«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Центр непрерывного образования

Факультета компьютерных наук

АТТЕСТАЦИОННАЯ РАБОТА

"ОБНАРУЖЕНИЕ ДЕФЕКТОВ СТАЛИ: СЕГМЕНТИРОВАНИЕ ДЕФЕКТОВ СТАЛЬНОГО ЛИСТА"

Выполнил:

Хмарская Ольга Владимировна

Научный руководитель:

Мурашкин Вячеслав Владимирович

Москва 2019

ОГЛАВЛЕНИЕ:

ВВЕДЕНИЕ

Сталь является одним из важнейших строительных материалов современности. Стальные здания и сооружения устойчивы к естественному и искусственному износу, что сделало материал используемым во всем мире.

Применение методов машинного обучения для детектирования дефектов листа направлено позволит повысить автоматизированность и эффективность стального производства.

Летом 2019 года “Северсталь”[1] объявила соревнование на сегментирование дефектов стального листа.[2] «Северсталь» является мировым лидером горнодобывающей и металлургической отрасли, традиционно занимающим первое место в объемах изготовления стального листа. В

Виды представленных дефектов:

- *расслоение* - дефект поверхности в виде трещин на кромках и торцах листов и других видов проката, образовавшихся при наличии в металле усадочных дефектов, внутренних разрывов, повышенной загрязненности неметаллическими включениями и при пережоге;

- *закат* - дефект поверхности, представляющий собой прикатанный продольный выступ, образовавшийся в результате закатывания уса, подреза, грубых следов зачистки и глубоких рисок.

- *прокатная плена* - дефект поверхности, представляющий собой отслоение металла языкообразной формы, соединенное с основным металлом одной стороной, образовавшееся вследствие раскатки или расковки рванин, подрезов, следов глубокой зачистки дефектов или сильной выработки валков, а также грубых механических повреждений

[1] https://www.severstal.com/rus/about/
[2] https://www.kaggle.com/c/severstal-steel-defect-detection

- *деформационная рванина* - дефект поверхности в виде раскрытого разрыва, расположенного поперек или под углом к направлению наибольшей вытяжки металла при прокатке или ковке, образовавшийся вследствие пониженной пластичности металла

- *трещина напряжения* - дефект поверхности, представляющий собой разрыв металла, идущий вглубь под прямым углом к поверхности, образовавшийся вследствие напряжений, связанных со структурными превращениями или неравномерным нагревом и охлаждением.

В ходе соревнования участникам предложено сегментировать изображения стального листа, с целью обнаружения дефектов разных типов для последующего применения результатов в автоматизированных системах контроля качества.

ОБЗОР ЛИТЕРАТУРЫ

В данном исследовании решается задача сегментации, когда каждому пикселю изображения присваивается класс, соответствующий классу дефекта или фону.

При решение задач сегментирования будут использоваться сети энкодер-декодеры, возвращающие на выход массивы равные размеру входного массива или меньше исходного.

Фотографии - изображения, имеющие свою собственную внутреннюю структуру:

1) исходные данные представляют собой многомерный массив
2) среди размерностей этого массива есть ось, порядок вдоль которой играет важную роль: расположение пикселов.
3) другие оси - каналы, описывающие свойства каждого элемента по предыдущему подмножеству осей: три компонента для изображений

В решении поставленной задачи будем применять сверточные нейронные сети, наилучшим образом зарекомендовавших себя для работы с изображениями.

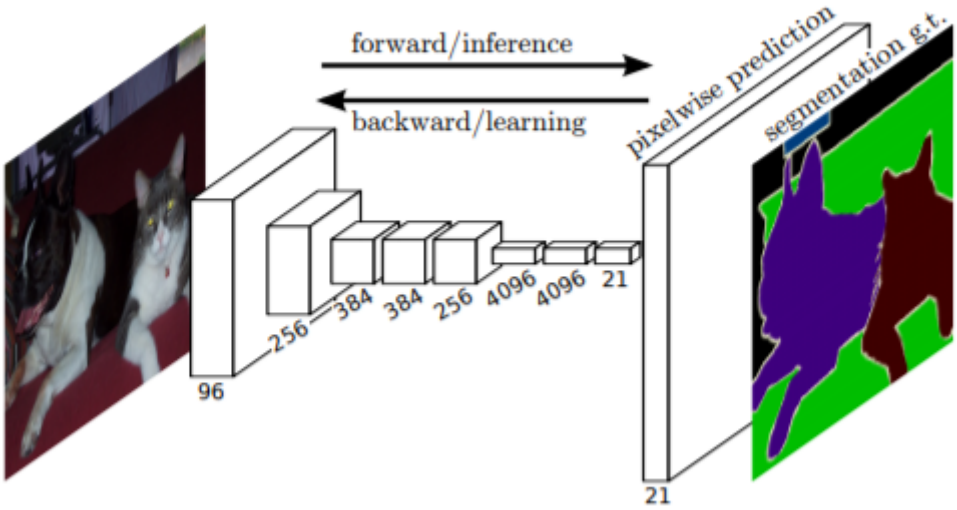
Применение сверточных нейронных сетей к анализу изображений обусловлено:

1. наилучшие показатели качества при задачах распознавания и классификации
2. меньшее количество весов по сравнению с полносвязной нейронной сетью (ядро свертки применяется по всему изображению, вместо того, чтобы определять каждому пикселю свои персональные весовые коэффициенты.
3. удобность распараллеливания и возможность использования GPU
4. относительная устойчивость к повороту и сдвигу

Вместе с тем, применение сверточных нейроннных сетей сопряжено с большим количеством настраиваемых параметров (количество слоёв, размерность ядра свёртки для каждого из слоёв, количество ядер для каждого из слоёв, шаг сдвига ядра при обработке слоя, необходимость слоёв субдискретизации, степень уменьшения ими размерности, функция по уменьшению размерности (выбор максимума, среднего и т. п.) и т.д.). Все эти параметры существенно влияют на результат, но выбираются исследователями эмпирически.

Впервые попытка применения сверточных полносвязных сетей (Fully Convolutional Networks) к задачам семантической сегментации была предложена в 2016 году. [3] Предложенный тип архитектуры превзошел качество на PASCAL VOC на 30%, которое в итоге составило 67,3%.

Image: FCN



[3] https://arxiv.org/pdf/1605.06211.pdf Fully Convolutional Networks for Semantic Segmentation Evan Shelhamer∗ , Jonathan Long∗ , and Trevor Darrell, Member, IEEE 20.5.16

До появления FCN предшественники использовали сложную процедуру предобработки и оптимизации, от чего была избавлена новая архитектура.[4]

Сеть вычисляет нелинейный фильтр. FCN естественно работает на входе любого размера, и производит вывод соответствующего (возможно пересчитанного).

Затем для работы с сегментацией/классификацией начинают применять более сложные и глубокие модели: U-net, ResNet, encoder-decoder модели.

На сегодняшний момент существует ряд архитектур сверточных нейронных сетей, успешно применяемых для решения задач сегментации. Однако, следует помнить, что каждая конкретная задачи может требовать дополнительной донастройки сети.

В рассматриваемом проекте мы будем решать задачу сегментации изображения - то есть присвоению класса каждому пикселю изображения.

В ходе исследования будут рассмотрены три архитектуры сверточных нейронных сетей.

U-net:[5]

В качестве базовой модели для выполнения проекта будем использовать сверточную архитектуру U-net. Данная архитектура считается одной из основных при решении задач, когда нужно не только определить класс изображения целиком, но и сегментировать части этого изображения, то есть создать маску/маски, которая будет делить изображение на несколько классов. Сеть обучается сквозным способом на небольшом количестве изображений и способная достигать хороших результатов на сравнительно небольших объемах базы фотографий.

---

[4] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano, “Toward automatic phenotyping of developing embryos from videos,” Image Processing, IEEE Transactions on, vol. 14, no. 9, pp. 1360–1371, 2005.

[5] https://arxiv.org/pdf/1505.04597.pdf U-Net: Convolutional Networks for Biomedical Image Segmentation Olaf Ronneberger, Philipp Fischer, and Thomas Brox 18.5.15

Основные характеристики сети можно обозначить следующим образом:

- конкатенирует активации энкодера с расширенными (upsampling) признаками декодера.
- быстрая обучаемость
- простота реализации

Архитектура U-net состоит из сужающегося пути (типичной архитектуры сверточной нейронной сети) и расширяющегося пути.

Сужающий путь состоит из повторяющихся сверток с размером ядра 3*3 с нелинейной активацией и операцией максимального объединения для понижения разрешения.

На каждом этапе понижающей дискретизации каналы свойств удваиваются. Каждый шаг в расширяющемся пути состоит из операции повышающей дискретизации карты свойств, за которой следуют:

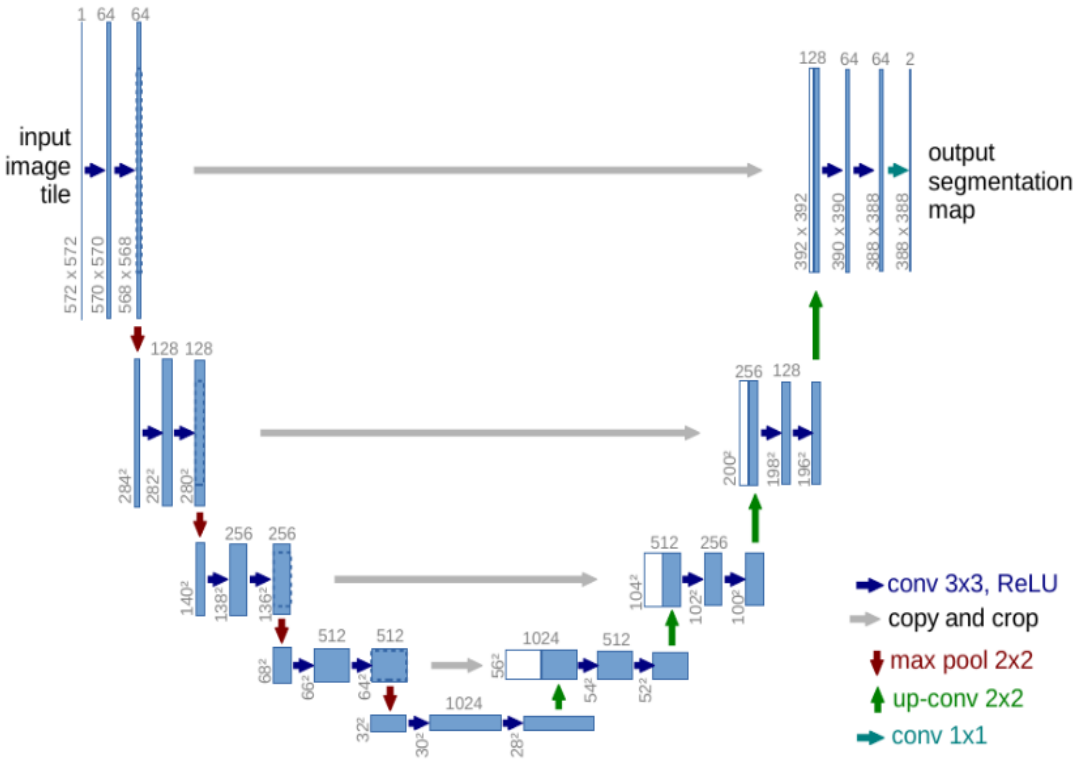
- свертка 2×2, которая уменьшает количество каналов свойств;
- объединение с соответствующим образом обрезанной картой свойств из стягивающегося пути;
- две 3×3 свертки, за которыми следует ReLU.

На последнем слое используется свертка 1×1 для сопоставления каждого 64-компонентного вектора свойств с желаемым количеством классов. Всего сеть содержит 23 сверточных слоя.

Сеть обучается методом стохастического градиентного спуска на основе входных изображений и соответствующих им карт сегментации. Применяемая попиксельно, функция soft-max вычисляет энергию по окончательной карте свойств вместе с функцией кросс-энтропии.

Архитектура сети:

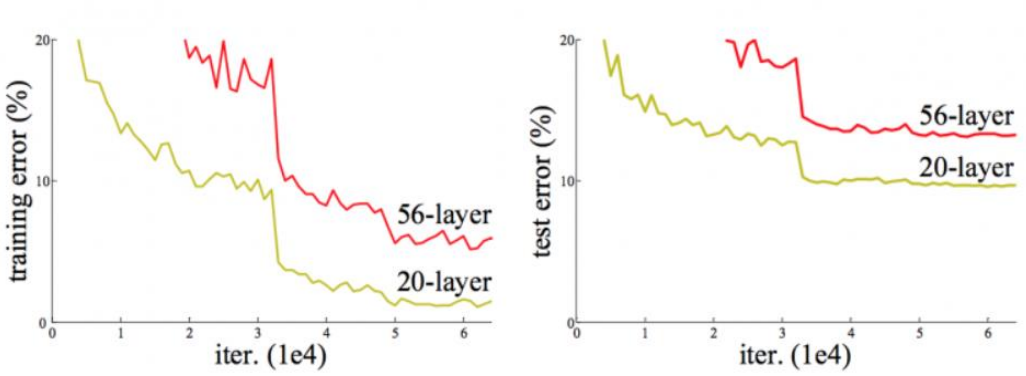
Image: Unet



2) **U-Net with ResNet encoder**Вторая и третья модели исследования связаны с еще одной сверточной нейронной сетью, получившей название ResNet[6] (остаточная нейронная сеть).

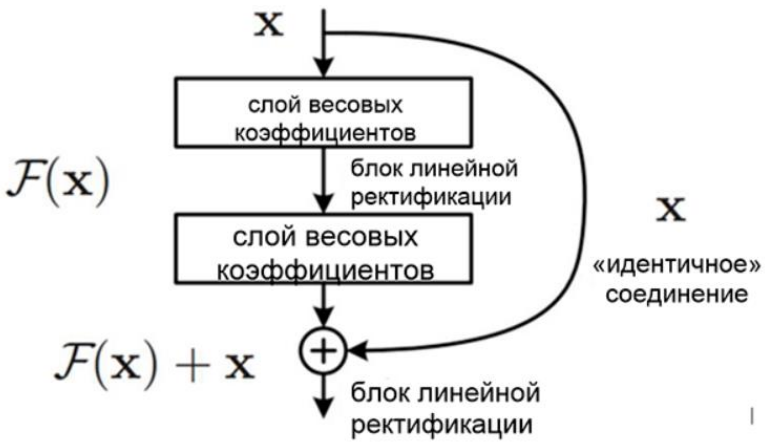
Многолетние исследования соревнования ImageNet показали, что увеличение глубины нейронной сети способно повышать качество модели, но лишь до определенного момента.

[6] https://arxiv.org/pdf/1512.03385.pdf Deep Residual Learning for Image Recognition Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun Microsoft Research



С увеличением глубины качество сначала увеличивается, а затем начинает ухудшаться (проблема затухаемого градиента). Кроме того, не все сверточные сети легко оптимизируемы.

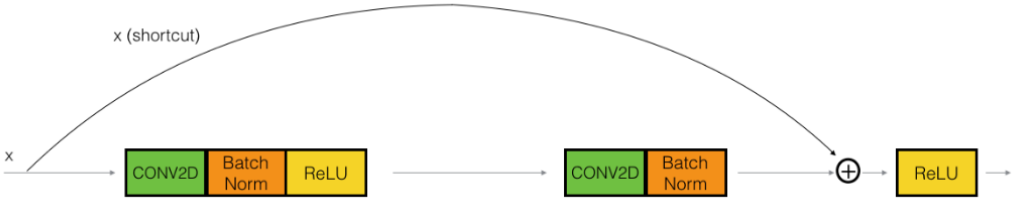
Для решения проблемы затухающего градиента была предложена идея — позволить сети изучать остаточное отображение (элемент, который следует добавить ко входным данным) вместо отображения как такового. Технически это выполняется с помощью обходного соединения:



“Идентичное» соединение не приводит к созданию новых весов, а значит сеть не усложняется. Данная идея показала очень хорошие результаты и позволила более качественно обучать очень глубокие сети.

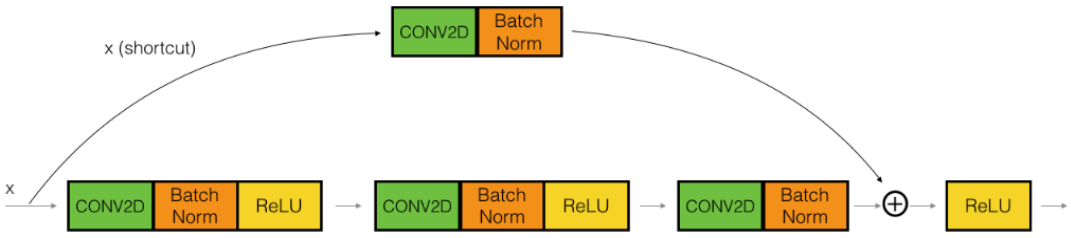
Остаточная сверточная нейронная сеть состоит из двух основных типов блоков.

1. The Identity_block - стандартная часть ResNet архитектуры. Применяется, когда размерность входящей активации равна размерности выходящей. Верхний путь "shortcut" - кратчайший. Нижний путь "main path" - длинный[7].



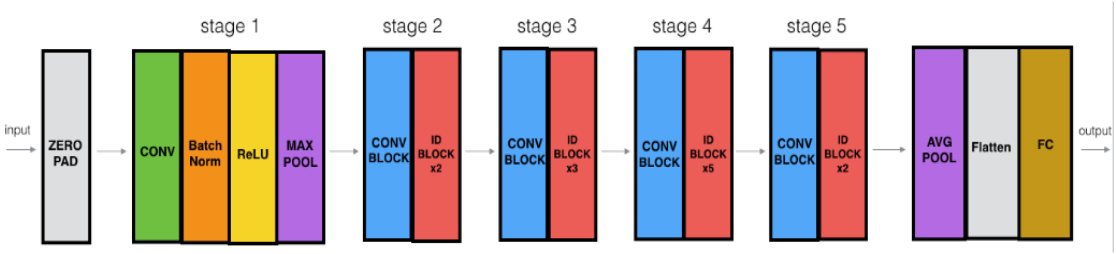
2 The convolutional block.

Сверточный блок также является неотъемлемой частью ResNet архитектуры и применяется для тех случаях, когда размерность входа и выхода не совпадают. Главное отличие двух блоков модели состоит в том, что shortcut путь не просто добавляется к выходу main path, но проходит сам через сверточный слой. Свертка применяется, чтобы понизить размерность shortcut. Её основная роль заключается в том, чтобы просто применить (выученную) линейную функцию, которая уменьшает размерность входного сигнала, так что размеры совпадают для последующего шага сложения.



Указанные блоки участвуют в формировании сети:

[7] https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Networks_yourself.ipynb



К очевидным достоинствам сети относятся:

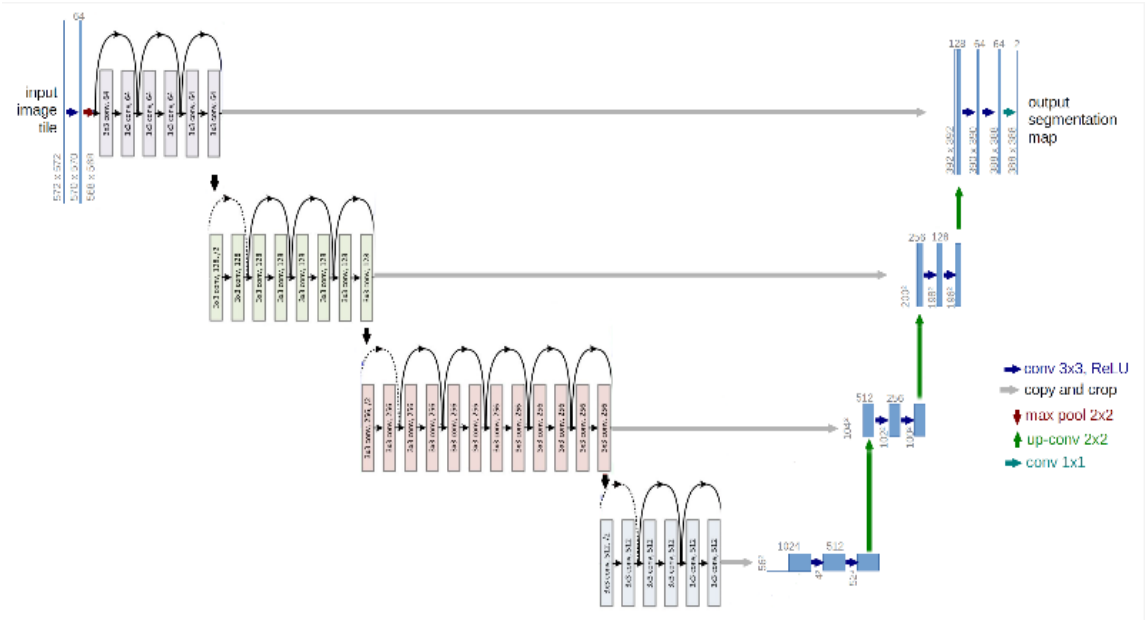
- ResNet относительно легко оптимизировать: «простые» сети (которые просто складывают слои) показывают большую ошибку обучения, когда глубина увеличивается.
- ResNet позволяет относительно легко увеличить точность благодаря увеличению глубины, чего с другими сетями добиться сложнее.

Вторая модель исследования предполагает добавление ResNet энкодера к U-net.

Четыре остаточных блока Resnet прокидывают свой выход не только вперед к следующему, но и на соответствующие слои декодера U-net.

Таким образом есть основания ожидать усиления сигнала на каждом из блоков сети, за счет добавления идентичного соединения.

Ниже показана схема такой сети:



3**) ResNet encoder-decoder**[8]

Как было сказано ранее, остаточные нейронные сети способны эффективно проводить обучения даже очень глубоких архитектур за счет борьбы с проблемой затухания градиента. Таким образом, есть основания ожидать, что глубокие слои будут способны выучивать более специфические паттерны изображений.

Применительно к объекту исследования следует помнить, что ResNet является сетью энкодером, неспособным качественно решать сложные задачи сегментирования. Поэтому в третьей модели классическую архитектуру ResNet50 мы дополним модель слоями соответствующей деконволюции для получения выходного слоя нужного размера[9].

[8] https://arxiv.org/abs/1806.01054?context=cs.CV Jindong Jiang, Lunan Zheng, Fei Luo, Zhijun Zhang RedNet: Residual Encoder-Decoder Network for indoor RGB-D Semantic Segmentation

[9] teleported.in/posts/decoding-resnet-architecture/

МЕТОДЫ

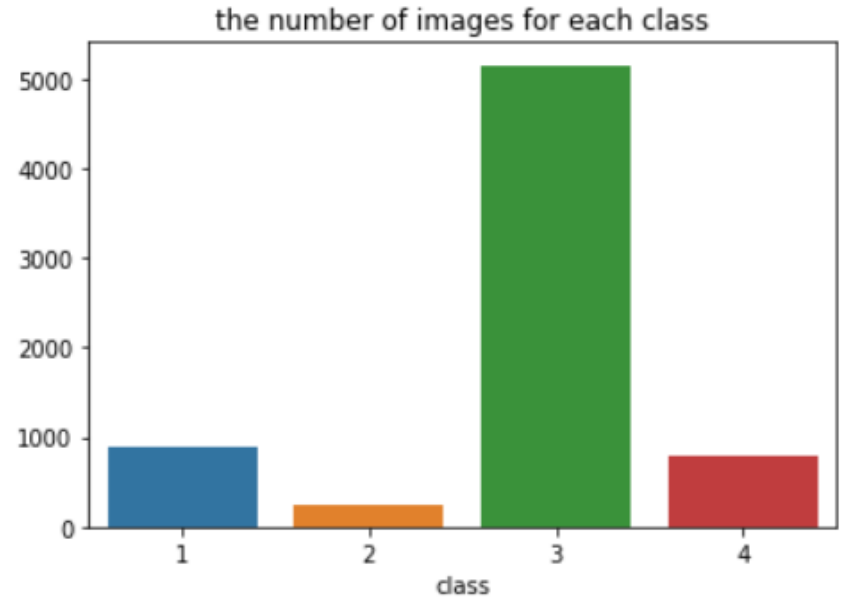
В данной работе для решения задачи определения дефектов (сегментации) мы будем использовать три основные архитектуры:

- U-net
- U-net with ResNet encoder
- ResNet with decoder

Входные данные

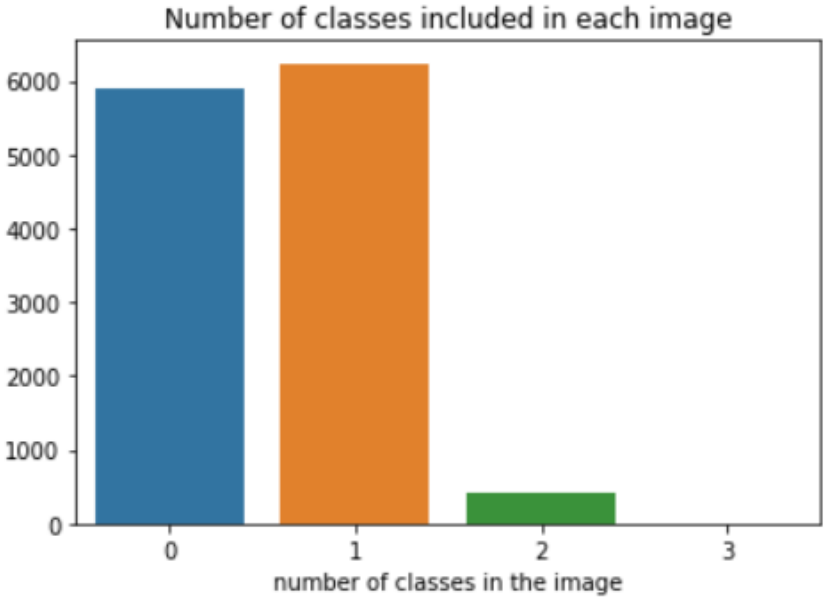
Собранный датасет содержит информацию о классе дефекта листа по 12 568 образцам. Фотографии размером 1600*256, формат RGB. Представленные изображения содержат информацию о 5150 образцах с дефектом №3, 897 - с дефектом №1, 801 - с дефектом №4 и 247 - с дефектом № 2:

```
defaultdict(int, {1: 897, 3: 5150, 4: 801, 2: 247})
```



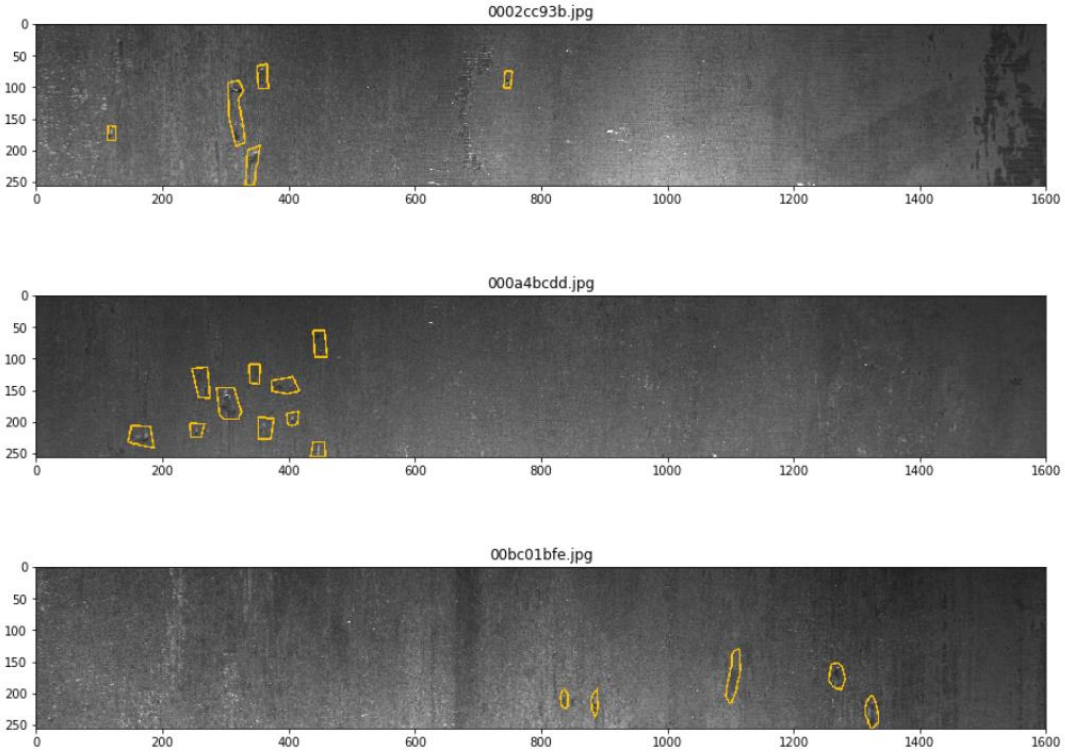
Имеются 427 изображений с несколькими видами дефектов:

[9]:
```
defaultdict(int, {1: 6239, 0: 5902, 2: 425, 3: 2})
```

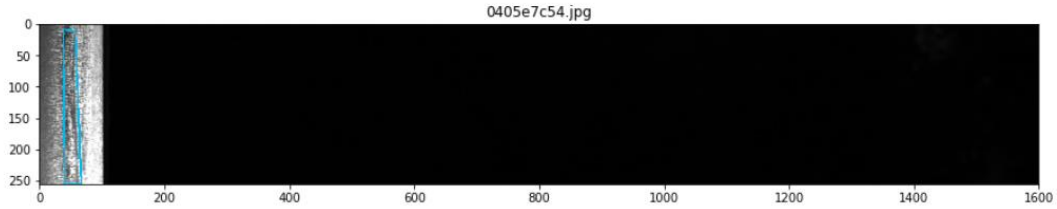


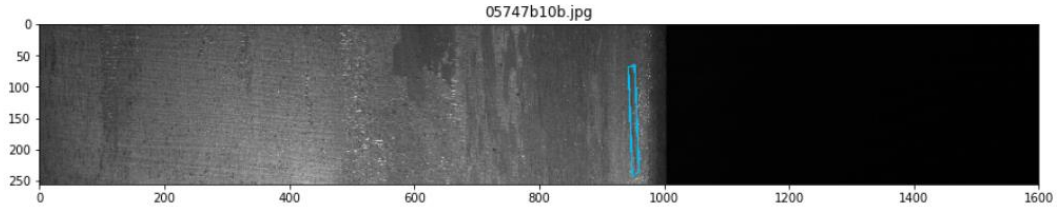
Количество фотографий, не имеющих искомых дефектов 6000 образцов.

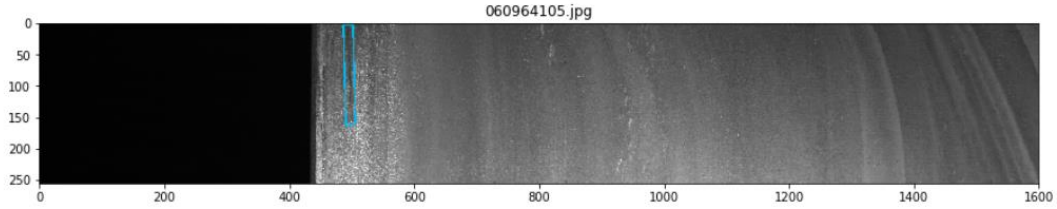
Дефект №1: Прокатная плена



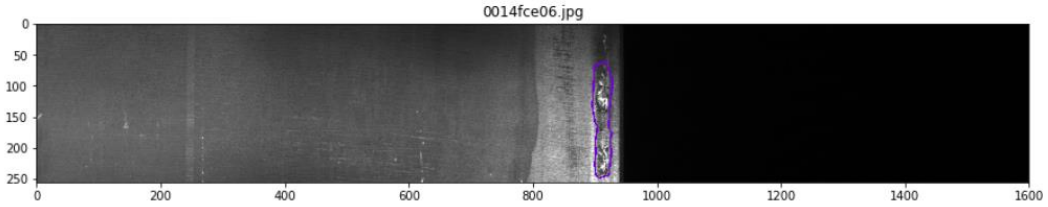
Дефект №2 (расслоение):

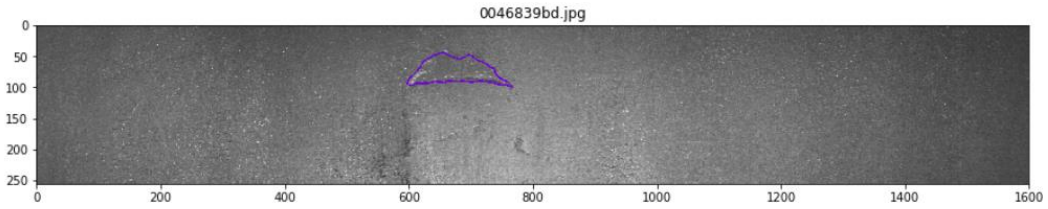


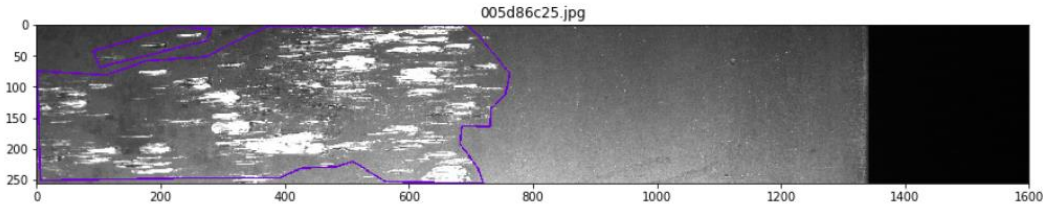




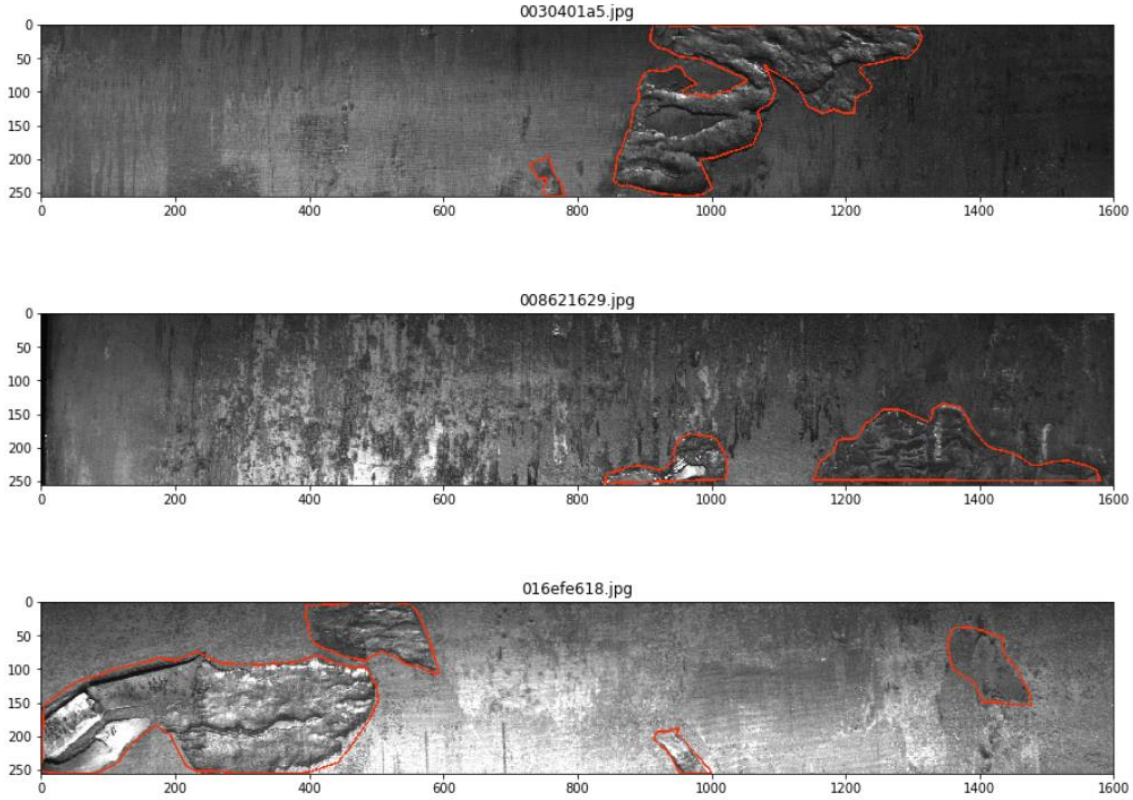
Дефект №3 (деформационная рванина):



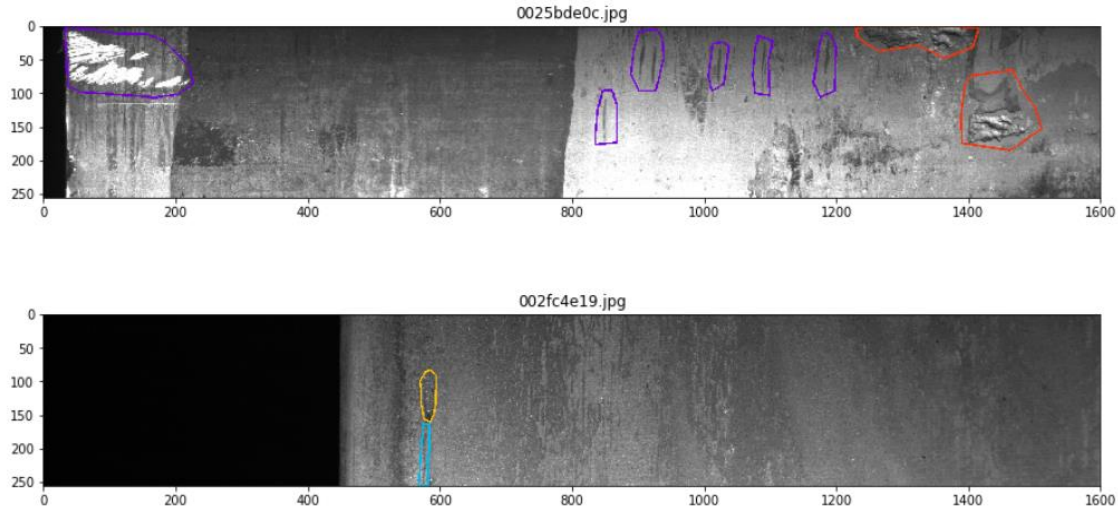


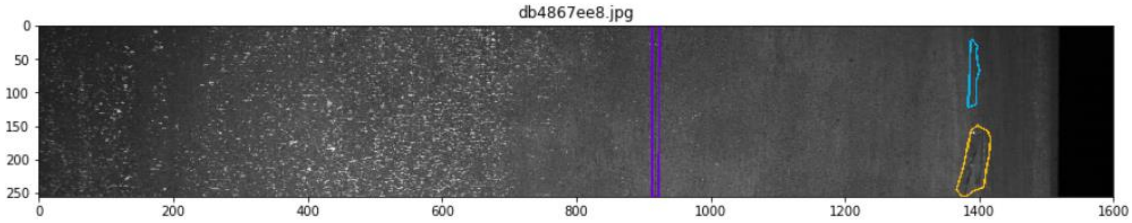


Дефект №4 (закат):



Также представлен ряд изображений, содержащих более одного вида дефекта:



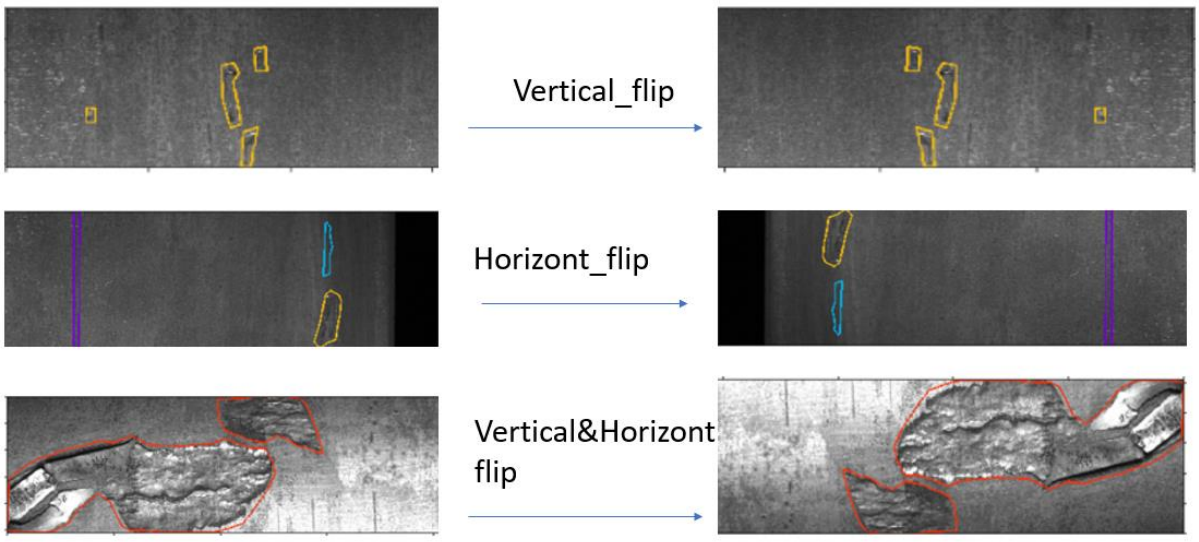


Представленный датасет является несбалансированным по числу представленных дефектов. Кроме того, следует обратить внимание на наличие еще одного дисбалансирующего фактора – превалирование числа пикселей фона над числом пикселей, занятых дефектами разных классов.

Предобработка данных.

Поскольку предложенный датасет является несбалансированным по количеству образцов изображений каждого дефекта и размерам фона/дефекта, проведем ряд процедур, которые должны снизить экстремальность двойного дисбаланса:

1. удалим изображения, содержащий 4 пустые маски дефекта (то есть изображения, не имеющие дефектов)
2. сделаем аугментацию изображений, содержащих заполненные маски миноритарных дефектов. В аугментации используем горизонтальный/вертикальный flip.



Разбиение на обучающую, валидационную и тестовые выборки

Поскольку датасет является экстремально несбалансированным по количеству заполненных масок различных дефектов, это нужно учитывать при разбиении данных на подвыборки для дальнейшего обучения.

При разбиение датасета на обучающую, валидационную и тестовую выборки мы учитывали несбалансированность числа присутствующих дефектов разных видов.

Разбиение проходило по двум направлениям:

1. отобраны изображения с одной заполненной маской и применен метод StratifiedShuffleSplit (по номеру заполненного класса) библиотеки Sklearn d в соотношении 50:30 и 50:50.
2. отобраны изображения с мультидефектностью и разделены между выборками с помощью метода train_test_split

DATA	total	class_1	class_2	class_3	class_4
full data	6666	897	247	5150	801
train data	4665	622	174	3603	565
validation data	1001	139	35	775	118
test data	1000	136	37	772	118

В результате размер обучающей, валидационной и тестовой выборки составил - 4665,1000 и 1001 соответственно.

Заметим, что перед формированием подвыборок была произведена очистка рассматриваемого датасета от изображений, содержащий 4 пустые маски, это уже

на начальной этапе исследования позволило существенно сократить переобучение моделей на фоновых пикселях изображений датасета.

Метрика качества.

В качестве заявленной метрики качества результатов соревнования была заявлена dice метрика. Dice может быть использован для сравнения пиксельного согласия между предсказанной маской и фактически данной:

$$DICE = \frac{2 * |X \cap Y|}{|X| + |Y|},$$

где X предсказанное множество пикселей, Y - настоящее. Коэффициент определяется как 1, когда X и Y пусты или полностью повторяют друг друга.

В терминах TP, FP, TN, FN метрика может быть переписана следующим образом:[10]

$$DICE = \frac{2 * TP}{(TP + FP) + (TP + FN)}$$

Учитывая факт несбалансированности выборки по количеству дефектов и характер предоставления таргета в моделях исследования (4 маски дефекта /4 маски дефекта + 1 маска фона) мы также будет отслеживать дополнительный dice только по заполненным маскам дефектов в разрезе каждого класса.

Функция потерь

В ходе проведенного исследования было использовано несколько функций потерь, основанных на кросс-энтропийной функции потерь.

[10] https://medium.com/datadriveninvestor/deep-learning-in-medical-imaging-3c1008431aaf

Общий вид функции:

$CE = -\sum_{i=1}^{C} t_i log(s_i)$,

где t_i = это верное значение, а s_i – вероятность, с которой данное значение предсказано.

1) Первый этап исследования обучался при минимизации BCE, являющейся комбинацией функции бинарной кросс-энтропии и dice коэффициента.

Определим бинарную кросс-энтропию ниже:

$$CE = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

$$f(s_i) = \frac{1}{1 + e^{-s_i}}$$

Бинарная кросс энтропия в качестве функции потерь стремиться увеличить вероятность предсказания позитивного класса. В крайнем случае полной уверенности функция потерь примет пограничное значение равное нулю. В нашем случае речь будет идти о предсказании 4-х масок с активациями сигмоиды.

Часто в качестве функции потерь применяется не бинарная кросс-энтропия, а ее комбинация с иной функцией потерь, отвечающей за пиксельное согласие образца и предсказания.

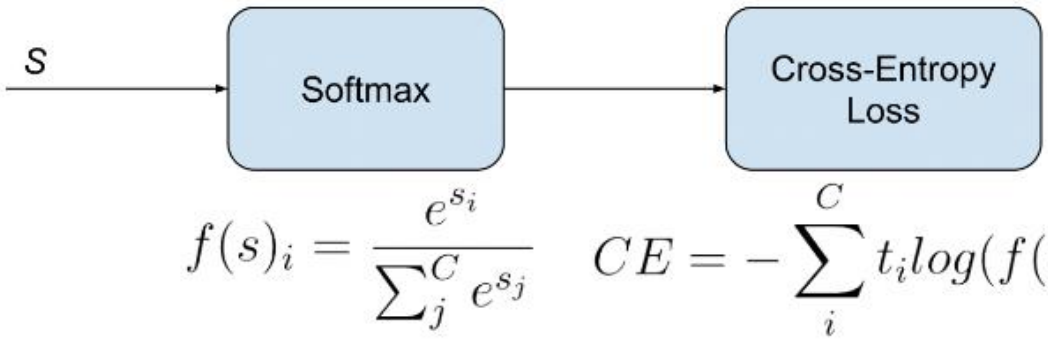
В ходе исследования на кросс-валидации было подобрано наиболее оптимальное сочетание двух функций в итоговой функции потерь:

BCE =0,7* binary_crossentropy - dice

2. Второй этап исследований обучался при минимизации функции потерь

CCE = 0,7*categorical_crossentropy - 0,25*dice.

Категориальная кросс-энтропия - еще один случай cross entropy loss для многоклассовой выборки с функцией активации softmax.



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \quad CE = -\sum_i^C t_i log(f(s)_i)$$

,

где s_i - вероятность позитивного класса.

В ходе обучения на кросс-валидации также были подобраны наиболее оптимальные коэффициенты функции потерь.

3) **Focal loss**[11] - модификация кросс-энтропийной функции потерь для несбалансированных классов. Первоначально функци была предложена в задаче детекции, с очень большой долей фона (background) по сравнению с долей площади, занимаемой искомыми объектами. Впоследствии стали применяться для работы в несбалансированными выборками в целом.

Идея focal loss состоит в оценивании каждого образца в функции потерь. Если образец уже классифицирован правильно, то его вклад в функцию потерь уменьшается. Минимизация функции потерь неявно сосредотачивается на проблемных классах. Кроме того, также взвешивается вклад каждого класса в проигрыш в более явном балансе классов.[12] Они используют сигмоидные активации,

[11] https://arxiv.org/abs/1708.02002 Focal Loss for Dense Object Detection. Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár (Submitted on 7 Aug 2017 (v1), last revised 7 Feb 2018 (this version, v2))

[12] https://gombru.github.io/2018/05/23/cross_entropy_loss/ Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. Raúl Gómez blog

поэтому фокальную потерю можно также рассматривать как двоичную потерю перекрестной энтропии, определяемой как:

$$FL = -\sum_{i=1}^{C} (1 - s_i)^{\gamma} t_i log(s_i),$$

где $(1 - s_i)^{\gamma}$ с параметром $\gamma \geq 0$ - фактор, снижающий влияние корректно предсказанных образцов (при $\gamma = 0$ focall loss превращается в бинарную (категориальную) кросс-энтропию).

Использование в качестве функции потерь комбинации focal loss с dice коэффициентом не привело (в отличие от предыдущих этапов) к повышению качества модели по сравнению с focal loss.

Таким образом, при проведении исследования в качестве функций потерь были использованы вариации кросс-энтропийной функции (binary crossentropy, categorical crossentropy, focal loss) с некоторыми поправками путем учета dice коэффициента.

ЭКСПЕРИМЕНТЫ

Эксперименты проводились по трем, указанным выше моделям, с параллельной data augmentation для изображений с миноритарными дефектами (flip, rotation).

Первый этап исследования (BCE Loss)

На первом этапе исследования рассматриваемые модели обучались при помощи sigmoid.

На вход данной модели подавалась фотография и четыре массива по размеру фотографии, характеризующие маски каждого из дефектов, применительно к конкретному изображению.

В качестве функции потерь (loss) при обучении данной модели использовалась BCE = 0,7 * binary_crossentropy - dice, показавшая лучшее качество модели по сравнению с обычной бинарной кросс-энтропией.Коэффициенты перед частями функции потерь подбирались на кросс-валидации.

Были получены следующие результаты:

1.1 U-net (4 classes sigmoid)

table1.1

items/dice	U-net_4_classes (BCE)				
	dice (all)	dice class_1	dice class_2	dice class_3	dice class_4
Dice	0,74	0,86	0,96	0,48	0,76

dice full mask	0,49	0,22	0,12	0,56	0,66

В целом модель дала неплохие результаты, однако, следует заметить, что качество сегментации на заполненных масках следует признать неудовлетворительным в отношении двух миноритарных классов (что скорее всего следует связать с крайней несбалансированностью датасета по заполненным маскам дефектов 1 и 2).

Также обратим внимание, что модель дает неудовлетворительный результат на TN по максимальному третьему классу.

Модель была удачно опробована на тестовом датасете соревнования и получила схожие с нашим решением метрики качества:[13]0.74 private_score и 0.73 public_score.

1.2 U-net with ResNet encoder (4 classes sigmoid)

table1.2

items	U-net with ResNet encoder(BCE)				
	dice (all)	dice class_1	dice class_2	dice class_3	dice class_4
dice	0.79	0.88	0.96	0.53	0.79
dice full mask	0.52	0.34	0.3	0.6	0.67

По сравнению с классической моделью U-net с аналогичной функцией потерь U-net with ResNet encoder дает более высокий общий коэффициент качества. Модель стала лучше разбираться с заполненными масками (особенно в отношении миноритарных классов). Также модель стала более правильно определять

[13] https://www.kaggle.com/olihma/unet-start-version

“чистую” маску, таким образом, что практически безошибочно угадывает отсутствие миноритарный классов.

1.3 ResNet with decoder (4 classes sigmoid)

table1.3

items	ResNet with decoder (BCE)				
	dice (all)	dice class_1	dice class_2	dice class_3	dice class_4
dice	0.87	0.86	0.96	0.6	0.92
dice full mask	0.49	0.0	0.0	0.6	0.51

Третья модель продолжает улучшать среднее качество модели с 0,79 до 0,87 dice. Модель приросла в качестве при сегментации дефектов третьего класса при работе с 4 классом (с 0,79 до 0,92).

При рассмотрении качества сегментации пустых масок легко заметить, что и здесь основные изменения коснулись третьего и четвертого дефектов. Качество сегментации мажоритарного класса повысилось в 2,3 раза. Что касается миноритарных классов, то модель практически безошибочно угадывает отсутствие дефекта.

Однако, предсказание модели заполненных масок поясняет, что общее улучшение качества детектирования было вызвано более точным пониманием дефекта 3 и 4 классов. Модель понизила качество при детектировании заполненных масок 3 и 4 дефектов и совершенно перестала понимать структуру 1 и 2. Модель переучилась на максимальном классе. Таким образом, несмотря на высокий dice модель невозможно признать качественной (устойчивость).

Второй этап исследования (CCE Loss)

В качестве дополнительного развития выход модели был дополнен 5 маской, характеризующей принадлежность каждого пикселя изображения к фону. Таким образом, пятая маска должна была провести классификацию пикселей и отметить те из них, которые не встречались ни в одной из 4 масок дефектов в качестве заполненных.

В качестве функции потерь была предложена CCE = 0.7*categorial_crossentropy - 0,25*dice (коэффициенты функции подобраны на cross validation).

Кроме того, была предпринята попытка скорректировать часть функции потерь (dice коэффициент). Подбирались коэффициенты для позитивных пикселей миноритарных классов. Наилучший результат был получен при (2, 2.5, 1, 2) для 1, 2, 3 и 4 классов дефектов.

Данные изменения позволили поднять качество модели на тесте, но привели к потере одного или двух миноритарных классов.

2.1 U-net 5 classes softmax

Применение иной функции активации, замена функции потерь и подбор параметров учета результатов качества сегментирования дефектов разных классов позволили существенно улучшить общее качество первой Unet модели.

Модель стала показывать большее качество заполненных масок. Однако, не удалось искоренить проблему переобучения на более представленных классах.

Правильно сегментировать хотя бы какие-то пиксели второго (меньшего класса) не получилось.

table2.1

items	U-net (CCE)				
	dice (all)	dice class_1	dice class_2	dice class_3	dice class_4
dice	0,84	0,88	0,96	0,6	0,94
dice full mask	0.59	0.38	0	0,66	0,57

2.2 U-net with ResNet encoder 5 classes softmax

Изменение параметров обучения второй модели, в отличие от первой не привело ни к каким значимым улучшениям качества модели. Общий dice снизился с 0,79 до 0,77, качество сегментации заполненных масок также упало, модель переучивается на больших классах и потеряла понимание миноритарных.

table2.2

items	U-net with ResNet encoder (CCE)				
	dice (all)	dice class_1	dice class_2	dice class_3	dice class_4
dice	0.77	0.88	0.97	0.56	0.88
dice full mask	0.40	0	0	0.46	0.5

Применительно к новой функции потерь, вторая модель показала существенно более низкое качество не только по сравнению с результатом U-net данного этапа, но и со своим предыдущим результатом. Модель понизила все показатели и перестала понимать структуру миноритарных (1 и 2) классов.

2.3 ResNet encoder-decoder 5 class softmax

Третья модель получила меньшее качества, чем первая, однако ее решение следует принять более стабильным. Модель хорошо научилась различать фон и дефект (лучше первой) и нашла все классы дефектов, не очень сильно переобучившись на больших классах.

table2.3

items	ResNet with decoder (CCE)				
	dice (all)	dice class_1	dice class_2	dice class_3	dice class_4
dice	0.76	0.75	0.89	0.52	0.9
dice full mask	0.54	0.42	0.39	0.55	0.65

Третий этап исследования (5 classes Focal Loss)

На третьем этапе исследования функция потерь была изменена на categorical focal loss.

Применение данной функции потерь к резко несбалансированной выборке должно было снизить тенденцию к переобучению на больших классах и лучше понять структуру наиболее сложного 3 класса.

3.1 U-net 5 classes softmax

Новая функция потерь была применена к наиболее быстро обучаемой и устойчивой модели Unet.

Указанная модель показывает немного более худшее качество по сравнению с предыдущей моделью Unet, однако результаты работы модели на заполненных масках более устойчивы.

table 3.1

items	U-net(categorical focal loss)				
	dice (all)	dice class_1	dice class_2	dice class_3	dice class_4
dice	0,81	0,77	0,96	0,6	
dice full mask	0.6	0.45	0,4	0,62	0,64

3.2 U-net with ResNet encoder 5 classes softmax

Обучение модели на новой функции потерь не принесло никакого прироста качества. Значение показателя dice коэффициента упало до рекордного 0,49. Определение заполненных масок также достигло рекордно низкого уровня. Модель абсолютно перестала определять отсутствие 3 и 4 класса дефектов на изображении.

table 3.2

items	U-net with ResNet encoder (Focal loss)				
	dice (all)	dice class_1	dice class_2	dice class_3	dice class_4
dice	0.49	0.74	0.78	0.29	0.16
dice full mask	0.33	0.12	0.11	0.28	0.28

3.3 ResNet with encoder-decoder

Третья модель показывает более высокое качество, однако, полностью теряет понимание миноритарных дефектов. Предыдущей своей версии данная модель проигрывает в своей предсказательной способности.

table 3.3

items	ResNet decoder (Focal loss)				
	dice (all)	dice class_1	dice class_2	dice class_3	dice class_4
dice	0.76	0.87	0.96	0.34	0.87
dice full mask	0.32	0	0	0.35	0.49

Сравнение результатов

table 4

Model	Dice (all)	Dice class_1	Dice class_2	Dice class_3	Dice class_4	Проблемы
U-net BCE	0,74	0,77	0,96	0,48	0,74	
U-net with ResNet encoder BCE	0,79	0,88	0,96	0,53	0,79	
ResNet with encoder_decoder BCE	0.87	0.86	0.96	0.6	0.92	потеря 1,2 классов
U-net CCE	0,84	0,88	0,96	0,6	0,94	потеря 2 класса
U-net with ResNet encoder CCE	0.77	0.88	0.97	0.56	0.88	потеря 1,2 класса
ResNet with decoder CCE	0.76	0.75	0.89	0.52	0.9	
U-net Focal loss	0,81	0,77	0,96	0,6	0,9	
U-net with ResNet encoder FL	0.49	0.74	0.78	0.29	0.16	низкое качество модели
ResNet with decoder FL	0.76	0.87	0.96	0.34	0.87	потеря 1,2 классов

Итак, в ходе выполнения работы были проведены три этапа исследований на моделях.

Были выявлены следующие способы повышения качества:

1. удаление изображений с пустыми масками всех 4 дефектов
2. генерация дополнительных изображений/масок для миноритарных классов
3. добавление дополнительной псевдо маски, характеризующей состояние фона изображений.
4. борьба с несбалансированность выборки при помощи подбора коэффициентов функции потерь (удалось не до конца) и применении Focal loss.

Модели в первом эксперименте сразу начали показывать неплохой результат, однако в показывали признаки переобучения на миноритарных классах. Добавление пятой маски и переход на категориальной кросс энтропии помогло улучшить структурное понимание дефектов, однако уйти от переобучения на первых двух моделях не удалось.

Наилучшие показатели были достигнуты при обучении классической архитектуры U-net с функцией потерь focal loss. Вероятно, это было связано с крайней несбалансированностью выборки сразу по нескольким направлениям: фон-дефект, дефекты между собой.

Перспективными следует признать первую и третью архитектуры с категориальной кросс-энтропийной функцией потерь, качество которых можно улучшать дальнейшим подбором коэффициентов для dice разных классов.

ЗАКЛЮЧЕНИЕ

В данной работе было проведено исследование по сегментированию дефектов стального листа, результаты которого могли бы в дальнейшем быть применены в целях контроля качества стального листа.

Предоставленный датасет имел крайне несбалансированную структуру представленности классов, что, вероятно, отражает распределение различных классов дефектов. Несбалансированность классов присутствовала одновременно с несбалансированностью распределения дефект/фон.

В работе было проведено обучение трех архитектур сверточных нейронных сетей (U-net, U-net with ResNet encoder, ResNet_decoder) на трех итоговых функциях потерь (BCE, CCE, Focal loss).

Практически все обученные модели хорошо предсказывали факт отсутствия дефектов разных классов, а потому с очень хорошим качеством могут быть использованы в решении задачи определения наличия дефектов.

Наиболее удачной архитектурой для решения поставленной задачи, среди рассматриваемых в данном исследовании, следует признать U-net, которая наилучшим образом справляется с качественным сегментированием заполненных масок.

Вследствие потери понимания структуры второго класса моделью U-net на втором этапе экспериментов, лучшей моделью следует признать U-net с Focal loss. Однако, следует заметить, что второй эксперимент, вероятно, может показать более качественное предсказание при дальнейшем (удачном) подборе параметров для работы с несбалансированными классами.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. https://www.kaggle.com/c/severstal-steel-defect-detection
2. https://arxiv.org/pdf/1605.06211.pdf Fully Convolutional Networks for Semantic Segmentation Evan Shelhamer∗ , Jonathan Long∗ , and Trevor Darrell, Member, IEEE 20.5.16
3. F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano, "Toward automatic phenotyping of developing embryos from videos," Image Processing, IEEE Transactions on, vol. 14, no. 9, pp. 1360–1371, 2005.
4. https://arxiv.org/pdf/1505.04597.pdf U-Net: Convolutional Networks for Biomedical Image Segmentation Olaf Ronneberger, Philipp Fischer, and Thomas Brox 18.5.15
5. https://arxiv.org/pdf/1512.03385.pdf Deep Residual Learning for Image Recognition Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun Microsoft Research
6. https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Networks_yourself.ipynb
7. https://arxiv.org/abs/1806.01054?context=cs.CV Jindong Jiang, Lunan Zheng, Fei Luo, Zhijun Zhang RedNet: Residual Encoder-Decoder Network for indoor RGB-D Semantic Segmentation
8. teleported.in/posts/decoding-resnet-architecture/
9. https://medium.com/datadriveninvestor/deep-learning-in-medical-imaging-3c1008431aaf
10. https://arxiv.org/abs/1708.02002 Focal Loss for Dense Object Detection. Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár (Submitted on 7 Aug 2017 (v1), last revised 7 Feb 2018 (this version, v2))
11. https://gombru.github.io/2018/05/23/cross_entropy_loss/ Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. Raúl Gómez blog
12. https://gombru.github.io/2018/05/23/cross_entropy_loss/ Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. Raúl Gómez blog

Приложение

Unet

```
def model_Unet():


    inputs = Input((None, None, 1))
    bnorm1 = BatchNormalization()(inputs)
    conv1 = Conv2D(32, (3, 3),init='he_uniform', W_regularizer=regularizers.l2(0.0001), activation='relu', padding='same')(bnorm1)
    conv1 = Conv2D(32, (3, 3), activation='relu', W_regularizer=regularizers.l2(0.0001), padding='same')(conv1)
    #drop1 = Dropout(0.25)(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(64, (3, 3),activation='relu', padding='same')(pool1)
    bnorm2 = BatchNormalization()(conv2)
    conv2 = Conv2D(64, (3, 3),activation='relu', padding='same')(bnorm2)

    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)
    bnorm3 = BatchNormalization()(conv3)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(bnorm3)

    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool3)
    bnorm4 = BatchNormalization()(conv4)
    conv4 = Conv2D(256, (3, 3),  activation='relu', padding='same')(bnorm4)

    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)
```

```
conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool4)
bnorm5 = BatchNormalization()(conv5)
conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(bnorm5)


up6 = concatenate([Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(conv5), conv4], axis=3)
conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(up6)
bnorm6 = BatchNormalization()(conv6)
conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(bnorm6)

up7 = concatenate([Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(conv6), conv3], axis=3)
conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(up7)
bnorm7 = BatchNormalization()(conv7)
conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(bnorm7)

up8 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(conv7), conv2], axis=3)
conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(up8)
bnorm8 = BatchNormalization()(conv8)
conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(bnorm8)

up9 = concatenate([Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(conv8), conv1], axis=3)
conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(up9)
bnorm9 = BatchNormalization()(conv9)
conv9 = Conv2D(32, (3, 3),  activation='relu', padding='same')(bnorm9)

conv10 = Conv2D(5, (1, 1), activation='softmax')(conv9)

model = Model(inputs=[inputs], outputs=[conv10])
return model
```

Unet with ResNet encoder

```
def bn_act(x, act=True):
    'batch normalization layer with an optinal activation layer'
    x = BatchNormalization()(x)
    if act == True:
        x = Activation('relu')(x)
    return x
def conv_block(x, filters, kernel_size=3, padding='same', strides=1):
    'convolutional layer which always uses the batch normalization layer'
    conv = bn_act(x)
    conv = Conv2D(filters, kernel_size, padding=padding, strides=strides)(conv)
    return conv
def stem(x, filters, kernel_size=3, padding='same', strides=1):
    conv = Conv2D(filters, kernel_size, padding=padding, strides=strides)(x)
    conv = conv_block(conv, filters, kernel_size, padding, strides)
    shortcut = Conv2D(filters, kernel_size=1, padding=padding, strides=strides)(x)
    shortcut = bn_act(shortcut, act=False)
    output = Add()([conv, shortcut])
    return output
def residual_block(x, filters, kernel_size=3, padding='same', strides=1):
    res = conv_block(x, filters, kernel_size, padding, strides)
    res = conv_block(res, filters, kernel_size, padding, 1)
    shortcut = Conv2D(filters, kernel_size, padding=padding, strides=strides)(x)
    shortcut = bn_act(shortcut, act=False)
    output = Add()([shortcut, res])
    return output
def upsample_concat_block(x, xskip):
    u = UpSampling2D((2,2))(x)
    c = concatenate([u, xskip])
    return c
```

```
def ResUNet():
    f = [16, 32, 64, 128, 256]
    inputs = Input((None, None, 1))
    #inputs = preprocess_input(inputs)

    ## Encoder
    e0 = inputs
    x = GaussianNoise(0.2)(e0)
    e1 = stem(x, f[0])
    e2 = residual_block(e1, f[1], strides=2)
    e3 = residual_block(e2, f[2], strides=2)
    e4 = residual_block(e3, f[3], strides=2)
    e5 = residual_block(e4, f[4], strides=2)

    ## Bridge
    b0 = conv_block(e5, f[4], strides=1)
    b1 = conv_block(b0, f[4], strides=1)

    ## Decoder
    u1 = upsample_concat_block(b1, e4)
    d1 = residual_block(u1, f[4])

    u2 = upsample_concat_block(d1, e3)
    d2 = residual_block(u2, f[3])

    u3 = upsample_concat_block(d2, e2)
    d3 = residual_block(u3, f[2])

    u4 = upsample_concat_block(d3, e1)
    d4 = residual_block(u4, f[1])

    outputs = Conv2D(5, (1, 1), padding="same", activation="sigmoid")(d4)
    model = Model(inputs, outputs)
    return model
```

ResNet

```
    # Stage 2
    X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2, block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')

    ### START CODE HERE ###

    # Stage 3 (≈4 lines)
    X = convolutional_block(X, f = 3, filters = [128, 128, 512], stage = 3, block='a', s = 2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    # Stage 4 (≈6 lines)
    X = convolutional_block(X, f = 3, filters = [256, 256, 1024], stage = 4, block='a', s = 2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    # Stage 5 (≈3 lines)
    X = convolutional_block(X, f = 3, filters = [512, 512, 2048], stage = 5, block='a', s = 2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    # AVGPOOL (≈1 line). Use "X = AveragePooling2D(...)(X)"
    X = AveragePooling2D((2,2), name="avg_pool")(X)
```

```
### END CODE HERE ###

# output layer
X = Flatten()(X)
X = Dense(classes, activation='softmax', name='fc' + str(classes), kernel_initializer = glorot_uniform(seed=0))(X)


# Create model
model = Model(inputs = X_input, outputs = X, name='ResNet50')

return model
```

ResNet decoder

```
def autoencoder_RES():
    X_input = Input((img_h, img_w, 1))

    # Zero-Padding
    X = ZeroPadding2D((3, 3))(X_input)

    # Stage 1
    X = Conv2D(64, (7, 7), strides=(2, 2))(X)
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    # Stage 2
    X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2, s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2)
    X = identity_block(X, 3, [64, 64, 256], stage=2)

    ### START CODE HERE ###

    # Stage 3 (≈4 lines)
    X = convolutional_block(X, f = 3, filters = [128, 128, 512], stage = 3, s = 2)
    X = identity_block(X, 3, [128, 128, 512], stage=3)
    X = identity_block(X, 3, [128, 128, 512], stage=3)
    X = identity_block(X, 3, [128, 128, 512], stage=3)

    X = convolutional_block(X, f = 3, filters = [256, 256, 1024], stage = 4,s = 2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4)
    X = identity_block(X, 3, [256, 256, 1024], stage=4)
    X = identity_block(X, 3, [256, 256, 1024], stage=4)
    X = identity_block(X, 3, [256, 256, 1024], stage=4)
    X = identity_block(X, 3, [256, 256, 1024], stage=4)
```

```
# Stage 5 (≈3 lines)
X = convolutional_block(X, f = 3, filters = [512, 512, 2048], stage = 5,s = 2)
X = identity_block(X, 3, [512, 512, 2048], stage=5)
X = identity_block(X, 3, [512, 512, 2048], stage=5)

X = AveragePooling2D((2,2), name="avg_pool")(X)


X = UpSampling2D((4,4)) (X)


X = dec_identity_block(X, 3, [512,512,2048], stage=5)
X = identity_block(X, 3, [512, 512, 2048], stage=5)
X = dec_convolutional_block(X, f = 3, filters = [512,512,1024], stage = 5, s = 2)

X = dec_identity_block(X, 3, [256, 256, 1024], stage=4)
X = dec_identity_block(X, 3, [256, 256,1024], stage=4)
X = dec_identity_block(X, 3, [256, 256, 1024], stage=4)
X = dec_identity_block(X, 3, [256, 256,1024], stage=4)
X = dec_convolutional_block(X, f = 3, filters = [256, 256, 512], stage = 4, s = 2)

X = dec_identity_block(X, 3, [128, 128, 512], stage=4)
X = dec_identity_block(X, 3, [128,128, 512], stage=4)
X = dec_identity_block(X, 3, [128,128, 512], stage=4)
X = dec_convolutional_block(X, f = 3, filters = [128, 128, 256], stage = 4, s = 2)


X = dec_identity_block(X, 3, [64, 64, 256], stage=2)
X = dec_identity_block(X, 3, [64, 64, 256], stage=2)
X = dec_convolutional_block(X, f=3, filters=[64,64,256], stage=2, s=1)
```

```
X = UpSampling2D((2,2))(X)
X= ZeroPadding2D((0, 16))(X)
X = Conv2D(5, (1, 1), activation='softmax')(X)

#out = Reshape((img_h, img_w,3))(X)
model = Model(inputs=[X_input], outputs=[X])

return model
```