# Towards Generator Network Verification

Zach Stoebner

*CS 6315 – Automated Verification*

*Vanderbilt University*

*Spring 2021*

**Abstract**

This report details a study on deep generator network verification using the Neural Network Verification (NNV) toolbox – a set-based deep neural network (DNN) and learning-enabled verification framework. NNV defines exact and over-approximate reachability algorithms for several input set representations, such as zonotopes, star sets, and abstract-domain polytopes. In this project, NNV's reachability algorithms were applied to an MNIST convolutional classifier network that takes as input the output of a variational autoencoder (VAE) and a SegNet to verify the classifier's robustness in correctly classifying a fake image. The input set representations used were the ImageZonotope and the ImageStar. The output set methods were the approximated zonotope, star, and abstract polytope. When the output of the generator network was left untainted, the classifier robustly classified the fake image in most cases. Similarly, attacking the brightness of the real and fake images expectedly confused the model more than the unperturbed images. Large attacks resulted in greater confusion, while the classifier was still robust in a few cases for small brightness attacks. Raw input classification was not more robust than the fake inputs, which are clearly imperfect, while brightness attacks can confound the model.

## 1 Introduction

### 1.1 Background

In recent years, neural networks have gained unprecedented traction in both research and industry. Unfortunately, current DNNs are often confused easily by imperceptible disturbances, i.e., blurring or intensity changes, and typically do not generalize well to all types of input. As DNNs have entered the mainstream through the CPS economy, tunnel vision in DNNs leads to a host of safety concerns. In turn, research into efficient and effective methods to verify the safety and robustness of this technology has garnered increasing interest [1].

Algorithms for verifying of neural networks generally aim to analyze layer-by-layer reachability to determine bounds for the nodes and output or falsify an assertion through search or optimization algorithms [2]. In terms of reachability, a neural network can be visualized as a transition system where each layer transforms the input so that, theoretically, the reach set at each layer could be computed given any possible input set. After the set is transformed through all layers, then the intersection of the transformed set and a property can yield counterexamples to determine the robustness of the model. Unironically, verifying DNNs is just as challenging as building and training them, if not more. First off, input domains for these models are often infinite and dynamic; for example, image datasets are constantly growing due to the myriad ways to obtain an image of the same subjects. Secondly, standard classifier networks, such as VGG-16 and VGG-19 [3], have upwards of 100 million parameters, which can take hours to train and likely many more to verify all possibilities on those parameters for a set of inputs. Lastly, the activation functions can split the set, making the reach set computation non-trivial. In fact, few DNN verification frameworks support activation functions besides the rectified linear unit (ReLU) [2].

### 1.2 Motivation

As a result of the novelty of DNN verification, little investigation has been conducted on verifying generator networks, particularly due to few frameworks supporting upsampling layers. Thankfully, NNV [4] has budding support for semantic segmentation architectures and is developing more support for DNN architectures that employ upsampling. To help set a baseline and justify the need for further investigation into deep generator network verification, this project is a case study on how a small MNIST [5] convolutional classifier responds to the fake outputs of two different, well-known autoencoder frameworks: the VAE [6] and SegNet [7].

## 2 Methods

All training and experiments were run on a MacBook Pro using CPU. All training, code, and experiments were written in MATLAB. The small convolutional classifier was ported from the MNIST CNN examples included in NNV.

## 4.1 VAE

The VAE was assembled traditionally, splitting the encoder and decoder networks into two separate components and manually sampling the latent space to feed into the decoder. The ELBO [8] loss function was used to compute the gradients for training and the VAE was trained for 50 epochs mini-batch size of 512 and a learning rate of 1e-3; the runtime for VAE training was approximately 1 hour on CPU. Figure 1 displays the layer graphs for the encoder and decoder components of the VAE. Figure 2 shows examples of the real inputs and fake outputs of the VAE.

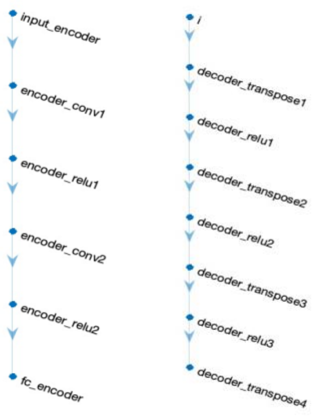

Figure 1. Layer graphs for the VAE encoder (left) and decoder (right). The encoder is comprised of two convolution-ReLU blocks followed by a fully connected layer to generate the latent vector. The decoder is comprised of three transposed convolution-ReLU blocks followed by a final transposed convolutional layer to obtain the original image's dimension. The sample latent vector is generated from the encoder via the VAE's additional mean and standard deviation latent vectors.

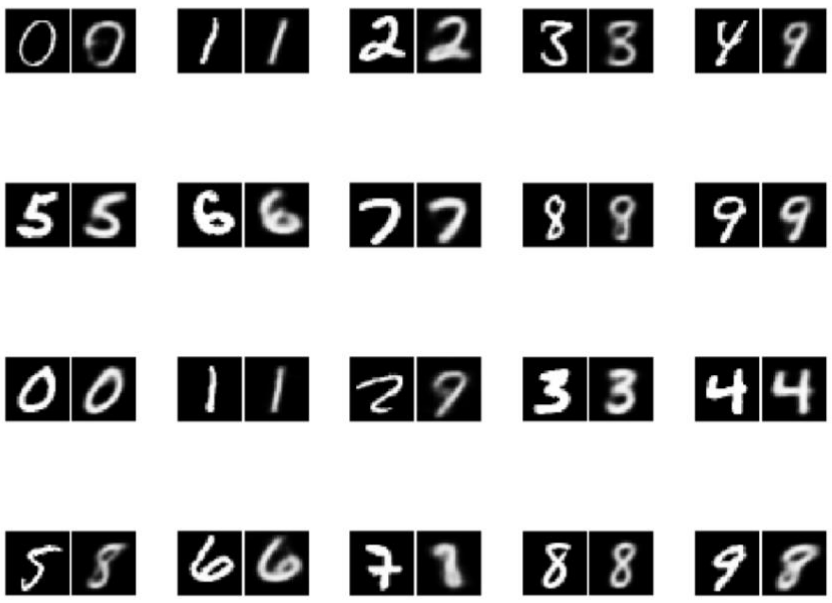

Figure 2. Examples of real inputs (left) and fake outputs (right) for the VAE. The pairs demonstrate examples where the VAE performs well and where it does not, noticeably a ‘9’ that becomes an ‘8’ and a ‘4’ that becomes a ‘9’.

## 4.2 SegNet

The SegNet was assembled using MATLAB’s built-in SegNet constructor and modified to generate a fake image instead of a segmentation. To convert the traditional SegNet architecture into an autoencoder, the pixel classification layer and SoftMax layer were dropped from the end of the model, which permitted the generation of a fake image. To keep the training even, the ELBO loss function was also used to compute the gradients for the SegNet with a mini-batch size of 512 and a learning rate of 1e-3 for 20 epochs; the runtime for SegNet training was approximately 1.5 hours on CPU. Figure 3 contains the layer graph for the SegNet. Figure 4 shows examples of the real inputs and fake outputs of the SegNet. The training scripts for both the VAE and SegNet were adapted from the MATLAB example on VAEs.

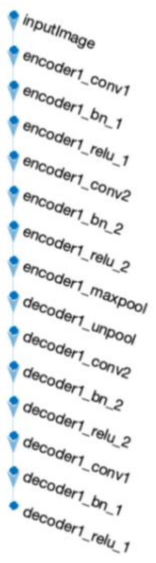

Figure 3. Layer graph for the SegNet autoencoder. The encoder is composed of two convolutional-batch normalization-ReLU blocks followed by a max pooling layer at the bottleneck. The decoder is composed of an unpooling layer at the bottleneck followed by two transposed convolutional-batch normalization-ReLU blocks. Unlike the VAE, the SegNet does not require sampling after encoding and can therefore be joined as a single SeriesNetwork.

## 4.3 Classifier Verification

To verify the classifier against the fake output images from each model, NNV parsed the model into a CNN representation. Then, 10 images were randomly drawn from the MNIST test set, one for each class. For each image, either no attack or a brightness attack occurred after the prediction. Two separate attacks were made, eliminating the top 1% or 5% of intensities where the lower bound image zeroed out the original intensity while the upper bound kept 5% of the original intensity. For untainted images, the lower and upper bounds were the same as the predicted image. After computing lower and upper bounds, the ImageZonotope set was constructed from the bounds and then the ImageStar set was computed directly from the ImageZonotope set [9]. Finally, the sets were passed to the NNV representation of the classifier to approximate the zonotope, star, and abstract polytope reach sets.

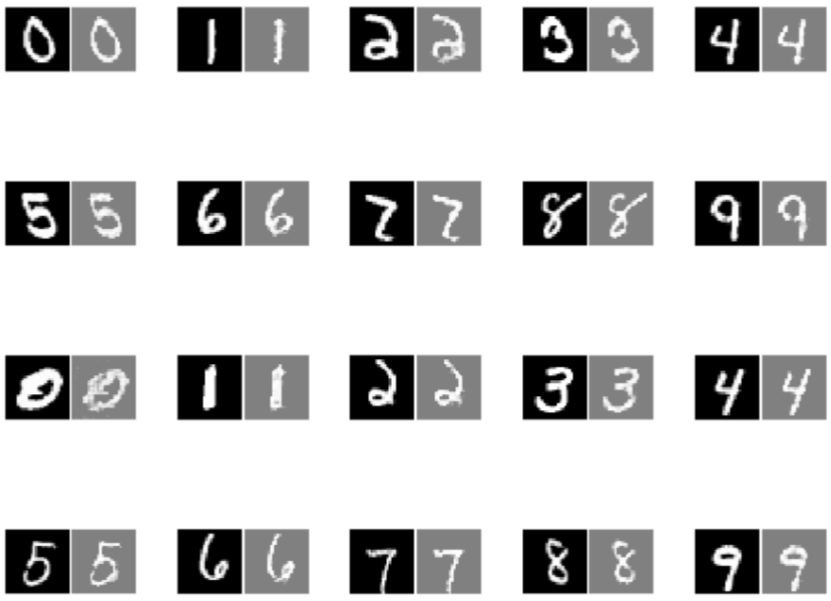

Figure 4. Examples of real inputs (left) and fake outputs (right) for the SegNet autoencoder. Compared to the VAE, the SegNet is less confused about the structure of the input images. However, the two channel output results in an average of the binary masks which gives a gray image.

## 4.4 Analysis

After computing the reach sets, the error bars for each class in each output set were plotted. NNV's MNIST CNN example plotting code was used for the error charts. Three points of comparison were used for the analysis: classifying the raw MNIST images, the VAE's fake images, and the SegNet's fake outputs. To understand the robustness of each of the three tasks, the error bar charts were empirically analyzed to discern whether the classifier was robust for a given class for a given task, as well as trends across the three tasks. The classifier is robust for a given class for a given model if the lower bound of the error bar for the class is greater than the upper bound of all other class' error bars on any one of the three output sets' charts.

## 4.5 Implementation Details

Due to the sigmoid function converting the images to [0,1] greyscale, fake images were interpolated to [0,255] to conform to the expected input values of the MNIST classifier. To verify the SegNet autoencoder, the predicted images have two channels, for each segmentation class, whereas the classifier expects only one. To conform, the mean image across the channels of the SegNet prediction was used. Afterwards, a threshold of 150 was applied to the SegNet images where all intensities below that threshold were zeroed out. Since the segmentation classes are 0 and 1, the SegNet learned semi-binary masks for each class on each layer which were then averaged so pixels that assuredly do not contain part of the number resulted in 0.5, which ultimately lead to 127.5 from the interpolation. Therefore, scaling those pixels down to 0 plus some grace for any straggling gray pixels resulted in the "true" fake output of the SegNet autoencoder. Figure 5 displays an example of the predicted image, lower bound, and upper bound images for the SegNet output after thresholding, which are analogous to the classifier input images of the other two tasks.

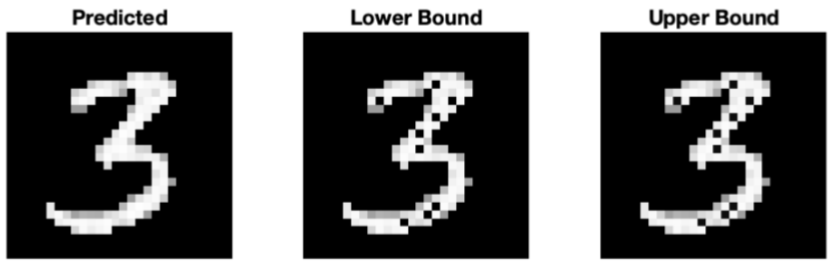

Figure 5. Example of SegNet images after interpolation and thresholding. These classifier input images are also analogous to those for the VAE and the baseline.

## 3 Results

For all three sets of images, the ImageZonotope set yielded inconclusive results about the robustness of the classifier. Not one zonotope result showed a robust error bar; all of them overlapped for every input image. On the other hand, the ImageStar and abstract polytope sets had almost identical results and were indicative of the classifier's robustness. Hence, their error charts were used for the analysis.

Both the VAE and SegNet generated surprisingly convincing fake images, for their relatively short training times, often "tricking" the classifier into classifying them as the correct class as if they were the real image. When no attack occurred, the classifier was less robust against fake images produced by the VAE than those of SegNet after thresholding. On the other hand, the classifier's robustness against fake images from SegNet was on par with that of the baseline when no attack occurred.

MATLAB repeatedly hung during attacks on the baseline; unfortunately, no results were obtained after the fourth digit for the 1% attack on the baseline to offer a complete comparison with the autoencoders. Considering only the first four digits of the 1% attack, the classifier was more robust to SegNet fake images than the baseline images and the classifier was more robust to VAE images than those of the baseline and SegNet. Comparing VAE images and SegNet images on the 1% attack, the classifier's robustness was similar for both, but it favored different classes for each.

For the 5% attack, the classifier was ostensibly less robust against the VAE images compared to the SegNet images. For both sets, the more intense attack resulted in worse performance, as expected. Additionally, the attack resulted in more overlapping error bars, suggesting that the model was confounded and could not recognize the input with confidence. Table 1 summarizes the results of these attacks on the three different sets of images fed to the classifier.

*All error charts for the baseline, VAE, and SegNet images are included as artifacts with the submission.*

Across all three methods, the classifier was seemingly most robust to the digit '3'. This trend is further confirmed by their narrow error bars for the attacks; other classes for which the classifier is less robust typically had much wider error bars. Figure 6 displays the error bars for the three methods from the 1% attack for the ImageStar set.

Table 1. Summary of results per class for baseline, VAE, and SegNet images passed to the classifier. - = robust, {0,1,2,3,4,5,6,7,8,9} = not robust, misclassified to digit, ? = not robust, uncertain, overlapping ranges, ~ = no data.

| | Baseline | | | | | | | | | | VAE | | | | | | | | | | SegNet | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| No attack | - | - | - | - | - | - | 2 | - | - | - | - | 4 | - | - | 9 | - | 4 | - | - | - | - | - | - | - | - | - | - | 2 | - | - |
| 1% attack | ? | ? | - | - | ? | ~ | ~ | ~ | ~ | ~ | - | 8 | - | - | - | ? | ? | 2 | - | ? | 3 | 7 | - | - | - | - | ? | ? | ? | - |
| 5% attack | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | ~ | 8 | ? | ? | - | ? | ? | ? | ? | ? | ? | ? | ? | - | - | ? | - | 4 | ? | ? | ? |

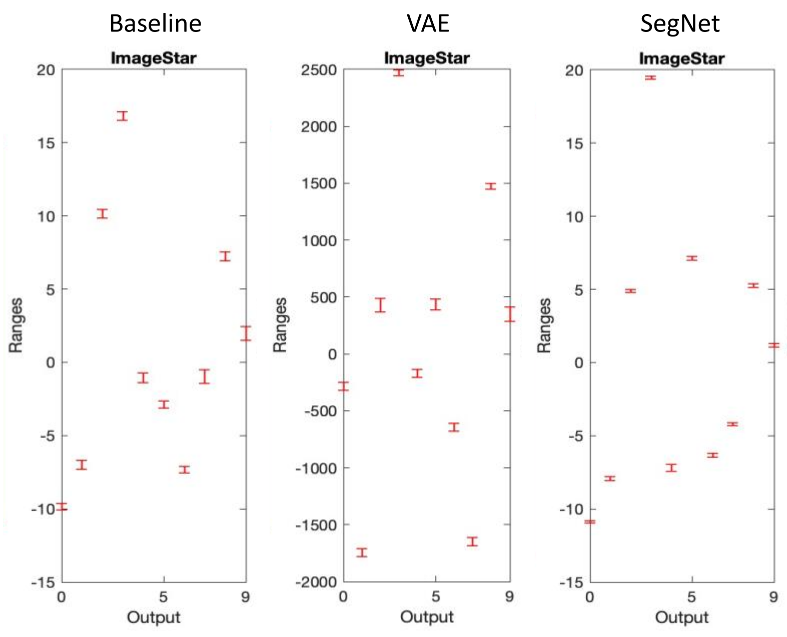

Figure 6. Error bars for the digit ‘3’ for each method for the ImageStar set for the 1% attack. The classifier was distinctly robust to this class for the three types of images, for both attacks. Compared to other classes, the error bars for the ‘3’ class are very narrow for an attacked image.

Attempts to directly verify the robustness of the classifier resulted in uncertainty when approximating the output set. Computing the exact set to verify robustness was not feasible given the CPU constraint, preventing the acquisition of counter examples for analysis.

# 4 Discussion

## 4.1 Takeaways

Given the contrast between the ImageZonotope and the ImageStar sets in yielding conclusive results, these findings suggest that NNV’s ImageStar implementation is the superior representation of image input sets for CNNs. Interestingly, the abstract polytope set performs almost identically so perhaps, at least for this problem, it is as informative as the ImageStar.

Also demonstrated in the results, the classifier is less robust in the face of attacks and becomes even more confused as the percentage of attacked pixels increases. Even as few as 1% of the pixels in a 784-pixel image can noticeably confound a DNN and hinder its performance, even for a DNN as trivial as a feed-forward classifier. These findings corroborate known shortcomings of DNNs and justify the need for improvements to deep learning itself and deep learning verification to qualify deployed DNNs.

The most important takeaway from these results is that the classifier is as robust against the generated fake images from two relatively simple autoencoders as it is against the raw input images that it was originally trained on. This finding is interesting because the fake images are obviously different from their associated real images, as shown in Figures 2 and 4. Yet, dropping the brightness of only the top 1% of an image's pixels can confuse the classifier and even make the reach set computation more challenging. Justifying further investigation into generator network verification, a method to find the reach set of these autoencoders alongside the classifier would hopefully illuminate more about the underpinnings of the autoencoder. Such information would help to not only understand how an autoencoder can trick a classifier with an imperfect example, while lightly changing brightness can stupefy it, but also improve the fake output of the autoencoder to truly recreate the real input.

## 4.2 Considerations

The input set representation is important to consider because if it does not align well with the set expected by the model then it will not elucidate any useful information about the robustness. The inconclusive results of the ImageZonotopes for every experiment evinces this aspect of DNN verification. Another aspect of this consideration is likely evidenced by the occasional hanging for certain attacks on the images. The activations for the attacked images are likely not as strong and causs the ReLU to receive values that are not as clearly separated as those that occur for a recognizable digit.

Another important consideration is the randomization factor. Relative to most statistical analyses, only a few dozen reach sets were computed for each of the of three tasks. Therefore, the results obtained may be a non-representative subset of the possible results. However, these models are rigid in this context so they should yield the same output given the same input. Since the MNIST digits are standard, it is likely that these are representative results.

# 5 Future Work

## 5.1 Short-Term

To proceed with future work, a GPU will be necessary. Immediate future work includes a true statistical battery on the robustness of the classifier for each of the cases and attempting to complete the reachability analysis of attacks before prediction on the real images and attacks on the real images fed directly to the classifier for the baseline. However, this goal for future work assumes that a GPU could compute these tasks in a reasonable amount of time, yet they could be nearly intractable, if not entirely.

## 5.2 Long-Term

Long-term future work includes extending NNV to support autoencoders, or CNN objects that can verify transposed convolutional layers and max unpooling layers, beyond the segmentation task. After this extension, solving the concatenation of two separately trained DNN components into a single network would allow the verification of the entire network, instead of the CNN only. In this project, each autoencoder and the classifier were separately trained, which unfortunately prohibited composition into a single SeriesNetwork. This problem is likely related to MATLAB programming, not NNV, and may potentially already be possible, despite the limited read-only functionality of MATLABs deep learning API.

# 6 References

[1] X. Huang *et al.*, “A Survey of Safety and Trustworthiness of Deep Neural Networks: Verification, Testing, Adversarial Attack and Defence, and Interpretability ∗,” *arXiv*, pp. 0–94, 2018.

[2] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer, “Algorithms for verifying deep neural networks,” *arXiv*, pp. 1–126, 2019, doi: 10.1561/2400000035.

[3] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–14, 2015.

[4] H. D. Tran *et al.*, “NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12224 LNCS, pp. 3–17, 2020, doi: 10.1007/978-3-030-53288-8_1.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Ha, “LeNet,” *Proc. IEEE*, no. November, pp. 1–46, 1998.

[6] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *2nd Int. Conf. Learn. Represent. ICLR 2014 - Conf. Track Proc.*, no. Ml, pp. 1–14, 2014.

[7] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, 2017, doi: 10.1109/TPAMI.2016.2644615.

[8] A. A. Alemi, B. Poole, I. Fische, J. V. Dillon, R. A. Saurous, and K. Murphy, “Fixing a broken elbo,” *35th Int. Conf. Mach. Learn. ICML 2018*, vol. 1, pp. 245–265, 2018.

[9] H. D. Tran *et al.*, “Star-based reachability analysis of deep neural networks,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11800 LNCS, pp. 670–686, 2019, doi: 10.1007/978-3-030-30942-8_39.