

(Challenge 01)
Rust Detection

By :

Naveenraj Devaraj (ndevaraj17@ubishops.ca)

A report submitted to the **Mohammed Ayoub** of Department of
Computer Science in conformity with the requirements for the
degree of Master of Science.

Bishop's University
QC, Canada
August 2019

Abstract

The Primary Objective of this Challenge is to identify the rust portion of the given image using U-Net Architecture. U-Net model is a well defined architecture in the Convolutional Neural Network and it will easily integrate with it to increase the accuracy and the prediction rate. We generally use the prepared dataset to train our model and use it to predict the rust portion of the image. The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. We show that such a network can be trained end-to-end from very few images and outperforms the prior best method in a sliding-window convolutional network.

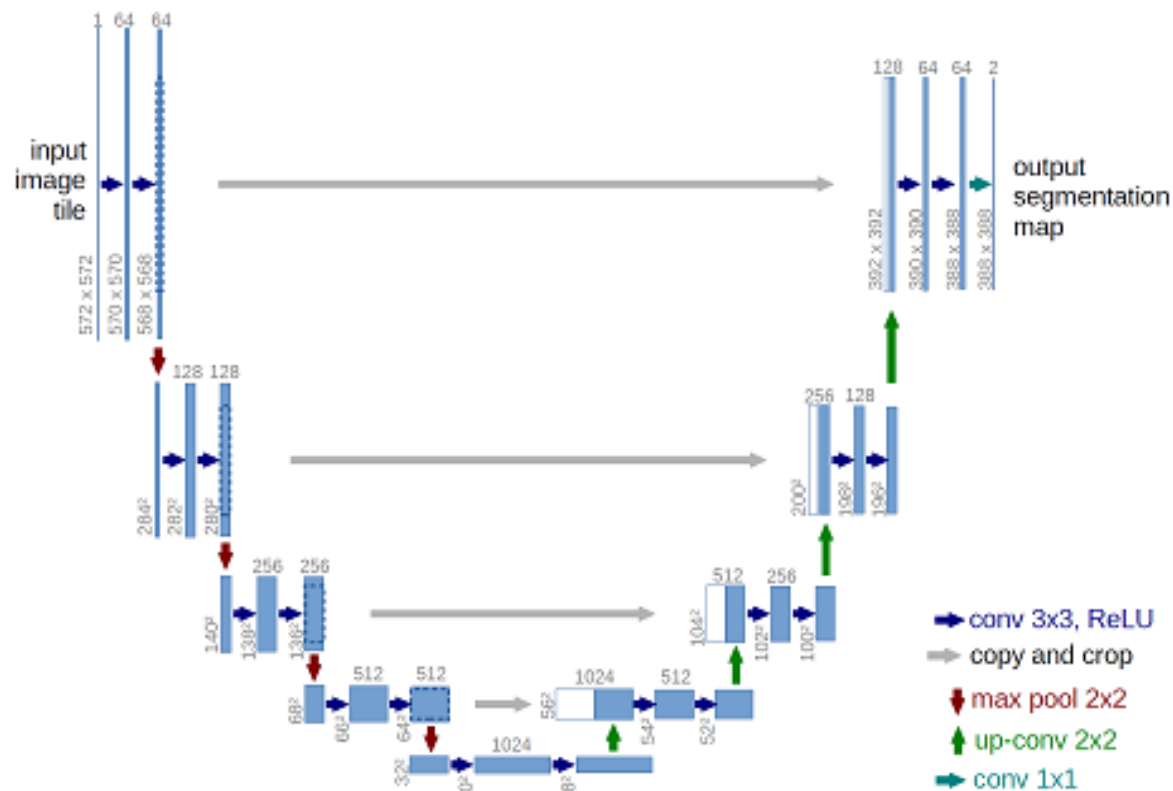
I. Introduction

Initially processing the given source image is the computer vision task where a model takes an input and classifies each pixel, it has been widely subjected over well defined architecture to make it more precious and accurate. we begin by preparing the known dataset as a trained model it has been used as a best sort of annotation image to perform our our prediction. There are many approaches to perform the segmentation of the image to identify the rust part of the image using step by step pixel process. Currently we are achieving it by U-Net architectural design for building the model. Typically, the Convolution Networks will perform the classification by applying the class labels to the output of the given image and it is achieved by pixel by pixel especially in rust identification we need to perform a detailed localization towards the rust part to identify it with high accuracy rate. It follows the sliding window concept to identify the class label of each pixel by defining the local patch around that pixel.

II. Implementation/ Elaboration

U-Net Architecture:

U-net is a deep segmentation network which have architecture is described below:



U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

- The main idea of Unet is using the auto encoder-decoder network. The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization.
- The left network using some 2D-convolution layer (3x3) and down-sampling for extract this information and compress the information in one vector (1x1024) and the right network contain some up-sampling , 2D-convolution (3x3) for reconstruction the information. It means It can localize the rust.

Conv2D:

Generally, the layers near the input is having more convolutional filter when compare to the layer nearest to the output. Conv2D is used to learn the convolution filter and at the beginning of the process it will learn more convolutional filter when compare to the end layer closer to the output. The number of filters for each layer will be in increasing order and later will be in decreasing order to increase the accuracy of identifying the rust location.

MaxPooling2D:

Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. Max pooling is done by applying a *max filter* to (usually) non-overlapping sub regions of the initial representation.

Methodology:

Step 1: Build the model using the main.py

We use the trainGenerator to specify the data set images and the annotations images and which help us to can generate image and mask at the same time use the same seed for image_datagen and mask_datagen to ensure the transformation for image and mask is the same. We internally implement the model using the tensorBoard, ModelCheckpoint, ReduceLROnPlateau, EarlyStopping from the tensorflow package.

Step 2: Prepare the annotation Images using test.py

We generate the annotation images to identify the accuracy percentage of the rust prediction using the HSV methodology it involves the open cv package by generating the lower mask (0-5) and upper mask (175-180) of RED.

Step 3: Rust detection using the model in Predict.py

In this phase, we use the developed model with the source images to identify the rust prediction using the UNET architecture by implanting the convolution filter, Max pooling for the up sampling for each and every image to read the image and resize to the same dimension with input of the model and then predict the image and get the result of the image and change to opencv format then write it in the result file.

Step 4: Accuracy Calculation using Eval.py

It will predict the image and compare with annotation image for calculate the accuracy of the prediction, the annotation images are created by using the standard HSV based lower and upper mask prediction. So when comparing the predicted image with the annotation image will provide you the most accuracy of the prediction percentage of the result.

For example: Model will predict the image-1_result.png in 'images' folder and compare with image-1.png in 'annotations' folder to find the accuracy.

Required packages:

All the required packages are included in the requirement.txt file, While performing the execution need to run the requirement file to preinstall all the dependent packages using the following command

Pip3 install -r Requirement.txt





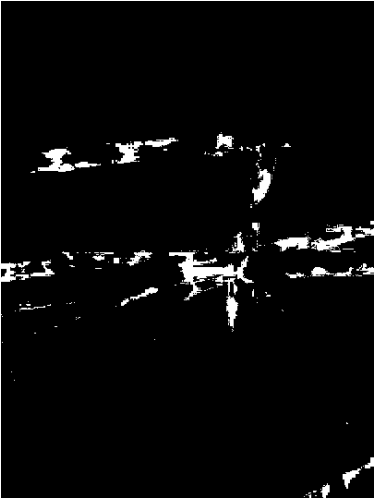

Dataset

The dataset is using for training is in this link :

<https://drive.google.com/open?id=1bVfL5Owni2ZVC6H1oM98ucXrZEy6iFHz>

Result

- This is some example results of best model :
 - + Epochs : 19
 - + Accuracy : 96.445%

Original Image	Annotation images	Result
		
		



How to run this code :

1. Install python enviroment and install the package in file requirement.txt
2. Predict the image :
 - File : predict.py
 - Change the image directory in 'img_path' line 14
 - Change the model path in 'model_dir' line 15 < default in folder "trained_weights" >
 - Change saving image directory in 'save_dir' line 16
 - run "python predict.py" and the result image is in save_dir

3. Eval the accuracy :

- File : eval.py
- Change the image directory in 'img_path' line 29
- Change the model path in 'model_dir' line 30 < default in folder "trained_weights" >
- Change annotation path in 'anno_path' line 31

Run "python eval.py" and you can see the accuracy

Drawback:

It is quite slow because the network must be run separately for each patch, and there is a lot of redundancy due to overlapping patches

There is a trade-off between localization accuracy and the use of context. Larger patches require more max-pooling layers that reduce the localization accuracy, while small patches allow the network to see only little context.

Conclusion:

The u-net architecture achieves very good performance on different rust segmentation applications. It only needs very few annotated images and has a very reasonable training time of an hour to provide the prediction accuracy up to 90% in the result. I am sure that the u-net architecture can be applied easily to many more tasks.

Reference:

1. <https://arxiv.org/pdf/1505.04597.pdf?fbclid=IwAR2N8LSD3NDONUWlj6540wLOcdaBCyjRPkbHdsnMUwmqiPsA0racp79RIIA>
2. <https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>
3. <https://openreview.net/pdf?id=Skft7cijM>