Image Denoising Using Autoencoders

Term Project

Nicolas Bernier

40088104

Ana Lucio

40032548

COMP 478/6771

A. Ben Hamza

April 25th 2021

Image Denoising using Autoencoders

Abstract – **Image denoising consists of estimating the original and sharper version of an image from a noisy and degraded source. Convolutional Neural Networks (CNN) or Autoencoders have been shown to be a worthwhile method to treat image data. This deep learning approach differs from traditional image processing methods in its ability to perform more evenly across a wider range of degradation problems. The method consists of flattening the 2D image into a 1D array with each element representing a pixel in the image. This is used as the input. The CNN consists of hidden layers which deconstructs the data, then attempts to infer features and reconstruct it [1]. In order to validate the effectiveness of the method we tested it over images from different open source datasets. The method is able to improve quality more consistently than the traditional methods alluded to further in the reading, which each only apply to a strict subset of the noise and deterioration patterns. The implementation below is shown to be transferable from one dataset to the next with minimal changes.**

I. INTRODUCTION

The process of image restoration has a wide range of applications ranging from forensics to coloring for older pictures. It is a term that groups together techniques such as denoising and complex degradation estimation [2]. For algorithms to properly address the full range of user needs, they need to not only have the ability to identify the various types of degradation affecting an image, but to also adjust responses and parameters in order to perform better across the problem domain.

For instance, noise, which is qualitatively characterized as random dots with varying intensity obscuring image details [3], can appear in images with varying structures. In order to counter its effects on the image, this structure must be estimated or inferred by the program treating the image. Deep learning is a section of computer science that naturally has the ability to solve these types of challenges.

To address the various roadblocks involved in image denoising, we propose a denoising autoencoder, a particular algorithm using deep learning for image restoration.

Autoencoders are convolutional neural network architectures consisting of two subnetworks, the encoder and the decoder, linked to each other by the latent space. The encoder processes a noisy image and encodes it to fit the latent space. This data is used by the decoder to generate a clean image on the output side. Autoencoder training primarily consists in designing an efficient latent space to generate the inverted version of the encoder network [4]. Processing image data requires a neural network for modeling image data, namely, Convolutional Neural network (CNN) or Convolutional Autoencoder. Convolutional Autoencoders are suitable for image restoration as they are built to extract specific information through a convolution layer while keeping the spatial information of the original image data [1]. To summarize, the main contributions of this paper are as follows:

We introduce the convolution layer which involves a feature map preserving the relationship between pixels and the input image. By scanning the original picture, a mathematical operator determining a feature is applied to each pixel changing the value of that pixel [5]. The original image is filtered using scores assigned to the parts of the image that match our specific feature. Features score a perfect match through padding. Scanning input image data using various different filters allows the model to extract a variety of features. However, more filters require longer

training time. Therefore, we use the minimum amount of features possible for this paper.
We present the Leaky reLu function as an activation function. This function is one of the possible newer activation functions based on a ReLu, or Rectified Linear Unit, which attempts returning zero for identified negative values while directly returning positive values resulting in a linear series of positive values [6]. Leaky ReLu helps increase the range of the ReLu function by allowing a leak in the calculation of negative values. The standard ReLu function has a gradient of 0 for all the values of inputs that are less than 0, which deactivates the neurons in that region. As a solution to this problem, the Leaky ReLu function defines an extremely small linear component of x in the following formula:
$f(x) = max(0.01 * x, x)$. This function returns x if it receives a positive input, but for negative values of x , it will return a small value which is 0.01 * x. This yields an output for negative values as well in order to avoid dead neurons.

We demonstrate the Batch Normalization which is used to apply a transformation that maintains the mean input close to 0 and the output standard deviation close to 1 in order to avoid large differences in the dynamic range of the values of various inputs resulting in an issue in neural networks called internal covariance shift [7]. During training, the layer normalizes its output using the mean and standard deviation of the current batch of inputs. This is not the case when Batch Normalization is not applied. When computing Batch Normalization we use an axis and preserve the dimensions of the array. After implementing the convolution-layer, the dimensions of the figures in our dataset are normalized along the last axis, which in our case is the channel. This is done, because we want the shape of the values encoded in this channel. By default axis = channel dimension of -1 for pure images [7]. Batch Normalization does not only work as a regularizer. It also speeds up training, on account that the outputs of every layer changes. This leads to a distribution of correct input data for every following layer during every iteration [7]. Thus, showing efficient representations of the proposed model on the MNIST, MNIST Fashion, FER-2013 and VinBigData datasets.

The presentation of this paper is organized as follows. First, we introduce related topics and traditional approaches to solving the problem of image denoising. Then, we present the detailed structure of the proposed Autoencoder. Lastly, we demonstrate the efficiency of the Autoencoder by running it over various datasets. Within this last section, we also demonstrate the steps that were taken in order to normalize and seperate the datasets in training and test groups.

II. RELATED WORKS

To properly restore an image affected by degradations such as noise, the application must identify the patterns proper to the particular image. Different approaches exist involving the use of convolutional neural networks. *Jain et al.* put forth a method, which draws on the complexity of neural networks and their adjustable parameters to outperform methods like Markov's random fields (MRF). The paper discusses the use of thresholding in order to estimate the performance of the model compared with other methods like MRF [8]. Depending on the application, there are many measures by which to judge a model. To test the performance of convolutional networks with a very simple architecture, a dataset of electron microscopic images of neural tissue was used. Through this work, we recognize that the restoration of image data could possibly encounter some difficult computer vision problems due to the small size of pixels, dense packing of structures, and noise caused by imperfection in the imaging. In order to address these problems, *Jain et al.*[8]. extends the idea of a noisy image thresholded to produce binary restoration and compare it to a real restoration of that same image. The error value associated with the training set is found

and the noisy test image is thresholded at this value to then, again compare it with a real restoration.

The problem with this strategy is that the algorithm is constructed to differentiate between light or dark regions but misinterprets regions that are not of uniform intensity. Therefore, some regions were wrongly classified by this model. A convolutional network is proposed as a solution that outperforms thresholding. An architecture built upon an input layer that encodes input images and an output layer that encodes output images. This model is comparatively narrow and requires five hidden layers. Each layer receives input from the original image as well as information from only one previous layer. The model is trained with respect to adjustable variables in accordance to filters, biases and offsets. Therefore, convolutional network restoration calculates exactly the gradient of the objective function for learning by relying on the principle of error minimization resulting in fewer errors than the two former models. Consequently, the approach provides better performance, most importantly, in high frequency regions of the images. In the following section we explain our proposed implementation of a convolutional neural network by presenting the motivation and problem behind our proposed architecture, the different layer components of our model, and our model training implementation.

III. PROPOSED METHOD

In this section, we stress the motive behind introducing the denoising autoencoder. We present the important building blocks of our proposed implementation.

A. Motivation and Problem Statement

Denoising or noise reduction is one of the many applications of autoencoders. The main motivation behind this implementation is to achieve good representation through decomposition and reconstruction of the image and allowing the model to identify features. Denoising autoencoders allow for the prediction of an unblemished original image from one that is affected by noise. This has many implications as a component of larger applications. For instance, computer aided diagnosis which leverages computer vision and feature extraction in order to arrive at some conclusion about the patient. Weakly labeled data can also benefit from such a technique as part of processes to provide better input to other deep learning models.

Problem Formulation:

Given the data vector g , which distorts an image represented as a matrix H. The image restoration problem involves solving for the unknown vector f that will produce an image free of deterioration. The primary theoretical approach to solve for the unknown vector is to apply a simple matrix inverse. However, in practice this approach unfortunately does not lead to usable solutions. Problems estimating f' arise due to four specific factors. Firstly, due to the presence of noise in an image, there might not be an existing f for a given observation in g. Secondly, in cases in which the magnitude of filter H (i.e high-pass filter) approaches zero in a set of constant images, there could be a problem of solution uniqueness. In these cases, many choices of f that produce the same set of identical observations from which it must be decided which one is the right one. Third, the estimation of f might remain the same regardless of perturbations in observations due to noise or numeric roundoff. Fourth, the standard approach to inverting H says nothing about prior knowledge about the solution. A way to include more information in the solution is needed [9].

B. Proposed Neural Architecture

In response to the image restoration problem mentioned above, we have concentrated on the application of successive

approximation iterations to restore noisy observations to recover fine-grained details of input images. We demonstrate this solution by extracting and composing robust features with a denoising autoencoder. The robustness of internal layers relies on purposely introducing random or gaussian noise to the original images. Using an encoder-decoder network we pre-process the image, improve the quality and increase the accuracy of the output data. The ConvAutoencoder class takes four arguments, namely, the size in height and width of the input image in pixels, the depth which refers to the number of channels of the input volume, a tuple (32, 64) representing the set of filters for convolution operations, and the number of neurons in our connected latent vector (by default set to 16). The input shape is initialized to the height, width, and depth passed to the ConvAutoencoder class. Thereby, defining the Input to the encoder. We use 32, 64 filters for the convolutional layers in the encoder blocks. Each block in the encoder consists of a 3 x 3 convolutional layer followed by a leaky ReLu or leaky rectified linear unit activation. The convolutional layer applies a filter to an input to create a feature map that outputs a summary of the features present in the input. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. BN takes a normalized axis as parameter equal to the channel dimension (by default set to -1 as we are implementing images).

On the other hand the decoder performs the reverse of the encoder. We loop over the filters in reverse order with the help of a 3 x 3 convolution transpose layer followed by a leaky ReLu/ leaky rectified linear unit activation. The batch normalization layer with the axis set to the default channel dimension. We use 64, 32 filters for the convolution layer which correspond to the number of features maps. The final step in our proposed neural architecture is to recover image depth by implementing a 3 x 3 convolutional transpose layer.

C. Convolution Layer

The convolutional layer creates a convolution kernel that is convolved with the layer input to produce outputs. This layer is used as the first layer in our model. The model iterates over the filters and assigns the 3 x 3 convolutional layer kernel convoluted with the layer of inputs to *temp*. The *Conv2D* function takes four arguments namely, the dimensionality of the output space, the kernel size(3x3), the number of strides of the convolution along the height and width, and the padding, which in this case is set to *same*, as the padding should be even to the left/right and up/down of the input such as the output has the same height/weight dimensions.

D. Leaky ReLU Layer

In neural networks, the activation function is used to transform the summed weighted input from one node into the activation of the output. The rectified linear activation function is a piecewise linear function that will output the input directly if it is positive. Otherwise, it will output zero. This function has become the default activation function for many types of neural networks because it facilitates training and enhances performance of the training model.

The Leaky ReLU implementation modifies the function to allow small negative values when the input is less than zero. By allowing this small modification, the gradient of the left side of the graph comes out to be a non-zero value. Thus, the purpose of the Leaky ReLu is to solve the dying ReLu problem in which the gradient is 0 for all the negative values leading to the deactivation of the neurons in that specific area. The ReLU function returns the standard ReLU activation *max(x,0)*. X refers to the input tensor and 0 refers to the element-wise maximum of 0 while the Leaky ReLu function returns x for any positive value, but for any negative value it returns a really small value which equals to 0.01 * x. The

difference between the two functions are visually depicted in the figure below.

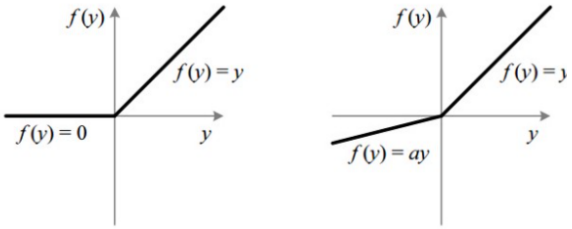


Fig : ReLU v/s Leaky ReLU

Fig[1]: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Moreover, the LeakyReLU function helps to increase the range of the ReLu function. By increasing the range from [0 to infinity) to (-infinity to infinity), any negative input given to the LeakyReLU function will be mapped as *f(y) = ay*, where a is by default 0.01. One of the consequences of implementing the Leaky ReLu function in neural networks is that it helps the model to become a lot sparser as the training process tends to be impacted only by the features in the dataset that actually contribute to the model's decision power. The Leaky ReLu function does not perform better than the traditional ReLu function which is normal in our model since the leaky variant is only expected to work on cases where there are many dead neurons and our model is a small scale representation. The result of both functions on the MNIST dataset is shown the figures below:

Our model's results using Leaky ReLu on the MNIST dataset.

Our model results using ReLu on the MNIST dataset.

E. Batch Normalization

Our primary assumption was to use a pooling layer in order to select the largest value on the feature maps. Pooling layer is used in neural networks to downsample or reduce the size of each feature map by a factor of 2. However, using the pooling layer returned an object of the pool size passed to the ConvAutoencoder class resulting in the InputLayer taking a value of [(None, 28, 28, 1)], in the case of the MNIST dataset, and the decoder taking a value of (None, 2, 2, 1).

To fix this issue, we decided to use Batch Normalization to standardize the inputs to a layer for each batch. Our implementation of the Batch Normalization would make our autocoder more stable during training. While the pooling layer operates on each feature map independently[15], Batch Normalization normalises a InputLayer by subtracting the batch mean and dividing it by the batch standard deviation. The normalization ensures that the inputs have a mean of 0 and a standard deviation of 1 [14].This will ensure that the input distribution is the same throughout the network. Given a dataset with a un-normalized input distribution, gamma and beta in the formula below, will converge to √Var[x] and E[x] ensuring an optimal normalisation for the given dataset.

$$y = \gamma \cdot \hat{x} + \beta$$

By using Batch Normalization, we do not only normalize the inputs to the network but we normalize the inputs to layers within the network. During training, each layer's input is normalized by using the mean and variance of the values of the current batch. The benefits of Batch Normalization on our model as a whole include faster training, higher learning rates allowance, viable activation function and simplified creation of deeper networks. Networks train faster as Batch Normalization allows faster convergence of each iteration even if iterations are slower due to extra calculations during the forward pass [17]. As the network gets deepened, Batch Normalization allows higher learning rate which further increases speed at which networks train. Usually gradient descent requires small learning rates to converge which leads to more iterations and a decrease in training speed[17]. Batch Normalization also makes our activation function viable as sigmoids tend to lose their gradient quickly and cannot be used in deep networks. Leaky ReLu/ ReLu functions tend to also die during training if the wrong range of values are fed into it [17]. Batch Normalization helps us to regulate values throughout the network by making non-linearities viable again. This regulation of values also simplifies the creation of deeper networks and helps the model to build and faster to train.

F. Model Training

As in traditional machine learning workflows the model is trained with a *fit(...)* function. The model iterates over a number of samples per gradient update [10]. It also performs the training and adjusting its internal parameters after each iteration. The number of iterations was chosen around the point where the marginal improvement to the model nears reasonably close to 0. In this case the number of iterations is specified to 50 and the size of the sample for each training is 32.

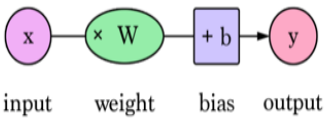
In this case it is the mean squared error (MSE). Mean squared error can be expressed mathematically as:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}||\hat{y}^{(i)} - y^{(i)}||^2$$

The function can be described as the difference between predictions and the ground truth, squared and averaged across the whole dataset [11]. It is also common in multiclass predictions as is our case. For this use case, it performed better than other traditional loss functions such as the cross-entropy, meaning that it produced more accurate results both quantitatively and qualitatively.

The Adam optimizer with a learning rate of 0.001 was used for its computational efficiency over other optimizers [12].

As the model performs the training, its parameters are tweaked in order to minimize the loss function specified. The parameters that are ultimately used in order to make predictions are those of the iteration which produced the least amount of validation loss. Parameters in the diagram are referred to as weights. The output is produced by multiplying the parameter with the input, which is a gradient element. That result is then sometimes added to a bias in order to produce the output. The loss for that element can then be evaluated using the loss function above using the autoencoder's output and its corresponding element in the image used as ground truth image.



After the model is trained, we then use it in order to make predictions on images that it has not seen before.

IV. EXPERIMENTS

This section is meant to walk through the experiments conducted that allow us to understand the proposed model's performance.

A. Datasets

Each dataset will be composed of grayscale images. This allows for an easier qualitative comparison between the model's inference and the actual base image. The Autoencoder requires a constant input in terms of image size. Therefore, we have implemented a step to resize all images of a given dataset to a constant size.

Here is a brief description of each data set, as well as, the source used to gather the images in said dataset:

- **MNIST**: Set of 60 000 training and 10 000 test images. Each image's size is 28x28 pixels. Each image is that of a handwritten number between 0 and 9. In order to use the dataset in Google Colab, we must simply import it from keras.datasets.

- **MNIST FASHION:** Set of 28x28 pixel images. The dataset is composed of grayscale images. The images represent clothing items such as shirts, pants and shoes. In order to use the dataset in Google Colab, we must simply import it from keras.datasets.

- **FER-2013:** Set of 48x48 pixel images. This dataset is composed of labeled images of people's faces. It is usually used in order to train models to recognize a limited set of human emotions. In order to use this dataset in Google Colab, we must first store it in a folder in our Google Drive. This dataset can be obtained from kaggle here: https://www.kaggle.com/msambare/fer2013

- **VinBigData:** Set of 512x512 pixel images. The dataset is composed of chest radiographies. The dataset is annotated in order to, most commonly, train models to identify thoracic abnormalities. The size of the images needed to be brought down to 52x52 pixels using cv2's INTER_AREA interpolation. This needed to be done due to the limited RAM available on Google Colab. This dataset can be obtained from kaggle here: https://www.kaggle.com/awsaf49/vinbigdata-512-image-dataset

B. Baselines

MNIST & MNIST FASHION :
The workflow includes a step to add synthetic noise to images with a noise coefficient of 0.5. Setting the coefficient to 0 results in no noise added and setting it to 1 results in maximum noise added. In order to produce usable and realistic results, the coefficient should be set between 0.3 and 0.7 for these two datasets. Two examples are shown here:

Understandably, the original noiseless images are used as baselines to their corresponding noisy image.

Datasets from KAGGLE:
Since the API is different from Keras' datasets, another method is required in order to add noise. For both VinBigData and FER-2013, gaussian noise is added. The level of noise is controlled by the mean and variance parameters. The mean is set to 0 and the variance to 550. These parameters produced images with reasonable decay.

FER-2013 Data Augmentation:
FER-2013 has a much larger variety of images. To produce qualitatively acceptable

results the dataset must be augmented. Prior to the randomized train/test split and gaussian noises, 100 copies of baseline images were taken. This produces a dataset that is 100 times larger and whose corresponding noisy images are all different.

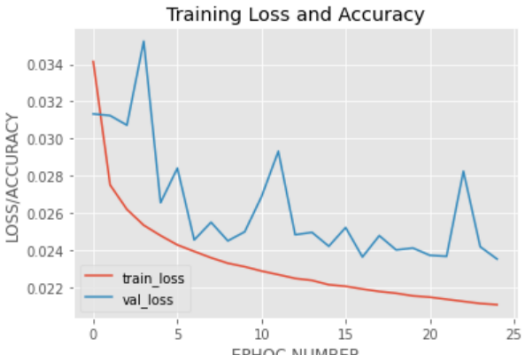
C. Evaluation Metrics

In order to evaluate the fitting performance of our proposed network against baseline methods, we use the mean square error as a loss function.

The mean square error loss function was used on the four datasets, that is, MNIST, MNIST_FASHION, FER-2013, and VinBigData. We decided to use mean square error as a loss function because each of the values that the neural network will predict for these datasets will be image pixel values which are numbers. The means square error function calculates the dissimilarity between the original pixel values and the predicted pixel values [16]. In order to evaluate our model we use our train and test data. In this section, we compare the prediction of the model to the original image before any noise is added to it. In general, if our training loss is smaller than the validation loss, it indicates some level of overfitting. If the training loss is greater than the validation loss, there it indicates some level of underfitting. If the training loss is equal to the validation loss, then we have a perfect fitted model [11]. Training loss refers to the error on the training data set while validation error refers to the error after running the validation set of data through the trained model [11]. Ideally, as epochs evolve, the validation loss decreases as the model learns the data better and better. As shown in the figures below, we can see that our model learned the problem of achieving more or less zero error over 50 training epochs. We can see that in the four datasets, the model converged reasonably quickly and both train and performance remain equivalent. This suggests that the mean squared error is a good match for our model learning this problem.

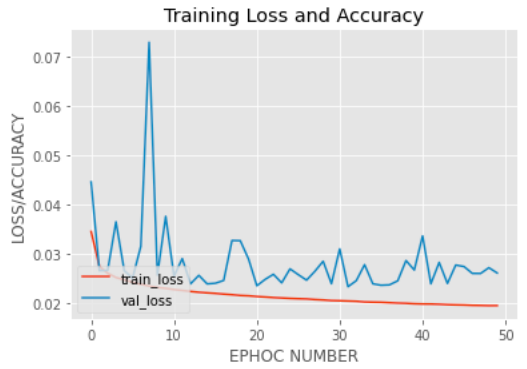
MNIST FASHION

Originally, the number of EPOCHS was set around 25. This yielded the following loss graph:



Epoch 25/25 - loss: 0.0209 - val_loss: 0.0235

Training the MNIST Fashion for 25 EPOCHS was not enough since the slope of the training loss was not nearly close enough to 0 at the twenty fifth iteration. Also, the validation loss still showed high amounts of volatility. Therefore, this dataset was instead trained for 50 EPOCHS. This produced the following graph:

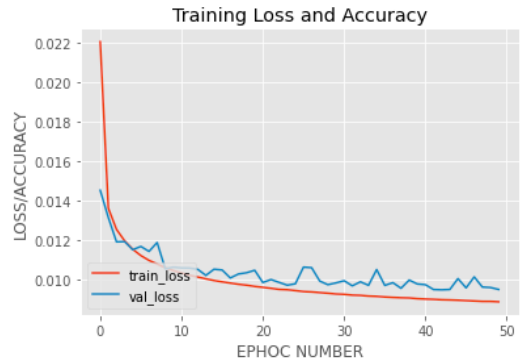


Epoch 50/50 - loss: 0.0193 - val_loss: 0.0262

The volatility in validation loss lowers as the EPOCH number increases. This occurred at different points for each dataset. However, resulting graphs showed evidence of visibly significant improvement when the number of

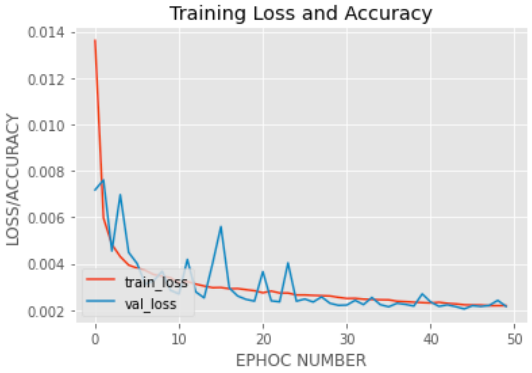
EPOCHS was set to 50. This resulted in longer fitting time.

MNIST



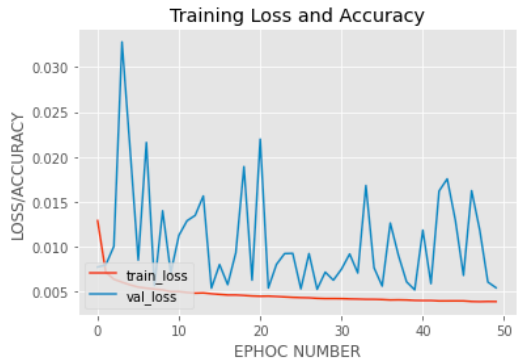
Epoch 25/25 - loss: 0.0088 - val_loss: 0.0095

FER-2013



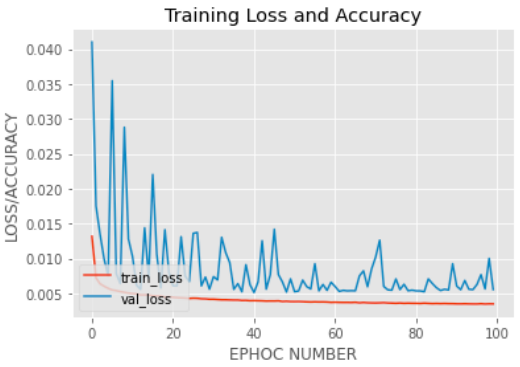
Enter epoch 50/50 - loss: 0.0022 - val_loss: 0.0022

VinBigData



Enter epoch 50/50 loss - loss: 0.0038 - val_loss: 0.0054

Increasing the number of EPOCHS for VinBigData to 100 yields a graph which finally demonstrates a lower validation loss volatility. This graph is shown here:



However, in order to maintain consistency as well as low processing times across datasets, 50 EPOCHS is maintained for the model fitting process.

.

D. Image Sharpening

As is the case with traditional methods, denoising an image, the use of an Convolutional Neural Network will result in a qualitatively blurred image. Naturally, the next in many workflows involving such a simple and portable Autoencoder is to perform some sort of image sharpening operation. In this case, the following 3x3 high-pass filter was applied to the image predicted:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Almost invariably, applying this filter creates an image whose edges are visually closer to the ground truth image than the model's inference prior to the high pass filter being applied.

E. Predictions & Qualitative Results

The model was used to predict baseline images from various datasets. As described above, the workflow for each dataset is roughly the same. The model's *.predict([...])* method takes an array as an input and outputs its predict for each corresponding element in the form of an array.

MNIST

The baseline images for the MNIST dataset are all binary. However, grayscale noise was added rather than salt and pepper noise. This was done in order to demonstrate the model's ability to closely predict the image's ground truth when affected by a more general noise pattern. Qualitatively, the model performed well over the entire domain of the dataset.

Shown below are three examples of predictions the Autoencoder made on noisy images.

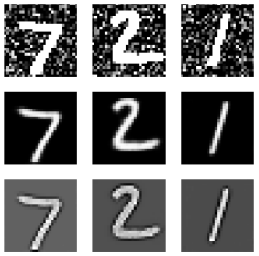


Figure X: Denoising Autoencoder result from test on the MNIST dataset. From top to bottom, the rows represent noised image, mode output and sharpened output.

MNIST FASHION

The MNIST FASHION dataset is composed of grayscale images, to which grayscale noise was added.

Consistently, the model produced results very close to ground truth images. Some information found in the base image seemed lost due to noise being introduced. A logo found in the ground truth image which was lost in the corresponding noisy instance found at the top of the second column, in the figure below, was recovered from the model's prediction. Shown below are three examples of predictions the Autoencoder made on noisy images.

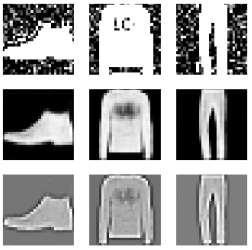


Figure X: Denoising Autoencoder result from test on the MNIST FASHION dataset. From top to bottom, the rows represent noised image, mode output and sharpened output.

FER-2013

Due to the size of the dataset and the limited resources of Google Colab, only the "disgust" faces were used. The FER-2013 dataset was the most complex. Human faces vary much more than the other datasets that were tested over the Autoencoder. Shown directly below is the result of the Autoencoder without augmentation.

Figure X: Denoising Autoencoder result from test on the FER-2013 dataset prior to augmentation. From top to bottom, the rows represent noised image, mode output and sharpened output.

The result is noticeably blurred. Then, by augmenting the dataset prior to the train/test split. We are able to grow the size while keeping drive capacity high enough. After augmenting the dataset by replicating baseline images, the model's predictions improved substantially as expected. Shown below are the results of the model's prediction after the augmentation of the dataset.

Figure X: Denoising Autoencoder result from test on the FER-2013 dataset. From top to bottom, the rows represent noised image, mode output and sharpened output.

VinBigData

There is comparatively little variance in this dataset. All the images follow the same general structure. Therefore, the model produced better qualitative results than the unaugmented FER-2013 dataset. Shown directly below is the result of the Autoencoder, as well as the result of the output after an image sharpening kernel was applied.

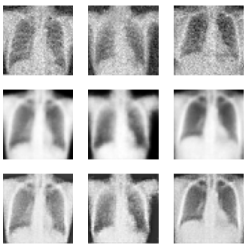


Figure X: Denoising Autoencoder result from test on the VinBigData dataset resized to 52x52 pixels. From top to bottom, the rows represent noised image, mode output and sharpened output.

Visually, the VinBigData set produced the best set of predictions out of all the experimental datasets.

V. CONCLUSION

In this paper, we introduced a technique to restore corrupted images using a denoising autoencoder constructed of convolutional layers. The proposed procedure is to add noise to both the train and test images which then are fed into the network. The model is a 2 layer structure associated with the encoder and the decoder. The encoder transforms the flattened input image into a vector of the desired dimensional space making the latent space. The decoder does the exact opposite of the encoder as it transforms the dimensional vector space back to the flattened vector making the reconstruction of the original image. Experimental results demonstrate that the proposed architecture that the model can consistently perform on different datasets of corrupted images. More importantly, the network achieved significant performance improvement through training. For future work, it could be interesting to explore different techniques to enhance the learning rate and the accuracy of the proposed model. Ideally, methods for introducing more complexity into the predictions, while keeping processing speed at similar levels would be further explored.

Our denoising autoencoder can be accessed here: https://colab.research.google.com/drive/15AWABw5t8EKmu7CgubTVJRxPA9uRsQsu?usp=sharing

Bibliography:

1. https://towardsdatascience.com/convolutional-autoencoders-for-image-noise-reduction-32fce9fc1763
   [1]*Convolutional Autoencoders for Image Noise Reduction*. 2019. [online] Available at: <https://towardsdatascience.com/convolutional-autoencoders-for-image-noise-reduction-32fce9fc1763 > [Accessed 15 April 2021].
2. https://arxiv.org/pdf/1812.09629.pdf

   [2] K. Uchida, M. Tanaka, and M. Okutomi, "Estimation and Restoration of Compositional Degradation Using Convolutional Neural Networks," *arXiv.org*, 23-Dec-2018. [Online]. Available: https://arxiv.org/pdf/1812.09629.pdf. [Accessed: 15-Apr-2021].

3. https://www.premiumbeat.com/blog/understanding-film-video-image-noise/
   [3]"Understanding Image Noise in Your Film and Video Projects", *The Beat: A Blog by PremiumBeat*, 2018. [Online]. Available: https://www.premiumbeat.com/blog/understanding-film-video-image-noise/. [Accessed: 15- Apr- 2021].
4. https://towardsdatascience.com/image-noise-reduction-in-10-minutes-with-convolutional-autoencoders-d16219d2956a
   [4]"Image Noise Reduction in 10 Minutes with Convolutional Autoencoders". 2020. [Online]. Available: https://towardsdatascience.com/image-noise-reduction-in-10-minutes-with-convolutional-autoencoders-d16219d2956a . [Accessed: 15- Apr- 2021].
5. [5]Class notes - Lecture 2
6. [6]Class notes - Lecture 3
7. https://www.machinecurve.com/index.php/2020/01/15/how-to-use-batch-normalization-with-keras/
   [7]"How to use Batch Normalization with Keras? – MachineCurve", *MachineCurve*, 2020. [Online]. Available: https://www.machinecurve.com/index.php/2020/01/15/how-to-use-batch-normalization-with-keras/. [Accessed: 15- Apr- 2021].
8. https://virenjain.org/pdf/supervised_learning_of_image_restoration_with_convolutional_networks.pdf

   [8] V. Jain, J. F. Murray, F. Roth, S. Turaga, V. Zhigulin, K. L. Briggman, M. N. Helmstaedter, W. Denk, and H. S. Seung, "Supervised Learning of Image Restoration with Convolutional Networks," *2007 IEEE 11th International Conference on Computer Vision*, 2007.

9. https://www.sciencedirect.com/topics/engineering/image-restoration-problem
   [9]"Image Restoration Problem - an overview | ScienceDirect Topics", *Sciencedirect.com*, 2005. [Online]. Available: https://www.sciencedirect.com/topics/engineering/image-restoration-problem. [Accessed: 15- Apr- 2021].
10. https://keras.rstudio.com/reference/fit.html

[10]”Train a Keras model — fit", *Keras.rstudio.com*. [Online]. Available: https://keras.rstudio.com/reference/fit.html. [Accessed: 15- Apr- 2021].

11. https://towardsdatascience.com/estimators-loss-functions-optimizers-core-of-ml-algorithms-d603f6b0161a
    [11]"Estimators, Loss Functions, Optimizers —Core of ML Algorithms", *towards data science*, 2019. [Online]. Available: https://towardsdatascience.com/estimators-loss-functions-optimizers-core-of-ml-algorithms-d603f6b0161a . [Accessed: 16- Apr- 2021].

12. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
    [12]”Gentle Introduction to the Adam Optimization Algorithm for Deep Learning", *Machine Learning Mastery*, 2021. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accessed: 15- Apr- 2021].
13. https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/
    [13]"Fashion-MNIST", *Research.zalando.com*. [Online]. Available: https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/. [Accessed: 15- Apr- 2021].
14. https://towardsdatascience.com/batch-normalisation-explained-5f4bd9de5feb
    [14]”Batch Normalisation Explained". 2020. [Online]. Available: https://towardsdatascience.com/batch-normalisation-explained-5f4bd9de5feb. [Accessed: 15- Apr- 2021].
15. https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8
    [15]”The best explanation of Convolutional Neural Networks on the Internet!". 2016. [Online]. Available: https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8. [Accessed: 15- Apr- 2021].
16. https://debuggercafe.com/autoencoder-neural-network-application-to-image-denoising/
    [16]”Autoencoder Neural Network: Application to Image Denoising", *DebuggerCafe*, 2020. [Online]. Available: https://debuggercafe.com/autoencoder-neural-network-application-to-image-denoising/. [Accessed: 15- Apr- 2021].
17. https://towardsdatascience.com/batch-normalization-8a2e585775c9
    [17]"Batch Normalization", *TowardsDataScience*, 2017. [Online]. Available: https://towardsdatascience.com/batch-normalization-8a2e585775c9. [Accessed: 15- Apr- 2021].