CS545 Final Project

Dave Howell, Reid Wahl, and Amila Ferron

October 3, 2022

Roles

Dave Howell: adaptation of original CNN project, implementation of distortion filters

Reid Wahl: convolutional auto-encoder implementation, Python/Google Colab consultation

Amila Ferron: CNN example code interpretation, CNN design and experimentation

Project Design

For our final project, we set out to restore the quality of images that had been distorted, and to compare the results produced by a Convolutional Neural Network versus the results of an Auto-Encoder.

We used four distortion methods: JPEG compression and decompression at 30% quality, Gaussian blur with a 15-pixel radius, a uniform noise generator at 25% intensity, and randomly-located horizontal and vertical lines rendered atop the image. We started with images from the Blur dataset, which provided 350 undistorted source images. Our code ingested those clean source images and divided each into 12 smaller tile images, yielding a total of 4,200 images, sized 512×512 pixels, to divide between training and validation sets. We then scaled these image tiles down to 64×64 pixels in order to reduce training time.

For each distortion type, we produced a corresponding set of distorted images. We then cleaned the distorted images using various CNN and Auto-Encoder parameters, and com-

pared the cleaned images with the originals using mean squared error to calculate loss. We compared the results across model types to evaluate effectiveness.

We built a "dirtify" tool to tile and scale the source images and to apply the distortion filters. Because we were applying our own distortion filters to create the training set and performing image tiling for data augmentation, we did not rely on any labeled dataset or on source images of any particular dimensions. We used the unfiltered images from the Blur dataset referenced by [RATH], but could have used any image set with at least 64x64-pixel images.

We derived our initial CNN parameters from the GitHub project image-deblurring-using-deep-learning [RATH], whose architecture is in turn based on the project [ALBLUWI] and described in [ALBLUWI *et al* 2018]. That model includes three convolutional layers with channels translating from 3→64, 64→32, and 32→3. We reduced the kernel sizes from 9×9, 1×1, and 5×5 down to 5×5, 1×1, and 3×3, to better suit our smaller image sizes. In addition, the padding was originally set to 2 for every layer, so we calculated the ideal padding sizes for each kernel size and set it accordingly per layer.

For the convolutional auto-encoder, we started with [RATH 2020], a tutorial about auto-encoders based on [MASCI 2011]. Our auto-encoder class used [RATH 2020] as a guide, with the forward operation consisting of a sequence of "encode" steps followed by a symmetric sequence of "decode" steps. ReLU was run on the result of each step and fed into the subsequent step. (Leaky ReLU did not consistently improve or worsen the results.) Our auto-encoder used the same number of encode layers as our regular CNN, with each layer consisting of the same number of input channels and output channels as the corresponding

layer of the regular CNN (3→64, 64→32, and 32→3). The decode steps ran in the reverse direction (3→32, 32→64, and 64→3). However, each layer used a kernel size of 3×3 in the auto-encoder, while the kernel size varied among layers in the regular CNN.

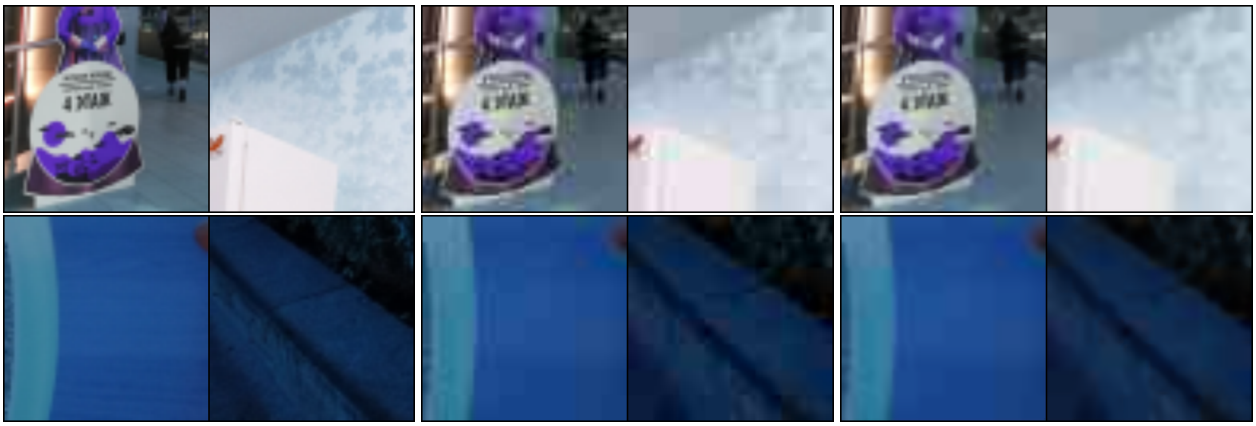
Both the convolutional auto-encoder and the more conventional CNN employed the Adam optimizer [DIEDERIK *et al* 2015], an efficient Stochastic gradient-based optimizer. We employed ReLU activation in the auto-encoder and Leaky ReLU in the CNN. Leaky ReLU allows a small positive gradient when the unit is not active, and in theory speeds up training. But we did not observe any benefit from it in the auto-encoder. We also employed PyTorch's ReduceLROnPlateau feature, which reduces the learning rate on more frequent features that stabilize faster, tending to allow more uncommon features to stabilize as well. We used MSE (Mean Square Error) as our loss function.

Results

In each of the following image sequences, the image on the left is the raw ground-truth undistorted source image. The second image is the distorted image. The third is our model's attempt at removing the distortion to restore the original.

Larger Dataset: CNN

JPEG:

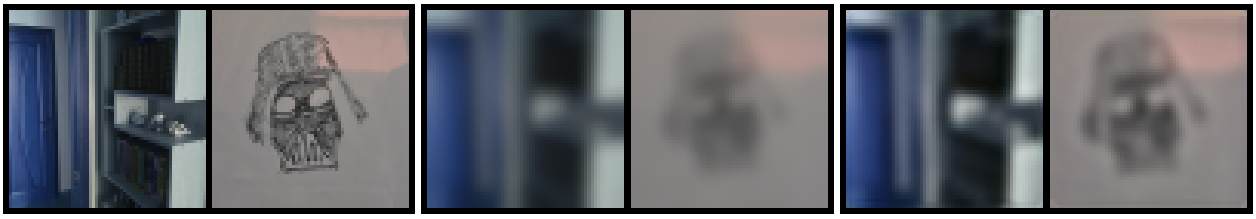


Blurring:

Smaller Dataset: CNN

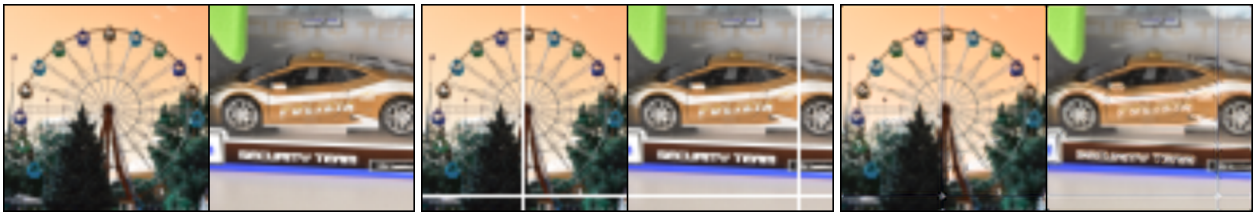
JPEG:

Blurring:



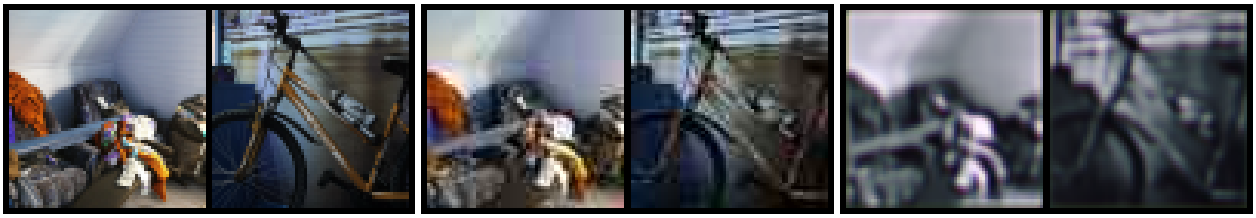
Noise:

Random Plus:

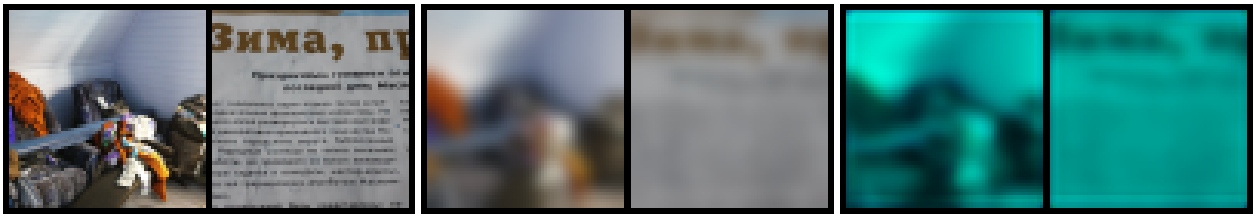


Smaller Dataset: Auto-Encoder with Padding

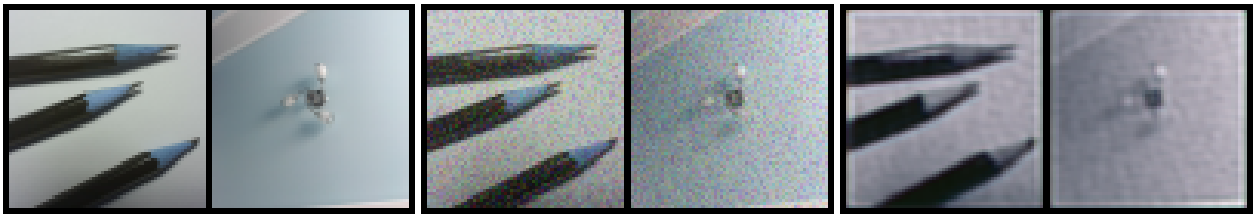
JPEG:



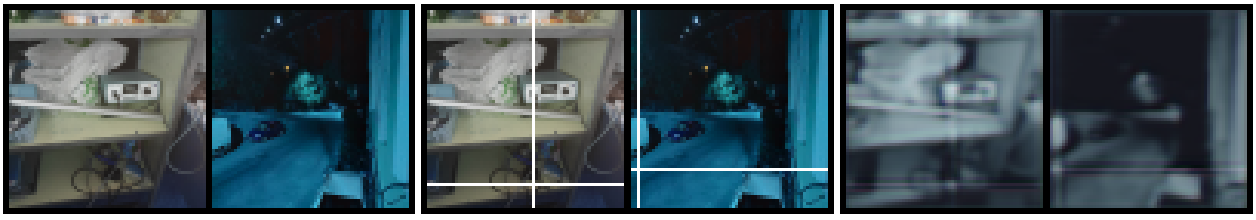
Blurring:



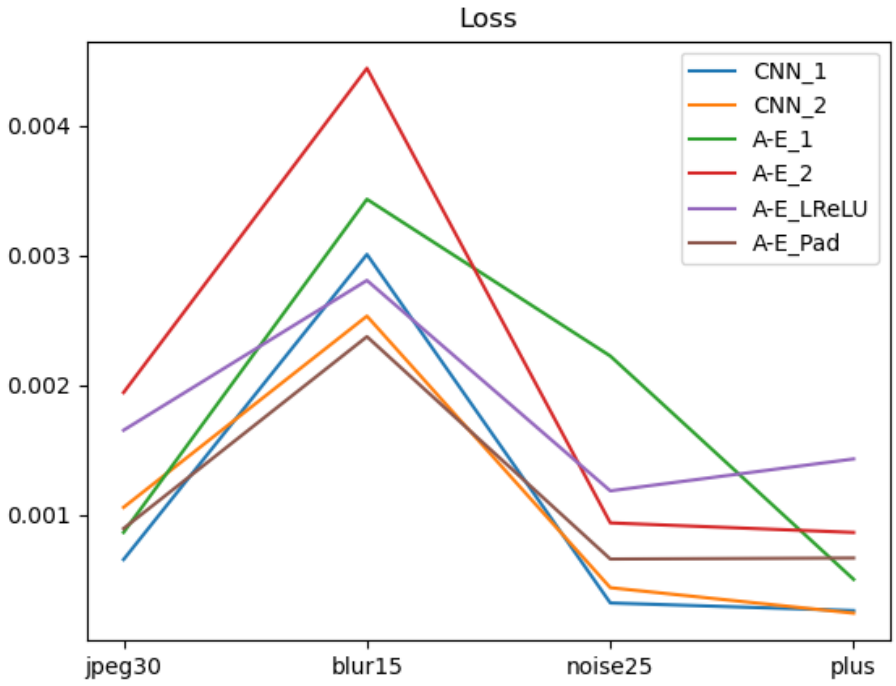
Noise:



Random Plus:



Loss by distortion type and architecture (two instances of conventional CNN, two instances of Auto-Encoder, an Auto-Encoder using leaky ReLU, and an Auto-Encoder with padding):



Discussion

We implemented a variety of distortion filters: JPEG compression artifacts, Gaussian blur, radial blur, horizontal and vertical lines at random positions, an X drawn through the image, and a three-channel random noise generator. After experimentation, we chose from these the representative set of four distortions and settings noted above.

We found that JPEG artifact removal was only slightly effective in both models. The model did smooth some of the blockiness characteristic of JPEG's 8×8 discrete-cosine transform blocks, but had no real effect on the noise around sharp edges that derives from JPEG's vector quantization stage.

More successful was the models' ability to sharpen Gaussian blurs and to remove lines drawn over the images. The blur removal was a solid improvement on the distorted image; its results closely resembled those of a Sharpen filter. The blur removal did not perform as well when applied to blurs with very large radii. We attribute that to the blur radius exceeding the dimensions of our convolution kernel. The results of our models on solid-line distortion were very good, leaving only a faint trace of the lines and a little splotch where the horizontal and vertical lines overlapped.

In general we found that the conventional CNN performed better, perceptually, than the auto-encoder. The auto-encoder tended to leave a ring around the reconstructed image, and it often changed the color significantly. Interestingly, both the conventional CNN and the autoencoder yielded similar values for loss. We attribute that to the choice of loss function. It could be that a more perceptual loss function, such as the structural similarity index (SSIM) or the multi- scale structural similarity index (MS-SSIM) [Zhao *et al* 2016] would

have performed better. But that exploration was outside the scope of this project.

In the absence of time constraints, we would have liked to train a set number instances of each type of model on each type of distortion, and to record the mean, median, and minimum loss among the model instances. More systematic testing in this way would have produced a clearer picture of accuracy variations and of each model's best-case performance.

Codebase

The codebase is hosted at https://github.com/howlium/cleanify.

To easily test the functionality described in this paper:

1. Open Google Colab in a web browser.
2. Open the following notebook:

   https://github.com/howlium/cleanify/blob/main/cleanify.ipynb.
3. Select **Runtime** -> **Change runtime type** -> **Hardware accelerator** -> **GPU**.

References

1. Rath, S. R., GitHub, image-deblurring-using-deep-learning (link).

2. Albluwi, F., GitHub, DBSRCNN. (link)

3. F. Albluwi, V. A. Krylov and R. Dahyot, "Image Deblurring and Super-Resolution Using Deep Convolutional Neural Networks," 2018 *IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2018, pp. 1-6, doi: 10.1109/MLSP.2018.8516983. (link)

4. Rath, S. R., "Machine Learning Hands-On: Convolutional Autoencoders," *Debugger Cafe: Machine Learning and Deep Learning.* (link)

5. Masci, J., U. Meier, d. Ciresan, and J. Schmidhuber, "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction," *Proceedings of the 21st International Conference on Artificial Neural Networks* 1 (2011): 52-59. (link)

6. Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization," International Conference on Learning Representations, 2015. (arXiv:1412.6980)

7. Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz, "Loss Functions for Image Restoration with Neural Networks," *IEEE Transactions on Computational Imaging*, 2016. (arXiv:1511.08861)