How Deep Can We Go? A Case Study on Image Classification

Erika Gutierrez, Shaney Sze
Barcelona School of Economics
June 10, 2023

1 Introduction

This paper seeks to contribute to the growing literature of deep learning model enhancement through data-centric approaches. Currently, much of deep learning research is focused on intricate model creation and optimization. Calls to shift focus to developing systematic engineering practices for improving data come from some of the top minds in the field, for example, Andrew Ngo's Campaign for Data-Centric AI. Furthermore, it has been shown that the performance of state of the art (SOA) image classification deep learning models is susceptible to corruptions in the images, adversarial or naturally occurring [4]. Therefore, research in how to make deep learning models more robust is also necessary. This brings us to our research question: Can we enhance the robustness of a SOA image classification model using deep learning informed data-centric techniques? We demonstrate a proof of concept to the answer of this question while simultaneously introducing a data processing pipeline framework that could be useful for industry.

2 Literature Review

Within the research of deep neural networks (DNNs) the topic of model robustness has started to receive considerable attention. The robustness of a DNN is its ability to accurately predict inputs outside the distribution of data it was trained on. In computer vision, input images considered outside the distribution of training data can be classified as either adversarially perturbed or corrupted. Adversarially pertrubed images are those with intentional manipulations to pixels (often unnoticeable to a human observer) with the purpose of degrading the performance of a DNN. Corrupted images are those with naturally occurring defects such as blurs, fog, or shot noise. The primary approaches to enhance model robustness include: enhancing model architecture, data augmentation, or custom model optimization [4]. Due to the data-centric approach we wanted to take in this paper, we focused our literature review on data augmentation techniques used to enhance model robustness. Devries and Taylor introduce a regularization technique called cutout, whereby a randomly selected set of pixels in the input imaged is masked during training [2]. Cohen et. al. use the k-NN clustering algorithm to detect adversarial images and remove them from the training data [1]. Xie et. al. introduce a denoising layer within their network architecture [7]. This is similar to our approach, but we have implemented image denoising to occur outside any network architecture. We do this because we believe that in order to make DNNs useful for industry application, model explainability is of key importance and breaking out DNN powered image processing steps instead of using one extremely intricate model architecture helps enhance explainability. Our research is focused on designing a human-in-the-loop frame work for deploying a DNN. We are not the only researchers who are exploring concepts like these, Lanthao et. al. designed an Optical Character Recognition (OCR) human-in-the-loop frame work to automate the processing of shopping receipts for the UK's Living Costs and Food Survey [6]. In their paper, they design a data pipeline where a Mask R-CNN (essentially a denoiser) is applied to scanned images of shopping receipts, those images are then ran through a DNN specialized in OCR in order to extract information from the

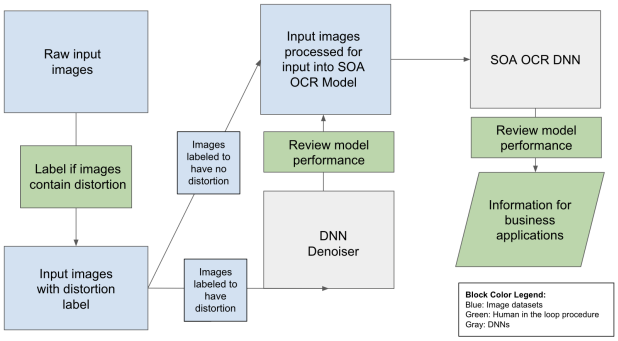

Figure 1: Proposed Image Processing Pipeline

receipts, and finally that information is passed through a machine learning classifier to group purchases to their 5-digit COICOP codes. The human-in-the-loop component of the process occurs during the scanning of the receipts and at various checkpoints along the process.

3 Methodology

Figure 1 shows our proposed data pipeline framework. In the pipeline, raw images are classified to have some type of distortion. The images that were identified to have a distortion are run through a DNN denoiser that outputs a "cleaned" image. These "cleaned" images are then placed back in the dataset of images that were identified to be clean. These images are then passed to an OCR DNN to extract information from the images. We have a proxy implementation of this pipeline in the Pipeline.py file of our Github repository. In this file we take different datasets containing a mix of clean and distorted images and perform inference on them using an SOA OCR model. The images that were not classified correctly are then passed to a DNN denoiser. The "cleaned" images as a result from this denoiser are added back to the dataset and ran through the SOA OCR model again.

3.1 Data

3.1.1 MNIST

We make use of two MNIST database datasets from the Datasets module of the Tensorflow package. The first dataset contains original clean images from the MNIST database. The second contains the same images but with different distortions applied. The distortions avail-

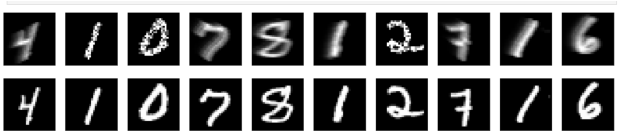

Figure 2: Examples of corrupted MNIST images with their clean counterparts

able are: shot noise, impulse noise, glass blur, motion blur, shear, scale, rotate, brightness, stripe, fog, spatter, dotted line, zigzag, and canny edges. We only query the Datasets module for the test images from these datasets, resulting in 1,500 images per each quality grouping. We receive a dictionary with two keys, one with a tensor containing the pixel values of the image and one with a tensor containing the scalar value of the digit label associated with each image. The dimensions of the images are 28 by 28 pixels along one channel (grayscale). We built our denoiser DNN to accept these dimensions as input. However, the OCR SOA model used did not accept these dimensions. The first layer in the model is a 2D convolution that accepts the dimensions (batch size, number of channels, height, width) and expects 3 channels and heights and widths of 244. Therefore we developed a dimension warping algorithm in order to run inference for the images. The warping algorithm contained the following procedures:

1. Normalize the pixel values by dividing them by 255
2. Convert Tensorflow tensor to Pytorch tensor since the SOA model was built using Pytorch
3. Reshape the array to move the channel dimension to the front
4. Add a batch dimension at the first position
5. Expand the image from 28x28 to 224x224 pixels using bilinear interpolation
6. Copy the grayscale dimension 3 times

To conduct error analysis on the OCR SOA Model we run inference on a set of breakouts of the MNIST testing sets (each breakout contains 1,500 images, see code in Data_Prep.py file for details). Table 1 shows the accuracy results of the model for each breakout.

Breakout	Accuracy
100% clean	99.00%
50% clean, 50% shot noise	97.93%
50% clean, 50% motion blur	83.86%
50% clean, 50% fog	80.86%
25% clean, 25% shot noise, 25% motion blur, 25% fog	82.06%

Table 1: MNIST quality breakouts and performance of SOA OCR Model pre image denoising

3.1.2 SVHN

Another dataset we explored is the Stanford Street View House Numbers (SVHN) Dataset. SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

The SVHN dataset images are MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides). It differs to MNIST as the image dimensions are 32 by 32 pixels instead of 28 by 28 pixels, which we have adjusted our denoiser models accordingly. Similar to MNIST, the SVHN has 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10.

We used the same method to extract the images and labels as we did for the MNIST dataset. We downloaded the dataset from HuggingFace and loaded the TensorFlow objects and parsed them into image and label dictionaries, which were then fed to the model.

3.2 Models

3.2.1 Image Classification SOA

The model we are using to conduct our proof of concept data pipeline is called vit-base-mnist which is a fine tuned version of Google's vit-base-patch16-224-in21k on the MNIST Dataset. The vit-base-patch16-224-in21k is a transformer encoder image classification model trained in a supervised fashion on the ImageNet-21k

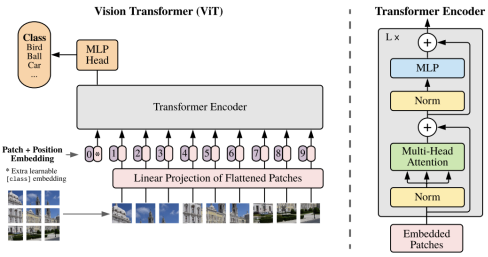

Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Figure 3: vit-base-patch16-224-in21k model structure from the original paper [3]

dataset. This dataset contains 14 million images encompassing 21,843 classes at a resolution of 224x224 pixels.

To find this model we searched the leaderboard for "Image Classification on MNIST" on paperswithcode.com. On the MNIST evaluation dataset, it has a loss of 0.024 and an accuracy score of 0.995. The vit-base-patch16-224-in21k was one of the first implementations of a transformer for image classification tasks. It is based on the Visual transformer (ViT) architecture. Similar to its natural language processing counterpart, the ViT adds a learnable embedding to the input vector that will contain the image representation upon arriving at the end of the transformer.

3.2.2 Denoising Autoencoder

The autoencoder neural network is a technique which uses the extracted latent space from an input source to reduce the image size for storage, and then reconstructs the source within an acceptable loss range for use.

An autoencoder maps its input, usually an image, to a latent vector space via an encoder function, and then decodes it back to an output that is the same as the input, via a decoder function. It is trained to reconstruct the original input. By trying to minimize the reconstruction MSE error, on the output of the encoder, you can get the autoencoder to learn interesting latent representations of the data. Historically, autoencoders have been used for tasks such as dimensionality reduction, feature learning, and outlier detection.

There are 4 main types of autoencoders:

1. Sparse Autoencoders: Where the hidden layer is

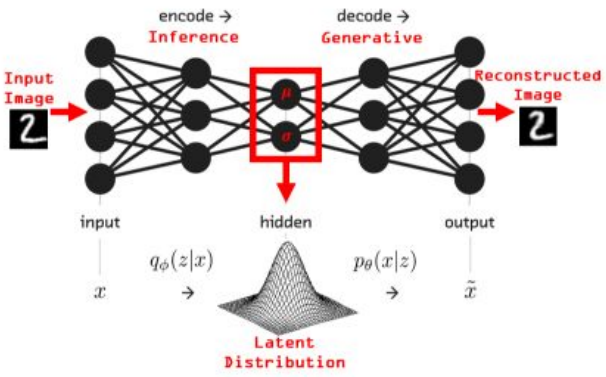

Figure 4: Workflow of Denoising Autoencoders [5]

greater than the input layer but a regularization technique is applied to reduce overfitting. Adds a constraint on the loss function, preventing the autoencoder from using all its nodes at a time.

2. Denoising Autoencoders: Another regularization technique which handle noisy input by reconstructing clean versions of corrupted data.

3. Contractive Autoencoders: Adds a penalty to the loss function to prevent overfitting and copying of values when the hidden layer is greater than the input layer.

4. Stacked Autoencoders: When you add another hidden layer, you get a stacked autoencoder. It has 2 stages of encoding and 1 stage of decoding.

Denoising autoencoders uses corrupted data as input and the uncorrupted data as the output. The input feature vector x is corrupted by randomly selecting some of the input values and setting it to zero. The corrupted input vector is denoted by $\hat{x}$. The Denoising Autoencoder takes the corrupted noisy data and try to de-noise it. See figure 4, hence it is called Denoising Autoencoder.

4 Results

4.1 MNIST

4.1.1 Denoiser Training

For the MNIST dataset, we utilized a Convolutional Neural Network (CNN) model to address the issue of noise

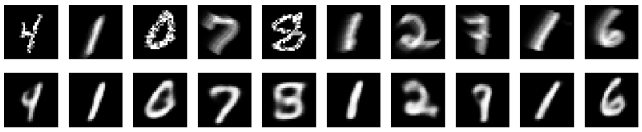

Figure 5: Constructed MNIST Images from Denoising Autoencoder model

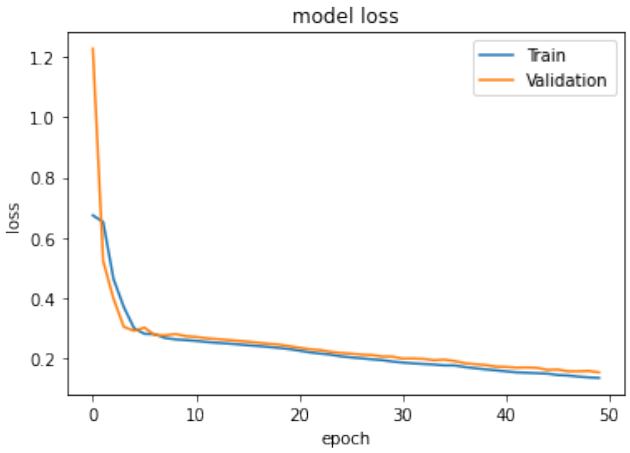

Figure 6: Model loss of MNIST Denoising Autoencoders model

in images. Specifically, we focused on denoising autoencoders. We worked with a corrupted MNIST dataset from Tensorflow Datasets. Initially, we obtained a sample of images from the dataset, consisting of both clean and noisy versions. The clean dataset comprised the uncorrupted images, while the noisy dataset included images corrupted by either motion blur or shot noise.

To create the noisy dataset, we alternated between selecting images corrupted by motion blur and shot noise, randomly choosing between the two for each image. We proceeded with the training and testing phases of our denoising autoencoder model. The ConvDenoiser model employed Conv2DTranspose layers to perform the denoising task. These layers enable the model to upsample the input data, gradually reconstructing the clean image representation.

In figure 5, we show the constructed MNIST images. In the top row, we have the original training images of corrupted handwritten digits images. The row below shows the output of the encoder model, which was trained on the noisy images and and generates the images. Figure 6 shows the model loss, which converges at `0.1365` after `50` epochs.

4.1.2 Inference

Breakout	Accuracy
100% clean	99.00%
50% clean, 50% shot noise	97.93%
50% clean, 50% motion blur	84.06%
50% clean, 50% fog	81.46%
25% clean, 25% shot noise, 25% motion blur, 25% fog	82.86%

Table 2: MNIST quality breakouts and performance of SOA OCR Model post image denoising

Table 2 shows the results of the SOA OCR model on the combination of clean and "cleaned" MNIST images. No improvements were made for the breakouts where the model already had high performance (100% clean and 50% clean and 50% shot noise). For the rest of the breakouts the denoising improved the SOA OCR model performance by on average .53%. The breakout where it had the most impact was the breakout that included images from all the different distortions tested.

4.2 SVHN

For the SVHN dataset, we first set a baseline score using a Convolutional Neural Network (CNN) model, which gave us an accuracy score of `77.0%`. The CNN model consists of the following layers, and is built in the `model.py` file. For this dataset, we randomly sampled `10,000` images for training and `2,000` images for testing. For our SVHN model, the model's parameters are set as:

```
input_shape = (32, 32, 3)
batch_size = 16
validation_split = 0.2
epochs = 15
```

and the model was created as follows:

```
model = Sequential([
    Conv2D(32, (5,5), activation='relu',
        padding="same", input_shape=
        input_shape),
    BatchNormalization(),
    Conv2D(64, (5,5), activation='relu',
        padding="same"),

    Dropout(0.2),
    MaxPooling2D((2,2), strides=2),

    Dense(32, activation='relu'),
    Flatten(),
    Dense(10, activation='softmax')
])
```

We then compare our score of the baseline CNN model to the denoising autoencoding CNN model, which transforms the previous model by adding an encoder and a decoder. Since our inputs are images, it makes sense to use convolutional neural networks (convnets) as encoders and decoders. In practical settings, autoencoders applied to images are always convolutional autoencoders. The encoder will consist in a stack of Conv2D and MaxPooling2D layers (max pooling being used for spatial down-sampling), while the decoder will consist in a stack of Conv2D and UpSampling2D layers. We achieved a `79.5%` accuracy with the denoising autoencoder model, which was higher than the baseline CNN model. The loss and accuracy graphs are shown in Figure 7. In Figure 8, we show the probabilities of the correct labels given each test image.

Finally, the scores between the two models are compared graphically in Figure 9. The denoising autoencoder overall does better than the CNN model in the SVHN dataset.

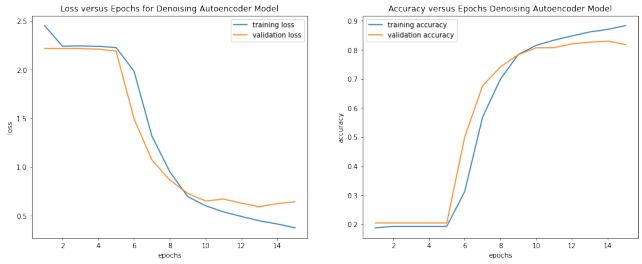

Figure 7: Loss and Accuracy vs. Epochs on SVHN Denoising Autoencoder model

5 Conclusion

In conclusion, this paper contributes to the literature on enhancing deep learning models through data-centric approaches. We delved into the application of deep learning techniques, specifically denoising autoencoders with Conv2DTranspose layers, to tackle the issue of image noise. By utilizing a corrupted MNIST dataset and training our model on both clean and noisy images, we successfully showcased the model's ability to reconstruct clean representations from noisy inputs.

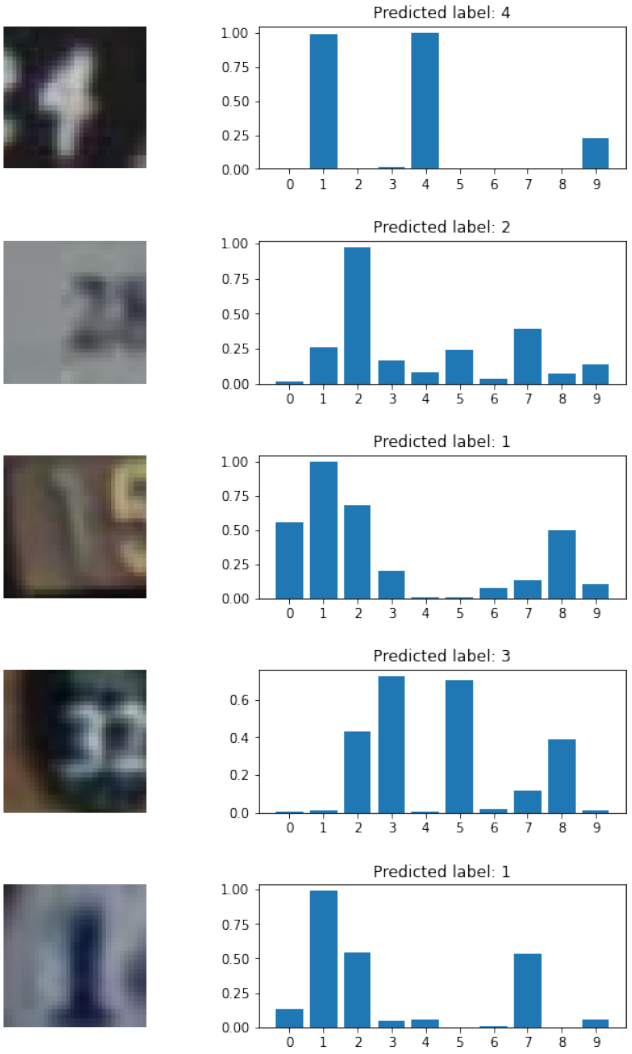

Figure 8: Predicted Labels for SVHN Denoising Autoencoders

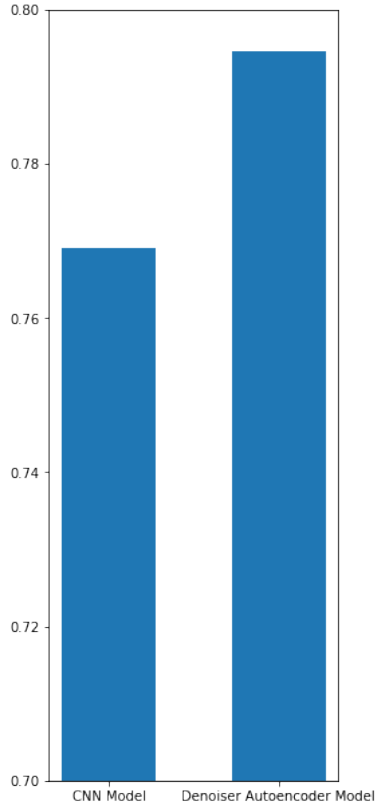

Figure 9: Scores between CNN and SVHN Denoising Autoencoder model

Our experimentation yielded promising results in terms of denoising performance. However, it is essential to emphasize that the duration of training and convergence greatly influence achieving optimal performance. Extending the training period holds the potential for further improvements in denoising accuracy and reconstruction quality.

Moreover, the exploration of techniques like early stopping and adaptive learning rate schedules can help strike a balance between training time and model performance. These strategies effectively prevent overfitting while maximizing the denoising capabilities of the model.

This study serves as a foundational stepping stone for future research and experimentation in deep learning-based denoising techniques. By prolonging the training time and exploring additional optimization strategies, researchers can continue to push the boundaries of denoising performance. This progress will enable applications in diverse domains where noise reduction plays a critical role in ensuring accurate image analysis and interpretation. Furthermore, it is worth exploring other variants of autoencoders such as Contractive Autoencoders to expand the scope of investigation.

Lastly, it is important to note that the computation in our experiments was conducted in a serial fashion. To significantly reduce training time through parallel processing, we encourage the use of GPUs, which can offer substantial computational advantages.

References

[1] Cohen, G., Sapiro, G., and Giryes, R. Detecting adversarial samples using influence functions and nearest neighbors, 2020.

[2] DeVries, T., and Taylor, G. W. Improved regularization of convolutional neural networks with cutout, 2017.

[3] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[4] Drenkow, N., Sani, N., Shpitser, I., and Unberath, M. A systematic review of robustness in deep learning for computer vision: Mind the gap?, 2022.

[5] Kumar, V., Nandi, G., and Kala, R. Static hand gesture recognition using stacked denoising sparse autoencoders. In *2014 Seventh International Conference on Contemporary Computing (IC3)* (2014), pp. 99–104.

[6] Lanthao Benedikt, Chaitanya Joshi, L. N. N. d. W., and Schouten, B. Optical character recognition and machine learning classification of shopping receipts, 2020.

[7] Xie, C., Wu, Y., van der Maaten, L., Yuille, A., and He, K. Feature denoising for improving adversarial robustness, 2019.