LEARNING-BASED DENOISING WITH ADVERSARIAL NOISE

Dylan Plummer
dmp131

Kristen Hauser
kxh514

February 10, 2022

1 Introduction

All measurement devices are susceptible to random noise which can obscure the true signal that we are interested in. Sometimes the noise is independent of the true underlying signal such as white noise in an audio recording and sometimes the noise depends on the signal itself such as photographs in which low-light regions of an image are typically noisier than regions of higher intensity. This work focuses on using machine learning methods to remove different sources of noise from images. While we only focus on images, the theoretical frameworks applied can generalize to different signals as well.

Recent work in learning-based denoising methods has shown impressive performance across a wide variety of noise distributions. In this project, we analyze the performance of some of these methods and attempt to show that while they can perform quite well, they are also susceptible to adversarial attacks just as many other machine learning methods are. By attacking these denoising methods with adversarial examples we hope to gain a better understanding of how these methods actually learn to remove noise. It is also possible that the adversarial examples can give us insight into potential improvements to learning-based denoising methods after seeing where they fall short.

Since adversarial attacks on learning based denoising methods are not a well studied problem, we first focus on understanding how adversarial attacks are formulated and their suspected causes in the well studied problem of image classification. We suspect that the techniques and suspected causes will translate from the image classification problem to image denoising since both fields utilize neural networks. Finally, all code used in this project is provided for reference[1].

2 Adversarial Examples

This portion of the final project was completed by Kristen Hauser and focuses on understanding on how neural networks can fail. Specifically in case of adversarial examples, I am to answer what adversarial examples are and how they operate on current machine learning models. This portion is split into three sections, the first to define what an adversarial example is and two common methods of generating adversarial examples. The second section to examine current hypotheses on why adversarial examples exist. The third section is an experimental section to investigate how noise and adversarial examples can make a classifier fail. Two methods of generating adversarial examples discussed in the second section will be implemented and the impact of the adversarial examples on the classification network will be visualized.

Several tools are used throughout this section, the primary tools and/or packages used in the implementations are: PyTorch, NumPy, Plotly. PyTorch was used to train and models, was the source of the MNIST digit data set and was used as a source for distributions (Gaussian) to be applied to data. NumPy was the underlying data structure mechanism that was used between PyTorch and Plotly. Plotly was the visualization library used throughout.

[1] Code for this project can be found at: https://github.com/hauserkristen/adversarial_denoising

2.1 Background

Adversarial machine learning is a technique in the field of machine learning which attempts to fool or manipulate models. There are four general types of attacks: Evasion, Extraction, Poisoning and Logic Corruption. A common theme within adversarial machine learning is an adversarial ex ample which is defined as an input from the data set with a small but intentional worst-case perturbation which results in the model outputting an incorrect answer [1]. Adversarial examples fall under the evasion attack category and have been mostly studied with respect to the machine learning task of image classification. This portion of the project focuses on trying to understand how adversarial examples fool current image classification models.

Two methods for generating adversarial examples will be implemented in this project. The first is an early and computationally efficient method known as the fast gradient sign method (FGSM) [1] while the second is a more recent attack known as the One Pixel Attack. These two methods are described in more detail below. These methods were chosen due to their popularity and difference in technique. The FGSM is a white box method while the One Pixel Attack is a black box method. These methods refer to the amount of knowledge of the model that the attacker has:

- White Box - In a white box attack, the adversary has total or some knowledge of the model (ie. model structure, model weights, algorithm, initialization state, etc.) and/or data used.
- Black Box - In a black box attack, the adversary has no knowledge of the model itself but may be privy to outputs such as confidence scores, besides the final classification or value.

2.1.1 FGSM

The fast gradient sign method is one of the first adversarial example generation methods, as shown in Equation 1 where θ are the parameters of the model, x is the input to the model, y is the target class, J is the cost function used to train the model and ϵ controls the amount of perturbation to apply to the input.

$$\eta = \epsilon \cdot sign\big(\nabla J(x, y, \theta)\big) \tag{1}$$

For an image classification task, this method modifies images by adding η to produce high confidence misclassification. This method was able to fool Google's GoogLeNet which is rather complex model consisting of a 22-layer network [2]. One example of the application of this method can be seen in Figure 1.

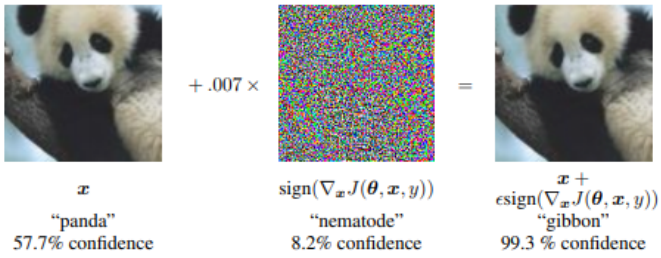


Figure 1: FGSM Acting on GoogLeNet

This method was shown to produce an error rate of 99.9% on the MNIST digit data set with an average confidence of 79.3%. Similar results were shown for the CIFAR-10 data set.

2.1.2 One Pixel Attack

The One Pixel Attack focuses on minimizing the number of pixels modified within an image to create these misclassifications. This method is based on differential evolution which is an optimization technique that tries to improve a candidate solution (in this case, pixel and its value) in an interative manner with respect to a measure of quality. One example of the application of this method can be seen in Figure 2.

Figure 2: One Pixel Attack Acting on Three Classic Image Classification Models

Through this method, it has been shown that 67.97% of images in the CIFAR-19 and 16.04% of images in the ImageNet test data sets can result in misclassification by modifying one pixel.

2.2 Explanations in Literature

When adversarial examples were first proposed in 2013 by Szegdy et al. [3], the original cause was unknown but was suspected to be due to the highly complex, non-linear model and/or overfitting. This was shown not to be the case because it was expected that these pockets would vary with random seeds but it was shown that these pockets remained the same and even resulted in the same or similar output. One of the first hypotheses was that adversarial examples stemmed from model linearity. Recently, another hypothesis has become widely popular that suggests adversarial examples are a product of non-robust features. Both of these hypotheses will be discussed in detail in the following sections.

2.2.1 Model Linearity

Goodfellow et al. [1] argued that model complexity nor overfitting explained that adversarial examples exist for linear models. Since precision of features is limited, such as 8 bits per pixel for images, additional information is discarded for input features. Define a linear model s.t. $y = w^T \cdot x$, they define an adversarial example as $\bar{x} = x + \eta$ where x was an existing data point. The authors continue to show given these definitions then the output of x should be the same as $\bar{x}$ as long as $||\eta||_\infty < \epsilon$ where epsilon is small enough to be discarded given the feature representation. With this formulation, considering the dot product:

$$\bar{y} = w^T \cdot \bar{x} = w^T \cdot x + w^T \cdot \eta$$

Goodfellow et al. go on to show that if w has n dimensions and the average magnitude of the values within w is m, then the input to the activation function will grow by $\epsilon \cdot m \cdot n$. It is important to note that $||\eta||_\infty$ does not increase as the dimensionality of the input increases, while the activation input grows linearly with n. This allows negligible changes to be made for each dimension of the input that can result in a significant change to the output. This work shows that when w has high dimensionality even a linear model can result in adversarial examples.

The authors go on to discuss more complex neural networks. Deep neural networks are non-linear by definition if their activations are not non-linear. Some common structures and configurations of neural networks include ReLU (Rectified Linear Units), LSTMs (Long-Short Term Memory) and maxout networks. These networks components and structures were designed to behave in a highly linear manner to reduce diminishing/exploding gradients and reduce complexity to optimize. Other common utilities are the activation function of sigmoid and hyperbolic tangent. These activations are piece-wise highly linear and are displayed in Figure 3.

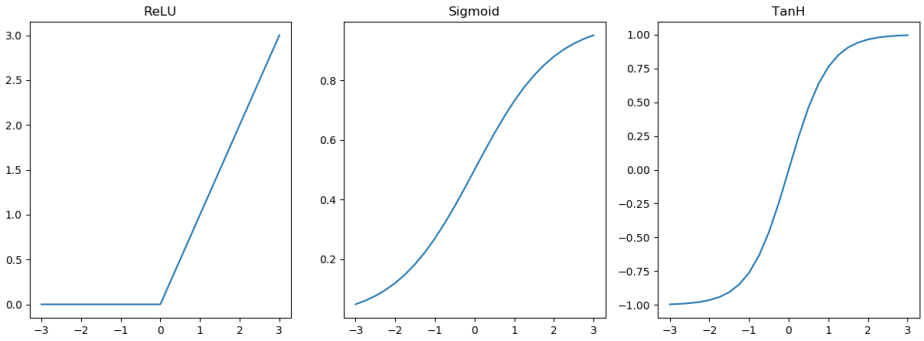


Figure 3: Common Activations that are Piece-Wise Linear

These linear attributes lead the authors to apply the linear perturbation described above to non-linear models which resulted in the FGSM attack described in the previous section. This method was found to be very successful which supports the hypothesis that adversarial examples, to some extant, are a product of the linear attributes of common features used within non-linear models.

A final note regarding the mode linearity hypothesis is that DeepMind published some research[4] that found that training on adversarial examples caused networks to behave more linearly. This contradicts the hypothesis proposed by Goodfellow et al., however no alternative explanation were provided.

2.2.2 Non-Robust Features

Ilyas et al.[5] argued that adversarial examples are a product of non-robust features. Non-robust features are defined by the authors as "features (derived from patterns in the data distribution) that are highly predictive, yet brittle and (thus) incomprehensible to humans." First the authors define a set of features $\mathcal{F}$ that is comprised of features (f). Each feature is a mapping from an input space ($\mathcal{X}$) to a set of real numbers ($\mathbb{R}^n$). They further assume all the features are normalized to have a mean of 0 and unit variance. They further define the three key feature types that are used within their framework:

- ρ-Useful Features - For a given distribution D, the author refers to a feature f as ρ-useful if it is correlated with the true label in expectation.
- λ-Robustly Useful Features - Assuming a feature, f, is ρ-useful, f is defined as a robust feature if, under adversarial perturbation, f remains λ-useful.
- Useful, Non-Robust Features - A useful, non-robust feature is a feature which is ρ-useful but is not a λ-robust feature.

Examples of these features are visually depicted in Figure 4 which is from the poster of the paper previously cited. These non-robust features are what are manipulated to produce adversarial examples. The authors show that if the only non-robust features are used to train a classifier then the accuracy of adversarial examples (generated by various methods) is minimal (0%). Furthermore, training with only robust features also significantly reduces the accuracy of adversarial example to 50%. This result suggests that standard training on only robust features will produce a robust model.

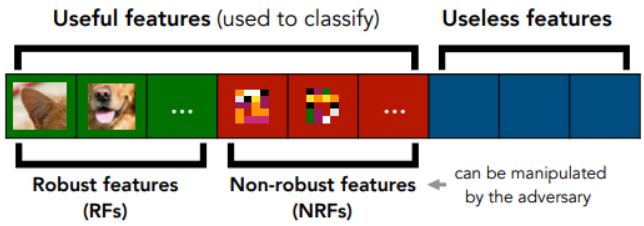


Figure 4: Example of Different Features

The authors further investigate non-robust features and methods to produce robust models. They conclude that in order to obtain models that are robust and interpretable to humans, that some amount of prior human knowledge is required to be explicitly incorporated into the training process.

2.3 Experiments on MNIST Digit Data Set

The experiments focus on the MNIST digit data set which classifies hand written digits as values zero through nine. A single classifier was trained for examination of the MNIST data set. The structure for the convolutional neural network was found within the PyTorch examples for the MNIST digit data set which can be seen in Figure **??**. This network architecture can be separated into convolutional and classification portions. This convolutional portion consists of two convolutional layers that both learn 5-by-5 filters, however the depth of each filter varies. The input of the first layer assumes one channel since the MNIST digit data set is grayscale and outputs three channels since 3 filters 5x5x1 are applied. The second layer transforms the 3 channels into 5 channels by applying 5x5x3 filters.

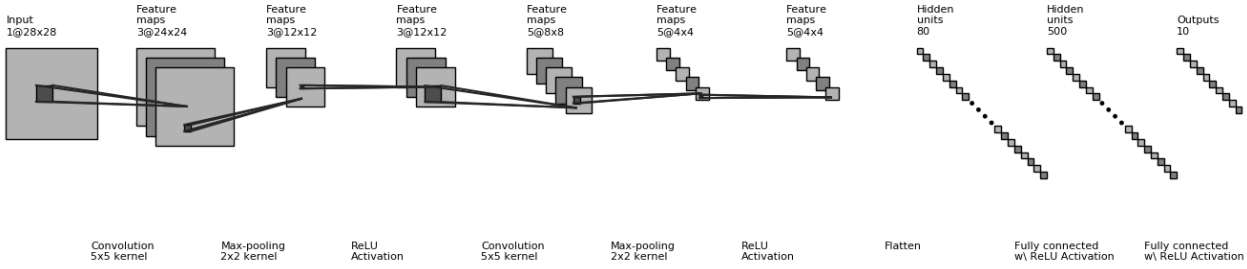


Figure 5: MNIST Classifier Network Structure

The PyTorch example utilizes ReLU activation and max pooling after each convolutional layer. ReLU is a commonly used activation function depicted in a previous section (Figure 3). Max pooling is a downsampling technique that chooses the maximum value from a subset of the image to represent that subset. In this architecture, 2x2 max pooling is used thus reducing the image size by 2 in each dimension. The second portion of the network is used for classification. The output from the convolution portion is flattened and fed into 2 sequentially full connected layers each utilizing ReLU activation. The final layer, commonly referred to as logits, is used to produce probabilities for each class through the Softmax function.

The learned filters from the convolutional neural network are visualized below in Figure 6. It is important to note that the color scale is the same across filters within a layer but is not consistent across the two layers. Furthermore, in the scale used throughout this paper, higher values are depicted as white while lower values are depicted as black.

(a) Layer 1 (1 Input Channel, 3 Output Channels)

(b) Layer 2 (3 Input Channels, 5 Output Channels)

Figure 6: MNIST Classifier Convolution Filters

2.3.1 Affect of Noise

When examining the affects of classic noise, two noise types were considered: Gaussian and Salt-And-Pepper. Each of these noise types were applied to a percentage of the pixels within an image. A uniform random distribution was used to select the pixels modified for each image. Then a Gaussian sample or Salt-And-Pepper value was used in place of the raw image value. The Gaussian distribution parameters used were $\mu = 0.5$ and $\sigma = 0.2$. These values were chosen so 98.76% of the samples fell within $[0, 1]$. All values outside of that range were clamped to 0 or 1. Salt-And-Pepper noise values were 0 or 1, and were chosen using a uniform random distribution.

Accuracy of the test data set was measured as the percent of noise applied to the image was increased. Both noise types had similar trends while Guassian noise showed a reduced accuracy of up to 8.6% compared to Salt-And-Pepper noise at a given replacement percentage. After 20% of the image had been replaced with noise, the degradation in accuracy became more significant as more pixels were replaced. Salt-And-Pepper noise had less of an effect since the majority of the image was black or white. Thus an image may have had not as many pixels modified if a white pixel was selected and was replaced with a white pixel.

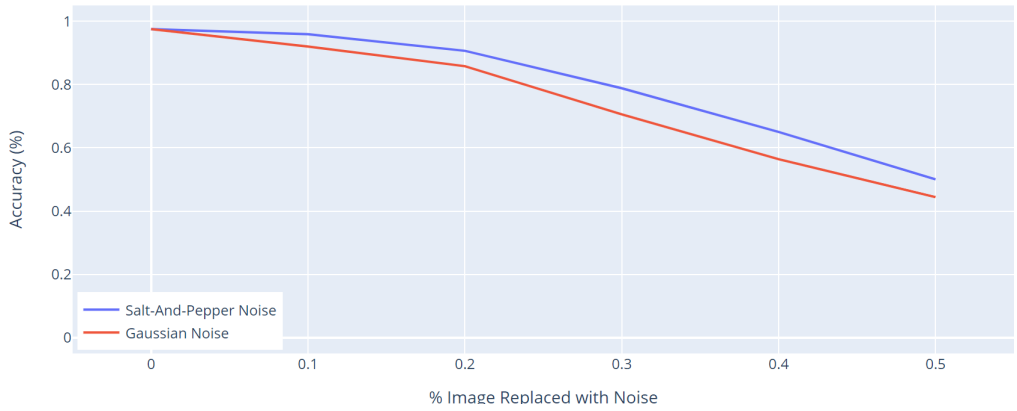


Figure 7: Affect of Noise on MNIST Classifier Accuracy

The remaining portion of the section focuses on examining specific examples from the MNIST digits test data set where the clean image was classified correctly while the noisy image was misclassified. The most interesting case seemed to be when minimal noise was able to change the label, so for these examples 10% (78 pixels) noise was added to the images. The goal is to try to understand visually why these images are being misclassified. To do this, the images were examined as they pass through each convolutional layer of the network and the resultant feature maps are shown.

One example for each noise type was examined and the examples selected are shown below in Figure 8 were chosen. These examples were chosen due to the similarities the clean image had to the misclassified digit class that resulted from the noisy image. The clean images were labeled 5 (76%) and 9 (98%) respectively while the noisy images were labeled 8 (85%) and 4 (65%) respectively.

(a) Gaussian (b) Salt-And-Pepper

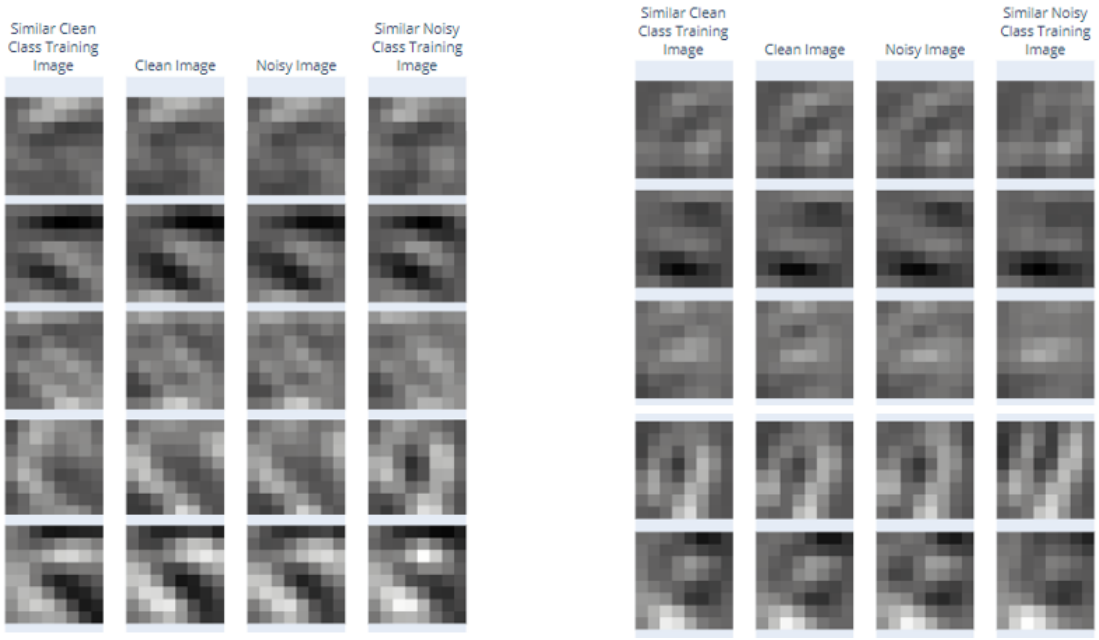
Figure 8: Noise Examples

The feature maps were examined after each layer prior to max pooling and activation being applied to see how the noise is directly affecting the convolution operation. The feature maps from the layer showed to be more useful in understanding how the features were being altered. The first layer feature maps can be found in the appendix for all examples in this paper. After the examples were chosen, similar data points within the training data set were identified based on the similarity of their feature maps after the second layer. A similar training data set point was chosen for each example, clean and noisy, and these are shown below in Figure 9.

(a) Gaussian (b) Salt-And-Pepper

Figure 9: Similar Training Data Set Images Based on Second Layer Feature Map

Then the feature maps after the second layer are shown for each of the images above are shown in Figure 10. The two center columns represent the feature maps of the clean and noisy images, while the first and fourth column represent the feature maps of the similar images. The first column represents the training data set image that was most similar to the clean image within its original class. The fourth column represents the training data set image that was most similar to the noisy image within the noisy labeled class. Differences between the two center columns of each example have negligible differences since only a few pixels had been altered. However, it was very interesting to find that the clean training images were so similar with regards to their feature maps.



(a) Gaussian (b) Salt-And-Pepper

Figure 10: Feature Maps After Second Layer For Noise Examples

For the Gaussian noise example, the clean training images are most notable different in the last two representations (rows within the grid). For the third feature map, the clean and noisy images align more with the correct label training example than the misclassification label training example. However, the noisy image aligns more with the fourth feature map of the misclassification label training example.

For the Salt-And-Pepper noise example, the first three images are extremely similar in their feature maps. While there are significant similarities between the misclassification label training example's feature maps, there are some clear differences. The first feature map lacks the diagonal striations that can be seen in the first three. Additionally, in the fourth feature map, the horseshoe shape curve is broken into two diagonal lines.

For the Gaussian noise example, I believe the noisy image was misclassified due to the somewhat diagonal pattern of noise that could be seen as the missing cross for an 8, as annotated in Figure 11. The convolutional operation can potentially identify that as a diagonal feature which would push the classification toward 8, which was seen.

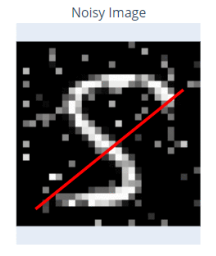


Figure 11: Annotation for Likely Structure for Gaussian Example

As for the Salt-And-Pepper noise example, I believe the noisy image was misclassified due to the general similarities between 9s and 4s. While looking for examples, a common error for the classifier was when 9s were classified as 4s. This error existed even without noise. Two types of fours exist in the data set, closed topped and open topped fours. Closed topped fours are similar to 9s, even furthermore when the loop of the 9 is flattened and highly angled like in this example. The addition of the noise did not seem to change the feature maps significantly visually but was enough to reduce the confidence in the original label.

In general it seems that by the nature of convolution, altering a single point can significantly impact the resultant value of a region since its value influences into neighboring values. This leads to the concept of a OnePixel attack. If 10% random noise, can reduce accuracy of a classifier by up to 5.5%, than targeted pixel modification could be as effective at a lower scale.

2.3.2 Affect of Adversarial Example

The two adversarial generation method's affect on accuracy is examined in this section. For the FGSM curve, the x-axis is the parameter ϵ which controls how much of the gradient calculated is applied to the images. Accuracy drops below 70% when ϵ 0.1. This shows that these target noise affects accuracy very effectively. For the One Pixel curve, the number of pixels modified is along the x-axis. With modifying one pixel, the accuracy was affected by 1.5%, however when modifying 40 pixels accuracy was able to be decreased by 12.5%. The random noise adjusted 78 pixels with only a decrease in accuracy of 5.5%. As expected the effectiveness of the OnePixel attack is an improved random noise attack. My results agree with the original OnePixel paper, that eluded to the MNIST digit data set being minimally effected by the OnePixel attack.

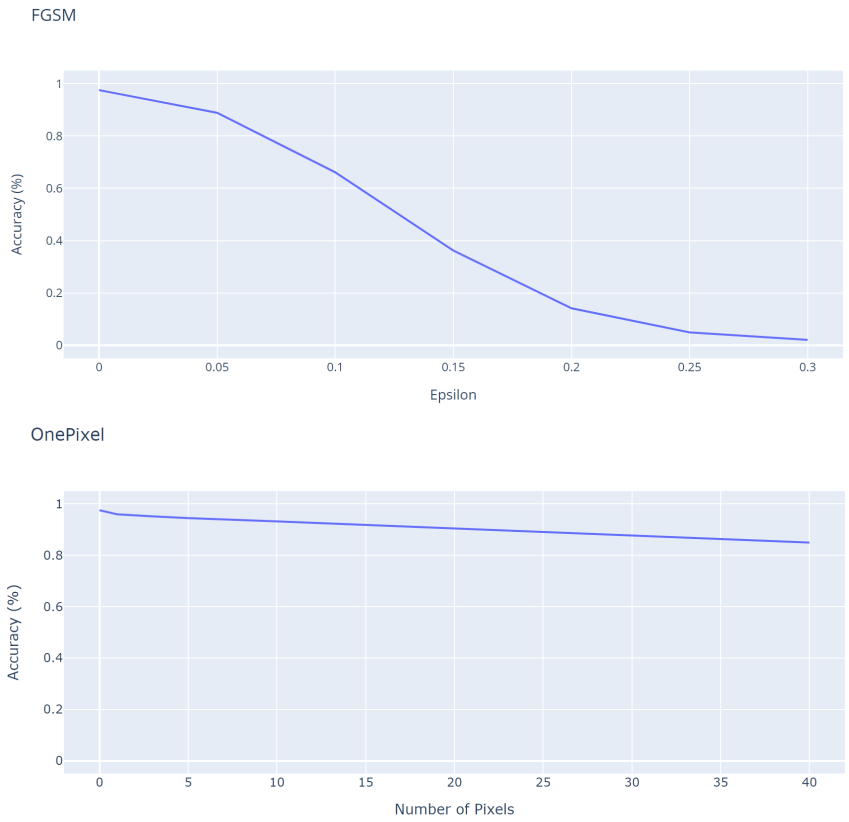


Figure 12: Affect of Adversarial Examples on MNIST Classifier Accuracy

The remaining portion of the section focuses on examining specific examples from the MNIST digits test data set where the clean image was classified correctly while the adversarial example was misclassified. The same methodology used for the noisy examples will be followed. However, only the FGSM method will be examined since the OnePixel attack is highly similar to a restricted Gaussian noise example. When looking for an example, a common theme across all the adversarial examples was that in general a large portion or majority of the white of the digit was darkened. I believe this was a general effort to lessen the distinguishable features across the image. In Figure 13, the similar training images are shown compared to the clean and adversarial images. This image was chosen as an example since the portions of the nine that were darkened can clearly be seen as an effort to lessen the lower loop of the nine. Without the lower loop, the nine looks more like the seven from the training data set. The clean image was labeled 9 (83%) while the adversarial images were labeled 7 (99%) respectively.

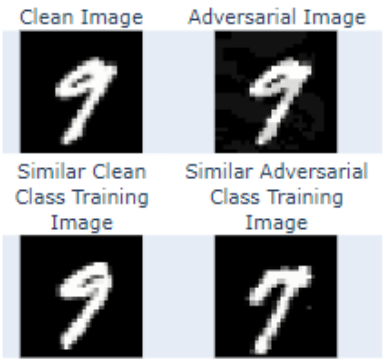


Figure 13: Similar Training Data Set Images Based on Second Layer Feature Map For FGSM Example

It is interesting to note that the differences between the all of the feature maps shown below in Figure 14 are extremely similar and almost create a gradient effect across a row. The major difference here lies in the second feature map focusing on the location of the darkest region.

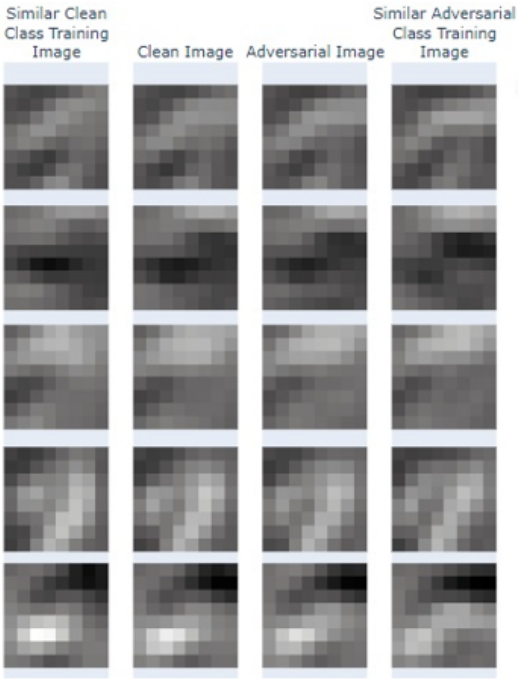


Figure 14: Feature Maps After Second Layer For FGSM Example

In general, the feature map differences between the clean and noisy images did not vary significantly visually even when the resultant classification changed notably. The difference between the first layer feature maps can be seen in Appendix A. These values were also computed for the second layer, and was never exceptional in magnitude. Even with a rather simple image classification task, minor modifications were able to change the classification even when visually the digit was able to still be recognized by a human. If time had allowed, I would have tried to classify the features as ρ-useful and λ-robustly useful as per [5]. However, with out this information, both hypotheses seem like plausible causes.

3 Learning-Based Denoising

This portion of the project was completed by Dylan Plummer and aims to construct learning-based methods for removing different types of noise from images. Specifically, the method that we are interested in involves learning to denoise images without access to clean data. Recent work in this area has shown that, based on certain statistical assumptions about the noise distribution, it is possible to learn to restore noisy images from corrupted observations only. In the following sections we will explore the theoretical background that explains why this is possible and explore some different noise distributions that we are interested in removing and eventually attacking via adversarial attacks.

3.1 Background

3.1.1 Risk Minimization

We will approach learning-based denoising methods probabilistically from the framework of Risk Minimization. This is a general definition for supervised learning problems where we have two spaces X and Y which could represent different data distributions. For example, the spaces of images and class labels, or the spaces of sentences and sentiment labels, or in the case of image denoising, the space of noisy images and clean, noise-free images. The goal of risk minimization is to learn a hypothesis function $h : X \to Y$ that maps any point in X to some corresponding point in Y coming from an unknown, underlying target function. We define the problem such that the hypothesis function is learned based on a training set of examples $(\mathbf{x}, \mathbf{y})$. These examples come in the form of (x_i, y_i) pairs where $x_i \in X$ and $y_i \in Y$ and y_i is the desired output that we want the hypothesis function to return when given x_i.

Risk minimization can be defined probabilistically by assuming that our training examples are drawn independently from the joint distribution $p_{(x,y)}(x, y)$ defined over X and Y. Then, we define a loss function $L(\hat{y}, y) = L(h(x), y)$ which takes the output or prediction from the hypothesis function $h(x) = \hat{y}$ and the corresponding true value y and computes some difference or distance between the two. Using this probabilistic definition we can define the risk with respect to our hypothesis function by taking the expectation of the chosen loss function.

$$R(h) = \mathbb{E}[L(h(\mathbf{x}), \mathbf{y})] = \int \int L(h(\mathbf{x}), \mathbf{y}) dp_{(x,y)}(\mathbf{x}, \mathbf{y}) \tag{2}$$

Once we have defined the risk, a learning algorithm is selected to search for the optimal hypothesis inside a set of functions referred to as the hypothesis space $\mathcal{H}$. This hypothesis space could be as simple as the class of all linear functions and it can be as complex as the class of neural networks. The optimal hypothesis is defined as the function which minimizes the risk:

$$h^* = \arg\min_{h \in \mathcal{H}} R(h) \tag{3}$$

Since the joint distribution $P(x, y)$ is the unknown distribution that we wish to model, it is not possible to compute the risk directly. Instead, we use our training set $(\mathbf{x}, \mathbf{y})$ to approximate the risk. This is known as empirical risk minimization:

$$R_E(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(\mathbf{x_i}), y_i) \tag{4}$$

This approximation allows us to redefine risk minimization as an optimization problem which, under certain assumptions about our training set and hypothesis space, will find an approximation to the true target function:

$$h^* \approx \hat{h} = \arg\min_{h \in \mathcal{H}} R_E(h) = \arg\min_{h \in \mathcal{H}} \left[\frac{1}{n} \sum_{i=1}^{n} L(h(\mathbf{x_i}), y_i) \right] \tag{5}$$

3.1.2 Learning-Based Denoising

We can frame the denoising problem in the same form of Risk Minimization. Suppose we have noisy input images $\tilde{\mathbf{x}}$ and noise-free training targets $\mathbf{y}$. We assume that the noisy input images are all distributed according to their corresponding clean targets, $\tilde{x}_i \sim p(\tilde{\mathbf{x}} \mid y_i)$. Then we can apply the empirical risk minimization framework defined in Eq. 5 where our hypothesis class is convolutional neural networks.

Suppose we have a neural network f_θ parameterized by θ. Then the risk minimization framework defines the optimal parameters as the following:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(\tilde{\mathbf{x}},\mathbf{y})}[L(f_\theta(\tilde{\mathbf{x}}), \mathbf{y})] \tag{6}$$

Using the definition of conditional probability, we can rewrite Eq. 6 as the following:

$$\theta^* = \arg\min_\theta \mathbb{E}_{\tilde{\mathbf{x}}}[\mathbb{E}_{\mathbf{y}\,|\,\tilde{\mathbf{x}}}[L(f_\theta(\tilde{\mathbf{x}}), \mathbf{y})] \tag{7}$$

When we train a neural network by interpreting the loss function as negative log-likelihood, then we can think of a neural network trained using Eq. 7 as finding the most likely ground-truth image y_i given noisy image $\tilde{x}_i$. We are performing maximum likelihood estimation on a high dimensional distribution $p(y_i \,|\, \tilde{x}_i)$ which contains all ground-truth images which are consistent with the noisy image $\tilde{x}_i$. If we use L_2 loss in Eq. 7, then the model will learn to output the expectation of all consistent ground-truth images.

3.1.3 Noise2Noise

Noise2Noise [6] is a recent work in computer vision research that makes a simple but powerful statistical observation regarding image denoising and empirical risk minimization. Lehtinen et al are able to demonstrate that it is possible to learn to restore images using *only* corrupted examples. Without any a priori models of the noise, *Noise2Noise* performs just as well and sometimes *better* than training using clean targets.

The main observation behind *Noise2Noise* is that in expectation, the estimate learned by the model remains unchanged even if we replace the training target distributions $p(y_i \,|\, \tilde{x}_i)$ with arbitrary distributions with the same conditional expectation. Thus if we corrupt the training targets of a neural network with mean zero noise, then the learned parameters will remain unchanged.

Suppose now that we have corrupted training targets $\tilde{\mathbf{y}}$ such that $\mathbb{E}[\tilde{y}_i \,|\, \tilde{x}_i] = y_i$ for all i. Thus we can approximate the risk minimization problem defined in Eq. 7 using the following empirical risk minimization problem:

$$\theta^* \approx \arg\min_\theta \sum_i L(f_\theta(\tilde{x}_i, \tilde{y}_i) \tag{8}$$

We can take this observation a step further as well. The corrupted training target distribution does not need to be the same as the the clean training target distribution in expectation, but rather in any measure such as median or mode. While this requires a more rigorous exploration, this can likely be explained by a concept in high dimensional probability known as measure concentration. Measure concentration tells us that all reasonable summary statistics in high dimensions are roughly equivalent up to an independent constant. This means that we can use L_2 minimization to recover the expectation of all consistent clean images given noisy observations, L_1 minimization to recover the median of consistent clean images, and Lehtinen et al derive a "mode-seeking" L_0 minimization that allows us to recover the mode of consistent clean images. The figure below demonstrates these different minimization strategies applied to images corrupted with random impulse noise.

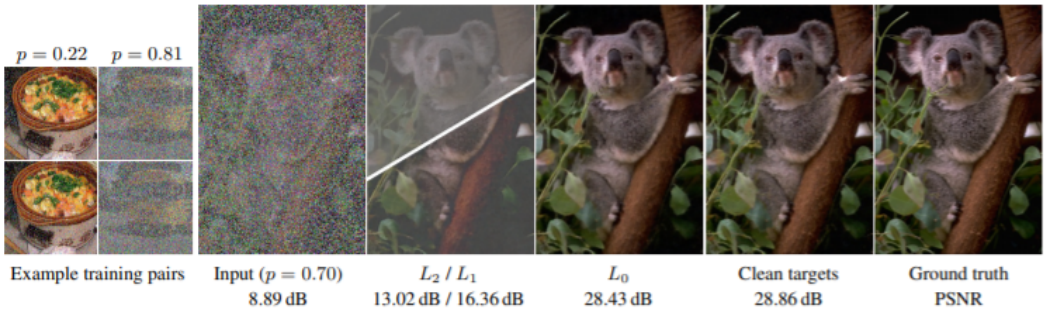


Figure 15: Noise2Noise demonstration of different minimization objectives applied to random impulse noise corruptions.

Random impulse noise is applied to each image by applying mean zero Gaussian noise to pixels with some uniform probability p. So overall, the noise is not mean zero and the corrupted image distribution is not the same as the clean

image distribution in expectation, but it is in mode. Thus in Figure 15, we see that the mode-seeking L_0 minimization recovers the true clean image while L_1 and L_2 minimization recover the clean image offset by some constant intensity. This difference is likely explained by measure concentration, though that theoretical analysis is beyond the scope of this report.

3.2 Experiments

The denoising experiments for this section focus on the CIFAR-10 dataset which contains 50,000 training images and 10,000 validation images from 10 classes. Each image is only 32x32 pixels which is certainly low resolution, however for this project it was not feasible to train all of our models on high resolution datasets due to time and hardware constraints.

We used a U-Net [7] model architecture identical to that of Noise2Noise. A U-Net is a fully convolutional autoencoder architecture with skip connections between each depth of the network. The encoder path of the network consists of blocks of convolutional layers followed by max pooling and the decoder path consists of upsampling layers followed by blocks of convolutional layers. The figure below illustrates the general architecture:

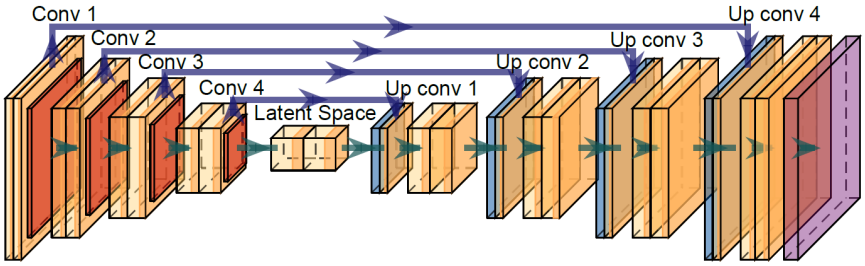


Figure 16: U-Net model architecture drawn using PlotNeuralNet [8].

We tested three different noise distributions: mean zero Gaussian, Poisson shot-noise, and random impulse noise. The following sections detail the differences between these noise distributions and how we modified the model training process to account for these differences.

3.2.1 Gaussian Noise

The first noise distribution that we trained a model to remove is additive mean zero Gaussian noise. To generate training target pairs we uniformly sample two standard deviations up to a maximum standard deviation set prior to training, then we corrupt each image with mean zero Gaussian noise based on these standard deviations respectively. This means that the model must first be able to estimate the noise level since all input images have different amounts of noise. The figure below shows an example of a corrupted training pair using a max standard deviation of $\sigma = 50$:

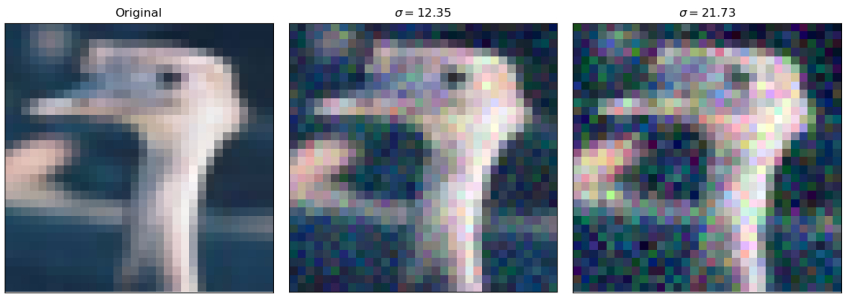


Figure 17: Gaussian training pair with maximum standard deviation of $\sigma = 50$.

We trained the model using L_2 minimization using mean-squared error until the validation loss stopped improving (~ 50 epochs) and evaluated the model on various examples from the CIFAR-10 validation set:

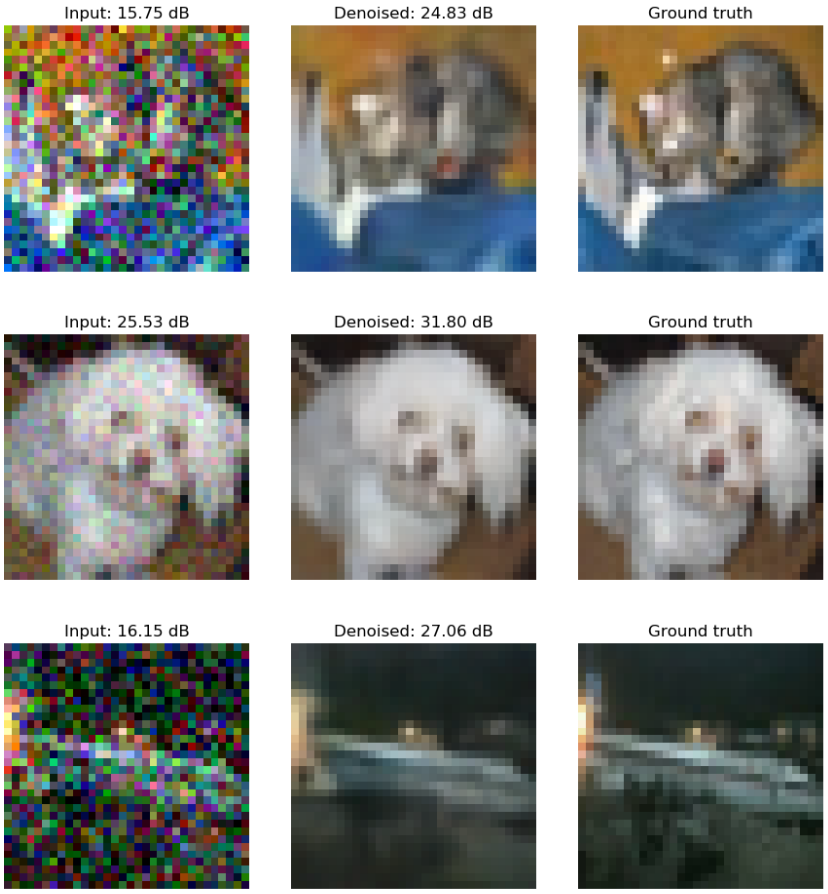
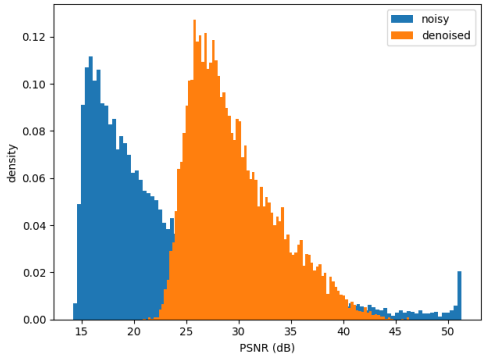


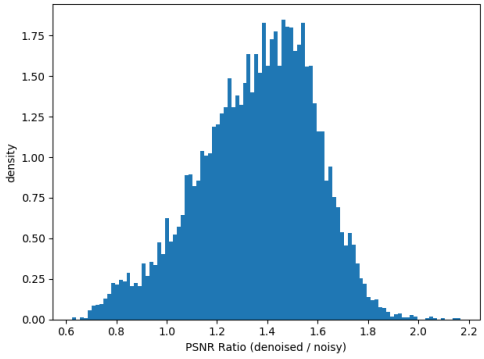
Figure 18: Gaussian denoising results with maximum standard deviation of $\sigma = 50$.

Overall the performance is quite good and nearly all of the noise is removed. However, it does seem that some detail is lost in the process such as the texture of the dog's fur in the middle example. We note that the model performs well regardless of the noise level, though based on the PSNR values lower noise levels make it much easier to recover a closer approximation to the ground truth.

We also evaluated the Peak Signal-to-Noise Ratio (PSNR) between both the noisy images and the denoised images with the ground truth clean images. This allows us to analyze the full distribution of denoising performance across all 10,000 test images. Based on the PSNR ratio histogram nearly all of the images in our test set are closer to the ground truth image after denoising.



(a) Histogram of PSNR values across test images.



(b) Histogram of PSNR ratio between denoised and noisy images.

3.2.2 Poisson Shot-Noise

The idea behind Poisson shot-noise corruptions is that natural images are captured by observing photons interacting with a camera setting. We can model the arrival of these photons as a Poisson distribution and model the average number of photons that are observed at each pixel. The fewer photons observed, the more noise is present in the image. This is analogous to simulating a low-light image where there are very few photons to observe at all. The figure below demonstrates the effect of different numbers of simulated photons per pixel:

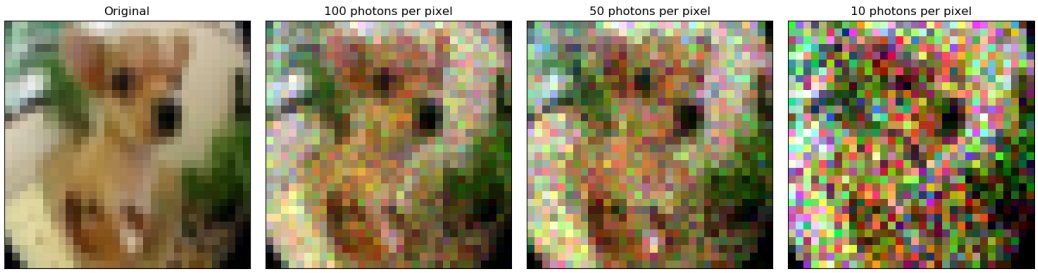


Figure 20: Poisson shot-noise as the average number of photons per pixel.

Similar to the Gaussian denoising training we wanted the model to have to first estimate the noise level before removing it, so prior to training we pick an average number of photons per pixel and we sample values around this average for each image in each training pair to vary the noise level in each example. To sample this value, we first tried a Poisson distribution however the values were too concentrated around the mean and we wanted more variation in the noise level. To get around this we used a negative binomial distribution which allows us to adjust the dispersion amount. The figure below illustrates the distributions that were sampled for the number of photons per pixel in an image:

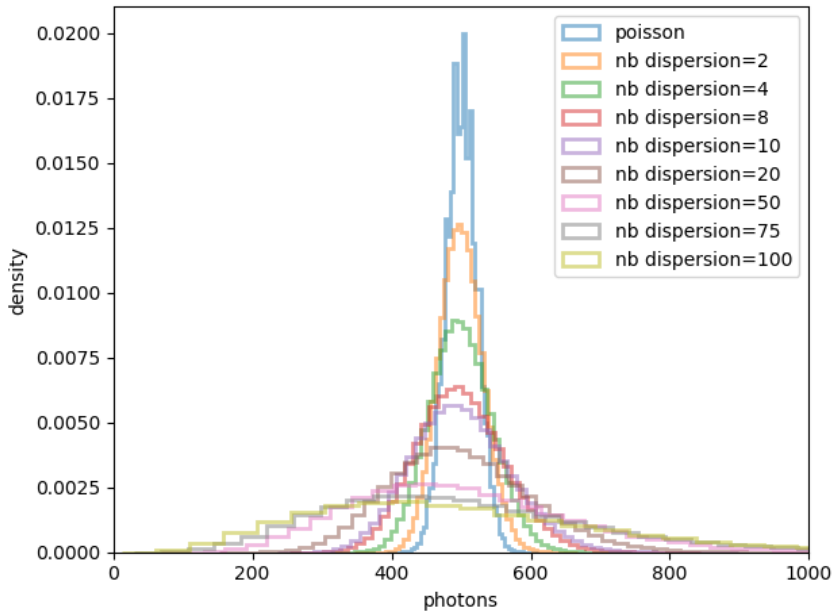


Figure 21: Distributions of a negative binomial with different dispersion parameters centered at 500.

After selecting an average number of photons per pixel, we independently draw a Poisson sample at each pixel to represent the number of photons detected at that location and then we map these photon counts back to RGB space. The figure below shows a training pair example:

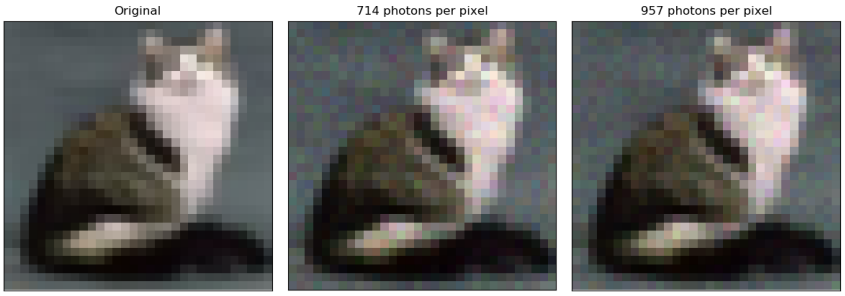


Figure 22: Poisson training pair with average photons per pixel sampled from negative binomial centered at 500 with dispersion parameter 75.

Since the Poisson noise that we applied to the images is not mean zero, we cannot expect that the corrupted image distribution is the same as the clean image distribution in expectation, but we do expect that it is the same in median. So for this model, we can train with L_1 minimization using mean-absolute error to recover the median of consistent images given noisy observations. The figure below demonstrates some examples of our Poisson denoising model in action:

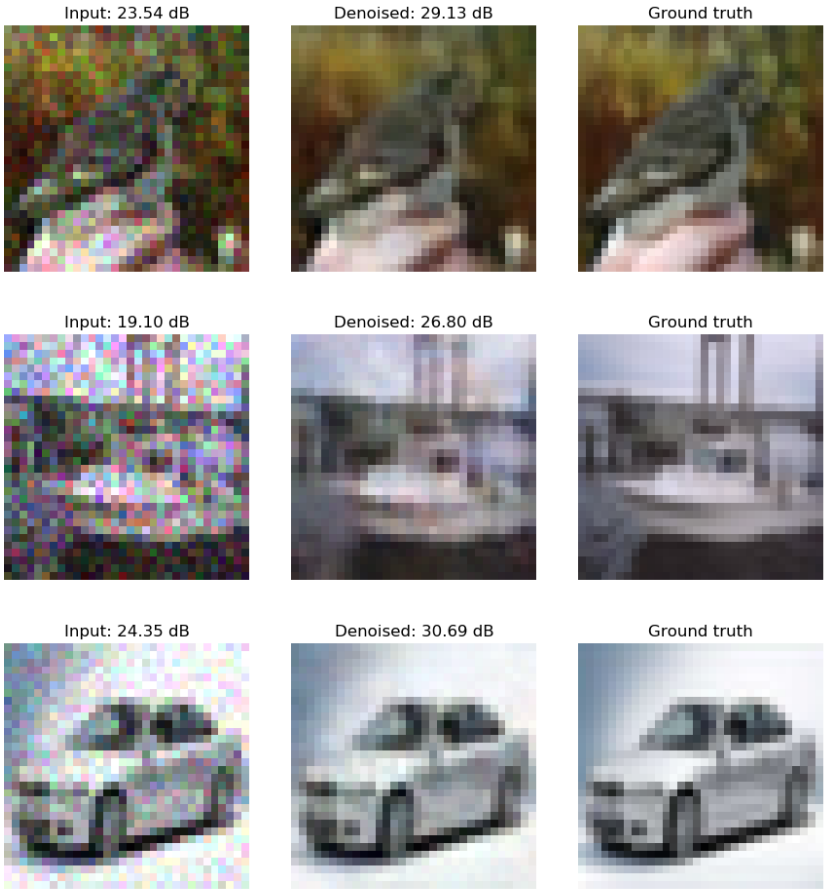
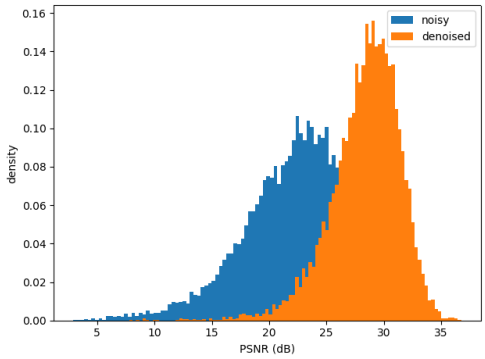


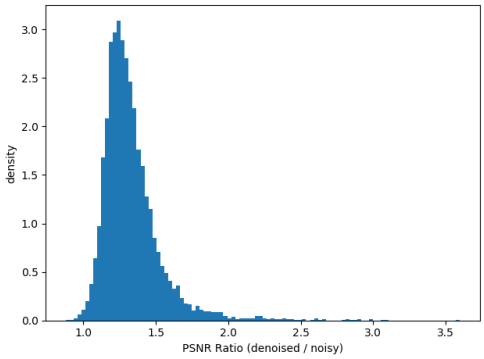
Figure 23: Poisson denoising results with average photons per pixel of 100.

Once again the noise is removed almost completely, though at the cost of some detail. This makes sense because there are many potential clean images that could explain the noisy image, each with slight variations on certain details. When

we learn to recover the median of these consistent clean images, we will likely recover the image with the least amount of detail variations.



(a) Histogram of PSNR values across test images.



(b) Histogram of PSNR ratio between denoised and noisy images.

3.2.3 Random Impulse Noise

Random impulse noise is identical to additive mean zero Gaussian noise except that it is not applied to every pixel. Instead, we only apply it to pixels with some uniform probability p. So if $p = 1$ we have the same Gaussian noise process, and if $p = 0.5$ then Gaussian noise is only applied to half of the pixels at random. This added condition means that the overall noise is no longer mean zero and thus recovering the expectation of all consistent clean images given noisy observations will not recover the true ground-truth image. Instead, we want the mode of the consistent clean images. This is because given a single pixel in an image, on average it is not corrupted and thus over the whole corrupted image distribution the mode of that pixel is the same as the mode of the pixel in the clean image distribution. To recover the mode, we use a mode-seeking L_0 loss.

The L_0 is defined as $(|f_\theta(\hat{x} - \hat{y}| + \epsilon)^\gamma$ where $\epsilon = 10^{-8}$ and γ is annealed linearly from 2 to 0 during the training process. We can quickly show that L_0 minimization approximates the mode. Consider a continuous density function for our data $p_y(y)$ and suppose we want to find the point x that minimizes the expected p-norm from points in the distribution:

$$x^* = \arg\min_x \mathbb{E}_{y \sim p_y} \frac{1}{p}|x - y|^p = \arg\min_x \int \frac{1}{p}|x - y|^p p_y(y) dy \tag{9}$$

We can find the minimum by differentiating and finding a root:

$$0 = \frac{\partial}{\partial x} \int \frac{1}{p}|x - y|^p p_y(y) dy = \int sign(x - y)|x - y|^{p-1} p_y(y) dy \tag{10}$$

Now if we take the limit as $p \to 0$, we can determine what is recovered by L_0 minimization:

$$\lim_{p \to 0} \int sign(x - y)|x - y|^{p-1} p_y(y) dy = \int sign(x - y)|x - y|^{-1} p_y(y) dy = \int \frac{1}{x - y} p_y(y) dy \tag{11}$$

Interestingly, the resulting expression is the same as the formula for the Hilbert transformation of $p_y(y)$ up to some absolute constant. So while L_0 minimization does not recover the exact mode which would be a root of the derivative of the density itself, it does recover a root of its Hilbert transformation which approximates the true mode up to an absolute constant.

Below is an example of a training pair using random impulse noise:

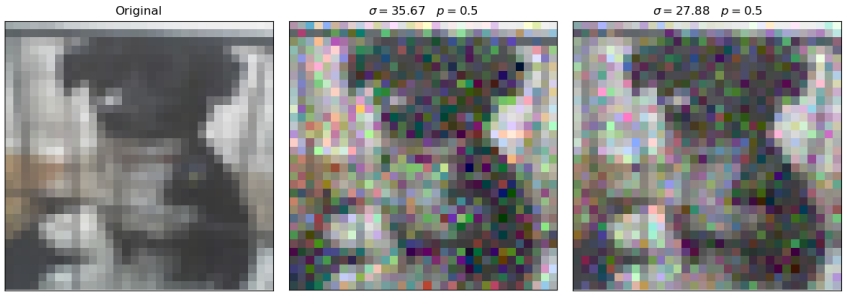


Figure 25: Random impulse noise training pair using a standard deviation of $\sigma = 50$ and a corruption probability of $p = 0.5$.

After training a model using L_0 minimization we mostly recover the ground truth images as illustrated below, though the loss of detail appears slightly more noticeable than the two previous models.

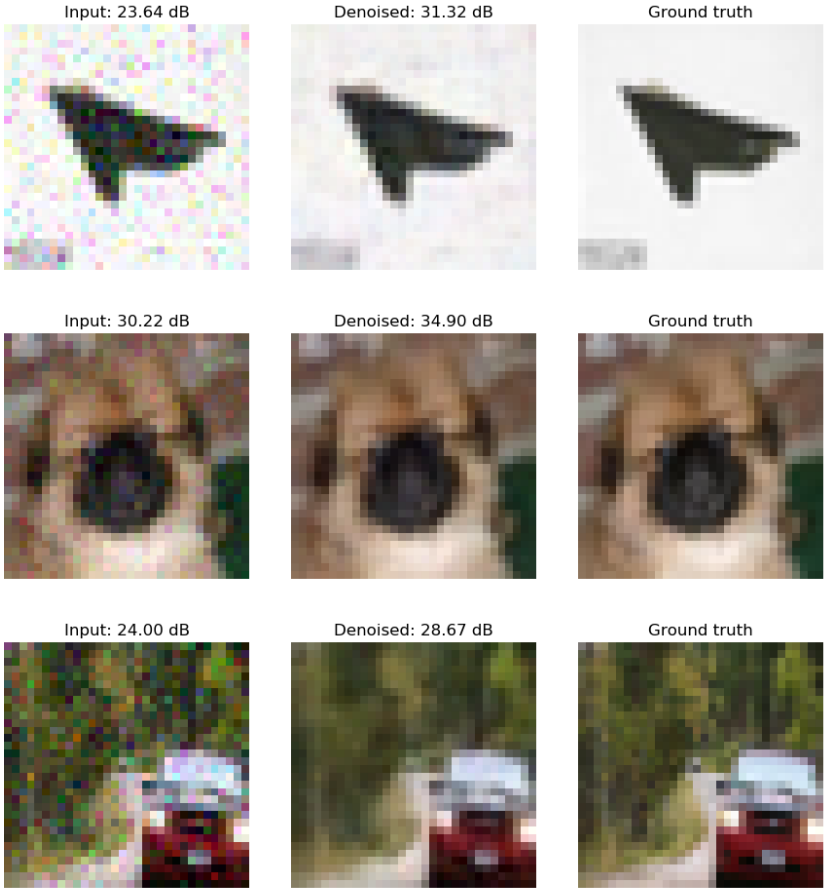
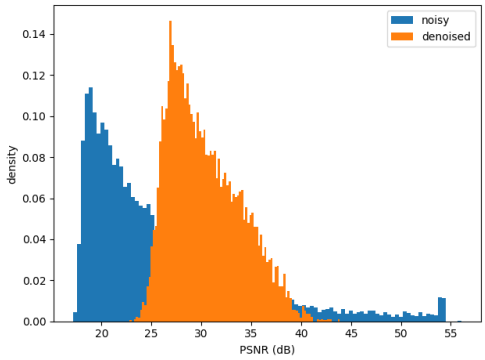
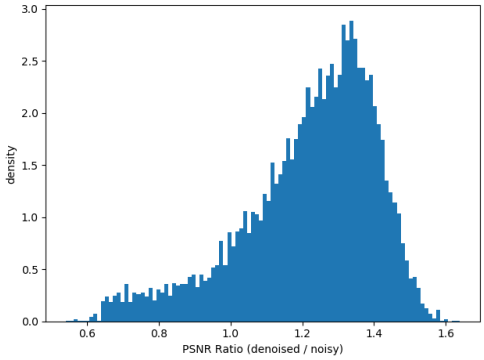


Figure 26: Random impulse denoising results with standard deviation of $\sigma = 50$ and a corruption probability of $p = 0.5$.

Indeed the PSNR histograms confirm this as the ratio histogram shows that the PSNR of some images actually decreases after denoising most likely indicating that when the noise level is low enough, the model may have trouble differentiating noise from detail.



(a) Histogram of PSNR values across test images.



(b) Histogram of PSNR ratio between denoised and noisy images.

4 Adversarial Noise

This portion of the final project was completed as a joint effort to explore how an adversarial attack could affect the learning-based denoising network produced by Dylan in the previous section. Denoising has been proposed by numerous papers over the past couple of years as a mitigation to adversarial examples. However, if the denoiser used is a learning based method, it may lead to a false sense of security. This portion is split into two sections, the first to define the adversarial attack used and the second to explore the results of applying the attack on networks trained on various types of noise.

4.1 Adversarial Attack

Adversarial examples are studied extensively within the classification domain as previously discussed. However, constructing adversarial examples for tasks such as image segmentation, object tracking, or object detection seem to be less studied and more difficult, while adversarial examples for learning based denoising methods seem to be missing from literature. However, we aimed to adopt an algorithm for generating adversarial examples for semantic segmentation to the denoising domain. Xie et al. [9] proposed a novel algorithm, Dense Adversary Generation (DAG), which generates adversarial examples to misclassify regions within the image, which results in errors in object detection as well. Perturbations generated by DAG were also shown to transfer to other networks, which can be seen in Figure 28.

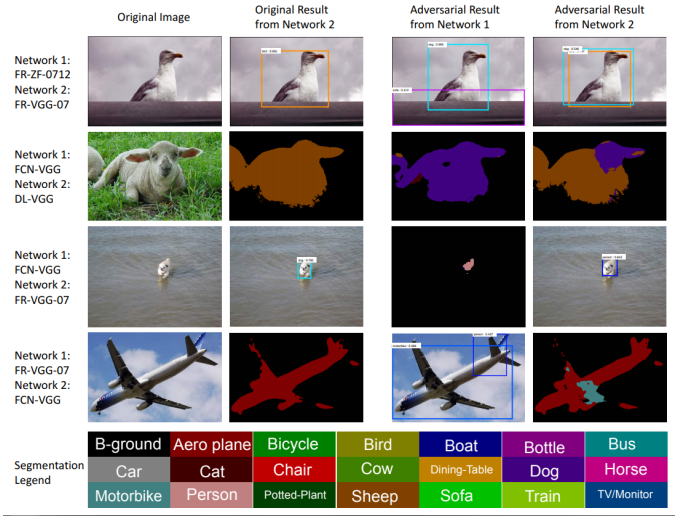


Figure 28: DAG Acting on Segmentation and Object Detection Networks

The DAG algorithm operates by identifying which pixels need to be adjusted by creating a pixel-wise mask, T, which operates on the clean output, L_c of the network and the desired adversarial output L_a. The adversarial output is

constructed by the adversary, which could switch a region or portion of a regions label. Then at each iteration, T, is defined as the mask where the clean label does not match the adversarial label. This mask is applied to the gradients to only perturb the pixels which are still being classified correctly but are desired to be misclassified. The loss function used is:

$$L(x, L_a, L_c, T) = \sum_{i=0}^{n} T \cdot f(x) \cdot (L_a - L_c) \tag{12}$$

where f is the network function and n is the batch size. The gradient, referred to as r_m in the paper, is computed with respect to the input image, x, by pixel. The authors note that for numerical stability r_m is normalized as follows:

$$r'_m = \frac{\gamma}{||r_m||_\infty} r_m \tag{13}$$

where $\gamma = 0.5$ is a fixed hyperparameter. The result of each iteration is an adversarial perturbation, r, which is the cumulative sum of r'_m.

In order to apply this algorithm to the denoising domain, we redefined L_c and L_a to be the clean image and adversarial image respectively. This would optimize the noise added to the clean image to produce the adversarial image. For every image, a noisy image is produced under the constraints of the noise type used to train the network. For example, when using the Gaussian trained network a noisy image is produced with Gaussian noise. For all our experiments, the noisy image was used as the adversarial target. We were hoping this would produce adversarial noise that would resemble the noise distribution that was being used. Furthermore, we adjusted the hyperparameter, $\gamma = 0.1$, since the images we were using were small and the affect of each step was too significant.

4.2 Experiments

The experiments in this section focus two areas: adversarial noise on individual images and the success of adversarial noise on the test set as a whole. For the first topic, two images were chosen from the CIFAR-10 test set which are shown below in Figure 29. These images were chosen for their diversity of background, object size and detail.

(a) Ship Image

(b) Truck Image

Figure 29: Original CIFAR-10 Test Set Examples

Adversarial examples were generated for each of these images with respect to seven denoising network. The seven denoising networks used were:

- Gaussian - $\mu = 0$, $\sigma = 50$
- Gaussian - $\mu = 0$, $\sigma = 50$
- Poisson - $\mu = 100$, $d = 75$
- Poisson - $\mu = 200$, $d = 75$
- Poisson - $\mu = 500$, $d = 75$
- Impulse - $p = 0.2$
- Impulse - $p = 0.5$

These networks are directly from Dylan's individual portion of this project. The first image to be examined is the ship image shown in Figure 30. Each row corresponds to a different noise type and corresponding network. The first

and third columns represent the image with noise and adversarial noise applied, respectively. The second and fourth columns represent the image after being passed through the denoising network. While all the original images were able to be denoised fairly well, the adversarial noise was only partially removed in most cases. It is interesting to note that some adversarial noise, such as that produced under the Gaussian models, produce less noticeable noise than the original noise type. Adversarial noise generated under the Poisson and Impulse models was most similar, visually, to the original noise type.

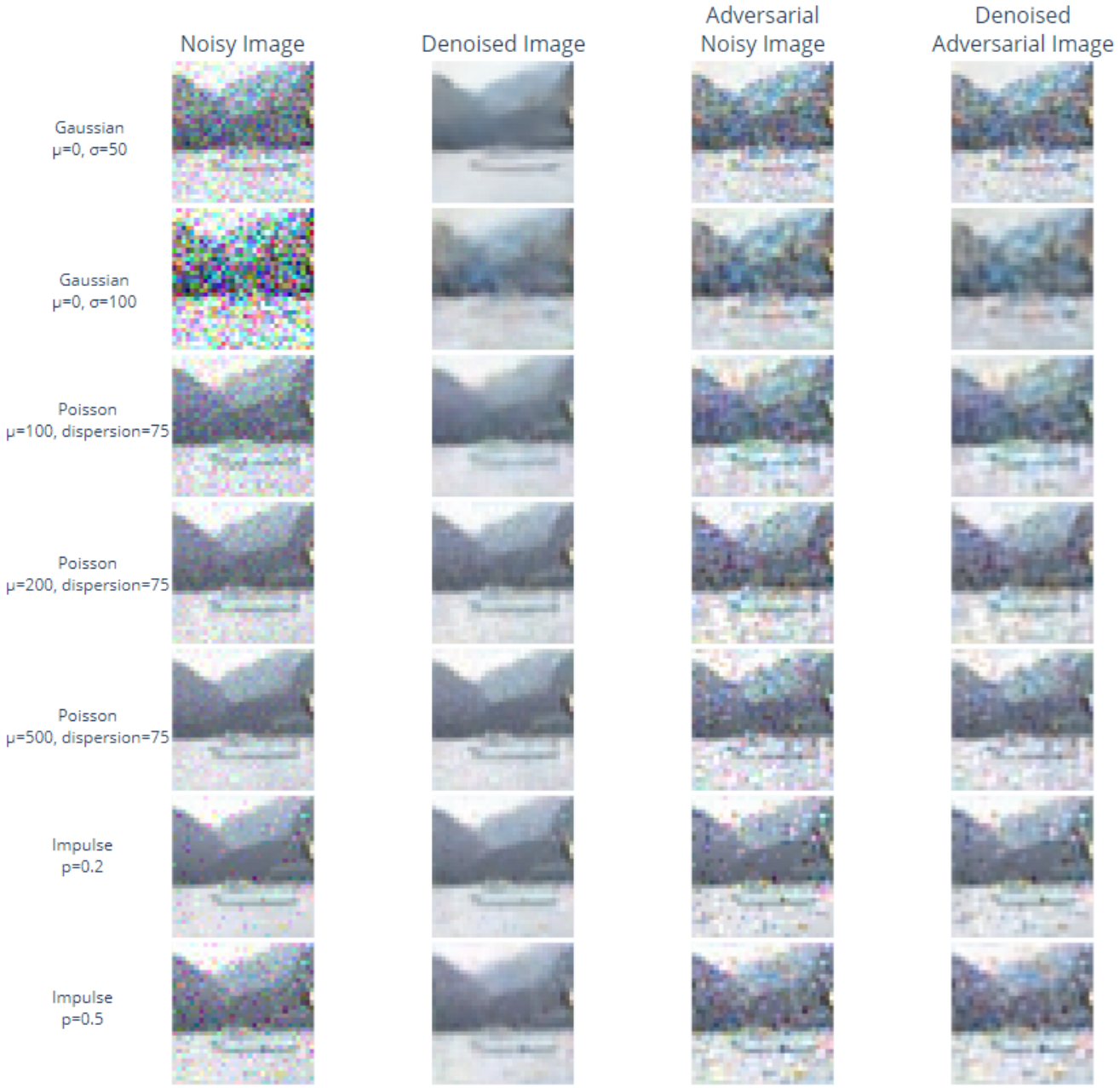


Figure 30: Ship Adversarial Examples Generated Under Various Noise Type

Based on how the denoising models work—by recovering the clean image most consistent with the noisy observation—the adversarial examples should contain noise that mimics the details present in the image. This limits the amount of

clean images consistent with the observed image by adding noise that could be explained by image detail rather than a random process. As we have seen already, sometimes the model confused high frequency image details for noise and smooths them out. Thus if the adversarial noise resembles high frequency image details, then the result of passing the adversarial noisy images into the model should be nearly the same, just with some minor smoothing. This is confirmed by the examples above as the adversarial examples remain largely unchanged after passing them into the model.

To further examine the adversarial noise, we plotted the histogram of the original noise type and the resultant adversarial noise. We were surprised to find that the adversarial noise distributions were similar to that of the original noise type in most cases.

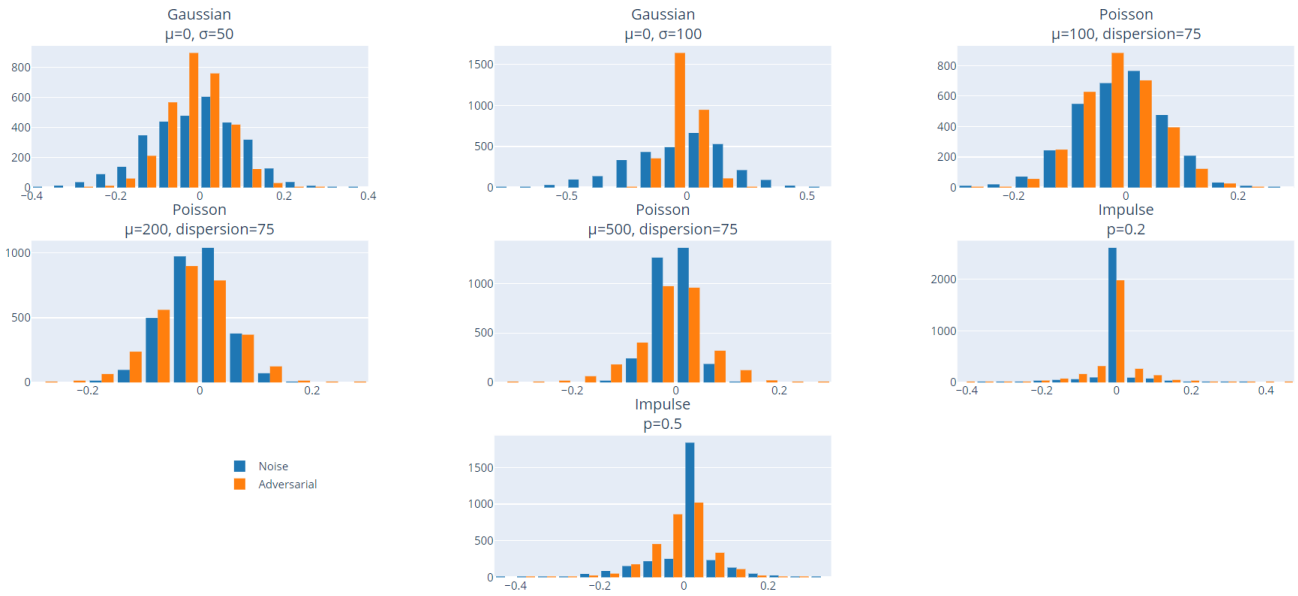


Figure 31: Noise and Adversarial Noise Distributions for Ship Example

These figures were generated for the truck, Figures 32 and 33. The adversarial examples reveal one of the main shortcomings of the models that we built: they have no contextual understanding of the images they are denoising. The adversarial noise appears to be independent of the image content and instead just resembles random high frequency details. It is possible that these models could be more robust if we were able to introduce some prior knowledge about the input images. Since we are using an image dataset with labels for each image, it is possible that these models could be further improved by conditioning the denoiser based on these class labels. For example, if the model knew that the image it was denoising contained a cat or a dog, it might be less likely to smooth out the detailed texture of the fur. In the figure below, if the model knew that the image contained a truck, it might be more likely to smooth out the flat texture of the back of the truck. Unfortunately these improvements would introduce a whole layer of complexity into the training process as we would have to simultaneously train the model to perform classification as well as denoising.

Figure 32: Truck Adversarial Examples Generated Under Various Noise Type

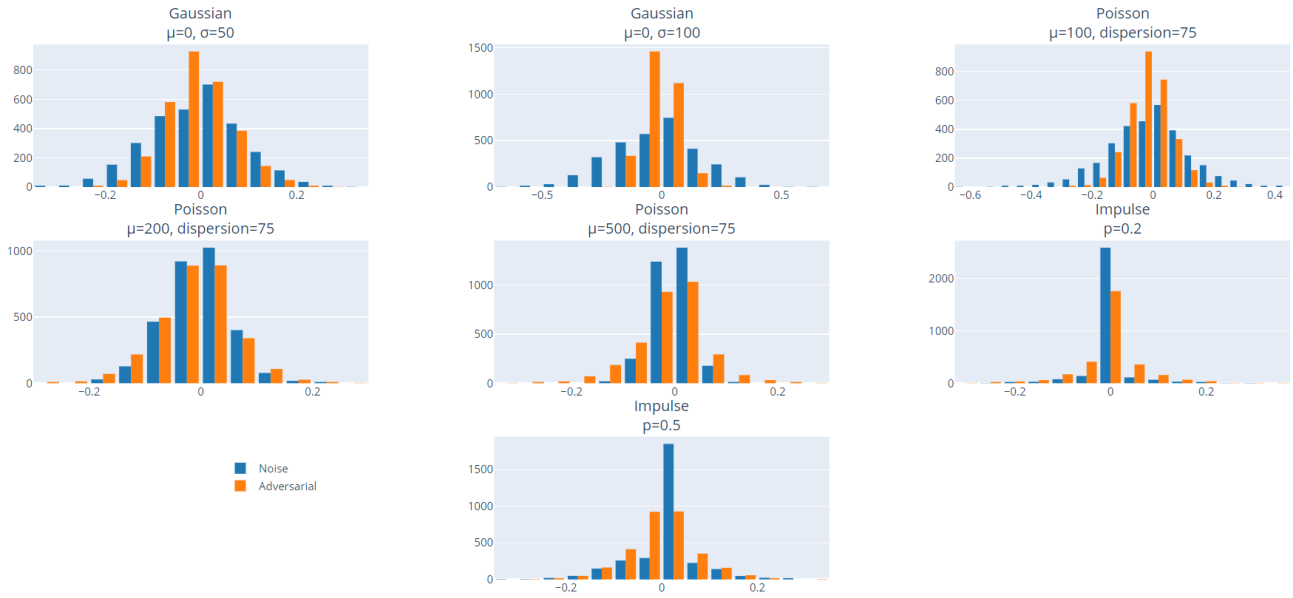


Figure 33: Noise and Adversarial Noise Distributions for Ship Example

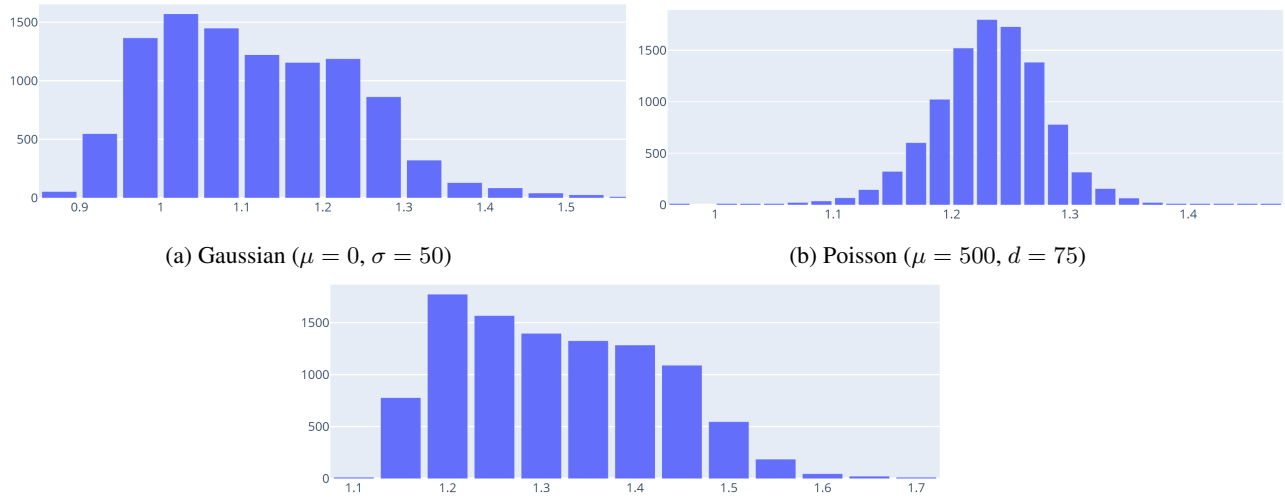
The second topic, the general performance of the adversarial generation method was to be evaluated. In order to evaluate its performance the PSNR was calculated for the each of the denoised images, non-adversarial and adversarial, with respect to the ground truth image. Then the ratio of the image was stored for each example in the test set. We defined the ratios as follows:

- PSNR Ratio > 1: the error of the denoised adversarial example was higher than the non-adversarial example.
- PSNR Ratio < 1: the error of the denoised adversarial example was lower than the non-adversarial example.

One parameterization of each noise type was considered for this section and the resultant histograms can be found in Figure 34. For adversarial examples generated under the Gaussian model, 29.33% were roughly equivalent, $[0.95, 1.05]$, to the non-adversarial examples while the average PSNR ratio was in the range $[1.1, 1.15]$. It is interesting to note that 1.5% of the samples had PSNR ratios of greater than 1.4, which is show significant difference. For adversarial examples generated under the Poisson model, only 0.09% of were roughly equivalent, $[0.96, 1.06]$ and the average PSNR ratio was in the range $[1.22, 1.24]$. While only 0.17% has PSNR ratios of greater than 1.4. Finally, under the impulse model, only 0.09% of examples has a PSNR ratio of less than 1.25. The average PSNR ratio was in the range $[1.275, 1.325]$ while 31.58% of the examples had a PSNR ratio of higher than 1.375.

These results show, on average, the impulse noise model was most susceptible to adversarial perturbation while Gaussian was the least susceptible. We believe these experiments show that learning based denoisers are prone to adversarial examples like image classification networks.

However, one of the goals of these experiments was to provide some insight into if the adversarial examples were exploiting the linearity of the neural network or the useful, non-robust features. While we modified a adversarial generation methodology, we were unable to make conclusions on the reason for success of this methodology. These experiments did not provide enough evidence to make a conclusion either way. If more time was available, completing the non-robust feature identification procedure on the denoising network and comparing them to the adversarial noise would have been a more convincing argument to determine which adversarial example literature explanation would be more likely.



(a) Gaussian ($\mu = 0$, $\sigma = 50$)

(b) Poisson ($\mu = 500$, $d = 75$)

(c) Impulse ($p = 0.2$)

Figure 34: PSNR Ratio Histograms

5 Conclusion

Overall we have demonstrated that it is possible to build learning-based denoising models using only corrupted images and that these models are robust to different noise distributions and intensities. However, we also have shown that these denoisers are susceptible to adversarial examples just like many similar models for different tasks. We were able to modify an adversarial generation method for image segmentation to be utilized on our image denoising networks. Using the adversarial examples generated by this novel method, we were able to further understand these models and where they fall short offers insights into how they can be improved in the future.

Through our experiments, we found that the Noise2Noise training strategy gives us the best performance on Poisson shot-noise corruptions which is promising because this noise distribution is the closest to the noise generally present in natural images. The models seemed to struggle the most when removing random impulse noise which is reasonable because it was the most complex noise process we tested.

Based on the adversarial examples in the previous sections, we believe that the denoisers do not remove noise based on contextual scene information, but rather based on a measure like expectation of the pixel distribution. This is one potential reason that these models are open for adversarial attacks. However, with the current experiment set we were not able to explain the adversarial examples causes in terms of the state of the art literature explanations. In future work it would be interesting to try introducing some prior knowledge about the scene into the training process to see if we can teach the model to look at higher-level structure rather than individual pixel distributions. Additionally, comparing useful, non-robust features to the adversarial noise may provide further insight into their fundamental cause.

6 Appendix A

The following are the feature maps for the first layer for the Gaussian and Salt-And-Pepper noise examples.

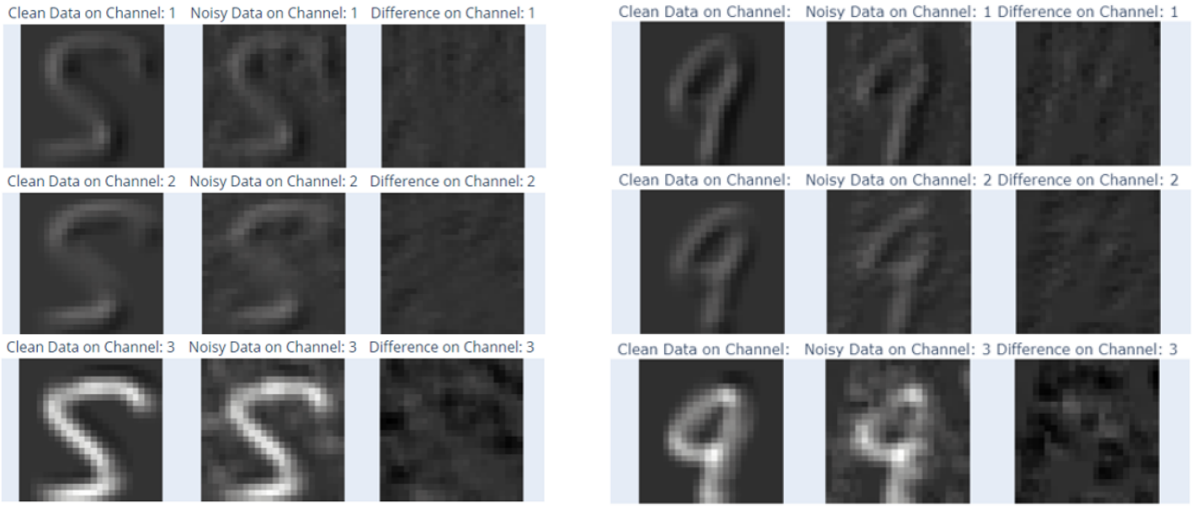


Figure 35: Feature Maps After First Layer For Noise Examples

The following is the feature map for the first layer for the FGSM example.

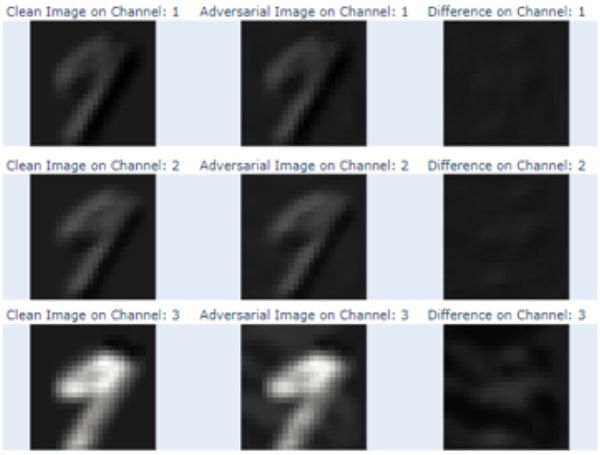
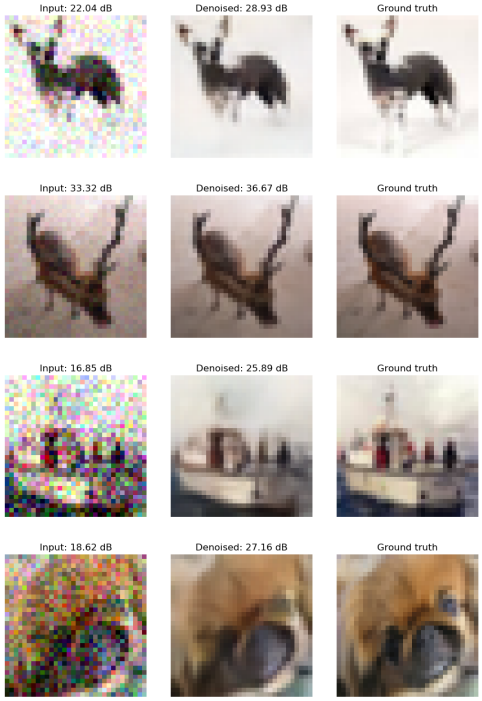


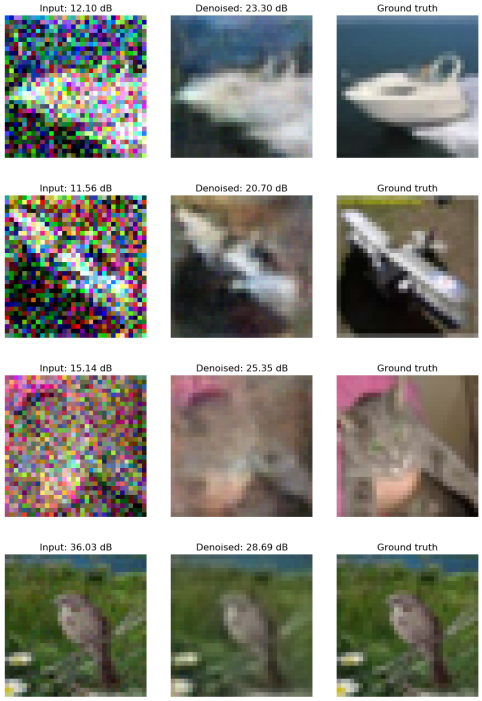
Figure 36: Feature Maps After First Layer For FGSM Example

7 Appendix B

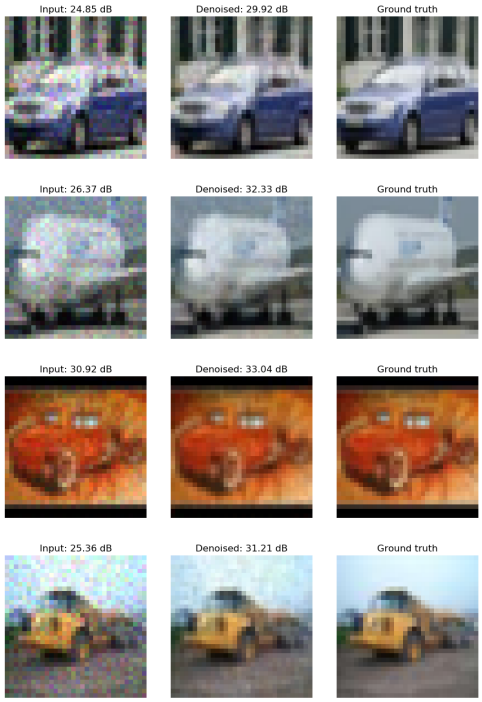
The following are denoised examples from each of the models tested.



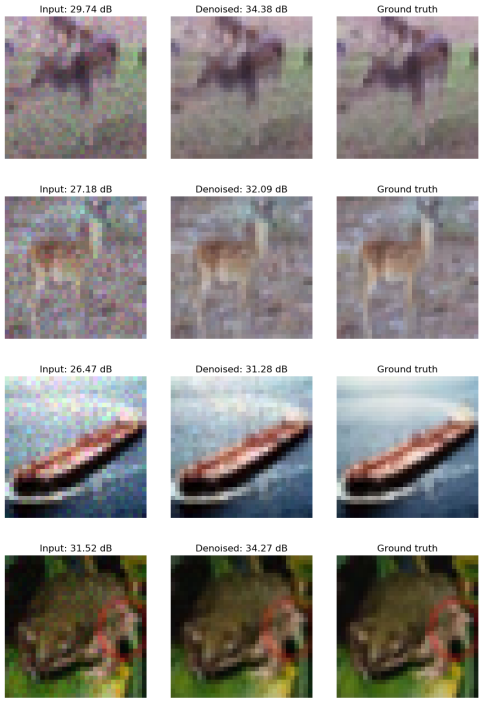
(a) Gaussian denoising with max standard deviation of $\sigma = 50$.



(b) Gaussian denoising with max standard deviation of $\sigma = 100$.



(a) Poisson denoising with mean photons per pixel of 200.



(b) Poisson denoising with mean photons per pixel of 500.

(a) Impulse denoising with corruption probability of $p = 0.2$.

(b) Impulse denoising with corruption probability of $p = 0.5$.

References

[1] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[3] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.

[4] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. Robustness via curvature regularization, and vice versa, 2018.

[5] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features, 2019.

[6] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2Noise: Learning Image Restoration without Clean Data. *35th International Conference on Machine Learning, ICML 2018*, 7:4620–4631, mar 2018.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. Technical report.

[8] Haris Iqbal. HarisIqbal88/PlotNeuralNet v1.0.0, dec 2018.

[9] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection, 2017.