A Color Vision Approach Considering Weather Conditions Based on Autoencoder Techniques Using Deep Neural Networks

Mohammad Mainuddin Raj, Samaul Haque Tasdid, Maliha Ahmed Nidra, Jobaer Noor, Sanjana Amin Ria and Md. Ashraful Alam

Department of Computer Science and Engineering Brac University, Dhaka, Bangladesh

2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE) | 978-1-6654-9552-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/CSDE53843.2021.9718453

***Abstract*—Color machine vision is a riveting technology crucial in pioneering innovations like autonomous vehicles, autonomous drone deliveries, automated stores, robots, infrastructure and surveillance monitoring programs for security, manufacturing defect monitoring and more. When it comes to real life applications of automated machines, safety is a major concern and to ensure utmost safety the unpredictable has to be taken into consideration. We propose and demonstrate a color vision approach that allows image normalization hinged on autoencoder techniques employing deep neural networks. The model is composed of image preprocessing, encoding and decoding. The images are resized in preprocessing portion the images go through a cognitive operation where the input image becomes suitable to enter the autoencoding technique section. The autoencoder is comprised of two core components – encoder and decoder. To employ this system deep neural network is applied which generates a code of an image in the encoding process. Sequentially, the code changes over to decoding. Decoder portion decodes it and regenerates the initial image extracting it from the code of the encoder portion. It allows normalizing color images under different weather conditions such as images captured during rainy or foggy weather conditions. We devise it such that rainy and foggy images are normalized concurrently and in real time. The autoencoder is trained with numerous rainy and foggy datasets utilizing CNN. In this research we investigate the model normalizing images in two different weather conditions – rainy and foggy conditions in real time. We used SSIM and PSNR to verify the accuracy of the model and confirm its capability reconstructing images in real time for advanced real life color vision implementations.**

***Index Terms*—Color Vision; Deep Neural Network; CNN; Autoencoder.**

I. Introduction

With the advancement of technology, the autonomous approach is rapidly being adopted by the world to solve different practical problems. These complex modern problems challenge the ability of the modern world to function autonomously and demand the world to adopt the autonomous approach of solving various problems.

Computer vision pilots computers to analyze perceptible environments using images and videos from cameras. But this has to be done in a measured and accurate way so that there remains no chance for any catastrophic failure. Usage of deep learning models enable machines to recognize and classify objects correctly and respond accordingly. Deep learning algorithms and to be specific, convolutional networks, have rapidly become a popular procedure for analysis of medical images [1]. Our model aims eliminating any possible imprecise analysis. In order to do so, a lot of factors like nature, weather, light is taken into consideration while analyzing images. We mainly focus on different aspects of weather conditions while analyzing, monitor their effects on the images and remove any occurrence of anomaly and noise because of those using autoencoder techniques.

In this context we formulate this color vision approach that allows normalizing color images under different weather conditions. Here, we applied autoencoder technique for image normalization in adverse weather conditions like rainy and foggy weather using deep neural networks which generates and analyzes accurate real-time image information that can be used in various practical and advanced color vision/ color machine vision applications.

We have developed a model that will consider different environmental elements like rain and fog to enable composite analyzation and normalization of images. In order to do so, we employed autoencoder. Autoencoders are basically learning circuits. The core function of autoencoder is to convert inputs into outputs with as little distortion as possible [2]. We also exerted convolutional neural networks (CNN) for analyzing visual images. The structure of CNN allows it to master spatial hierarchies of attributes through backpropagation utilizing multiple building blocks deliberately, some of which are convolution layers, pooling layers, and fully connected layers, etc. [3]. For training the model we utilized open-source libraries like Keras and TensorFlow with the convenience of swift results. We used SSIM to predict distortions and evaluate the accuracy of the model. The formidable representation power of CNN was utilized to project structural similarity maps (SSIM) which demonstrates distortion information promptly and it is done precisely: end-to-end and pixel-to-pixel [4].

A. Research Objectives

An image captured in different weather conditions like rainy or foggy weather is disrupted by having raindrops or haziness, resulting in the implementation of that particular image to be inconsistent and inaccurate. Thus, by using our model, that particular image is converted to an image as if it

was captured in normal daylight condition by normalization, getting rid of the impact of unwanted weather conditions. Our main objective was to identify objects through image processing in different weather conditions and in that track, our brief procedures were-

(i) Preparing a dataset of images in various weather conditions.
(ii) Training the model by utilizing the data.
(iii) Reconstructing images through deep neural networks.
(iv) Detecting abnormality and noise in images by training the model with numerous images to make it persistent in identifying and differentiating objects.
(v) Monitoring the performance of the model.
(vi) And thus, render clear images for detecting objects accurately despite the weather condition.

II. Literature Review

In today's world, image processing is becoming a vital part of computer science. Computer vision is becoming the most important factor in this digital universe. In order to interconnect material world with technological, it plays a significant role and is one of the most prominent method. We are working on a paper where we can auto-encode our image to remove specific weather factor and create an abstract free image. Our approach is based on autoencoder technique using Deep Neural Network and taking account of weather conditions. There are many research papers that have been published on image processing. Also, there are many papers regarding the noise removal issues. But in our paper, we try to solve specific weather factor issues like rain and fog. We found many research papers related to the topic. And we try to understand their methodology and find their limitations. We found a paper titled "UNSCARF-a color vision system for the detection of unstructured roads" [5], where the authors try to build a color vision system to detect road for an intelligent mobile robot. Here they used a UNSCARF algorithm. This algorithm collects image and checks for similar pixel in the color image from its training dataset. First it reduces the color image size and then removes the noise and converts the input color image in required size, then it matches it with its own data. This system is very slow and costly. Besides it cannot always generate the actual proper images for the robot. In our paper we will be using a system where we used auto encoder techniques which is far more efficient and effective. In our studies we found another paper titled "LLNet: A deep autoencoder approach to natural low-light image enhancement" [6]. Here the authors used deep autoencoder techniques for LLNet or low light net, where they tried denoising it using BM3D, K-SVD and non-linear filters. Then they used various deep learning techniques for LLNet. LLNet used 3 DA layers. And these 3 layers becomes bottleneck layer. They trained their dataset with patches extracted from their standard test images. They used two performance metric PSNR and SSIM to evaluate the proposed framework's performance. We used similar method like autoencoder and bottleneck in our system. Beside that we are also going to use our test images to train our dataset. Their [6] paper purely focused on low light image enhancement and our system is focused on removing weather condition from image, so our output and other methodology is different from their [6] system. We came across another paper titled "A joint deep neural networks-based method for single nighttime rainy image enhancement". In this paper [13] the authors try to build a new system which will derain image in low-illumination situations. They used a symmetric contextual autoencoder neural network to enhance a single night time image.

They also [7] used a retinex theory to decompose image. On our paper we used deep autoencoder to reconstruct rain and fog free images. In our paper we mainly focused on overall weather factors in any image. We try to build a hybrid model to point out the rain and fog and then try to remove them from our test image. As in theirs [7], they are specifically removing only rain from single low-illuminating image; hence our way of work and methodology is different from theirs. We found another paper named "Clearing the Skies: A deep network architecture for single-image rain removal" [8]. In this paper [8] authors implement a new deep convolutional neural network system called DrainNet to remove rain streaks from an image. With the help of the DrainNet system they [8] also try to improve the visual quality of the image. They train their model with high pass detail layer. They [8] also have given different parameter settings for Kernel size, Network width and Network depth. As we are trying to work with multiple weather factors, so we cannot directly use DrainNet. But instead, we will be using convolutional neural network and auto encoder. We also try to enhance our output image quality using auto encoder. Beside that we will also be using different parameters for the training of our model. We came across another paper related to our paper, this paper is named "Image recognition with deep neural networks in presence of noise – Dealing with and taking advantage of distortions" [9]. This paper is mainly focused on experimental examination of the influence of different types of noise on the convolutional neural network. This paper also come up with a denoiser using deep neural network. They evaluated two possible strategies of dealing with noise; one is training data augmentation and another is denoising prior to classification. They also did classification performance under various noise settings and different noise model. We studied this paper [9] thoroughly for understanding denoising and deep neural network. Since in our paper we will be using a deep neural network model. Another paper we found named "Medical image denoising using convolutional denoising autoencoders" [10]. In this paper, authors used a convolutional denoising autoencoder to remove noise from various Medical imaging including X-rays, Magnetic Resonance Imaging (MRI), Computer Tomography (CT), ultrasound etc. They [10] used two datasets, mini-MIAS database of mammograms (MMM) and a dental radiography database (DX). They sized all the images from the dataset and trained their model. We studied this paper [10] to find out their methodology which helped us to understand how autoencoder works in different

situations, as their [10] work mainly focuses on medical imaging and we work on normal images. The last paper we are going to discuss is named "Noise Removal from Color Images" [11]. Here, Zheng and Gauch[11] talked about Color image analysis, chromaticity-preserving median filter, color perception, monochromatic like processing, vector signal processing and mean type filtering. Although this paper was written long ago and the technology developed a long way after their paper [11] was published, but here they tried to describe the very basic buildup of image processing, filtering and vector modeling to remove noise. Even though in our paper we will be using more modern ways but from this paper [11] we tried to understand the basic methodology.

III. Methodology

As mentioned earlier, the primary goal of this paper is to develop a system that normalizes color images under different weather conditions such as rain and fog. For this, we proposed autoencoder technique for normalizing the images under weather conditions like rainy and foggy weather using deep neural networks.

A. Basic Architecture of Proposed Model

The basic architecture of the proposed model is illustrated below: -

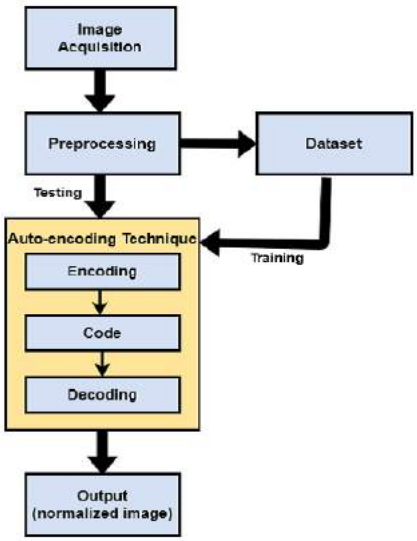

Fig. 1. Basic Architecture of Proposed Model

In Fig 1, the model we proposed has been divided into different steps for successfully generating a clear normalized image by performing image acquisition followed by image pre-processing and finally the autoencoding technique is implied in order to get our result. On the very first step, in image acquisition, real time data is to be captured and collected using a camera and passed to the second step, Image pre-processing. This is where the initial raw data is transformed into a required format which we have used as our dataset. Using this collected dataset, we move onto our third step, the auto-encoding technique, where the autoencoder is trained. After all of these steps are performed successfully and the auto-encoder has been trained effectively, the model is then able to take the pre-processed images and directly normalize them through the trained autoencoder.

B. Data Analysis

Data analysis is defined as a process of cleaning, transforming, and modelling data to discover useful information, eliminating unnecessary information and features from the data to retrieve the most suitable and readable form we can accept as input.

During our research we realized collecting required steady images of rain and fog in a huge quantity using a camera is very challenging as the required weather conditions have not come by as often during our research period and we couldn't gather much open-source datasets of this mass containing naturally occurring rainy or foggy images. Hence, creating our unique dataset is what we believe will best define the problem domain and work accordingly.

As our autoencoder's purpose is to take an image that contains rain/fog and delivering a clean image by removing the traces of rain or fog from the images, hence we will be needing images containing rain and fog for training and testing our autoencoder. Therefore, for our data analysis, firstly we will prepare our data collection to best serve the purpose.

1) Data Acquisition: We had taken some help from other sources and we also tried to gather adequate amount of data by ourselves. To find these datasets from other sources, we kept two things in mind, the first thing is the dataset quality and the second thing is the dataset's usability. In dataset quality we look for best possible dataset where images are in good condition and images have clear conditions. We have come across various datasets where images were pixelated, some had dataset images that were blurry, and furthermore, some datasets had images that has random regulations and quality which is not very suitable for our thesis. The second thing we looked for was dataset's usability. Some datasets that we found had usability as low as 2, some had usability of 2.5. For that reason, we tried to find dataset which has high usability rate. After our search, we actually managed to find two sets of datasets of rainy and foggy images that satisfies both of our conditions, i.e., the datasets have clear and good condition image with high usability rate and hence, we finally were able to use those in our thesis. Few of the rain dataset that were added were taken from the dataset of Wang, Tianyu and Yang, Xin and Xu, Ke and Chen, Shaozhe and Zhang, Qiang and Lau, Rynson W.H [12] and for haze dataset, we took help from the dataset of Ashwath,B [13]. These datasets have a high usability rate, higher than 9 and used synthesized data to add rain and fog. So, we took to

these datasets to increase our training dataset by merging with ours that we personally gathered.

2) Image Pre-processing: Image processing is the use of computer algorithms to perform image processing on digital images. Digital image pre-processing can be done using a wide range of algorithms that are applied to the input data to prepare the image for further machine usage- the aim of digital image processing is to improve the image data (features) by suppressing unwanted distortions and/or enhancement of some of the more important image features that will be needed in order for our Autoencoder model to be benefitted by this improved data, so that it can further work on it.
An image is nothing more than a two-dimensional array of numbers (or pixels) ranging between 0 and 255. It is defined by the mathematical function $f(x, y)$ Where x and y are the two co-ordinates horizontally and vertically.
The value of $f(x, y)$ at any point is giving the pixel value at that point of an image.

The color images are normally represented using 3 channels of RGB: Red, Green and Blue. When we use RGB image as input to CNN, the depth of filter (or kernel), it will always have to be same as the depth of image. For RGB this depth is 3. Hence, if the input image $M * M * 3$, the filter will have to be $N * N * 3$ (N will be the height and width of filter here). Hence, we have a filter with 3 two dimensional matrices where separately in every single plane each of the two-dimensional matrices has been combined with a small portion of the provided input image and these results from each of the plane are added to generate a single output value in the feature map.

For our operations, we started by loading a single image, converting the image into NumPy arrays and manipulating it according to our requirements. We can manipulate, if required, multiple features of an image, including size, pixels, height, width, etc. of an image.

We have been loading a single image and we've looked into the data formats specific to that image and extracted mode and size of the image. The mode of an image defines the type and depth of a pixel in the image. The size of an image is (height $*$ width).

After that is done, we then resized all the images to 256*256 pixel size, as the images that are captured by the camera and fed to our encoder varied in sizes, therefore, we needed to establish a fixed base size for all the images that we have been feeding to our encoder.
In the next step, in order to visualize the change, we created two functions that displayed the images, the first being the one to display one image and the second for two images. After that, we then created a function called processing that will just receive the images as a parameter and work on it.

For the system to be more accurate we also considered the fact that rain droplets usually don't have the same angle and that the angle varies considering both terminal vertical speed of raindrops (VT) and the average horizontal wind speed near the ground (VS). Rain streaks are generally at an angle (θ) compared to the vertical between, $0(ver) < \theta(rad) < \pi/2$. Hence, under the condition of assuming the angle of the rainfall to be constant within each time interval, we can calculate the angle using the formula:

$$\theta = \textbf{\textit{(arctan)}}\frac{VS}{VT} \tag{1}$$

Size and density of rain streaks also normally vary with rain's intensity in an image. Therefore, we can say that with the change of these factors, the image of the rain will also change in very different ways. Hence, one of the major element that we must consider about rain is its intensity as well. With higher intensity, rain streaks will have bigger size and higher density, whereas a drizzle will have smaller streak size and lower density. Density can be generated by summing the pixel intensity (Ri) by layering rain streaks along the line of sight to the background where 'n' is the number of rain streaks and 'i' is the number of layers. By assuming the background (B) is constant, we can model the image (O) using the formula:

$$O = B + \sum_{i}^{n} \textbf{\textit{(Ri)}} \tag{2}$$

We have reduced image's brightness as well because rainy days are usually shady because of the rain. Here, we can also change the dimension of our blur filter to showcase the effects of different rain conditions further.

In Fig. 3, the dataset of rain and the clear image without rain is shown. Here, as you can see, we have given 14 different images from our dataset, where rain streaks were added to the clear image shown below. Like we said, we have processed the image by adding rain streaks to the same image several times by changing angle of rain droplets. Hence, in these 14 images in Fig. 3, it can be seen that rain is falling at different angles in each picture with the intensity of the rain shown varying in each of the pictures.

For fog we have used image augmentation as well to create our required dataset, that we can use for training and testing purpose. As fog intensity is an important parameter for training, therefore, we will be taking random patches from all over the images, and increase the image's lightness within those patches and with a simple blurring technique, this will give us a nice hazy effect of fog.

In Fig. 2, we have shown the results. Here, 14 different images can be seen which were all created from that one clear picture provided on top of the 13 hazy ones. In each of the pictures, it can be seen that fog of different intensity is shown, some of them are hazier while some are less and some are in between. We have tried to display this varying density below in Fig. 2, which will help improve our training.

With this process completion, our dataset images are prepared according to the requirements needed for our autoen-

coder that we have been able feed as input and hence allowed us to move forward towards our next action, i.e., training the encoder using these processed data.

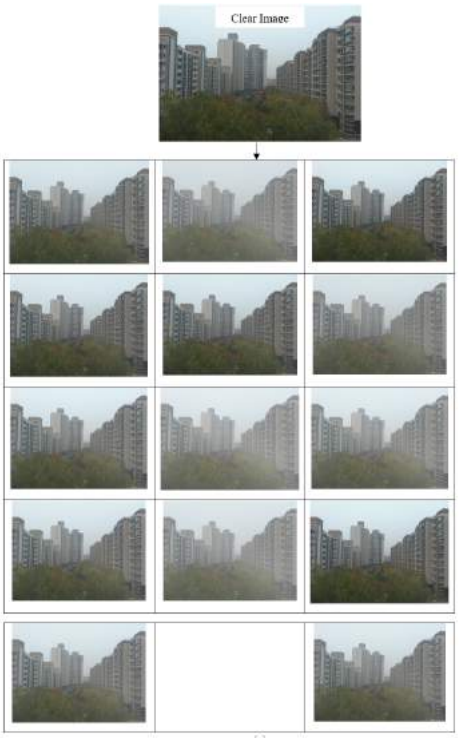

Fig. 2. Fog Dataset (Clear image along with its 14 different density fog sample)

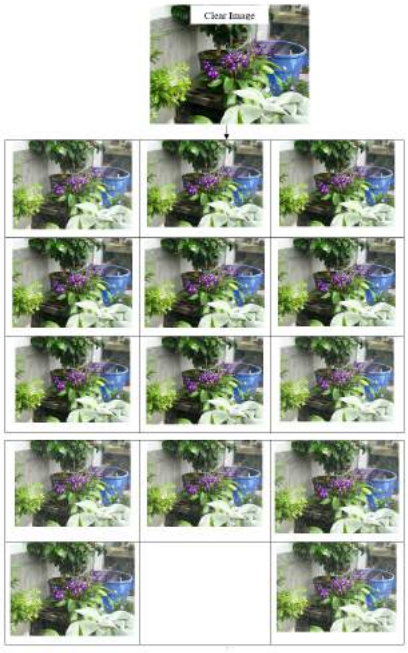

Fig. 3. Rain Dataset (Clear image and 14 different density rain)

C. Convolutional Autoencoder

We have opted to choose Convolutional Neural Networks for our approach, since it has yielded over the years to provide higher accuracy than any other Deep Learning model. Convolutional Autoencoder is established on general autoencoder architecture along with the layers of convolutional encoding and decoding. The core task of this technique is to train the model enough to encode and decode a given image, by reconstructing the given input with the removal of unwanted features [14]. In addition to that, it is better suitable for image processing as the image structures are utilized to maximum capability. Generally, large chunk of information goes missing when the normal image data are chopped and stacked, but by using convolutional autoencoders, the spatial information of the input image data are preserved and the information can be extracted into Convolutional Layer.

1) Working Principle of Autoencoder: Autoencoding technique, implemented along with neural network, has been used to reconstruct an image which compresses the input image that has been affected by weather conditions, the middle layers hold the reduced representation of the input and later decompresses it to form a clear image. The output is reconstructed from this reduced representation of the input. Also, the output layer has the same dimensionality as the input layer.

D. Autoencoder Architecture

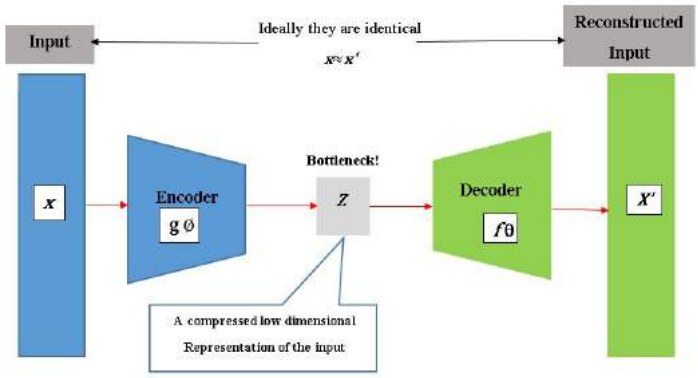

Fig. 4. The Architecture of Autoencoder

The Autoencoder comprises of three major components: Encoder, Code (Bottleneck Layer) and Decoder. As an image is fed onto the Encoder, it extracts the prominent features of the image, reducing its dimensions, i.e., it encodes the input image as a compressed representation in a reduced dimension. The encoding layer consists of multiple layers i.e., Convolutional layers, which form this compressed version of the image which is actually the distorted version of the original image in the Code Layer. The code is a representation of the input learned through the unsupervised training procedure. The bottleneck is also called the maximum point of compression since at this

point the input is compressed to its maximum. Finally, we have the Decoding layer, also called the Deconvolutional Layer, which has a similar structure to the encoder, but performs the reverse operation of the encoder. From the dimensionally compressed image, by taking the extracted pixel features, it creates an image which is seen to have removed the rain or fog effects, reconstructing each dimension of the input by passing it through the network; thus, we get a normalized image.

E. Mathematical Perspective

In Convolutional Autoencoder, the spatiality is safeguarded by distributing the weights within all the inputs. The compressed layer in the encoding part can be described by:

$$\mathbf{h}^i = s(x * W^i + b^i) \tag{3}$$

where b is the bias signal to be shared within the map, W denotes the weight, s denotes activation function, $*$ is the 2D convolution and h is the next layer (compressed).
In the decoding portion, the reverse of the previous operation can be performed to get the reconstructed image x by applying similar activation function, s.

$$\mathbf{y} = \mathbf{s}(\mathbf{h}^i * \tilde{W}^i + c) \tag{4}$$

Where c denotes bias per input channel and y is the reconstructed version of input image.

IV. Implementation

A. Model Training

For our proposed model to work efficiently, where any rain or fog effects will be removed, our model needs to be adequately trained with large amount of dataset as mentioned above. The processed training image data, containing the disturbing weather factors are fed into the Autoencoder along with its corresponding clear images of the dataset as label, in order to train our Autoencoder.

During this training period, the Autoencoder will be able to reconstruct a clear image from the contaminated image that we have fed as input and then compare the constructed image with the original clear image, used as label. This training will continue for a large sum of images. The more the model will train, the higher the accuracy will be with the original image. The objective of a reliable AE is to minimize the reconstructed error.

B. Building the Autoencoder and the Parameters Involved

The encoder and decoder of an AE further consists of multiple layers amongst themselves as shown below:

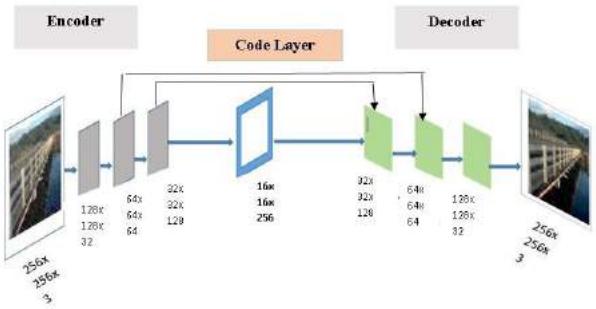

Fig. 5. Workflow of the Model

The encoding layer is comprised of four layers where we use Convolutional layer functions along with Maxpooling. Conv2D (Convolutional function) creates tiny squares called feature maps, which preserve the pixel features and relationship within the input image. Here we have implied different number of filters of weight matrix having dimension of 3*3 at different layers. This reduces the dimensionality of the image as the image is slightly compressed. Figure 5, is a short illustration of our Encoder portion where we are taking input image of size 256*256 and the 3 besides denotes the three RGB channels.

In each layer, Conv2D is followed by Maxpooling2D. Maxpooling (MaxPooling2D) is a pooling operation which measures the maximum (largest) value in every patch of each feature map. It tends to hold the spatial presence of any possible pattern on the feature map of the input image. i. e. output generated from Conv2D. In our model, we have used 2 by 2 window for the pool side, thus on the code, we used MaxPooling2D ((2,2)). This procedure is to be repeated by the requirement of the hidden layers, and as it reduces the size of the image, if the target size in the Bottleneck layer is to make smaller, 16*16 in our model, it can be used multiple times by adjusting the parameters. The output from the first convolutional layer is then again passed to the input of Conv2D along with Maxpooling2D to further compress the image until we reach the Bottleneck Layer (Code Layer), having dimension of 16*16. To begin with, as our input image of 256*256 pixel is fed into the autoencoder, the first layer compresses it into 128*128*32, since the filter used is 32. As shown in Figure 5, it is fed into another layer having filter of 64 and now the feature extraction results an output of 64*64*64. By repeating this procedure for next two layers, we reach the Bottleneck layer (Code layer) where the dimensionality is 16*16*512.

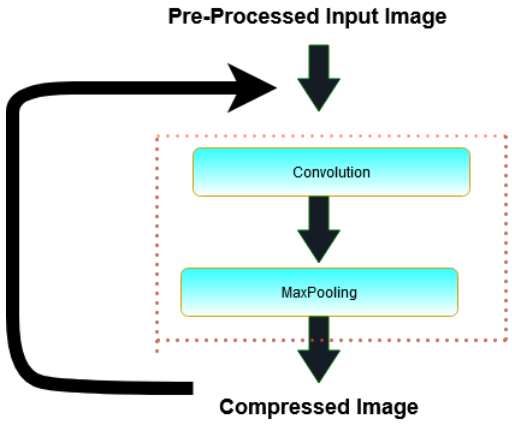

Fig. 6. Addition of multiple Conv2D and Maxpooling

Code Layer

Figure 5 shows, this layer holds the compressed representation of our data, as it drives the network to create a compressed form of the input image. From the input image of 256*256*3, the dimensionality has been reduced to 16*16*512. This holds the spatial features extracted from the input and will now be used by the Decoder Layer to reconstruct the image.

Decoder

At decoding part, the model is to take the compressed data from the Code layer image as input and perform the exact mirror operations to the encoding part, illustrated at the decoder of Fig 5. Similar to Encoder part, decoder comprises of four layers where in each layer we are enlarging the dimension of the image. In each layer, we are using Conv2DTranspose and UpSampling. From the bottleneck layer, the 16*16 dimensional data is taken and the output of this layer is 32*32. Likewise, the generated output is passed onto another layer to generate 128*128 and finally by passing through the last layer of the decoding layer we get our desired reconstructed image. Here, we had to implement the parameters (strides and number of filters) in such a way that the reconstructed image is of same dimensionality as of input image and this was done by ensuring that the number of hidden layers in the encoder is same as number of layers in the decoder. Lastly, we used Conv2D with a depth of three as we want produce the RGB images.

Formation of Feature Maps

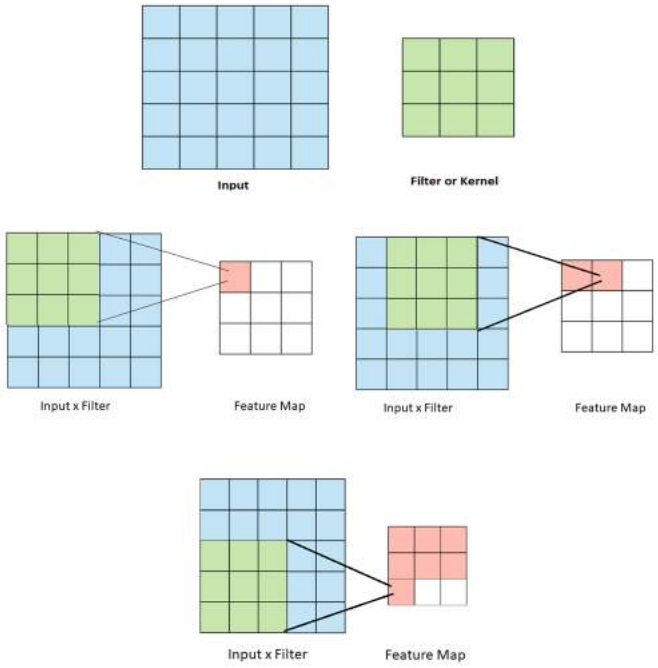

Fig. 7. Basic Formation of Feature Map

Fig 7 illustrates how the feature map is formed in a simpler form by the application of convolutional operation on 2D input. The filter is passed over the image as shown in the figure. For each location, matrix multiplication is performed and its sum forms part of the feature map. As we can see, having stride of 1, the filter is passed over different region of the input image, and this forms the feature map, containing the information of the pixel in a compressed form. However, since we are using RGB images as input, it has three channel and hence there are improvisation to the procedure. For the three channels, the filter to be used much also have three channels. Also, as mentioned earlier, since we are using multiple number of filters, for each filter a feature map is formed and the results are stacked together to form a 3-dimensional feature map as shown in Figure 8 and 9, where the different colors indicate usage of multiple filters.

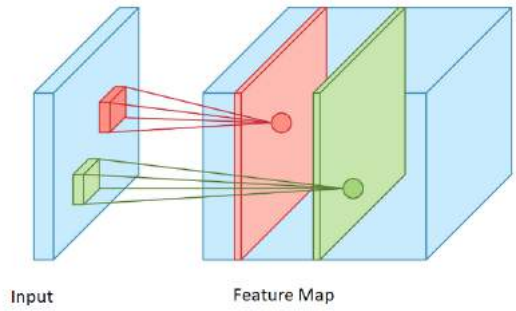

Fig. 8. Detailed Formation of Feature Map for Three Channel (RGB) Image

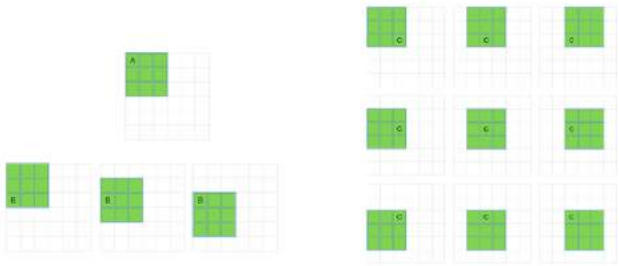

Fig. 10. Simulation of Filters Over Image Without Padding

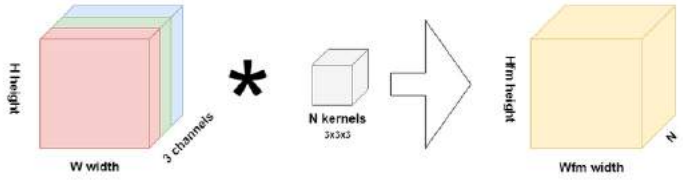

Fig. 9. Overview of The Feature Map

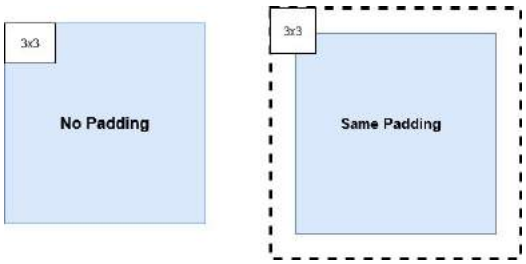

Fig. 11. Consequences of Adding Padding

Parameters

Padding: Padding is amongst the noteworthy parameters involved to train our models, taken by Conv2D function. As shown in Figure 10, as the kernel of stride 1 moves over the input image, the corner pixel information is not extracted as well as compared to the middle pixels, which will result in lack of information on the feature map. To overcome such problem, we need padding. In our model, we have used padding = same. Padding works by the extension of the area of the input image pixels while processing it. Small extensions are done on the frame of the image for the kernel to cover the image, preventing the loss of spatial information on the feature map. In most cases, padding increases the scope of accurate evaluation of image, by preventing shrinkage of the image and also this ensures that pixels at the border of the image have similar contribution to the feature map compared to the pixels in the middle of the image. Though there are different types of padding but we opted to keep the padding as same due to large training dataset, since the training time will be affected. Furthermore, padding of other type would be helpful if our image was of uneven number pixel size in terms of width of height; but our images are of 256*256 pixels.

Stride: Stride is a process for the compression of pictures in neural networks. Stride is a measure of movement of kernel over the picture.. Normally, as the stride is moving, the subsequent yield will be more modest. Stride is a boundary that works related to padding, the element that adds clear, or void pixels to the casing of the picture to take into consideration a limited decrease of size in the yield layer. Generally, it is a method of expanding the size of a picture, to check the way that stride decreases the size. Padding and stride are the central boundaries of any convolutional neural organization.

Activation Function: ReLU (Rectifier Linear Unit) function has been used in our model. ReLU is a piecewise linear function. This function will give output only if the input is positive otherwise it will show zero. The benefits of ReLU are sparsity and decrease in chance of vanishing gradient. Besides, ReLU is among the quickest of all, as it doesn't actuate all neurons simultaneously. ReLU activation function (ReLU) is defined as:

$R(x) = max(0, x)$

One reason that this function is added into a neural network

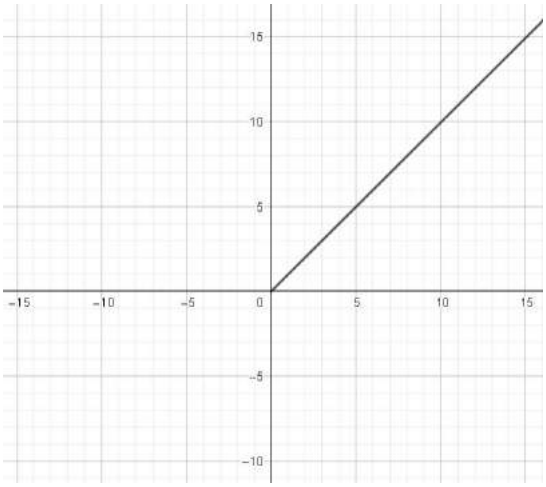

Fig. 12. Graphical representation of ReLU

is to assist the network with learning complex pattern in the data. The figure shows that the output value can either be zero or any positive value, which can be expressed by the equation. This function acquaints nonlinear real-world properties with neural network. Activation function like ReLU have two main part, help a model account for interaction effects and non-linear effects. The reason we are using ReLU activation function over other function as it has no complex math and its computation time is less than other activation function. For that ReLU will take less time to train our model. Beside that we used ReLU for its sparsity as its give zero value for all the negative value and has no range of infinity of positive value, as shown in Figure. Moreover, whenever the matrix multiplication occurs, ReLU ensure that the feature map does not contain any negative value. Moreover, it solves the vanishing gradient problem. The ReLU work has a derivative of 0 over a large portion of its range. For positive value, the derivative is 1. When preparing on a sensible sized batch, there will typically be some information directs giving positive qualities toward some random hub. So, the normal subsidiary is seldom near 0, which permits gradient descent to continue advancing. In addition to that, ReLU is better than Sigmoid or tanh, as it is faster but most importantly it has better accuracy.

Optimization Function

While compiling the Autoencoder, Adam Optimizer is an optimization algorithm. We have used Adam Optimizer, which computes adaptive learning rate for each parameter. Adam optimization algorithm is an expansion to stochastic gradient descent in light of versatile assessment of first order and second order moments for training deep learning model. Adam consolidates the AdaGrad and RMSProp calculations to give an enhancement calculation to manage sparse gradients on noise issue. Adam Optimizer is a versatile learning rate technique, which implies, it figures singular learning rates for various parameters. In our Keras api we used Adam optimizer to get most accurate value from our model. Beside that it take less memory, have straight forward implementation and have efficient output. As we have a large dataset the Adam optimizer is the best solution for us. It is appropriate for rainy/hazy noise in image and its hyper parameters have instinctive understanding and usually require small tuning. The progression size taken by the Adam in every cycle is roughly limited the progression size hyper parameter. Step size of Adam update rule is invariant to the size of the slope, which assists a great deal while experiencing territories with little gradients. It is similar and an extension to Adadelta plus momentum (which smooths gradients based on accumulated first-order information); having a learning rate of 0.001.

Loss Function

In our model, the encoding and decoding involve some loss of information in the process and to keep a track of it we have used Mean Squared Error (MSE). Firstly, the difference between the reconstructed output image and origInal version of the input image is calculated, and the MSE is calculated by averaging the square of the calculated error.[14] MSE can be calculated by the formula,

$$\boldsymbol{MSE} = \frac{1}{W * H} \sum_{i=0,j=0}^{W-1,H-1} [R(i,j) - K(i,j)]^2 \qquad (5)$$

where R(i,j) and G(i,j) denote reconstructed image and the original image respectively, W and H are the width and height of these images. The row and column of the original and reconstructed image are denoted by i and j. MSE embraces the variance along with the bias of the estimate [15]. Compared to other loss function, MSE is suitable as it simply compares between two images and if the two images are exactly same then the difference would be zero, indicating that there is no difference between the images. [16] Throughout the process, our goal was to minimize this value.

V. Result And Analysis

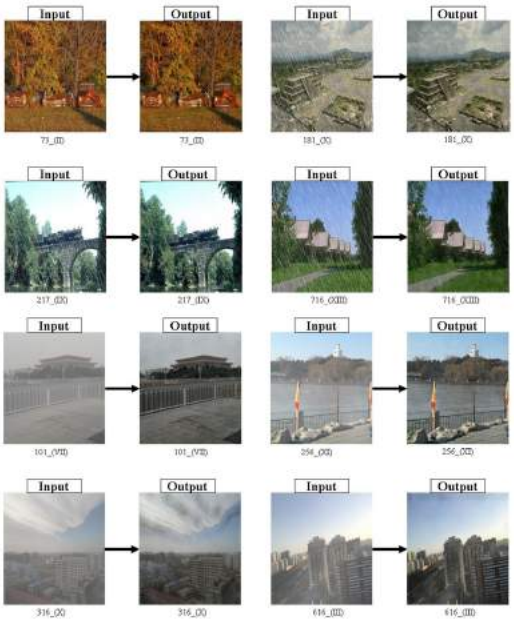

Fig. 13. Illustration of Model Output with Corresponding Output

In our thesis we have taken fourteen (xiv) different orientation images to train our model. Then we fed that data into our autoencoder's model. As we can demonstrate that with a small pictorial illustration as in Figure 14.

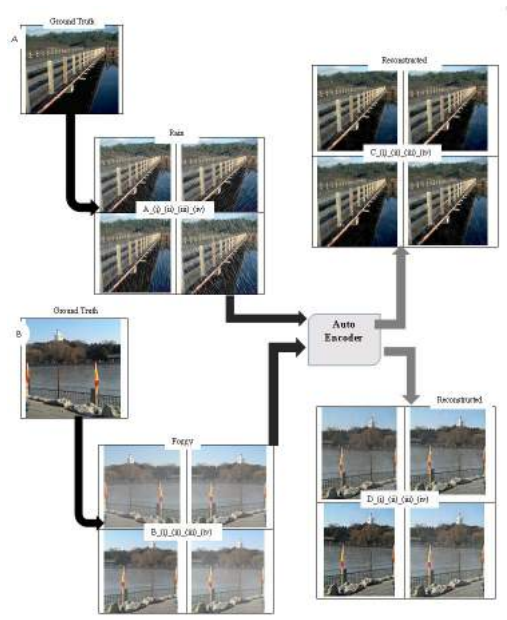

Fig. 14. Reconstruction of Image Using Autoencoder

For keeping this pictorial demonstrate small and simple we are showing only four (iv) images has been fed into the autoencoder model but in our actual code we have added fourteen images. In the Figure 14, we can see ground truth A has four different orientation raining scenarios. They are A(i), A(ii), A(iii) and A(iv). These four images are then fed into the auto encoder for model training. Upon training the autoencoder gives us the result C(i), C(ii), C(iii) and C(iv). These reconstructed images are actually the clear image result of the rainy image. Similarly, we can see ground truth B has four different density fog scenarios. They are B(i), B(ii), B(iii) and B(iv). These four images are then fed into the auto encoder for model training. Upon training the autoencoder gives us the result D(i), D(ii), D(iii) and D(iv). These reconstructed images are actually the clear image result of the foggy image. So, this is how our actual autoencoder training and reconstruction of the image occurring in the actual code.

Structural Similarity Index

Structural Similarity Index, known by SSIM, is used to check the reliability of our model. It is salient for us to evaluate and analyze our method of data training. SSIM looks at a group of pixels which makes it more effective. It evaluates images considering sensitive changes in local structure. Using SSIM our goal is for the network to learn to produce distortion free and structurally correct images. The SSIM index is calculated between two windows x and y of common size W*W is:

$$\textbf{SSIM(x,y)} = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{6}$$

Here, x, y are the W*W frame taken from ground truth and reconstructed image. is the average of x, is the average of y, is the variance of x, is the variance of y, is the covariance of x and y, and are two variables to stabilize the division with weak denominator and is the dynamic range of the pixel values.

SSIM examines the identicality with the original image [17], here in our model, it will be ground truth of the image. Calculated value of SSIM varies from 0 to 1, where the closer the value is to 1, the better is the output generation with respect to the original ground truth image. SSIM is helpful in assessing the quality of the reconstructed image [16].

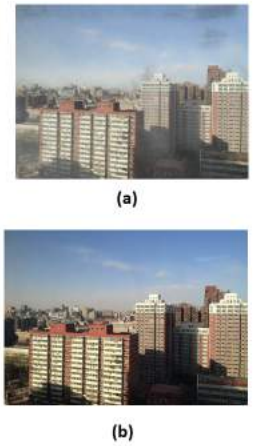

Fig. 15. Comparison of Output and It's Ground Truth (Original Image): (a) Reconstructed Image (b) Ground truth

Figure 15 shows the visual comparison between (a) the reconstructed image by the model and (b) the actual image (ground truth). The calculated SSIM value is 0.90 which depicts the accuracy of our model with the testing dataset.

In both the images, Figure 16 (a) and (b) for raining condition and Figure 17 (a) and (b) for fog condition, the green boxes are seemed to be in identical position, this denotes that our algorithm for SSIM is finding the dissimilarities within these regions. For better visualizing, grey plotting is done, as shown in Figure 16 (c) and Figure 17 (c), where places of similarity are whitened and on dissimilarity regions it forms grey. Furthermore, we have used masking as shown in 16 (d) and 17 (d) where the green plots that can be seen are the dissimilarities. The dissimilarities are then plotted in the main image, to see the places of having certain mismatches.

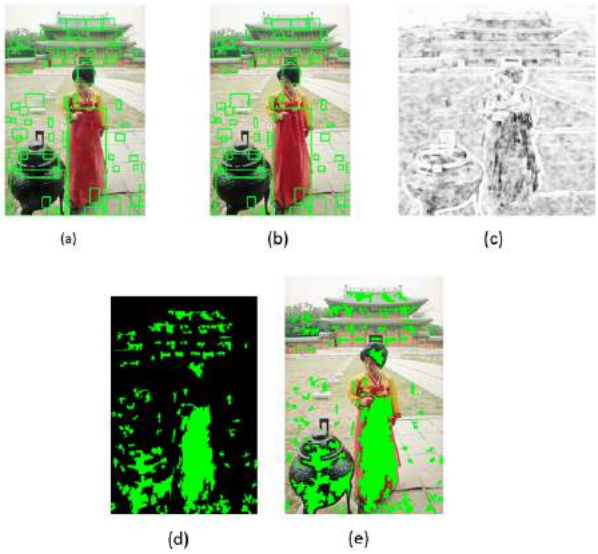

Fig. 16. Detailed Analysis of Rainy Images (a) reconstructed image, (b) ground truth, (c) differences plot in grayscale, (d) mask plot and (e) filled mask on clear image

PSNR (Peak Signal to Noise Ratio)

To assess the quality of the output image of our model, we have used Peak signal to noise ratio methodology. It is the ratio of maximum signal's energy and the noise of the newly formed image, where the signal is data of actual image which is stated as the ground truth and the noise denotes the error in the image. It is amongst one of the prominent methods for evaluating the quality of reconstructed image [15], and it is expressed in decibel. The higher the value the better will be the quality of the image.

$$\boldsymbol{PSNR} = 10 * \log_{10} \frac{MAX_I^2}{MSE} \tag{7}$$

where, $MAXi$ is usually 255, which is maximum pixel value and MSE is Mean Squared Error.

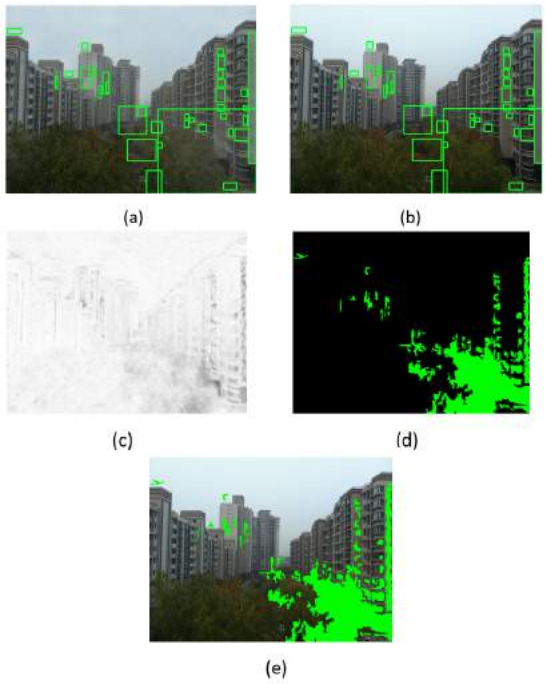

Fig. 17. Detailed Analysis of Foggy Image (a) reconstructed image, (b) ground truth, (c) differences plot in grayscale, (d) mask plot and (e) filled mask on clear image

Fig. 18. (a) PSNR of Rainy Image

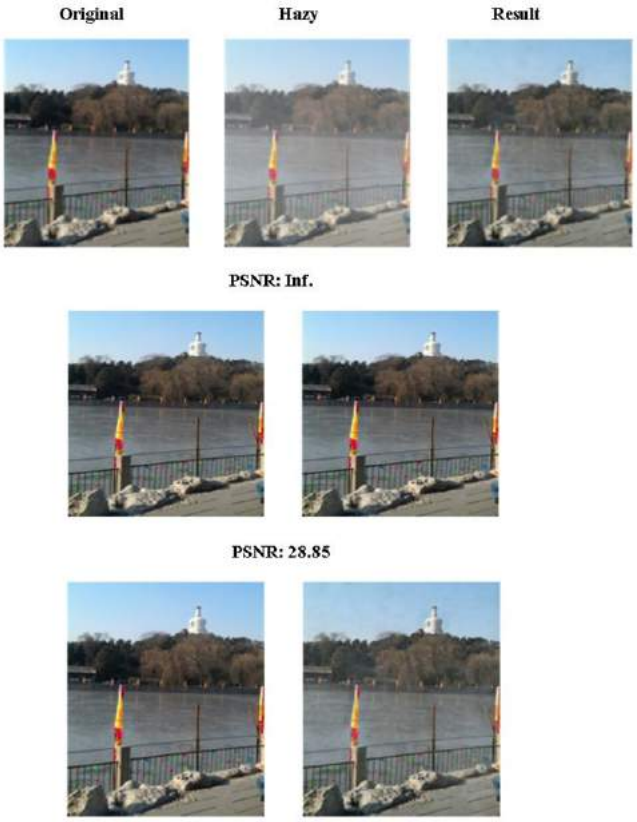

Fig. 19. (b) PSNR of Foggy Image

VI. Conclusion

Computer vision is being used in providing solutions to many real-life issues, traffic Monitoring, number plate detection, crime detection, social distancing monitoring, mask detection due to the recent pandemic, agriculture and farm monitoring etc. With the rapid implementation of these technologies, the accuracy and effectiveness in all situations have also become a huge concern. These are sensitive real-life models and cannot afford to have any defect which would lead to severe aftermath. We came up with the proposed idea to make computer vision not only effective, but also more accurate and more intelligent. Humans have always fixated themselves on improving life across every spectrum, and basically, the use of our technology aims to become the vehicle for doing just that. We tried to work on a solution on major hindrance caused by weather disturbance in the way of a successful image analyzation. Hence, the purpose of our thesis paper is to develop a model which considers different types of environmental elements like rain and fog concurrently to enable composite analyzation and normalization of images to make these real-life implementations more effective.

References

[1] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[2] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49.

[3] R. Yamashita, M. Nishio, R. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology. insights imaging 9 (4), 611–629 (2018)."

[4] S. Wang, Y. Zhang, D. Yang, and Z. Chen, "Ssim prediction for h. 265/hevc based on convolutional neural networks," in *2019 IEEE Visual Communications and Image Processing (VCIP)*. IEEE, 2019, pp. 1–4.

[5] J. D. Crisman and C. E. Thorpe, "Unscarf-a color vision system for the detection of unstructured roads." in *ICRA*, 1991, pp. 2496–2501.

[6] K. G. Lore, A. Akintayo, and S. Sarkar, "Llnet: A deep autoencoder approach to natural low-light image enhancement," *Pattern Recognition*, vol. 61, pp. 650–662, 2017.

[7] Z. Shi, Y. Feng, M. Zhao, and L. He, "A joint deep neural networks-based method for single nighttime rainy image enhancement," *Neural Computing and Applications*, pp. 1–14, 2019.

[8] X. Li, T. Pang, B. Xiong, W. Liu, P. Liang, and T. Wang, "Convolutional neural networks based transfer learning for diabetic retinopathy fundus image classification," in *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. IEEE, 2017, pp. 1–11.

[9] M. Koziarski and B. Cyganek, "Image recognition with deep neural networks in presence of noise–dealing with and taking advantage of distortions," *Integrated Computer-Aided Engineering*, vol. 24, no. 4, pp. 337–349, 2017.

[10] L. Gondara, "Medical image denoising using convolutional denoising autoencoders," in *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2016, pp. 241–246.

[11] J. Zheng, K. P. Valavanis, and J. M. Gauch, "Noise removal from color images," *Journal of Intelligent and Robotic Systems*, vol. 7, no. 3, pp. 257–285, 1993.

[12] T. Wang, X. Yang, K. Xu, S. Chen, Q. Zhang, and R. W. Lau, "Spatial attentive single-image deraining with a high quality real rain dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 270–12 279.

[13] B. Ashwath, "Indoor training set (its) [reside-standard]," Oct 2020. [Online]. Available: https://www.kaggle.com/balraj98/indoor-training-set-its-residestandard/metadata

[14] A. Ceballos-Arroyo, S. Robles-Serrano, and G. S. Torres, "Remoción de lluvia en imágenes por medio de una arquitectura de autoencoder," *Investigación e Innovación en Ingenierías*, pp. 140–167, 2020.

[15] D. Asamoah, E. O. Oppong, S. O. Oppong, and J. Danso, "Measuring the performance of image contrast enhancement technique," *International Journal of Computer Applications*, vol. 181, no. 22, pp. 6–13, Oct 2018. [Online]. Available: http://www.ijcaonline.org/archives/volume181/number22/30015-2018917899

[16] A. Z Wang, H. C Bovik, R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[17] D. Asamoah, E. Ofori, S. Opoku, and J. Danso, "Measuring the performance of image contrast enhancement technique," *International Journal of Computer Applications*, vol. 181, no. 22, p. 6–13, 2018.