520.638 Deep Learning
Homework 4 - Autoencoders for Reconstruction

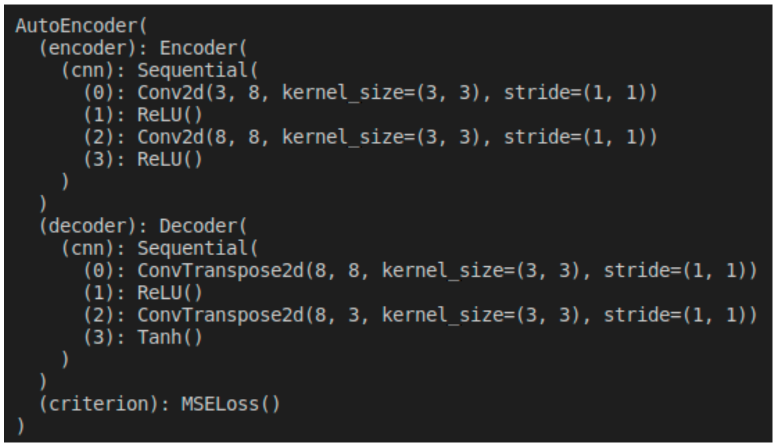
Nicholas Hinke

April 29, 2022

1 Part 1: Autoencoder for Image Reconstruction

Within this section, a convolutional autoencoder was designed and implemented using PyTorch in order to reconstruct images from the CIFAR-10 dataset. Note that the CIFAR-10 dataset is comprised of 60,000 32×32 color (3-channel) images from ten classes split between two sub-datasets (training and test), and is publicly available online at `https://www.cs.toronto.edu/~kriz/cifar.html`. For this particular task, the input to the autoencoder was a normalized version of the image (on the range [-1.0,1.0]) that was to be reconstructed (*i.e.* no noise or other augmentations were added to the image). As such, the goal of the network was to reproduce the normalized version of that same image, which could subsequently be *un*-normalized to the more typical [0,255] pixel range and visualized.

The design of the autoencoder network itself consisted of an encoder and decoder, each containing two convolutional layers. Between each convolutional layer, an appropriate activation function was used. Consequently, the combined network as a whole can be described simply as four convolutional layers with an activation function after each layer. For this implementation, ReLU was used for the first three activation functions, but hyperbolic tangent was used for the last activation function in order to bound the outputs to the same range as the inputs (that range being [-1.0,1.0], which makes sense since the network is reproducing the normalized version of the original image). A summary of the network architecture is provided below.



```
AutoEncoder(
  (encoder): Encoder(
    (cnn): Sequential(
      (0): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1))
      (1): ReLU()
      (2): Conv2d(8, 8, kernel_size=(3, 3), stride=(1, 1))
      (3): ReLU()
    )
  )
  (decoder): Decoder(
    (cnn): Sequential(
      (0): ConvTranspose2d(8, 8, kernel_size=(3, 3), stride=(1, 1))
      (1): ReLU()
      (2): ConvTranspose2d(8, 3, kernel_size=(3, 3), stride=(1, 1))
      (3): Tanh()
    )
  )
  (criterion): MSELoss()
)
```

Figure 1: Summary of autoencoder model architecture

Upon attaining the outputs from the network, the mean-squared error (MSE) reconstruction loss (otherwise known as the “L2 loss”) was used to train the network. When training the network, both stochastic gradient descent and Adam were tested as optimizers, and it was found that Adam produced notably faster convergence. It should also be noted that 10% of the original training set (or 5,000 of the original 50,000 images) was reserved for use as a validation set; this was done in order to monitor if the model starting. A plot of the convergence of the MSE loss function (with a maximum loss of 0.232 at the start and a minimum loss of 0.012 at the end) using this training strategy is provided below.

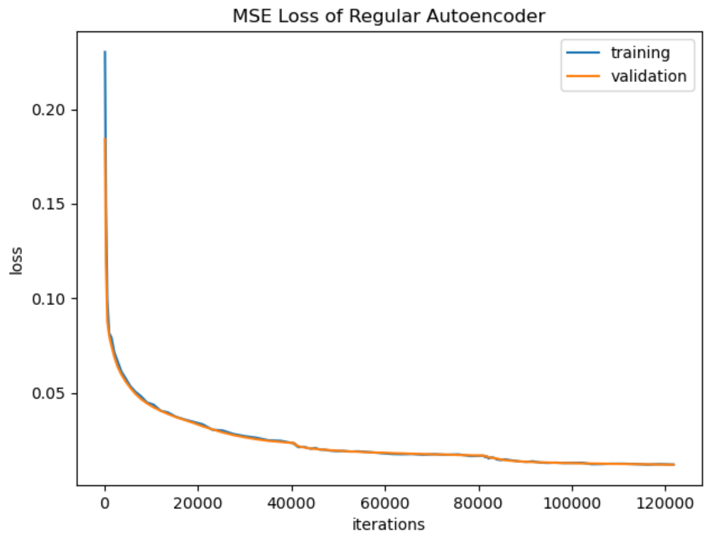


Figure 2: MSE loss of autoencoder during training

After training the model, its performance was evaluated by attempting to reconstruct the 10,000 images in the test dataset. When performing this evaluation, not only could the MSE loss between the network outputs and the original inputs be computed once again, but the model performance could also be represented visually! This was done simply by *un*-normalizing both the original inputs and the model outputs, and then plotting them side-by-side. The average MSE loss was computed over the entire test dataset (and was found to be 0.012), and the first ten images within the test dataset were visualized in the aforementioned manner. Those visualizations are provided on the following five pages.

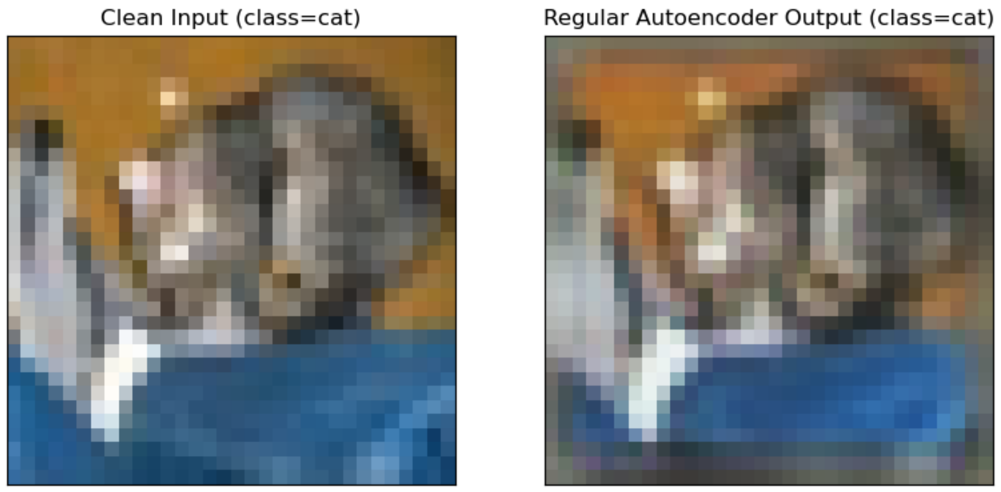


Figure 3: Autoencoder reconstruction of image 1 from test dataset

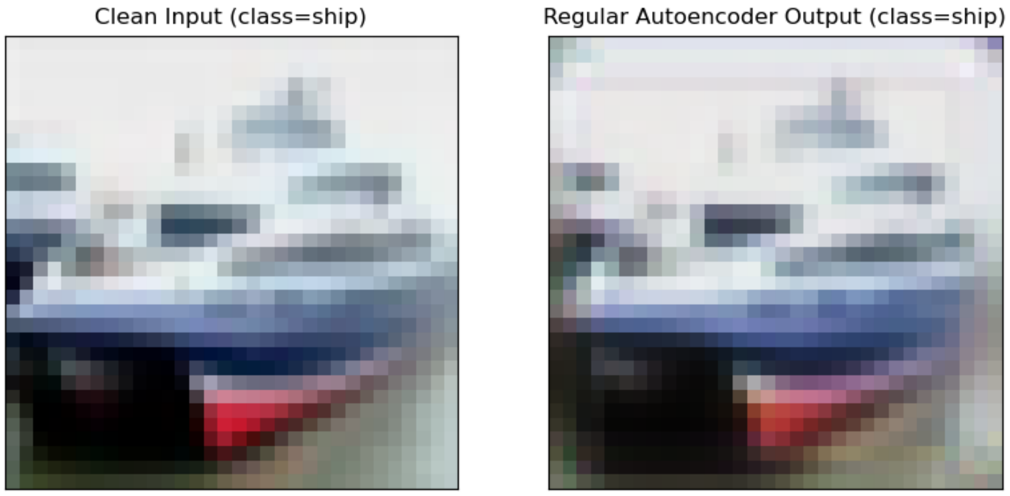


Figure 4: Autoencoder reconstruction of image 2 from test dataset

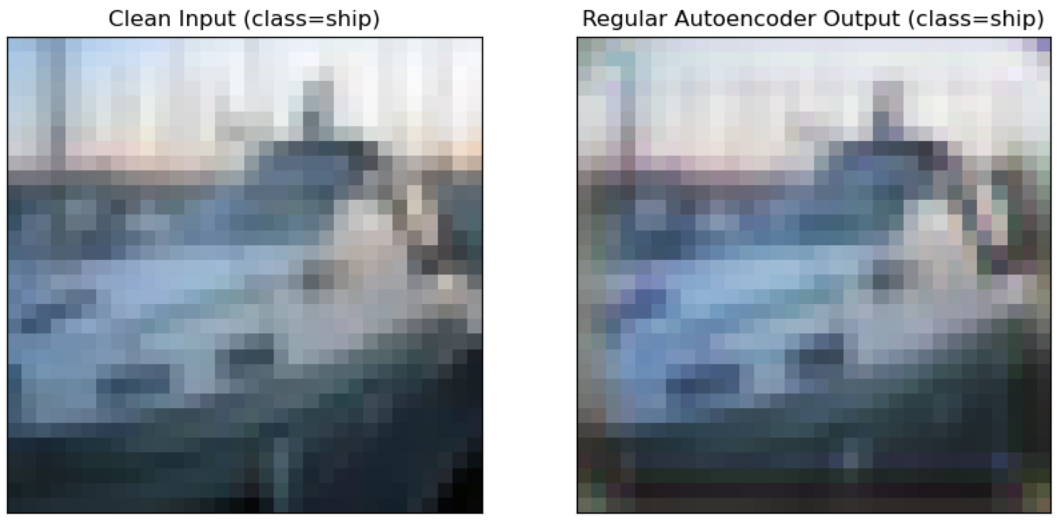


Figure 5: Autoencoder reconstruction of image 3 from test dataset

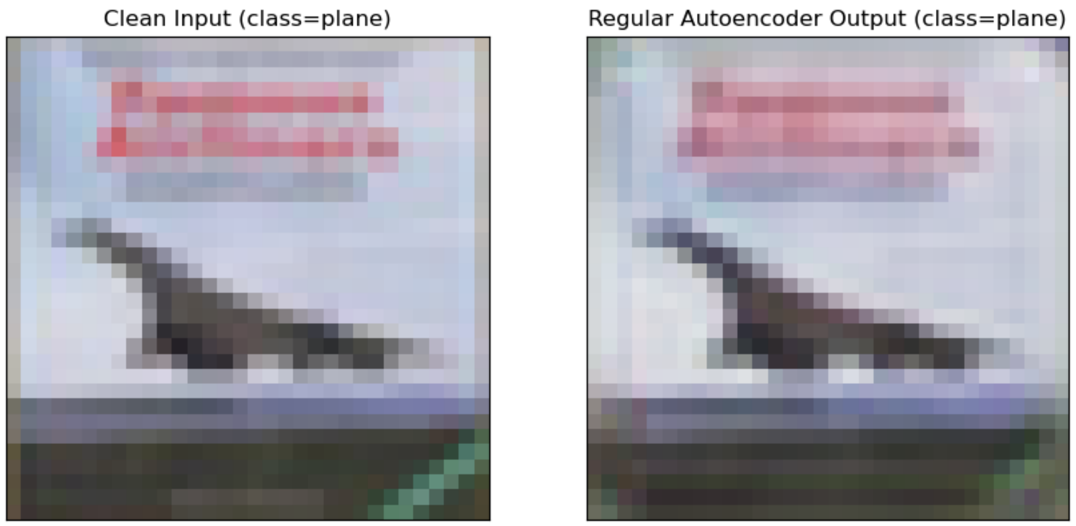

Clean Input (class=plane)
Regular Autoencoder Output (class=plane)

Figure 6: Autoencoder reconstruction of image 4 from test dataset

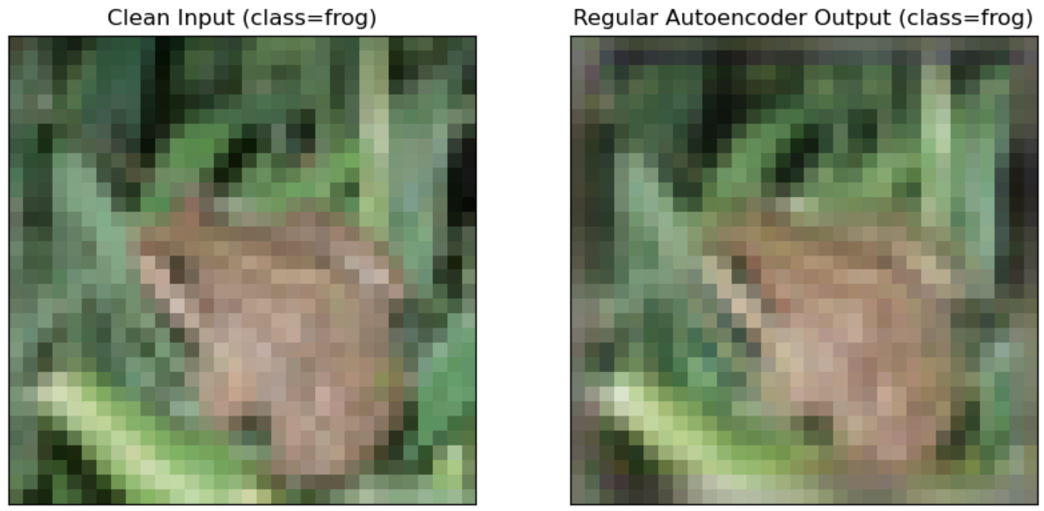


Figure 7: Autoencoder reconstruction of image 5 from test dataset

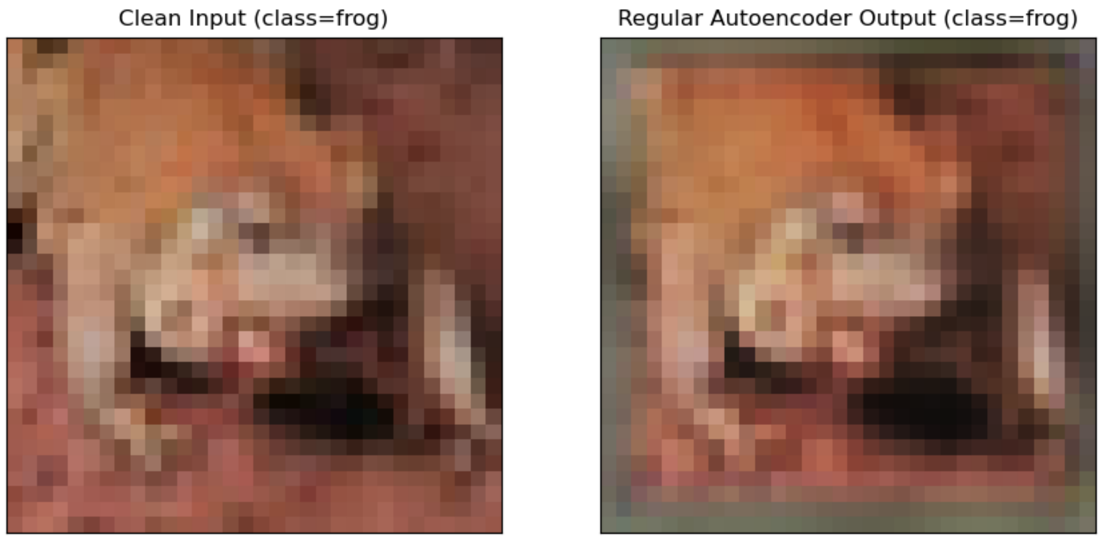


Figure 8: Autoencoder reconstruction of image 6 from test dataset

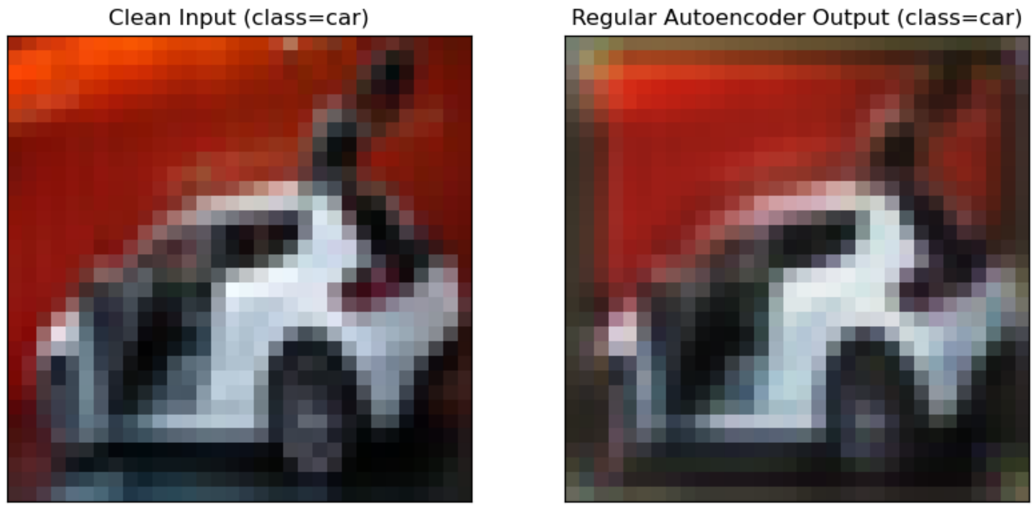


Figure 9: Autoencoder reconstruction of image 7 from test dataset

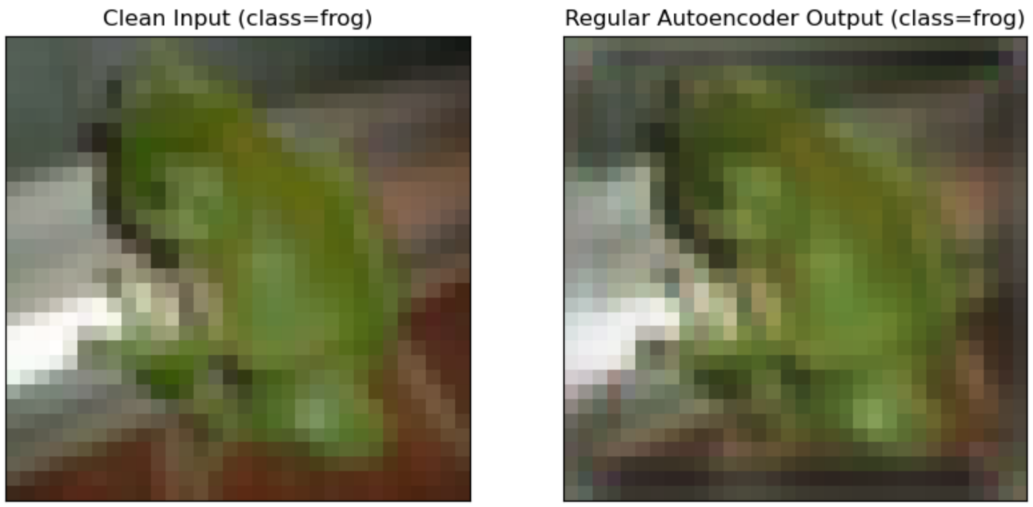


Figure 10: Autoencoder reconstruction of image 8 from test dataset

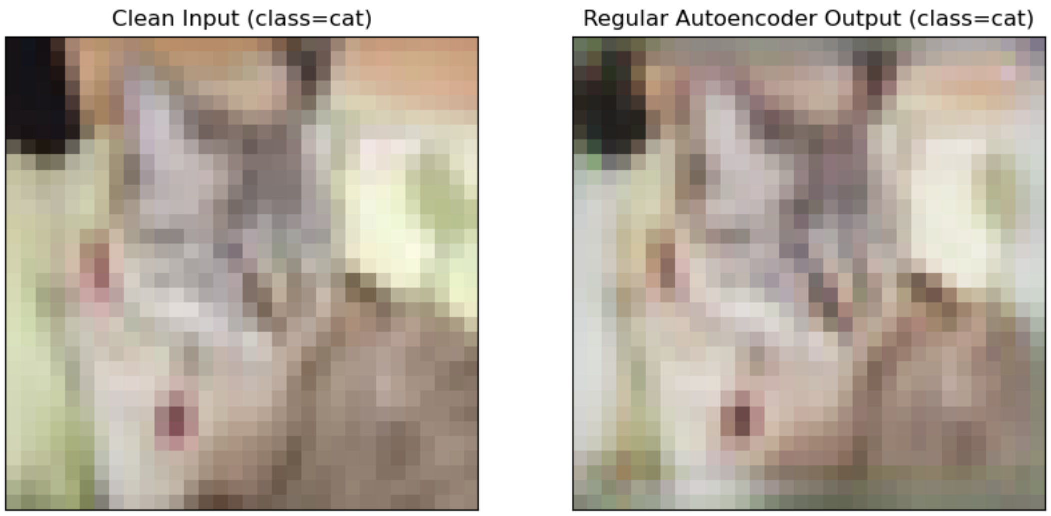


Figure 11: Autoencoder reconstruction of image 9 from test dataset

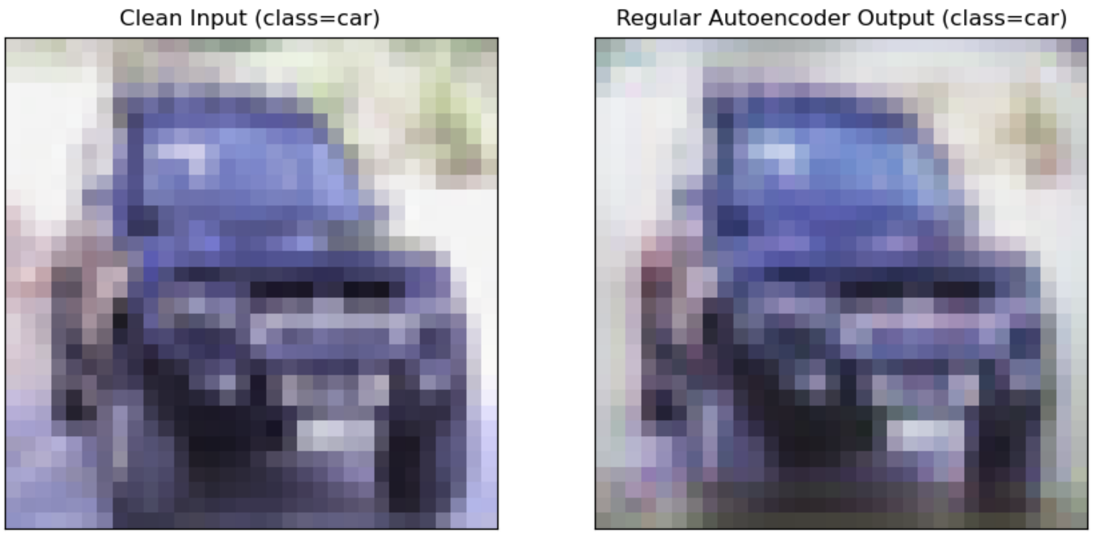


Figure 12: Autoencoder reconstruction of image 10 from test dataset

As can be seen from the visualizations on the previous pages, the model appears to be performing quite well. Additionally, as the average MSE loss on the test set was roughly the same as that of the training set (at the end of the training), it seems as though the model did **not** overfit to the training set. Despite the apparent success of the autoencoder, however, the challenge it faced was frankly not that challenging. Indeed, all the network had to do was reproduce the input it was given. In order to make things more interesting, we will add Gaussian noise to the input images in the next section, and task the network with reproducing the original "clean" images. In order to do this, we will be constructing what is known as a *denoising* autoencoder. It should also be noted that throughout this section, all code used to complete these experiments was written in Python utilizing the PyTorch framework (and is publicly available at `https://github.com/nhinke/deeplearning-repo/tree/master/Homework/hw4`) with the use of the following libraries:

- **time**
- **torch**
- **numpy**
- **torchvision**
- **matplotlib.pyplot**

2 Part 2: Denoising Autoencoder

As briefly mentioned at the end of last section, the goal of this section was to design and implement a *denoising* autoencoder. The task of this network was nearly identical to that of the previous section–the task being to reproduce the original images–with the added difficulty of receiving noisy input images, rather than simply the original images themselves like in the last section. It should be noted that the noise was sampled from a zero-mean Gaussian distribution with a variance of 0.1, and was added to the normalized version of the images on the range [-1.0,1.0]. As such, additional care was taken to clamp any resulting values that fell outside of the given range back to -1.0 or 1.0 as appropriate. Then, using the noisy images as inputs to the autoencoder (which was constructed using an identical architecture to that of the previous section), the same MSE reconstruction loss (and Adam optimizer) was used to train the network to reproduce the original clean images. A plot of the convergence of the MSE loss function using this training strategy is provided below, as well as screenshots of terminal outputs that were printed at the beginning and end of training for each autoencoder (‘AE’ refers to the autoencoder from last section, whereas ‘DAE’ refers to the denoising autoencoder from this section’).

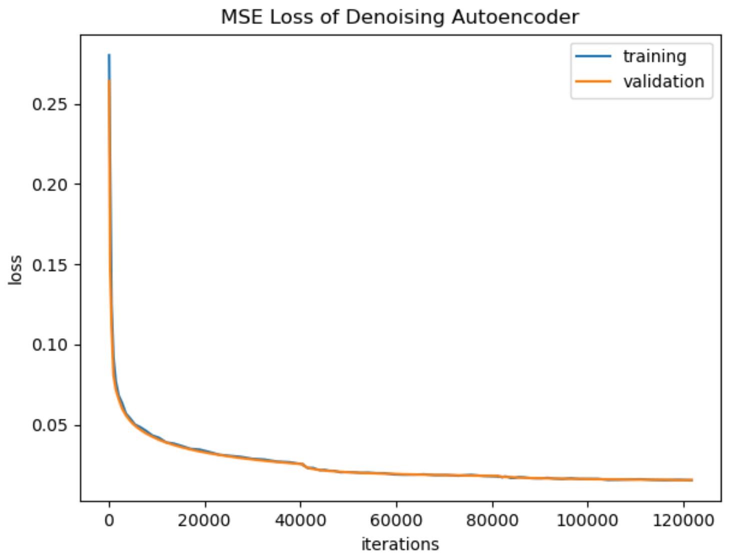
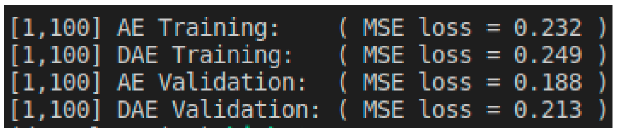
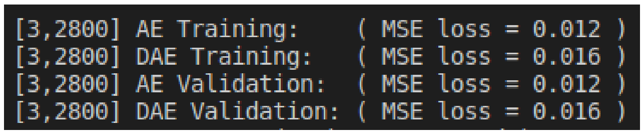


Figure 13: MSE loss of denoising autoencoder during training



```
[1,100] AE Training:    ( MSE loss = 0.232 )
[1,100] DAE Training:   ( MSE loss = 0.249 )
[1,100] AE Validation:  ( MSE loss = 0.188 )
[1,100] DAE Validation: ( MSE loss = 0.213 )
```

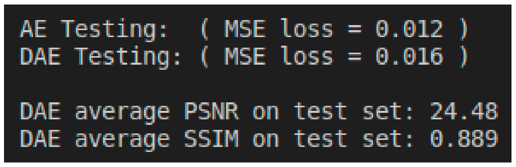
Figure 14: MSE loss of both autoencoders at start of training



```
[3,2800] AE Training:    ( MSE loss = 0.012 )
[3,2800] DAE Training:   ( MSE loss = 0.016 )
[3,2800] AE Validation:  ( MSE loss = 0.012 )
[3,2800] DAE Validation: ( MSE loss = 0.016 )
```

Figure 15: MSE loss of both autoencoders at end of training

After training the model, its performance was evaluated by attempting to reconstruct the 10,000 images in the test dataset from noisy versions of them. When performing this evaluation, visualizations were once again created in the same manner as last section (this time including both the original image *and* the noisy input, as well as the network output). Using the same approach as last section, the average MSE loss was computed over the entire test dataset (and was found to be 0.016), and the first ten images within the test dataset were visualized in the aforementioned manner. In addition, the peak signal to noise ratio (PSNR) and structural similarity index measure (SSIM) were computed for each image as further metrics to determine how well the denoising autoencoder could reproduce the original clean images. A screenshot of terminal outputs containing the average MSE loss of both autoencoders on the test dataset as well as the average PSNR and SSIM of the denoising autoencoder on the test dataset is provided below. Additionally, the aforementioned visualizations of the denoising autoencoder performance are provided on the following five pages.



```
AE Testing:  ( MSE loss = 0.012 )
DAE Testing: ( MSE loss = 0.016 )

DAE average PSNR on test set: 24.48
DAE average SSIM on test set: 0.889
```

Figure 16: Error metrics for both autoencoders on test dataset

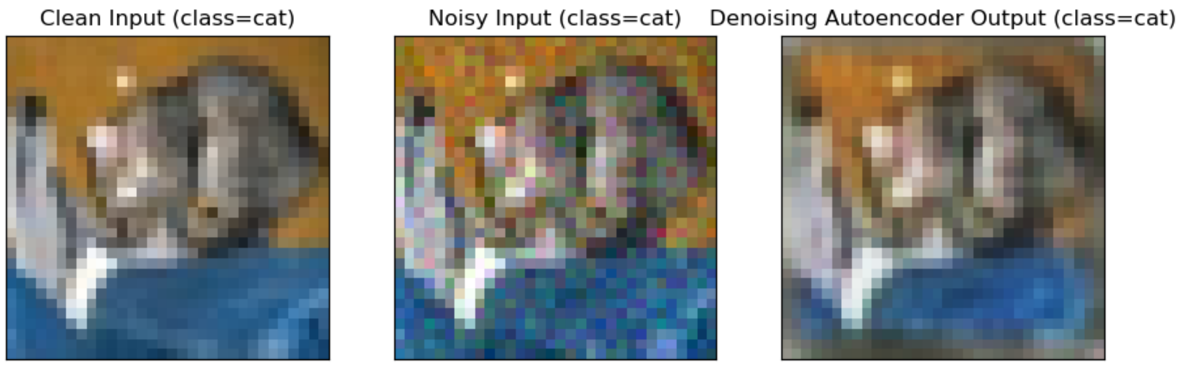


Figure 17: Denoising autoencoder reconstruction of image 1 from test set

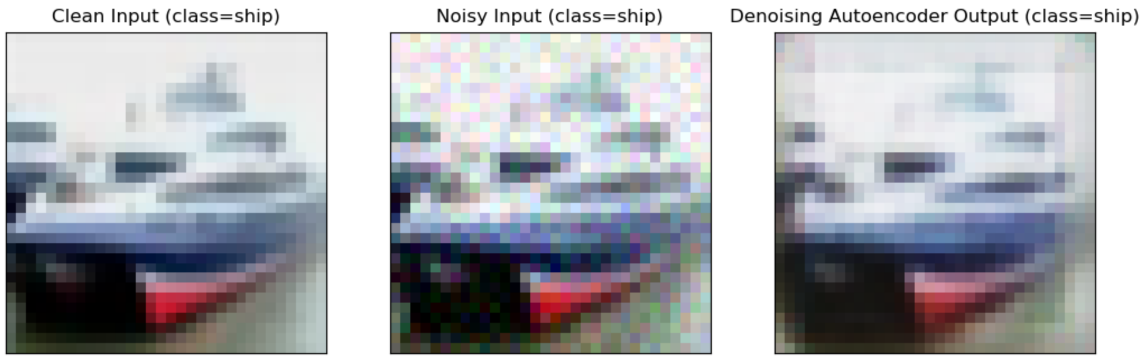


Figure 18: Denoising autoencoder reconstruction of image 2 from test set

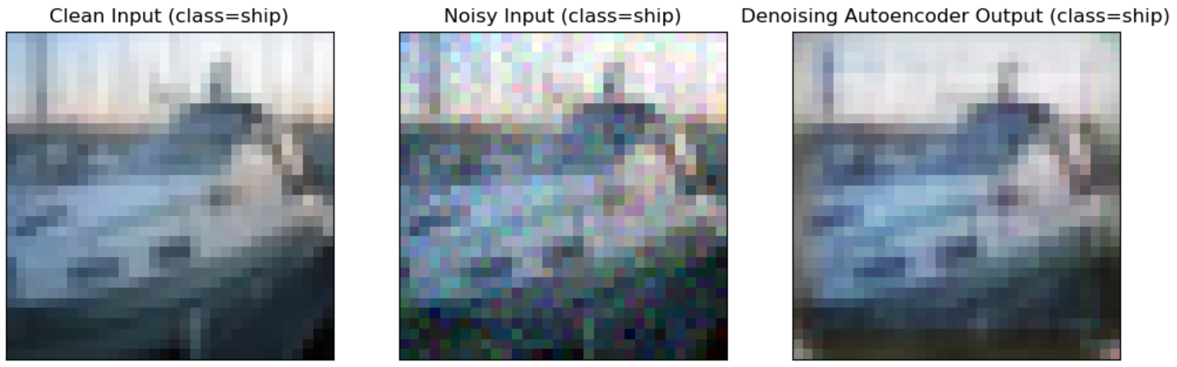


Figure 19: Denoising autoencoder reconstruction of image 3 from test set

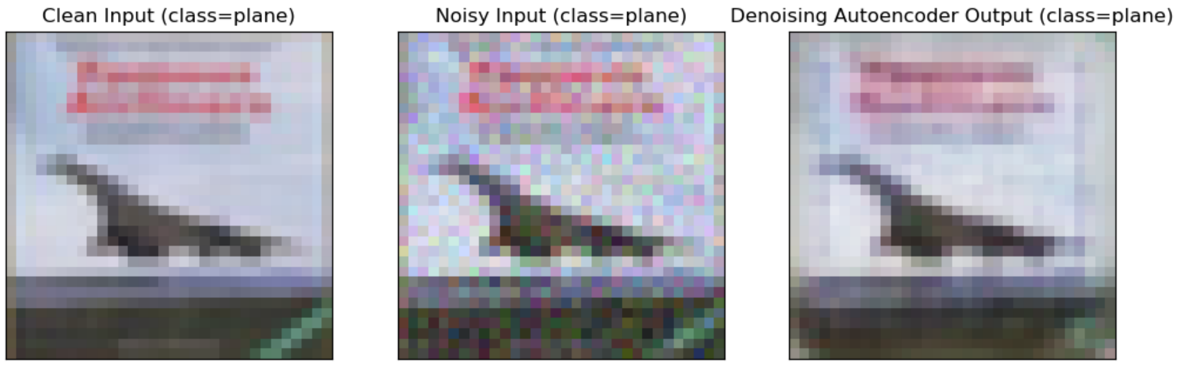

Clean Input (class=plane)
Noisy Input (class=plane)
Denoising Autoencoder Output (class=plane)

Figure 20: Denoising autoencoder reconstruction of image 4 from test set

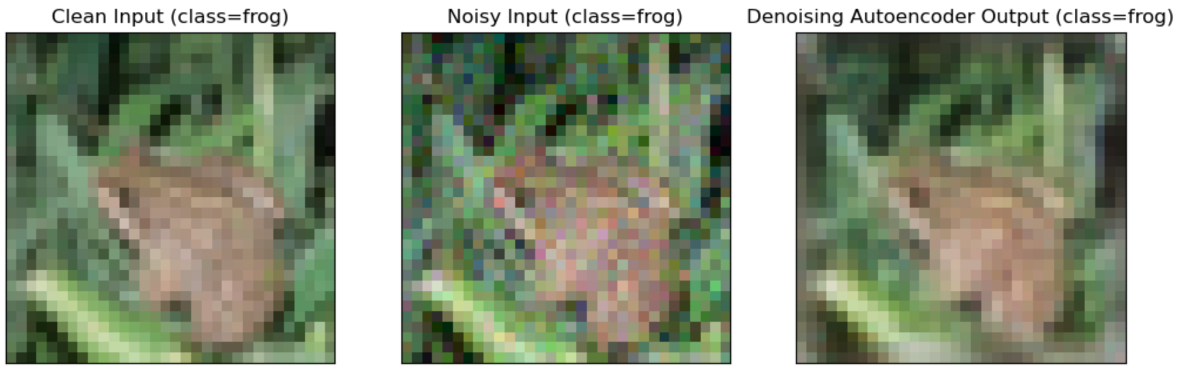


Figure 21: Denoising autoencoder reconstruction of image 5 from test set

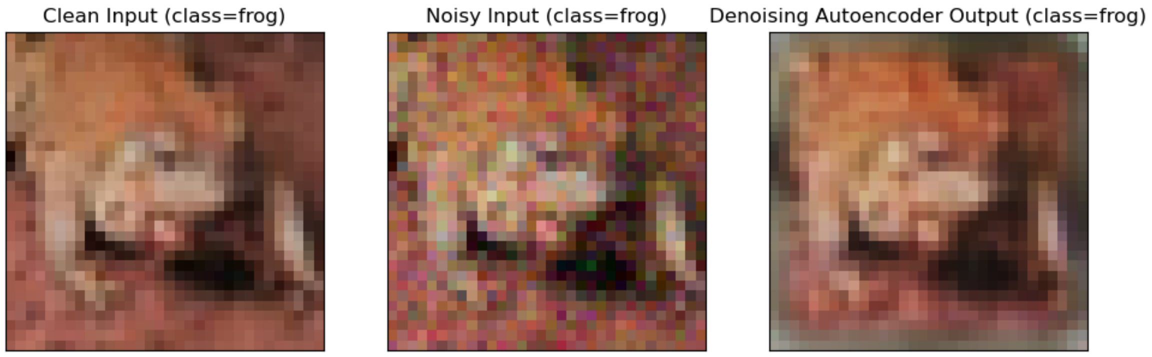


Figure 22: Denoising autoencoder reconstruction of image 6 from test set

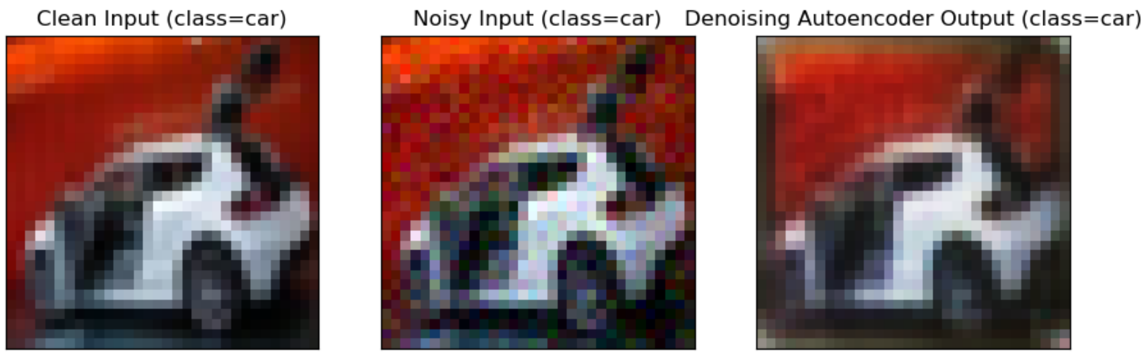


Figure 23: Denoising autoencoder reconstruction of image 7 from test set

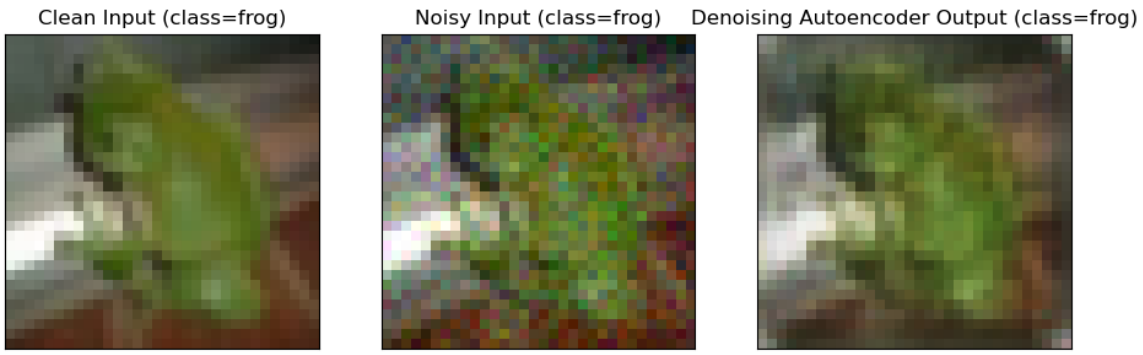


Figure 24: Denoising autoencoder reconstruction of image 8 from test set

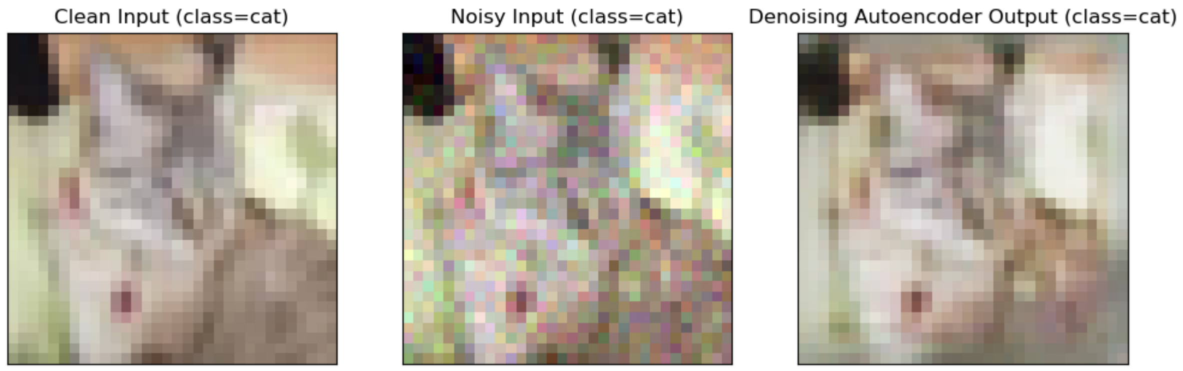


Figure 25: Denoising autoencoder reconstruction of image 9 from test set

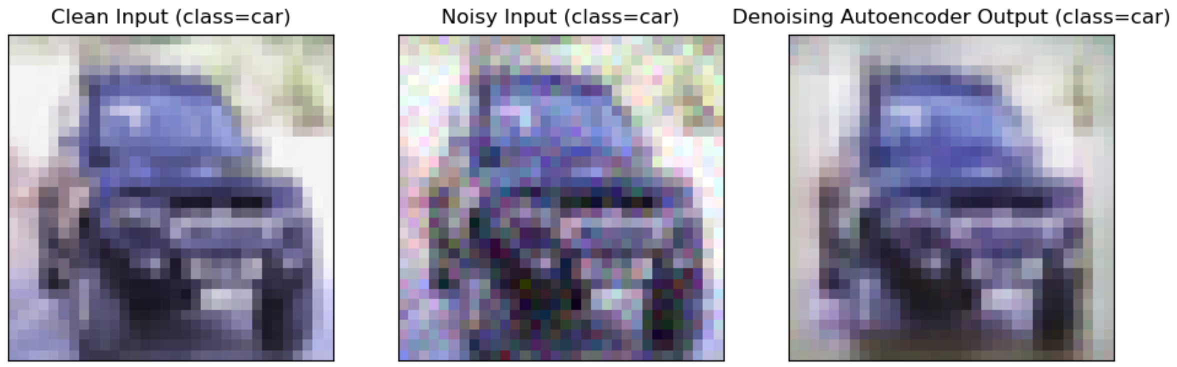


Figure 26: Denoising autoencoder reconstruction of image 10 from test set

As can be seen from the visualizations on the previous pages, the model appears to be performing quite well from a simple visual inspection. Additionally, as the average MSE loss on the test set was roughly the same as that of the training set (at the end of the training), it seems as though the model did **not** overfit to the training set. Despite these apparent successes, however, the other utilized metrics of PSNR and SSIM point to only adequate performance, and suggest that there is further room for improvement. Several strategies could be implemented in order to continue improving the resulting reconstructions, such as: further hyperparameter tuning via cross-validation, increasing the model complexity by adding more convolutional layers (but being careful to avoid overfitting using techniques such as L1 regularization or batch-normalization), training for more epochs on the training dataset, using data augmentation techniques (for instance, creating several noisy inputs all for the same original image) to increase the size of the training dataset, *etc.*. That being said, due to the visual similarity of the network outputs on such a low-resolution dataset (recall each image is only 32×32!), we will declare this denoising autoencoder a success. It should also be noted that throughout this section, all code used to complete these experiments was written in Python utilizing the PyTorch framework (and is publicly available at `https://github.com/nhinke/deeplearning-repo/tree/master/Homework/hw4`) with the use of the following libraries:

- **time**
- **torch**
- **numpy**
- **torchvision**
- **skimage.metrics**
- **matplotlib.pyplot**