Steel Defect Analysis

Advance Statistics – II Project Report

kirti khade (45733130) || Shresth Shetty ()

2019

Table of Contents

[Problem and significance 2](#_Toc23410591)

[Problem Statement 2](#_Toc23410592)

[Semantic Segmentation 2](#_Toc23410593)

[Significance 2](#_Toc23410594)

[Solution 3](#_Toc23410595)

[Understanding the data 3](#_Toc23410596)

[Data Available 3](#_Toc23410597)

[Data Analysis 3](#_Toc23410598)

[Class Imbalance 3](#_Toc23410599)

[Number of Defects per image 4](#_Toc23410600)

[Distribution of data 4](#_Toc23410601)

[Conclusion from Data Analysis 6](#_Toc23410602)

[Model Selection 6](#_Toc23410603)

[Types of models for semantic segmentation 6](#_Toc23410604)

[Number of operations required 6](#_Toc23410605)

[Power Consumption 7](#_Toc23410606)

[Number of Model Parameters 8](#_Toc23410607)

[Conclusion from Model Selection 8](#_Toc23410608)

[Description of Model 8](#_Toc23410609)

[UNET 8](#_Toc23410610)

[**VGG-16**:(Simonyan & Zisserman, 2015) 9](#_Toc23410611)

[**ResNet**: (He, Zhang, Ren, & Sun, 2015) 10](#_Toc23410612)

[Training Results 11](#_Toc23410613)

[Evaluation 11](#_Toc23410614)

[Loss Function 11](#_Toc23410615)

[Model Parameters 11](#_Toc23410616)

[UNET 12](#_Toc23410617)

[ResUNET 12](#_Toc23410618)

[VGG 14](#_Toc23410619)

[Other models tried 16](#_Toc23410620)

[Comparison 16](#_Toc23410621)

[Findings 16](#_Toc23410622)

[Limitation 17](#_Toc23410623)

[Conclusion 17](#_Toc23410624)

[Reference 18](#_Toc23410625)

# Problem and significance

## Problem Statement

The gross value of steel industry in USA is USD $500 billion with over 6 million people employed in this industry. Steel industry is often considered to be an indicator of economic progress, because steel plays a critical role in infrastructure and overall development of a country. Steel is also one of the worlds most recycled material with a recycling rate of 60% globally.

Like every industry, steel industries face quality issues, surface defects is one of the major issues faced by these industries. With the help of recent development in Machine Learning  & Artificial Intelligence technologies, our client Severstal is looking for an algorithm that would classify surface defects on steel sheet. They have recently created largest industry data lake with petabytes of data of steel images. We are accessing this data source from Kaggle(‘Severstal’, n.d.) where Severstal is hosting a competition. The classification required is of 4 levels, each more severe than the last. This algorithm when deployed, would increase the efficiency of production, reduce man-hours and will maintain high quality of steel production.

The given problem is a classification and image localization problem. The most popular algorithm used for image classification is convolutional Neural networks. Neural networks is an algorithm in machine learning that is used for classification and regression. The given problem is a branch of image classification known as semantic segmentation, where each pixel has a label of which class it belongs to. Below, we will be giving more details on Semantic Segmentation.

## Semantic Segmentation

Semantic Segmentation is the process of linking each pixel in a image to a class label. In our case, this is the method we will be using as it can classify each pixel into different classes. Semantic Segmentation is image classification at pixel level. For example, if an image has many cars, it will segment all the pixel with cars in one class. To label the cars, or to specify if the different cars, we need to use something called an instance segmentation – which is not our use case now.

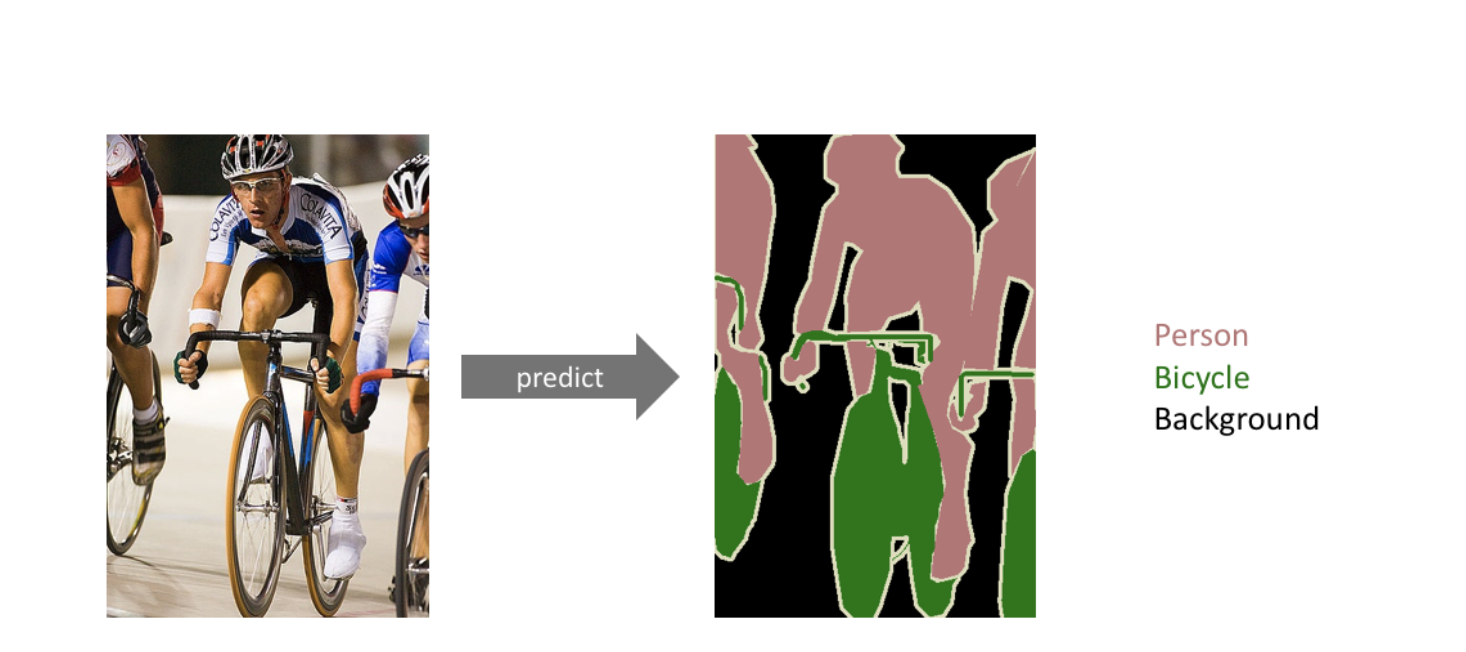


Figure 1 - Semantic Segmentation Prediction

## Significance

During the production of flat steel, there are many processes and machines involved which could cause a breakage like heating, rolling, drying as well as cutting of steel. If we successfully deploy a defect detection algorithm to localise and classify the defect in each sheet, it will help in increasing the efficiency of operation as well as ensuring a high-quality sheet.  This will also help to identify if any steel sheet can be used or processed again thus reducing the cost of operation manifolds

# Solution

## Understanding the data

Severstal are looking for cutting edge machine learning based solution to classify surface defects on steel sheet. Each image has 4 labels, each belonging to a defect. The model used for semantic segmentation must classify each of the pixels as one of the four classes. If a pixel doesn’t belong to any class, the model will just classify it as a zero. This dataset is hosted at Kaggle – which is an online community of data science and machine learning, this website hosts various challenges, Severstal has hosted its competitions here as well.

## Data Available

The company created a largest industrial data lake, with petabytes of data that were previously discarded. For each image, they have provided a mask of the image, class of this mask. Additionally, an image could have more than one mask.

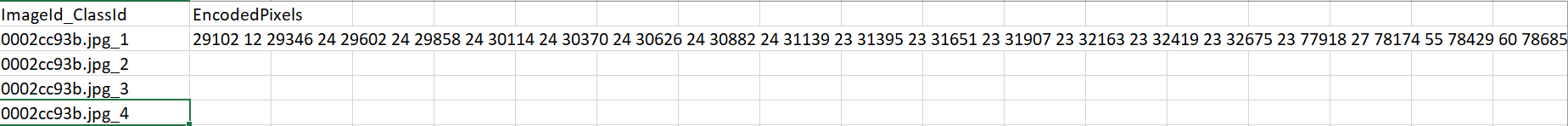
The labels are encoded in a manner called as run length encoding. In this, for each image, for each label, the start pixel and the run length of that label is given (Starting from left to right, then top to bottom). An example of run length encoding is given below:

Figure 2 – Encoded Data Sample

For test data, there were no labels given by Kaggle. For evaluation, we divided the dataset into training and validation set.

## Data Analysis

To further understand the data, we did a few Exploratory data analysis.

### Class Imbalance

***Code Snippet - SDA\_Analysing\_data.ipynb***

Machine learning algorithms work the best when the number of instances for each class is roughly the same. This helps the algorithm to not get biased against one class and treat each class equally. However, most real-world problems display some level of class-imbalance; the classes are not represented equally in the dataset.

Upon plotting the class distribution of this dataset, it is observed that defect type 3 is disproportionately high whereas the other classes have low representation in the dataset.

A screenshot of a cell phone

Description automatically generated

Figure 3 – Number of Images for each class

### Number of Defects per image

***Code Snippet - SDA\_Analysing\_data.ipynb***

### 

Since there were images with more than one mask, we wanted to understand the distribution of defects per image. This is what we found:

A screenshot of a cell phone

Description automatically generated

Figure 4 – Number of classes in each image

### Distribution of data

***Code Snippet - SDA\_Analysing\_data\_t\_SNE.ipynb***

As the images looked pretty much same to naked human eyes - we decided to see the distribution of the images using a visualization technique called t-distributed Stochastic Neighbour Embedding (t-SNE)

t-Distributed Stochastic Neighbour Embedding (t-SNE) is a technique for dimensionality reduction which is particularly well suited for the visualization of high-dimensional datasets.The algorithms starts by calculating the probability of similarity of points in high-dimensional space and calculating the probability of similarity of points in the corresponding low-dimensional space. The similarity of points is calculated as the conditional probability that a point A would choose point B as its neighbour if neighbours were picked in proportion to their probability density under a Gaussian (normal distribution) cantered at A.

For our use case, we first reduced the size of the image using PCA and then plotted the data using t-SNE package in python. From this image, we can conclude that, data is cultured, there are no specific dimension with a heavy presence of class 3, hence we need to rely on deep learning techniques to solve this non-linear data.

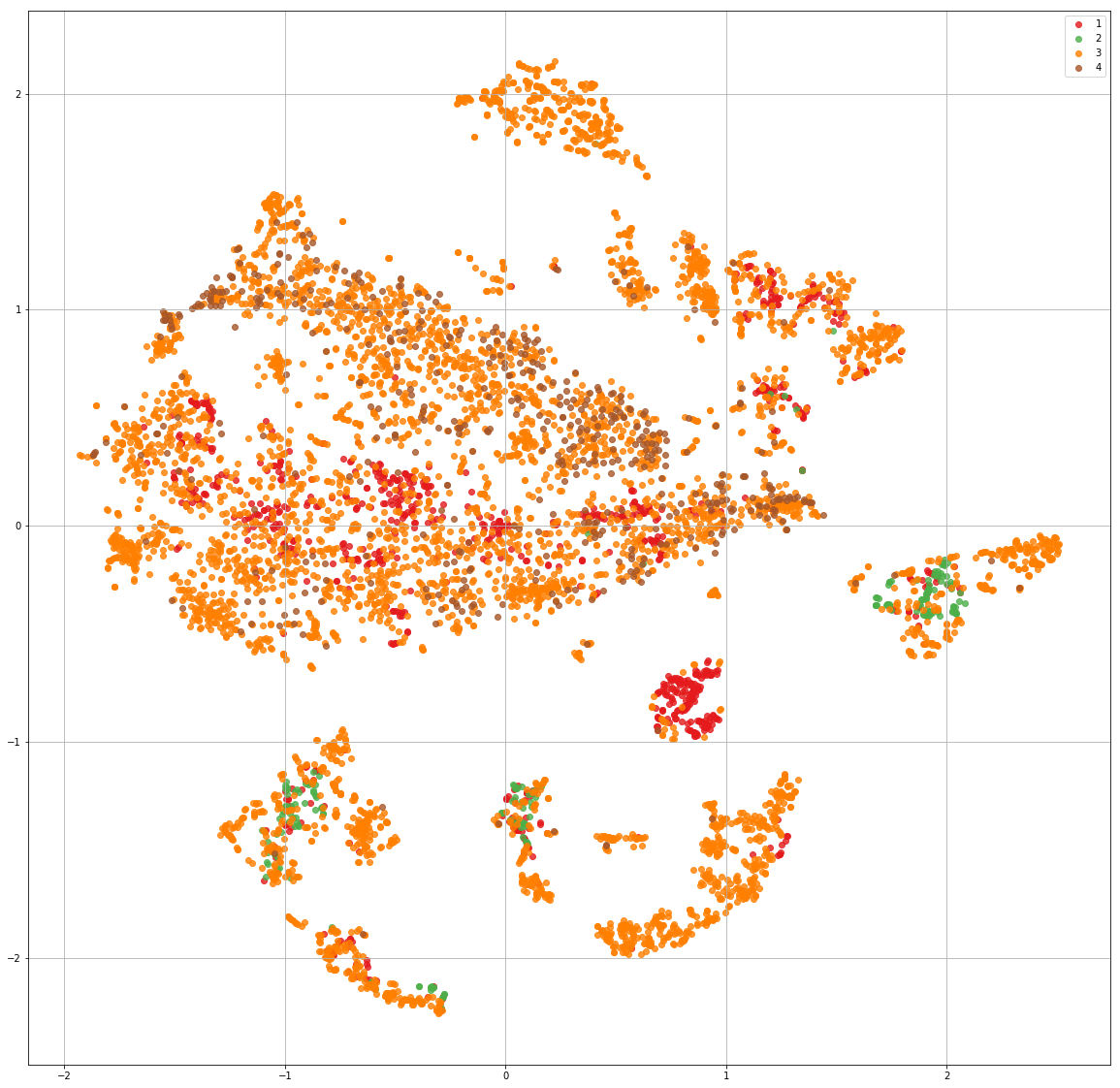


Figure 5 – t-SNE distribution of data

### Conclusion from Data Analysis

In conclusion, we can say:

* Need to treat the data for class imbalance
* Need to predict for multiple classes
* Need to use Deep learning algorithms

## Model Selection

There is various option available in terms of model architecture for semantic segmentation. For choosing algorithms that will give good result for our particular case, we considered the below parameters.

### Types of models for semantic segmentation

Top 1 accuracy % for single-model architectures, can be seen below. We can see that the top performing algorithms are – Resnet, VGG and Inception – v3

A screenshot of a cell phone

Description automatically generated

Figure 7 – Source - [Link](https://www.semanticscholar.org/paper/An-Analysis-of-Deep-Neural-Network-Models-for-Canziani-Paszke/9a786d1ecf77dfba3459a83cd3fa0f1781bbcba4)

### Number of operations required

Top 1 accuracy % versus amount of operations required for a single forward pass for various algorithms can be seen below. The sißze of the blobs is proportional to the number of network parameters. As we can see, among ResNet, Inception-v3 and VGG, VGG requires the maximum number of operations. Inception & Resnet are require relatively smaller number of operations.

A screenshot of a cell phone

Description automatically generated

Figure 8 – Source - [Link](https://www.semanticscholar.org/paper/An-Analysis-of-Deep-Neural-Network-Models-for-Canziani-Paszke/9a786d1ecf77dfba3459a83cd3fa0f1781bbcba4)

### Power Consumption

Net power consumption (due only to the forward processing of several DNNs) for different batch sizes can be seen below. The idle power of the TX1 board, with no HDMI screen connected, was 1.30W on average. Resnet has high power consumption, while VGG and Inception have lower power consumptions.

A close up of a map

Description automatically generated

Figure 9 – Source - [Link](https://www.semanticscholar.org/paper/An-Analysis-of-Deep-Neural-Network-Models-for-Canziani-Paszke/9a786d1ecf77dfba3459a83cd3fa0f1781bbcba4)

### Number of Model Parameters

From the below table we can

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Trainable** | **Non- Trainable** | **Total Parameters** |
| Xception\* | 20,815,148 | 54,528 | 20,869,676 |
| VGG | 14,716,740 | 0 | 14,716,740 |
| Inception V3 | 21,776,548 | 34,432 | 21,810,980 |
| UNET | 6,287,828 | 7,296 | 6,287,828 |
| ResUNET | 1,944,100 | 2,944 | 1,947,044 |

\* added xception to comparison, as it is better then Inception V3

### Conclusion from Model Selection

* Xception and Inception V3 have a sizeable parameter, and although these models give good results, it’s very difficult to find computing resources that large.
* UNET, ResUNET, VGG have reasonable number of computing resources, and its relatively easy to train them with the current computing capability we have (Google Colab GPU)

## Description of Model

In accordance to the needs of this competition, we are looking at “Semantic Segmentation” algorithms.

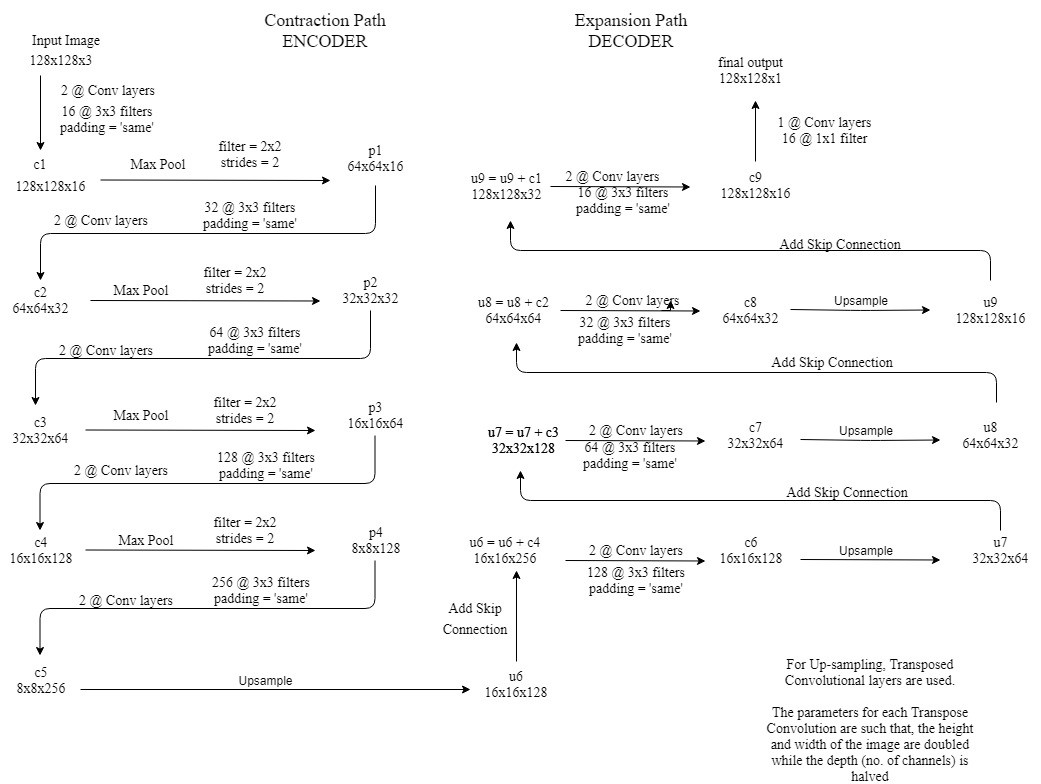
Semantic segmentation saw heavy use of deep learning techniques via the academic submissions to the *ImageNet Large Scale Visual Recognition Challenge*, or *ILSVRC*. The ILSVRC(Brownlee, 2019) is an annual computer vision competition developed upon a subset of a publicly available computer vision dataset called ImageNet, which is a very large collection of human annotated photographs designed by academics for developing computer vision algorithms. The ILSVRC tasks have led to milestone model architectures and techniques in the semantic segmentation. So, we looked at the winning algorithm from this competition, evaluated them in terms of -complexity and accuracy .In particular, we will be looking at the following algorithms:

### UNET

The UNET(Ronneberger, Fischer, & Brox, 2015) architecture consists of 3 parts:

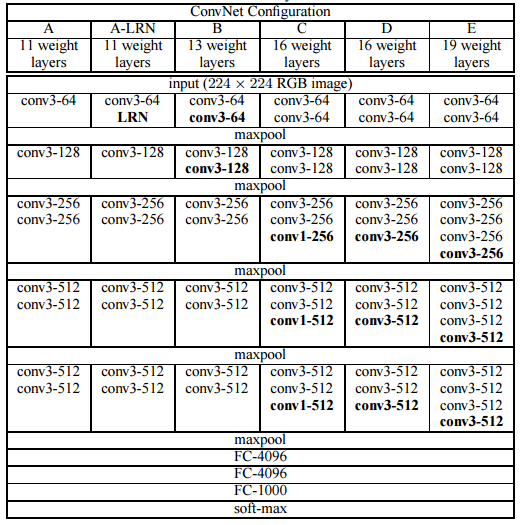
* 1. Encoder: The encoder part of the architecture is used to capture the context of the image. This means, it identifies which class each pixel belongs to. This is done by down-sampling the image. In this part the image dimensions are reduced.
  2. Decoder: The decoder part of the architecture is used to capture image localisation. During the encoder stage, the image dimensions were reduced so the decoder section works to increase the image dimensions back to original.
  3. Output: This is just the output layer which outputs the image which the the pixels labelled.

#### Model architecture



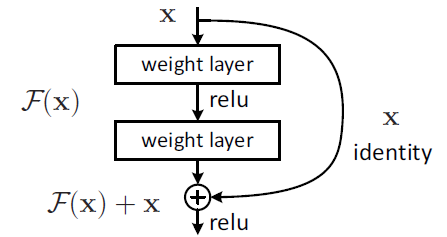
### **VGG-16**:(Simonyan & Zisserman, 2015)

This Oxford’s model won the 2013 ImageNet competition with 92.7% accuracy. It uses a stack of convolution layers with small receptive fields in the first layers instead of few layers with big receptive fields.



### [**ResNet**](https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf): (He, Zhang, Ren, & Sun, 2015)

This Microsoft’s model won the 2016 ImageNet competition with 96.4 % accuracy. It is well-known due to its depth (152 layers) and the introduction of residual blocks. The residual blocks address the problem of training a really deep architecture by introducing identity skip connections so that layers can copy their inputs to the next layer.



## Training Results

The training was done on 3 models. Since there were no labels given for the test set, we measured performance of the model using validation error. The metric used for the model was the dice coefficient.

### Evaluation

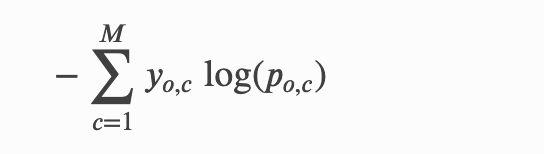
As specified in Kaggle, we will be using “Dice Coefficient”(‘Understanding the dice coefficient’, 2017) as a metrics to evaluate our model. The Dice coefficient can be used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth. Formula is given by:

where X is the predicted set of pixels and Y is the ground truth. The Dice coefficient is defined to be 1 when both X and Y are empty.

### Loss Function

For loss function, in order for the Neural Network to train faster, we used two loss function

* Dice Loss – As we needed to evaluate on basis of that
* Binary Cross entropy – It is represented by



* where y is the binary indicator (0 or 1)
* M is the number of classes
* p is the predicted probability of observation o of class c
* This observation is used because mask is a binary variable per pixel (0 for no mask and 1 for mask)

### Model Parameters

Model Parameters used for this analysis are as follows:

* 50 epochs for training and validation
* Step size per training epoch was 100
* Batch size was set to 16
* Loss function is described above
* Metrics is Dice Coefficient
* Adam optimizer is used as an optimizer
* Learning rate is 0.001
* Class imbalance is applied to most of the codes using “class\_weight” in keras

### UNET

#### With class imbalance treated

***Code Snippet - SDA\_UNET\_NoPretrain\_Code.ipynb***

|  |  |  |  |
| --- | --- | --- | --- |
| Epochs | Training loss | Training dice coefficient | Validation dice coefficient |
| 1 | 1.060 | 0.028 | 0.032 |
| 5 | 0.888 | 0.172 | 0.163 |
| 10 | 0.815 | 0.236 | 0.258 |
| 15 | 0.786 | 0.265 | 0.253 |
| 20 | 0.751 | 0.301 | 0.287 |
| 25 | 0.797 | 0.256 | 0.3 |
| 30 | 0.755 | 0.296 | 0.315 |
| 35 | 0.738 | 0.314 | 0.325 |
| 40 | 0.709 | 0.338 | 0.328 |
| 45 | 0.721 | 0.328 | 0.338 |
| 50 | 0.705 | 0.342 | 0.335 |

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

#### **Observations**

* The Dice coeffect metrics varies from 3% to 33.5%
* The loss decreases from 1.060 to 0.705
* There is a constant increase in the dice metrics and decrease in loss
* This model doesn’t use a pretrained model. This is why the training starts from scratch and is slower in general compared to a model which uses a pretrained model.

### ResUNET

#### With class imbalance treated

***Code Snippet - SDA\_ResNET.ipynb***

|  |  |  |  |
| --- | --- | --- | --- |
| Epochs | Training loss | Training dice coefficient | Validation dice coefficient |
| 1 | 0.998 | 0.112 | 0.055 |
| 5 | 0.795 | 0.919 | 0.135 |
| 10 | 0.710 | 0.339 | 0.374 |
| 15 | 0.658 | 0.388 | 0.355 |
| 20 | 0.629 | 0.415 | 0.37 |
| 25 | 0.619 | 0.426 | 0.349 |
| 30 | 0.596 | 0.45 | 0.43 |
| 35 | 0.554 | 0.483 | 0.325 |
| 40 | 0.56 | 0.468 | 0.446 |
| 45 | 0.55 | 0.489 | 0.476 |
| 50 | 0.552 | 0.485 | 0.357 |

#### A screenshot of a social media post Description automatically generated A screenshot of a social media post Description automatically generated

#### **Observations**

* The Dice coeffect varies from 5% to 35.7%
* The loss decreases from 0.998 to 0.552
* There is a constant increase in the dice metrics
* This model doesn’t use a pretrained weight too. Similar to the model above, the training starts from scratch and thus the convergence is slow.

#### Without class imbalance treated

***Code Snippet - SDA\_ResNET\_No\_class\_imbalance.ipynb***

|  |  |  |  |
| --- | --- | --- | --- |
| Epochs | Training loss | Training dice coefficient | Validation dice coefficient |
| 1 | 1.002 | 0.122 | 0.082 |
| 5 | 0.836 | 0.222 | 0.103 |
| 10 | 0.699 | 0.351 | 0.334 |
| 15 | 0.646 | 0.403 | 0.176 |
| 20 | 0.611 | 0.435 | 0.365 |
| 25 | 0.625 | 0.420 | 0.424 |
| 30 | 0.636 | 0.408 | 0.44 |
| 35 | 0.588 | 0.455 | 0.368 |
| 40 | 0.562 | 0.478 | 0.456 |
| 45 | 0.547 | 0.494 | 0.490 |
| 50 | 0.549 | 0.489 | 0.487 |

A screenshot of a map

Description automatically generated A screenshot of a cell phone

Description automatically generated

#### **Observations**

* The Dice coeffect varies from 8% to 48.7%
* The loss decreases from 1.002 to 0.549
* There is a constant increase in the dice metrics
* ***As Dr.NaN suggested, the no class imbalance treatment works better than class imbalance. This is because for different mask (4 classes) we have different channels, so class imbalance treatment is not needed.***
* This model doesn’t use a pretrained weight too. Similar to the model above, the training starts from scratch and thus the convergence is slow.

### VGG

#### With class imbalance treated

***Code Snippet - SDA\_VGG16\_NFrozen\_Pretrain\_Code.ipynb***

|  |  |  |  |
| --- | --- | --- | --- |
| Epochs | Training loss | Training dice coefficient | Validation dice coefficient |
| 1 | 1.214 | 0.023 | 0.044 |
| 5 | 0.879 | 0.173 | 0.189 |
| 10 | 0.824 | 0.231 | 0.214 |
| 15 | 0.856 | 0.196 | 0.212 |
| 20 | 0.843 | 0.209 | 0.215 |
| 25 | 0.828 | 0.219 | 0.225 |
| 30 | 0.828 | 0.224 | 0.229 |
| 35 | 0.845 | 0.205 | 0.236 |
| 40 | 0.819 | 0.237 | 0.244 |
| 45 | 0.775 | 0.275 | 0.296 |
| 50 | 0.761 | 0.288 | 0.243 |

#### A screenshot of a social media post Description automatically generated A screenshot of a cell phone Description automatically generated

#### **Observations**

* The Dice coeffect varies from 4% to 24.3%
* The loss decreases from 1.214 to 0.761
* There is a constant increase in the dice metrics
* As pointed out by Dr.Nan, training and test data is not diverging anymore in any of the models. We primary changed the sampling method during the training and validation

#### With class imbalance treated, No pretrain

***Code Snippet - SDA\_VGG16\_NFrozen\_Code.ipynb***

|  |  |  |  |
| --- | --- | --- | --- |
| Epochs | Training loss | Training dice coefficient | Validation dice coefficient |
| 1 | 1.1174 | 0.0337 | 0.0326 |
| 5 | 1.0146 | 0.0351 | 0.0514 |
| 10 | 0.9761 | 0.0783 | 0.0785 |
| 15 | 1.0080 | 0.0382 | 0.0403 |
| 20 | 1.0119 | 0.0334 | 0.0393 |
| 25 | 1.0111 | 0.0402 | 0.0409 |
| 30 | 1.0109 | 0.0352 | 0.0344 |
| 35 | 1.0118 | 0.0379 | 0.0365 |
| 40 | 1.0087 | 0.0408 | 0.0349 |
| 45 | 1.0115 | 0.0333 | 0.0325 |
| 50 | 1.0098 | 0.0352 | 0.0316 |

A screenshot of a cell phone

Description automatically generated A close up of a map

Description automatically generated

#### **Observations**

* The Dice coeffect varies from 2% to 3%
* The loss decreases from 1.117 to 1.0098
* There is a constant but very slow increase in the dice metrics
* This model doesn’t use a pretrained weight too. Similar to the model above, the training starts from scratch and thus the convergence is very slow.

## 

## Other models tried

We had multiple failed attempts in terms of model, almost all of them failed due to memory error. In particular, we were looking into the following models:

* Linknet
* PSPNet
* Deeplab v3
* Xception
* Inception v3

## Comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | UNET | ResUNET | ResUNET(without class imbalance treated) | VGG16 |
| Number of epochs | 50 | 50 | 50 | 50 |
| Training dice coefficient | 0.342 | 0.485 | 0.489 | 0.288 |
| Validation dice coefficient | 0.335 | 0.357 | 0.487 | 0.243 |
| Pre-train | No | No | No | Yes |

# Findings

1. Out of all the models tried, the ResUNET gives the best results in terms of dice-coefficient. ResUNETs perform well as they don’t suffer from the vanishing or exploding problem at any stage.
2. The ResUNET model where class imbalance is being treated slightly overfits to the training set as the validation dice coefficient isn’t as good as training dice coefficient. (Figues above)
3. Notably, performance of the model is better when class imbalance isn’t being treated. The validation dice coefficient is highest in this case. This can be because, we have different channels for different classes (4 channels for 4 classification classes) which is why class imbalance is not needed.
4. The UNET model also seems to be good. The model has almost reached convergence and seems to have a similar performance on the validation set as compared to the training set.
5. For VGG pretrain model, the accuracy is not at par with UNET and ResUNET model i.e. just 24.3%. For VGG without Pretrain model, accuracy is just 3%.
6. VGG is performing badly even after pretrain weights being presents, this proves the dominance of UNET and ResUNET architecture over VGG. In UNET, we have encoder, decoder and skip connection present, while VGG is a simple connection of convolution and Maxpool layer.
7. After the feedback from Dr.Nan stating that “training should not be diverged” we looked into our model and his suggestion about trying with various learning rate and different optimizers. Even after trying different learning rates (0.5, 0.1, 0.2) and optimizers (Adagrad, Adamax), the training loss was not converging. Next, we tried a different sampling method for our data generator which worked.

# Limitation

1. Due to the class imbalance problem, the model isn’t good at predicting classes which aren’t highly represented.
2. The lack of computational resources also hindered our project to some extent.
   1. We couldn’t experiment computationally expensive models such as Inception v3 and deeplab v3 plus.
   2. We were also not able to try methods to handle class imbalance such as data augmentation which requires large amounts of storage.
   3. The dice metrics in all our models seems to be increasing, due to lack of training resources, we were only able to train for 50 epochs with steps per epoch of 100. Ideally, we should be training until convergence.
3. The maximum model accuracy that we were able to get is just 48%, which is not enough to be trusted and deployed as an industrial based solution. Accuracy must be around 95% for this to be a reliable solution
4. Since it was a kaggle competition, we did not have the test labels to verify our model

# Conclusion

Deep learning seems to be the correct route to follow for such type of a problem. Deep learning models are appropriate in cases where there is large amounts of labelled data. Although there are many architectures available for a problem, it is a challenge choosing which model is the best as they depend on multiple factors such as pre-trained weights, class imbalance, computational resources, optimizer used among other factors. These were our learnings from this problem

1. In our problem, ResUNETs have the best performance in terms of training and validation performance.
2. Deeper models such as Resnets take longer time training compared to traditional models such as UNET.
3. Class imbalance is a big problem and without being treated effectively, the classes with class imbalance are not classified properly.
4. Pre-trained models are faster to train as the weights are already initialized. However, the pre-trained models are not always the best as they have been tuned to another dataset. They are a good option when there are lesser computational resources available.

# Reference

Brownlee, J. (2019, April 30). A Gentle Introduction to the ImageNet Challenge (ILSVRC). Retrieved 30 October 2019, from Machine Learning Mastery website: https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *ArXiv:1512.03385 [Cs]*. Retrieved from http://arxiv.org/abs/1512.03385

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *ArXiv:1505.04597 [Cs]*. Retrieved from http://arxiv.org/abs/1505.04597

Severstal: Steel Defect Detection. (n.d.). Retrieved 30 October 2019, from https://kaggle.com/c/severstal-steel-defect-detection

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv:1409.1556 [Cs]*. Retrieved from http://arxiv.org/abs/1409.1556

Understanding the dice coefficient. (2017, September 29). Retrieved 30 October 2019, from Deep Learning Course Forums website: https://forums.fast.ai/t/understanding-the-dice-coefficient/5838

[Kaggle Notebooks](https://www.kaggle.com/c/severstal-steel-defect-detection/notebooks)