IMAGE DE-RAINING: AN AUTOENCODER BASED APPROACH

Members:
Siddhant Jain, Akash Hasamnis, Hardik Ruparel, Sri Charan Vempati

ABSTRACT

Computer Vision has found ubiquitous applications in the present age. From image robustness to weather conditions, each of these is a pre-requisite for most of the AI applications. Hence, the loss in image quality due to rain and other weather conditions is a significant problem. Image deraining is a computer vision task that helps restore a clean scene from a degraded image captured on a rainy day. With the advancement in autonomous vehicle technology, the ability to derain the images is a highly desirable and fundamental component for improving the efficiency and safety of autonomous vehicles. Although the task of deraining images has been going on for decades, the newer approaches in deep learning have made it possible to achieve results that were previously considered only theoretical. In this project, we leveraged the inherent restorative capabilities of autoencoders to remove raindrops and other such impurities and improve the image quality.

1 INTRODUCTION

Image sensors and cameras find a wide range of applications in a multitude of AI applications like self-driving cars, navigation, face detection, cybersecurity etc. However, all these are susceptible to weather conditions like fog and raindrops. Raindrops and condensation have a different refractive index compared to the lens of the camera. Thus, if a raindrop falls on the camera lens, light doesn't fall directly on the lens, but gets refracted and partially reflected due to the raindrop. This results in the distortion or blurring of the images. In the above mentioned applications, receiving such an image can severely affect the performance of the agent/model which receives that image percept.

Image denoising is the process of estimation of the original image by suppressing the noise in the contaminated image. The different techniques used for image denoising include CNN's, Spatial domain filtering, data adaptive transforms etc. While de-raining of images is similar to de-noising of images and enhancement of images having bad weather visibility, there are some key differences which makes de-raining require specifically designed algorithms. While de-noising can be seen as image transformation, de-raining is more of raindrop detection.

We attempt to provide a solution to this problem by using the autoencoder decoder artificial neural network. We will provide our network with noisy images and a ground truth and the network will try to reconstruct its output to be as close as possible to the ground truth(7). The encoder and decoder networks are usually trained as a whole. The loss function penalizes the network for creating an output that differs from the ground truth. By doing so, the encoder learns to preserve as much of the relevant information needed in the limitation of the latent space, and cleverly discard irrelevant parts i.e the rainy part. The decoder learns to take the compressed latent information and reconstruct it into a full error-free input. We are using autoencoders because it can learn non-linear transformations with a non-linear activation function and multiple layers.(8) It is more efficient to learn with several layers in an autoencoder rather than learn one huge transformation with PCA. An autoencoder provides a representation of each layer as the output. It can make use of pre-trained layers from another model to apply transfer learning to enhance the encoder/decoder.

2 Basic Concepts

2.1 Autoencoder

Autoencoders are Artificial Neural Networks(ANN) in which we use the power of neural networks for the task of representation learning(3). The autoencoder learns a representation of the data, typically by reducing the dimensions of the data, thereby ignoring the insignificant data or the "noise" in the data. In autoencoders, we enforce a bottleneck in the network which results in a compressed representation of the data. After the compression step, we use a decoder network to reconstruct the data from the compressed representation, thereby ignoring the noise in the data. Hence, an encoder-decoder has 4 parts:(1)

- Encoder: In which the model learns how to reduce the input dimensions and compress the input data into an encoded representation.
- Bottleneck: which is the layer that contains the compressed representation of the input data. This is the lowest dimensions of the input data in that network.
- Decoder: In which the model learns how to reconstruct the data from the encoded representation to be as close to the original input as possible.
- Reconstruction Loss: This is the method that measures how well the model is performing and how close the output is to the original input.

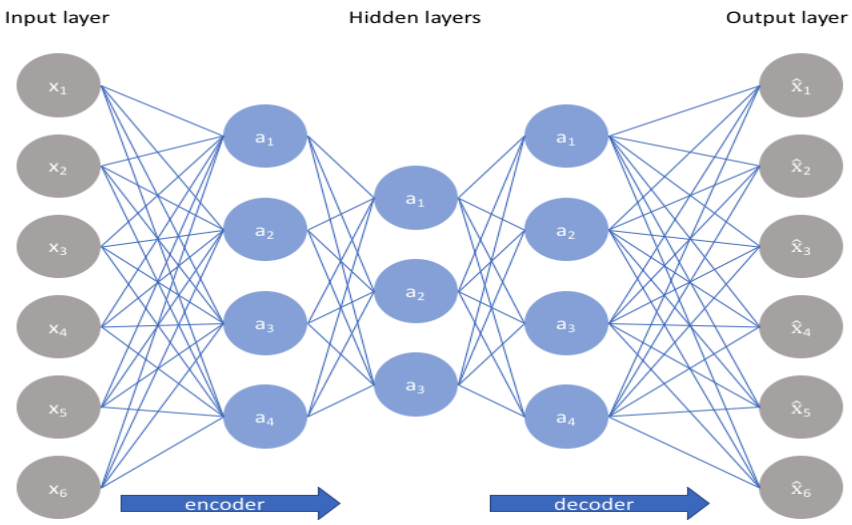


Figure 1: Structure of an Autoencoder network

2.2 Normalization

Normalization is the process of transforming the data in such a way that the dimensions of the data are uniform and are suitable for the model being used. Normalization is also known as standardization or feature scaling. It is an essential step in data pre-processing in any machine learning application and model fitting. There are many normalization techniques. Some of the popular ones are: (4)

- Rescaling: also known as "min-max normalization", it is the simplest of all methods and calculated as:

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

- Mean normalization: This method uses the mean of the observations in the transformation process:

$$x' = \frac{x - average(x)}{max(x) - min(x)}$$

- Z-score normalization: Also known as standardization, this technique uses Z-score or "standard score". It is widely used in machine learning algorithms such as SVM and logistic regression:

$$z = \frac{x - \mu}{\sigma}$$

Image normalization is a process, often used in the preparation of data sets, in which multiple images are put into a common statistical distribution in terms of size and pixel values.

The RGB values of the pixels are in the range 0 to 255 and matrix multiplication will be performed by the model on these pixels by the chosen model. Hence, there is a possibility of integer overflow when these matrix multiplications take place. Thus, we need to rescale the values from the range 0 to 255 to 0 to 1. We use rescaling (min-max) normalization for the same.

2.3 Peak Signal to Noise Rato(PSNR)

PSNR is a ratio used for quality measurement between two images. It is calculated in decibels. (6)It is an expression for the ratio between the actual value of a pixel in an image and the distortion present in the reconstructed image.The higher the PSNR, the better the quality of the reconstructed image. Before we find the PSNR of the reconstructed image, we need to find the Mean Squared Error(MSE).

The Mean Squared Error(MSE) or the Mean Squared Deviation(MSD) averages the squares of the errors i.e the difference between the expected value and predicted value.

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)}_{\text{The square of the difference between actual and predicted}}^{2}$$

Thus, we can calculate PSNR as:

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}\left(MAX_I\right) - 10 \cdot \log_{10}\left(MSE\right) \end{aligned}$$

PSNR is most commonly used to measure the quality of reconstruction of lossy compression (e.g., for image compression). The signal in this case is the original data, and the noise is the error introduced by compression. When comparing images, PSNR is an approximation to human perception of reconstruction quality. In the absence of noise, the two images I and K are identical, and thus the MSE is zero. In this case the PSNR is infinite

3 Dataset

In the initial dataset, we had 3 files:Training data, Validation data, Testing data. We combined the training data and the validation data to divide the entire dataset into two parts, the training set and

the testing set. We divided the training set in a 80:20 ratio into the training set and the validation set while training the model. The definitions for the same are as follows:

-Training set: A set of examples used for learning, that is to fit the parameters of the classifier.

-Validation set: A set of examples used to tune the hyper-parameters of a classifier.

-Test set: A set of examples used only to assess the performance of a fully-trained classifier.

The total size of the training and the validation set was 1110 images. The total size of the test set was 58 images. Since we are using supervised learning, we needed to show the model how an ideal derained/denoised image would look like. Hence the training and the validation set consist of rainy/noisy images and a ground truth which contains a derained/denoised image. The presence of the validation set helps us to keep an eye on the accuracy and loss of the model when the model is training and helps us tune the model parameters better.

4 Proposed Model

We are using a combination of autoencoders and decoder network for a supervised learning algorithm. As a first step towards de-noising images, we make use of autoencoders. An autoencoder is a neural network architecture capable of discovering structure within data in order to develop a compressed representation of the input. This compressed representation is a generalized idea of how the data looks like in lower dimensions. This compressed representation will then help us to retrieve the original input. We try to improve the performance of autoencoders by minimizing the reconstruction loss we get on reconstructing output images compared to the original/truth image. As autoencoders are meant to learn the generalised model of the input data, one of its applications is in de-noising images where it helps us remove the impurities in the images data by learning the general idea of the image and not the noisy part of it.

The autoencoder network compresses the image and develops a representation of that image. The decoder network helps us to convert this compressed representation into a derained/denoised image.

5 Methodology

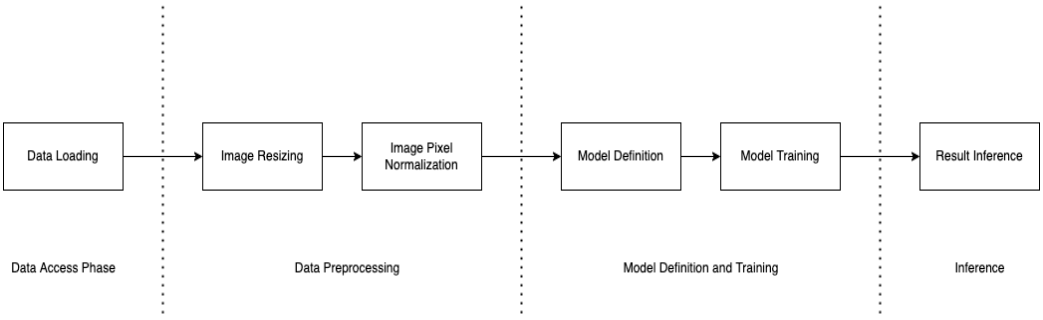


Figure 2: Machine Learning Pipeline

As shown in the above figure, we have a four-step pipeline. First, we will load the images from the data folder. After storing the images in an array, we will start the data processing. In the data preprocessing step, we will resize the image and then normalize the images. After the image preprocessing step, we will define our Deep Learning model. Once the model is defined, we will then train the ML model on our training dataset, and then we train the model by fitting the model on our data for a finite number of epochs. After training our model we evaluate the performance of our model on the test data. The loss function of our model is the Root-mean-squared-error (RMSE) which is used to calculate the Peak signal-to-noise ratio (PSNR). The higher the PSNR value is, the better is the performance of the model. For this project, we trained three variations of autoencoders and compared the output PSNR values of these models on the test data of 58 images. The three variations of autoencoders that we have implemented in our project differ in the input image sizes,

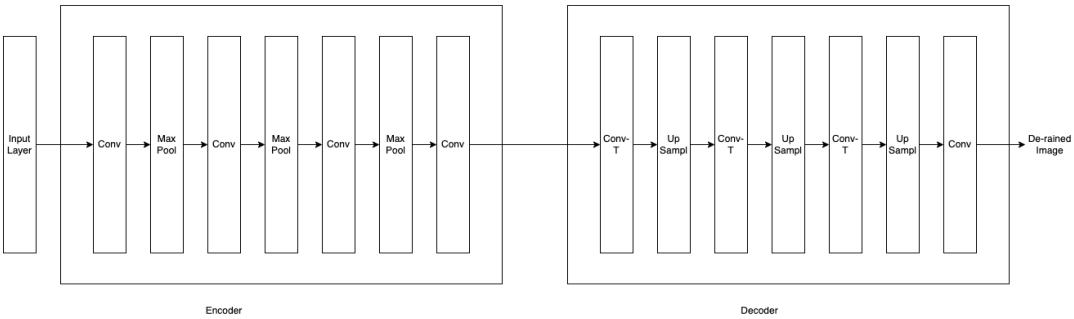


Figure 3: Model Architecture

batch-sizes, layers in the network and the number of epochs for which the model will be trained. However, the base-line architecture of our model is same which is shown in 3.

5.1 Encoder

In the encoding step, there are blocks of Convolution2D (mentioned as Conv in the above image) layer and max-pooling (mentioned as Max Pool in the above image) layers and a final convolution layer (mentioned as Conv in the above image) which will contain only the important extracted features in the images. In each convolution layer, first, padding will be added to the input image so that the input image is fully covered by the kernel. Furthermore, each convolution layer has a ReLU activation function. After each convolution layer, the max-pooling layer will select the maximum pixel(signal) value from the specified kernel size thereby, reducing the image size after each max pool layer by a factor of the size of the kernel. The reason for choosing the max-pool layer as the sampling layer in our model is to help us extract the sharpest feature of the image that will help us identify low-level features like rain-drop edges, points and reduces variance and computations later in the network.

5.2 Decoder

In the decoder, we have blocks of convolution transpose (mentioned as Conv-T in the above image) and upsampling layers (mentioned as Up Sampl in the above image), and finally, a convolution layer(mentioned as Conv in the above image) after the model will output de-rained images. In each of the convolution transpose layers, padding of zeros will be added to the input data to help the kernel traverse the image. Each of these convolution transpose layers has ReLU as the activation function. To restore the size of the image, we are using the upsampling layers to increase the size of the encoded image. Finally, the last convolution layer of the decoder and the model, in general, pads the input data and uses sigmoid as the activation function. In the training phase, this output image will be used to calculate the loss in the backpropagation phase. In the testing phase, the generated image is used to compare with the ground-truth images to calculate psnr value to evaluate the overall performance of the model.

6 Libraries and Environment

- Google Colab
- Google Cloud Platform
- Keras
- Tensorflow
- Numpy
- Matplotlib
- glob
- opencv

7 RESULTS

We made 3 architectures of encoder-decoder networks with different hyper-parameters like batch-size, layers, number of epochs and size of pre-processed image. In the first architecture (fig. 4), we have 2 layers (each layer consisting of a Conv layer and a max-pooling layer) to get a encoded representation of the image. We set the input size image in the model as 256x256 after pre-processing. In the decoder network, we have 2 layers (each consisting of conv2d transpose and up-sampling layers). Using standard Adam optimizer and MSE loss, we train this model for 150 epochs and obtain the results as below. The output of this model can be seen as in fig.5,fig.6 and fig.**??**

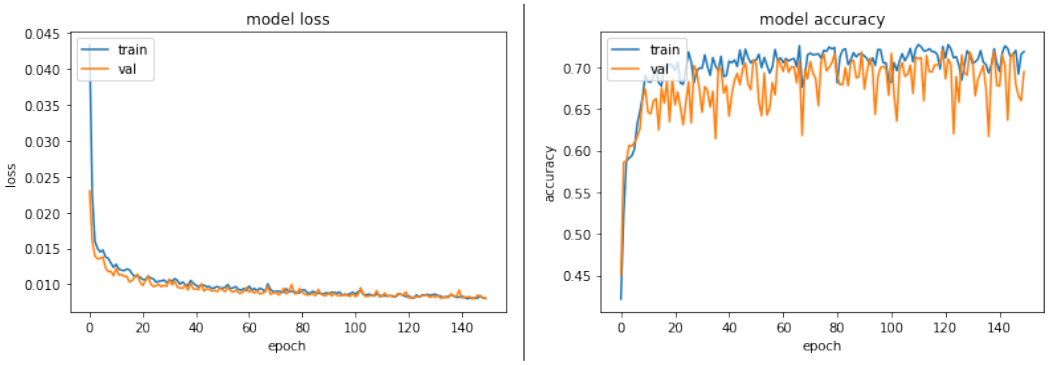
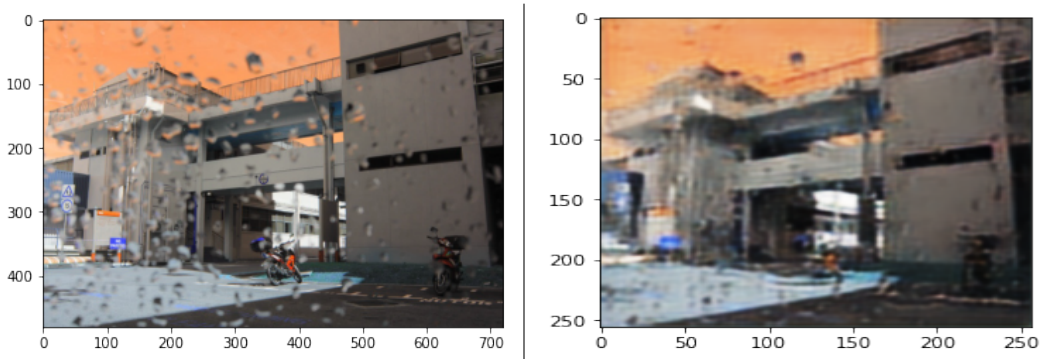
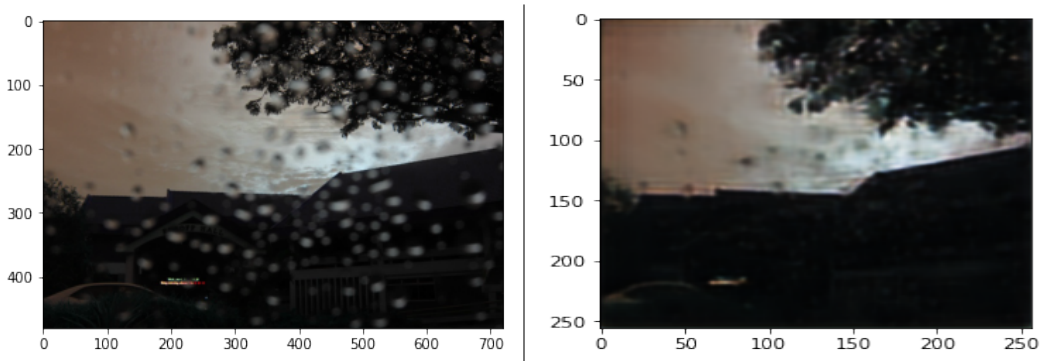


Figure 4: Accuracy and Loss visualization of Model-1



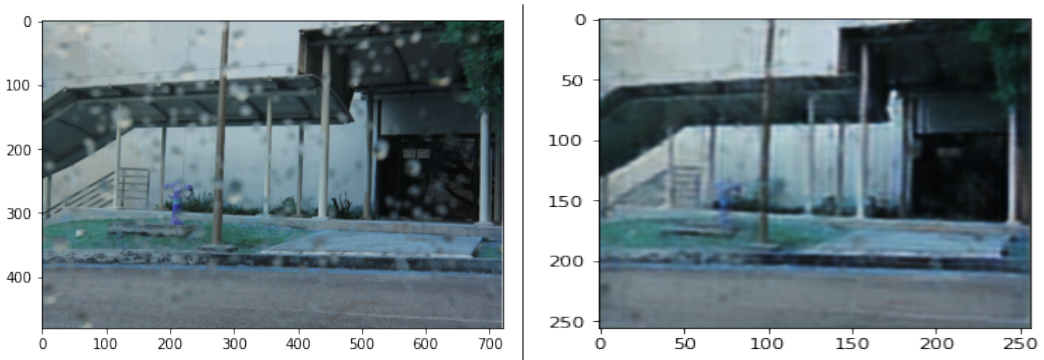
(a) Before (b) After

Figure 5: Deraining of Test Image #1 by Model-1



(a) Before (b) After

Figure 6: Deraining of Test Image #2 by Model-1



(a) Before (b) After

Figure 7: Deraining of Test Image #3 by Model-1

In the second architecture(fig. 8), we have 3 layers (each layer consisting of a Conv layer and a max-pooling layer) to get a encoded representation of the image. We set the input size image in the model as 256x256 after pre-processing. In the decoder network, we have 3 layers (each consisting of conv2d transpose and up-sampling layers). Using standard Adam optimizer and MSE loss, we train this model for 100 epochs and obtain the results as below. The output of this model can be seen as in fig.9, fig.10 and fig.11.

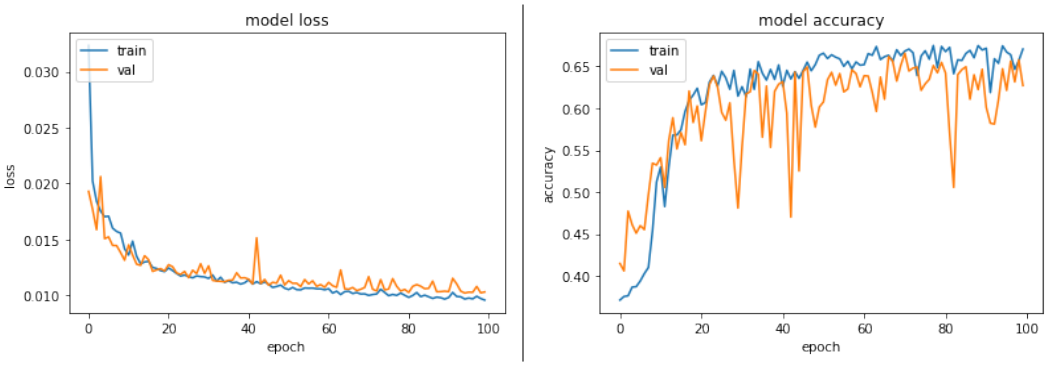
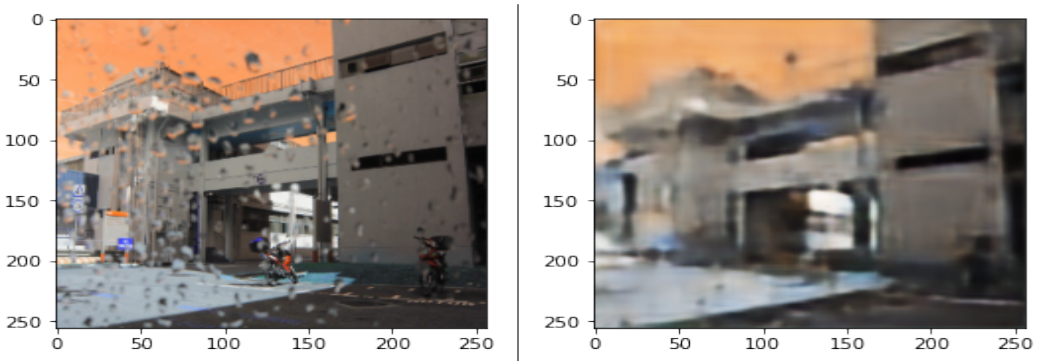
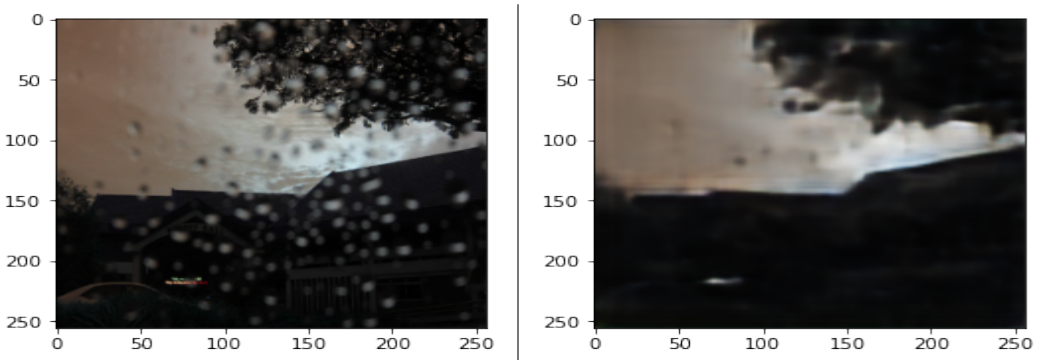


Figure 8: Accuracy and Loss visualization of Model-2



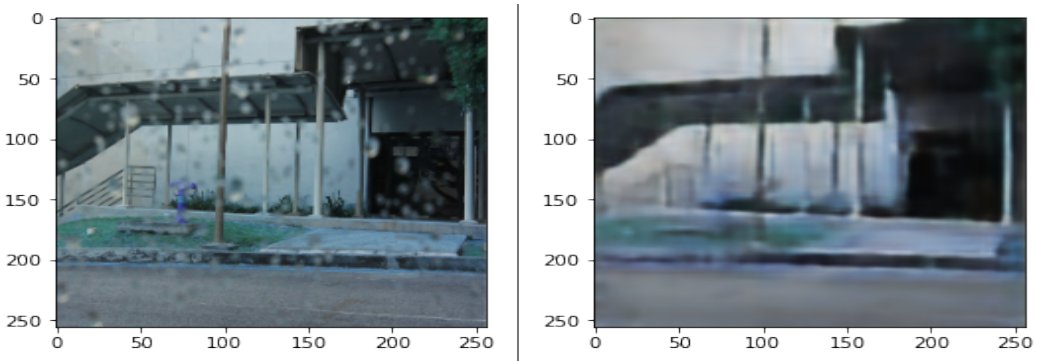
(a) Before (b) After

Figure 9: Deraining of Test Image #1 by Model-2



(a) Before (b) After

Figure 10: Deraining of Test Image #2 by Model-2



(a) Before (b) After

Figure 11: Deraining of Test Image #3 by Model-2

In the third architecture(fig. 12), we have 3 layers (each layer consisting of a Conv layer and a max-pooling layer) to get a encoded representation of the image. We set the input size image in the model as 512x512 after pre-processing. In the decoder network, we have 3 layers (each consisting of conv2d transpose and up-sampling layers). Using standard Adam optimizer and MSE loss, we train this model for 100 epochs and obtain the results as below. The output of this model can be seen as in fig.13, fig.14 and fig.15.

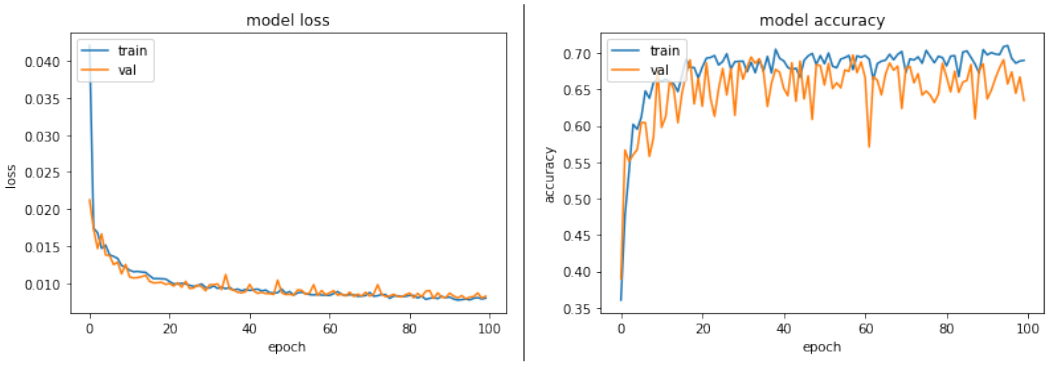
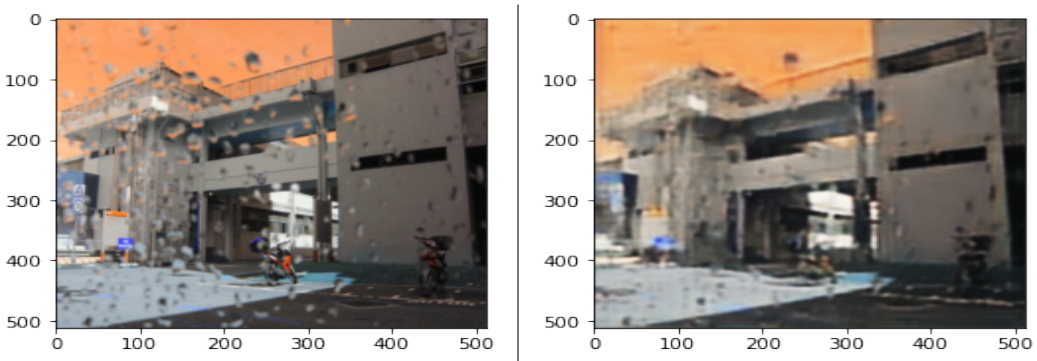
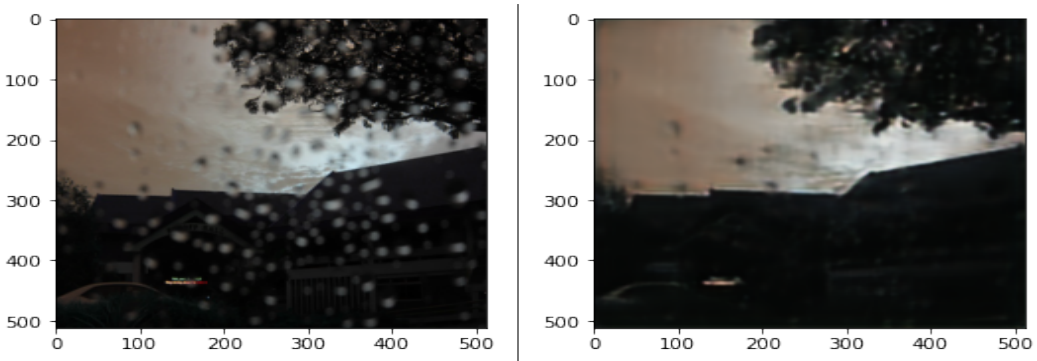


Figure 12: Accuracy and Loss visualization of Model-3



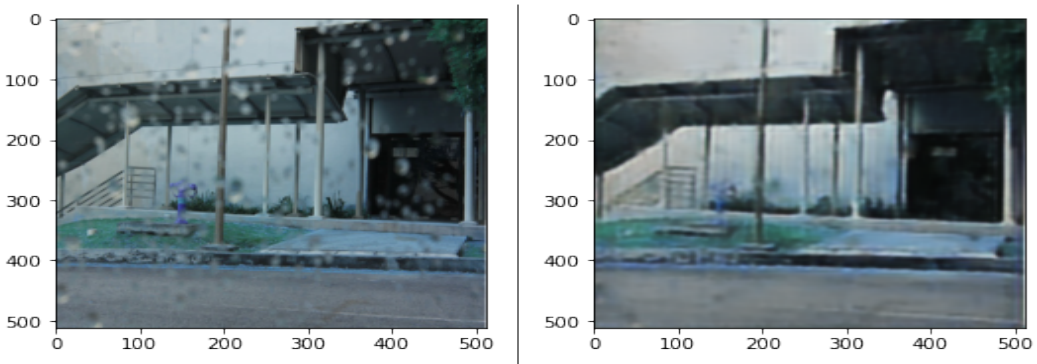
(a) Before (b) After

Figure 13: Deraining of Test Image #1 by Model-3



(a) Before (b) After

Figure 14: Deraining of Test Image #2 by Model-3



(a) Before (b) After

Figure 15: Deraining of Test Image #3 by Model-3

Models	Number of Layers	Resized Input size	Accuracy	MSE	PSNR
Model 1	2	256x256	**72.1**%	**0.0083**	23.5
Model 2	3	256x256	66.57%	0.01	22.3
Model 3	3	512x512	69.73%	0.0085	**23.66**

Table 1: Comparative Analysis of the Performance of Model 1, 2 and 3

Table 1 shows a comparative analysis of the performance of our models. As seen in the table, we can see that the Model 3 performed the best in de-raining the images.

8 Ablation Studies

From the three encoder-decoder models that we built, we determine the significance of an extra layer and input size of the image in the model. In the second model, we used three layers each for encoder-decoder networks with 256x256 size input images. Here, the features extracted in the network were less due to the smaller input image size as compared to the 3rd model. Also, due to a deeper network compared to our 1st model, our model might have slightly over-fitted the dataset which would decrease the accuracy and PSNR overall. From these observations, we can say that having an extra layer (deeper model) will make sense if we have larger-sized images or a larger dataset that would help the model extract and learn more specific features from the data. Also, we notice that a higher number of epochs didn't play a major role as the model training converged pretty early during the training cycle. This might be because of the limited size of the dataset that we used owing to which the model converged quickly. A higher number of epochs would have been helpful if we had tens of thousands of images and a deeper network which would take time to train.

9 Conclusion and Future Work

We have proposed a solution for the deraining/denoising of images using autoencoders which creates a compressed representation of the image eliminating the noise and then recreate the image without the noise. Our model creates a random validation set while training the data, in order to better tune the hyper-parameters of the model. We achieved the given result after training the model for 100-150 epochs.

A very limited dataset is available for the given problem. In the future, given the availability of better resources like better GPU's and more time, we would like to train the model on a larger dataset using larger networks for possibly better results. We would also explore the possibility of using Generative Adverserial Networks(GANs) for the same problem and compare the results with our proposed model.

References

[1] https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726

[2] https://en.wikipedia.org/wiki/Autoencoder

[3] https://www.jeremyjordan.me/autoencoders/

[4] https://towardsdatascience.com/data-normalization-in-machine-learning-395fdec69d02

[5] https://radiopaedia.org/articles/image-normalization

[6] https://en.wikipedia.org/wiki/Peaksignal-to-noiseratio

[7] https://towardsdatascience.com/autoencoder-for-denoising-images-7d63a0831bfd

[8] https://www.edureka.co/blog/autoencoders-tutorial/