Severstal: Steel Defect Detection

Utkrisht Rajkumar
A91060509
CSE Dept., UCSD
urajkuma@eng.ucsd.edu

Subrato Chakravorty
A53322325
ECE Dept., UCSD
suchakra@eng.ucsd.edu

Chi-Hsin Lo
A53311981
ECE Dept., UCSD
c2lo@eng.ucsd.edu

1 Introduction

Steel has been one of the most important building materials in modern times. Its resistance towards natural and man made wear has made it ubiquitous in the construction industry. To meet the growing demands, steel manufacturing companies maximize production while minimizing production defects. Steel defects primarily emerge during four stages of the steel manufacture process: heating, rolling, drying and cutting. Raw materials and scraps are sent to blast furnaces to be melted. The melted material is cooled in rolling mills, reheated, and undergoes chemical treatment. Lastly, the steel is cut to desired specifications. Due to the lengthy process and natural instability of the heating and chemical treatments, various types of defects often manifest in the final product. It is absolutely essential that the produced material is devoid of defects and meets safety standards to ensure reliability during construction.

Manual detection of these defects in steel is a tedious process and is often riddled with errors [6]. Recently, there has been a lot of emphasis on intelligent visual inspection systems to automate steel defect detection [17]. Machine learning approaches [17],[20],[6] have especially been successful due to their superior capability of learning from training data. Convolutional neural networks (CNNs) have been shown to provide state of the art results in problems such as steel surface defect classification [10, 7], surface defect segmentation [13] and detection and segmentation of manufacturing defects in casting[6].

Although there are ample amounts of research that demonstrate the efficacy of deep learning in classifying and localizing manufacturing defects, none of them, to our knowledge, have experimented with more recent architectures such as U-net[14], DeepLab[2, 3, 4], Dilated Residual Networks[18], etc. Another main challenge is that much of the existing research faced a shortage of a properly annotated dataset. Annotating defects in images of steel is laborious and requires an expert to do so. For reasons, existing models were limited by data availability.

In this work, team 3Nature-2Journal-8GPU used the newly released Kaggle dataset [16], to perform a comparative study of recent deep learning innovations in the field of semantic segmentation applied to the task of segmentation of steel surface defects. All code for this project can be found at `https://github.com/UCRajkumar/ece285`.

2 Background

Deep neural networks, such as AlexNet, have consistently outperformed traditional image processing algorithms in classifications tasks such as the ImageNet Challenge in 2012 [11]. In 2014, fully convolutional networks (FCNs) were constructed to perform pixel-wise classification to produce semantic segmentation on the input image [12]. Many variations of FCNs have since emerged to perform semantic segmentation [14, 18, 19, 4]. In this work, we will evaluate and fine-tune four semantic segmentation models: U-net [14], U-net with a ResNet encoder [8], U-net with an inverted residual encoder [15], and DeepLabV3+ [4] on their performance in the 2019 Kaggle competition [16] to detect defects in images of steel sheets.

2.1 U-Net

U-net [14], a type of FCN, was initially proposed for biomedical image segmentation by Ronneberger et al. and attained success in the ISBI cell tracking challenge 2015 by a large margin. Similar to its predecessor, the FCN by Long et al., the U-net is composed of two parts: the encoder and the decoder. The encoder is referred to as the contracting path as each layer halves the spatial dimensions of the

input. Conversely, the decoder (expansive path) gradually doubles the dimensions until the original resolution is obtained (Figure 1A). A standout feature of U-net, albeit it was initially proposed in [12], is the use of skip connections which concatenates feature maps from the contracting layer to the corresponding expanding layer. This concatenation greatly aids the network in recovering lost information during down-sampling layers and learning even with relatively small datasets. We refer the reader to [14] for further details of the model implementation. We explain our custom modifications to this model in section 3.1.

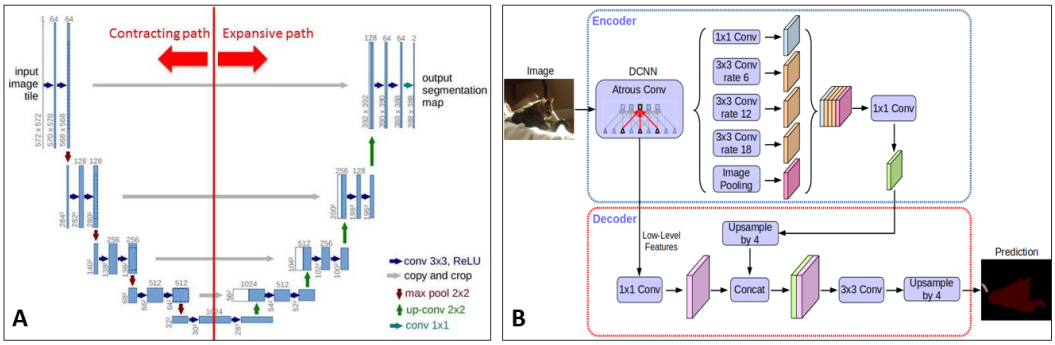

Figure 1: (A) U-net architecture. (B) DeepLabv3 architecture (see section 2.4).

2.2 Residual Net

Residual Net (ResNet) gained attention after winning in multiple categories at the 2015 ILSVRC and COCO competitions [8]. One of its primary contributions is identity shortcut skip connections which adds the input to the output of two consecutive convolutional layers before feeding it to a non-linear activation function (Figure 2A). This innovation was inspired by the need to have deeper networks while avoiding vanishing/exploding gradients and performance degradation. Let x_l and x_{l+1} be the input and output of the l-th layer respectively, $h(x_l) = x_l$ is the identity mapping, F is the learned residual function, and f represents the activation function. Then the forward of the network can be formulated as:

$$y_l = h(f(y_{l-1})) + F(x_l, W_l)$$

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i)$$

Copy of the deeper layers are also considered when computing the gradient of the shallow layers, which mitigates the weight degradation problem. The authors propose that simply stacking layers that perform an identity mapping should not produce a training error higher than its shallower counterpart. Moreover, allowing the stacked layers to fit a residual mapping is easier than directly fitting the desired mapping. We refer the reader to [8] for greater detail regarding ResNets.

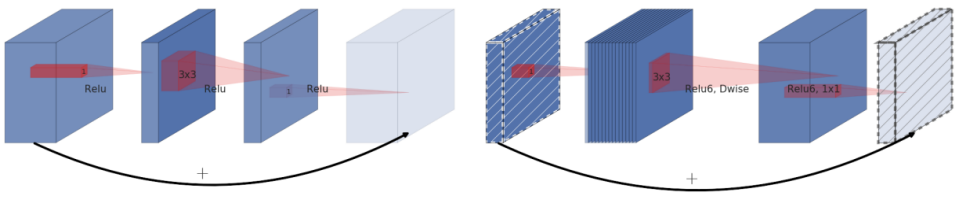

Figure 2: (A) Residual block. (B) Inverted Residual Block. From [15]

2.3 Inverted Residual Block

Traditional residual blocks(Figure 2A) connect the beginning and end of a convolutional block with a skip connection. The original bottleneck design for residual blocks in [8] have a sequence of $1 \times 1, 3 \times 3$, and 1×1 convolutions, in which the 1×1 layers are responsible for reducing and restoring the number of channels so that the 3×3 layer operates on smaller dimensions. It follows a wide $\rightarrow$narrow$\rightarrow$wide scheme.

Contrary to this, a inverted residual block [15] uses a narrow$\rightarrow$wide$\rightarrow$narrow scheme. It is also a stack of three layers but the 1×1 convolutional layers are responsible for *expanding* and then restoring the dimensions of the input tensors. The middle 3×3 convolutional layer is replaced with a depth-wise separable convolutional layer (Figure 2B). Since, the depth-wise convolution layer has significantly less parameters compared to a traditional convolutional layer, it is computationally feasible to operate on larger input dimensions. The last 1×1 convolutional layer restores the input dimension in order to have a skip connection.

2.4 DeepLab V3+

Google's DeepLabV3+ is the state-of-the-art semantic segmentation model designed and open-sourced by Google in 2016 [4]. There have been multiple improvements since the original model came out which include DeepLabV2 [2], and DeepLabV3 [3]. DeepLabV3+ uses inverted residual blocks and an additional decoder module to especially refine the segmentation results along object boundaries. DeepLabV3+ uses aligned Xception[5] as its main feature extractor whereas DeepLabV3 uses a ResNet architecture with atrous convolutions. DeepLabV3+ further employs depth-wise separable convolutions for atrous spatial pyramid pooling [2] which results in a more robust encoder-decoder architecture (Figure 1B). We refer the reader to [4] for greater detail regarding the mentioned features.

In this project, we build upon the code from a open source repository [1]. It supports MobileNetV2 and Xception backbones. Although, Xception backbone is generally considered more robust, it has 25 times more parameters than the MobileNetV2 backbone. Hence, we use the MobileNetV2 backbone for our training and inference on the steel defects dataset. To control the number of parameters of the network, the authors of [9] use a width multiplier, α, to modulate the number of filters in each layer. α can take a value of $(0, 1]$ and sets the number of filters in each layer to α times the originally proposed number of filters. Our custom modification to this model is further explained in Section 3.5.

3 Methods and Model Architecture

3.1 U-net

We modified the original U-Net in two ways. First, we changed the input dimensions from 572×572 to 256×1600, the size of each image in our dataset. Given more time, we would crop the image to 256×256. We explicate our reasoning for this decision in the discussion section. Secondly, we changed the number of input filters from 64 to 8. Note that U-net doubles the number of filters in very convolutional block in the encoder and halves in the decoder. Hence, the number of filters for all layers can be deduced using just the number of input filters.

We extensively fine-tuned our model by performing grid search on the number of filters with 64, 32, 16, and 8 input filters. We also trained the model with ReLU and LeakyReLU. Lastly, we trained the model with just BCE loss and BCE + Dice loss (see section 3.6). We found that using 8 input filters, using LeakyReLU, and BCE+Dice performed the best. The loss and accuracy for the final U-net model are presented in Figure 6.

3.2 Multi-Scale Context Aggregation by Dilated Convolutions

Classical FCN models were based on adaptations of convolutional networks that had originally been designed for image classification. Yu et al. proposed a multi-scale context aggregation using dilated convolutions (MCAD) that is specifically designed for pixel-wise classification. The proposed module uses dilated convolutions to systematically aggregate multi-scale contextual information without

losing resolution. The architecture is based on the fact that dilated convolutions support exponential expansion of the receptive field without loss of resolution or coverage.

The module consists of successive convolutional layers with a dilation rate that doubles every layer. The success of the module lies in terminating the number of layers when the receptive field of the last convolutional layer is at least as large as the input dimensions to the module. The *basic* version of the module keeps the number of feature maps the same in each layer while the *large* version doubles number of feature maps every 2 layers as seen in Figure 3.

Layer	1	2	3	4	5	6	7	8
Convolution	3×3	3×3	3×3	3×3	3×3	3×3	3×3	1×1
Dilation	1	1	2	4	8	16	1	1
Truncation	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Receptive field	3×3	5×5	9×9	17×17	33×33	65×65	67×67	67×67
Output channels								
Basic	C	C	C	C	C	C	C	C
Large	$2C$	$2C$	$4C$	$8C$	$16C$	$32C$	$32C$	C

Figure 3: Multi-scale Context Aggregation by Dilated Convolutions

In this work, we use the *basic* version of MCAD, that is, we keep the number of feature maps constant. We make two modifications to the originally proposed design. First, we remove the final two convolutional layers that have a dilate rate of 1. We empirically tested our model accuracy with and without those two layers and noticed that our model was more stable without the use of those 2 layers.

Second, our final convolution has a dilation rate of 12, instead of 16 as in the original paper (see Table 1). The input to our MCAD is 8×50 and our output receptive field is 57×57. We terminated the increase in dilation at his point since the original paper advises to end the increase in dilation as soon as the receptive field is larger than the input dimensions. We appended this module at the end of encoder and before the decoder in ResU-net and inverted ResU-net.

	1	**2**	**3**	**4**	**5**	**6**
Convolution	3×3	3×3	3×3	3×3	3×3	3×3
Dilation rate	1	1	2	4	8	12
Receptive field	3×3	5×5	9×9	17×17	33×33	57×57

Table 1: Custom MCAD

3.3 U-net with Residual Encoder (ResU-net)

In this model, we kept the same decoder as as the one use in section 3.1. However, we replaced each convolutional block with residual blocks. Although the original U-net performed quite well, we wanted to test if using more layers along with skip connections would truly ensure that the new model would be at least as good if not better than the base U-net as suggested in [8]. Each residual block is comprised of two 3×3 convolutional layers each following by batch normalization (batch norm), then the input is added to the output of the second batch norm, and lastly, we applied LeakyReLU to the output. Our architecture is presented in Figure 4.

As suspected, this model performed even better than the base U-net. By adding residual blocks, the model is able to regain partial information in the short skip connections and the extra layers did not cause the model to overfit.

We once again extensively tested this model by performing grid search on the number of filters. We also tested the model with just BCE loss and BCE + Dice loss. We found that 8 filter sizes and BCE + Dice loss performed the best.

3.4 U-net with Inverted Residual Encoder

We used the MobileNetv2 backbone as our feature extractor in DeepLab-v3+ which has inverted residual blocks in lieu of normal convolutional blocks or residual blocks. It gave us the idea that we

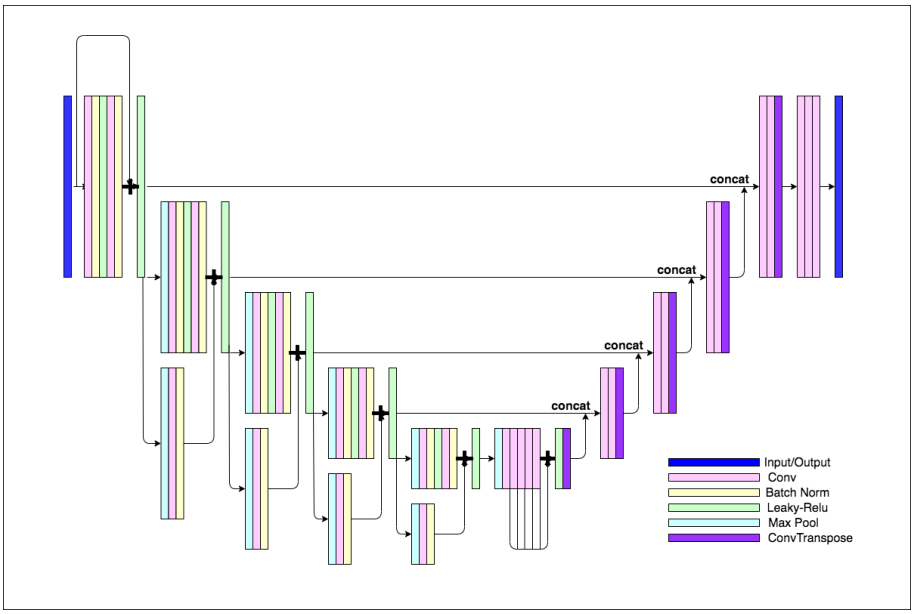

Figure 4: U-net with Residual Encoder.

can do something similar with our U-net with residual encoder model (section 3.3). We replace the residual blocks in the encoder part of ResU-net with inverted residual blocks, explained in Section 2.3. The intuition behind this strategy was to see whether using inverted residual blocks provide any improvement over residual blocks as it performs similar input-output transformation, i.e., the size of input and output tensors are same, but using less parameters. We trained using both BCE loss and BCE + Dice loss and found BCE+Dice performed better. Although, the results weren't better in terms of dice coefficient on test set when submitted on Kaggle, looking at some sample predictions provided us with some interesting conclusions which are discussed in Section 6.

3.5 DeepLabV3+

We trained a DeepLab-v3+ model on our steel defect dataset using the MobileNetV2 backbone as our feature extractor. The MobileNetV2 backbone has a width multiplier α which is used to modify the number of channels in each layer. Using a α value of 0.5 will halve the number of channels in each layer of the feature extractor. The width multiplier provides a trade-off between accuracy and computational cost. The role of α parameters is to thin a network uniformly at each layer. For a given layer and width multiplier α, the number of input channels M becomes αM and number of output channels N becomes αN. The computational cost of a depthwise separable convolution as function of α becomes:

$$D_K.D_K..D_F.D_F + \alpha M.\alpha N.D_F.D_F \tag{1}$$

where $\alpha \in (0, 1]$ and with typical values: 1, 0.75, 0.5, 0.25.

Due to large input image size (256×1600), using a default α value of 1 leads to memory errors, i.e., tensors don't fit in the GPU. Furthermore, because our dataset only has 6666 images with defects, a smaller model may generalize better on the test set. Therefore, we use α values less than 1 in our experiments. We perform a grid search over $\alpha \in 0.1, .25, 0.5, 0.75$. We found the optimal value of α to be 0.5 in our case. From our experiments, we inferred that Deeplab-v3+ may not be a suitable architecture for our problem mainly because of our input image size. Our input image size proved to be too big for the tensors to fit in our GPU when used with the default α value of 1. To tackle this issue, we had to reduce the width multiplier α for the Mobilenet-v2 backbone, which affected the model's capability to fit training data. As a result, DeeplabV3+ didn't seem to learn anything useful with subsequent epochs as evident from plots for dice coefficient and loss in Figure 6.

3.6 Loss

We defined loss L as binary cross entropy (BCE) plus the Sørensen-Dice loss (Dice). Specifically, the BCE loss for class $c \in \mathcal{C} = \{b, n, h, e\}$, where b, n, h and e represent each of the four defects, was computed using:

$$BCE[x] = -\frac{1}{\mathcal{C}} \sum_{c \in \mathcal{C}} \left[y_c(x) \ln \left(\frac{1}{1 + e^{-P_c(x)}} \right) + (1 - y_c(x)) \ln \left(1 - \frac{1}{1 + e^{-P_c(x)}} \right) \right].$$

Similarly, we compute Dice loss as:

$$Dice = \left[1 - \frac{2 \sum_c \mathbf{P_c} \cdot \mathbf{y_c}}{\sum_c (\mathbf{P_c}_1 + \mathbf{y_c}_1)} \right]$$

The net loss was computed using

$$L = \frac{1}{\mathcal{P}} \sum_x (BCE[x] + Dice)$$

3.7 Accuracy

Since the official Kaggle competition will be using Dice to evaluate our test predictions, we measure accuracy of our model for each image using Dice:

$$Dice = \frac{2 \sum_c \mathbf{P_c} \cdot \mathbf{y_c}}{\sum_c (\mathbf{P_c}_1 + \mathbf{y_c}_1)}$$

It measures the overlap of the ground truth and the prediction for all the classes. Note that the dice coefficient is defined to be zero if both sets P and y are empty. The final accuracy of the model as presented in Table 2 is simply the mean of the Dice coefficients for each image in the test set.

4 Experimental settings

4.1 Dataset

We obtained 12568 RGB images of dimensions $256 \times 1600 \times 3$ from [16]. The ground truth is of dimension $256 \times 1600 \times 4$ where each channel contains a binary mask denoting a different type of defect. Because not much scientific detail is provided in the competition regarding the different types of defects, we will denote the classes as $\mathcal{C} = \{b, n, h, e\}$ where b, n, h and e represent each of the four defects, respectively (Figure 5B). Out of the 12568 images, only 6666 images contained at least 1 and a maximum of 4 types of defect. In Figure 5A we present the number of images containing each class of defects and the number of images containing multiple defects. As can be seen, majority of the images contain defect of class C.

4.2 Pre-process

Sincere there are 5902 images that do not contain any defects, we removed them from the dataset in order to avoid severe class imbalance. There are 6239 images that contain only a single defect. The remaining 3 channels will act as the primary no-defect case during training. To reduce computation, we converted each image to gray-scale as the original were black and white and re-scaled the pixel value range to $[0, 1]$.

We performed a $95 - 5$ training-validation split on the data set 6332 images for training and 334 images for validation. The same training and validation data was used for the training of each model. We bench-marked accuracy on a test set containing 1801 images.

4.3 Training procedure

We trained each of our experimental models model for 20 epochs using the Adam optimizer with a default learning rate of 0.001. We used the validation set to track loss and Dice score during training. At each epoch, we only saved the model with best accuracy. All the training was performed using Keras on a single GPU.

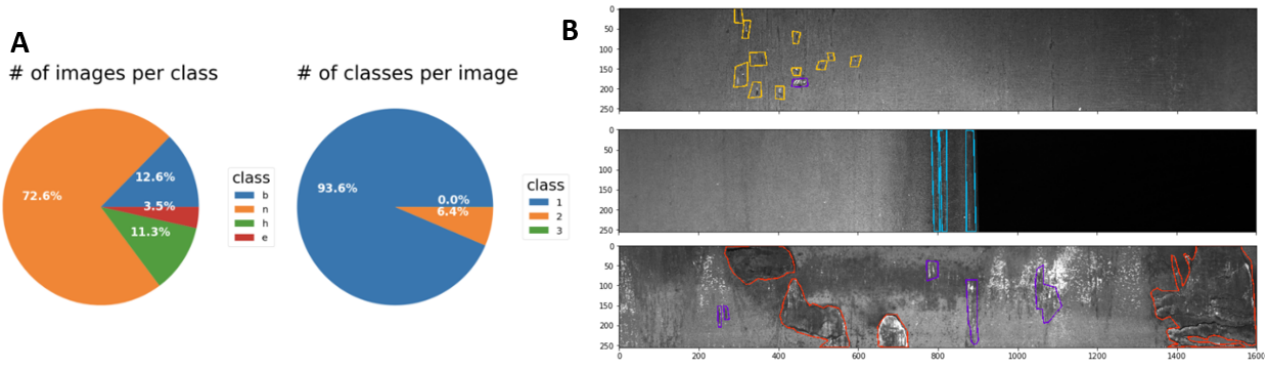

Figure 5: (A) Dataset statistics. (B) Image examples. Yellow, blue, purple, and red represent class b, n, h and e, respectively.

5 Results

We present our training and validation results in Figure 6. The residual encoder based U-net outperforms all the other models in terms of Dice on Kaggle. Whereas, the base U-net and U-net with inverted residual encoder have similar Dice scores on test set with the latter one performing slightly better. DeepLabV3+ performed the worst on all metrics with a final Dice score of 0.14 on test set. In terms of Dice and loss on training and validation set, all the models except DeepLabV3+ have well-defined and intelligible plots. The U-net with inverted residual blocks performed better on training set but not so well on validation, which could be due to over-fitting. The curves for DeepLabV3+ show that the model isn't learning anything useful with time, therefore, it can be inferred that the architecture with reduced α may not be a good fit for our problem. U-net Residual encoder outperformed other models with a score of 0.87 whereas the base U-net and U-net with inverted residual encoder have similar test scores of 0.85476 and 0.85674 respectively. Although, U-net and U-net with inverted residual blocks have similar quantitative scores, we found out that they make very different kind of errors (see section 6).

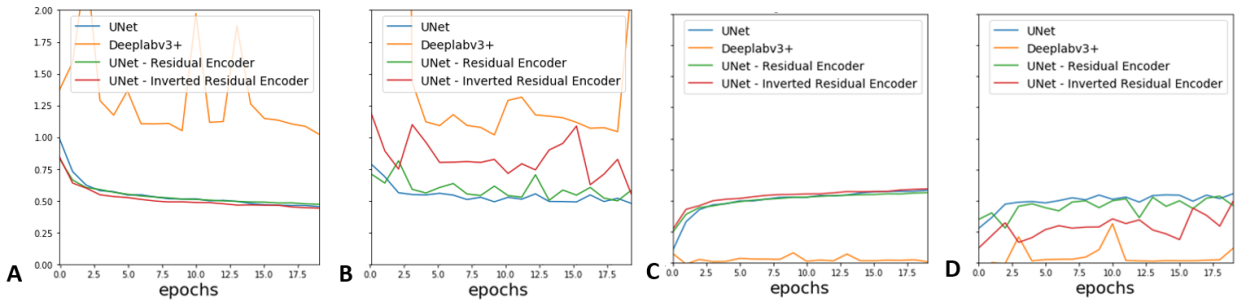

Figure 6: (A) Training Loss. (B) Validation Loss. (C) Training Accuracy. (D) Validation Accuracy.

Architecture	**Accuracy**
U-Net	0.85476
U-net with Residual Encoder	**0.87253**
U-net with Inverted Residual Encoder	0.85674
DeepLabV3+	0.14957

Table 2: Results on Kaggle test set (Public Score)

We qualitatively analyzed our model on the validation and test data. We refer the reader to Demo.ipynb in our Github repository (`https://github.com/UCRajkumar/ece285`) to view our qualitative results. As expected, DeepLab's predictions are illegible. To generate these plots, we add all the

feature maps for each defect along the channel dimension. Therefore, we go from a tensor of shape $4 \times H \times W$ to $1 \times H \times W$. Consequently, the feature map represents the probability of each pixel being a defect irrespective of its defect type.

Please refer to section 3 for discussion regarding detailed model setting and hyper-parameter failures, primarily determined using empirical observation.

6 Discussion

We can see a clear trend in Training loss and Training dice coefficient with increasing epochs for all the three U-net models. Although, same cannot be said for their respective validation plots, but it can be observed that there is a generally decreasing trend for validation loss over successive epochs and vice versa for validation Dice Coefficient.

All three U-net architectures perform well when evaluated qualitatively by visualizing segmentation maps on samples from validation data set. It is clear from Figure 7 that ResU-net gives better segmentation maps in terms of minimum false positives and false negatives when compared to other two U-net architectures. The base U-net architecture seems to be very conservative in deciding whether a pixel is a defect or not. Therefore, it makes less false positive errors but ends up increasing number of false negatives. Whereas, U-net with inverted residual blocks learns to do the exact opposite. It is very lenient in deciding whether a pixel is defect or not, therefore, although it finds most of the true positives but ends up making a lot of false positive errors. ResU-net seems to be able to strike the perfect balance between the making less false positive and less false negatives. Consequently, it performs better than both the models both qualitatively and quantitatively.

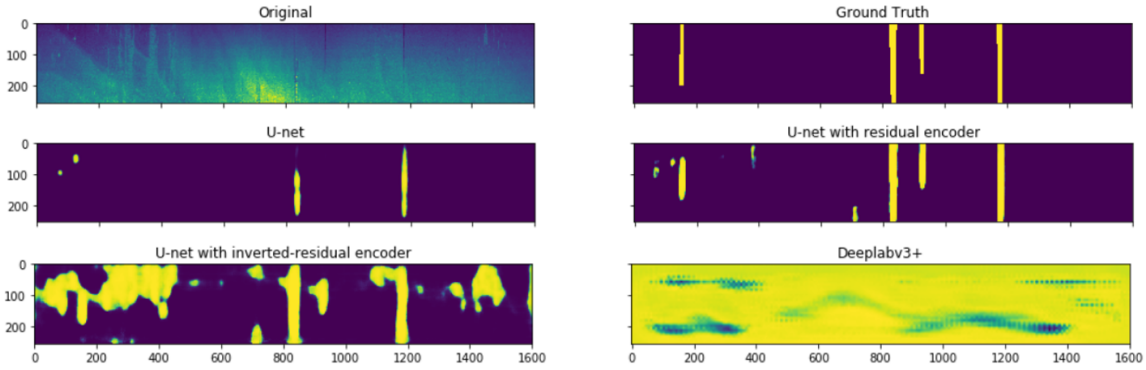

Figure 7: Predictions on validation data.

One of the main reasons behind the slightly sporadic validation curves could be the noise in the dataset. On extensive visual inspection of the dataset, we found many samples where the input image did not seem to have any defect in it but the ground truth contained defects. Similarly, we also found samples the input image had clear marks which seemed similar to defects in other images but the corresponding ground truth did not have any defect. One such example is Figure 8.

6.1 Lessons Learned

The primary difficulties we faced in this project are the following. **(1)** Working with unconventional and unfamiliar data. As we were not experts on the subject of steel defects, we had to rely solely on the description statements provided by Severstal and did not have any interpretive power over the ground truths, making it difficult to figure out where to fine-tune. There were also very few images compared to traditional deep learning datasets. We dealt with this by extensively reviewing prior work and examining best practices by other users on Kaggle. **(2)** Moreover, the dataset had quite a few images that we strongly believe were not properly labeled. Hence, the model was constantly being fed confusing ground truths. In order to combat this, we built models with less parameters so that the network did not over-fit on the data and generalized well. **(3)** We had limitations in terms of computational power. Semantic segmentation models are generally computationally expensive as they

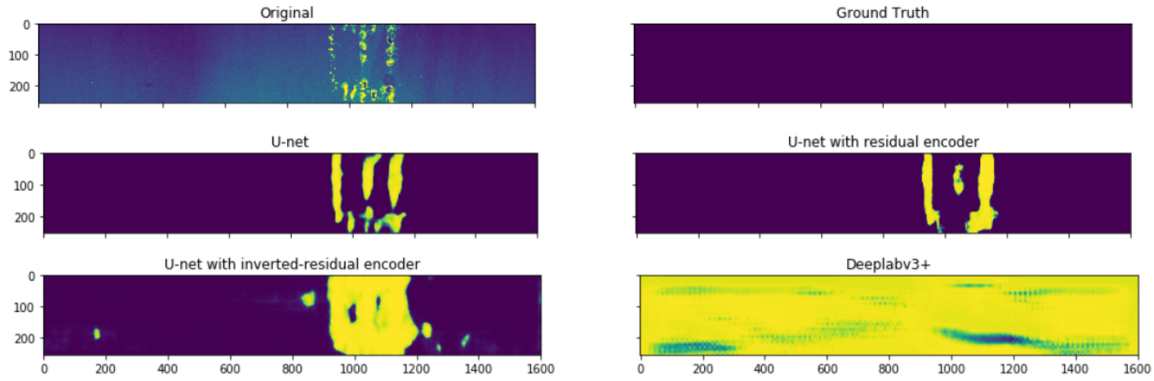

Figure 8: Contradiction between ground truth and prediction.

are dense predictions. In order to make the computation more feasible, we found that switching from multi-processing to a single thread based processing boosted results. If we allow multi-processing, then multiple batches are generated and queued to be sent to the GPU. However, the parallelization in Keras hasn't been fine-tuned when using generators and thus requires us to use a single thread-based processing to generate batches. This significantly mitigated training hurdles and stabilized our loss curves. **(4)** Early on, we experimented briefly with dilated residual networks and found that even the smallest version of the model was still too complex for our data and we abandoned fine-tuning it. Essentially, any model with greater than 3 million learning parameters, given the size and complexity of our data cause the model to not properly train.

6.2 Future Work

Given more time, there are a number of things we would do. Firstly, instead of using input image dimensions of 256×1600, we would first look for where there are defects and crop patches of 256×256 around the defect. This would yield multiple images from a single input image. A number of benefits can be attained through this. It would reduce class imbalance of non-defect pixels and make the model train better. Due to the smaller image sizes and greater number of images we could test more complex models.

Secondly, we would perform in-depth analysis of which class of defects is performing most poorly and try to fine-tune that class. We would also visually analyze what type of errors our model is making, more false positives, negatives, etc. On the basis of such analysis, we can try multiple approaches such as having more data augmentations for that particular class of defects and custom weighted loss functions.

We also found that the best scoring models on Kaggle performed an ensemble method by combining predictions from multiple models. Qualitative analysis of our models strongly suggests that using an ensemble method between base U-net, ResU-net, and U-net with inverted residual blocks might perform better than any of the models by themselves. This is something we would also test given more time.

Lastly, we would apply some post-processing modules to smoothen our segmentation and suppress prediction spurs. Post-processing modules we'd like to try are dense conditional random fields and non-max suppression algorithms.

Acknowledgments

We thank our TA Anurag Paul for his technical advice and invaluable guidance.

References

[1] bonlime. Keras implementation of Deeplabv3+. https://github.com/bonlime/keras-deeplab-v3-plus.

[2] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[3] L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[4] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018.

[5] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[6] M. Ferguson, R. Ak, Y. T. Lee, and K. H. Law. Detection and segmentation of manufacturing defects with convolutional neural networks and transfer learning. *CoRR*, abs/1808.02518, 2018.

[7] Y. Gao, L. Gao, X. Li, and X. Yan. A semi-supervised convolutional neural network-based method for steel surface defect recognition. *Robotics and Computer-Integrated Manufacturing*, 61:101825, 2020.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[10] M. S. Kim, T. Park, and P. Park. Classification of steel surface defect using convolutional neural network with few images. In *2019 12th Asian Control Conference (ASCC)*, pages 1398–1401, June 2019.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012.

[12] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation, 2014.

[13] G. Roberts, S. Y. Haile, R. Sainju, D. J. Edwards, B. Hutchinson, and Y. Zhu. Deep Learning for Semantic Segmentation of Defects in Advanced STEM Images of Steels. *Scientific Reports*, 9(1):1–12, 2019.

[14] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[15] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[16] Severstal. Kaggle 2019 Featured Code Competition. https://www.kaggle.com/c/severstal-steel-defect-detection.

[17] X. Sun, J. Gu, S. Tang, and J. Li. Research progress of visual inspection technology of steel products—a review. *Applied Sciences*, 8(11), 2018.

[18] F. Yu, V. Koltun, and T. A. Funkhouser. Dilated residual networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 636–644, 2017.

[19] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network, 2016.

[20] S. Zhou, S. Wu, H. Liu, Y. Lu, and N. Hu. Double low-rank and sparse decomposition for surface defect segmentation of steel sheet. *Applied Sciences (Switzerland)*, 8(9), 2018.