Challenge - 1
Identification of Corroded Areas in an Image

Done By,
Mobin Mustakali Momin
002251088

Introduction:

Corrosion is the process of breakdown the materials, because of the chemical reactions. It is mainly caused due to oxidation with the air molecules and due to the presence of H2O. corrosion is also caused because of the acidic reaction between materials. Corrosion is usually caused on metals. Whenever a material gets corroded, the physical property of that element changes and hence making corrosion a very dangerous problem. The corrosion on an element can cause any structure collapse. For example, a corroded bridge, pipeline, vessel or a building is very dangerous as it can cause the structure to collapse.
Due to these side effects of corrosion it is very important to detect corrosion in an element to avoid any such catastrophic event from happening. The old school way of detecting corrosion is by detecting the corrosion manually. Where in a human being will search for the corroded area in and mark it as corroded, but this process can be very costly and dangerous as there is a possibility that a human being might miss a potential and corroded area. Also it is not possible for a human being to look out for corrosion on certain places due to the limitation of a human's capabilities.
Computer Vison is an approach that solves all the above problems of detecting corrosion and it is a very cost effective and accurate detection of corrosion with little or no manual intervention. In this project we use the deep learning model to determine whether a given image is a rust image or not and detect the corrosion in an image using the computer vision techniques.

Methodology:

1. Deep Learning for Image Classification

The most favorable approach for deep learning on small dataset is by making use of the pre-trained network. A pre-trained network is a network that comprises of a data from the previously trained large dataset on a large scale image classification task. If the original dataset is huge enough and general enough, then the spatial feature hierarchy learned by the pre-trained network can play the role of generic model, and hence its features can be useful for our image classification. In my project I have made use of a large convent trained on ImageNet dataset, and use this to do binary classification of whether a give image is a rust or no rust image. An ImageNet is a dataset of more than 15 million images that belong to 22,000 categories. ImageNet contains variable-resolution images and hence for this reason the images are down sampled to a resolution of 256*256.

a. Feature Extraction

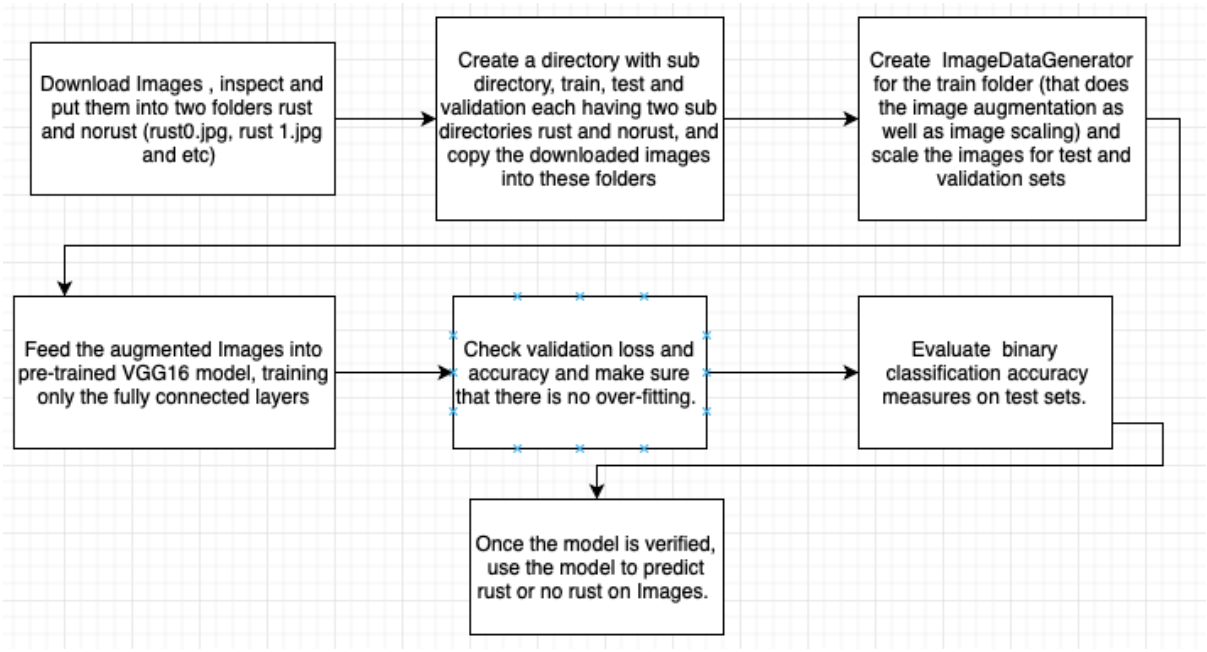


Fig 1: Image Classification Flow

In this project, I have made use of the VGG16 architecture which is a simple and widely used convent architecture for ImageNet. The VGG16 architecture was proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolution Networks for Large Scale Image Recognition. In the VGG16 architecture, the image is passed through a stack of convolution layers, where filters are used with a very small (3*3) convolution filter layers. Convents used for Image classification are divided into two parts, it starts with a series of pooling and convolution layers and ends with a very densely-connected classifier. The first part is called convolution base. In convnets the feature extraction simply consists of using the convolution base of the previously-trained network, applying the new data on that base and training the new classifier on top of the output. By using the VGG16 network, trained on ImageNet, to extract interesting features from the rust and no-rust images, and train the rust vs norust classifiers on top of these features.

Since I have a very limit limited number of data sets i.e. 70 rust and 60 no rust images. Data augmentation proves to be a very efficient way of training the deep learning models. In order to use this technique, the Keras provides us with a very terrific API called the Keras ImageDataGenerator by producing more images by zooming, shifting and rotating the image. The images are scaled by dividing the pixel intensities by 255.

Once the training data has been created, the next approach is to create our own Convolution Neural Network. To start the image classification, the first step of image Classification is to use the VGG16 model, with weights from the ImageNet trained model which is provided by the keras out of the box via a simple API. Since we are using the transfer learning approach, we will freeze the convolution base from the pre trained model and train only the only the last fully connected layers. The reason we freeze the convolution base is, as we can see in fig 2 below the convolution base of the VGG16 model has 14,714,688 parameters, which is very huge considering the fact that the classifier that we will be adding on the top has around 2 million parameters. As a result of freezing the convolution layer it will prevent the weights of convolution layer to be updated during the training and the number of trainable parameter in the new model is reduced from 14,7 million to 2 million. If we do not freeze the layer then the representations that

are previously learned by the convolution base layer will get modified and updated during the training. In keras freezing a network layer is done by simply setting the trainable attribute to false (set conv_base.trainable = False), which by default is set to True.
In my model, I have made use of RMSprop optimizer and binary cross-entropy loss. I have trained the model for 30 epochs, passing the output of ImageDataGenerator as a parameter, and in the evaluation step we get amazing results.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
```

Fig 2:

A good approach to knowing that our model is performing well loss is by plotting the training and the validation loss/accuracy with the epoch number. As we can see in the graph plotted below in fig 4. We can conclude that the results obtain gives us an idea that the validation data fits well with the model and that there is no overfitting.
After performing the above procedure, comes an important step. The validation of the model, we do this by exposing our model to images that it has not seen before (i.e. the test images).
By using this approach, we are getting an accuracy of about 85%, which is a great result. This concludes that the model is performing excellently on the test images. We can check the predictions that are generated by the model on the sample images. In order to perform this task, we load the test images along with the target size, and then we convert this images into an array representation and use to predict the output class of an image, which is rust or no rust image in our case. If the prediction made by the model gives a probability of 50% or more, then the given image is classified as a rust image. Else if the prediction made by the model predicts the probability of less than 50% then the image is classified as a no-rust image.

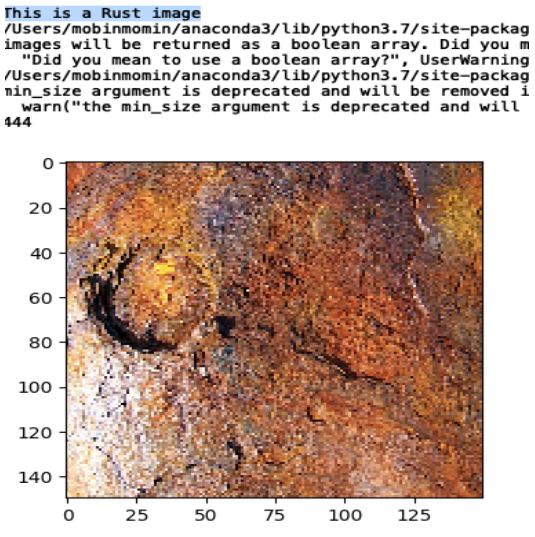


Fig 3a: The model predicts it as a Rust Image

Fig 3b: The model predicts this as No Rust Image.

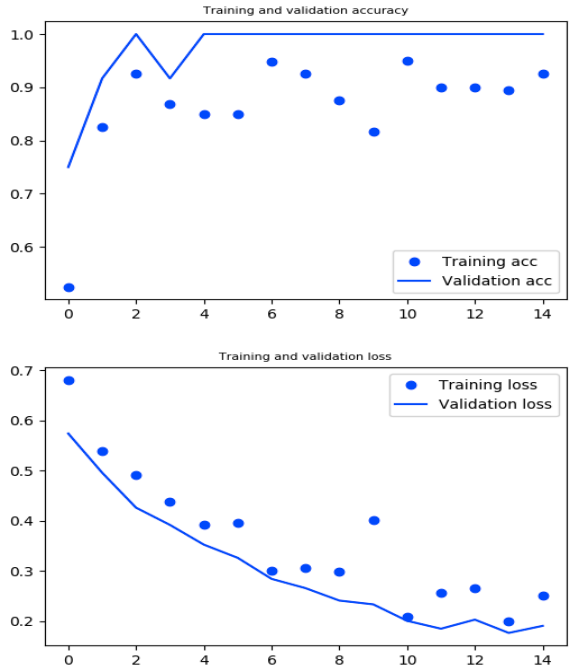


Fig 4: Training and validation loss/accuracy.

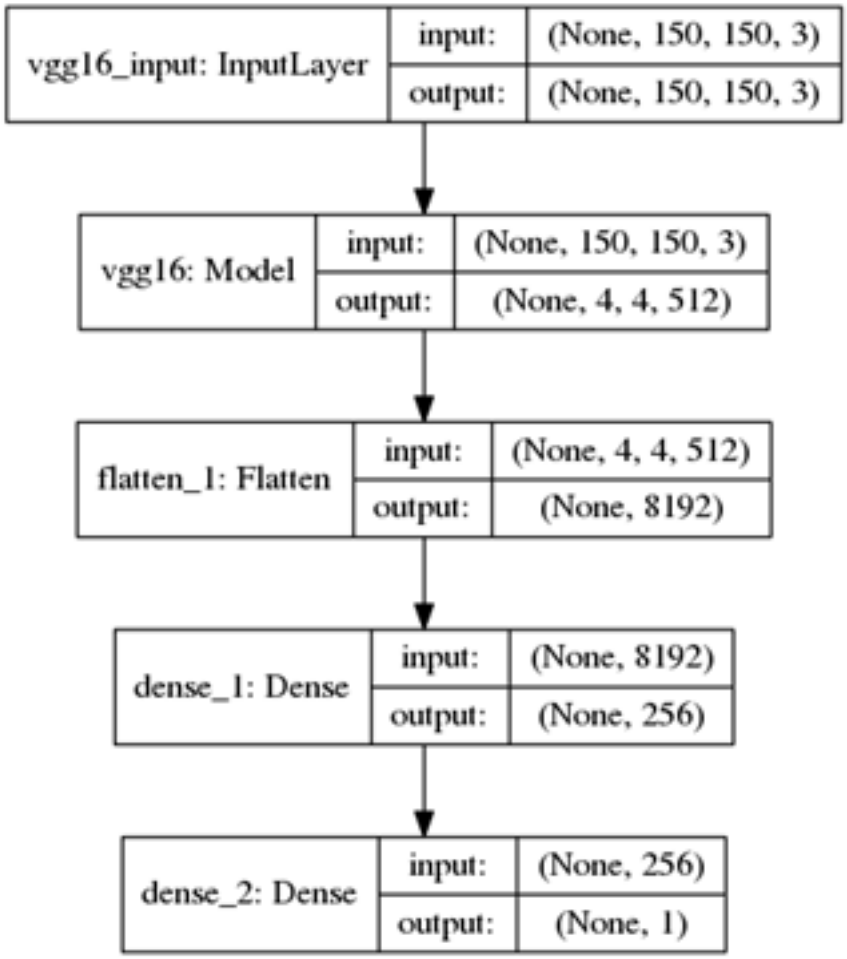


Fig 5: Custom Model using the pre- trained VGG16

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fig: 6

2. Get Corroded Region in an Image.

Now, after successfully differentiating weather a given image is a rust or no rust image, the next step will be to identify the corroded region's in that image. The first step in the approach is to get a RGB image, and convert the RGB image into HSV. The reason we convert a RGB image into HSV is to reduce the impact of illumination on an image. After converting the

image into HSV the next step is to extract the red components from the HSV image. In order to get the red component from the image, we get the lower and the upper range of the red component. After running through a number of filters, the best upper range of HSV discovered was [H=2, S=60, V=75] and the upper range discovered was [H= 12, S = 250, V = 250].

Since a rust image does not have any shape or structure, determining the corroded area in an image using a shape or structure will be very difficult. But one thing about a rust is the rough texture. Roughness is related to the energy of the gray-level Co-occurrence matrix. We calculate the energy level in the gray scale image. But before calculating the GLCM, we would like to get the list of blocks containing at least the block threshold proportion of red pixels. After we get this blocks it will be interesting to calculate the energy levels of the GLCM. If the energy level of these blocks are less than the threshold value, which shows that it exhibits a rough texture and are to be inspected more. The threshold value is set to 0.07. Meaning any block of an image have an energy level less than 0.07 are to be inspected for detection of corrosion in an image. After we determine the potential block to be inspected we make use of the color information that appears in the corroded areas. It is here where we use our HSV values. If a color is close to white or black, then it is classified as non rust region. In other case we will built an histogram and subsequently zero out the entities that have values less than the 10% of the highest peak. And hence in the end the pixel with HS(h, s) value greater than 0 will be classified as corroded. In the end we will create a boundary around every corroded region in the image.

In order to configure the parameters for the roughness stage we have to consider several values for calculating the GLCM. Values such as the distance and the orientation. The distance is assigned to 5 pixels and orientation to 0.

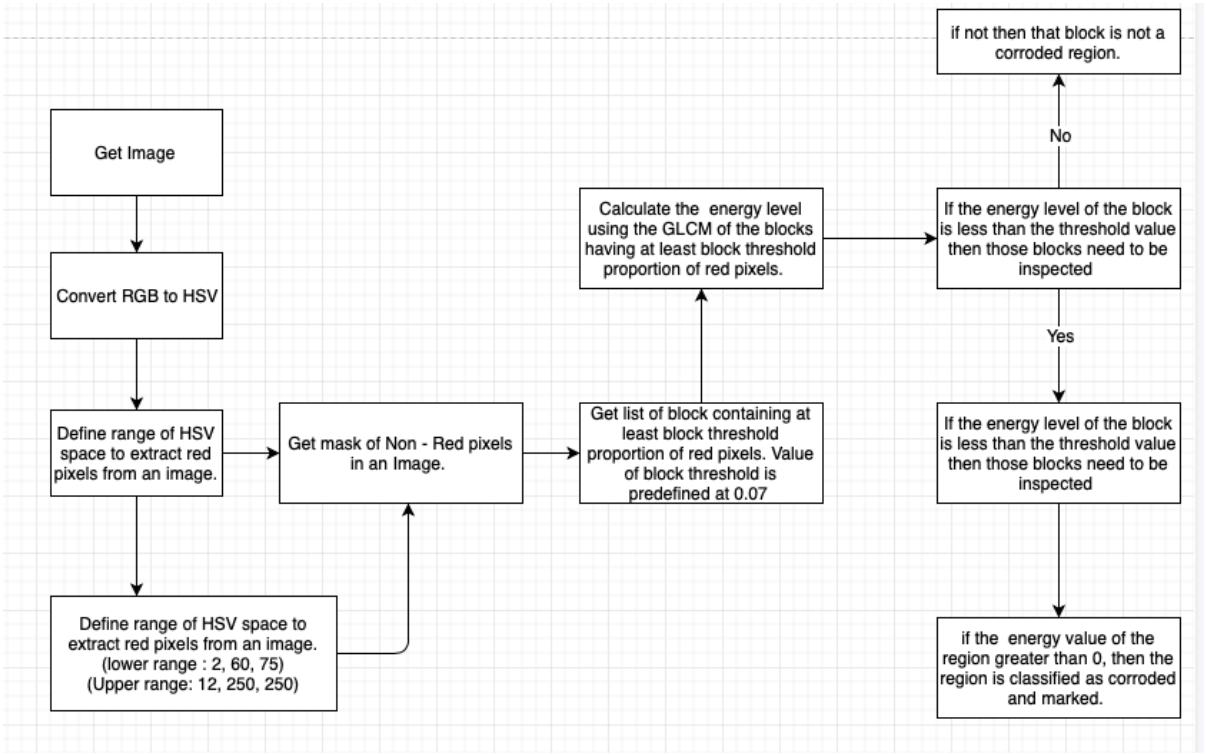
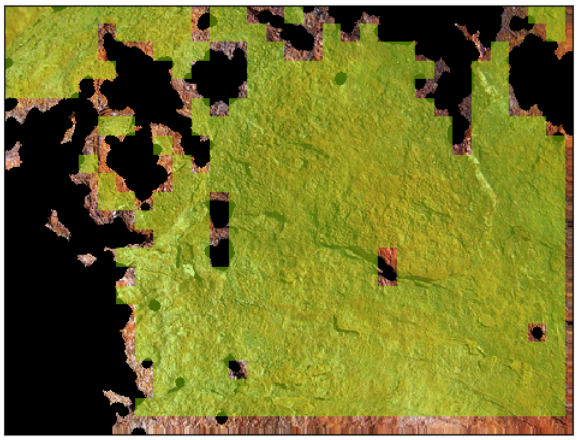
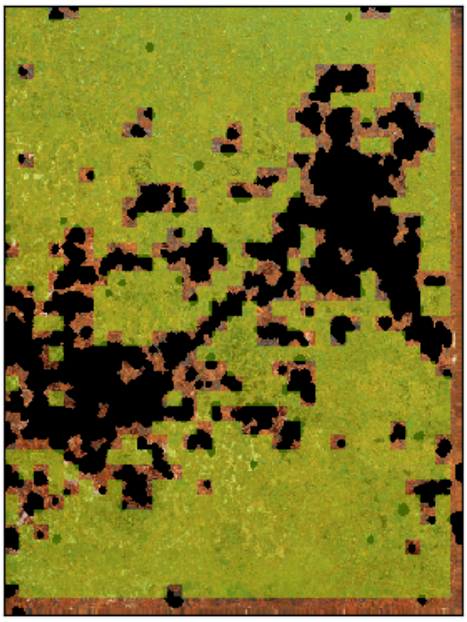
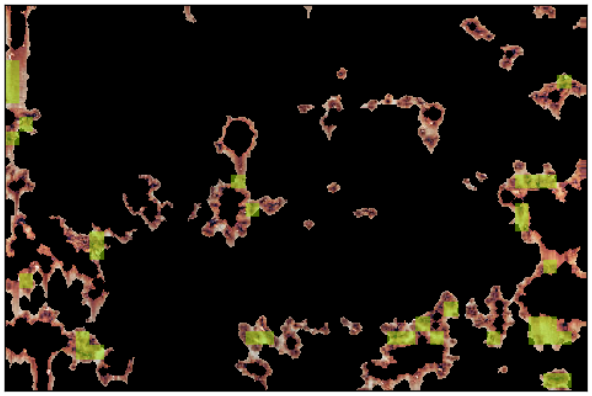
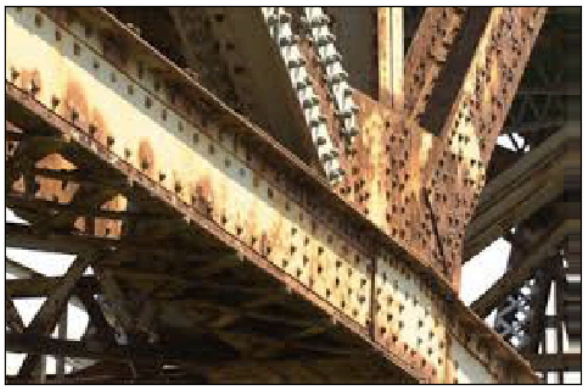


Fig 7: WCCD algorithm flowchart









References:

1. Francisco Bonnin-Pascual and Alberto Ortiz. Corrosion detection for Automated Visual Inspection.
2. Karen Simonyan & Andrew Zisserman. Very Deep Convolution Network for large Scale Image Recognition.
3. B.B.Zaidan, A.A.Zaidan, Hamdan.O.Alanazi, Rami Alnaqeib. Towards Corrosion Detection System.
4. Venkatasainath Bondada, Dilip Kumar Pratihar, Cheruvu Siva Kumar. Detection and quantitative assessment of corrosion on pipelines through image analysis.
5. http://web.pdx.edu/~jduh/courses/Archive/geog481w07/Students/Hayes_GreyScaleCoOccurrenceMatrix.pdf
6. https://neurohive.io/en/popular-networks/vgg16/
7. https://www.youtube.com/watch?v=iYsSmnma8jA&t=1575s