Semantic segmentation of corrosion in metal structures

Shreyansh Dwivedi
2101196

Advisor: **Dr. Moumita Roy**

Department of Computer Science and Engineering
Indian Institute of Information Technology Guwahati

IIIT Guwahati April 2024

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Shreyansh
Dwivedi
April 2024

Table of contents

Motivation

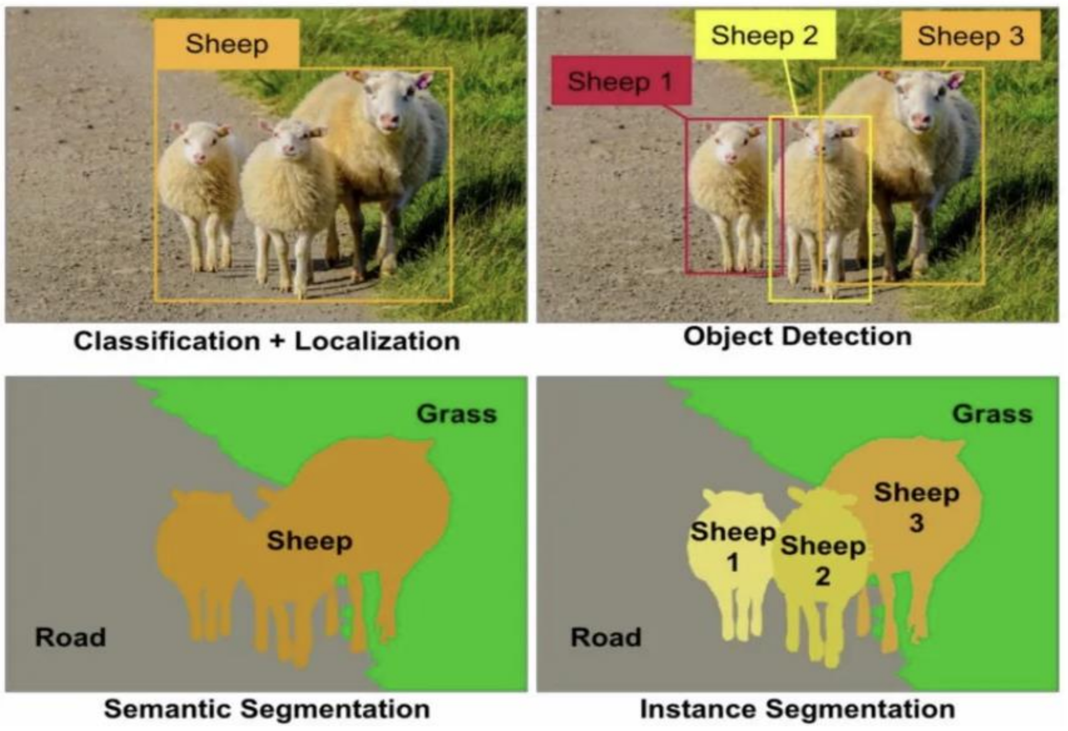
Corrosion in metal structures is a significant problem in India due to several factors:

1.Infrastructure Age: Much of India's infrastructure, including bridges, pipelines, and buildings, is aging. Older structures are more susceptible to corrosion due to factors such as poor maintenance practices and outdated materials.

2.Climate: India's varied climate, including -high humidity in coastal areas and monsoon rains, accelerates corrosion rates.

3.Industrial Pollution: Many parts of India experience high levels of industrial pollution, which can lead to increased corrosion rates, especially in urban and industrialized areas.

Why Semantic Segmentation?



Fine-grained Localization: Semantic segmentation provides pixel-level classification, enabling precise localization of corrosion areas within an image. This fine-grained detail allows for more accurate assessment of the extent and severity of corrosion compared to bounding box detection, which only provides a rough estimate of the affected area.

Research Papers And Literature Summary

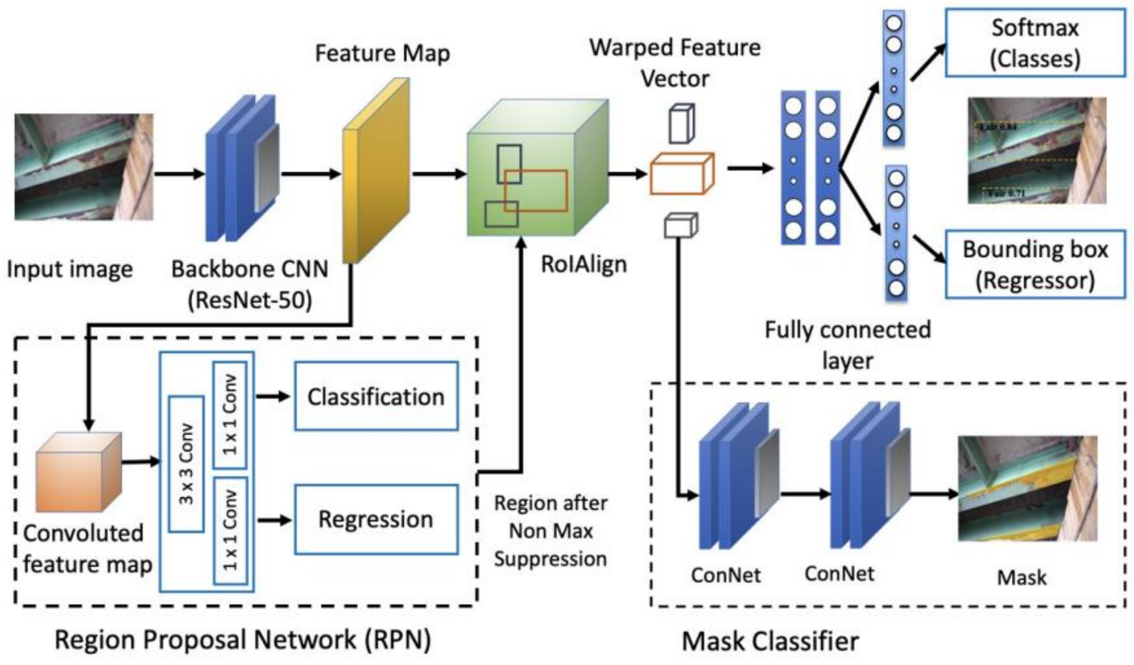
1. R Mitra - A Comparative Analysis of Two Deep Learning Neural Networks for Defect Detection in Steel Structures Using UAS

There database contains of 8,552 images, which 6,629 are equally split in images with and without corrosion. InceptionResNetV2 and ResNet152V2 models were used in transfer learning mode. Model was trained for 50 epochs are an F1 core of 0.89 and 0.90 was achieved for InceptionResNetV2 and ResNet152V2 respectively.

2. Zahra Ameli , Shabnam Jafarpoor Nesheli and Eric N. Landis - Deep Learning-Based Steel Bridge Corrosion Segmentation and Condition Rating Using Mask RCNN and YOLOv8 – 2023

There database contains 514 images with various corrosion severity levels, gathered from a variety of steel bridges. The “Fair”, “Poor”, and “Severe” classes in the annotated images are shown with green, yellow, and red colors. They trained Yolov8 and mask RCNN for 250 epochs 200 epochs respectively. The trained Mask RCNN and YOLOv8 models achieved mAP50 values of 0.674 and 0.726 on the test images(94).

Below Architecture was used in this paper



Feature Maps
Feature maps are typically generated using a process called convolution, where a filter is applied to a small region of the input image, called a receptive field. The filter is then moved across the entire input image, generating a feature map for each position. This process is repeated for each filter in the CNN, resulting in multiple feature maps that represent different learned features of the input image. Each filter is designed to detect a specific type of feature, such as edges, corners, or textures. The output of each filter is a feature map,

which is then used as input to the next layer of the CNN.

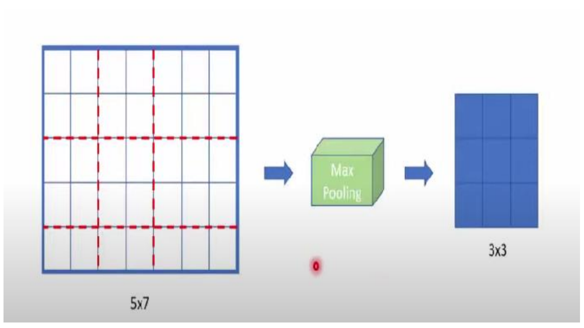
Region Proposal Network

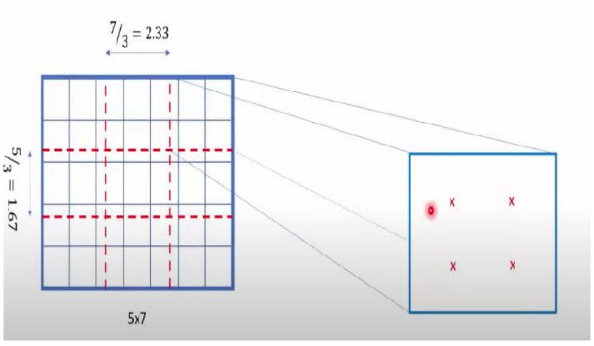
1. Purpose of RPN:
    1. The RPN is a critical component in modern object detection frameworks like Mask R-CNN.
    2. Its primary task is to generate candidate bounding boxes (proposals) that potentially contain objects of interest.
    3. These proposals serve as input for subsequent classification and regression steps.

2. The RPN is a fully convolutional network that operates on feature maps extracted from a base network (usually a CNN).
    1. It combines localization and classification tasks to predict the following for each anchor (a predefined bounding box centered at various positions):
        1. Objectness Score: The probability of an

anchor being foreground (contains an object) or background (does not contain an object).

2. Bounding Box Regression: Refines the coordinates of the anchor to better fit the object.





Fully Connected Layers

Fully Connected Layers Transforms the height and width of intermediate layer back to the size of input image through the transpose convolution operation so that predictions have one to one correspondence with input image in spatial dimensions. So features are upsampled by doing so, it creates a **pixel-wise segmentation mask** for each object class within the bounding boxes.

3. Image Segmentation Using Deep Learning: A Survey
Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulo

Grouped deep learning-based Segmentation model into following categories:

1) Fully convolutional networks
2) Convolutional models with graphical models
3) Encoder-decoder based models
4) Multi-scale and pyramid network based models
5) R-CNN based models (for instance segmentation)
6) Dilated convolutional models and DeepLab family
7) Recurrent neural network based models
8) Attention-based models
9) Generative models and adversarial training
10) Convolutional models with active contour models
11) Other models

For this project, Model number 3 (Encoder-decoder based model) was utilized to implement the baseline model, while Model number 8 (Attention-Based Model) was employed to propose a new architecture

.4. Reza et. al. Loss Functions in the Era of Semantic Segmentation: A Survey and Outlook

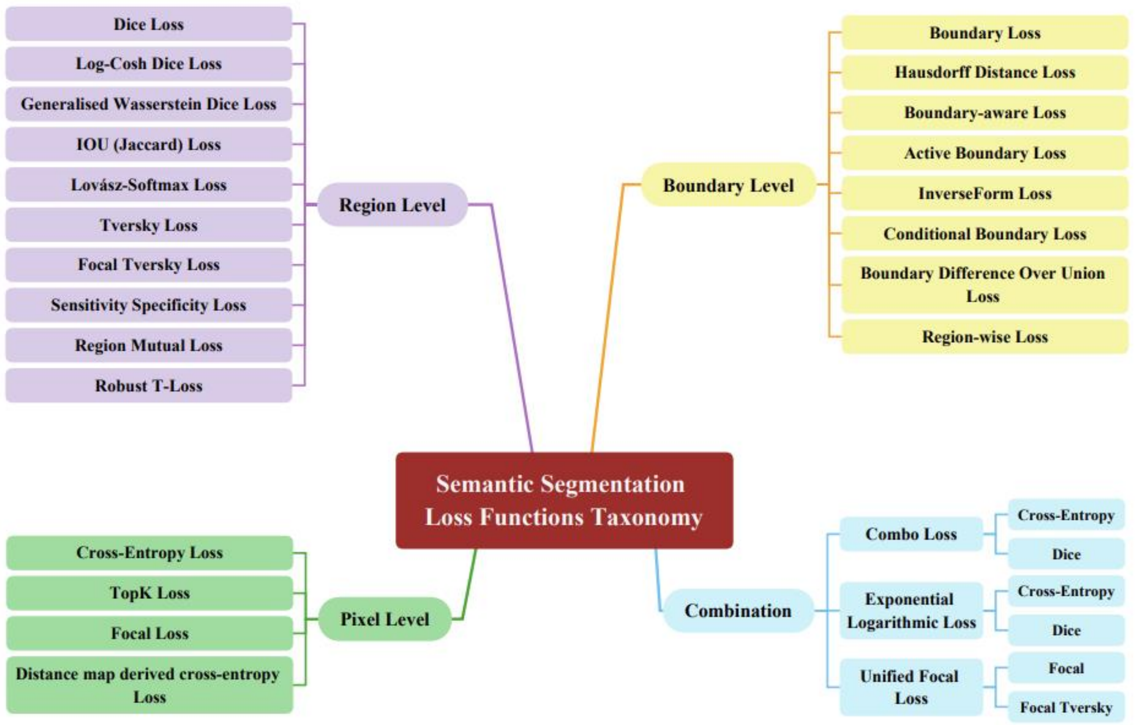
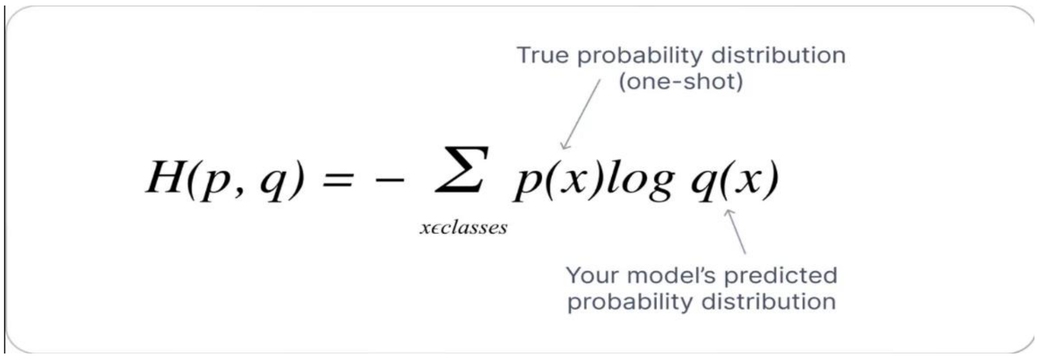


Figure 1: The taxonomy subsections delineate four distinct groups: (1) Region-Level, (2) Boundary-Level, (3) Pixel-Level, and (4) Combination.

Cross-Entropy Loss - Cross-entropy (CE) measures the difference between two probability distributions for a given random variable. In segmentation tasks, the cross-entropy loss is used to measure how well the model's predictions match the target labels. Using the softmax function, the model generates pixel-wise probability maps representing the likelihood of each pixel belonging to each class. The cross-entropy loss is then calculated by taking the negative logarithm of the predicted probability for the target class at each pixel.



$$H(p, q) = -\sum_{x \epsilon classes} p(x) log\, q(x)$$

True probability distribution (one-shot)
Your model's predicted probability distribution

Dice Loss The Dice loss originates from the Dice Coefficient which is a measure of similarity between two sets of data. It is commonly used in image segmentation to evaluate the overlap between a predicted segmentation mask and the target segmentation mask. It is defined as the size of the intersection of the predicted segmentation mask and the ground truth segmentation mask, divided by their sum. It is calculated separately for each class and average is reported.

$$\text{Dice Coefficient} = \frac{2|Y \cap T|}{|Y| + |T|}$$

where Y is the binary segmentation prediction mask and T is the binary segmentation target mask for a single class.

5. Vasawani et. al. Attention is All you Need

Attention Mechanism: The paper proposes a new mechanism called "self-attention" or "scaled dot-product attention." This mechanism allows the model to weigh the importance of different parts of the input sequence when processing each element. Unlike traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs), which process sequences sequentially or hierarchically, self-attention enables the model to consider all elements of the sequence simultaneously.

Transformer Architecture: The authors introduce the Transformer architecture, which is based entirely on self-attention mechanisms and feed-forward neural networks. The model consists of an encoder and a decoder, both composed of multiple layers of self-attention and feed-forward neural network modules. This architecture allows the model to capture dependencies between elements in the input and output sequences efficiently.

Positional Encoding: Since the Transformer model does not inherently understand the order of elements in a sequence, positional encoding is introduced to provide the model with information about the position of each element in the sequence. Positional encoding is added to the input embeddings before feeding them into the model, allowing the model to take into account the sequential order of elements.

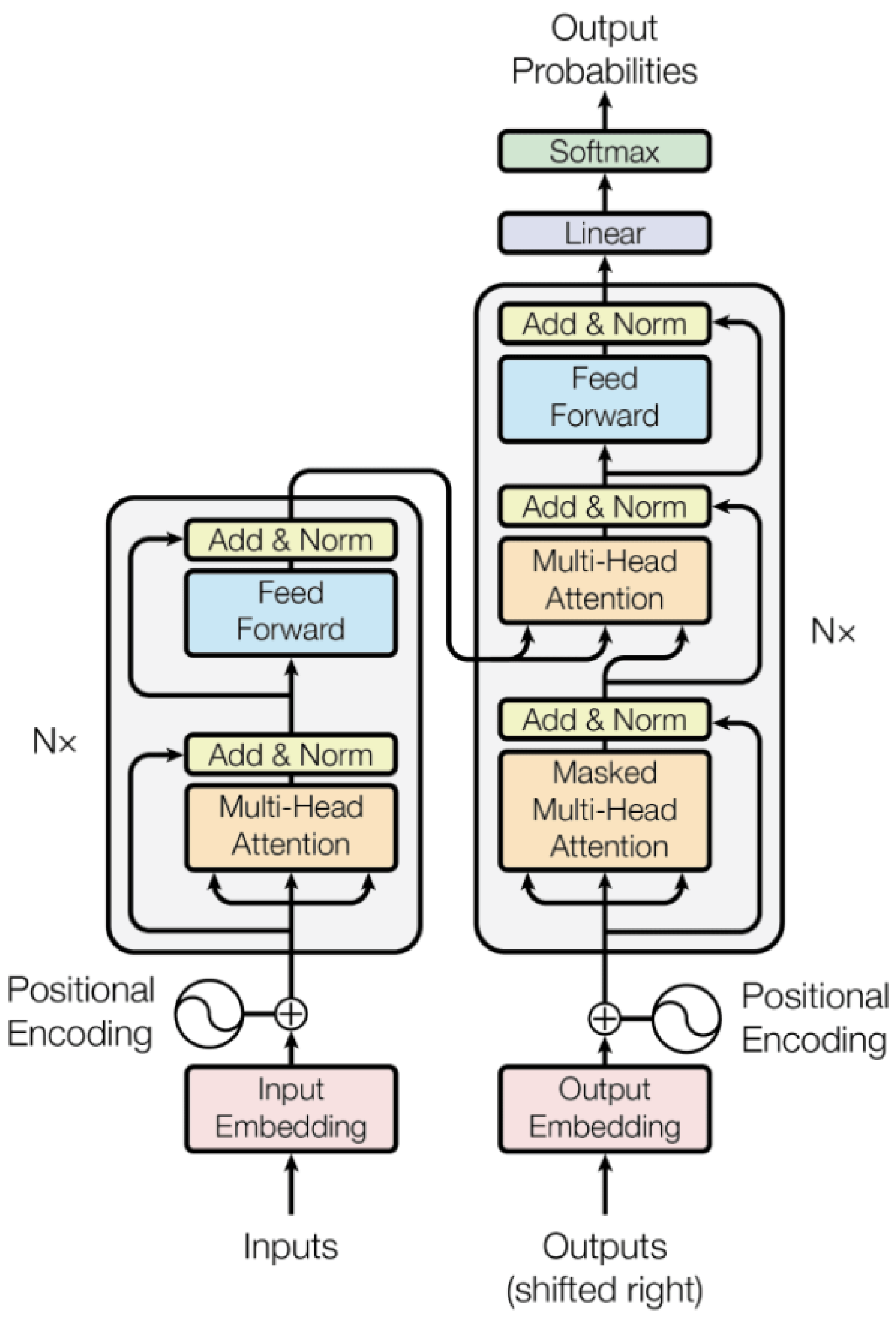


Image provided by the authors of the paper

6 Dosovitskiy et al An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

Patch Representation: Images are divided into smaller square regions called patches.

Analogous to Words: Each patch is analogous to a "word" in natural language processing.

Sequence Representation: The patches form a sequence, where each patch represents a unit of visual information.

Positional Encoding – As Positional Information gets lost while patching, all patches are given a positional encoding.

Patch Size: The paper specifically mentions using patches of size 16x16 pixels, although other sizes can also be used.

Transformer Model Application: The Transformer model can be applied to process these sequences of patches, leveraging its self-attention mechanism.

Image Processing Tasks: Treating images as sequences of patches enables the Transformer model to perform various computer vision tasks such as image classification, object detection, and image generation.

Advantages: This approach allows the model to capture long-range dependencies and global context within images, potentially leading to improved performance on certain tasks compared to traditional convolutional neural networks.

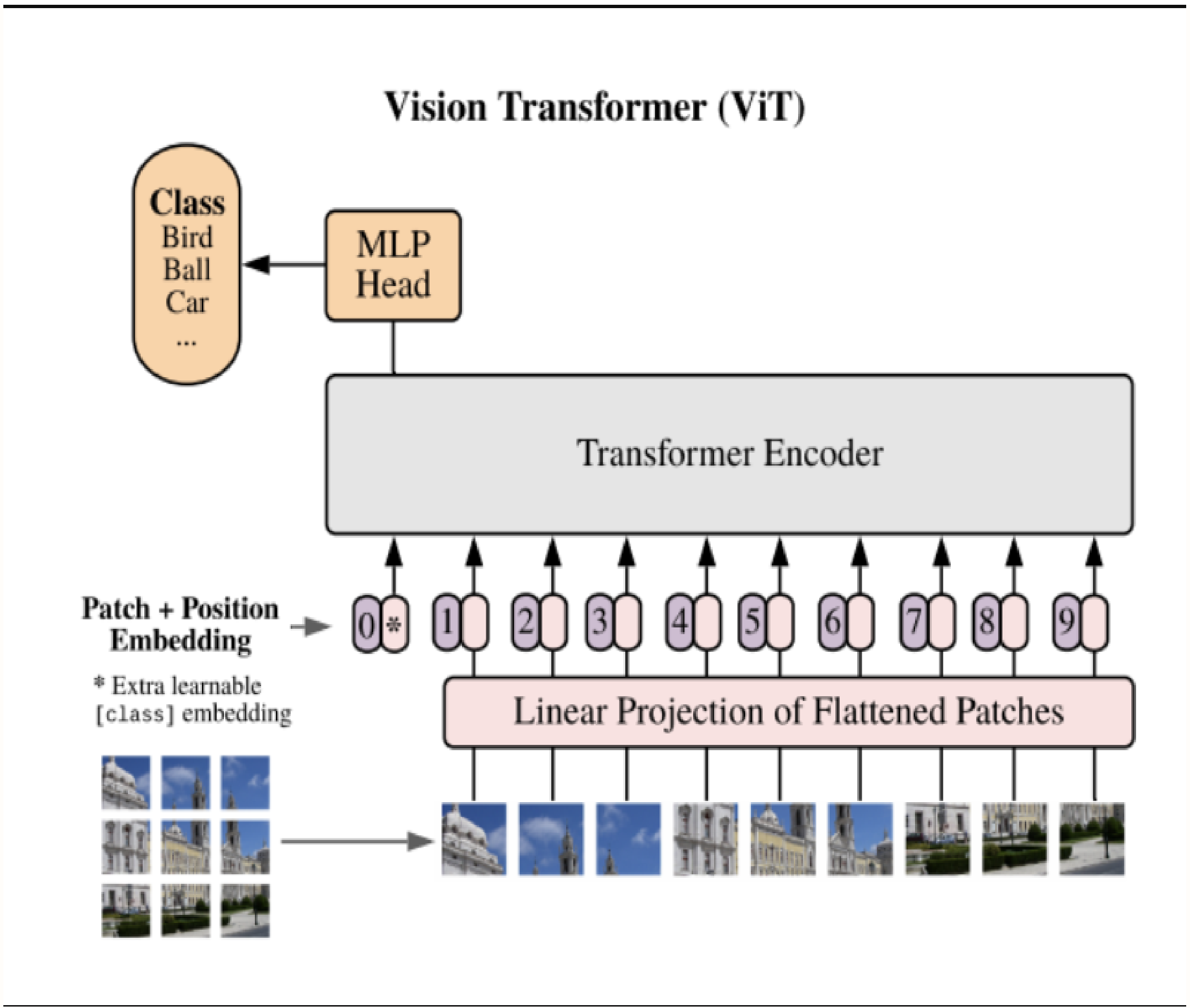

Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

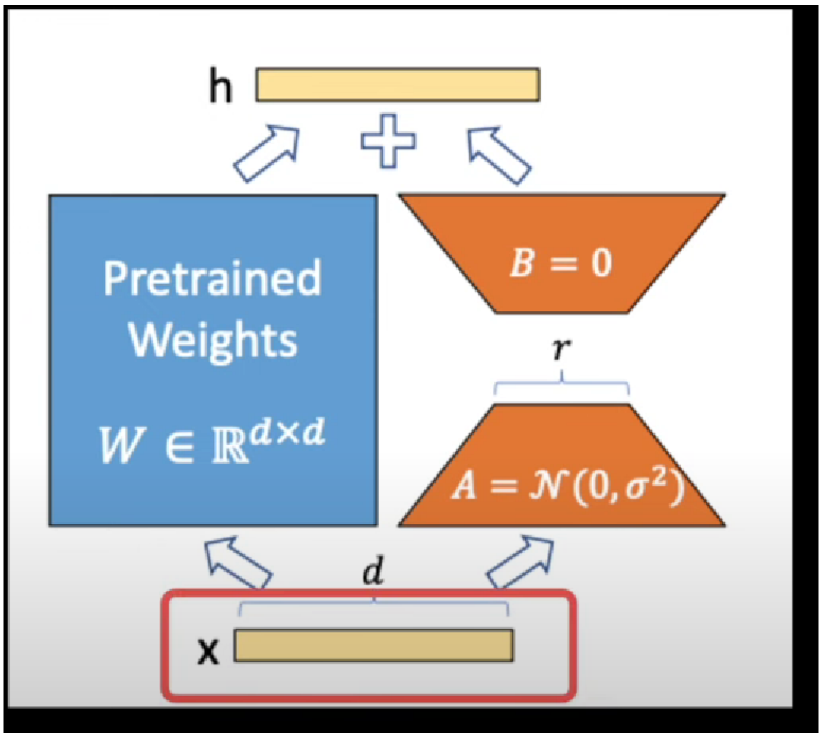
Image provided by the authors of the paper

7 **Hu et. al.** LoRA: Low-Rank Adaptation of Large Language Models

1)- The paper proposes Low-Rank Adaptation (LoRA), a method that freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture. This greatly reduces the number of trainable parameters for downstream tasks.

2)- The paper also provides an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA

3)- Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times

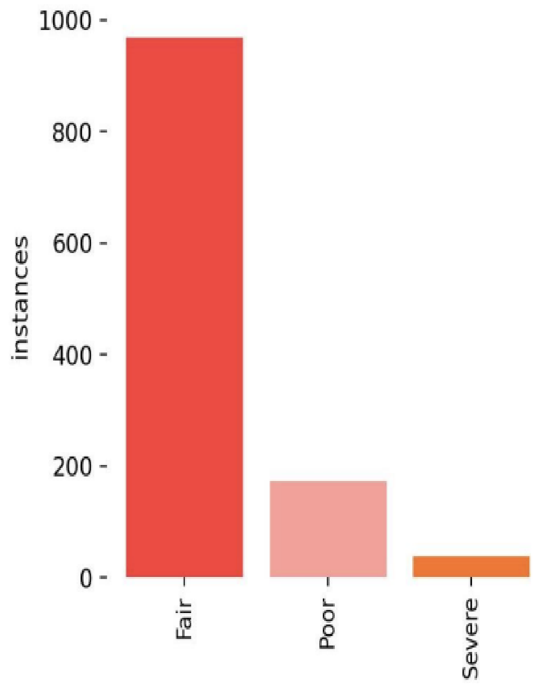

h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Data Collection And Analysis

Dataset 1 – Dataset from the authors of Paper on - Deep Learning-Based Steel Bridge Corrosion Segmentation and Condition Rating Using Mask RCNN and YOLOv8.

Dataset 2 – Manual Annotation along with using a model for prediction. It contains a total of 763 images belonging to three different classes “Fair”, “Poor” and “Severe”.

Distribution of Data in Dataset 1



Distribution of Data in Dataset 2

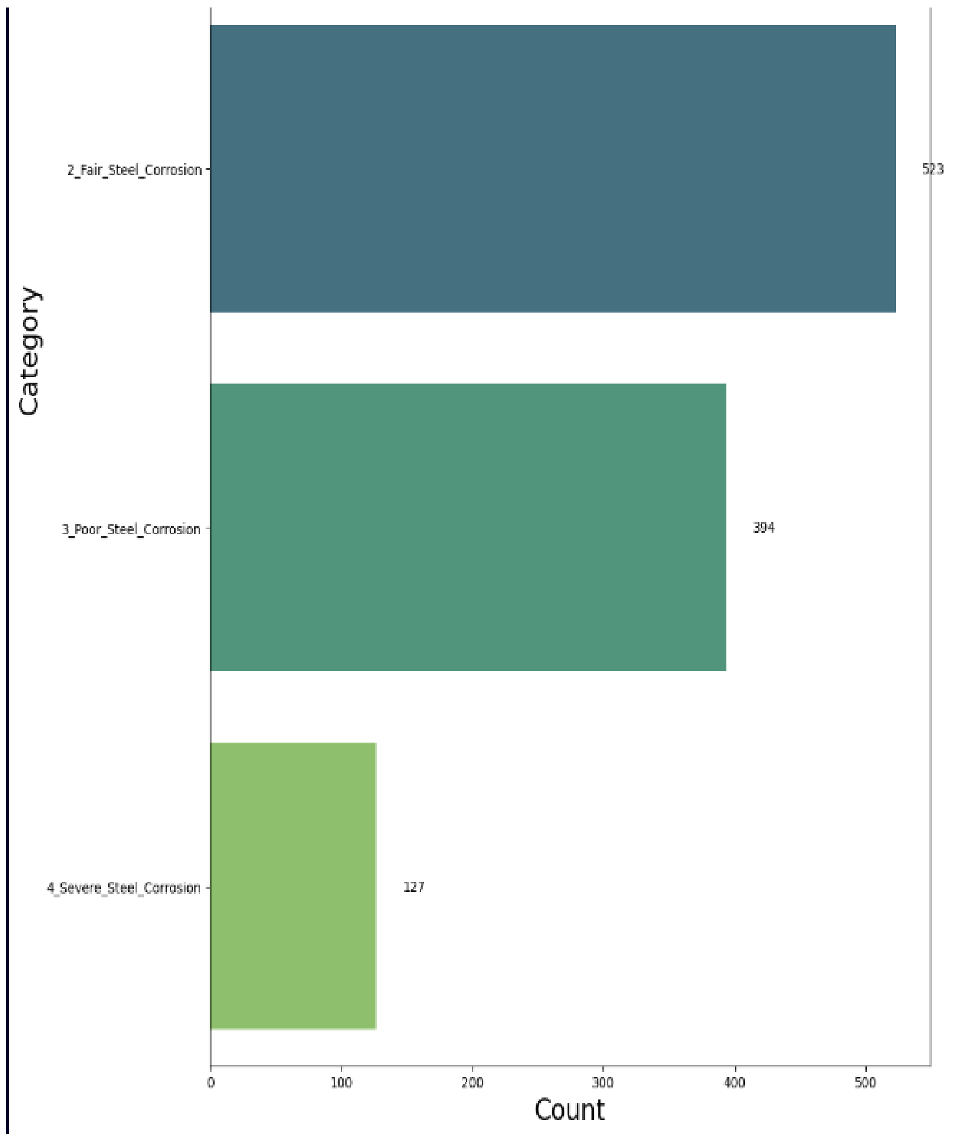
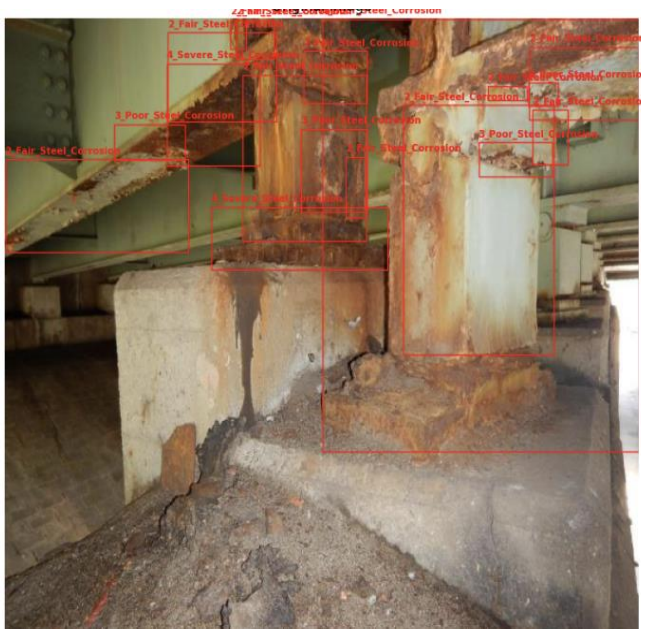
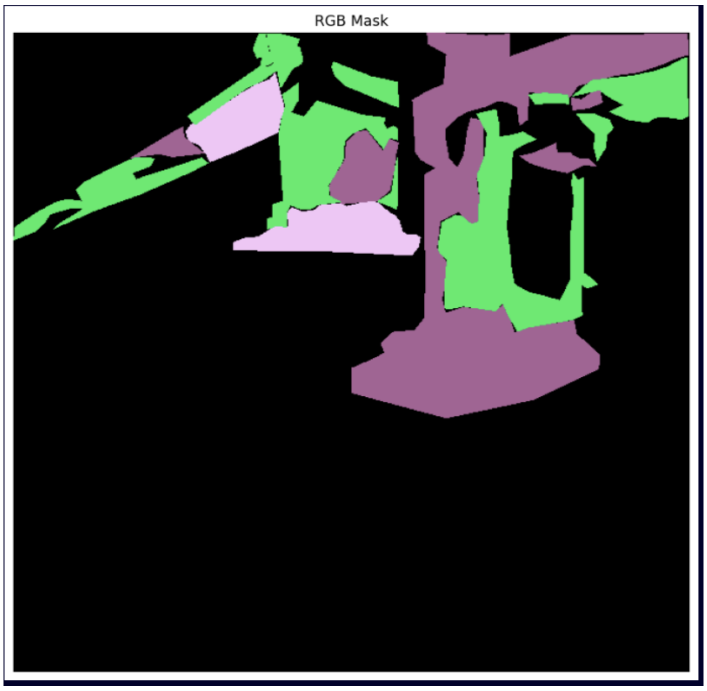
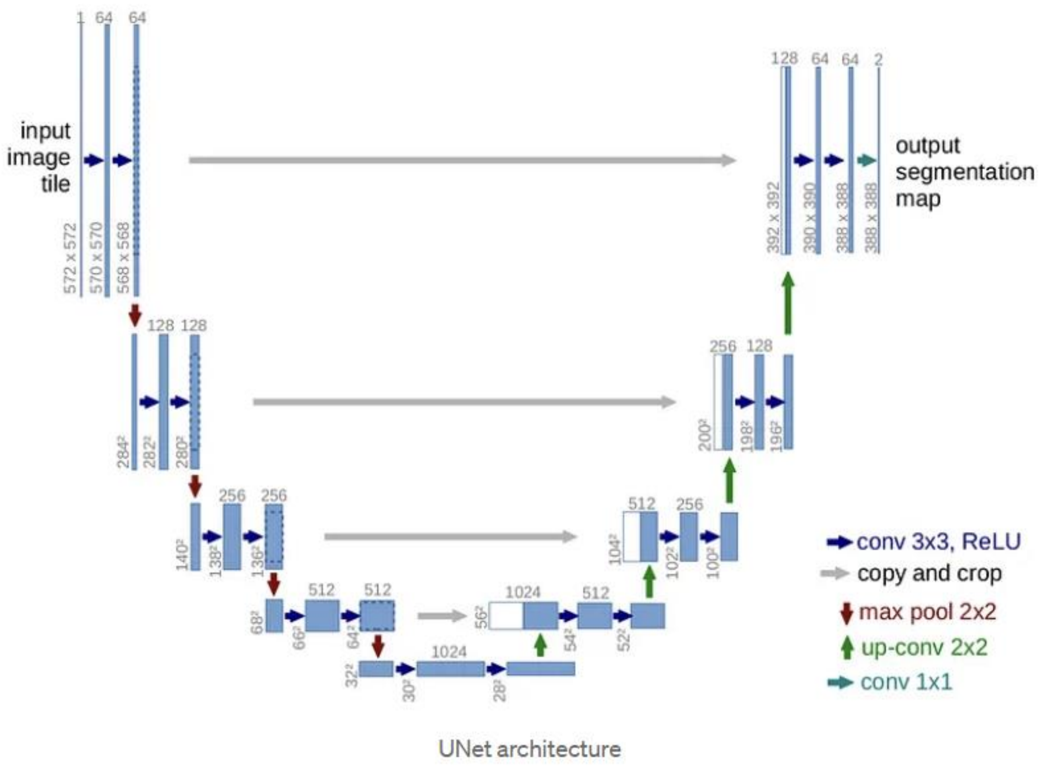


Image and Its Corresponding mask in Dataset





Baseline Architecture And Results



Encoder-Decoder Architecture: U-Net follows a CNN(Convolutional Neural Network) based encoder-decoder architecture. The encoder part consists of convolutional layers that progressively downsample the input image to extract high-level features. The decoder part consists of upsampling layers that gradually recover the spatial resolution of the features.

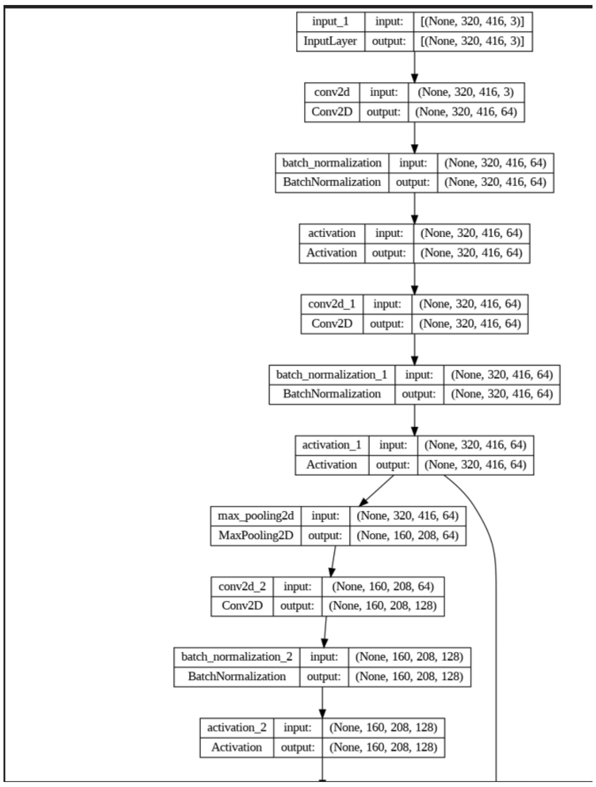
Skip Connections: One of the distinctive features of U-Net is the extensive use of skip connections. These connections pass feature maps from the encoder to corresponding layers in the decoder. They help in preserving spatial information and enable the model to localize objects more accurately.

Contracting Path: The encoder part of U-Net is often referred to as the contracting path. It typically consists of convolutional layers followed by max-pooling layers, which reduce the spatial dimensions of the feature maps while increasing their depth.

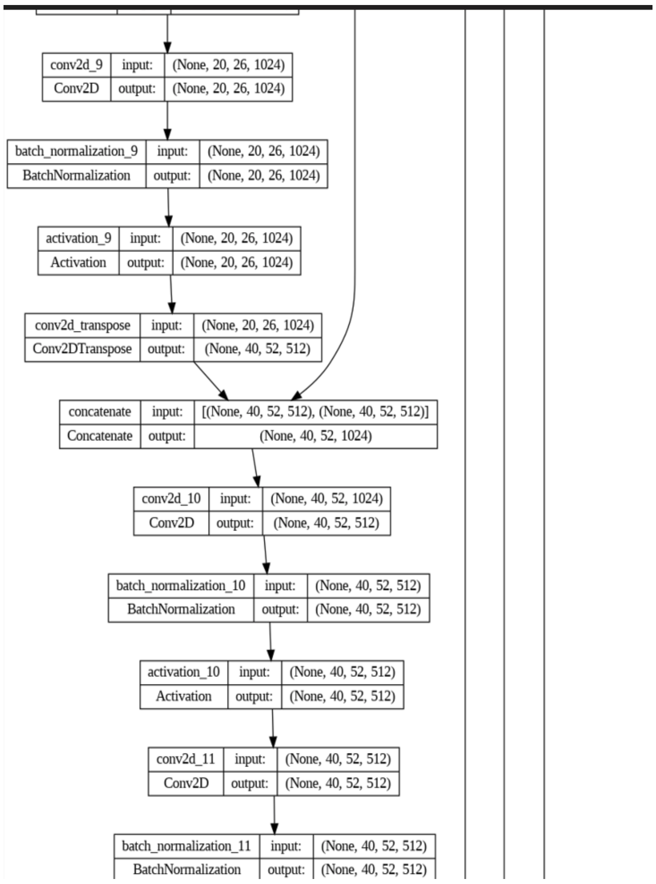
Expanding Path: The decoder part of U-Net is known as the expanding path. It consists of upsampling layers followed by convolutional layers. Upsampling layers increase the spatial resolution of the feature maps, while convolutional layers help refine the segmentation output.

Final Layer: The final layer of U-Net typically consists of a convolutional layer with a softmax activation function. This layer produces a probability map for each pixel in the input image, indicating the likelihood of each pixel belonging to the target class.

Implement Architecture of Unet

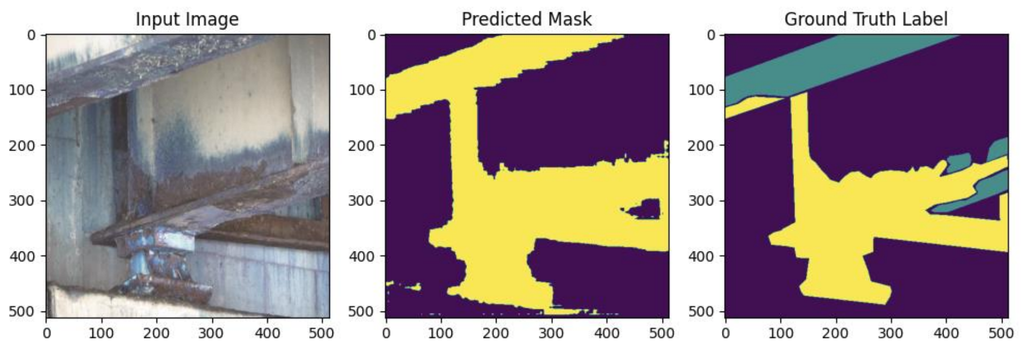


input_1 | input: | [(None, 320, 416, 3)]
InputLayer | output: | [(None, 320, 416, 3)]
conv2d | input: | (None, 320, 416, 3)
Conv2D | output: | (None, 320, 416, 64)
batch_normalization | input: | (None, 320, 416, 64)
BatchNormalization | output: | (None, 320, 416, 64)
activation | input: | (None, 320, 416, 64)
Activation | output: | (None, 320, 416, 64)
conv2d_1 | input: | (None, 320, 416, 64)
Conv2D | output: | (None, 320, 416, 64)
batch_normalization_1 | input: | (None, 320, 416, 64)
BatchNormalization | output: | (None, 320, 416, 64)
activation_1 | input: | (None, 320, 416, 64)
Activation | output: | (None, 320, 416, 64)
max_pooling2d | input: | (None, 320, 416, 64)
MaxPooling2D | output: | (None, 160, 208, 64)
conv2d_2 | input: | (None, 160, 208, 64)
Conv2D | output: | (None, 160, 208, 128)
batch_normalization_2 | input: | (None, 160, 208, 128)
BatchNormalization | output: | (None, 160, 208, 128)
activation_2 | input: | (None, 160, 208, 128)
Activation | output: | (None, 160, 208, 128)



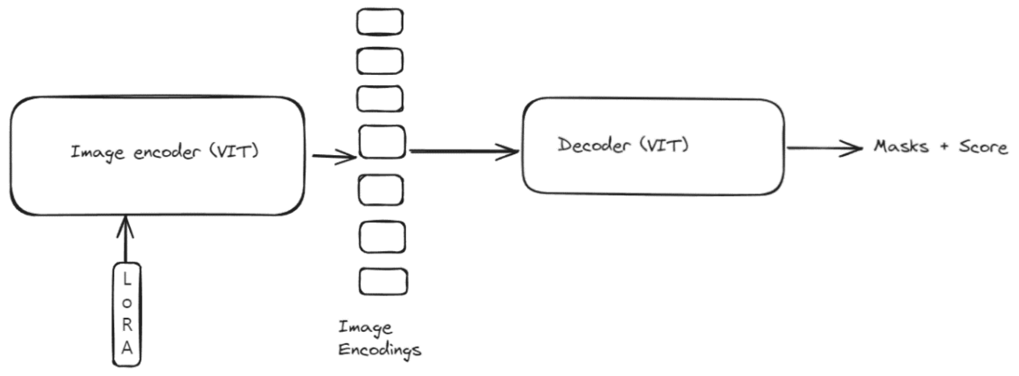
conv2d_9 | input: | (None, 20, 26, 1024)
Conv2D | output: | (None, 20, 26, 1024)
batch_normalization_9 | input: | (None, 20, 26, 1024)
BatchNormalization | output: | (None, 20, 26, 1024)
activation_9 | input: | (None, 20, 26, 1024)
Activation | output: | (None, 20, 26, 1024)
conv2d_transpose | input: | (None, 20, 26, 1024)
Conv2DTranspose | output: | (None, 40, 52, 512)
concatenate | input: | [(None, 40, 52, 512), (None, 40, 52, 512)]
Concatenate | output: | (None, 40, 52, 1024)
conv2d_10 | input: | (None, 40, 52, 1024)
Conv2D | output: | (None, 40, 52, 512)
batch_normalization_10 | input: | (None, 40, 52, 512)
BatchNormalization | output: | (None, 40, 52, 512)
activation_10 | input: | (None, 40, 52, 512)
Activation | output: | (None, 40, 52, 512)
conv2d_11 | input: | (None, 40, 52, 512)
Conv2D | output: | (None, 40, 52, 512)
batch_normalization_11 | input: | (None, 40, 52, 512)
BatchNormalization | output: | (None, 40, 52, 512)

Results After just 30 epochs of Training



Average Accuracy = 0.51

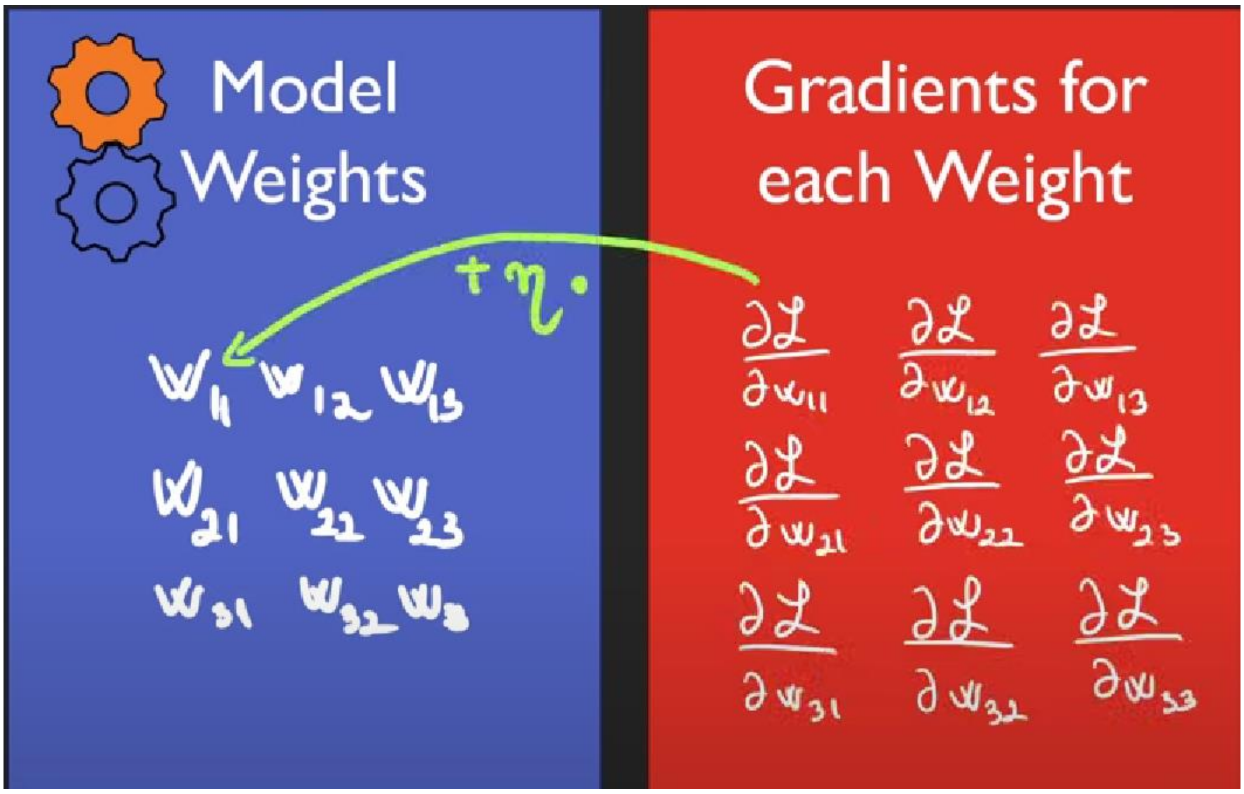
Proposed Architecture



The main idea is to replace the CNN based encoder and decoder in UNet with a ViT(Vision Transformer) based encoder decoder.

1. **Image Encoder** – Image is passed to an image encoder which is in itself is a pre-trained VIT-b (Vision Transformer). Low rank adaptation is applied on the layers of this Vision Transformer to reduce the number of parameters that require to be trained.

ViT-b has around 86m parametes. Fine Tuning them take a lot of space and time. Here LoRA comes into picture which breaks the weight matrix of ViT-b in comparatively smaller matrix using property of linear independence of rows or columns in a matrix.


Model Weights
Gradients for each Weight
$+\eta\cdot$
w_{11} w_{12} w_{13}
w_{21} w_{22} w_{23}
w_{31} w_{32} w_{33}
$\frac{\partial \mathcal{L}}{\partial w_{11}}$ $\frac{\partial \mathcal{L}}{\partial w_{12}}$ $\frac{\partial \mathcal{L}}{\partial w_{13}}$
$\frac{\partial \mathcal{L}}{\partial w_{21}}$ $\frac{\partial \mathcal{L}}{\partial w_{22}}$ $\frac{\partial \mathcal{L}}{\partial w_{23}}$
$\frac{\partial \mathcal{L}}{\partial w_{31}}$ $\frac{\partial \mathcal{L}}{\partial w_{32}}$ $\frac{\partial \mathcal{L}}{\partial w_{33}}$

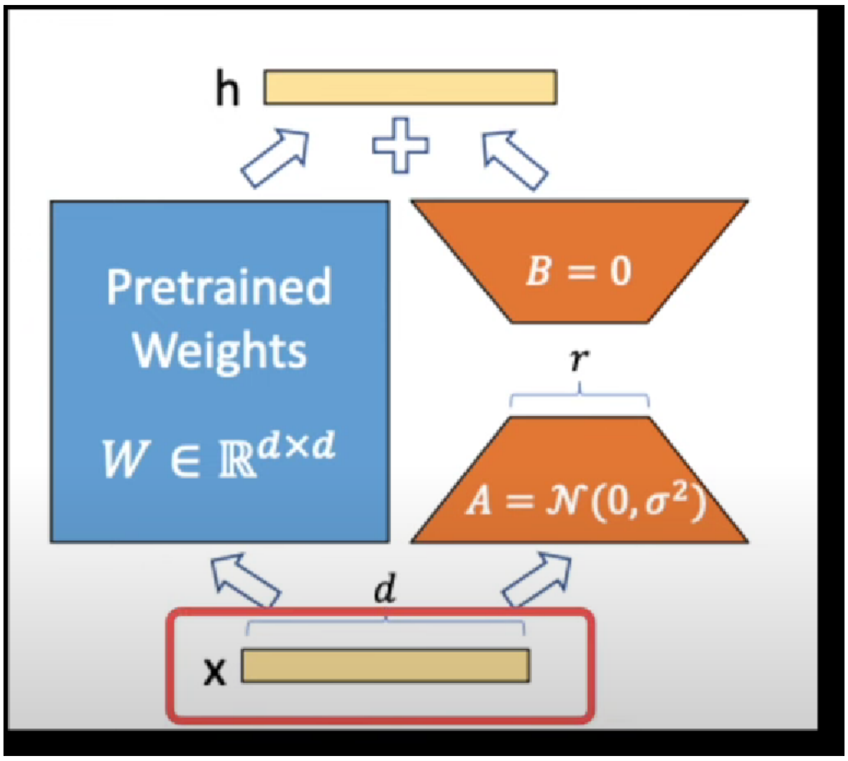
$$\underset{3\times 3}{\begin{bmatrix} 2 & 20 & 1 \\ 4 & 40 & 2 \\ 6 & 60 & 3 \end{bmatrix}} = \underset{3\times 1}{\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}} \times \underset{1\times 3}{[2 \quad 20 \quad 30]}$$

In the initial matrix, a single column, say column '3', comprising [1, 2, 3], can represent other two rows as well, just by multiplication. For instance, multiplying by 2 can represent the first row, and multiplying by 20 can represent the second row. This initial matrix has only one linearly independent column.

M = 500x500 (rank 2)	-	250000
A = 500X2	-	1000
B = 2 X 500	-	1000

Let M be a Matrix of dimension 500x500.Then it will have 250000 parameter. But if it is a rank 2 matrix(**number of linearly independent columns are 2**) then matrix M can be broken into matrix A and B which will have 1000 parameters each, much lesser than previous matrix.

So in a similar fashion weight matrix of ViT-b is broken down into two matrices of some rank, which is a hyperparameter to be tuned.


h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

2.Decoder - The input of the decoder is output produced by the ViT encoder. It is again, in itself is a ViT based decoder.A transpose convolution in used to generate mask and their final scores are also output

Loss Used:

The loss is calculated as the weighted sum of Cross-Entropy Loss and Dice loss which as pixel level and region level loss respectively.

Loss = a* (Cross-Entropy Loss) + b*(Dice Loss)

a and b are some constant between 0-1.

The implementation of these two losses are in accordance with those mentioned in pape Reza et. al. Loss Functions in the Era of Semantic Segmentation: A Survey and Outlook.

TRANING AND RESULTS:

The Model was trained for 150 epochs on both the datasets, For dataset 1 –> 450 images were put in train, 50 images in test and rest in valid. For dataset 2 –> 650 images were put in train, 50 in test and rest in valid. Results were almost similar for both the datasets.

Different ranks were experimented as

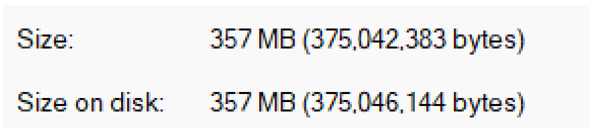
hyperparametes for LoRA and rank of 256 was found optimal.

Adam optimizer with learning rate 0.0001 was used.

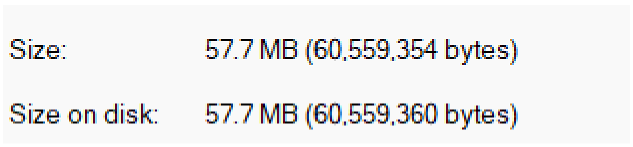
Final Losses obtained were -

1- **loss : 0.182638**
2- **loss_ce: 0.020557**
3- **loss_dice: 0.255457**

There was a significant improvement in space as a result of Using LoRA.



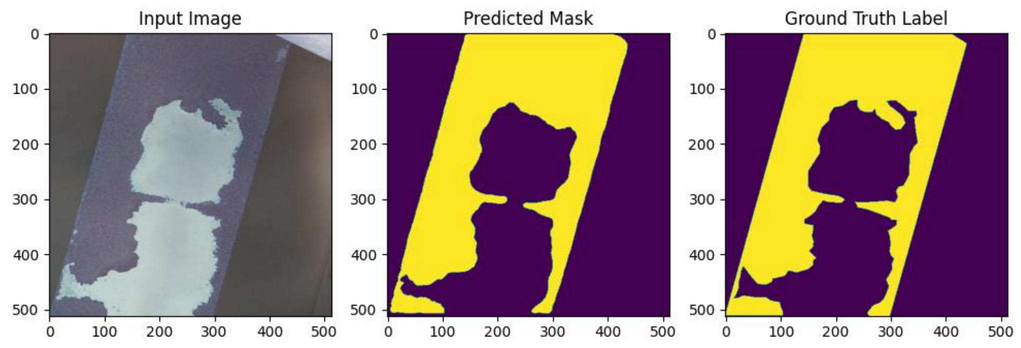
Size:	357 MB (375,042,383 bytes)
Size on disk:	357 MB (375,046,144 bytes)

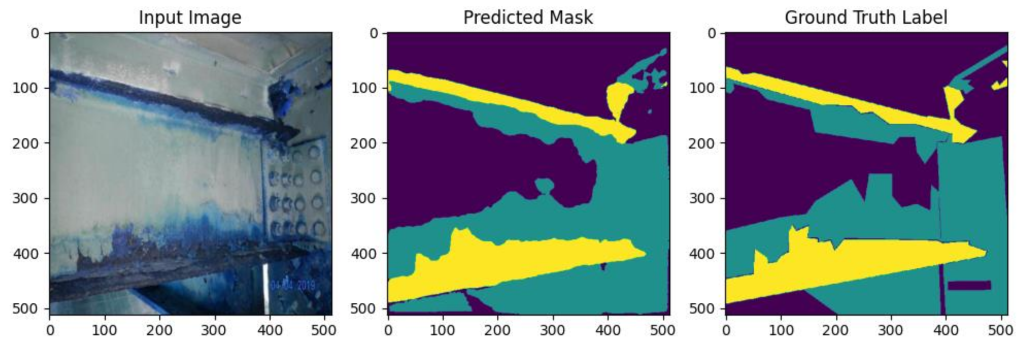


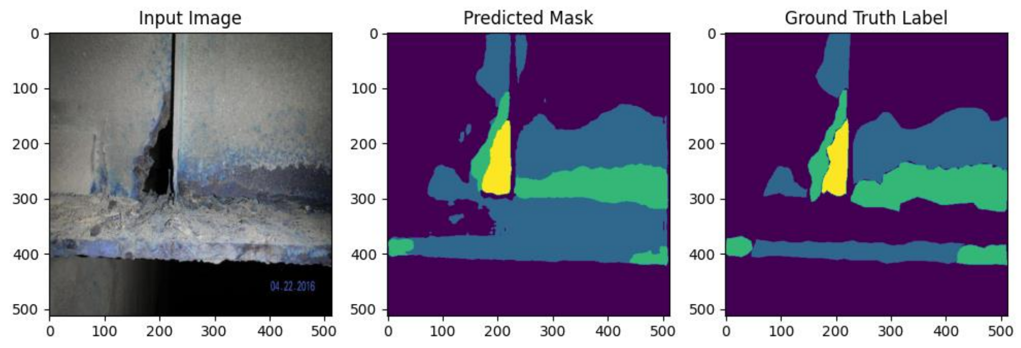
Size:	57.7 MB (60,559,354 bytes)
Size on disk:	57.7 MB (60,559,360 bytes)

Pre-trained parameters which were **357Mb** before were now only **57.7MB** thus reducing the space requirement by approximately **85%.**

Visual Results from the implementation


Input Image
Predicted Mask
Ground Truth Label
0
100
200
300
400
500


Input Image
Predicted Mask
Ground Truth Label
0
100
200
300
400
500


Input Image
Predicted Mask
Ground Truth Label
04.22.2016
0
100
200
300
400
500

Future Plans

1 – Collect more datasets to reduce disparity in different classes

2- Use bigger models like ViT-Huge ot ViT-Large to see if there is any significant improvements.

3-Use better Loss Techniques that penalize class imbalances

THANK YOU