Описание полученной модели

В качестве архитектуры модели была выбрана Unet.
Архитектура полученной модели представлена на рисунке 1.

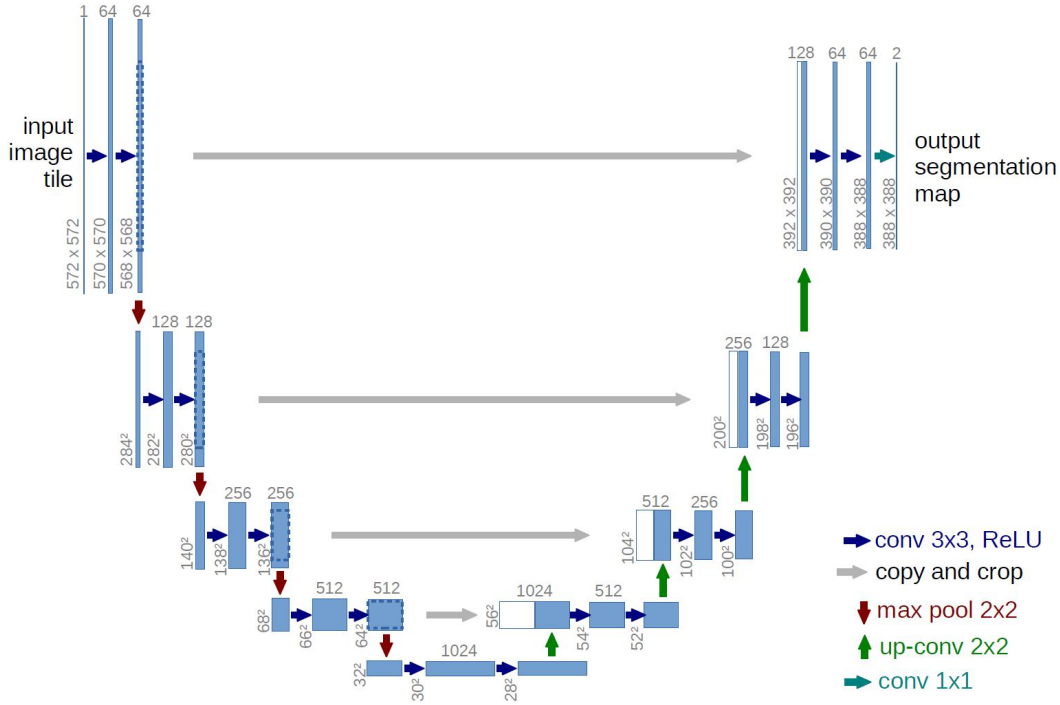

Рисунок 1 - Unet архитектура

Левая половина модели называется encoder или feature extractor, эта часть преобразует изображения в карты признаков, а правая восстанавливает по ним маску.

Входные данные модели:
Изображение с формой (height, width, 1) - это изображение с одним каналом - оттенки серого.

Выходные данные модели:
Тензор с формой (height, width, 3). Число каналов равно количеству классов. Каждому пикселю соответсвует массив с вероятностями пренадлежности пикселя к каждому из классов.
Далее к этому тензору применяется операция argmax, в результате чего пиксель соотносится к единственному классу с наибольшей вероятностью, получаются маски для каждого класса. С использованием полученных масок, на исходное изображение наносятся контуры объектов, для трещин считается их ширина.

Архитектура Unet была выбрана по причине простоты реализации и неплохих результатов, что очень хорошо для прототипизации.

Планируется исследовать качества детектирования в зависимости от архитектуры.

Недостатки Unet:
Модель задизайнена под небольшие разрешения изображений, а датасет представляет из себя изображения со сверх высоким разрешением, приходится очень сильно сжимать изображения, из-за чего может теряться информация.

Особенности датасета:

1. Классы очень несбалансированны (~90% пикселей фона, 9% трещин, 1% зазоров). В таком случае модель становится практически необучаема, ведь если она отнесет все пиксели к 1му классу, то значение потери будет невелико.
Проблема решается добавлением весов к функции потерь, которые уравняют вклад каждого класса в ошибку.

2. Небольшой объем датасета.
Решается случайным применением операций вертикального/горизонтального поворота к изображениям.

3. Черные прямоугольники.
Всегда имеют чисто черный цвет, закрашиваются средним значением цвета по всему изображению, модель не замечает их.

Используемые технологии:

Язык программирования - Python, т.к. он имеет большое количество библиотек и фреймворков, реализующих весь низкий уровень реализации машинного и глубокого обучения, обработки изображений и т.д.

Фреймворк глубокого обучения - Keras, т.к. он наиболее высокоуровнев и прост в освоении, очень хорош для прототипизации.

Структура программной реализации:

Модель обучалась в облаке, с использованием Google Colab,
Ссылка на Google Drive со скриптами и тестовыми результатами:

https://drive.google.com/drive/folders/16i7_nCmUm6NzKEgRxWrl7qzZeKYqrHz4?usp=sharing

train.ipynb - Jupiter Notebook в котором обучалась модель
utils.py - Модуль с написанными инструментами для обучения (генераторы данных для обучения, загрузка и форматирования данных) и визуализации (рисование обводки и ширины объектов на исходном изображении)

test.ipynb - Jupiter Notebook в котором модель прогоняется на отложенной тестовой выборке, и везуализированный результат сохраняется в папку test.

dataset.csv - таблица с ссылками на образцы и их разметки.

model.h5 - файл с весами модели, полученный в результате обучения

test - папка с результатами прогона модели и желаемым результатом:
n_predict.png - предсказание n-го изображения моделью
n_truth.png - разметка n-го изображения, которая ожидается.

В итоге получена модель, которая неплохо справляется с детекцией трещин, но не очень хорошо с детекцией зазоров, после добавления новых кадров, проблема должна решиться.