# Improving SRGAN for Object Detection

Ivan Abraham
University of Illinois at Urbana-Champaign
1308 W Main Street
itabrah2@illinois.edu

Wyatt McAllister
University of Illinois at Urbana-Champaign
1308 W Main Street
wmcalli2@illinois.edu

Justin Wasserman
University of Illinois at Urbana-Champaign
1308 W Main Street
jbwasse2@illinois.edu

## 1. Summary

In our proposal we called for implementing the network introduced in [2] in the production version of PyTorch as a way to learn more about neural networks as well as the new PyTorch library. We hypothesized that the network could be made more robust by changing the down-sampling techniques it is exposed to during training. Recall, in the original paper the authors implemented a convolutional neural network which, when trained on sample images, was able to perform science-fiction like image *enhancement* as commonly shown in television. Termed, single image super-resolution (SISR), this problem had received extensive attention in the community. As is well known, this is an ill-posed problem exacerbated by loss of high frequency information due to down-sampling. Their approach to solving the problem was based on a content loss measuring perceptual similarity in super resolved images, in combination with adversarial training to create *Generative Adversarial Network for Super Resolution (SRGAN)*. We were able to get the network implemented in PyTorch; further we were able to test our hypothesis with positive results vis-a-vis an object detection metric.

We first studied the algorithm the authors made in [2]. The authors implemented the network in a very old version of TensorFlow ($1.2 \sim 1.4$). We re-implemented it with latest release of the PyTorch library and ran it on our GPU. A very similar model in [6] had a more objective performance metric of object detection, while [2] has a subjective metric Mean Opinion Score (MOS), based on human viewers. To avoid the use of human opinions when testing our hypothesis, we used an object detection performance metric. Specifically, we compared the number of objects detected in super resolved images generated by the original network, and by a network trained with our modifications.

## 2. SRGAN

We first discuss the basic premise behind the SRGAN network. Many previous methods focused exclusively on pixel-wise reconstruction methods such as peak signal-to-noise ratio, which focus only on the image pixel data, and do not correlate with the image content perceived by humans. SRGAN instead focuses on a content loss, $l_X^{SR}$ (Equation 1), capturing the difference between the features of the high resolution and generated images, where $I^{HR}$ and $I^{LR}$ are the high resolution and low resolution images, $\phi$ is the feature map, and $G$ is the generator of super resolution images.

$$l_X^{SR} = \phi\left(I^{HR}\right) - \phi\left(G\left(I^{LR}\right)\right) \tag{1}$$

During training the content loss $\phi$ was implemented using pre-trained VGG16 and VGG19 networks available by default in the PyTorch library [**?**]. In the original paper the total loss for the generator was a combination of the content loss from (1), the pixel-wise mean square error (`torch.nn.MSELoss()`) and adversarial loss where the adversarial loss, $l_G^{SR}$ is given as in (Equation 2). Here $D$ is the discriminator, which outputs the probability that the generated image is a natural high resolution image.

$$l_G^{SR} = -\log D\left(G\left(I^{LR}\right)\right) \tag{2}$$

SRGAN consist of two parts: a convolutional feed forward network that learns features to create super-resolved images, and a discriminative network that functions as an adversary, and is trained to tell apart real and super resolution images. The novel approach in this paper is the use of a content loss, which associates local textures with patches of the image, and uses them to fill in detail in the high resolution image. The neural network finds a feature representation for the image, and determines textures for each patch based on these features. These textures are then combined and smoothed to create the final super resolution image.

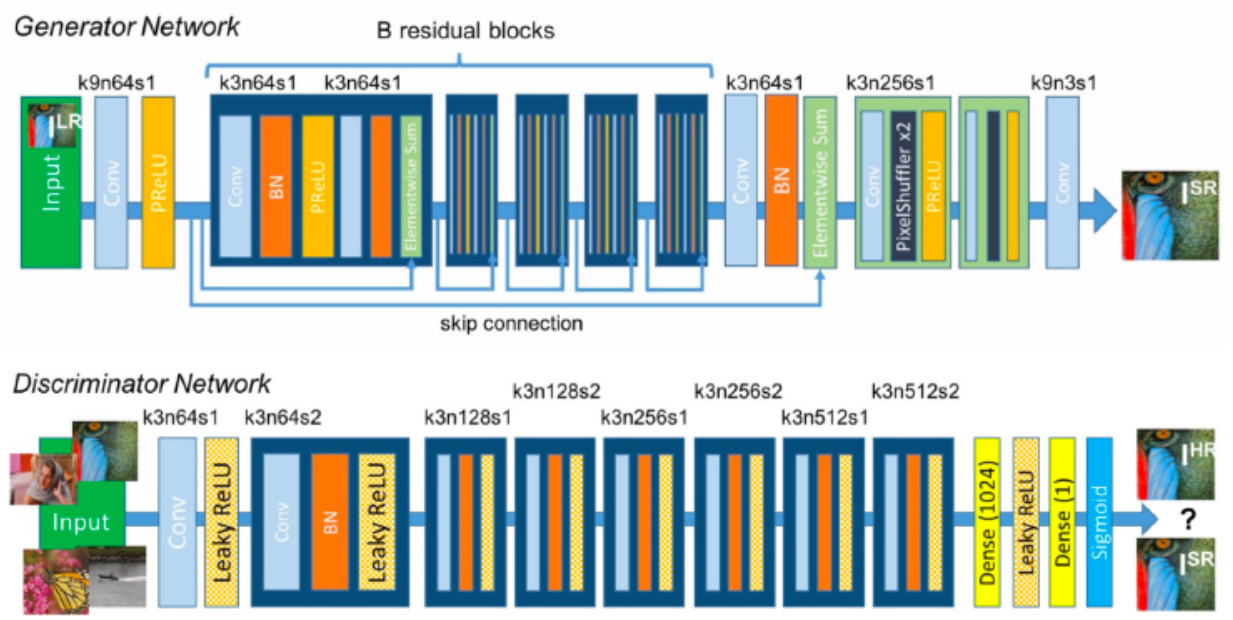



Figure 1: Architecture of Generator and Discriminator Networks with corresponding filter size (k), feature number (n), and filter time step (s) indicated for each convolutional layer.

### 2.1. Discussion

The neural network architecture used in [6] and [2] presented several advantages over past work. The exclusive use of convolutional layers allowed a single model to be trained on an input image of any size. The use of a feed-forward architecture rather than a recurrent architecture improved efficiency by passing the image through only once to get the result. Rather than provide a bicubic upsampling of the low resolution (LR) input image, and then add in texture information, the authors instead upsampled feature activations so that the super resolved image is generated as features are passed through the layers of the network. Additional convolutional layers were added for extra smoothing of the output image features. The use of residual blocks instead of stacked convolutional layers improved efficiency.

## 3. Project Description

We used the object recognition method from [6], as implemented in [5] to benchmark the performance of SRGAN, from [2]. We used the Python Imaging Library (PIL) [**?**], with Nearest Neighbor, Linear, Bicubic, and Sinc downsampling methods to try and improve robustness. We then performed a second experiment where we removed the Linear and Nearest Neighbor interpolations, and updated the loss from VGG16 to VGG 19, which is known to have higher performance, and examined the affect on the performance for object detection.

## 4. Description of Network

Let $k$, $n$ and $s$ denote filter (or kernel) size, number of features detected by each layer, and the time step for each filter. Then the architecture used by the authors, [2], modified from the implementation in [3], is given as in Figure 1. All kernels were $3 \times 3$. We used 16 residual blocks. Conv stands for a convolutional layer, and (P/l)ReLU stands for (Parametric/leaky) rectified linear unit, BN stands for batch normalization, $\oplus$ denotes elementwise sum.

For the object detection metric we used the third iteration of a pre-trained neural network called YoLo (You Only Look Once!), trained on the Common Objects in Context (COCO) Dataset [**?**]. This is a state of the art real-time object detection network that has been validated on many major data-sets including images and video. We refer the reader to [5] for details on the network's architecture.

## 5. Methods and Experiments

First, we implemented a convolutional neural net to super resolve images, before adding the generative adversarial net, which give SRGAN its high performance. We then used the object detector in [5] as metric to compare performance for the proposed modification to training. We finally benchmarked our implementation, on the aforementioned object recognition metric to see if training the network on images downsampled with different methods improved the performance under this metric with positive results.

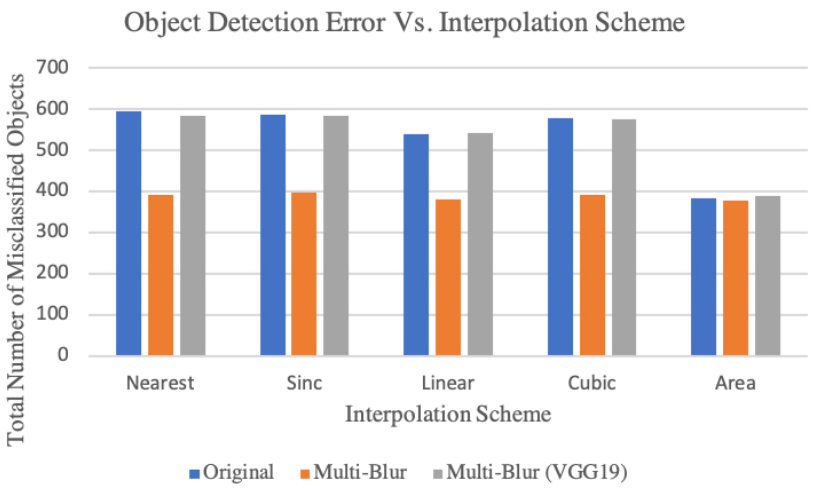



Figure 2: Object Detection Error Results

## 6. Resources Used

We ran the experiments on a NVIDIA GeForce GTX 1070 GPU on Mr. Wasserman's PC. To train SRGAN, we utilized the Visual Object Classes (VOC) 2012 Dataset, which consists of over seventeen thousand images, labeled with object classes [**?**].

## 7. Implementation

The production version of PyTorch used gave several advantages over the original TensorFlow implementation. In the implementation from [2], the VGG network and its API had to be written into the network, whereas PyTorch already provided pre-trained models as we have already noted. The use of PyTorch made it much easier to implement the content loss of (1) using VGG models. Further, one could easily swap VGG16 model for the VGG19 model by simply changing a few lines. The module `torch.utils.data.dataset.DataSet`, when sub-classed properly, requires the definition of the internal objects `__getitem__()` and `__len__()`. This allows commands to be more "Pythonic" vis-a-vis generator and list comprehensions. Moreover, the pre-processing steps to be done on the image (crop, transform etc.) could be built into the `__getitem__()` method so that this does not need to be handled separately. Finally, PyTorch also provided the related module `torchvision.transforms`, which contains most common image transformations one uses. By converting the loaded data to a Python Imaging Library (PIL) image format, many transformations could be composed together easily and conversions from images to PyTorch tensors to PIL Images could all be handled automatically. This supports the folk-news that PyTorch is indeed a more Pythonic framework for neural networks.

## 8. Results

### 8.1. Object Detection Metric

For the object detection metric [5], the original SRGAN Network fails for several objects in many images, such as the one in Figures 4b and 5b. As shown in Figures 4d and 4f, the network trained with the use of multiple downsampling schemes, which we call **Multi-Blur**, outperforms the original SRGAN network. When the Multi-Blur network is retrained without the Linear and Nearest Neighbor interpolations, which are suboptimal for upsampling, the performance drops again, as shown in Figures 5d and 5f. This fact, along with the fact this new scheme included the VGG19 loss instead of the VGG16 loss, which is known to perform worse without additional multi-scale training tricks [4], suggests that the loss of some texture information leads to the increased performance. In essence, a loss of some texture content creates increased robustness of object detection to the downsampling method used.

As shown in Figure 2, the training scheme Multi-Blur was tested on the VOC 2012 data set, composed of images with pre-labeled object classes, and achieved superior performance in terms of object detection error for all interpolation schemes tested. The Multi-Blur (VGG19) scheme does not show the same improvement without the Linear and Nearest Neighbor interpolations. The reason for this is hypothesized to be the fact that the learned weights for the generative network for the super-resolved images reflect all of the interpolation schemes. This results in a smoothing out of textures across the entire image. This smoothing causes a lower variation in the texture information within objects, improving object detection across all interpolation schemes, at the cost of reducing photo-realism.

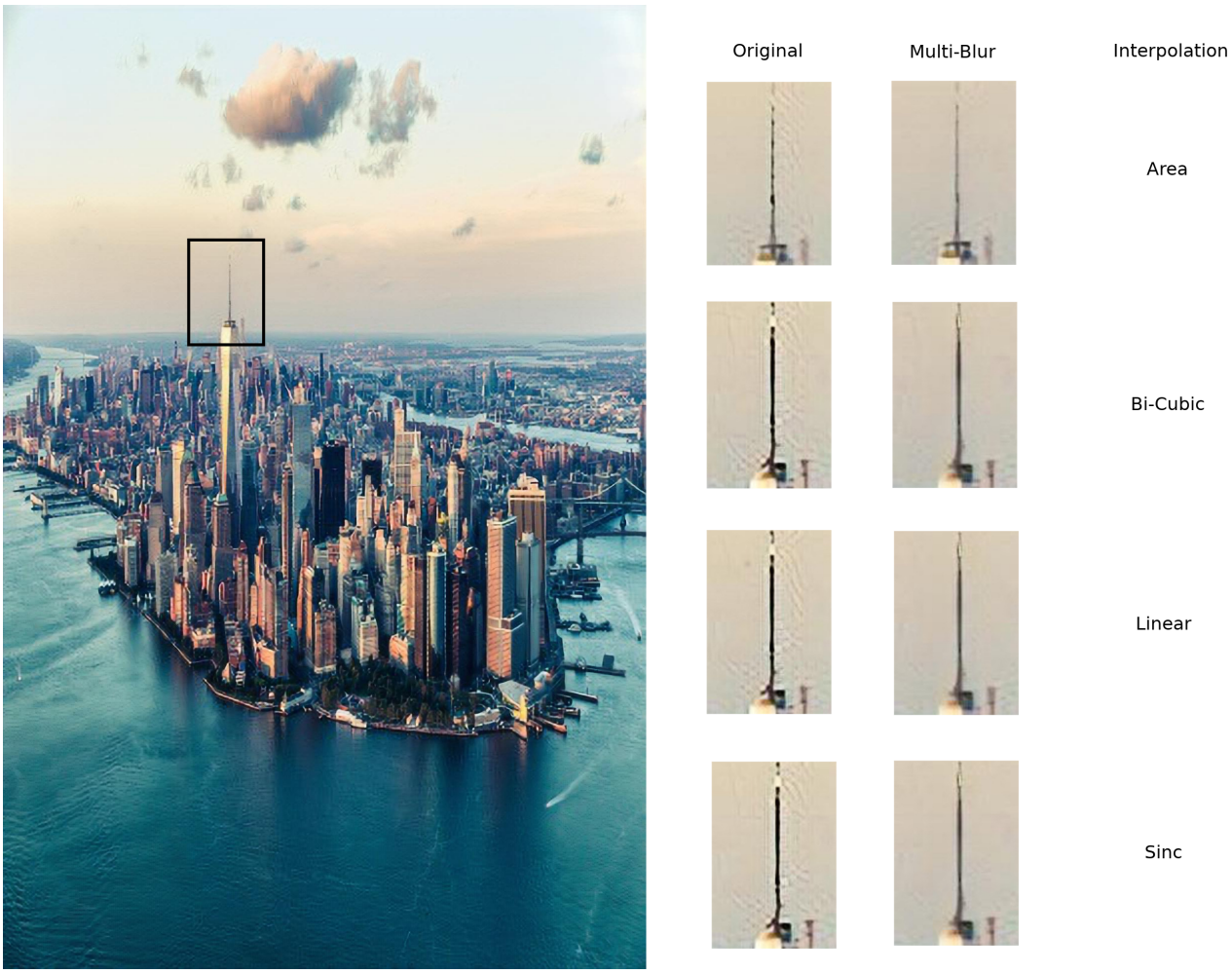



Figure 3: Super Resolution Comparison

## 9. Discussion of Multi-Blur Performance

As seen in Figure 3, our network training scheme, which we call Multi - Blur, reduces image artifacts in homogeneous image regions, at the expense of smoothing textures within objects. We see from Figures 4a, 4c, and 4e, it also does well with fine grained features such as text. As shown in Figure 4, the Multi-Blur training scheme produces readable text for this excerpt of Moby-Dick, while the original network fails to do so. From Figures 5a, 5c, and 5e, we see Multi-Blur (VGG19) offers a slight improvement for text, despite only having a slightly higher classification accuracy.

## 10. Conclusions

This training scheme improves the SRGAN network performance for object detection. This is at the expense of smoothing out image textures. While less photorealistic, the SR images have less image artifacts, and do well on images with detailed texture information, such as text.

## 11. Possible Extensions

The authors of SRGAN comment that super-resolution of text images is hard because the loss function used tends to 'hallucinate' texture details which may corrupt text content. One could investigate whether this is an artifact of the network architecture, or that of the loss used from [7]. The substitution of *more appropriate* loss function suited to text may allow the same network to perform well on SR of textual images. This may possibly be done by training modifying the VGG network to train on text images.

A possible extension to this work is to utilize optimal character recognition (OCR) in conjunction with the adversarial network so the adversary compares OCR generated text from SR image (i.e. SR-text) to OCR generated text from the HR image in an effort to force the generative network to learn parameters that are suited for text-image SR. A version of this idea has appeared at the 2018 International Conference on Pattern Recognition [1].

## References


[1] A. Lat and C. Jawahar. Enhancing OCR Accuracy with Super Resolution. In *2019 International Conference on Pattern Recognition (ICPR)*. IEEE, jul 2018.

[2] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114. IEEE, jul 2017.

[3] leftthomas on GitHub. Pytorch Implementation of SRGAN. https://github.com/leftthomas/SRGAN, 2018.

[4] Medium. A Review of VGGNet The 1st Runner - Up for The Image Classification Task, in ILSVRC. https://medium.com /coinmonks/paper-review-of- vggnet-1st-runner-up-of -ilsvlc-2014-image-classification-d02355543a11, 2014.

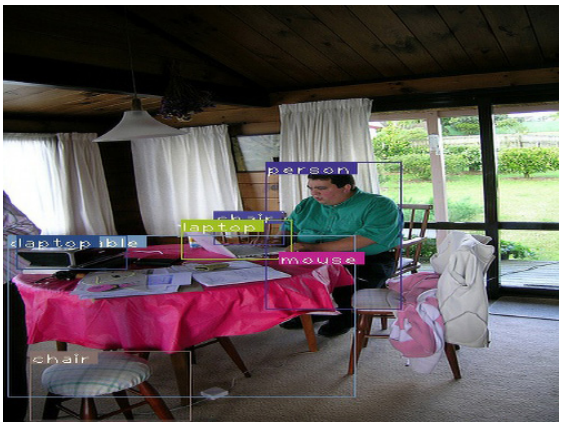
[5] ryoppippi and ayooshkathuria. An Implementation of the YOLO v3 Object Detection Algorithm. https://github.com/ayooshkathuria/pytorch-yolo-v3, 2018.

[6] M. S. M. Sajjadi, B. Schölkopf, and M. Hirsch. EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis. *Computer Vision (ICCV), 2017 IEEE International Conference on*, dec 2016.

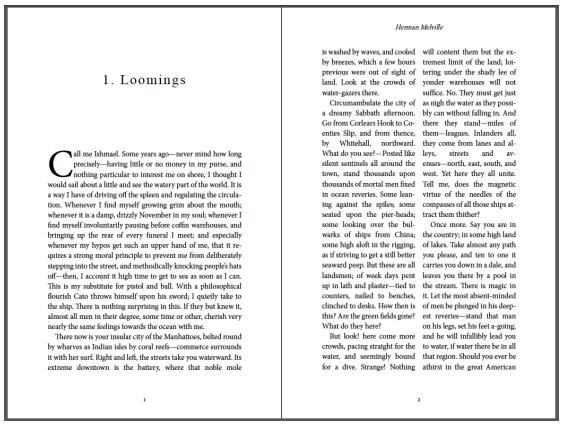
[7] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv*, sep 2014.

## 12. Appendix: Additional Examples

Figure 4 shows additional examples.

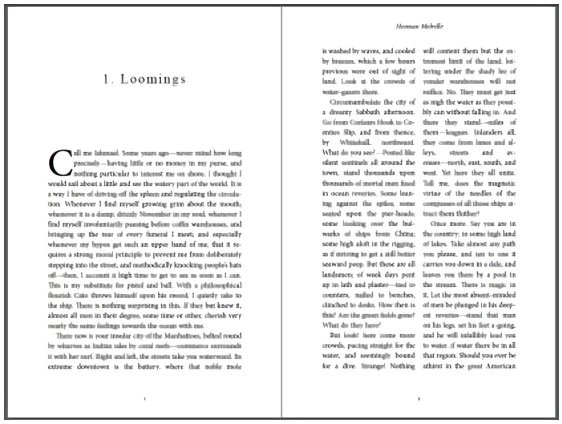



(a) Object Detection: Original Image



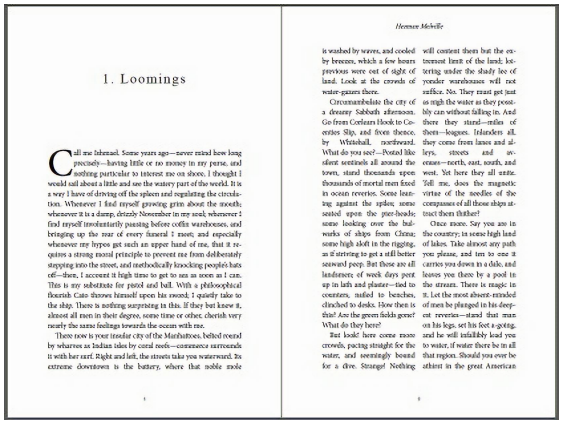
(b) Text: Original Image


(c) Object Detection: SR Image - Multi-Blur



(d) Text: SR Image - Multi-Blur

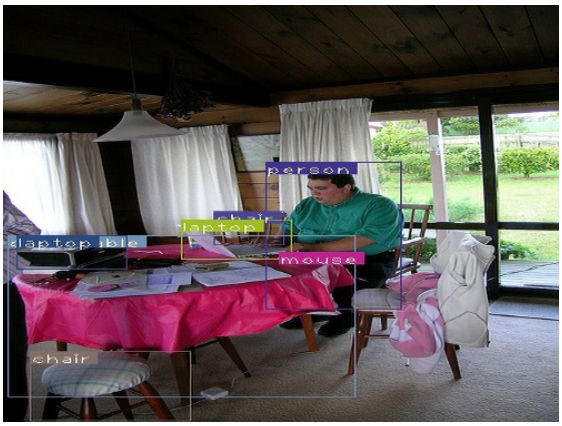

(e) Object Detection: SR Image - SRGAN



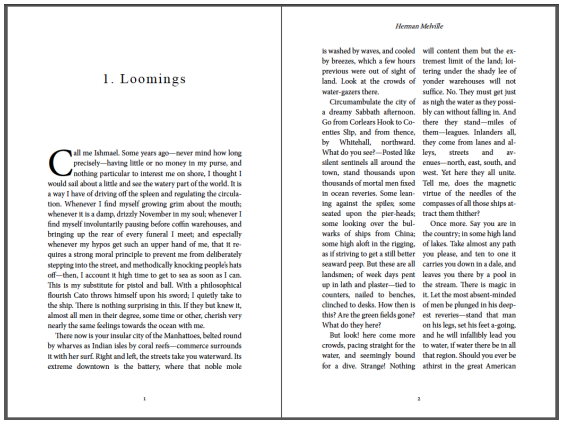
(f) Text: SR Image - SRGAN

Figure 4: Examples with Multi - Blur. Top row shows original image. Bottom row shows output of SRGAN when given LR images downsampled with algorithm different than what it trained on. Middle row shows our algorithm, which is more robust to downsampling technique.

Figure 5 shows additional examples.




(a) Object Detection: Original Image

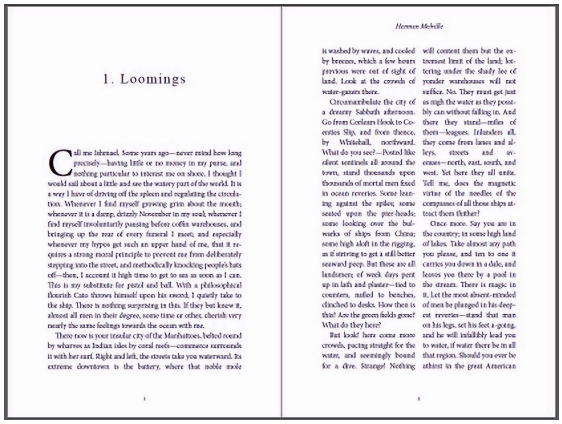


1. Loomings

Herman Melville

(b) Text: Original Image

(c) Object Detection: SR Image - Multi-Blur (VGG19)

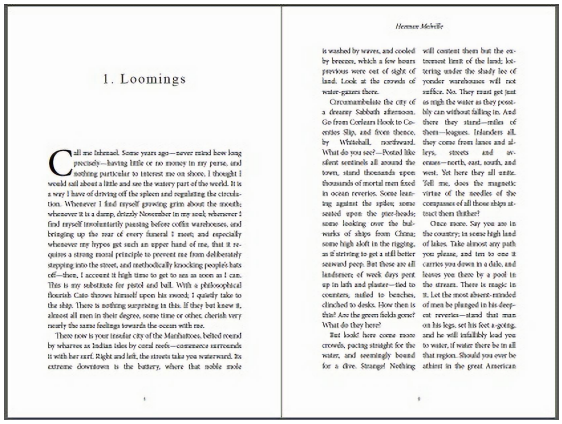


1. Loomings

Herman Melville

(d) Text: SR Image - Multi-Blur (VGG19)


(e) Object Detection: SR Image - SRGAN



1. Loomings

Herman Melville

(f) Text: SR Image - SRGAN

Figure 5: Examples with Multi - Blur (VGG19) with Linear and Nearest Neighbor Interpolations Removed. In this case the middle row is only slightly better than the bottom row showing that inclusion sub-optimal interpolation algorithm during training make for a more robust network.