



**ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
(РОСАВИАЦИЯ)
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)**

Кафедра прикладной математики

**ОТЧЕТ
о прохождении преддипломной практики.
Тема: «Распознавание морских судов на аэрофотоснимках
методами компьютерного зрения».**

Дата приема:

Оценка: «_____»

Выполнил:

студент группы ПМ4-1

Фейзуллин К.М.

Проверили:

Руководитель ВКР

к.ф.-м.н., доцент

_____ Филонов П.В.

Руководитель преддипломной практики
заведующий кафедрой, д.т.н., профессор

_____ Кузнецов В.Л.

Оглавление

Введение	3
Постановка задачи	5
Задача классификации	5
Задача семантической сегментации	6
Исходные данные	7
Выбор метрики	9
Описание экспериментальной установки	12
Построение базового решения	13
Решение с использованием человеческих ресурсов	14
Исследование гипотез	15
Опорная гипотеза	19
Гипотеза о применимости алгоритма оптимизации Adam	23
Гипотеза об увеличении ядра свертки	27
Гипотеза об уменьшении свертки	31
Гипотеза об увеличении количества эпох при уменьшенном ядре свертки	33
Гипотеза о достижении заложенной асимптоты	35
Первые результаты задачи семантической сегментации	37
Заключение	39
Список литературы	40
Приложения	41

Введение

Морской грузопоток быстро растет. Большое количество судов увеличивает вероятность возникновения происшествий в море, таких как экологически разрушительные аварии, катастрофы на судах, пиратство, незаконный лов рыбы, незаконный оборот наркотиков и незаконные перевозки грузов. Это вынудило многие организации, от природоохранных учреждений до страховых компаний и национальных государственных органов, более внимательно следить за обстановкой в открытом море.

В результате чего была поставлена задача нахождения кораблей на спутниковых снимках. С задачей распознавания объектов на изображении прекрасно справляются свёрточные нейронные сети. Сверточная нейронная сеть - специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году¹ и нацеленная на эффективное распознавание образов, входит в состав технологий глубокого обучения. 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. Таким образом, идея свёрточных нейронных сетей заключается в чередовании свёрточных слоёв и субдискретизирующих. Структура сети — однонаправленная, принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов (передаточная функция) — любая, по выбору исследователя.

Название архитектура сети получила из-за наличия операции свёртки, суть которой в том, что каждый фрагмент изображения

¹ Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998.

умножается на матрицу - ядро свёртки поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения.

Работа свёрточной 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 (последовательности карт признаков), фильтруя маловажные детали и выделяя существенное.

Для актуальности темы можно добавить, что данный тип нейронной сети сейчас активно используется компанией Google, тот же Google – переводчик, в котором присутствует функция перевода текста с изображения. Та же компания ввела в свой поисковый сервис функцию поиска изображений, аналогичных нашему. Ну и если перейти к более серьезным вещам, то сверточные сети используются для автопилотов в современных автомобилях, например, как в бортовом компьютере автомобилей компании Tesla.

Помимо абстрактных примеров, есть и конкретные приложения сверточных нейронных сетей в медицине. В статье ¹ применяется сверточная нейронная сеть для локализации пневмоторакса в легких на рентгеновских снимках, если он присутствует. Реализовано с помощью архитектуры нейронной сети U-Net², которая на данный момент стала стандартной для решения задач сегментации изображений. Аналогично можно использовать

U-Net и для обработки изображений для нахождения раковых опухолей, что и применяется в работе³.

¹ the 2st-unet for pneumothorax segmentation in chest x-rays using resnet34 as a backbone for u-net. arXiv:2009.02805v1 [eess.IV] 6 Sep 2020

² U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597v1 [cs.CV] 18 May 2015

³ Colorectal Cancer Segmentation using AtrousConvolution and Residual Enhanced UNet. arXiv:2103.09289v1 [eess.IV] 16 Mar 2021

Постановка задачи

Задача классификации

Как было описано выше, у человечества появилась потребность в отслеживании движений морских судов в море для тех или иных целей. Для успешного отслеживания судов было принято использовать спутниковые снимки, но ведь вряд ли найдется человек, который будет скрупулезно рассматривать спутниковые снимки в поисках корабля на изображении, а если даже и будет, то компания не станет тратить свои ресурсы для оплаты труда такого работника. Из этой ситуации существует только одно рациональное решение – доверить монотонную работу автоматике. Вычислительной машине подается изображение, а далее следует ответ, присутствует ли морское судно на снимке.

Формализуя, задача будет классификации выглядеть следующим образом. Дана выборка изображений $X = \{x_i | i \in \{1, 2, \dots, n\}\}$, которую мы разделим на обучающую подвыборку $X' = \{x'_i | i \in \{1, 2, \dots, m\}\}$ и тестовую подвыборку $X'' = \{x''_i | i \in \{1, 2, \dots, j\}\}$ так, что $X = X' \cup X''$ и $X' \cap X'' = \emptyset$. Так же мы делим множество правильных ответов $Y = \{y_i | i \in \{1, 2, \dots, n\}\}$ на Y' и Y'' так, что $Y = Y' \cup Y''$ и $Y' \cap Y'' = \emptyset$. Итак, есть выборка изображений X и выборка правильных ответов Y . Пусть $\xi: \Omega \rightarrow x$ – случайная величина, представляющая собой случайное изображение из X . И пусть $\eta: \Omega \rightarrow y$ – случайная величина, представляющая собой случайный правильный ответ из Y . Тогда определим случайную величину $(\xi, \eta): \Omega \rightarrow (X, Y)$ с распределением $p(x|y)$, которое является совместным распределением объектов и их классов. Тогда размеченная выборка – это элементы из распределения $(x_i, y_i) \sim p(x|y)$. Определим, что все элементы независимо и одинаково распределены. Тогда задача классификации будет сведена

К задаче нахождения $p(x|y)$ и заданном наборе элементов $D = \{(x_i, y_i) \sim p(x|y), i = \overline{1, N}\}$.

С помощью обучающей выборки X' и правильных ответов Y' будем находить распределение $p(x|y)$, а уже на тестовой выборке X'' и наборе правильных ответов Y'' для нее, будем смотреть, как хорошо сверточная нейронная сеть может распознавать тестовые изображения, которые никогда не видела, натренированная на обучающей выборки.

Задача семантической сегментации

Казалось бы, автоматизированная система для нахождения изображений с кораблями – это уже полезное нововведение. Осталось прикрепить к изображениям географические метки и уже будет получена окрестность, внутри которой будет находиться морское судно. Но разве нет способа получить точные координаты? Конечно есть, но для этого нужно определить точное расположение корабля на изображении.

Данная задача является задачей семантической сегментации изображения. Суть задачи в нашем случае в том, что нужно выделить одним цветом каждый пиксель изображения, где находится корабль и другим цветом выделить каждый пиксель того, что не является кораблем. Формальная постановка задачи будет аналогичная с постановкой нашей задачи для классификации, только лишь с тем различием, что каждый элемент $y_i \in Y$ не будет правильным ответом, есть ли корабль на изображении $x_i \in X$, а будет являться матрицей, содержащая в себе правильный ответ для каждого пикселя элемента x_i .

Исходные данные

Исходные данные представляет из себя набор изображений $w \times h$, где $w = 768, h = 768$ пример на рисунке 1.



Рисунок 1

Далее предоставляется таблица, содержащее имя изображения и перечисляются пиксели, где находится корабль, если он есть на изображении, пример на рисунке 2.

▲ ImageId	▲ EncodedPi...
00003e153.jpg	
0001124c7.jpg	
000155de5.jpg	264661 17 265429 33 266197 33 266965 33 267733 33 268501 33 269269 33 270037 33 270805 33 271573 33 ...
000194a2d.jpg	360486 1 361252 4 362019 5 362785 8 363552 10 364321 10 365090 9 365858 10 366627 10 367396 9 368165...
000194a2d.jpg	51834 9 52602 9 53370 9 54138 9 54906 9 55674 7 56442 7 57210 7 57978 7 58746 7 59514 7 60282 7 6105...
000194a2d.jpg	198320 10 199088 10 199856 10 200624 10 201392 10 202160 10 202928 10 203696 10 204464 10 205232 10 ...

Рисунок 2

На основе этих данных будет производится обучение сверточной нейронной сети. Перед обучением производится операция перехода от

изображения к его матричной форме. Пример матричной формы первого столбца для одного изображения в таблице 1.

(w = 0, i)	Red	Green	Blue
i = 0	126	141	146
i = 1	126	141	146
i = 2	126	141	146
.....			
i = h - 3	117	134	138
i = h - 2	115	133	137
i = h - 1	116	134	138

Табл.1 – пример матричного представления изображения для обучения модели.

Для бинарной классификации, столбец пикселей корабля преобразуется в вектор – строку размерности два, пример для одного изображения на рисунке 3.

[0., 1.]

Рисунок 3

Для семантической сегментации столбец пикселей корабля заменяется на полную карту изображения, хранящая бинарное значение для каждого пикселя, являющееся признаком принадлежности к первому или второму классу. Пример карты для одного изображения на рисунке 4.

```
[0][0][0][0][0][0][0][0][0][0][0]...
[0][0][0][0][0][0][0][0][0][0][0]...
[0][0][0][0][0][0][0][0][0][0][0]...
[0][0][0][0][0][0][0][0][0][0][0]...
[0][0][0][0][0][0][0][0][0][0][0]...
[0][0][0][0][0][0][0][0][0][0][0]...
[0][0][0][0][0][0][1][1][1][0][0]...
[0][0][0][0][0][0][1][1][1][1][0]...
[0][0][0][0][0][0][1][1][1][1][1]...
[0][0][0][0][0][0][1][1][1][1][1]...
[0][0][0][0][0][0][0][1][1][1][1]...
...
```

Рисунок 4

Выбор метрики

Итак, определив наши задачи нужно понять, как же оценивать полученные результаты. Наиболее интуитивной для задачи бинарной классификации является такая метрика точность (accuracy). Чтобы определить ее и последующие метрики, введем карту обозначений, приведенную в таблице 1.

		Прогноз	
		+	-
Правильный ответ	+	TP (True – positives)	FN (False – negatives)
	-	FP (False – positives)	TN (True – negatives)

Табл.2 – карта обозначений.

Определим обозначения, в нашем случае, как TP (True-positives) – число раз, когда корабль действительно опознан на изображении, FP (False – positives) – когда корабль ошибочно опознан и FN (False – negatives) – когда корабль присутствие корабля ошибочно отрицается.

Тогда точность можно определить, как:

$$accuracy = \frac{TP+TN}{TP+FP+FN+TN}.$$

Как можно заметить по выражению точности, данная метрика не позволит корректно оценить качество прогноза, если в выборка есть перевес классов ту или иную сторону. Например, если выборка из 1000 элементов будет иметь 990 экземпляров с отрицательным ответом и 10 с положительным, то данная метрика не позволит оценить, как хорошо мы прогнозируем истинно положительный ответ, ведь точность будет представлять собой долю правильных ответов в 99%. Получается, что доля положительных ответов не представляет из себя никакой информации о качестве прогнозирования. Однако, если есть возможность сбалансировать выборку, то данная метрика будет очень информативной.

Но все же, если нет возможности сбалансировать выборку, лучше перейти к другим метрикам, таким как точность (precision) и полнота (recall):

$$precision = \frac{TP}{TP+FP},$$
$$recall = \frac{TP}{TP+FN}.$$

Точность (precision) показывает, какая доля положительных прогнозов является истинно положительными. Полнота (recall) же показывает, какая доля положительных ответов была определена.

На основании двух вышеописанных метрик можно построить график зависимости точности (precision) от полноты (recall), называемый PR – кривой. Данный график будет давать представление о качестве прогнозов и для несбалансированной выборки. Далее находим площадь под этой кривой и получаем количественное значение, представляющее собой меру качества прогнозов в сравнении с истинными ответами.

Далее нужно определить метрику для задачи семантической сегментации. Так как нам априори известно, что на изображениях будет преимущественно преобладать площадь морской глади над площадью морского судна, то требуется мера, которая будет противостоять данному, достаточно внушительному, перевесу. С данной проблемой отлично справится метрика, называемая коэффициентом Соренсена – Дайса (DICE коэффициент). Определяется она как:

$$DICE = \frac{2|A \cap B|}{|A \cup B|},$$

где A – прогнозируемое множество, а B – множество истинных ответов. Но для нашего, бинарного, случая метрика принимает более удобный вид:

$$DICE = \frac{2TP}{2TP+FN+FP}.$$

Данная метрика демонстрирует успешную борьбу с преобладанием одного класса над другим на одном изображении, что доказывается в очередной медицинской статье по семантической сегментации⁴.

⁴ V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. arXiv:1606.04797v1 [cs.CV] 15 Jun 2016

Описание экспериментальной установки

Сверточная нейронная сеть реализована на высокоуровневом языке программирования Python, с использованием встроенных библиотек: keras, scipy, numpy, tensorflow. Данные программные библиотеки позволяют строить нейронные сети, проверять те или иные гипотезы, не затрачивая время на программную реализацию самого механизма проектирования модели и на реализацию обучения модели. Запуск программы совершен на бесплатной облачной платформе Google Colab, которая позволяет совершать неподъемные для домашней машины вычисления.

Несмотря на преимущества облачных вычислений, бесплатная платформа имеет ограничения по объему файлового хранилища и времени работы программы. Время работы не должно превышать восьми часов, а объем файлового пространства не больше 15 Гб. Итак, фактор времени является наиболее весомым, так как время работы программы тратится не только на обучение модели сверточной нейронной сети, но и на предобработку данных. Поэтому решено задать количество изображений для исследования гипотез равное 20000, из которых 16000 непосредственно для обучения модели, а 4000 для проверки качества прогнозов на изображениях, которые модель никогда не обрабатывала. Выборка данного объема сбалансирована для бинарного классификатора и все элементы подобраны независимо друг от друга, что дает уверенность, что при исследовании, результаты для тех или иных гипотез будут релевантны относительно друг друга.

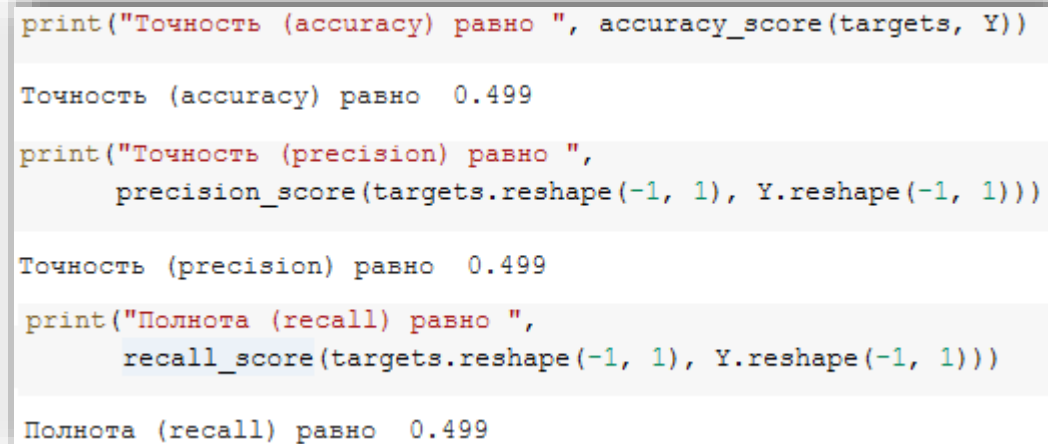
Обучение модели будет проходить на протяжении 50 эпох с объемом пакета одновременной обработки в 256 изображений.

Построение базового решения

Для исследования результатов работы сверточной нейронной сети требуется определить асимптоты. Так как для решения задачи семантической сегментации морских судов нужно полностью освоить упрощенную задачу в лице бинарной классификации изображений, следует задать граничное значение метрик, результаты гипотез ниже которых сразу будут отклонены.

Для задания минимального значения метрик будем использовать генератор случайного подбрасывания монеты, где результатом будет равновероятный исход орел или решка, интерпретируемые как нуль и единица. Проведем сравнение результатов генерации и истинных ответов на прежде описанной выборке в 20000 изображений и найдем для этого такие метрики, как: точность (accuracy), точность (precision), полнота (recall).

Результаты эксперимента приведены на рисунке 5.



```
print("Точность (accuracy) равно ", accuracy_score(targets, Y))

Точность (accuracy) равно  0.499

print("Точность (precision) равно ",
      precision_score(targets.reshape(-1, 1), Y.reshape(-1, 1)))

Точность (precision) равно  0.499

print("Полнота (recall) равно ",
      recall_score(targets.reshape(-1, 1), Y.reshape(-1, 1)))

Полнота (recall) равно  0.499
```

Рисунок 5

Итак, если округлить, то нижний порог для результатов гипотез составляет 0.5 для точности (accuracy), 0.5 для точности (precision), 0.5 для полноты (recall). Гипотезы с результатами ниже данного порога отклоняются.

Решение с использованием человеческих ресурсов

Для обоснования использования сверточной нейронной сети на практике нужно проверить, будет ли это решение более оптимальным, нежели человек сам будет помечать изображения с кораблями.

Для нахождения метрик качества определения класса изображения человеком создадим некий тестовый стенд. Для этого нам понадобится стационарный компьютер, клавиатура, монитор и программа, реализующая вывод изображения на экран и ввод ответа в бинарном виде.

Проведя данное исследование с использованием 100 изображений, за 3 минуты были получены ответы и следующие метрики качества ответов в сравнении с истинными. Результаты на рисунке 6.

```
print("Точность (accuracy) равно ", accuracy_score(Y_true, Y))  
  
Точность (accuracy) равно 0.78  
print("Точность (precision) равно ",  
      precision_score(Y_true.reshape(-1, 1), Y.reshape(-1, 1)))  
  
Точность (precision) равно 0.78  
print("Полнота (recall) равно ",  
      recall_score(Y_true.reshape(-1, 1), Y.reshape(-1, 1)))  
  
Полнота (recall) равно 0.78
```

Рисунок 6

Итак, метрики качества ответов человека на 100 изображениях за 3 минуты составляет 0.78 для точности (accuracy), 0.78 для точности (precision), 0.78 для полноты (recall). Гипотезы с результатами выше данного порога будут считаться оптимальными.

Исследование гипотез

Для исследований мы не будем составлять свою архитектуру сети, а позаимствуем уже готовую, под названием «VGG16».

«VGG16» — модель сверточной нейронной сети, предложенная К. Simonyan и А. Zisserman из Оксфордского университета в статье “Very Deep Convolutional Networks for Large-Scale Image Recognition”. Модель достигает точности 92.7% в задаче распознавания объектов на 14 миллионов изображений, принадлежащих к 1000 классам. Для нашей модели мы сократим размеры входных данных из-за технических ограничений и для решения нашей задачи бинарной классификации мы сократим количество классов с тысячи до двух классов. Структура данной модели изображена на рисунке 7:

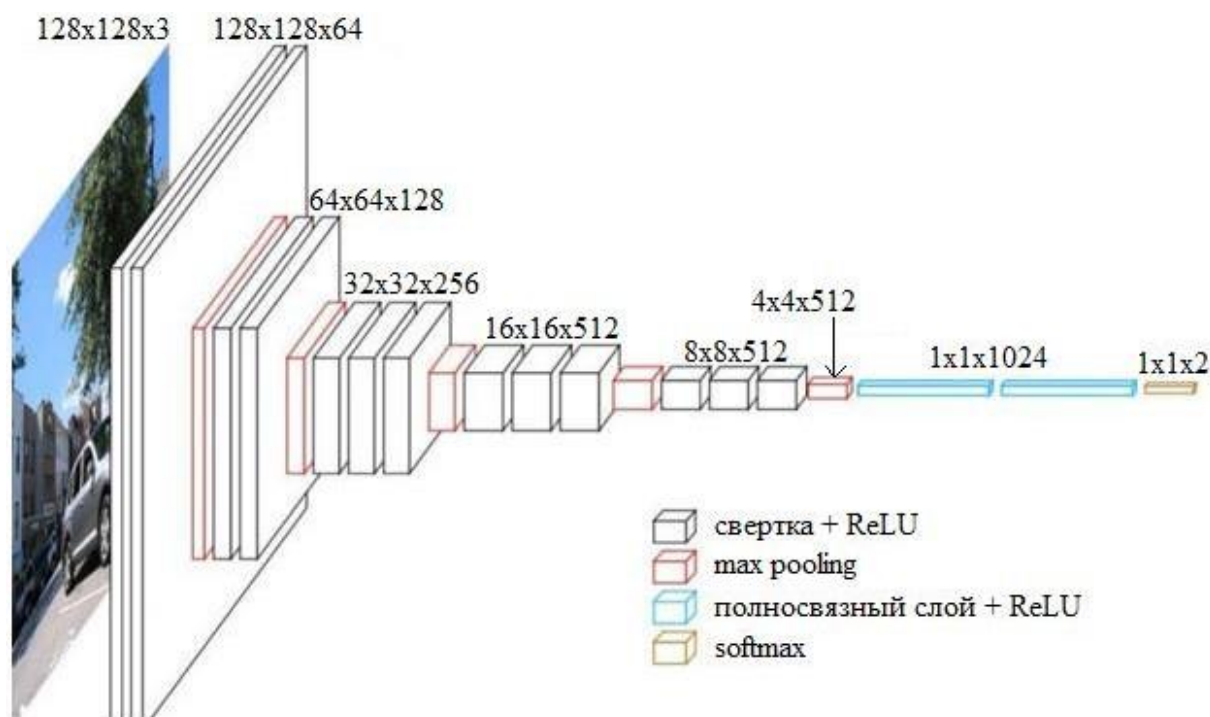


Рисунок 7

Так как используется изображение в формате Red-Green-Blue, каждый из трех каналов цвета обрабатывается отдельно, то сеть принимает три двумерной матрицы 128x128 с интенсивностями цвета. Если обратить внимание на легенду структуры на рисунке 7,

можно заметить, что помимо обозначения самого слоя мы также обозначаем и функцию активации, определенную на слое.

Начнем с функции активации ReLU (англ. Rectified linear unit) или иначе линейный выпрямитель. Функция определяется как

$$f(x) = \begin{cases} 0, & \text{при } x < 0 \\ x, & \text{при } x \geq 0 \end{cases}$$

и имеет график, изображенный на рисунке 8:



Рисунок 8

Сущность функции предельно понятна, если функция активации принимает отрицательное значение, она возвращает ноль и данное значение не искажает данные для дальнейшей обработки, если иначе, то значение следует в дальнейшие слои. Преимущество этой функции активации в том, что вычисление ReLU реализовано с помощью простого порогового преобразования матрицы активаций в нуле. Также к преимуществу можно отнести то, что ReLU не подвержен насыщению. Применение ReLU существенно повышает скорость сходимости стохастического градиентного спуска по сравнению с гиперболическим тангенсом. Считается, что это обусловлено линейным характером и отсутствием насыщения данной функции. К сожалению, ReLU не всегда достаточно надежны и в процессе обучения могут выходить из строя («умирать»).

Слишком большой градиент, проходящий через ReLU, приводит к такому обновлению весов, что данный нейрон никогда больше не активируется. Если это произойдет, то, начиная с данного момента, градиент, проходящий через этот нейрон, всегда будет равен нулю. Соответственно, данный нейрон будет необратимо выведен из строя. Например, при слишком большой скорости обучения может оказаться, что до 40% нейронов с ReLU «мертвы» или иначе говоря, больше не когда не активируются. Эта проблема решается посредством подбора приемлемой скорости обучения.

Далее разберем функцию активации softmax. Softmax – это обобщение логистической функции для многомерного случая. Функция преобразует z – вектор K – размерности в σ – вектор той же размерности, где каждая σ_i – координата полученного вектора представлена вещественным числом в интервале $[0,1]$ и сумма координат равна 1.

Координаты σ_i вычисляются следующим образом:

$$\sigma_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}, \text{ где}$$

$$z = w^T x - \theta, \text{ где}$$

x – вектор – столбец признаков объекта размерности $M \times 1$, а M – количество признаков объектов. w^T – транспонированная матрица весовых коэффициентов признаков, имеющая размерность $K \times M$, а θ – вектор – столбец с пороговыми значениями размерностью $K \times 1$, где K – количество классов объектов, в нашем случае $K = 2$.

Рассмотрим пример работы функции softmax на выходном слое сети для нашей задачи. Пусть $i = 1$ – отсутствие корабля на изображении, а $i = 2$ – присутствие. Допустим, что нашли вектора z по формуле

$$z = w^T x - \theta$$

вида

$$z = \begin{pmatrix} -1 \\ 4 \end{pmatrix}.$$

Тогда имеем

$$e^z = \begin{pmatrix} e^{-1} \\ e^4 \end{pmatrix} = \begin{pmatrix} 0.36 \\ 54.98 \end{pmatrix}$$

и получаем

$$\sigma = \begin{pmatrix} 0.007 \\ 0.993 \end{pmatrix},$$

выбирая максимальное значение получаем, что с вероятностью в 99% корабль присутствует на фотографии, то есть результатом обработки изображения сетью будет 1 – корабль присутствует на фотографии.

Теперь перейдем к функции потерь. Так как на выходном слое используется softmax, для определения потерь с ней используют функцию перекрестной энтропии, которая имеет вид:

$$L(\sigma, y) = - \sum_i^K y_i \ln(\sigma_i).$$

Свойства данной функции:

1. $L(\sigma, y) \geq 0$
2. Минимум функции достигается при $\sigma_i = y_i$

Так как мы будем обучать нашу сеть не по одной фотографии за раз, а сразу на нескольких, то функция потерь для всей выборки находится как:

$$L = - \frac{1}{|V|} \sum_{(x,y) \in V} \sum_i^K y_i \ln(\sigma_i).$$

Опорная гипотеза

Так как задача машинного обучения – это прежде всего задача оптимизации, то и результаты обучения во многом зависят от методов нахождения оптимального решения.

За опорное решение возьмем результаты прошлых исследований, где использовался метод оптимизации RMSProp.

RMSProp (от англ. Root Mean Square Propagation) среднеквадратичное распространение — это метод, в котором скорость обучения настраивается для каждого параметра. Идея заключается в делении скорости обучения для весов на сгруппированные средние значения градиентов для этого веса. Таким образом, первое сгруппированное среднее вычисляется в терминах среднеквадратичного. Как и все градиентные методы, этот имеет общий вид:

$$w = w + h \odot \Delta w,$$

где w – матрица весов нейронной сети, а Δw – направление для минимизации целевой функции, а h – шаг или в нашем случае это скорость обучения. Вся суть метода в том, как мы находим наше направление.

Возьмем нашу скорость h и скорость затухания ρ (параметр сглаживания для экстраполяции). Определим наше начальное значение весов w как нули. И определим малую константу $\varepsilon = 10^{-8}$ чтобы избежать деления на нуль в будущем. Задаем параметр для агрегирования градиента $g = 0$. И пока условие остановки не выполнено, повторяем следующие шаги:

1. Выбираем часть экземпляров количества m

$$\{x^1, \dots, x^m\}$$

из нашей выборки и соответствующие им y^i

2. Находим градиент

$$g = \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial w} L(f(x^i, w), y^i),$$

Где $L(f, y)$ – целевая функция, а $f(x)$ – функция активации.

3. Агрегируем квадраты градиента

$$r = r\rho + (1 - \rho)g \odot g,$$

где $g \odot g$ – это поэлементное произведение матриц градиента.

4. Вычисляем направление

$$\Delta w = - \frac{1}{\sqrt{(\epsilon + r)}} \odot g,$$

где операция $\frac{1}{\sqrt{(\epsilon + r)}}$ применяется к каждому элементу матрицы.

5. Находим новое значение

$$w = w + h \odot \Delta w$$

Условием остановки алгоритма может быть оптимальное значение целевой функции или ситуация, когда веса практически не меняются.

Эмпирически показано, что RMSprop - эффективный и практичный алгоритм оптимизации глубоких нейронных сетей. В настоящее время он считается одним из лучших методов оптимизации.⁵

Проведя обучение модели на протяжении 50 эпох, с размером пакета одновременной обработки изображений в 256 элементов были получены следующие результаты, описанные ниже.

⁵ Глубокое обучение / Ян Гудфеллоу, Иошуа Бенджио, Аарон Курвилль // ДМК Пресс, 2018г., второе цветное издание, исправленное.

Как можно судить по графику показаний функции потерь на рисунке 9, к 30 – й эпохе модель содержит уже достаточно оптимальные значения параметров и дальше происходит уже топтание вокруг оптимума, что иногда приводит к аномальному росту значения ошибки.

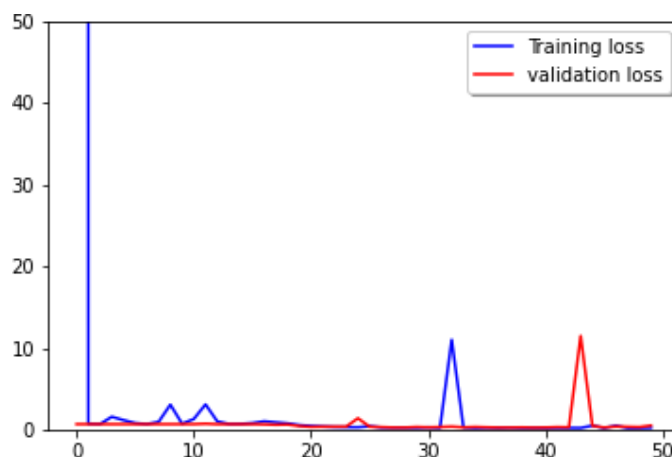


Рисунок 9

Аналогичная картина наблюдается и с графиком метрик, в частности точности (accuracy), максимальное значение которой достигается еще до 30 – эпохи и дальше происходит уже брожение вокруг оптимума с последующими аномальными скачками. Но очень положительным моментом является то, что значение метрики как для тренировочной выборки, так и для тестовой подобны, что говорит об отсутствии переобучения модели. Также стоит отметить, что во время обучения модели точность достигла своей заложенной асимптоты.

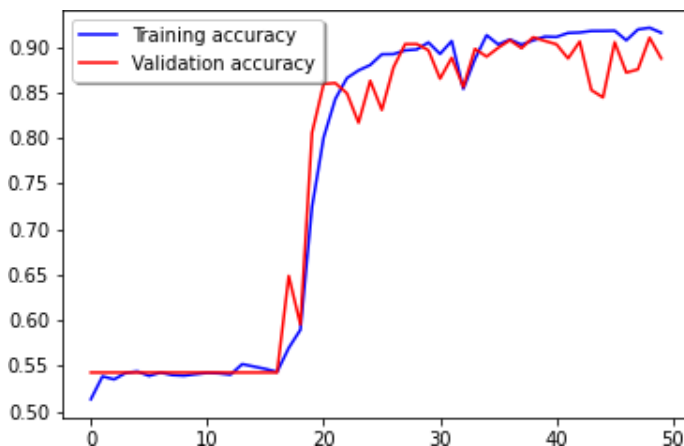


Рисунок 10

Площадь под кривой PR так же достигает своего максимума к 30 – й эпохе, а график метрики на тренировочной выборке так же подобен графику метрики на тестовой выборке, что еще раз доказывает отсутствие переобучения.

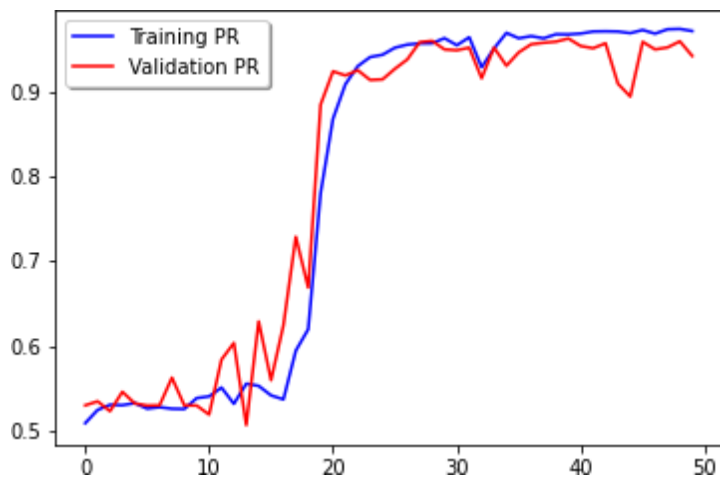


Рисунок 11

На основании данных результатов данная гипотеза принимается и считается одним из оптимальных решений.

Гипотеза о применимости алгоритма оптимизации Adam

Продолжим исследование влияния алгоритма оптимизации на обучение сверточной нейронной сети для бинарной классификации. Алгоритм Adam, переводится как адаптивные моменты (adaptive moments), является комбинацией вышеописанного алгоритма RMSProp и импульсного метода, суть которого заключается в том, что найденное направление и определяемый шаг будет затухать при приближении к оптимуму. Пример движение импульсного алгоритме оптимизации на рисунке 12, где красной линией отображена траектория импульсного метода.

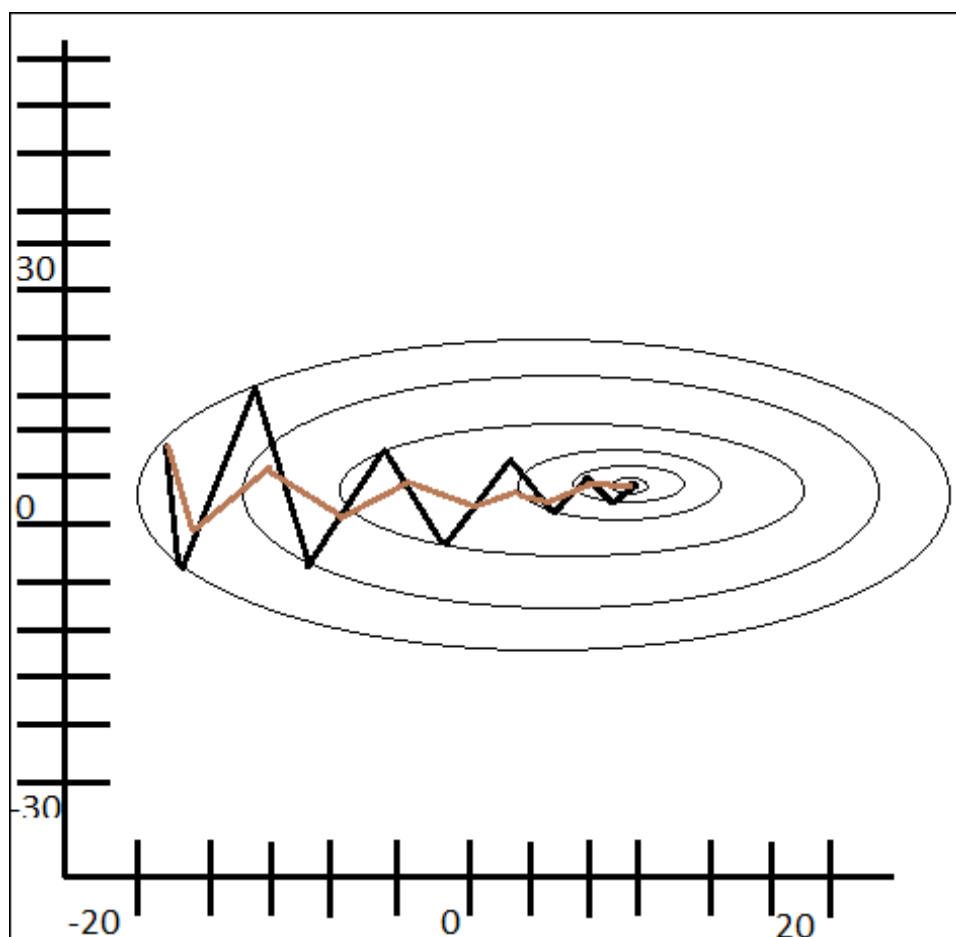


Рисунок 12

Метод Adam имеет следующий алгоритм:

Как и метод RMSProp, этот имеет общий вид:

$$w = w + h \odot \Delta w,$$

где w – матрица весов нейронной сети, а Δw – направление для минимизации целевой функции, а h – шаг или в нашем случае это скорость обучения. Вся суть метода в том, как мы находим наше направление.

Возьмем нашу скорость h и скорость затухания ρ_1 и ρ_2 для оценок моментов. Определим наше начальное значение весов w . И определим малую константу $\varepsilon = 10^{-8}$ чтобы избежать деления на нуль в будущем. Задаем параметр для первого и второго момента $s = 0$, $r = 0$. Задаем шаг по времени $t = 0$. И пока условие остановки не выполнено, повторяем следующие шаги:

1. Выбираем часть экземпляров количества m
 $\{x^1, \dots, x^m\}$

из нашей выборки и соответствующие им y^i

2. Находим градиент

$$g = \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial w} L(f(x^i, w), y^i),$$

Где $L(f, y)$ – целевая функция, а $f(x)$ – функция активации.

$$t = t + 1$$

3. Обновляем смещенную оценку первого момента

$$s = s\rho_1 + (1 - \rho_1)g$$

4. Обновляем смещенную оценку второго момента

$$r = r\rho_2 + (1 - \rho_2)g \odot g,$$

где $g \odot g$ – это поэлементное произведение матриц градиента.

5. Скорректировать смещение первого момента

$$\hat{s} = \frac{s}{(1 - \rho_1^t)}$$

6. Скорректировать смещение второго момента

$$\hat{r} = \frac{r}{(1 - \rho_2^t)},$$

7. Вычисляем направление

$$\Delta w = - \frac{\hat{s}}{\sqrt{\hat{r}} + \varepsilon}$$

8. Находим новое значение

$$w = w + h\Delta w$$

Обучив нейронную сеть на протяжении 50 – ти эпох, были получены разочаровывающие результаты, представленные на рисунках ниже.

На рисунке 13 отображено поведение функции потерь. По данному графику можно сказать, что данный метод справился со своей задачей, поэтому перейдем к показанию метрик точности (ассигасу) на рисунке 14, площади под кривой PR на рисунке 15.

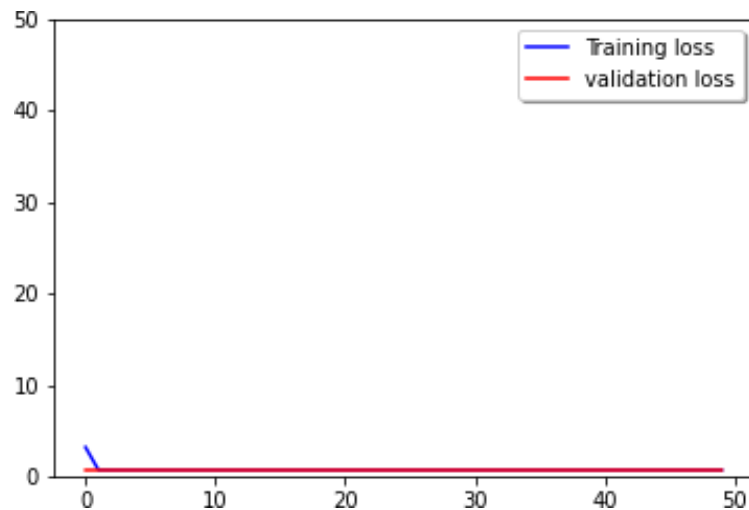


Рисунок 13

Показатели точности на рисунке 14 лишь на несколько сотых превышают значение базового решения, что является отрицательным результатом с учетом разницы вычислительных и временных затрат. Как можно судить по графику, можно предположить, что данный градиентный метод достиг локального минимума, из которого не удастся выбраться.

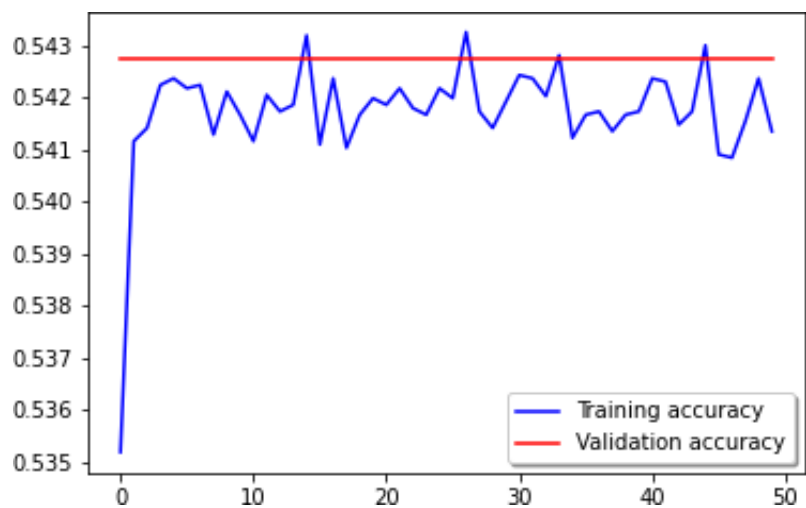


Рисунок 14

Значение площади под кривой PR на рисунке 15 может только увеличивает вероятность попадания в локальный минимум.

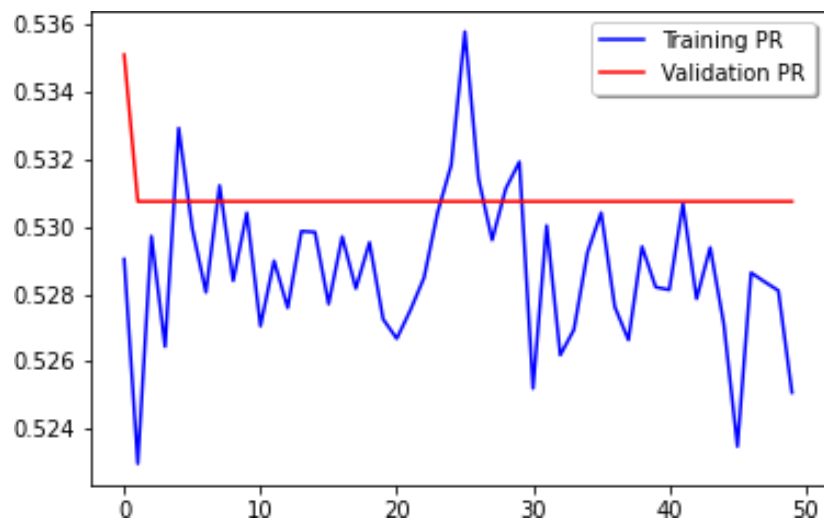


Рисунок 15

На основании вышеописанных результатов гипотеза о рациональности использования метода Adam в данной задаче отвергается.

Гипотеза об увеличении ядра свертки

Все эксперименты до этого проводились с ядром свертки размерности 3×3 . Ядро свертки – это матрица, в нашем случае квадратная, участвующая в линейной операции свертки, где второй операнд – наше изображение.

Рассмотрим операцию свертки на примере матричного вида изображения, только элемент будет в черно – белом варианте, то есть будет задана только одна интенсивность серого. Это будет иметь вид таблицы 1.

(0, i)	Greyscale	...	(w, i)	Greyscale
i = 0	126		i = 0	43
i = 1	126		i = 1	47
i = 2	126		i = 2	49
.....			
i = h - 3	117		i = h - 3	200
i = h - 2	115		i = h - 2	204
i = h - 1	116		i = h - 1	214

Табл.1 – пример матричного отображения черно-белого изображения.

Как видим, в данном случае мы имеем дело с двумерной матрицей. Далее скажем, что наше ядро свертки будет иметь размерность 3×3 , которая заполнена весами, как в таблице 2:

w_{00}	w_{01}	w_{02}
w_{10}	w_{11}	w_{12}
w_{20}	w_{21}	w_{22}

Табл.2 – ядро свертки с значениями весов.

Далее формально представим подматрицу в таблице 3, взятой из матрицы таблицы 1:

(w = j, h = i)	Greyscale		
	j = 0	j = 1	j = 3
i = 0	a_{00}	a_{01}	a_{02}
i = 1	a_{10}	a_{11}	a_{12}
i = 2	a_{20}	a_{21}	a_{22}

Табл.3 – подматрица матрицы интенсивностей серого цвета изображения.

Далее проведем операцию свертки:

$$\sum_{j=0}^2 \sum_{i=0}^2 (a_{ji} * w_{ji}) = a_{00} * w_{00} + a_{01} * w_{01} + a_{02} * w_{02} + \dots + a_{22} * w_{22} = c_{kl}.$$

Так как ядро свертки проходит над матрицей изображения каждый участок, например, с шагом в одну клетку, то образуется следующая матрица в таблице 4, результат свертки:

c_{00}	...	c_{0n}
...		...
c_{m0}		c_{mn}

Табл.4 – результат свертки.

В цикле процедура свертки будет выглядеть как на рисунке 16:

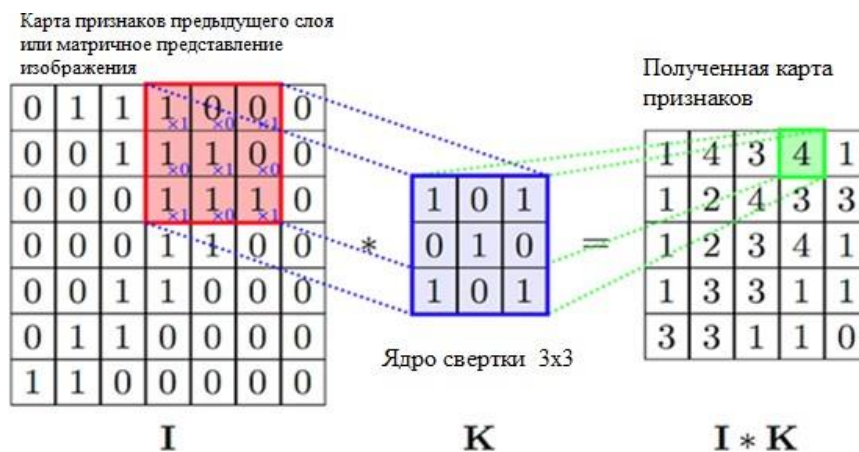


Рисунок 16

Далее результат свертки C передается в функцию активации $F(C)$, которая образует выходной сигнал нейрона, множество которых образует новую матрицу для дальнейшей обработки в следующих слоях.

Итак, ядро свертки напрямую влияет на качество обучения нашей модели. Во – первых, от размерности ядра свертки и от количества ядер в одном слое будет зависеть количество параметров нашей сети. Во – вторых, размерность ядра влияет на емкость модели, то есть на размерность получаемой матрицы после операции свертки, которая уже передается в следующие слои и обрабатывается дальше.

Далее исследуем поведение модели, если задать матрицу свертки размерности 5×5 .

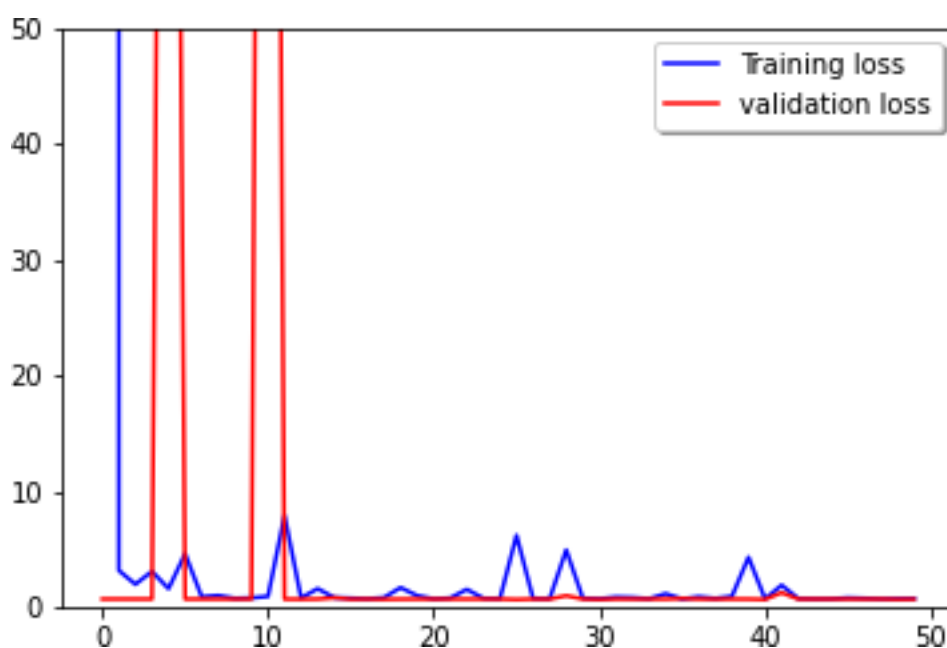


Рисунок 17

Как можно наблюдать на рисунке 17, функция ошибки стремится к нулю, хоть и присутствуют anomальные скачки. Из чего можно сделать вывод, что положение весов близко к оптимальному состоянию.

Однако, показатели точности (accuracy), на рисунке 18 говорят об отрицательном результате, ведь увеличение ядра свертки вдвое увеличило количество параметров в сети, а максимальное значение метрики снизилось почти на 30%, не говоря уже о том, что результаты меньше, чем у решения с помощью человеческих ресурсов.

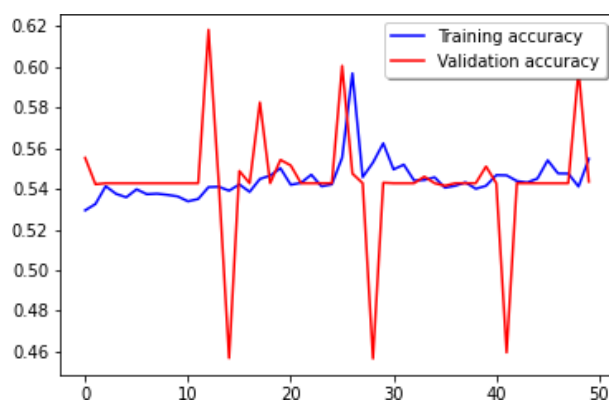


Рисунок 18

Подобные результаты и у метрики площади под кривой PR на рисунке 19. Максимальное значение метрики уменьшилось на 30%.

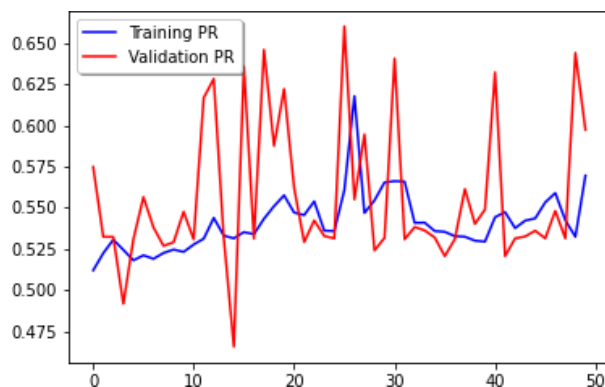


Рисунок 19

Итак, по количественным результатам, гипотеза об увеличении свертки отклоняется. Причем, судя по графикам метрик, модель попала в локальный минимум на обучающей выборке, так как показатели тестовой выборки в среднем выше тренировочной.

Гипотеза об уменьшении свертки

Продолжая эксперименты с размерностью ядра, уменьшим его, что позволит уменьшить количество обучаемых параметров модели и обрабатывать больше данных на последующих слоях.

Показатели функции потерь на рисунке 20 говорят о равномерной оптимизации параметров модели, что говорит об оптимальном состоянии сверточной нейронной сети.

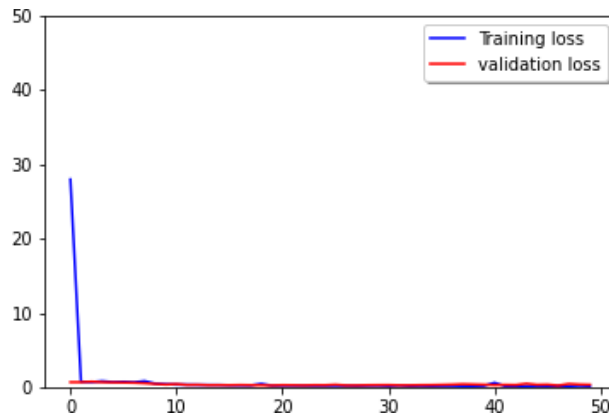


Рисунок 20

Точность (ассигасу) на рисунке 21 имеет уже знакомый вид, причем максимальное значение метрики еще больше приблизилось к заложенной асимптоте. Но увеличивающийся, с номером эпохи, разрыв между показателем тренировочной и тестовой выборки может говорить о переобучении, что требует дополнительного исследования.

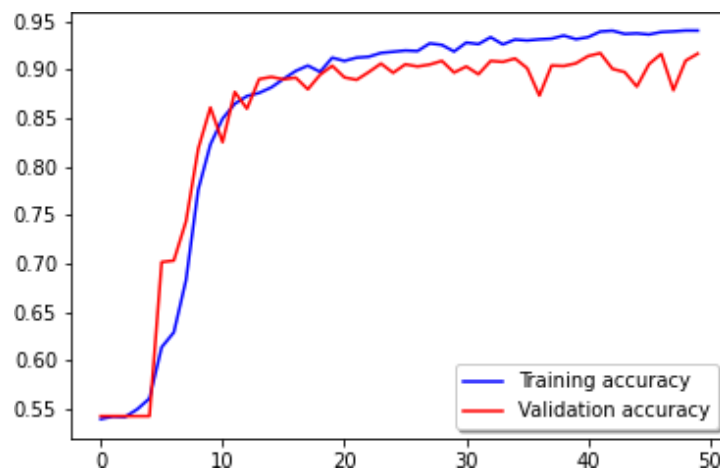


Рисунок 21

На графике площади под кривой PR на рисунке 22 так же присутствует увеличивающееся отклонение между метрикой на тренировочной выборке от тестовой, что так же может говорить о переобучении.

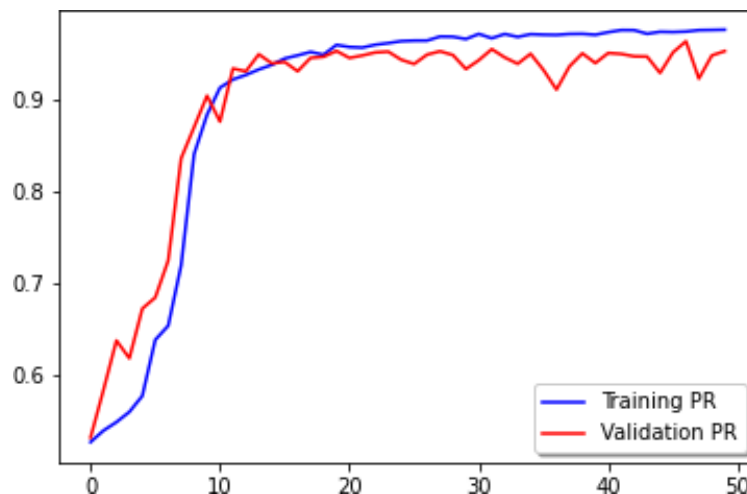


Рисунок 22

По результат метрик гипотеза принимается и будет исследоваться дальше.

Гипотеза об увеличении количества эпох при уменьшенном ядре свертки

Так как результаты предыдущего исследования говорят о возможном переобучении, нужно проверить этот аспект путем увеличения количества эпох.

Обучив модель на протяжении 100 эпох, были получены следующие результаты.

На графике значений функции ошибки на рисунке 23 неожиданно появились аномальные скачки, даже там, где до этого они не появлялись, что можно обусловить случайным шумом.

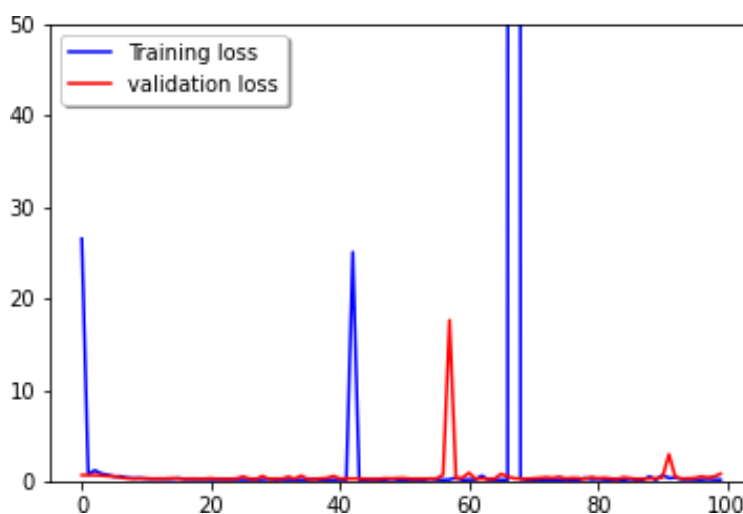


Рисунок 23

На графике метрики точности (ассигасу) на рисунке 24 можно наблюдать, что переобучение отсутствует, а максимальное значение точности еще больше приблизилось к асимптотике в 92.7%

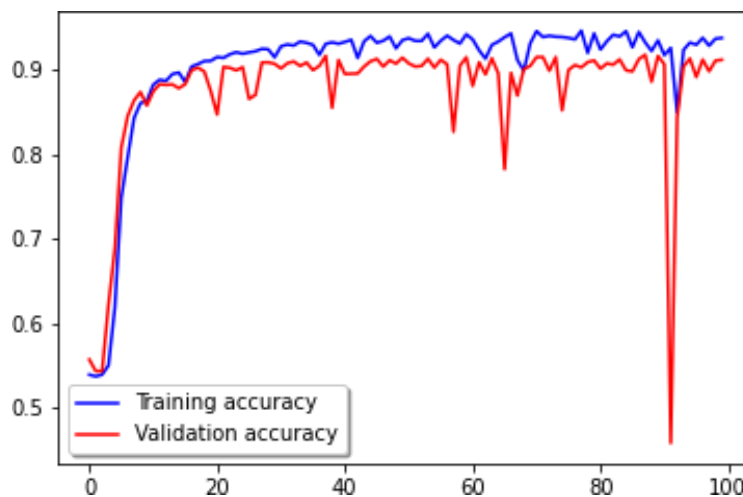


Рисунок 24

По графику значения площади под кривой PR на рисунке 25 так же наблюдается отсутствие переобучения.

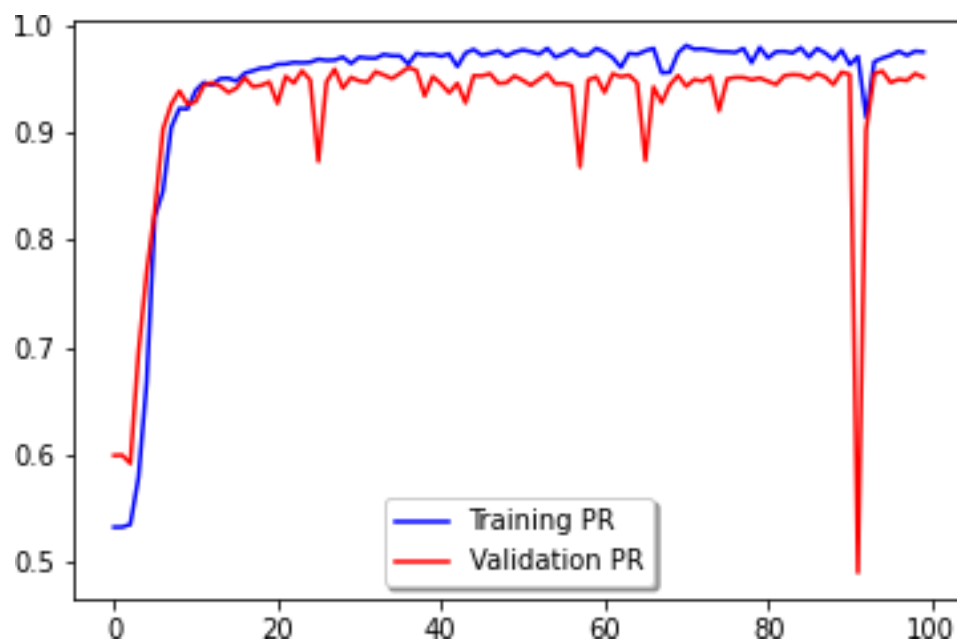


Рисунок 25

Увеличение эпох опровергло подозрение о переобучении, а также доказало, что предел значений метрик все еще не достигнут. Причем по кривым метрик для обучающей и тестовой выборке видна некоторая зависимость, что является положительным аспектом.

Гипотеза о достижении заложенной асимптоты

Проведя ряд исследований, мы определили оптимальные параметры модели и приблизились к заложенной в структуру данной сверточной нейронной сети асимптоте по точности (ассурасу) и появилась идея совершить масштабное исследование и обучить модель на протяжении 1000 эпох.

Были получены следующие результаты. Показатели функции потерь на рисунке 26 говорят о достижении оптимального устойчивого состояния весов к 200 – й эпохе, после этого порога возрастает количество аномальных скачков и вид кривой принимает колебательный характер, а после достижения 700 – й эпохи, судя по графику, модель попадает в локальный минимум, откуда ей не удастся выбраться все оставшееся время обучения, из – за чего ожидается и ухудшение показателей метрик после 700 – й эпохи.

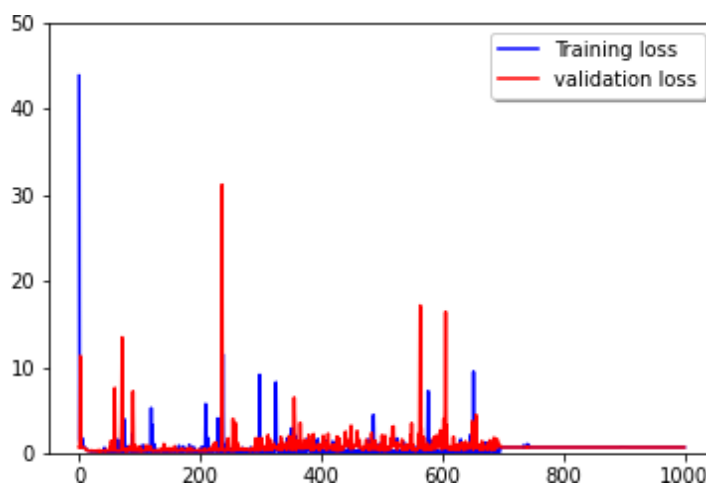


Рисунок 26

Как и ожидалось, график метрики точности (ассурасу) на рисунке 27 подтверждает состояние локального минимума у параметров модели и на графике все же отчетливо видно расхождение кривой обучающей и тестовой выборке, что говорит все же о наличие переобучения, но нельзя не отметить обновленное максимальное значение метрики в 92.5%, что является максимум по

итогах исследований. В данном случае, бороться с переобучением может помочь увеличение объема обучающей выборки.

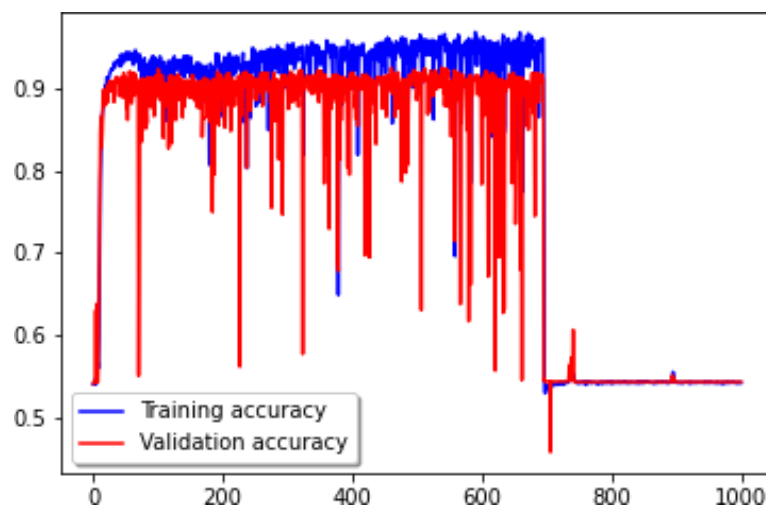


Рисунок 27

График площади под кривой PR на рисунке 28 так же отчетливо отражает попадание в локальный минимум и присутствие переобучения, причем оно здесь более ярко выражено.

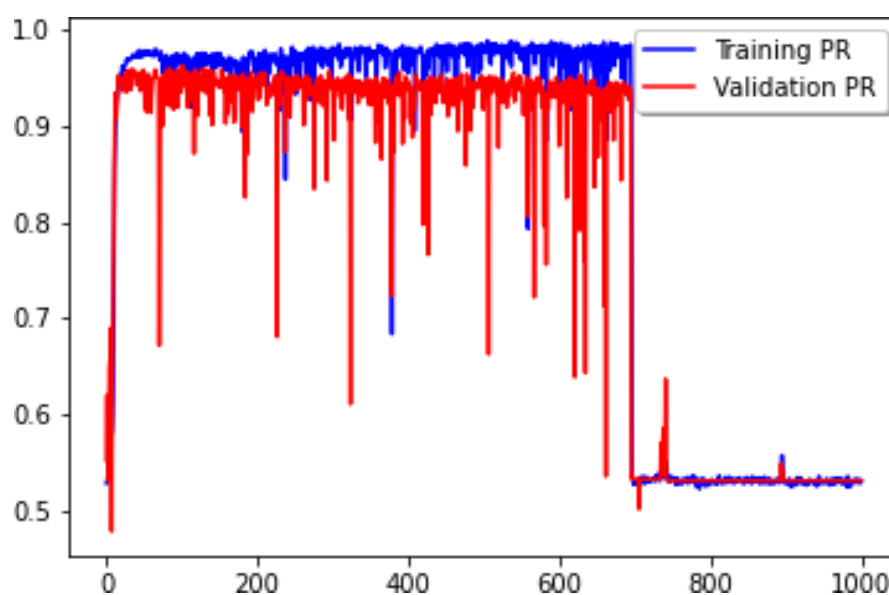


Рисунок 28

Хоть данное исследование и позволило достичь нового предела метрики для тестовой выборки, но результат вряд ли нивелирует временные затраты, так что обучение на протяжении 1000 эпох можно считать избыточным, и гипотеза отклоняется.

Первые результаты задачи семантической сегментации

Исследовав задачу бинарной классификации, стоит так же упомянуть и результаты первых результатов решения задачи семантической сегментации.

В данном решении используется уже упомянутая раньше модель сверточной нейронной сети U – Net, структуру которой мы отобразим в приложении. Обучение сети проходит на том же наборе данных, что и для бинарной классификации, на протяжении 50 – ти эпох, а параметры модели оптимизировали уже знакомым методом RMSProp.

Так как модель только предстоит исследовать в дипломной работе, не будем обращать внимание на метрики, а продемонстрируем ее работу.

На рисунке 29 отображен результат работы данной сети. В первом столбце находится исходное изображение. Во втором столбце отображен результат обработки исходного изображения моделью, где каждому пикселю изображения ставится в соответствие вероятность того, что этот пиксель принадлежит кораблю, чем краснее – тем больше вероятность. В третьем столбце предыдущий результат проходит через фильтр, оставляя лишь пиксели, превышающие порог. В последнем столбце отображен истинный ответ.

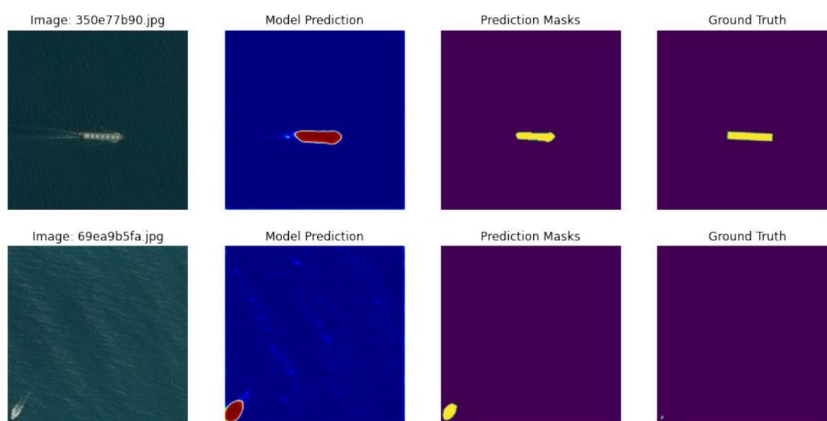


Рисунок 29

Как можно судить по рисунку 29, модель научилась распознавать изображения на эталонных изображениях – когда корабль четко отличим от фона.

На рисунке 30 видно, что модель хорошо распознает корабли даже на тех изображениях, которые имеют некорректный истинный ответ.

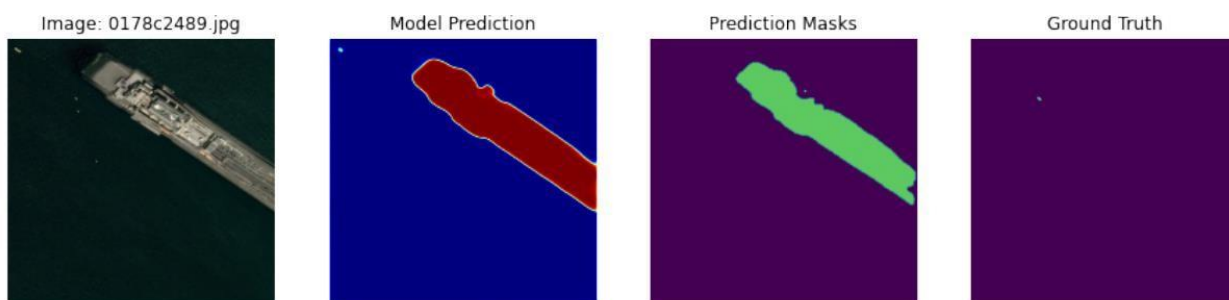


Рисунок 30

Однако, на зашумленных изображениях, которое присутствует на рисунке 31, модель ведет себя некорректно и не может даже близко пометить истинное местоположение морского судна. Борьба с такой проблемой нужно «обогащением» обучающей выборки, то есть добавлением в нее зашумленных изображений и удалением изображений, на которых корабль отсутствует.

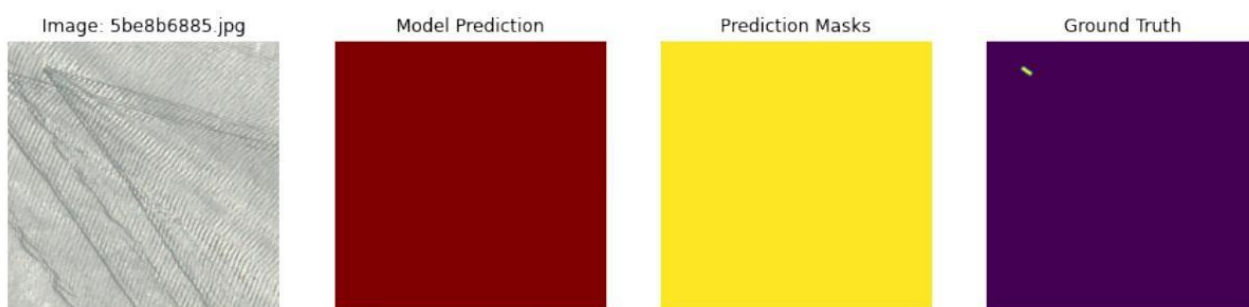


Рисунок 31

Заключение

В ходе преддипломной практики была поставлена задача бинарной классификации изображений на аэрофотоснимках, а также задача семантической сегментации на аэрофотоснимках.

Задача бинарной классификации была исследована и были успешно подобраны параметры модели, предоставляющие наилучший результат по метрическим показателям. Помимо качественного улучшения работы модели, также удалось ее оптимизировать, сократив количество параметров модели до 16 млн., что является долей в 66% от начального количества параметров. Это позволяет занять ограниченные объемы оперативной памяти полезной нагрузкой. Например, увеличением количества сверточных слоев, что должно положительно сказаться на метрических показателях работы данной модели.

Так же были совершены первые попытки в решение задачи семантической сегментации, которая будет более детально исследована уже в дипломной работе.

Список литературы

1. The 2st-unet for pneumothorax segmentation in chest x-rays using resnet34 as a backbone for u-net. arXiv:2009.02805v1 [eess.IV] 6 Sep 2020
2. U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597v1 [cs.CV] 18 May 2015
3. Colorectal Cancer Segmentation using AtrousConvolution and Residual Enhanced UNet. arXiv:2103.09289v1 [eess.IV] 16 Mar 2021
4. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. arXiv:1606.04797v1 [cs.CV] 15 Jun 2016
5. Глубокое обучение / Ян Гудфеллоу, Йошуа Бенджио, Аарон Курвилль // ДМК Пресс, 2018г., второе цветное издание, исправленное.
6. Глубокое обучение. / Николенко С., Кадури А., Архангельская Е. // СПб: Питер, 2018. — 480 с.: ил. — (Серия «Библиотека программиста»).
7. Very deep convolutional networks for large-scale image recognition / Karen Simonyan, Andrew Zisserman
8. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998.

Приложения

Приложение 1

Программная реализация исследования гипотез для задачи бинарной классификации

Оптимизация RMSProp

```
from google.colab import drive
drive.mount('/content/drive')

import tensorflow as tf
import os
import gc
import numpy as np
import pandas as pd
import time
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from PIL import Image
SEED = 42

data = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Input/data_
data_target = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Inpu

print("array sizes of data array: ", data.shape)
print("array sizes of target array: ", data_target.shape)
print("example of one image in data array\n", data[0])
print("example of target for one image in array: ", data_target[0])

#Set target to one hot target for classification problem
from sklearn.preprocessing import OneHotEncoder
targets = data_target.reshape(len(data_target),-1)
enc = OneHotEncoder()
enc.fit(targets)
targets = enc.transform(targets).toarray()
print(targets.shape)
del data_target
targets

#Split Training data to training data and validate data to detect overfi
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(data, targets, test_siz
x_train.shape, x_val.shape, y_train.shape, y_val.shape

#Data augumation
from keras.preprocessing.image import ImageDataGenerator
img_gen = ImageDataGenerator()

#Load ResNet50 model with Keras
#from keras.applications.vgg16 import VGG16 as PTModel, preprocess_input
#from keras.applications.densenet import DenseNet169 as PTModel, preproc
#from keras.applications.resnet50 import ResNet50 as ResModel
#from keras.applications.vgg16 import VGG16 as VGG16Model
#img_width, img_height = 256, 256
#model = VGG16Model(weights = 'imagenet', include_top=False, input_shape

gc.collect()

#On this case, we only need predict 2 category (1. have ship, 2. no ship
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Bat
from keras.callbacks import ModelCheckpoint
```

```

# creating the final model
model = Sequential()

model.add(Conv2D(filters = 64, kernel_size = (3, 3), padding = 'Same', a
model.add(Conv2D(filters = 64, kernel_size = (3, 3), padding = 'Same', a
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 128, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 128, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 256, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 256, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 256, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(1024, activation = "relu"))
model.add(Dense(2, activation = "softmax"))

filepath="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1
callbacks_list = [checkpoint]

image_file = 'model_1.png'
tf.keras.utils.plot_model(model, to_file = image_file, show_shapes = Tru

```

```

In [ ]: #Set Hyperparameter and Start training
from keras import optimizers
from keras.optimizers import RMSprop
epochs = 50
lr = 0.001 #learning rate
batch_size = 256
decay = lr/epochs # Learning rate decay over each update
optimizer = RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e-08, decay = deca
#model.load_weights("transfer_ship_v1_5.h5")
model.compile(loss = 'categorical_crossentropy', optimizer = optimizer,
#model_final.summary()

```

```

In [ ]: model.summary()

```

```

In [ ]: gc.collect()

```

```

In [ ]: history = model.fit(img_gen.flow(x_train, y_train, batch_size = batch_si
        steps_per_epoch = int(len(x_train)/batch_size)
model.save('transfer_ship_exp.h5')

```

```

In [ ]: import matplotlib.pyplot as plt

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)

```

```
plt.show()

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accurac")
plt.plot(history.history['val_accuracy'], color='r', label="Validation ac")
legend = plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.show()
```

```
In [ ]: gc.collect()
```

```
In [ ]: plot_path = "/content/drive/MyDrive/Диплом/Ship_detection/CNN_PRACT/PLOT"
plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.savefig(plot_path + "loss.png")
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accurac")
plt.plot(history.history['val_accuracy'], color='r', label="Validation ac")
legend = plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "acc.png")
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "roc.png")
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "pr.png")
plt.show()
```

```
In [ ]:
```

Гипотеза о применении оптимизатора ADAM

```
In [ ]: from google.colab import drive
drive.mount('/content/drive')
```

```
In [ ]: import tensorflow as tf
import os
import gc
import numpy as np
import pandas as pd
import time
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from PIL import Image
SEED = 42
```

```
In [ ]: data = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Input/data_
data_target = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Inpu
```

```
In [ ]: print("array sizes of data array: ", data.shape)
print("array sizes of target array: ", data_target.shape)
print("example of one image in data array\n", data[0])
print("example of target for one image in array: ", data_target[0])
```

```
In [ ]: #Set target to one hot target for classification problem
from sklearn.preprocessing import OneHotEncoder
targets = data_target.reshape(len(data_target), -1)
enc = OneHotEncoder()
enc.fit(targets)
targets = enc.transform(targets).toarray()
print(targets.shape)
del data_target
targets
```

```
In [ ]: #Split Training data to training data and validate data to detect overfi
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(data, targets, test_siz
x_train.shape, x_val.shape, y_train.shape, y_val.shape
```

```
In [ ]: #Data augumatation
from keras.preprocessing.image import ImageDataGenerator
img_gen = ImageDataGenerator()
```

```
In [ ]: #Load ResNet50 model with Keras
#from keras.applications.vgg16 import VGG16 as PTModel, preprocess_input
#from keras.applications.densenet import DenseNet169 as PTModel, preproc
#from keras.applications.resnet50 import ResNet50 as ResModel
#from keras.applications.vgg16 import VGG16 as VGG16Model
#img_width, img_height = 256, 256
#model = VGG16Model(weights = 'imagenet', include_top=False, input_shape
```

```
In [ ]: gc.collect()
```

```
In [ ]: #On this case, we only need predict 2 category (1. have ship, 2. no ship
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Bat
from keras.callbacks import ModelCheckpoint

# creating the final model
model = Sequential()

model.add(Conv2D(filters = 64, kernel_size = (3, 3), padding = 'Same', a
model.add(Conv2D(filters = 64, kernel_size = (3, 3), padding = 'Same', a
model.add(MaxPool2D(pool_size=(2, 2)))
```

```

model.add(Conv2D(filters = 128, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 128, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 256, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 256, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 256, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(1024, activation = "relu"))
model.add(Dense(2, activation = "softmax"))

filepath="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1
callbacks_list = [checkpoint]

image_file = 'model_1.png'
tf.keras.utils.plot_model(model, to_file = image_file, show_shapes = True

```

```

In [ ]: #Set Hyperparameter and Start training
from keras import optimizers
from keras.optimizers import RMSprop, Adam
epochs = 50
lr = 0.001 #learning rate
batch_size = 256
decay = lr/epochs # Learning rate decay over each update
#optimizer = RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e-08, decay = dec
optimizer = Adam(learning_rate = 0.001, epsilon = 1e-08, decay = decay)
#model.load_weights("transfer_ship_v1_5.h5")
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', met
#model_final.summary()

```

```

In [ ]: model.summary()

```

```

In [ ]: gc.collect()

```

```

In [ ]: history = model.fit(img_gen.flow(x_train, y_train, batch_size = batch_si
steps_per_epoch = int(len(x_train)/batch_size)
model.save('transfer_ship_exp.h5')

```

```

In [ ]: import matplotlib.pyplot as plt

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.show()

```

```
plt.plot(history.history['accuracy'], color='b', label="Training accuracy")
plt.plot(history.history['val_accuracy'], color='r', label="Validation accuracy")
legend = plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc_2'], color='b', label="Training ROC")
plt.plot(history.history['val_auc_2'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc_3'], color='b', label="Training PR")
plt.plot(history.history['val_auc_3'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.show()
```

```
In [ ]: gc.collect()
```

```
In [ ]: plot_path = "/content/drive/MyDrive/Диплом/Ship_detection/CNN_PRACT/PLOT"
plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.savefig(plot_path + "loss.png")
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accuracy")
plt.plot(history.history['val_accuracy'], color='r', label="Validation accuracy")
legend = plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "acc.png")
plt.show()

plt.plot(history.history['auc_2'], color='b', label="Training ROC")
plt.plot(history.history['val_auc_2'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "roc.png")
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "pr.png")
plt.show()
```

```
In [ ]:
```

ADAM с иной функцией потерь

```
In [ ]: from google.colab import drive
drive.mount('/content/drive')
```

```
In [ ]: import tensorflow as tf
import os
import gc
import numpy as np
import pandas as pd
import time
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
```

```
from PIL import Image
SEED = 42
```

```
In [ ]: data = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Input/data_
data_target = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Inpu
```

```
In [ ]: print("array sizes of data array: ", data.shape)
print("array sizes of target array: ", data_target.shape)
print("example of one image in data array\n", data[0])
print("example of target for one image in array: ", data_target[0])
```

```
In [ ]: #Set target to one hot target for classification problem
from sklearn.preprocessing import OneHotEncoder
targets = data_target.reshape(len(data_target), -1)
enc = OneHotEncoder()
enc.fit(targets)
targets = enc.transform(targets).toarray()
print(targets.shape)
del data_target
targets
```

```
In [ ]: #Split Training data to training data and validate data to detect overfi
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(data, targets, test_siz
x_train.shape, x_val.shape, y_train.shape, y_val.shape
```

```
In [ ]: #Data augumation
from keras.preprocessing.image import ImageDataGenerator
img_gen = ImageDataGenerator()
```

```
In [ ]: #Load ResNet50 model with Keras
#from keras.applications.vgg16 import VGG16 as PTModel, preprocess_input
#from keras.applications.densenet import DenseNet169 as PTModel, preproc
#from keras.applications.resnet50 import ResNet50 as ResModel
#from keras.applications.vgg16 import VGG16 as VGG16Model

```

```
In [ ]: gc.collect()
```

```
In [ ]: #On this case, we only need predict 2 category (1. have ship, 2. no ship
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Bat
from keras.callbacks import ModelCheckpoint
```

```
# creating the final model
model = Sequential()
```

```
model.add(Conv2D(filters = 64, kernel_size = (3, 3), padding = 'Same', a
model.add(Conv2D(filters = 64, kernel_size = (3, 3), padding = 'Same', a
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 128, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 128, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 256, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 256, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 256, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
```

```

model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(Conv2D(filters = 512, kernel_size = (3, 3), padding = 'Same',
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(1024, activation = "relu"))
model.add(Dense(2, activation = "softmax"))

filepath="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1
callbacks_list = [checkpoint]

image_file = 'model_1.png'
tf.keras.utils.plot_model(model, to_file = image_file, show_shapes = True

```

```

In [ ]: #Set Hyperparameter and Start training
from keras import optimizers
from keras.optimizers import RMSprop, Adam
epochs = 50
lr = 0.001 #learning rate
batch_size = 256
decay = lr/epochs # Learning rate decay over each update
#optimizer = RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e-08, decay = dec
optimizer = Adam(learning_rate = 0.001, epsilon = 1e-08, decay = decay)
#model.load_weights("transfer_ship_v1_5.h5")
model.compile(loss = "binary_crossentropy", optimizer = 'adam', metrics
#model_final.summary()

```

```

In [ ]: model.summary()

```

```

In [ ]: gc.collect()

```

```

In [ ]: history = model.fit(img_gen.flow(x_train, y_train, batch_size = batch_si
steps_per_epoch = int(len(x_train)/batch_size)
model.save('transfer_ship_exp.h5')

```

```

In [ ]: import matplotlib.pyplot as plt

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accurac
plt.plot(history.history['val_accuracy'], color='r',label="Validation ac
legend = plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r',label="Validation ROC")

```



```
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.show()
```

```
In [ ]: gc.collect()
```

```
In [ ]: plot_path = "/content/drive/MyDrive/Диплом/Ship_detection/CNN_PRACT/PLOT"
plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.savefig(plot_path + "loss.png")
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accurac")
plt.plot(history.history['val_accuracy'], color='r', label="Validation ac")
legend = plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "acc.png")
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "roc.png")
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "pr.png")
plt.show()
```

```
In [ ]:
```

RMSProp с иной функцией потерь и увеличением ядра свертки

```
In [ ]: from google.colab import drive
drive.mount('/content/drive')
```

```
In [ ]: import tensorflow as tf
import os
import gc
import numpy as np
import pandas as pd
import time
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from PIL import Image
SEED = 42
```

```
In [ ]: data = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Input/data_')
data_target = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Inpu')
```

```
In [ ]: print("array sizes of data array: ", data.shape)
        print("array sizes of target array: ", data_target.shape)
        print("example of one image in data array\n", data[0])
        print("example of target for one image in array: ", data_target[0])
```

```
In [ ]: #Set target to one hot target for classification problem
        from sklearn.preprocessing import OneHotEncoder
        targets = data_target.reshape(len(data_target), -1)
        enc = OneHotEncoder()
        enc.fit(targets)
        targets = enc.transform(targets).toarray()
        print(targets.shape)
        del data_target
        targets
```

```
In [ ]: #Split Training data to training data and validate data to detect overfi
        from sklearn.model_selection import train_test_split
        x_train, x_val, y_train, y_val = train_test_split(data, targets, test_siz
        x_train.shape, x_val.shape, y_train.shape, y_val.shape
```

```
In [ ]: #Data augumation
        from keras.preprocessing.image import ImageDataGenerator
        img_gen = ImageDataGenerator()
```

```
In [ ]: #Load ResNet50 model with Keras
        #from keras.applications.vgg16 import VGG16 as PTModel, preprocess_input
        #from keras.applications.densenet import DenseNet169 as PTModel, preproc
        #from keras.applications.resnet50 import ResNet50 as ResModel
        #from keras.applications.vgg16 import VGG16 as VGG16Model
        #img_width, img_height = 256, 256
        #model = VGG16Model(weights = 'imagenet', include_top=False, input_shape
```

```
In [ ]: gc.collect()
```

```
In [ ]: #On this case, we only need predict 2 category (1. have ship, 2. no ship
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Bat
        from keras.callbacks import ModelCheckpoint
        base_filter_count = 64
        kernel = (5, 5)
        # creating the final model
        model = Sequential()

        model.add(Conv2D(filters = base_filter_count, kernel_size = kernel, padd
        model.add(Conv2D(filters = base_filter_count, kernel_size = kernel, padd
        model.add(MaxPool2D(pool_size=(2, 2)))
        model.add(Conv2D(filters = base_filter_count * 2, kernel_size = kernel,
        model.add(Conv2D(filters = base_filter_count * 2, kernel_size = kernel,
        model.add(MaxPool2D(pool_size=(2, 2)))
        model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel,
        model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel,
        model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel,
        model.add(MaxPool2D(pool_size=(2, 2)))
        model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
        model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
        model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
        model.add(MaxPool2D(pool_size=(2, 2)))
        model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
        model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
        model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
        model.add(MaxPool2D(pool_size=(2, 2)))
```

```

model.add(Flatten())
model.add(Dense(1024, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(1024, activation = "relu"))
model.add(Dense(2, activation = "softmax"))

filepath="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1
callbacks_list = [checkpoint]

image_file = 'model_1.png'
tf.keras.utils.plot_model(model, to_file = image_file, show_shapes = True

```

```

In [ ]:      #Set Hyperparameter and Start training
from keras import optimizers
from keras.optimizers import RMSprop, Adam
epochs = 50
lr = 0.001 #learning rate
batch_size = 256
decay = lr/epochs # Learning rate decay over each update
optimizer = RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e-08, decay = decay)
#optimizer = Adam(learning_rate = 0.001, epsilon = 1e-08, decay = decay)
#model.load_weights("transfer_ship_v1_5.h5")
model.compile(loss = "binary_crossentropy", optimizer = optimizer, metrics=
#model_final.summary()

```

```

In [ ]: model.summary()

```

```

In [ ]: gc.collect()

```

```

In [ ]: history = model.fit(img_gen.flow(x_train, y_train, batch_size = batch_size,
steps_per_epoch = int(len(x_train)/batch_size)
model.save('transfer_ship_exp.h5')

```

```

In [ ]: import matplotlib.pyplot as plt

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accuracy")
plt.plot(history.history['val_accuracy'], color='r', label="Validation accuracy")
legend = plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.show()

```

```
In [ ]: gc.collect()
```

```
In [ ]: plot_path = "/content/drive/MyDrive/Диплом/Ship_detection/CNN_PRACT/PLOT"
plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.savefig(plot_path + "loss.png")
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accuracy")
plt.plot(history.history['val_accuracy'], color='r', label="Validation accuracy")
legend = plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "acc.png")
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "roc.png")
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "pr.png")
plt.show()
```

```
In [ ]:
```

```
In [ ]:
```

RMSProp с иной функцией потерь и уменьшением ядра свертки

```
In [ ]: from google.colab import drive
drive.mount('/content/drive')
```

```
In [ ]: import tensorflow as tf
import os
import gc
import numpy as np
import pandas as pd
import time
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from PIL import Image
SEED = 42
```

```
In [ ]: data = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Input/data_')
data_target = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Inpu')
```

```
In [ ]: print("array sizes of data array: ", data.shape)
print("array sizes of target array: ", data_target.shape)
print("example of one image in data array\n", data[0])
print("example of target for one image in array: ", data_target[0])
```

```
In [ ]: #Set target to one hot target for classification problem
from sklearn.preprocessing import OneHotEncoder
targets = data_target.reshape(len(data_target),-1)
enc = OneHotEncoder()
enc.fit(targets)
targets = enc.transform(targets).toarray()
print(targets.shape)
del data_target
targets
```

```
In [ ]: #Split Training data to training data and validate data to detect overfi
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(data,targets, test_siz
x_train.shape, x_val.shape, y_train.shape, y_val.shape
```

```
In [ ]: #Data augumatation
from keras.preprocessing.image import ImageDataGenerator
img_gen = ImageDataGenerator()
```

```
In [ ]: #Load ResNet50 model with Keras
#from keras.applications.vgg16 import VGG16 as PTModel, preprocess_input
#from keras.applications.densenet import DenseNet169 as PTModel, preproc
#from keras.applications.resnet50 import ResNet50 as ResModel
#from keras.applications.vgg16 import VGG16 as VGG16Model


```

```
In [ ]: gc.collect()
```

```
In [ ]: #On this case, we only need predict 2 category (1. have ship, 2. no ship
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Bat
from keras.callbacks import ModelCheckpoint
base_filter_count = 64
kernel = (2, 2)
# creating the final model
model = Sequential()

model.add(Conv2D(filters = base_filter_count, kernel_size = kernel, padd
model.add(Conv2D(filters = base_filter_count, kernel_size = kernel, padd
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 2, kernel_size = kernel,
model.add(Conv2D(filters = base_filter_count * 2, kernel_size = kernel,
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel,
model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel,
model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel,
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel,
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(1024, activation = "relu"))
model.add(Dense(2, activation = "softmax"))
```

```

filepath="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1
callbacks_list = [checkpoint]

image_file = 'model_1.png'
tf.keras.utils.plot_model(model, to_file = image_file, show_shapes = True

```

```

In [ ]:      #Set Hyperparameter and Start training
from keras import optimizers
from keras.optimizers import RMSprop, Adam
epochs = 50
lr = 0.001 #learning rate
batch_size = 256
decay = lr/epochs # Learning rate decay over each update
optimizer = RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e-08, decay = deca
#optimizer = Adam(learning_rate = 0.001, epsilon = 1e-08, decay = decay)
#model.load_weights("transfer_ship_v1_5.h5")
model.compile(loss = "binary_crossentropy", optimizer = optimizer, metri
#model_final.summary()

```

```

In [ ]: model.summary()

```

```

In [ ]: gc.collect()

```

```

In [ ]: history = model.fit(img_gen.flow(x_train, y_train, batch_size = batch_si
steps_per_epoch = int(len(x_train)/batch_size)
model.save('transfer_ship_exp.h5')

```

```

In [ ]: import matplotlib.pyplot as plt

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accurac
plt.plot(history.history['val_accuracy'], color='r',label="Validation ac
legend = plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r',label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r',label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.show()

```

```

In [ ]: gc.collect()

```

```

In [ ]: plot_path = "/content/drive/MyDrive/Диплом/Ship_detection/CNN_PRACT/PLOT
plt.plot(history.history['loss'], color='b', label="Training loss")

```

```
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.savefig(plot_path + "loss.png")
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accuracy")
plt.plot(history.history['val_accuracy'], color='r', label="Validation accuracy")
plt.legend = plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "acc.png")
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "roc.png")
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "pr.png")
plt.show()
```

In []:

RMSProp с иной функцией потерь и уменьшением ядра свертки, обучение 100 эпох

```
In [ ]: from google.colab import drive
drive.mount('/content/drive')
```

```
In [ ]: import tensorflow as tf
import os
import gc
import numpy as np
import pandas as pd
import time
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from PIL import Image
SEED = 42
```

```
In [ ]: data = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Input/data_')
data_target = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Inpu')
```

```
In [ ]: print("array sizes of data array: ", data.shape)
print("array sizes of target array: ", data_target.shape)
print("example of one image in data array\n", data[0])
print("example of target for one image in array: ", data_target[0])
```

```
In [ ]: #Set target to one hot target for classification problem
from sklearn.preprocessing import OneHotEncoder
targets = data_target.reshape(len(data_target), -1)
enc = OneHotEncoder()
enc.fit(targets)
targets = enc.transform(targets).toarray()
```

```
print(targets.shape)
del data_target
targets
```

```
In [ ]: #Split Training data to training data and validate data to detect overfi
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(data, targets, test_size=0.2,
x_train.shape, x_val.shape, y_train.shape, y_val.shape)
```

```
In [ ]: #Data augumatation
from keras.preprocessing.image import ImageDataGenerator
img_gen = ImageDataGenerator()
```

```
In [ ]: #Load ResNet50 model with Keras
#from keras.applications.vgg16 import VGG16 as PTModel, preprocess_input
#from keras.applications.densenet import DenseNet169 as PTModel, preprocess_input
#from keras.applications.resnet50 import ResNet50 as ResModel
#from keras.applications.vgg16 import VGG16 as VGG16Model
#img_width, img_height = 256, 256
#model = VGG16Model(weights = 'imagenet', include_top=False, input_shape=(img_height, img_width, 3))
```

```
In [ ]: gc.collect()
```

```
In [ ]: #On this case, we only need predict 2 category (1. have ship, 2. no ship)
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from keras.callbacks import ModelCheckpoint
base_filter_count = 64
kernel = (2, 2)
# creating the final model
model = Sequential()

model.add(Conv2D(filters = base_filter_count, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 2, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 2, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(1024, activation = "relu"))
model.add(Dense(2, activation = "softmax"))

filepath="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
```



```
image_file = 'model_1.png'  
tf.keras.utils.plot_model(model, to_file = image_file, show_shapes = True
```

```
In [ ]:      #Set Hyperparameter and Start training  
from keras import optimizers  
from keras.optimizers import RMSprop, Adam  
epochs = 100  
lr = 0.001 #learning rate  
batch_size = 256  
decay = lr/epochs # Learning rate decay over each update  
optimizer = RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e-08, decay = decay)  
#optimizer = Adam(learning_rate = 0.001, epsilon = 1e-08, decay = decay)  
#model.load_weights("transfer_ship_v1_5.h5")  
model.compile(loss = "binary_crossentropy", optimizer = optimizer, metri  
#model_final.summary()
```

```
In [ ]: model.summary()
```

```
In [ ]: gc.collect()
```

```
In [ ]: history = model.fit(img_gen.flow(x_train, y_train, batch_size = batch_si  
steps_per_epoch = int(len(x_train)/batch_size)  
model.save('transfer_ship_exp.h5')
```

```
In [ ]: import matplotlib.pyplot as plt  
  
plt.plot(history.history['loss'], color='b', label="Training loss")  
plt.plot(history.history['val_loss'], color='r', label="validation loss")  
plt.legend(loc='best', shadow=True)  
plt.show()  
  
plt.plot(history.history['loss'], color='b', label="Training loss")  
plt.plot(history.history['val_loss'], color='r', label="validation loss")  
plt.legend(loc='best', shadow=True)  
plt.ylim(0, 50)  
plt.show()  
  
plt.plot(history.history['accuracy'], color='b', label="Training accurac  
plt.plot(history.history['val_accuracy'], color='r',label="Validation ac  
legend = plt.legend(loc='best', shadow=True)  
plt.show()  
  
plt.plot(history.history['auc'], color='b', label="Training ROC")  
plt.plot(history.history['val_auc'], color='r',label="Validation ROC")  
plt.legend(loc='best', shadow=True)  
plt.show()  
  
plt.plot(history.history['auc_1'], color='b', label="Training PR")  
plt.plot(history.history['val_auc_1'], color='r',label="Validation PR")  
plt.legend(loc='best', shadow=True)  
plt.show()
```

```
In [ ]: gc.collect()
```

```
In [ ]: plot_path = "/content/drive/MyDrive/Диплом/Ship_detection/CNN_PRACT/PLOT  
plt.plot(history.history['loss'], color='b', label="Training loss")  
plt.plot(history.history['val_loss'], color='r', label="validation loss")  
plt.legend(loc='best', shadow=True)  
plt.ylim(0, 50)  
plt.savefig(plot_path + "loss.png")  
plt.show()
```

```
plt.plot(history.history['accuracy'], color='b', label="Training accuracy")
plt.plot(history.history['val_accuracy'], color='r', label="Validation accuracy")
legend = plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "acc.png")
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "roc.png")
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "pr.png")
plt.show()
```

In []:

RMSProp с иной функцией потерь и уменьшением ядра свертки, обучение 1000 эпох

In []: `from google.colab import drive`
`drive.mount('/content/drive')`

In []: `import tensorflow as tf`
`import os`
`import gc`
`import numpy as np`
`import pandas as pd`
`import time`
`from tensorflow.compat.v1 import ConfigProto`
`from tensorflow.compat.v1 import InteractiveSession`
`from PIL import Image`
`SEED = 42`

In []: `data = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Input/data_`
`data_target = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Inpu`

In []: `print("array sizes of data array: ", data.shape)`
`print("array sizes of target array: ", data_target.shape)`
`print("example of one image in data array\n", data[0])`
`print("example of target for one image in array: ", data_target[0])`

In []: `#Set target to one hot target for classification problem`
`from sklearn.preprocessing import OneHotEncoder`
`targets = data_target.reshape(len(data_target), -1)`
`enc = OneHotEncoder()`
`enc.fit(targets)`
`targets = enc.transform(targets).toarray()`
`print(targets.shape)`
`del data_target`
`targets`

`#Split Training data to training data and validate data to detect overfi`

```
In [ ]: from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(data, targets, test_size=0.2,
x_train.shape, x_val.shape, y_train.shape, y_val.shape
```

```
In [ ]: #Data augumatation
from keras.preprocessing.image import ImageDataGenerator
img_gen = ImageDataGenerator()
```

```
In [ ]: #Load ResNet50 model with Keras
#from keras.applications.vgg16 import VGG16 as PTModel, preprocess_input
#from keras.applications.densenet import DenseNet169 as PTModel, preprocess_input
#from keras.applications.resnet50 import ResNet50 as ResModel
#from keras.applications.vgg16 import VGG16 as VGG16Model
#img_width, img_height = 256, 256
#model = VGG16Model(weights = 'imagenet', include_top=False, input_shape=(img_height, img_width, 3))
```

```
In [ ]: gc.collect()
```

```
In [ ]: #On this case, we only need predict 2 category (1. have ship, 2. no ship)
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from keras.callbacks import ModelCheckpoint

base_filter_count = 64
kernel = (2, 2)
# creating the final model
model = Sequential()

model.add(Conv2D(filters = base_filter_count, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 2, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 2, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 4, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = base_filter_count * 8, kernel_size = kernel, padding = 'same', activation = 'relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(1024, activation = "relu"))
model.add(Dense(2, activation = "softmax"))

filepath="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]

image_file = 'model_1.png'
tf.keras.utils.plot_model(model, to_file = image_file, show_shapes = True)
```

```
In [ ]: #Set Hyperparameter and Start training
from keras import optimizers
from keras.optimizers import RMSprop, Adam
```

```

epochs = 1000
lr = 0.001 #learning rate
batch_size = 256
decay = lr/epochs # Learning rate decay over each update
optimizer = RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e-08, decay = decay)
#optimizer = Adam(learning_rate = 0.001, epsilon = 1e-08, decay = decay)
#model.load_weights("transfer_ship_v1_5.h5")
model.compile(loss = "binary_crossentropy", optimizer = optimizer, metrics = ['accuracy'])
#model_final.summary()

```

```
In [ ]: model.summary()
```

```
In [ ]: gc.collect()
```

```
In [ ]: history = model.fit(img_gen.flow(x_train, y_train, batch_size = batch_size,
                                         steps_per_epoch = int(len(x_train)/batch_size),
                                         validation_data=(x_val, y_val)),
                           epochs = epochs,
                           verbose=1)
model.save('transfer_ship_exp.h5')
```

```
In [ ]: import matplotlib.pyplot as plt

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accuracy")
plt.plot(history.history['val_accuracy'], color='r', label="Validation accuracy")
legend = plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r', label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r', label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.show()

```

```
In [ ]: gc.collect()
```

```
In [ ]: plot_path = "/content/drive/MyDrive/Диплом/Ship_detection/CNN_PRACT/PLOT"
plt.plot(history.history['loss'], color='b', label="Training loss")
plt.plot(history.history['val_loss'], color='r', label="validation loss")
plt.legend(loc='best', shadow=True)
plt.ylim(0, 50)
plt.savefig(plot_path + "loss.png")
plt.show()

plt.plot(history.history['accuracy'], color='b', label="Training accuracy")
plt.plot(history.history['val_accuracy'], color='r', label="Validation accuracy")
legend = plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "acc.png")
plt.show()

```

```
plt.plot(history.history['auc'], color='b', label="Training ROC")
plt.plot(history.history['val_auc'], color='r',label="Validation ROC")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "roc.png")
plt.show()

plt.plot(history.history['auc_1'], color='b', label="Training PR")
plt.plot(history.history['val_auc_1'], color='r',label="Validation PR")
plt.legend(loc='best', shadow=True)
plt.savefig(plot_path + "pr.png")
plt.show()
```

In []:

Базовое решение - подбрасывание МОНЕТЫ

```
In [ ]: import tensorflow as tf
import os
import gc
import numpy as np
import pandas as pd
import time
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from PIL import Image
import random
SEED = 42
```

```
In [ ]: data = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Input/data_
data_target = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Inpu
```

```
In [ ]: print("array sizes of data array: ", data.shape)
print("array sizes of target array: ",data_target.shape)
print("example of one image in data array\n", data[0])
print("example of target for one image in array: ", data_target[0])
```

```
In [ ]: #Set target to one hot target for classification problem
from sklearn.preprocessing import OneHotEncoder
targets = data_target.reshape(len(data_target),-1)
enc = OneHotEncoder()
enc.fit(targets)
targets = enc.transform(targets).toarray()
print(targets.shape)
del data_target
targets
```

```
In [ ]: n = len(targets)
n
```

```
In [ ]: Y = np.array([[0] * 2] * n)
for i in range(n):
    rnd = random.random()
    Y[i] = [round(1 - rnd), round(rnd)]
Y
```

```
In [ ]: from sklearn.metrics import accuracy_score
```

```

print("Точность (accuracy) равно ", accuracy_score(targets, Y))

In [ ]: from sklearn.metrics import precision_score
print("Точность (precision) равно ",
      precision_score(targets.reshape(-1, 1), Y.reshape(-1, 1)))

In [ ]: from sklearn.metrics import recall_score
print("Полнота (recall) равно ",
      recall_score(targets.reshape(-1, 1), Y.reshape(-1, 1)))

In [ ]: from sklearn.metrics import precision_recall_curve
from sklearn import metrics
pr, rc, tr = precision_recall_curve(targets.reshape(-1, 1), np.array([1/
metrics.auc(pr, rc)

In [ ]:

```

Решение задачи человеком

```

In [ ]: import tensorflow as tf
import os
import gc
import numpy as np
import pandas as pd
import time
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from PIL import Image
import random
SEED = 42

In [ ]: data = pd.read_csv('/content/drive/MyDrive/Диплом/Ship_detection/Input/I
data_target = np.load('/content/drive/MyDrive/Диплом/Ship_detection/Inpu
#Set target to one hot target for classification problem
from sklearn.preprocessing import OneHotEncoder
targets = data_target.reshape(len(data_target), -1)
enc = OneHotEncoder()
enc.fit(targets)
targets = enc.transform(targets).toarray()
print(targets.shape)
del data_target
targets

In [ ]: data

In [ ]: Train_path = '/content/drive/MyDrive/Диплом/Ship_detection/Input/train/'

In [ ]: %%time
index = 0
n = 100
Y = np.array([[0] * 2] * n)
Y_true = np.array([[0] * 2] * n)
l = data['ImageId'].values
anss = data['exist_ship'].values
offset = 500
for i in range(n):
    image_name = l[offset + i]
    imageA = Image.open(Train_path+image_name).resize((256, 256)) #open

```

```
display(imageA)
ans = int(input())
Y[i][ans] = 1
Y_true[i][anss[offset + i]] = 1
```

```
In [ ]: np.save('/content/drive/MyDrive/Диплом/Ship_detection/Input/human_ans',
```

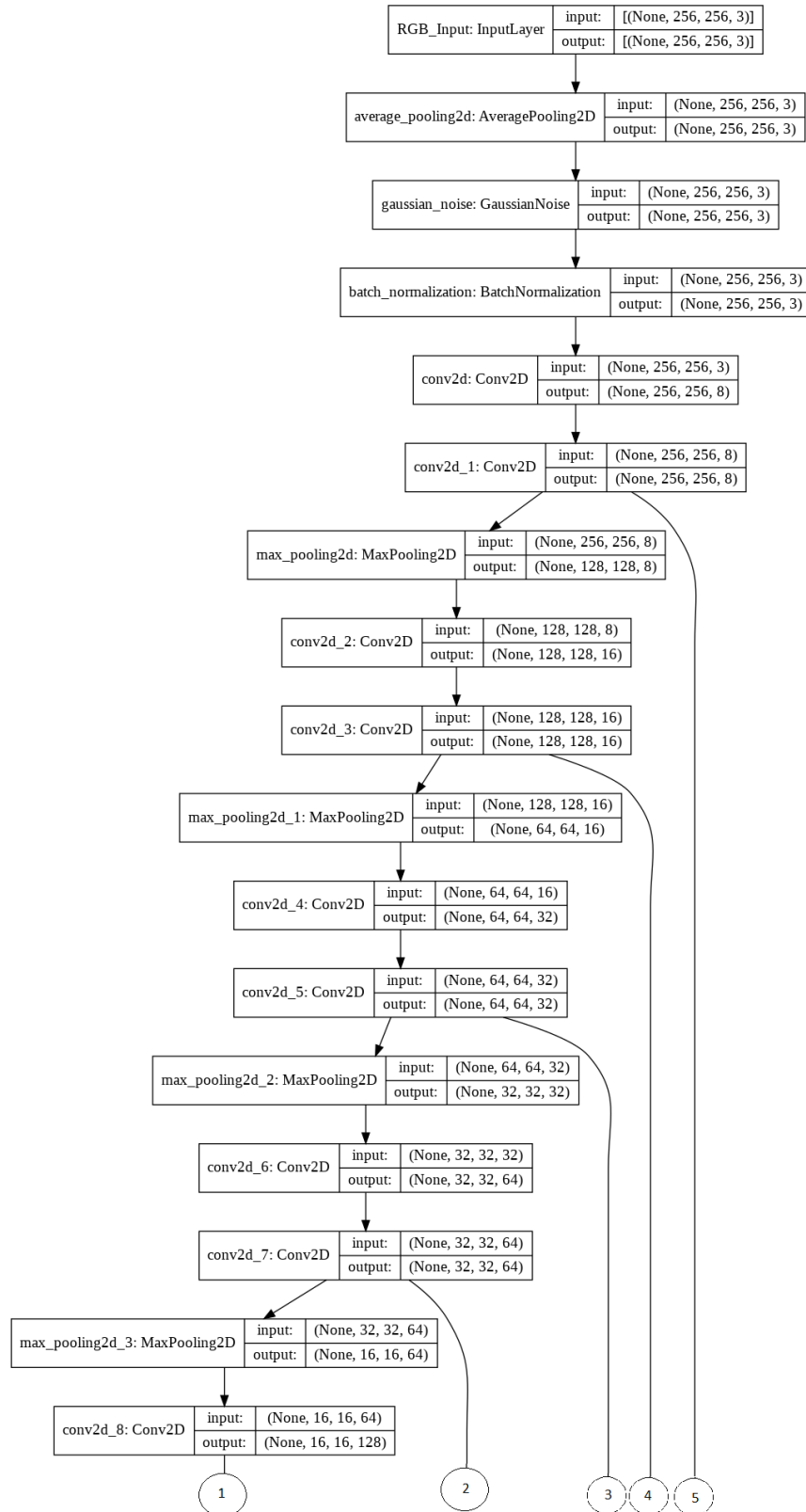
```
In [ ]: from sklearn.metrics import accuracy_score
print("Точность (accuracy) равно ", accuracy_score(Y_true, Y))
```

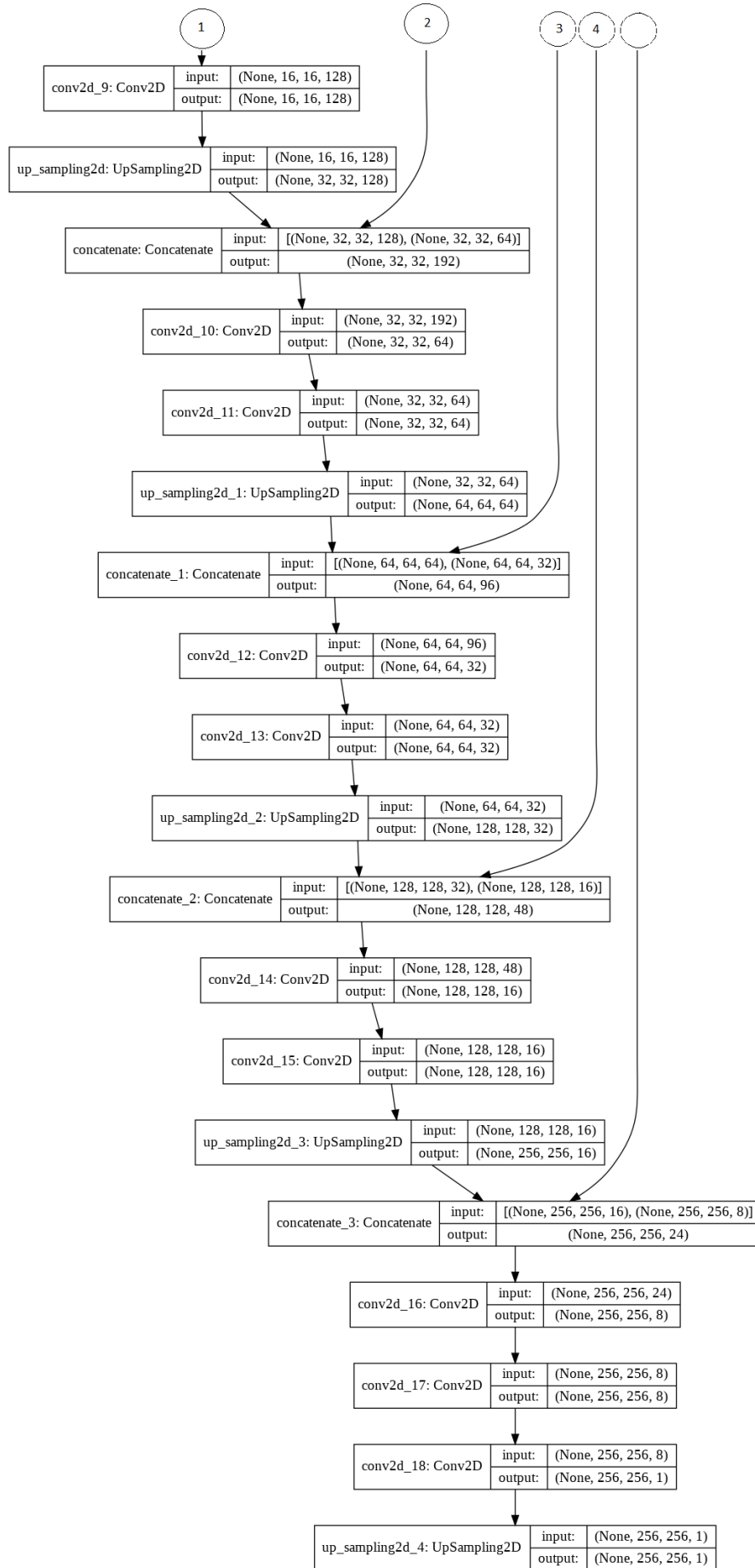
```
In [ ]: from sklearn.metrics import precision_score
print("Точность (precision) равно ",
      precision_score(Y_true.reshape(-1, 1), Y.reshape(-1, 1)))
```

```
In [ ]: from sklearn.metrics import recall_score
print("Полнота (recall) равно ",
      recall_score(Y_true.reshape(-1, 1), Y.reshape(-1, 1)))
```

Приложение 2

Изображение структуры сверточной нейронной сети для задачи семантической сегментации





Приложение 3

Программная реализация задачи семантической сегментации

Параметры модели

```
BATCH_SIZE = 64
EDGE_CROP = 16
GAUSSIAN_NOISE = 0.1
UPSAMPLE_MODE = "SIMPLE"

#Понижение дискретизации внутри сети
NET_SCALING = (1, 1)

#Понижение размерности в предобработке
IMG_SCALING = (3, 3)

#Число изображений для валидации
VALID_IMG_COUNT = 900

#Максимальное число шагов за эпоху при обучении
MAX_TRAIN_STEPS = 9
MAX_TRAIN_EPOCHS = 50
AUGMENT_BRIGHTNESS = False

SEED = 42
```

```
from skimage.util import montage
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from skimage.io import imread
import matplotlib.pyplot as plt
from matplotlib.cm import get_cmap
from skimage.segmentation import mark_boundaries
from sklearn.model_selection import train_test_split
import keras.backend as K

from keras.preprocessing.image import ImageDataGenerator
from keras import models, layers
import keras.backend as K
from keras.optimizers import Adam
from keras.losses import binary_crossentropy
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping
from tensorflow.keras.optimizers import RMSprop, Adam

from skimage.morphology import binary_opening, disk, label
import gc; gc.enable()

from PIL import Image

montage_rgb = lambda x: np.stack([montage(x[:, :, :, i]) for i in range(3)])
ship_dir = "/content/drive/MyDrive/Диплом/Ship_detection/Input/"
train_image_dir = os.path.join(ship_dir, 'train')
```

Определим вспомогательные процедуры для декодирования, кодирования и вывода изображения и маски корабля

```
In [ ]: def rle_encode(img, min_max_treshold = 1e-3, max_mean_treshold = None):
'''
img: numpy array, 1 - mask, 0 - background
Возвращает бегущую строку как форматированную
'''
if (np.max(img) < min_max_treshold):
    return ''
if (max_mean_treshold and np.mean(img) > max_mean_treshold):
    return ''
pixels = img.T.flatten()
pixels = np.concatenate([[0], pixels, [0]])
runs = np.where(pixels[1:] != pixels[:-1])[0] + 1
runs[1::2] -= runs[:-2]
return ' '.join(str(x) for x in runs)
```

```
In [ ]: def multi_rle_encode(img, **kwargs):
'''
Кодируем объединенные регионы как разделители масок
'''
labels = label(img)
if img.ndim > 2:
    return [rle_encode(np.sum(labels == k, axis = 2), **kwargs) for k
else:
    return [rle_encode(labels == k, **kwargs) for k in np.unique(lab
```

```
In [ ]: def rle_decode(mask_rle, shape = (768, 768)):
'''
mask_rle: бегущая - длина как форматированная строка (start length)
shape: (height,width) массив для возвратного значения
Возвращаем numpy array, 1 - mask, 0 - background
'''
s = mask_rle.split()
starts, lengths = [np.asarray(x, dtype = int) for x in (s[0:][::2],
starts -= 1
ends = starts + lengths
img = np.zeros(shape[0] * shape[1], dtype = np.uint8)
for lo, hi in zip(starts, ends):
    img[lo:hi] = 1
return img.reshape(shape).T
```

```
In [ ]: def masks_as_image(in_mask_list):
#Берем индивидуальную маску корабля и создаем отдельный массив масок
all_masks = np.zeros((768, 768), dtype = np.uint8)
for mask in in_mask_list:
    if isinstance(mask, str):
        all_masks |= rle_decode(mask)
return all_masks
```

```
In [ ]: def masks_as_color(in_mask_list):
#Берем индивидуальную маску корабля и создаем цветовую маску для каж
all_masks = np.zeros((768, 768), dtype = np.float)
scale = lambda x: (len(in_mask_list) + x + 1) / (len(in_mask_list) *
for i, mask in enumerate(in_mask_list):
    if isinstance(mask, str):
        all_masks[:, :] += scale(i) * rle_decode(mask)
return all_masks
```

Продемонстрируем работу

```
In [ ]: masks = pd.read_csv("/content/drive/MyDrive/Диплом/Ship_detection/Input/
```

```

masks = masks.drop(['Unnamed: 0', 'exist_ship'], axis=1)
masks.head()

```

```

In [ ]: not_empty = pd.notna(masks.EncodedPixels)
print(not_empty.sum(), "masks in", masks[not_empty].ImageId.nunique(), '
print((~not_empty).sum(), "empty images in", masks.ImageId.nunique(), "t

```

```

In [ ]: im = Image.open("/content/drive/MyDrive/Диплом/Ship_detection/Input/trai
im

```

```

In [ ]: fig, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5, figsize = (30, 5))
rle_0 = masks.query('ImageId == "8ce7d933f.jpg")["EncodedPixels"]
img_0 = masks_as_image(rle_0)
ax0.imshow(im)
ax0.set_title("Оригинальное изображение")

ax1.imshow(img_0)
ax1.set_title("Маска как изображение")

rle_1 = multi_rle_encode(img_0)
img_1 = masks_as_color(rle_0)
ax2.imshow(img_1)
ax2.set_title("Перекодированное")

img_c = masks_as_color(rle_0)
ax3.imshow(img_c)
ax3.set_title("Масква в цвете")

img_c = masks_as_color(rle_1)
ax4.imshow(img_c)
ax4.set_title("Перекодированное в цвета")
print("Проверка Декодирования -> Кодирование", 'RLE_0:', len(rle_0), '->
'RLE_1:', len(rle_1))
print(np.sum(img_0 - img_1), 'error')

```

Разделим данные на тренировочные и проверочные

```

In [ ]: #Поле, указывающее, есть ли корабль на картинке: 1 - есть, 0 - нет
masks['ships'] = masks['EncodedPixels'].map(lambda c_row: 1 if isinstance
unique_img_ids = masks.groupby("ImageId").agg({'ships': 'sum'}).reset_in
unique_img_ids['has_ship'] = unique_img_ids['ships'].map(lambda x: 1.0 i
unique_img_ids['has_ship_vec'] = unique_img_ids['has_ship'].map(lambda x
unique_img_ids['file_size_kb'] = unique_img_ids['ImageId'].map(lambda c_
unique_img_ids['file_size_kb'].hist()
masks.drop(['ships'], axis = 1, inplace = True)
unique_img_ids.sample(10)

```

Построим гистограмму от числа кораблей (копий изображения) для одного файла

```

In [ ]: unique_img_ids['ships'].hist(bins= unique_img_ids['ships'].max() + 1)

```

```

===== МЕСТО ДЛЯ ОТСЕИВАНИЯ ДУБЛИКАТОВ
=====

```

Сбалансируем выборку

```
In [ ]: train_ids, valid_ids = train_test_split(unique_img_ids, test_size = 0.25)
        train_df = pd.merge(masks, train_ids)
        valid_df = pd.merge(masks, valid_ids)

        print(train_df.shape[0], 'training masks')
        print(valid_df.shape[0], 'validation masks')
```

Декодируем данные в изображения

```
In [ ]: def make_image_gen(in_df, batch_size = BATCH_SIZE):
        all_batches = list(in_df.groupby('ImageId'))
        out_rgb = []
        out_mask = []
        while True:
            np.random.shuffle(all_batches)
            for c_img_id, c_masks in all_batches:
                rgb_path = os.path.join(train_image_dir, c_img_id)
                c_img = imread(rgb_path)
                c_mask = np.expand_dims(masks_as_image(c_masks['EncodedPixel']), 1)
                if IMG_SCALING is not None:
                    c_img = c_img[::IMG_SCALING[0], ::IMG_SCALING[1]]
                    c_mask = c_mask[::IMG_SCALING[0], ::IMG_SCALING[1]]
                out_rgb += [c_img]
                out_mask += [c_mask]
            if len(out_rgb) >= batch_size:
                yield np.stack(out_rgb, 0) / 255.0, np.stack(out_mask, 0)
                out_rgb, out_mask = [], []
```

```
In [ ]: """
x (2048, 256, 256, 3) 0.0 1.0
y (2048, 256, 256, 1) 0 1

train_gen = make_image_gen(train_df)
train_x, train_y = next(train_gen)
print('x', train_x.shape, train_x.min(), train_x.max())
print('y', train_y.shape, train_y.min(), train_y.max())
"""
```

```
In [ ]: '''
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (30, 10))
batch_rgb = montage_rgb(train_x)
batch_seg = montage(train_y[:, :, :, 0])
ax1.imshow(batch_rgb)
ax1.set_title('Images')
ax2.imshow(batch_seg)
ax2.set_title('Segmentations')
ax3.imshow(mark_boundaries(batch_rgb,
                           batch_seg.astype(int)))
ax3.set_title('Outlined Ships')
fig.savefig('overview.png')
'''
```

Сделаем набор для проверки

```
In [ ]: %%time
        valid_x, valid_y = next(make_image_gen(valid_df, VALID_IMG_COUNT))
        print(valid_x.shape, valid_y.shape)
```

```
In [ ]: valid_x[0]
```

```
In [ ]: s = 10
j = 0
for r in valid_y[-17]:
    k = 10
    i = 0
    for c in r:
        if(i > k):
            print("...")
            break;
        print(c, sep=' ', end='', flush=True)
        i += 1
    print
    j += 1
    if(j > k):
        print("...")
        break;
```

Дополним данные

```
In [ ]: dg_args = dict(featurewise_center = False,
                        samplewise_center = False,
                        rotation_range = 45,
                        width_shift_range = 0.1,
                        height_shift_range = 0.1,
                        shear_range = 0.01,
                        zoom_range = [0.9, 1.25],
                        horizontal_flip = True,
                        vertical_flip = True,
                        fill_mode = 'reflect',
                        data_format = 'channels_last')
# brightness can be problematic since it seems to change the labels diff
if AUGMENT_BRIGHTNESS:
    dg_args['brightness_range'] = [0.5, 1.5]
image_gen = ImageDataGenerator(**dg_args)

if AUGMENT_BRIGHTNESS:
    dg_args.pop('brightness_range')
label_gen = ImageDataGenerator(**dg_args)

def create_aug_gen(in_gen, seed = None):
    np.random.seed(seed if seed is not None else np.random.choice(range(
    for in_x, in_y in in_gen:
        seed = np.random.choice(range(9999))
        # keep the seeds synchronized otherwise the augmentation to the i
        g_x = image_gen.flow(255*in_x,
                             batch_size = in_x.shape[0],
                             seed = seed,
                             shuffle=True)
        g_y = label_gen.flow(in_y,
                             batch_size = in_x.shape[0],
                             seed = seed,
                             shuffle=True)

        yield next(g_x)/255.0, next(g_y)
```

```
In [ ]: """
x (64, 256, 256, 3) float32 0.0 1.0
y (64, 256, 256, 1) float32 0.0 1.0
cur_gen = create_aug_gen(train_gen)
```

```

t_x, t_y = next(cur_gen)
print('x', t_x.shape, t_x.dtype, t_x.min(), t_x.max())
print('y', t_y.shape, t_y.dtype, t_y.min(), t_y.max())
# only keep first 9 samples to examine in detail
t_x = t_x[:9]
t_y = t_y[:9]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (20, 10))
ax1.imshow(montage_rgb(t_x), cmap='gray')
ax1.set_title('images')
ax2.imshow(montage(t_y[:, :, :, 0]), cmap='gray_r')
ax2.set_title('ships')
"""

```

```
In [ ]: gc.collect()
```

Соберем модель

```

In [ ]: # Build U-Net model
def upsample_conv(filters, kernel_size, strides, padding):
    return layers.Conv2DTranspose(filters, kernel_size, strides=strides,
def upsample_simple(filters, kernel_size, strides, padding):
    return layers.UpSampling2D(strides)

if UPSAMPLE_MODE=='DECONV':
    upsample=upsample_conv
else:
    upsample=upsample_simple

input_img = layers.Input([256, 256, 3], name = 'RGB_Input')
pp_in_layer = input_img

if NET_SCALING is not None:
    pp_in_layer = layers.AvgPool2D(NET_SCALING)(pp_in_layer)

pp_in_layer = layers.GaussianNoise(GAUSSIAN_NOISE)(pp_in_layer)
pp_in_layer = layers.BatchNormalization()(pp_in_layer)

c1 = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(pp_in_
c1 = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(c1)
p1 = layers.MaxPooling2D((2, 2))(c1)

c2 = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(p1)
c2 = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(c2)
p2 = layers.MaxPooling2D((2, 2))(c2)

c3 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(p2)
c3 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(c3)
p3 = layers.MaxPooling2D((2, 2))(c3)

c4 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(p3)
c4 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c4)
p4 = layers.MaxPooling2D(pool_size=(2, 2))(c4)

c5 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p4)
c5 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c5)

u6 = upsample(64, (2, 2), strides=(2, 2), padding='same')(c5)
u6 = layers.concatenate([u6, c4])
c6 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u6)
c6 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c6)

```

```

u7 = upsample(32, (2, 2), strides=(2, 2), padding='same') (c6)
u7 = layers.concatenate([u7, c3])
c7 = layers.Conv2D(32, (3, 3), activation='relu', padding='same') (u7)
c7 = layers.Conv2D(32, (3, 3), activation='relu', padding='same') (c7)

u8 = upsample(16, (2, 2), strides=(2, 2), padding='same') (c7)
u8 = layers.concatenate([u8, c2])
c8 = layers.Conv2D(16, (3, 3), activation='relu', padding='same') (u8)
c8 = layers.Conv2D(16, (3, 3), activation='relu', padding='same') (c8)

u9 = upsample(8, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = layers.concatenate([u9, c1], axis=3)
c9 = layers.Conv2D(8, (3, 3), activation='relu', padding='same') (u9)
c9 = layers.Conv2D(8, (3, 3), activation='relu', padding='same') (c9)

d = layers.Conv2D(1, (1, 1), activation='sigmoid') (c9)
# d = layers.Cropping2D((EDGE_CROP, EDGE_CROP)) (d)
# d = layers.ZeroPadding2D((EDGE_CROP, EDGE_CROP)) (d)
if NET_SCALING is not None:
    d = layers.UpSampling2D(NET_SCALING) (d)

seg_model = models.Model(inputs=[input_img], outputs=[d])
seg_model.summary()

```

```

In [ ]: image_file = 'model_1.png'
        tf.keras.utils.plot_model(seg_model, to_file = image_file, show_shapes =

```

```

In [ ]: #https://lars76.github.io/2018/09/27/loss-functions-for-segmentation.htm
def dice_coef(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.math.sigmoid(y_pred)
    numerator = 2 * tf.reduce_sum(y_true * y_pred)
    denominator = tf.reduce_sum(y_true + y_pred)
    return numerator / denominator

def dice_loss(y_true, y_pred):
    return 1 - dice_coef(y_true, y_pred)

def true_positive_rate(y_true, y_pred):
    return K.sum(K.flatten(y_true)*K.flatten(K.round(y_pred)))/K.sum(y_t

#Cross entropy + DICE loss
def comb_loss(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    o = tf.nn.sigmoid_cross_entropy_with_logits(y_true, y_pred) + dice_l
    return tf.reduce_mean(o)

def balanced_cross_entropy(y_true, y_pred):
    weight_a = beta * tf.cast(y_true, tf.float32)
    weight_b = (1 - beta) * tf.cast(1 - y_true, tf.float32)

    o = (tf.math.log1p(tf.exp(-tf.abs(y_pred))) + tf.nn.relu(-y_pred)) *
    return tf.reduce_mean(o)

beta = 0.7
def tversky_loss(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.math.sigmoid(y_pred)
    numerator = y_true * y_pred
    denominator = y_true * y_pred + beta * (1 - y_true) * y_pred + (1 -

```



```
return 1 - tf.reduce_sum(numerator) / tf.reduce_sum(denominator)
```

```
In [ ]: weight_path="/content/drive/MyDrive/Диплом/Ship_detection/weights/u_net/

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,

reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.33,
                                   patience=1, verbose=1, mode='min',
                                   min_delta=0.0001, cooldown=0, min_lr=

early = EarlyStopping(monitor="val_loss", mode="min", verbose=2,
                      patience=20) # probably needs to be more patient,

callbacks_list = [checkpoint, early, reduceLROnPlat]
```

```
In [ ]: gc.collect()
```

```
In [ ]: RMS = RMSprop( learning_rate=0.001,
                       rho=0.9,
                       momentum=0.0,
                       epsilon=1e-07,
                       centered=False,
                       name="RMSprop")

adam = Adam(learning_rate=0.001,
            beta_1=0.9,
            beta_2=0.999,
            epsilon=1e-07,
            amsgrad=False,
            name="Adam")

seg_model.compile(optimizer = adam, loss= tversky_loss, metrics=['binary
#weight_path1="/content/drive/MyDrive/Диплом/Ship_detection/weights/u_ne
#seg_model.load_weights(weight_path1)
```

```
In [ ]: def fit(seg_model):
    step_count = MAX_TRAIN_STEPS
    #step_count = train_df.shape[0]//BATCH_SIZE
    aug_gen = create_aug_gen(make_image_gen(train_df))
    loss_history = [seg_model.fit(aug_gen,
                                  steps_per_epoch=step_count,
                                  # batch_size = BATCH_SIZE,
                                  epochs=MAX_TRAIN_EPOCHS,
                                  validation_data=(valid_x, valid_y),
                                  callbacks=callbacks_list
                                  )]

    return loss_history

loss_history = fit(seg_model)
```

```
In [ ]: def show_loss(loss_history):
    epochs = np.concatenate([mh.epoch for mh in loss_history])
    fig, (ax1, ax2, ax3, ax, ax5) = plt.subplots(1, 5, figsize=(22, 10))

    _ = ax1.plot(epochs, np.concatenate([mh.history['loss'] for mh in lo
    epochs, np.concatenate([mh.history['val_loss'] for mh i
    ax1.legend(['Training', 'Validation'])
    ax1.set_title('Loss')

    _ = ax2.plot(epochs, np.concatenate([mh.history['binary_accuracy'] f
    epochs, np.concatenate([mh.history['val_binary_accuracy
    ax2.legend(['Training', 'Validation'])
```

```

ax2.set_title('Binary Accuracy (%)')

_ = ax3.plot(epochs, np.concatenate([mh.history['dice_coef'] for mh
                                     epochs, np.concatenate([mh.history['val_dice_coef'] for
ax3.legend(['Training', 'Validation'])
ax3.set_title('DICE Coefficient (%)')

_ = ax4.plot(epochs, np.concatenate([mh.history['true_positive_rate']
                                     epochs, np.concatenate([mh.history['val_true_positive_r
ax4.legend(['Training', 'Validation'])
ax4.set_title('TFP')

_ = ax5.plot(epochs, np.concatenate([mh.history['false_positives']
                                     epochs, np.concatenate([mh.history['val_false_positives
ax5.legend(['Training', 'Validation'])
ax5.set_title('FPR')

fig.savefig('/content/drive/MyDrive/Диплом/Ship_detection/RMS_TSKY_B

show_loss(loss_history)

```

```
In [ ]: gc.collect()
```

```
In [ ]: seg_model.load_weights(weight_path)
seg_model.save('/content/drive/MyDrive/Диплом/Ship_detection/weights/u_n
```

```
In [ ]: pred_y = seg_model.predict(valid_x)
print(pred_y.shape, pred_y.min(axis=0).max(), pred_y.max(axis=0).min(),
```

```
In [ ]: fig, ax = plt.subplots(1, 1, figsize = (6, 6))
ax.hist(pred_y.ravel(), np.linspace(0, 1, 20))
ax.set_xlim(0, 1)
ax.set_yscale('log', nonposy='clip')
```

Подготовка для полноразмерной модели

```
In [ ]: if IMG_SCALING is not None:
    fullres_model = models.Sequential()
    fullres_model.add(layers.AvgPool2D(IMG_SCALING, input_shape = (None,
    fullres_model.add(seg_model)
    fullres_model.add(layers.UpSampling2D(IMG_SCALING))
else:
    fullres_model = seg_model
fullres_model.save('/content/drive/MyDrive/Диплом/Ship_detection/weights
```

```
In [ ]: gc.collect()
```

Визуализируем предсказание

```
In [ ]: def raw_prediction(img, path=train_image_dir):
    c_img = imread(os.path.join(path, c_img_name))
    c_img = np.expand_dims(c_img, 0)/255.0
    cur_seg = fullres_model.predict(c_img)[0]
    return cur_seg, c_img[0]

def smooth(cur_seg):
    return binary_opening(cur_seg>0.99, np.expand_dims(disk(2), -1))
```

```

def predict(img, path=train_image_dir):
    cur_seg, c_img = raw_prediction(img, path=path)
    return smooth(cur_seg), c_img

## Get a sample of each group of ship count
n_samples = 100
samples = valid_df.groupby('ships').apply(lambda x: x.sample(random_state=42))
fig, m_axs = plt.subplots(samples.shape[0], 4, figsize=(15, samples.shape[0]*4))
for c_ax, ax in zip(m_axs, m_axs.flatten()):

    for (ax1, ax2, ax3, ax4), c_img_name in zip(m_axs, samples.ImageId.value_counts().items()):
        first_seg, first_img = raw_prediction(c_img_name, train_image_dir)
        ax1.imshow(first_img)
        ax1.set_title('Image: ' + c_img_name)
        ax2.imshow(first_seg[:, :, 0], cmap=get_cmap('jet'))
        ax2.set_title('Model Prediction')
        reencoded = masks_as_color(multi_rle_encode(smooth(first_seg)[:, :, 0]))
        ax3.imshow(reencoded)
        ax3.set_title('Prediction Masks')
        ground_truth = masks_as_color(masks.query('ImageId=="{}"'.format(c_img_name)))
        ax4.imshow(ground_truth)
        ax4.set_title('Ground Truth')

fig.savefig('/content/drive/MyDrive/Диплом/Ship_detection/RMS_50_TSKY_BE')
plt.show()

```

In []: