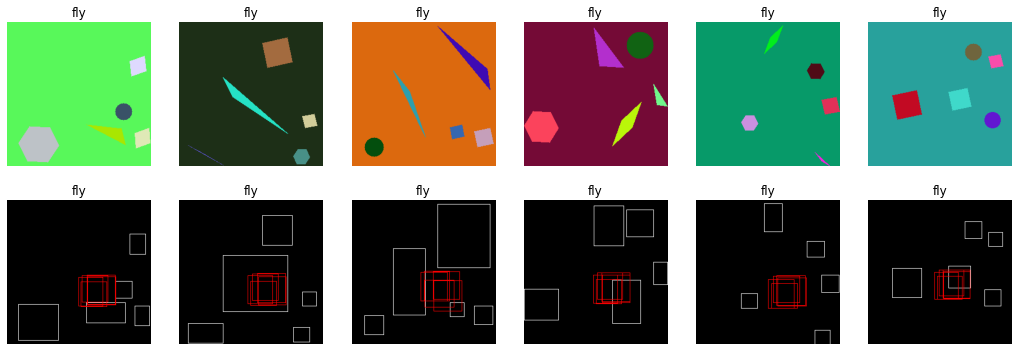
Количество rhombus в эпохе обучения: 986

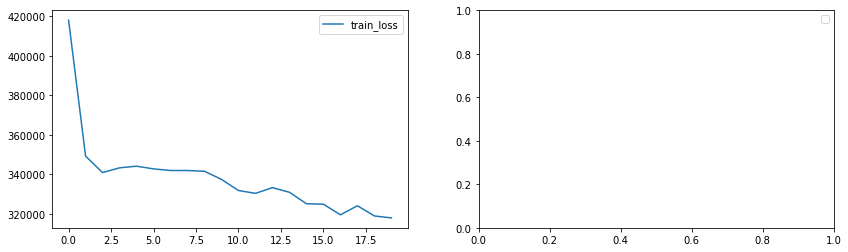
Количество triangle в эпохе обучения: 1001

Количество square в эпохе обучения: 983

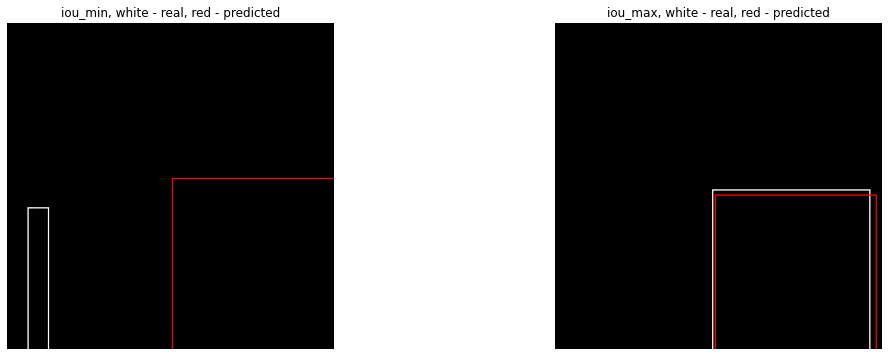
Количество фигур в эпохе обучения: 5000

Пример визуализации процесса обучения:

Пример графика loss:

После обучения была проведена валидация предсказаний модели на первой тестовой выборке.

Пример визуализации процесса валидации:

При валидации получены результаты при IoU >0.5:

precision: 0.024845

recall: 0.004167

8. Последующее обучения решено было провести на статической выборке из 5000 картинок.

9. Сделан Dataloader со статической генерацией картинок.

10. Загружен классShapeDetector, обучение будет проводиться на Cuda.

11. При обучении модели использовались:

- оптимизатор Adam

- за минимизацию ошибок и последующего обратного пересчета весов отвечает метрика MSE.

12. Обучение модели проводилось на с эпохой, равной 1000 картинок 20 эпох.

13. В процесс обучения велся подсчет детектируемых фигур, общее количество фигур, а также подсчет loss.

Количество triangle в эпохе обучения: 5000

Количество hexagon в эпохе обучения: 5029

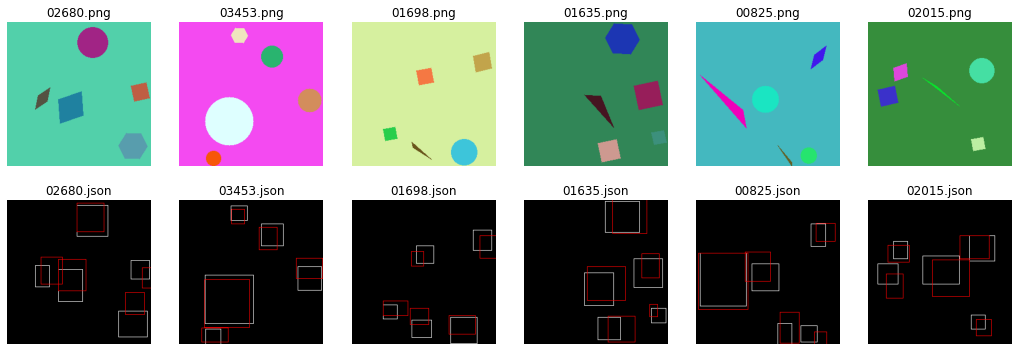
Количество rhombus в эпохе обучения: 4951

Количество square в эпохе обучения: 4988

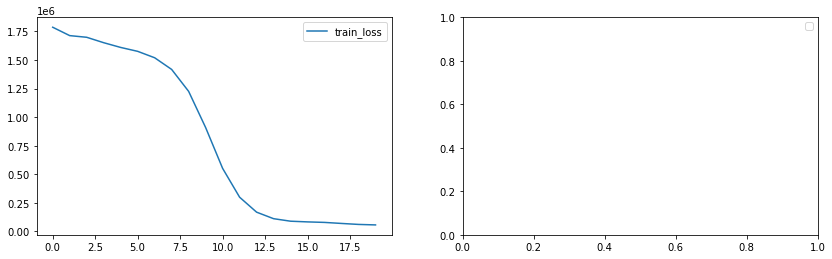
Количество circle в эпохе обучения: 5032

Количество фигур в эпохе обучения: 25000

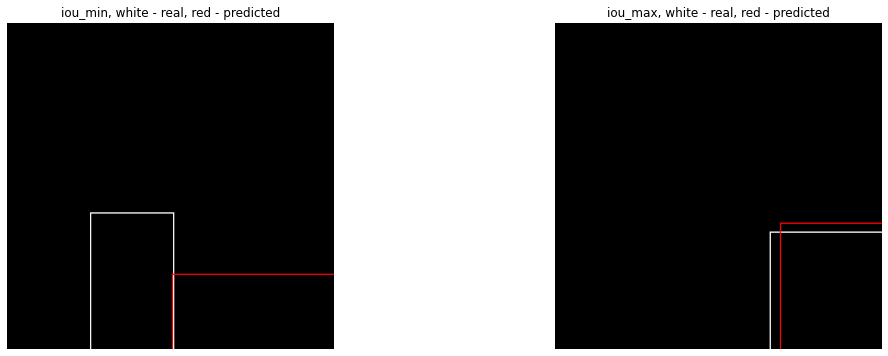
Пример визуализации процесса обучения:

**

Пример графика loss:

После обучения была проведена валидация предсказаний модели на второй тестовой выборке.

Пример визуализации процесса валидации:

При валидации получены результаты при IoU >0.5:

precision: 0.017341040462427744

recall: 0.003125

По данному пункту можно сделать вывод, что сеть обучается намного лучше число визуально, что не скажешь про показатели метрик precision и recall для данных случаев.

Огромный плюс, по моему мнению, в обучении на динамической выборке заключается в том, что сеть в данном конкретном случае получает 20 000 разных картинок на 20 эпохах, взамен 5 000 на статической выборке и не успевает переобучаться.

Но есть и неоспоримый недостаток такого подхода: сеть не успевает обучиться на маленьком количестве эпох.

***Исследования по дообучению нейронной сети***

Для проведения исследований по данному пункту были выполнены следующие шаги:

1. Подготовлены сбалансированные выборки по 3 000 изображений для двух тестовых выборок, одна из которых не содержит фигуру гексагон, другая содержит.

2. Сгенерированна сбалансированная обучающая выборки на 12 000 изображений без фигуры гексагон.

3. Обучена нейросетевая модель на 20 эпохах и сохранены веса модели для дальнейших дообучений.

4. После этого модель обучалась следующим порядком:

- проведены тестирование модели на тестовых выборках №1 и №2 с фиксацией результатов предсказаний в виде значений recision и recall для IoU > 0.5, а также количество детектированных фигур и общее количество для каждого типа фигуры;

- сгенерирована выборка из 400 изображений, содержащая обязательно гексагон (но не только), и добавлена в обучающую выборку (на первой итерации обучающая выборка стала 12400 изображений, на второй 12800 и т.д.);

- модель дообучена с позиции start\_learning.

В процессе валидации получены следующие результаты:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **test** | **precision** | **recall** | **IoU min** | **IoU max** | **IoU mean** | **triangle** | **square** | **rhombus** | **hexagon** | **circle** | **total figures** |
| 0 | test\_1 | 0.024845 | 0.004167 | 0.0 | 0.603122 | 0.580098 | 1003 | 988 | 996 | 1023 | 990 | 5000 |
| 1 | test\_2 | 0.017341 | 0.003125 | 0.0 | 0.529366 | 0.519541 | 5004 | 5075 | 5022 | 4955 | 4944 | 25000 |
| 2 | 0\_test | 0.132588 | 0.029433 | 0.0 | 0.877790 | 0.614414 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 3 | 0\_test\_hexagon | 0.153578 | 0.031206 | 0.0 | 0.888092 | 0.618081 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 4 | 1\_test | 0.111908 | 0.027660 | 0.0 | 0.853802 | 0.613129 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 5 | 1\_test\_hexagon | 0.144201 | 0.032624 | 0.0 | 0.866891 | 0.613922 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 6 | 2\_test | 0.139860 | 0.035461 | 0.0 | 0.834772 | 0.606267 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 7 | 2\_test\_hexagon | 0.151786 | 0.036170 | 0.0 | 0.836780 | 0.617295 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 8 | 3\_test | 0.141892 | 0.037234 | 0.0 | 0.855254 | 0.598709 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 9 | 3\_test\_hexagon | 0.137931 | 0.036879 | 0.0 | 0.911874 | 0.616230 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 10 | 4\_test | 0.134715 | 0.036879 | 0.0 | 0.850544 | 0.606172 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 11 | 4\_test\_hexagon | 0.156047 | 0.042553 | 0.0 | 0.882848 | 0.621525 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 12 | 5\_test | 0.139679 | 0.040071 | 0.0 | 0.861485 | 0.607974 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 13 | 5\_test\_hexagon | 0.157035 | 0.044326 | 0.0 | 0.925953 | 0.625002 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 14 | 6\_test | 0.134454 | 0.039716 | 0.0 | 0.847549 | 0.605287 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 15 | 6\_test\_hexagon | 0.175904 | 0.051773 | 0.0 | 0.926027 | 0.617956 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 16 | 7\_test | 0.146845 | 0.042908 | 0.0 | 0.868630 | 0.593208 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 17 | 7\_test\_hexagon | 0.179070 | 0.054610 | 0.0 | 0.937558 | 0.615646 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 18 | 8\_test | 0.149112 | 0.044681 | 0.0 | 0.855246 | 0.605358 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 19 | 8\_test\_hexagon | 0.190311 | 0.058511 | 0.0 | 0.917257 | 0.612220 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 20 | 9\_test | 0.152761 | 0.046099 | 0.0 | 0.910605 | 0.626248 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 21 | 9\_test\_hexagon | 0.176149 | 0.057092 | 0.0 | 0.940855 | 0.615862 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 22 | 10\_test | 0.142857 | 0.044326 | 0.0 | 0.936099 | 0.618312 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 23 | 10\_test\_hexagon | 0.177350 | 0.058865 | 0.0 | 0.938509 | 0.620321 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 24 | 11\_test | 0.145598 | 0.045745 | 0.0 | 0.916894 | 0.617157 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 25 | 11\_test\_hexagon | 0.190426 | 0.063475 | 0.0 | 0.931465 | 0.613619 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 26 | 12\_test | 0.175660 | 0.054255 | 0.0 | 0.912544 | 0.615353 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 27 | 12\_test\_hexagon | 0.181263 | 0.063121 | 0.0 | 0.902953 | 0.611864 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 28 | 13\_test | 0.177879 | 0.055319 | 0.0 | 0.875687 | 0.610082 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 29 | 13\_test\_hexagon | 0.183613 | 0.065957 | 0.0 | 0.955639 | 0.616619 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 30 | 14\_test | 0.177549 | 0.054965 | 0.0 | 0.883058 | 0.608761 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 31 | 14\_test\_hexagon | 0.170923 | 0.061702 | 0.0 | 0.864194 | 0.616860 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 32 | 15\_test | 0.200238 | 0.059574 | 0.0 | 0.897963 | 0.610766 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 33 | 15\_test\_hexagon | 0.178399 | 0.065603 | 0.0 | 0.886715 | 0.619425 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 34 | 16\_test | 0.175737 | 0.054965 | 0.0 | 0.904044 | 0.616814 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 35 | 16\_test\_hexagon | 0.183301 | 0.067730 | 0.0 | 0.937974 | 0.611243 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 36 | 17\_test | 0.181818 | 0.056738 | 0.0 | 0.923396 | 0.616749 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 37 | 17\_test\_hexagon | 0.195424 | 0.072695 | 0.0 | 0.921298 | 0.616831 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 38 | 18\_test | 0.193843 | 0.060284 | 0.0 | 0.861194 | 0.615406 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 39 | 18\_test\_hexagon | 0.207977 | 0.077660 | 0.0 | 0.905749 | 0.613879 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |
| 40 | 19\_test | 0.188914 | 0.059220 | 0.0 | 0.912690 | 0.629228 | 3000 | 3000 | 6000 | 0 | 3000 | 15000 |
| 41 | 19\_test\_hexagon | 0.201852 | 0.077305 | 0.0 | 0.904141 | 0.614610 | 3000 | 3000 | 3000 | 3000 | 3000 | 15000 |

Как можно увидеть в начале эксперимента значения precision, recall и средне IoU менялись от:

precision: 0.024845

recall: 0.004167

IoU: 0.580098

до:

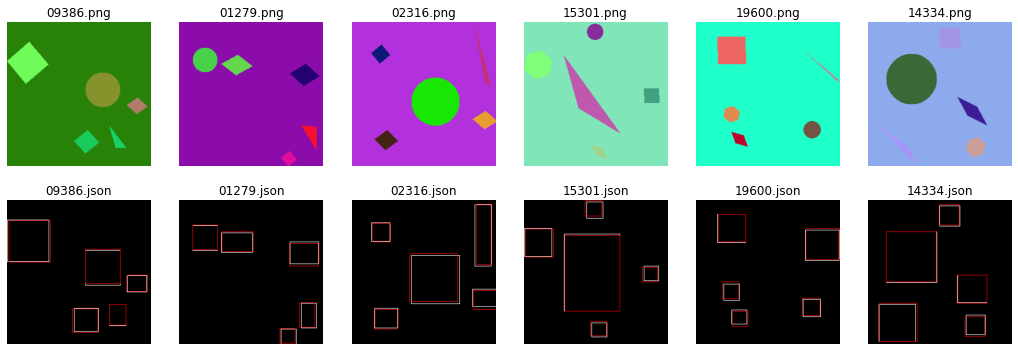
precision: 0.201852

recall: 0.077305

IoU: 0.614610

Данный факт говорит нам о том, что наша модель неплохо обучается, но есть куда стремиться. В выборке фигура гексагон детектируется хуже, чем остальные фигуры, но с увеличением числа фигуры в выборке, показатели метрик детекций растут.

Пример визуализации валидации после обучения:

 Для себя отмечу, что в следующий раз буду делать упор на следующие моменты:

1. Разобраться с низкими показателями precision и recall.

2. Построить более тяжелую сеть для лучшего предсказания на меньших эпохах

3. Если ресурсы важны, то отказаться от утяжеления нейронной сети и лучше эксперементировать с ее архитектурой.