Detecting Inappropriate Exposure in Image by Deep Feed Forward Neural Networks

B.Sc. Engineering Project

April 20, 2022

A Project submitted to Department of Computer Science and Engineering.
University of Rajshahi for partial fulfilment of the requirements
for the Degree of Bachelor of Science in Computer Science and Engineering

By

Faiz Ahmed
ID:1710676128
Session: 2016-2017

Tasrin Sultana
ID:1612476138
Session: 2015-2016

Supervised By

Sangeeta Biswas
Associate Professor

Department of Computer Science and Engineering
University of Rajshahi

Acknowledgements

Regarding the outcome of our project, we would like to express deepest sense of gratitude and respect to our supervisor **Sangeeta Biswas**, Associate Professor, Department of Computer Science and Engineering, University of Rajshahi for the complete supervision, advice, encouragement and continued support of my project. Her patience, motivation, collaboration and interaction were key factors in the success of our project. Without her dedicate involvement in every step throughout the process, this project would have never been accomplished.

We also like to thank all of our friends for their encouragement and supports to complete our project successfully. We want to express our gratitude to our family for always supporting us and which have always been our source of motivation and happiness for us. Finally, we would like to thank God from the deepest of our heart for making us able to finish this project work.

Contents

List of Figures

Abstract

Capturing photographs with wrong exposures remains a major source of errors in camera-based imaging. It happens not only in small cameras attached to mobile phones or computers but also in large cameras designed for capturing sensitive images, such as fundus cameras. Exposure problems are categorized as overexposure and underexposure. In overexposure, the camera exposure time is too long, resulting in bright and washed-out image regions. On the contrary, in underexposure, the camera exposure is too short, resulting in dark regions. Both underexposure and overexposure greatly reduce the contrast and visual appeal of an image. Sometimes inappropriate exposure creates a big problem so that the image cannot be used in its targeted fields, such as the medical sector and biometrics. Therefore, a camera operator needs to re-capture images. If a camera operator recognizes the inappropriate exposure problem just after capturing an image, re-capturing images will not be a big problem. However, patients and volunteers often do not want to come back for re-capturing images if they leave the location of the photo spot. Our project aims to automatically detect inappropriate exposure (i.e., overexposure and underexposure) in images. For that, we develop deep feed-forward neural network-based classifiers. By conducting experiments on a publicly available data set, in this work, we investigate the performances of both fully connected and convolutional feed-forward neural networks for classifying normal, overexposed, and underexposed images.

Chapter 1

Introduction

The exposure used at capture time directly affects the overall brightness of the final rendered photograph. Digital cameras control exposure using three main factors: (i) capture shutter speed, (ii) f-number, which is the ratio of the focal length to the camera aperture diameter, and (iii) the ISO value to control the amplification factor of the received pixel signals. In photography, exposure settings are represented by exposure values(EVs), where each EV refers to different combinations of camera shutter speeds and f-numbers that result in the same exposure effectalso referred to as equivalent exposures in photography.

Digital cameras can adjust the exposure value of captured images for the purpose of varying the brightness levels. This adjustment can be controlled manually by users or performed automatically in an auto-exposure (AE) mode. When AE is used, cameras adjust the EV to compensate for low/high levels of brightness in the captured scene using through-the-lens (TTL) metering that measures the amount of light received from the scene [5]. Exposure errors can occur due to several factors, such as errors in measurements of TTL metering, hard lighting conditions (e.g., very low lighting and backlighting), dramatic changes in the brightness level of the scene, and errors made by users in the manual mode. Such exposure errors are introduced early in the capture process and are thus hard to correct after rendering the final 8-bit image. This is due to the highly nonlinear operations applied by the camera image signal processor (ISP) afterwards to render the final 8-bit standard RGB (sRGB) image [6]. Figure 1.1 shows typical examples of images with exposure errors. In Fig 1.1, exposure errors result in either very bright im-

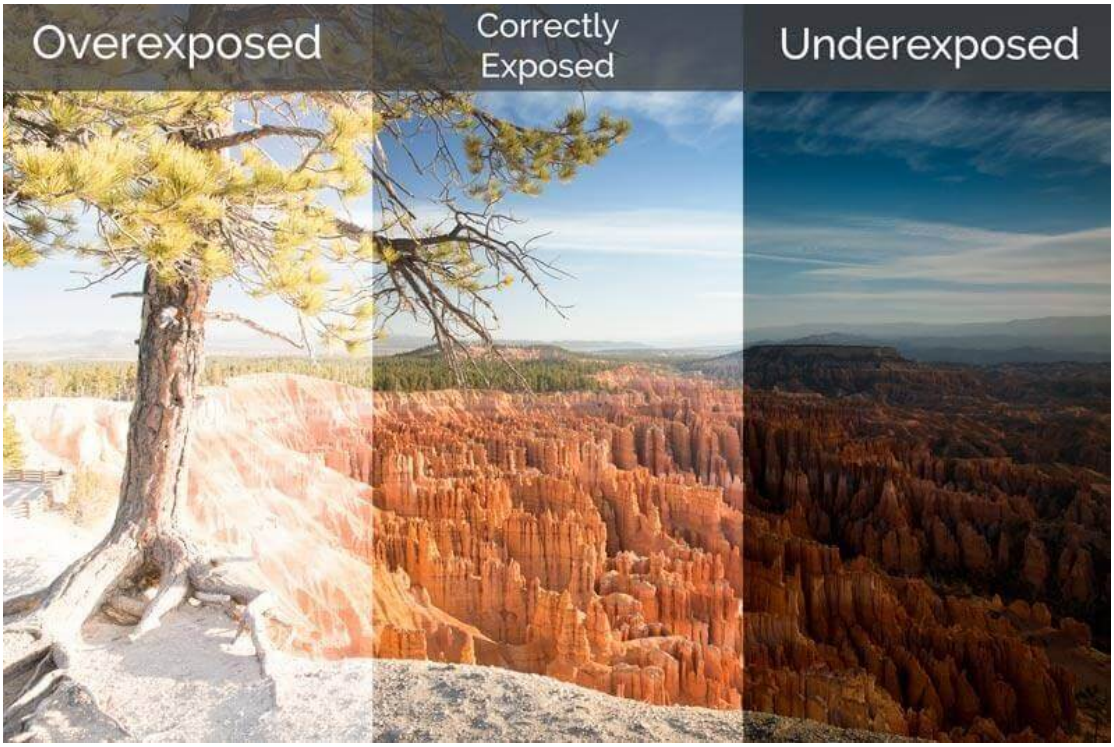

Figure 1.1: Example of over-exposed, under-exposed and correctly-exposed image (collected from studiobinder).

age regions, due to overexposure, or very dark regions, caused by underexposure errors, in the final rendered image.

Chapter 2

Background

2.1 What is Exposure ?

One of the most important terms in photography is exposure. When we snap a photograph, we click the shutter button to open the aperture of camera, allowing light to enter and trigger a sensor response. The amount of light that reaches the camera sensor or film is known as exposure. It has a big impact on how bright or dark our photos look. The amount of light that reaches the camera sensor after passing through the camera lens for a set period of time determines the final appearance of the image.

According to amount of ligh present in image, we can categorize image in 3 classes.

1. Over-exposed Image
2. Under-exposed Image
3. Correctly-exposed Image

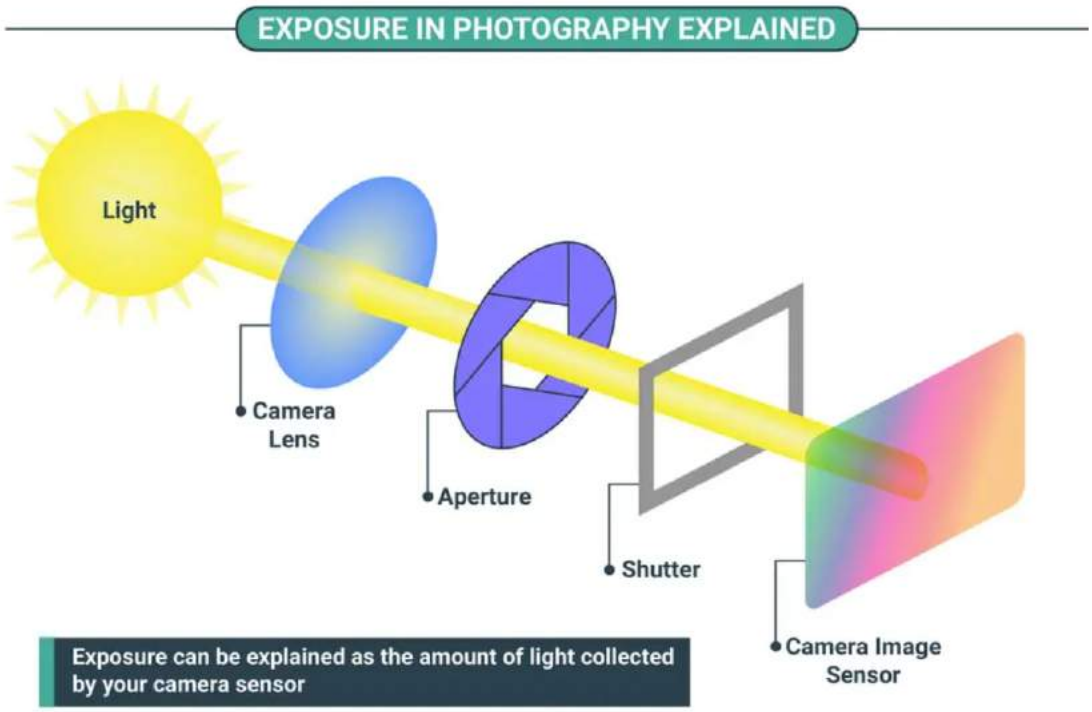

Figure 2.1: Concept of Exposure in Image (collected from capturetheatlas).

2.1.1 Over-exposed Image

Overexposure is when an image appears brighter than it should, or brighter than neutral exposure. When too much light hits sensor of camera, it results in an extremely bright image that is now overexposed. Overexposure limits detail in the photo and reduces any opportunity for shadowing or distinguishable highlights in the image. An overexposed photo tends to be washed out and it looks much brighter than it should be due to too much light hitting camera sensor. Details are less evident and the whole image appears too white.Below is an example of an over-exposed photo.

In 2.2 the brightness overpowers any distinguishing details we might see. No shadows exist and there are not any highlights other than the blobs of light all over the image.

Figure 2.2: Over-exposed image (collected from studiobinder).

2.1.2 Under-exposed Image

Underexposure is when an image appears darker than it should, or darker than neutral exposure. An underexposed image is the result of not enough light hitting the sensor of camera. Underexposed images often lack detail and the objects or subjects can even blend together in the shadows of the image. This type of image is too dark, details will be lost in the shadows and darkest areas. Underexposed photos blend the focus at the forefront with the background, making it hard to identify the aesthetics in image. Below is an example of an underexposed photo:

Figure 2.3: Under-exposed image (collected from vanceai).

2.1.3 Correctly-exposed Image

Correctly exposed image is the one that feels just bright or dark enough so that both the shadows and highlights are as they feel the most natural and comfortable to look at. Theoretically, such a photograph contains no lost highlights or shadows, meaning all the detail is clearly distinguishable and as close as possible to real life. The following version of the image sample can be considered as correctly-exposed:

Figure 2.4: Correctly-exposed image (collected from photographylife).

2.2 Deep Neural Network

Deep neural network (DNN) is one type of artificial neural network (ANN) with multiple hidden layers between the input layer and output layer. DNN models were originally inspired by neurobiology. A neuron (biological) takes numerous signals through the synapses contacting its dendrites and transmit a single stream of action potentials out by its axon. By categorizing its input patterns decrease the complexity of numerous inputs. Inspired by this system, ANN models are composed of units that integrate multiple inputs and generate a single output.These artificial neurons (AN) are collected into layers, and the outputs of one layer of neuron becoming the inputs of the next layer neuron in the sequence. DNNs are generally one type of feedforward networks in which data flows from the input layer to the output layer without looping back. At first, the deep neural network generates a map of all the virtual neurons and assigns every neurons random nu-

merical values which is called “weights”, which is used in connections between them. The given weights and the inputs are multiplied and return an floating number as output between 0 and 1. When the DNN can not accurately recognize a particular pattern, an algorithm would adjust the weights. In this approach the algorithm can make these parameters more influential and accurate, until it determines the accurate mathematical manipulation to fully process the data.There are two phase of neural network which are training and testing, training phase determines the weights and prediction is done by testing phase .

2.2.1 Why is it “Deep” ?

Complex DNN have many layers, that is why the name “deep” networks. Complex non-linear relationships can be modeled by DNN. DNN employ deep architectures by having many hidden layers in neural networks. Deep" means having higher complex function in the number of layers and units in a single layer. To build more accurte model, the model needs to train on huge dataset. DNN model can learn high level of patterns by huge dataset and many number of hidden layers.

2.2.2 Types of Deep Neural network

Following three types of DNN are very popular nowadays:

1. Feed-forward Neural Networks
2. Convolutional Neural Networks (CNN)
3. Recurrent Neural Networks (RNN)

2.2.3 Feed-forward Neural Networks

A feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle. As such, it is different from its descendant: recurrent neural networks. The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction forward from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the

network. This is the most basic type deep neural network. A feedforward neural network has input layer and output layer along with one or many hidden layers with numerous neurons combined together. Every neuron must have an activation function such as ReLU or sigmoid, softmax etc. Every layer is feeding the next layer with the result of their calculation. This procedure is followed all the way through the hidden layers to the output layer.

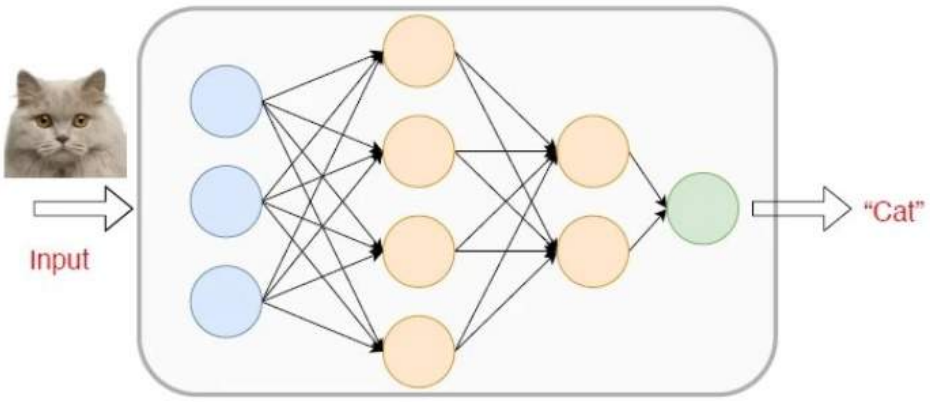

Figure 2.5: Feed-forward Neural Network (collected from viso.ai).

2.2.4 Convolutional neural network (CNN)

Convolutional neural networks (ConvNets or CNNs) is one of the most common types of neural networks used to recognize and classify pictures. CNN is one type of feedforward neural network. CNNs are widely employed in domains such as object detection, face recognition, and so on. To train and test deep learning CNN models, each input image will be passed through a sequence of convolution layers using filters (Kernals), pooling, fully connected layers (FC), and the Softmax function to classify an object with probabilistic values ranging from 0 to 1.

Multiple layers of artificial neurons make up convolutional neural networks. Artificial neurons are mathematical functions that calculate the weighted sum of various inputs and output an activation value, similar to their biological counterparts. Each layer creates many activation functions that are passed on to the next layer when you input an image into a ConvNet. Basic features such as horizontal or diagonal edges are usually extracted by the first layer. This information is passed on to the next layer, which is responsible for detecting more complicated features like corners and combinational edges. As we go deeper into the network, it can recognize even more complex elements like objects, faces, and so on.

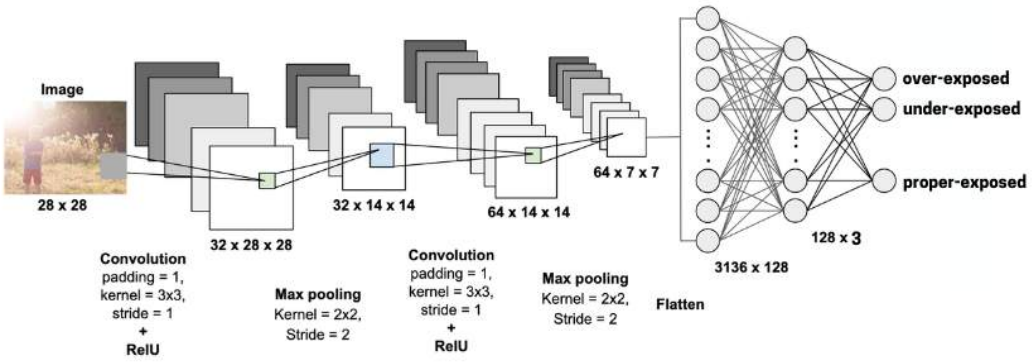

Figure 2.6: Classifying over-exposed, under-exposed, properly-exposed image by a CNN.

Convolution Layer

Before we get into how convolution layer of CNN works, let us go over some fundamentals like what an image is and how it is represented. A grayscale image is nothing more than a matrix of pixel values with a single plane, whereas an RGB image is nothing more than a matrix of pixel values with three planes. Take a look at this illustration to learn more.

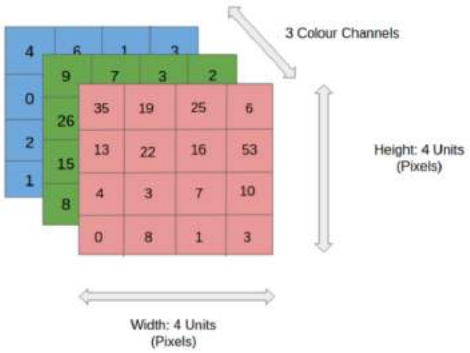

Figure 2.7: 3 matrix of 3 planes in a RGB image (collected from analyticsvidhya).

To keep things simple, we'll use grayscale photos to explain how CNNs convolution work on image.

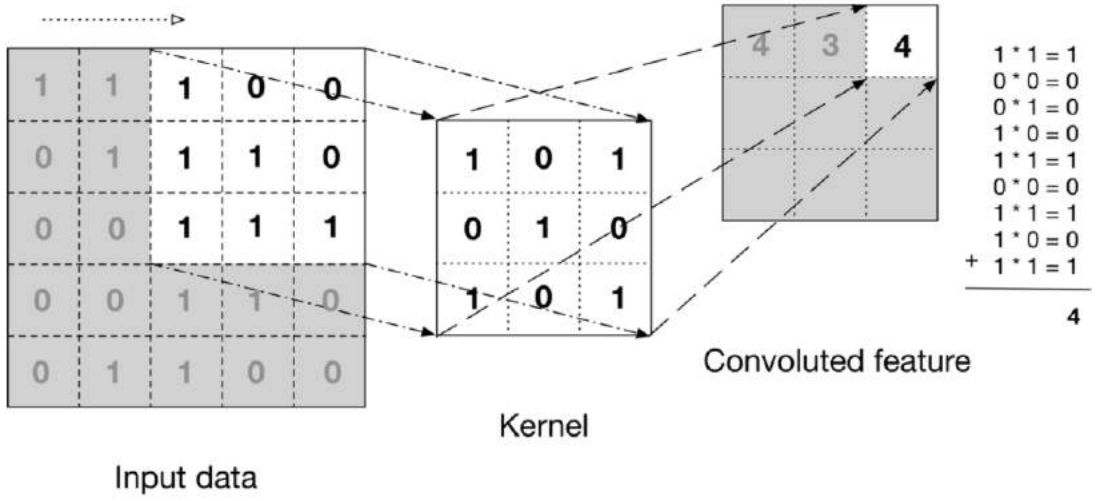

Figure 2.8: Convolution operation on gray scale image (collected from analyticsvidhya).

In convolution operation if we use,

an image matrix of dimension (h Œ w x d)

a kernel of dimension (k_h x k_w x d)

dimension of output will be (h - k_h + 1)(w - k_w + 1) x 1

In CNN, the first few hidden layers are convolutional layers where convolution and pooling operations take place [7]. Each convolution operation outputs a numerical value by applying a filter (i.e., a matrix of weights) to a sub-region of an image [8].

Pooling Layer

When the photos are too huge, the pooling layers portion would lower the number of parameters. Spatial pooling, also known as subsampling or downsampling, decreases the dimensionality of each map while preserving crucial data. Different types of spatial pooling exist :

1. Max Pooling
2. Average Pooling
3. Sum Pooling

The largest element from the corrected feature map is used in max pooling. The average pooling could be calculated by taking the average of all element. Sum pooling refers to the sum of all elements in a feature map.

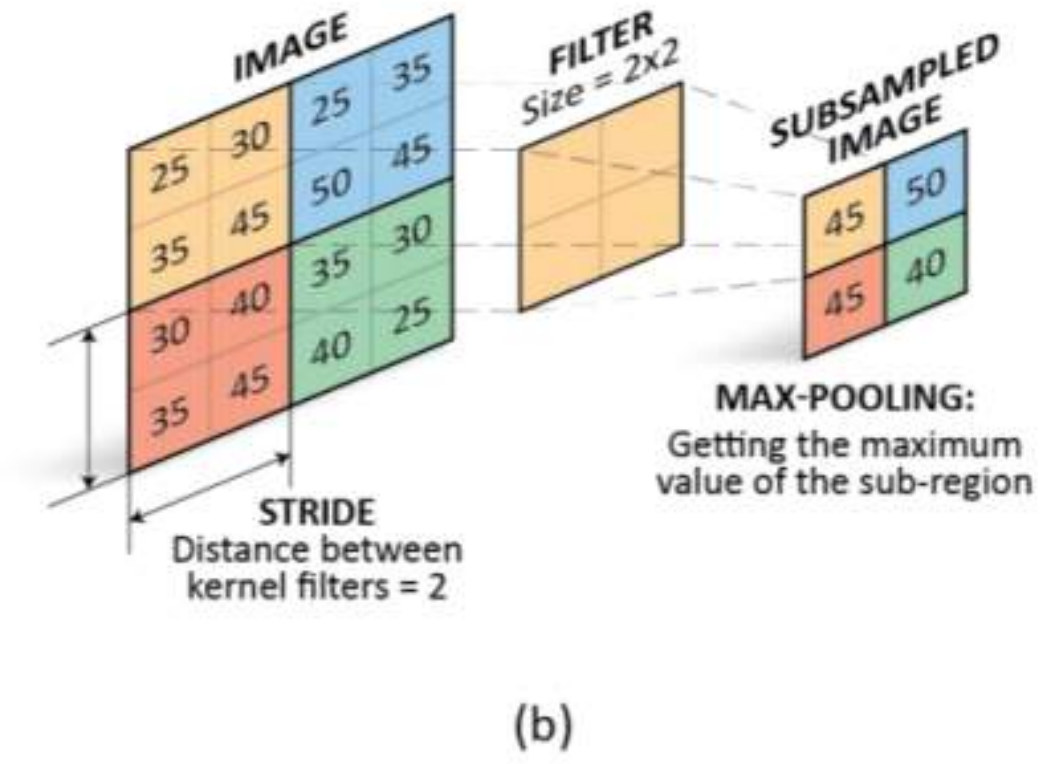

Figure 2.9: Example of max-pooling operation performed with a 2Œ2 filter

Figure 2.9 illustrates max-pooling, one of the most commonly used pooling operations, where a 2D image is divided into fixed-sized sub-regions (i.e., kernels) and the maximum value in each sub-region is passed to the next layer.

Strides

The number of pixels shifted over the input matrix is referred to as the stride. When the stride is set to 1, the filters are moved one pixel at a time. We shift the filters two pixels at a time when the stride is two, and so on.

Padding

When performing a standard convolution operation, the image shrinks by a factor equivalent to the filter size plus one. We have two options:

1. Same Padding:

   Same padding is the procedure of adding enough pixels at the edges so that the resulting feature map has the same dimensions as the input image to the convolution operation.

2. Valid Padding:

   Valid padding means that we only apply a convolutional filter to valid pixels of the input. Since only the pixels of the original image are valid, valid padding is equivalent to no padding at all.

2.2.5 Recurrent Neural Network (RNN)

Another type of artificial neural network that uses sequential data feeding is the recurrent neural network (RNN). RNN are neural networks in which data can flow in any direction. RNN have been designed to address the time-series problem of sequential input data. Input of RNN is made up of the current input and prior samples. As a result, the connections between nodes form a directed graph that follows a temporal sequence. Furthermore, each neuron in an RNN has an internal memory that stores the information from previous samples computations. RNN are called recurrent as they repeat the same task for every element of a sequence, with the output being based on the previous computations.

Because of it has superiority in processing data with a variable input length, RNN models are commonly employed in natural language processing (NLP). The goal of AI in this case is to create a system that can understand human-spoken natural language, such as natural language modeling, word embedding and machine

translation. Each subsequent layer in an RNN is made up of nonlinear functions of weighted sums of outputs and the preceding state. As a result, the basic unit of RNN is named "cell" and each cell is made up of layers and a series of cells that allow recurrent neural network models to be processed sequentially.

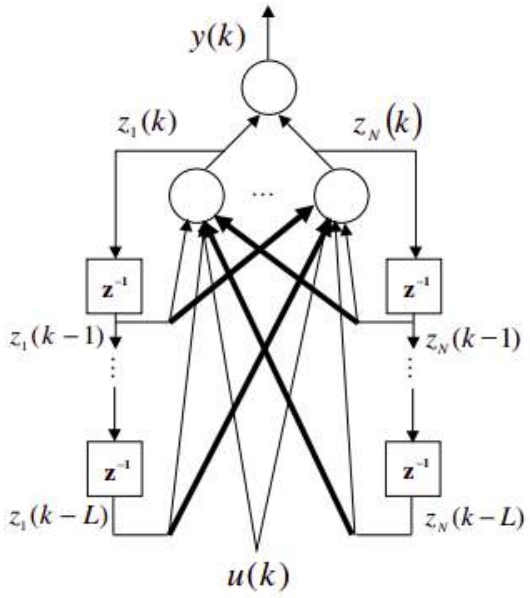

Figure 2.10: Concept of a Globally recurrent neural network architecture (GRNN) (collected from [1]

2.2.6 Transfer Learning

For a particular dataset, a CNN model can be trained from scratch. However, to achieve optimal results, a large amount of training data coupled with the proper selection of optimal hyper-parameters (e.g., number of layers, number of nodes in each layer, filter size, number of epochs, learning rate, and dropout) is required which might take substantial amount of time for training [8]. One way to overcome this challenge is to perform transfer learning, i.e., using a CNN model (e.g., GoogleNet, AlexNet, VGG-16) that is pre-trained with a different but related dataset, and partly re-trained with the desired dataset. Particularly, transfer learning allows the model to remember high- and mid-level features (e.g., edge, shape, color) learned from the source dataset and apply these features (with minor adjustment) to effectively distinguish the classes in the target dataset. Building upon previous studies that have found significantly better and consistent perfor-

mance using transfer learning. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning [9]. In this research, we used the VGG-16 model, pre-trained on the ImageNet dataset [10]. The VGG-16 architecture is selected, particularly, for its wide adaptation in various domain, consistent performance comparable to the state-of-the-art techniques [10], and manageable size (i.e., only 16 layers of convolution) that allows to port the model on embedded system (e.g., smartphone, drone, autonomous vehicles, portable smart devices) with limited computational power.

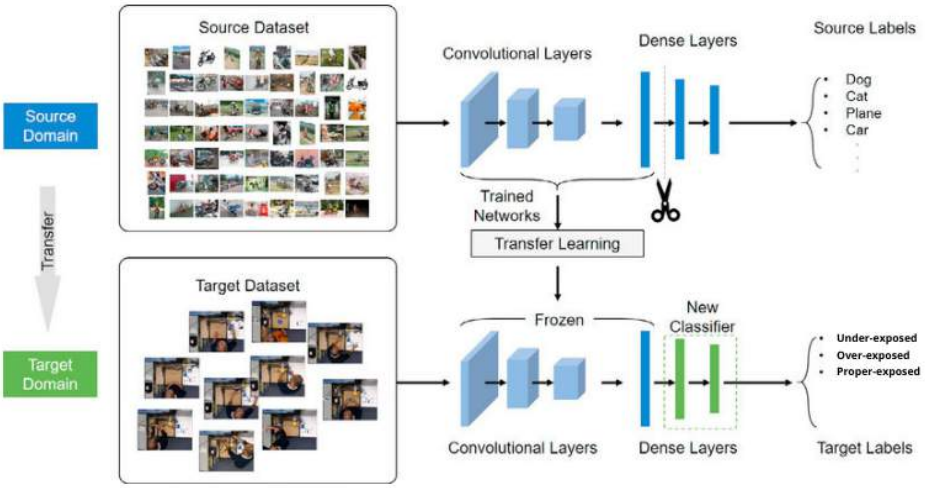

Figure 2.11: Classifying over-exposed, under-exposed, properly-exposed image by transfer learning method(Collected from [2])

2.3 Activation Function

An Activation Function(AF) are function which decides whether a neuron should be activated or not. This means that during the prediction phase, it will determine whether the input of neuron input to the network is essential or not. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).The primary role of activation function is to transform the summed weighted input from the node into an output value to be fed to the next hidden layer or as output as shown in Fig 2.12. These AF are often referred to as a transfer function in some literature.

Activation function can be either linear or non-linear depending on the func-

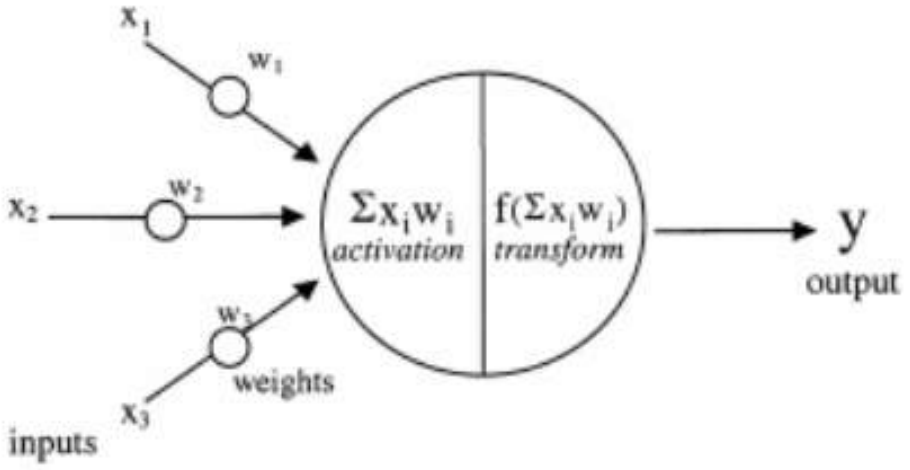

Figure 2.12: Activation function(collected from [3])

tion it represents, and are used to control the outputs of neural networks, across different domains from object recognition and classification [11], [12], to speech recognition [13], segmentation [14], scene understanding and description [15] and other domains to mention a few, with early research results by [16], validating categorically that a proper choice of activation function improves results in neural network computing.

For a linear model, a linear mapping of an input function to an output, as performed in the hidden layers before the final prediction of class score for each label is given by the affine transformation in most cases [17]. The input vectors x transformation is given by

$$f(x) = w^T x + b \tag{2.1}$$

where x = input, w = weights, b = biases.
Furthermore, the neural networks produce linear results from the mappings from equation 2.1 and the need for the activation function arises, first to convert these linear outputs into non-linear output for further computation, especially to learn patterns in data. The output of these models are given by

$$y = w_1 x_1 + w_2 x_2 + .. + w_n x_n + b \tag{2.2}$$

These outputs of each layer is fed into the next subsequent layer for multilayered networks like deep neural networks until the final output is obtained, but they are linear by default. The expected output determines the type of activation function to be deployed in a given network. However, since the output are linear in nature, the nonlinear activation functions are required to convert these linear inputs to non-linear outputs. These AFs are transfer functions that are applied to the outputs of the linear models to produce the transformed non-linear outputs, ready for further processing. The non-linear output after the application of the AF is given by

$$y = \alpha(w_1x_1 + w_2x_2 + .. + w_nx_n + b) \tag{2.3}$$

The need for these AFs include to convert the linear input signals and models into non-linear output signals, which aids the learning of high order polynomials beyond one degree for deeper networks. A special property of the non-linear activation functions is that they are differentiable else they cannot work during backpropagation of the deep neural networks.

2.3.1 Various types of activations function

There are different type of non-linear activation function. Given some below:

1. Sigmoid
2. Tanh
3. Rectified Linear Unit(ReLU)
4. Softmax

Sigmoid Function

The Sigmoid AF is sometimes referred to as the logistic function or squashing function in some literature [18]. The sigmoid is a non-linear AF used mostly in feedforward neural networks. This function takes any real value as input and outputs values in the range of [0 to +1].

$$real\ value = [-\alpha\ ,\ +\alpha] \tag{2.4}$$

The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below Fig: 2.13 and mathematically it can be represented as Equation 2.5

$$f(x) = \left(\frac{1}{1+e^{-x}}\right) \tag{2.5}$$

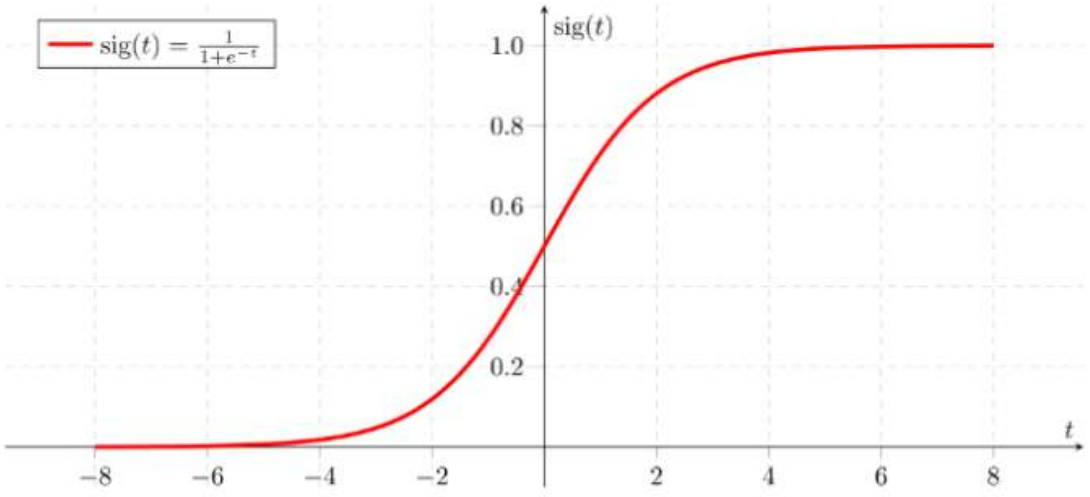

Figure 2.13: Sigmoid Function (collected from towardsdatascience.com).

The sigmoid function appears in the output layers, and they are used for predicting probability based output and has been applied successfully in binary classification problems, modeling logistic regression tasks as well as other neural network domains.

Tanh

tanh(x) activation function is widely used in neural networks.
tanh(x) is defined as:

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.6}$$

Historically, the tanh function became preferred over the sigmoid function as it gave better performance for multi-layer neural networks. But it did not solve the vanishing gradient problem that sigmoids suffered, which was tackled more

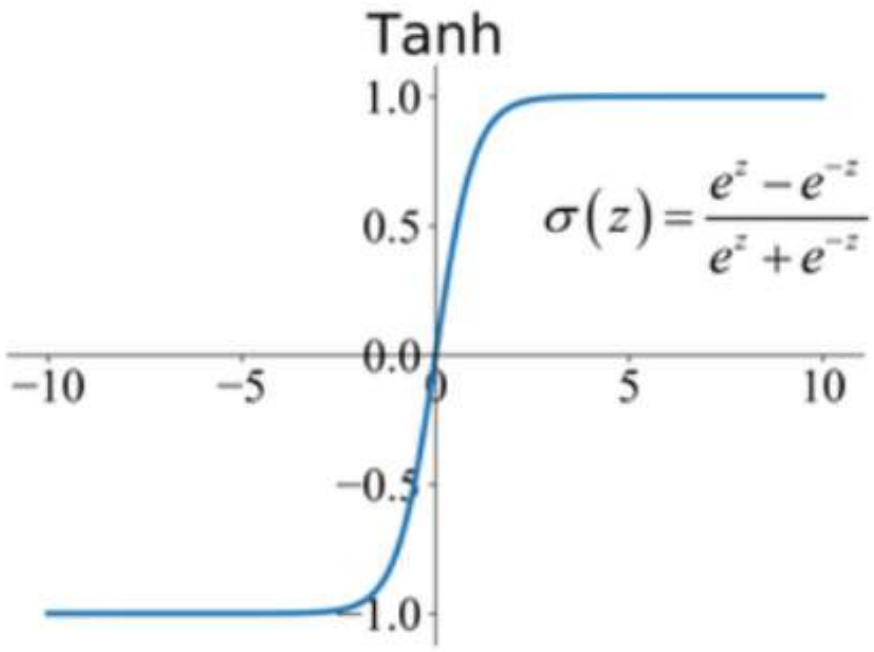

Figure 2.14: Tanh function

effectively with the introduction of ReLU activations.

There are two main reasons for using tanh(x) in neural networks:

1. tanh(x) can limit the value in [-1, 1]

2. tanh(x) can convert a linear function to nonlinear, meanwhile it is derivative.

Rectified Linear Unit(ReLU)

The rectified linear unit (ReLU) activation function was proposed by Nair and Hinton 2010, and ever since, has been the most widely used activation function for deep learning applications with state-of-the-art results to date [19]. The ReLU is a faster learning AF [20], which has proved to be the most successful and widely used function [21].

The rectified linear unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as:

$$f(x) = max(0, x) = \begin{cases} x_i, & if \quad x_i \geq 0 \\ \\ 0, & if \quad x_i < 0 \end{cases} \tag{2.7}$$

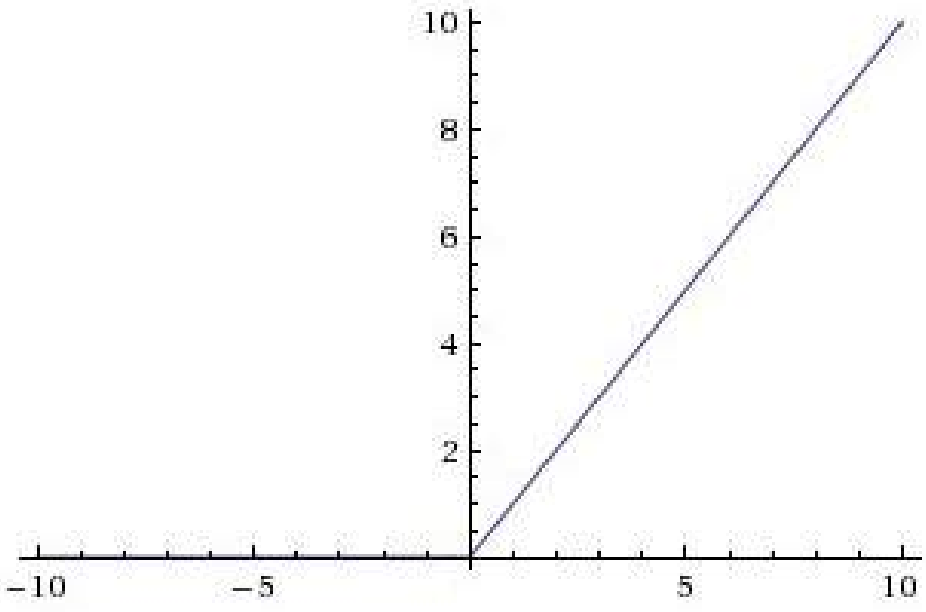

Figure 2.15: Rectified Linear Unit(ReLU) (collected from medium.com).

The gradient value is 0 on the negative side of the graph Fig: 2.15. As a result, weights and biases of some neuron are not updated during the backpropagation process. This can result in neurons that are never stimulated.

Softmax

Softmax function is great for classification problems, especially if you are dealing with multi-class classification problems.
softmax is defined as:

$$\frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_i}} \tag{2.8}$$

It might be daunting to look at first, but it is one of the simpler functions you will encounter while studying deep learning. It states that we need to apply a standard exponential function to each element of the output layer, and then normalize these values by dividing by the sum of all the exponentials. Doing so ensures the sum of all exponentiated values adds up to 1.
Weve prepared a (hopefully) helpful diagram:

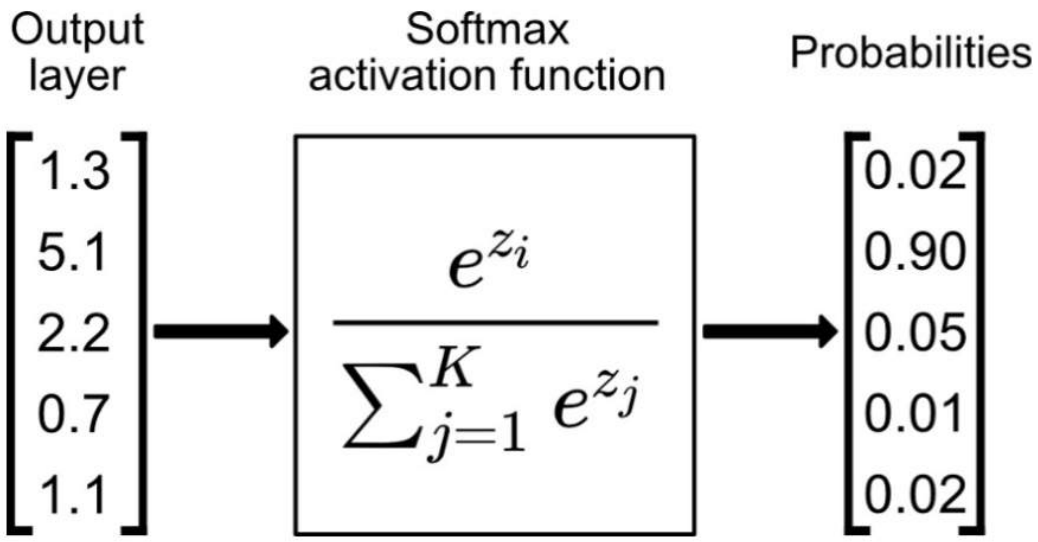

Figure 2.16: Softmax function

Here are the steps:

1. Exponentiate every element of the output layer and sum the results (around 181.73 in this case)

2. Take each element of the output layer, exponentiate it and divide by the sum obtained in step 1 (exp(1.3) / 181.37 = 3.67 / 181.37 = 0.02)

2.4 Loss Function

The goal of all loss functions is measuring how well your algorithm is doing on your dataset. You can keep that complexity in check. Loss functions can be classified into two major categories depending upon the type of learning task we are dealing with regression losses and classification losses. In classification, we are trying to predict output from set of finite categorical values (i.e Given large data set of images of hand written digits, categorizing them into one of 09 digits). Regression, on the other hand, deals with predicting a continuous value for example given floor area, number of rooms, size of rooms, predict the price of room.

Some of regression losses given below:

1. Mean Square Error
2. Mean Absolute Error

Some of Classification losses given below:

1. Multi class SVM Loss
2. Cross Entropy Loss

2.4.1 Mean squared error

Mean square error is measured as the average of squared difference between predictions and actual observations. Its only concerned with the average magnitude of error irrespective of their direction. However, due to squaring, predictions which are far away from actual values are penalized heavily in comparison to less deviated predictions.

$$MSE = \frac{\sum_{i=1}^{n}(y - \hat{y})^2}{n} \tag{2.9}$$

2.4.2 Cross Entropy Loss

This is the most common setting for classification problems. Cross-entropy loss increases as the predicted probability diverges from the actual label.

$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)) \tag{2.10}$$

Notice that when actual label is 1 (y(i) = 1), second half of function disappears whereas in case actual label is 0 (y(i) = 0) first half is dropped off. In short, we are just multiplying the log of the actual predicted probability for the ground truth class. An important aspect of this is that cross entropy loss penalizes heavily the predictions that are confident but wrong.

2.5 Optimizer

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses. How you should change your weights or learning rates of your neural network to reduce the losses is defined by the optimizers you use. Optimization algorithms or strategies are responsible for reducing the losses and to provide the most accurate results possible.

Some of Optimizer given below:

1. Gradient Descent (GD)
2. Stochastic Gradient Descent (SGD)
3. Mini-Batch Gradient Descent
4. Nesterov Accelerated Gradient
5. Adagrad
6. AdaDelta

7. Adam

8. RMSProp

2.5.1 Gradient Descent

The procedure starts off with initial values for the coefficient or coefficients(θ) for the function. These could be 0.0 or a small random value.

$$coefficient = 0.0 \tag{2.11}$$

The cost of the coefficients is evaluated by plugging them into the function and calculating the cost.

$$\begin{aligned} cost &= f(coefficient) \\ cost &= evaluate(f(coefficient)) \end{aligned} \tag{2.12}$$

The derivative of the cost is calculated. The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.

$$delta = derivative(cost) \tag{2.13}$$

Now that we know from the derivative which direction is downhill, we can now update the coefficient values. A learning rate parameter (alpha) must be specified that controls how much the coefficients can change on each update.

$$coefficient = coefficient - (alpha \times delta) \tag{2.14}$$

This process is repeated until the cost of the coefficients (cost) is 0.0 or no further improvements in cost can be achieved.

2.5.2 Momentum

Momentum is often referred to as rolling down a ball, as it is conceptually equal to adding velocity. The weights are modified through a momentum term, which is calculated as the moving average of gradients. The momentum term γ can be seen as air resistance or friction which decays the momentum proportionally. Momentum accelerates the training process but adds an additional hyperparameter.

$$V_i = \gamma V_i + \alpha \frac{\delta L}{\delta \theta_i} \tag{2.15}$$

where θ_i is the coefficient, α is a learning rate.

2.6 Back-Propagation

The Backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem. Backpropagation try to get the value of weight such that the error becomes minimum. Basically, we need to figure out whether we need to increase or decrease the weight value. Once we know that, we keep on updating the weight value in that direction until error becomes minimum. You might reach a point, where if you further update the weight, the error will increase. At that time you need to stop, and that is your final weight value.

Below are the steps involved in Backpropagation:

1. Forward Propagation
2. Backward Propagation
3. Putting all the values together and calculating the updated weight value

Before backpropagation can be done on a neural network, the regular/forward training pass of a neural network must be carried out. When a neural network is created, a set of weights is initialized. The value of the weights will be altered as the network is trained. When training a deep neural network, we need to

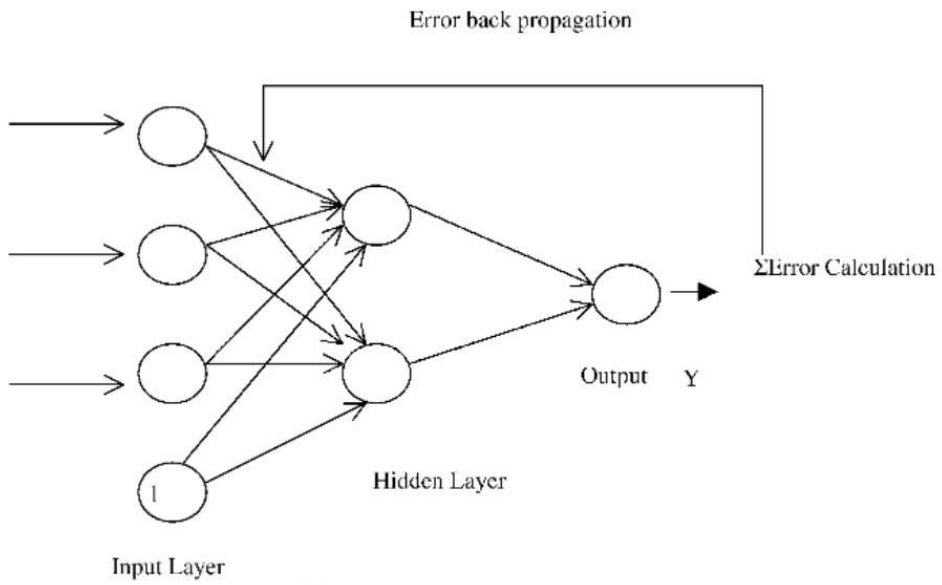

Figure 2.17: Backpropagation in neural network

make use of multiple mathematical functions. Neurons in a deep neural network are comprised of the incoming data and an activation function, which determines the value necessary to activate the node. The activation value of a neuron is calculated with several components, being a weighted sum of the inputs. The weights and input values depend on the index of the nodes being used to calculate the activation. Another number must be taken into account when calculating the activation value, a bias value. Bias values dont fluctuate, so they arent multiplied together with the weight and inputs, they are just added. All of this means that the following equation could be used to calculate the activation value:

$$Activation = sum(weight * input) + bias \tag{2.16}$$

After the neuron is activated, an activation function is used to determine what the output of the actual output of the neuron will be. Once the outputs of the neuron are calculated by running the activation value through the desired activation function, forward propagation is done. Forward propagation is just taking the outputs of one layer and making them the inputs of the next layer. The new inputs are then used to calculate the new activation functions, and the output of this operation passed on to the following layer. This process continues all the way through to the end of the neural network.

The process of backpropagation takes in the final decisions of a models training pass, and then it determines the errors in these decisions. The errors are calculated by contrasting the outputs/decisions of the network and the expected/desired outputs of the network. Once the errors in the networks decisions have been calculated, this information is backpropagated through the network and the parameters of the network are altered along the way. The method that is used to update the weights of the network is based in calculus, specifically, its based in the chain-rule. When doing backpropagation, the error for a specific neuron is calculated. After the errors for the network have been calculated, the weights in the network must be updated. As mentioned, calculating the error involves determining the slope of the output value. After the slope has been calculated, a process known as gradient descent can be used to adjust the weights in the network. Gradient descent is the process of updating the weights so that the error rate decreases. Backpropagation is used to predict the relationship between the neural networks parameters and the error rate, which sets up the network for gradient descent. Training a network with gradient descent involved calculating the weights through forward propagation, backpropagating the error, and then updating the weights of the network.

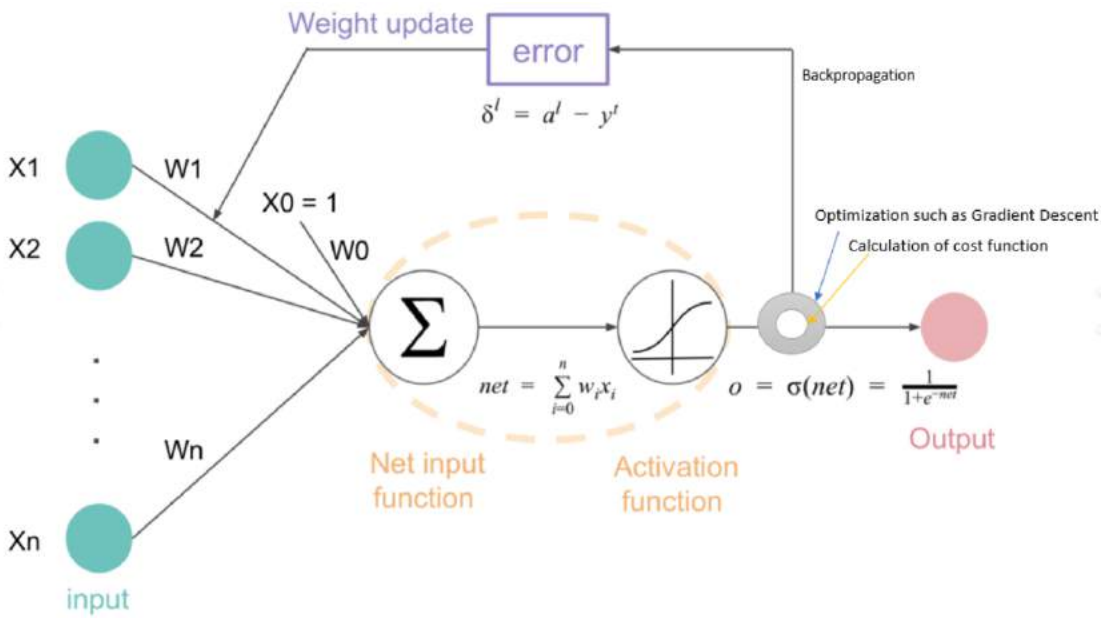

Figure 2.18: Weight updatation process in backpropagation

2.7 Image Frequency Domain

In the frequency domain, a digital image is converted from spatial domain to frequency domain which is used to decompose an image into its sine and cosine components. A Fast Fourier transformation is a tool of the frequency domain used to convert the spatial domain to the frequency domain. Frequency domain analysis is used to indicate how signal energy can be distributed in a range of frequency. The Discrete Fourier Transform(DFT) is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image. The number of frequencies corresponds to the number of pixels in the spatial domain image, i.e. the image in the spatial and Fourier domain are of the same size. For a square image of size NŒN, the two-dimensional DFT is given by:

$$F(k,l) = (\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j)e^{-i2\pi(\frac{ki}{N}+\frac{lj}{N})}) \tag{2.17}$$

where $f(i,j)$ is the image in the spatial domain and the exponential term is the basis function corresponding to each point $F(k,l)$ in the Fourier space. The equation can be interpreted as: the value of each point $F(k,l)$ is obtained by

multiplying the spatial image with the corresponding base function and summing the result.

The interpretation of spectra is made much easier if the results of the DFT are centred on the point (k = 0, l = 0), such that frequency increases in any direction away from the origin. This can be done by circular shifting of the four quadrants of the array or computing the DFT sums from $(-\frac{N}{2})$ to $(\frac{N}{2})$ rather than from 0 to N. Alternatively, by the shift theorem of the Fourier transform . Typically, all the spectra are represented with the centre point as the origin to see and analyse dominant image frequencies. Because the lower frequency amplitudes mostly dominate over the mid-range and high-frequency ones, the fine structure of the amplitude spectrum can be perceived only after a non-linear mapping to the gray scale range [0, 255]. The examples below show amplitude spectra of the image:

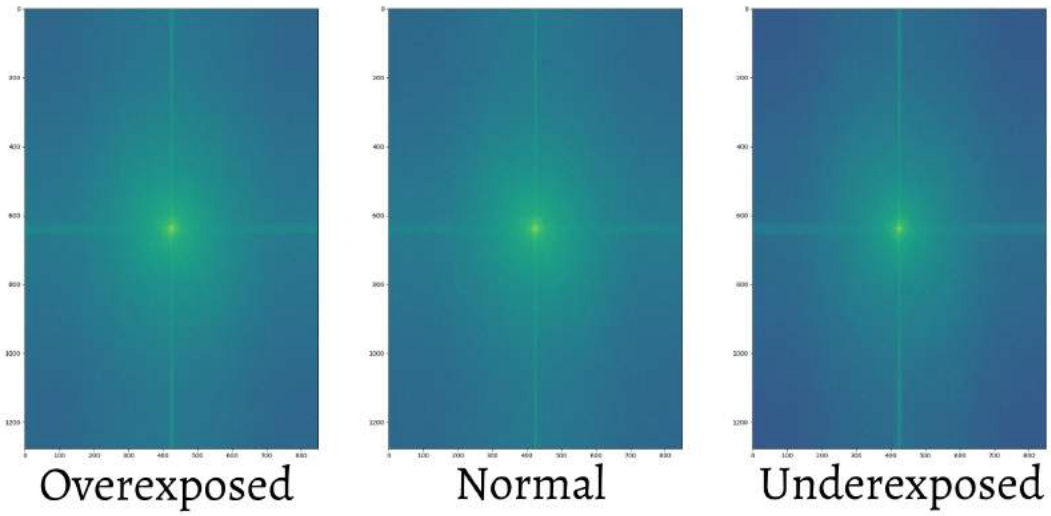

Figure 2.19: Frequency Domain Image of Fig 2.20

2.8 Image Histogram

An image histogram is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each tonal value. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance. Before discussing image histogram, we will first look at what histogram is, A histogram is a graph. A graph

Figure 2.20: Original Image of Fig 2.19

that shows frequency of anything. Usually histogram have bars that represent frequency of occurring of data in the whole data set.
A Histogram has two axis the x axis and the y axis.
The x axis contains event whose frequency you have to count.
The y axis contains frequency.
The different heights of bar shows different frequency of occurrence of data.

Histogram of an image, like other histograms also shows frequency. But an image histogram, shows frequency of pixels intensity values. In an image histogram, the x axis shows the gray level intensities and the y axis shows the frequency of these intensities.

Figure 2.21: An Image (collected from [4]

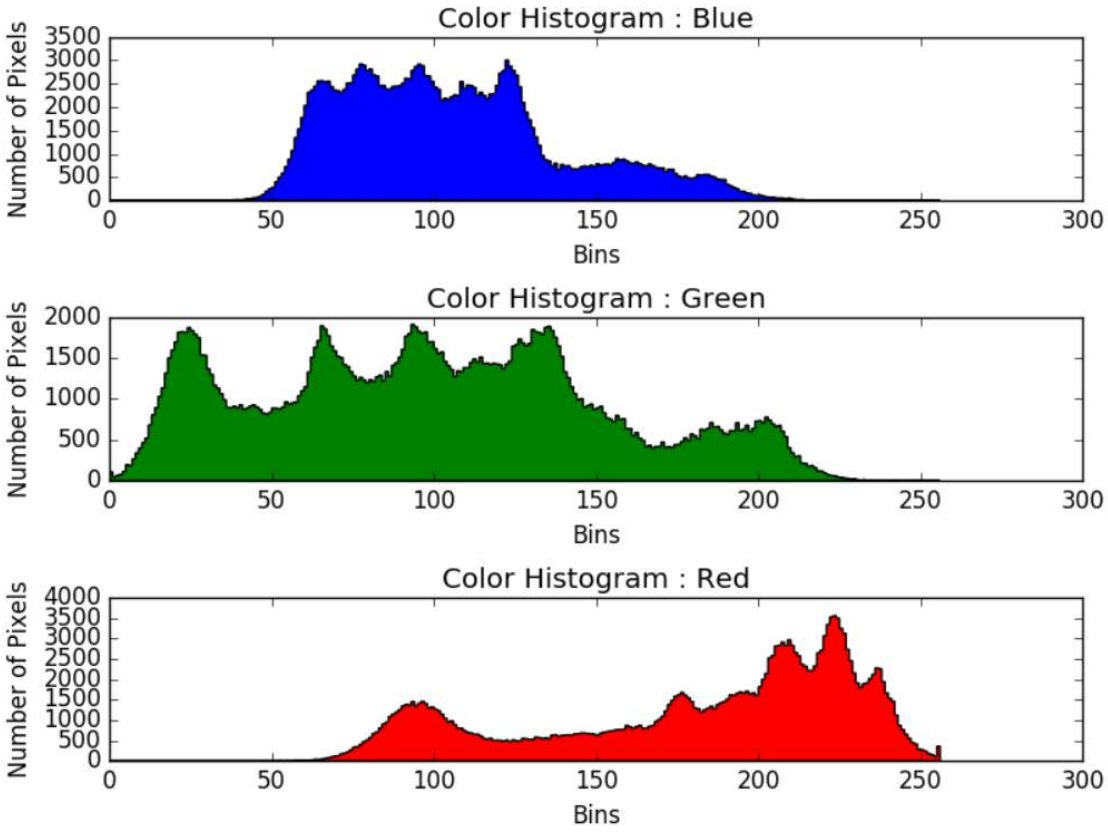

Figure 2.22: Histogram of Fig 2.21 (collected from [4]

2.9 Negative Image

Color Inversion (Negative Image) is the method of inverting pixel values of an image. Image inversion does not depend on the color mode of the image, i.e. inversion works on channel level. When inversion is used on a multi color image (RGB, CMYK etc) then each channel is treated separately, and the final result if formed by callibrating the results of all the channels.

The method used for getting the inverse of an image is subtraction of the maximum value/intensity of a pixel by the value of current pixel. The resultant value is guided by the formula:

$$INV = I_{MAX} - I(x, y) \tag{2.18}$$

Where INV is the resultant inverted pixel, I_{MAX} is the maximum intensity level in a given color mode and $I(x, y)$ is the intensity (pixel value) of image/color channel at a particular pair of coordinates.

Figure 2.23: Original Image Before Negation

Figure 2.24: Negative Image of Fig 2.23

2.10 Single Channel Images (i.e., Red, Green, Blue)

A typical color image consists of three color channels: red, green and blue. Each color channel has 8 bits and can represent 256 distinct values. Using a combination of all three channels, we can create 256 x 256 x 256 colors, which is around 16-million colors.

Figure 2.25: We create red, green, blue channel of this image, Figure 2.26 show image of these channel.

Figure 2.26: Red, Green, Blue Channel of Fig 2.25

2.11 Skewness

In statistics, skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable. The skewness value can be positive or negative, or even undefined. Qualitatively, a negative skew indicates that the tail on the left side of the probability density function is longer than the right side and the bulk of the values (possibly including the median) lie to the right of the mean. A positive skew indicates that the tail on the right side is longer than the left side and the bulk of the values lie to the left of the mean. A zero value indicates that the values are relatively evenly distributed on both sides of the mean, typically but not necessarily implying a symmetric distribution. Mathematically skewness can be given by

$$f(x,y) = \frac{\frac{1}{mn-1}\sum_{(r,c)\in W}\left(\frac{1}{mn-1}\left(\sum_{(r,c)\in W}\left(g(r,c) - \frac{1}{mn-1}\sum_{(r,c)\in W} g(r,c)\right)\right)\right)^3}{\left(\frac{1}{mn-1}\sum_{(r,c)\in W}\left(\frac{1}{mn-1}\left(\sum_{(r,c)\in W}\left(g(r,c) - \frac{1}{mn-1}\sum_{(r,c)\in W} g(r,c)\right)\right)\right)^2\right)^{\frac{3}{2}}} \tag{2.19}$$

In terms of digital image processing, Darker and glossier surfaces tend to be more positively skewed than lighter and matte surfaces. Hence we can use skewness in making judgements about image surfaces

2.12 Entropy

The purpose of an image is to convey information; thus information theory can play an important role in image analysis. Specifically, we use the notion of entropy to measure the amount of image information. image entropy is used in the context of image coding. Information theory provides a theoretical foundation to quantify the information content, or the uncertainty, of a random variable represented as a distribution. Formally, let X be a discrete random variable with alphabet X and probability mass function p(x), x X . The Shannon entropy of X is defined as

$$H(X) = -\sum_{x \in X} p(x) log p(x) \tag{2.20}$$

where $p(x) \in [0.0, 1.0], \sum_{x \in X p(x)=1.0, and - log p(x)}$ represents the information associated with a single occurrence of x. The unit of information is called a bit. Normally, the logarithm is taken in base 2 or e. For continuity, zero probability does not contribute to the entropy, i.e., 0 log 0 = 0. As a measure of the average uncertainty in X, the entropy is always nonnegative, and indicates the number of bits on average required to describe the random variable. The higher the entropy, the more information the variable contains. An important property of the entropy is that H(X) is a concave function and reaches its maximum of log(|X|) if and only if p(x) is equal for all x, i.e., when the probability distribution is uniform. As we shall see in Section 4, the notion of equal probability, maximum entropy is at the heart of probability function design in many of the visualization examples we will review. The key of applying the concept of entropy to visualization problems lies in how to properly specify the random variable X and define the probability function p(x). In most cases, these probability functions can be defined heuristically to meet the need of individual applications.

2.13 Image Brightness Enhancement

To adjust the brightness of an image, we change the value of all pixels by a constant. Adding a positive constant to all of the image pixel values makes the image brighter. Similarly, we can subtract a positive constant from all of the pixel values to make the image darker.

Chapter 3

Works on Inappropriate Exposure

3.1 Previous Works on Exposure

The focus of our paper is on the detection of exposure errors in camera-rendered 8-bit sRGB images. On our research topic, we don't get any previous work, instead we get works on image quality measure procedure and how to correct the inappropriate image exposure. Although both over- and underexposure errors are common in photography, most prior work is mainly focused on correcting underexposure errors [22] [23] [24] [25] [26] [27] or generic image quality enhancement [28] [29]. There are very few work on both over-exposed and under-exposed image quality enhancement [30]. Traditional methods for exposure correction and contrast enhancement rely on image histograms to re-balance image intensity values [31] [32] [33] [34]. Alternatively, tone curve adjustment is used to correct images with exposure errors. This process is performed by relying either solely on input image information [35] or trained deep learning model [36] [37] [38]. The majority of prior work adopts the retinex theory [39]by assuming that improperly exposed images can be formulated as a pixel-wise multiplication of target images, captured with correct exposure settings, by illumination maps. Thus, the goal of these methods is to predict illumination maps to recover the well-exposed target images. In contrast to the majority of prior work, our work is the first deep learning method to explicitly detect both overexposed and underexposed photographs with a single model.

The quality of the image can be controlled to a large extent using the camera

focus and camera exposure parameters. In this [40] paper, a statistical measure of image quality was developed and tested, that allows for the simultaneous optimization of focus and exposure. A set of basic test patterns was designed for the experiments. Statistical measures can be extracted from a digital image to quantify the image quality. A number of typical measures used in the literature were computed from the image gray level value histograms, namely:

- Mean
$$\mu = \sum_{i=0}^{N-1} ip(i) \tag{3.1}$$
- Standard Deviation
- Entropy
- Percentage of pixels in the gray level range {50,250}, and
- Absolute Central Moment (ACM). The ACM is computed using the following equation:
$$ACM = \sum_{i=0}^{N-1} |i - \mu|p(i) \tag{3.2}$$

Standard Deviation and the ACM measure have the best performance, with the ACM being slightly more discriminatory. Both the measures peak for the best image quality. The ACM has the advantage of being more symmetric about the centerline and computationally simpler than the Standard Deviation measure. The measure gracefully degrades in value for any decrease in focus or exposure in the image. Experimental results revealed that ACM is an excellent measure of image quality. It peaks in value when the image has the best exposure and sharpness.

3.2 Our Work on Exposure

For detecting inappropriate exposure in image, we classify image in three category (overexposed, underexposed and normal). We use supervised deep learning technique for classification. We use total five model to classify image. Two of them

are used in transfer learning technique. In transfer learning we use VGG16 a pre-trained CNN model to train our dataset. Another three model are our created model, first one of them is Fully Connected Model (all layers are fully connected in this model), second and third one are Convolutional Neural Network(CNN) named as CNN-1 and CNN-2.

To get good performance we explore some different techniques. We perform different pre-processing method on our dataset. We transform our main dataset into red channel image, blue channel image, green channel image, gray scale histogram image, inverted image, increased and decreased exposure of image. Then we find-out the accuracy and select the best technique and best model. Also we estimate skewness of all red, green, blue and gray channel image. Also we try to find relationship between skewness and exposure.

Chapter 4

Experimental Setup

4.1 Hardware & Software Toolbox

4.1.1 Hardware

- GPU - NVIDIA GeForce GTX 1070
- CPU - Intel(R) Core(TM) i7-7700 CPU @ 360GHZ

4.1.2 Software

- Operating System - Ubuntu 20.04
- Visual Studio Code
- Jupyter Notebook
- Python 3.7.4 environment.
  - TensorFlow (Free and open-source Python library), version 2.8.0.
  - Keras (Deep learning library for TensorFlow).
  - OpenCV (Open Source Computer Vision Library), version 4.5.5.
  - NumPy(Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.), version 1.22.3.

- Scikit-learn (a Python module for machine learning built on top of SciPy)
- Matplotlib (A comprehensive library for creating static, animated, and interactive visualizations in Python)
- PIL (Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.)

4.2 Data Set

To do our work, we need a large number of images rendered with realistic over- and under-exposure errors and corresponding properly exposed ground truth images. However, such data sets are currently not publicly available. We have found that the MIT-Adobe FiveK data set [41] is the closest publicly available data set which can full fill the requirements of our work. In this data set, there are 5,000 photographs captured with SLR cameras by a set of different photographers. All the photographs are in RAW format. That is, all the information recorded by the camera sensor is preserved. It is made sure that these photographs cover a broad range of scenes, subjects, and lighting conditions. Five photography students in an art school were hired to adjust the tone of the photos. They retouched each of the photo by hand. Each of them retouched all the 5,000 photos using a software (named Adobe Lightroom) dedicated to photo adjustment on which they were extensively trained. They rendered each raw-RGB image with different digital EVs (Exposure Values) to mimic real exposure errors. Specifically, they used the relative **EVs 1.5, 1, +0, +1, and +1.5 to render images with underexposure errors, a zero gain of the original EV, and overexposure errors**, respectively. See Fig. 4.1 for an example. The zero-gain relative EV is equivalent to the original exposure settings applied on board the camera during capture time.

They expressed adjustments as a remapping curve from input luminance into output luminance, using the CIE-Lab color space because it is reasonably perceptually uniform. The curve is represented by a spline with 51 uniformly sampled

Figure 4.1: Image exposure value 1.5, 1, +0, +1, and +1.5 to render images with underexposure errors, a zero gain of the original EV, and overexposure errors.

control points. They fit the spline to the pairs of input-output luminance values in a least-squares sense. Different factors such as the type of camera used for capturing a photo, the skill of the particular photographer, different camera metering systems, or a users manual settings, might result in different exposures for a given scene. This is why they normalized the exposure to the same baseline by linearly remapping the luminance values of each image so that the minimum is 0 and the maximum 100. They focused on learning the first PCA coefficient of the remapping curves, which is a good approximation to the full curve.

4.2.1 Preparing Training, Validation and Test Sets

Among 25000 (i.e., 5 × 5000) images of the MIT-Adobe FiveK dataset, we have taken 8,9854 8-bit sRGB images. We have considered images having EVs -1.5 as under-exposed images, images having EVs +0 as normal images, images having EVs +1.5 as over-exposed images. We have divided these images into three sets:

1. Training set: 85% images of the total images.

2. Validation set: 20% images of the training set.

3. Testing set: 15% images of the total images.

The training, validation, and testing sets are non-overlapped. This means the training, validation, and testing images do not have any images in common. Images in the MIT-Adobe FiveK data set are in variable size. We have resized our selected images to 32Œ32 images for reducing training and testing time.

Class	Train	Test	Total
Normal	2580	421	3,001
Over Exposed	2552	452	3,004
Under Exposed	2504	475	2,979
Total Images	7636	1348	8,984

Table 4.1: Number of images in the training, validation and test sets.

Note that in our project proposal, we mentioned to use the ImageNet data set. The ImageNet (available at https://image-net.org/) is a large visual database designed for object recognition research. There are 14,197,122 images, 21841 synsets indexed in this data set. The data of this data set is available for free to researchers for non-commercial use. Even though the ImageNet dominates computer vision research field, it does not have any information about exposure levels of images. Moreover, in the ImageNet we do not get enough number of over-exposed and under-exposed image because most of the image in the ImageNet are properly exposed image. Therefore, we had to annotate images and artificially change exposure value (EV) to get over-exposed and under-exposed images by ourselves if we wanted to use this data set for our work. We had to avoid this time consuming process because of the short duration of the project time.

4.2.2 Different Data Processing Techniques

To obtain a good accuracy, we have explored following techniques on our data set:

1. Images Frequency Domain
2. Single Channel Images (i.e., Red, Green, Blue)
3. Negative Images
4. Histograms of Gray Scale Images
5. Images with Increased and Decreased Exposure

Images in Frequency Domain

We have applied the First Fourier Transform (FFT) on all of our images. After applying the FFT, we have got Frequency domain output. We have shifted the

zero-frequency component to the center of the spectrum. Then we have applied natural logarithm on the frequency domain output. As shown in Fig 2.19, we can see two very clear distortions. The white vertical and horizontal lines refer to the sharp horizontal and vertical elements of the image. We have tried to find out pattern on those images.

Single Channel Images (i.e., Red, Green, Blue)

An RGB image is composed of three true color channel images. If the RGB image is 24-bit, each channel (i.e., red, green, and blue) has 8 bits. Each image can store discrete pixels with conventional brightness intensities between 0 and 255. Each color channel reacts to exposure in different level. For example, the *red* channel is, in general, affected by overexposure while the *blue* channel is affected by the underexposure. The green, in general, is not affected by the inappropriate exposure. We have separated our RGB images into red channel images, green channel images, blue channel images, to see how much this information is correct.

Negative Images

Color Inversion (Inverted Image) is the method of inverting pixel values of an image. Image inversion does not depend on the color mode of the image, i.e. inversion works on channel level. When inversion is used on a multi color image (RGB, CMYK etc) then each channel is treated separately, and the final result if formed by callibrating the results of all the channels. In this technique we first invert all of our dataset images and then apply CNN on the negated datasets.

Histograms of Gray Scale Images

A histogram plays an important role in exposure detection. Because it represents the number of pixels for each tonal value. If our image is overexposed then the histogram will be biased to the right side.On the contrary, if our image is underexposed then the histogram will be biased to the left side. In the histogram of a properly exposed image has almost equally distributed pixels. For that, we have created image histograms in gray scale and built a data set of the histogram.

Images with Increased and Decreased Exposure

PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities. The ImageEnhance of this library contains a number of classes that can be used for image enhancement. This class can be used to control the brightness of an image. An enhancement factor of 0.0 gives a black image. A factor of 1.0 gives the original image. An enhancement factor more than 1.0 value factor gives a more bright image. We have applied 1.18 factor effect on overexposed image and 0.82 factor on underexposed image. After applying factor effect, overexposed images became more overexposed as shown in Fig. 4.2 and underexposed images became more underexposed as shown in Fig. 4.3.

Figure 4.2: "Previous overexposed" is an overexposed image of dataset and after applying image brightness enhancement factor we get "Our overexposed".

4.3 Architecture of Our Deep Neural Networks

4.3.1 Architecture of VGG-16 Based Model

We have designed two CNN based classifiers, consisting of one input layer (i.e., 32Œ32 RGB images), 18 VGG-16 layers, three fully-connected layers as shown in Table 4.3.1. The VGG-16 layers are comprised of a series of convolutional and max-pooling layers with a total number of 14,714,688 pre-trained weights. In the convolutional layers, convolution is performed by using a 3×3 filter with a stride of

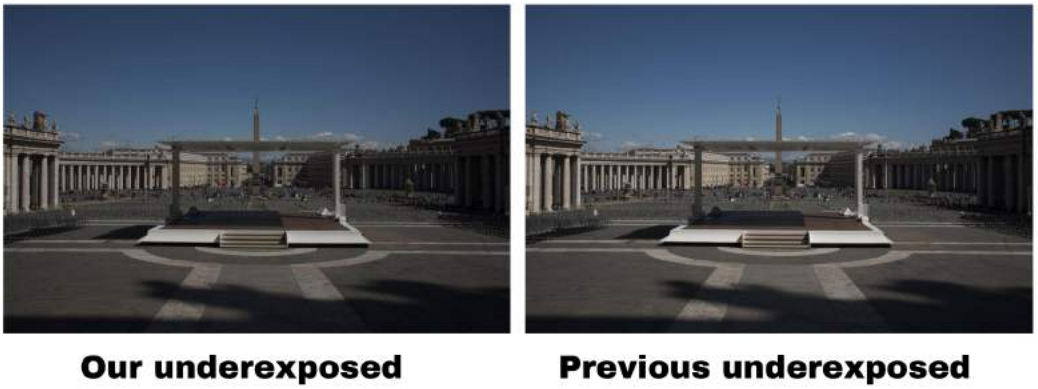

Figure 4.3: "Previous underexposed" is an underexposed image of data set and after applying image brightness enhancement factor we get "Our underexposed".

one pixel that preserves the size of the image. However, in the pooling layers, max-pooling is performed using a 2×2 filter with a stride of two pixels that reduces the size of the image by half in each direction. The outputs of the last VGG-16 layer are connected to a flattened layer consisting of 512 nodes. Next layer is a fully connected layer consisting of 256 nodes. In the next layer, a dropout operation is performed with 30% probability, i.e., during each iteration of the training session, 30% of the nodes are randomly excluded from weight updating. Next layer is a fully connected layer consisting of 64 nodes. This hidden-layer is connected to the output layer consisting of 3 nodes which yields a vector represented as "one-hot encoding". In this encoding, each element of the vector represents one class and can have a value of either one (i.e., the input image belongs to that class) or 0 (i.e., the input image does not belong to that class).

Since our task (i.e., exposure detection) is different from the task (image classification) the VGG-16 was trained for, we should go for fine-tuning. Fine-tuning means taking weights of a trained NN and using them as initialization for a new model being trained on data from the same domain (often e.g. images). It allows the previously frozen layers to adapt to the new data set (i.e., target data) without drastically changing their weights. It is used to speed up the training and overcome the limitations of a small data set. There are mainly two strategies: (1) training the whole initialized network and (2) "freezing" some of the pre-trained weights (usually whole layers). We have followed both strategies.

1. Fully fine-tuned VGG-16 Model
    - Total params: 14,862,659
    - Trainable params: 14,862,659
    - Non-trainable params: 0
2. Partially fine-tuned VGG-16 Model
    - Total params: 14,862,659
    - Trainable params: 147,971
    - Non-trainable params: 14,714,688

Layer (type)	Output Shape	Parameter
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 256)	131328
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 64)	16448
dense_2 (Dense)	(None, 3)	195

Table 4.2: Architecture of VGG-16 Model

4.3.2 Architecture of Our Own Deep Neural Network Models

We have created three deep neural networks:

1. Fully Connected Neural Network (FCNN)

2. Convolutional Neural Network-1 (CNN-1)

3. Convolutional Neural Network-2 (CNN-2)

Layer (type)	**Output Shape**	**Parameter**
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
flatten_1 (Flatten)	(None, 3072)	0
dense_3 (Dense)	(None, 1024)	3146752
dense_4 (Dense)	(None, 512)	524800
dense_5 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 128)	32896
dense_7 (Dense)	(None, 64)	8256
dense_8 (Dense)	(None, 32)	2080
dense_9 (Dense)	(None, 8)	264
dense_10 (Dense)	(None, 3)	27
Total params: 3,846,403		
Trainable params: 3,846,403		
Non-trainable params: 0		

Table 4.3: Architecture of our FCNN

Architecture of our CNN-1

Layer (type)	**Output Shape**	**Parameter**
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
conv2d (Conv2D)	None, 32, 32, 3)	39
max_pooling2d (MaxPooling2D)	(None, 16, 16, 3)	0
conv2d_1 (Conv2D)	(None, 16, 16, 3)	84
max-pooling2d_1 (MaxPooling2D)	(None, 8, 8, 3)	0
flatten (Flatten)	(None, 192)	0
dense (Dense)	(None, 256)	49408
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 32)	8224
dense_2 (Dense)	(None, 3)	99
Total params: 57,854		
Trainable params: 57,854		
Non-trainable params: 0		

Table 4.4: Architecture of our CNN-1

Architecture of our CNN-2

Layer (type)	Output Shape	Parameter
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
conv2d (Conv2D)	(None, 32, 32, 64)	1792
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2097408
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 32)	8224
dense_2 (Dense)	(None, 3)	99
Total params: 2,365,891		
Trainable params: 2,365,891		
Non-trainable params: 0		

Table 4.5: Architecture of our CNN-2

4.4 Training Models

We have used the training set to update parameters (i.e., weights and biases) of the NNs by the backpropagation algorithm and the validation set to tune hyper-parameters such as the number of epochs. We have used the stochastic gradient descent (SGD) optimization algorithm with a slow-learning rate, α. The hyper-parameter, α, is empirically set to $\alpha = 10^{-3}$. We have used *relu* activation function in our convolution layer, *sigmoid* activation function used in the fully connected

layers and *softmax* activation function used on the output layer.

4.5 Evaluation Metrics

To test the performance of the model, unseen testing data are fed to the trained model. The performance of the classifier model is evaluated using some well-established performance metrics such as accuracy, precision, and recall, F1 Score , as shown in Equation 4.1 through 4.4.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1Score = \frac{2(Recall * Precision)}{Recall + Precision} \tag{4.4}$$

The performance matrix can be evaluated by estimating the predicted image among four subsets: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). Here, TP, TN, FP, and FN refer to true positive (correctly classified to the class), true negative (correctly classified to other class), false positive (incorrectly classified to the class), and false negative (incorrectly classified to other class), respectively.

Macro Average

The macro-average of F1 score, Precision and Recall is computed by taking the arithmetic mean of each class F1 scores, precision and recall.

Weighted Average

To estimate the weighted average, at first, we need to calculate unbalanced data as shown in the Table **??**. In the test-set, total number of images is 1348. Among the 1348 images, the number of the normal images is 421, the overexposed images is 452 and the underexposed images is 475. The ratio of the normal images is 0.3123,

the ratio of the overexposed images is 0.3353 and the ratio of the underexposed images is 0.3523. The weighted-averaged F1 score is calculated as:

$$Score_{weighted-averaged} = .3123 \times F1Score_{normal} + 0.3353 \times F1Score_{overexpsoed} + 0.3523 \times F1Score_{underexposed} \tag{4.5}$$

Similarly we have calculates the weighted average of precision and recall.

Confusion Matrix

A confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known. It provides insights into the true and false positives and negatives of a model. The goal is to maximize the values of the diagonal of the matrix which encompass the true positives and true negatives, while minimizing the values that are off the diagonal that encompass the false positives and false negatives.

Chapter 5

Results and Discussion

Each of our deep feed forward neural networks takes an image as input, generates intermediate features through a series of convolution and max-pooling operations, passes the features to the fully-connected layer, and outputs the probabilities of the image belonging to each class. The intermediate features for a randomly selected image is labeled as either an overexposed image, underexposed image or a normal image. The results are demonstrated in the following sections.

5.1 RGB image

5.1.1 Partially Fine-tuned VGG-16 model

Figure 5.1 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.2 illustrates the training and validation accuracy of the model. This model doesn't perform well. Validation accuracy is increasing but training accuracy does not increasing as validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.1.

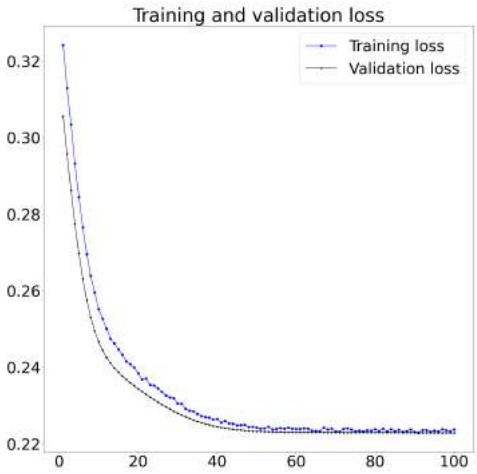

Figure 5.1: Training and Validation Loss

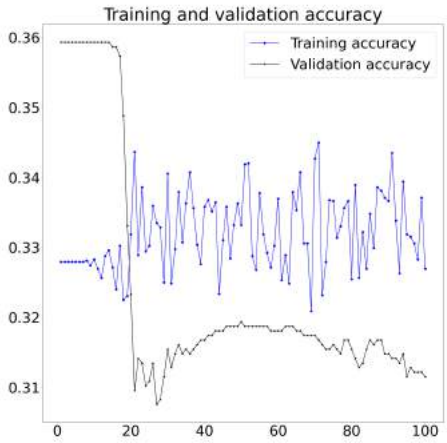

Figure 5.2: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	31%	88%	46%
Over Expsoed	36%	01%	02%
Under Expsoed	31%	09%	13%
Accuracy			31%
Macro Avg	32%	33%	20%
Weighted Avg	32%	31%	20%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	371	6	44
Over Expsoed	399	5	48
Under Expsoed	431	3	41

Table 5.1: Performance metrics and confusion matrix

5.1.2 Fully Fine-tuned VGG-16 model

Figure 5.3 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.4 illustrates

the training and validation accuracy of the model. This model perform quite well. Both validation accuracy and training accuracy is increasing. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.2.

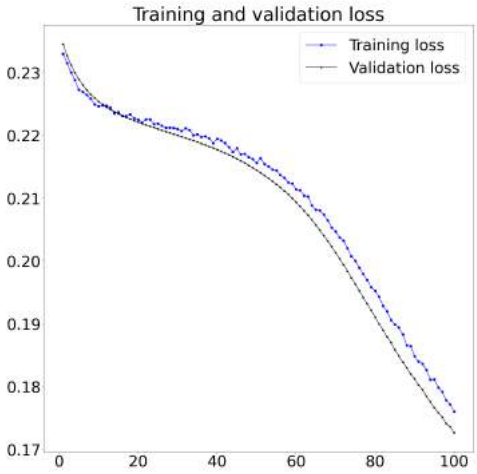

Figure 5.3: Training and Validation Loss

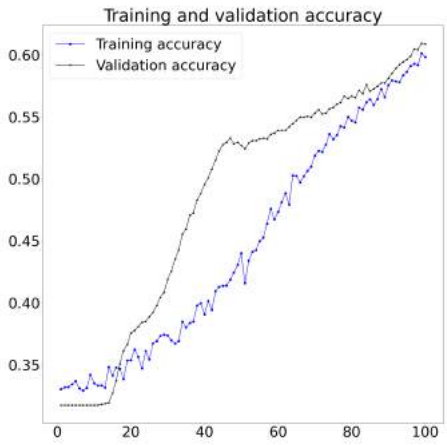

Figure 5.4: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	61%	64%	62%
Over Expsoed	77%	85%	81%
Under Expsoed	87%	76%	81%
Accuracy			75%
Macro Avg	75%	75%	75%
Weighted Avg	76%	75%	75%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	268	107	46
Over Expsoed	61	382	9
Under Expsoed	111	4	360

Table 5.2: Performance metrics and confusion matrix

5.1.3 FCNN model

Figure 5.5 illustrates the training loss and validation loss of the model. The training loss of this model decreasing with time. At first validation loss are decreasing, later validation loss does not decrease. Figure 5.6 illustrates the training and validation accuracy of the model.Training accuracy is increasing in this model.

Although at first validation accuracy increasing later it does not increase as training accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.3.

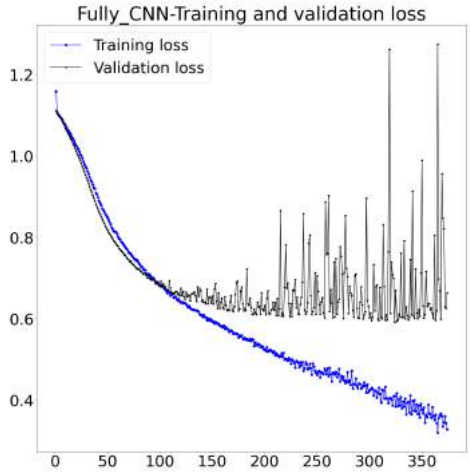

Figure 5.5: Training and Validation Loss

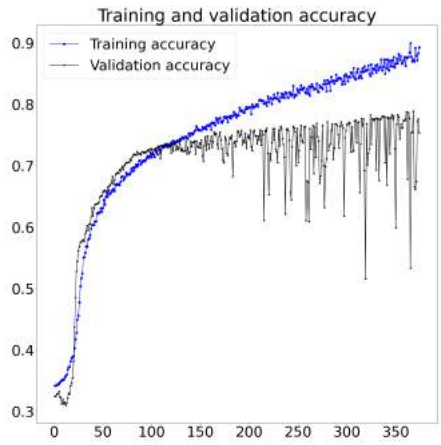

Figure 5.6: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	71%	76%	73%
Over Expsoed	92%	75%	82%
Under Expsoed	83%	93%	87%
Accuracy			81%
Macro Avg	82%	81%	81%
Weighted Avg	82%	81%	81%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	319	28	74
Over Expsoed	96	337	19
Under Expsoed	34	1	440

Table 5.3: Performance metrics and confusion matrix

5.1.4 CNN-1 model

Figure 5.7 illustrates the training loss and validation loss of the model. Figure 5.8 illustrates the training and validation accuracy of the model. Both validation accuracy and training accuracy are increasing at a level but later they do not increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.4.

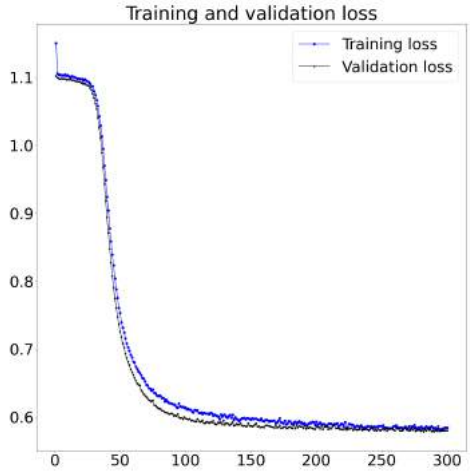

Figure 5.7: Training and Validation Loss

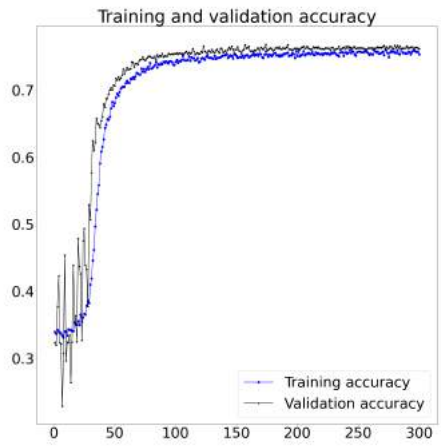

Figure 5.8: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	67%	68%	68%
Over Expsoed	84%	80%	82%
Under Expsoed	83%	86%	84%
Accuracy			78%
Macro Avg	78%	78%	78%
Weighted Avg	78%	78%	78%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	288	67	66
Over Expsoed	75	361	16
Under Expsoed	67	1	407

Table 5.4: Performance metrics and confusion matrix

5.1.5 CNN-2 model

Figure 5.9 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.10 illustrates the training and validation accuracy of the model. Both training accuracy and validation accuracy are increasing and validation accuracy is slightly lower than training accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.5.

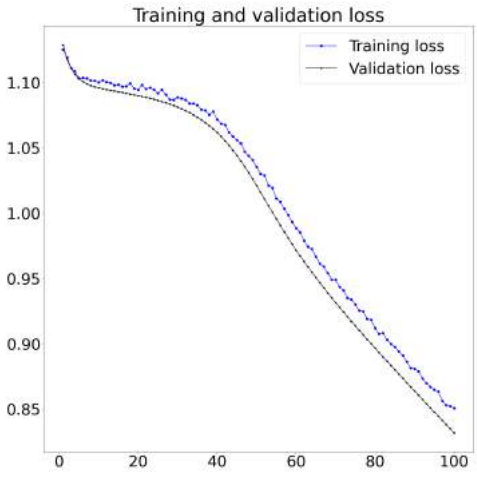

Figure 5.9: Training and Validation Loss

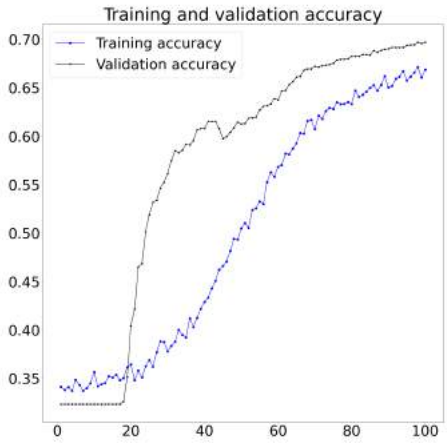

Figure 5.10: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	65%	61%	63%
Over Expsoed	79%	83%	81%
Under Expsoed	83%	83%	83%
Accuracy			76%
Macro Avg	76%	76%	76%
Weighted Avg	76%	76%	76%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	258	96	67
Over Expsoed	61	377	14
Under Expsoed	76	4	395

Table 5.5: Performance metrics and confusion matrix

5.2 Single Channel Images (Red)

5.2.1 Partially fine-tuned VGG-16 model

Figure 5.11 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well at first, later losses do not decrease with epochs. The Figure 5.12 illustrates the training and validation accuracy of the model. This model does not perform well. Validation accuracy is increasing very slowly and increase in training accuracy is more less than validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.6.

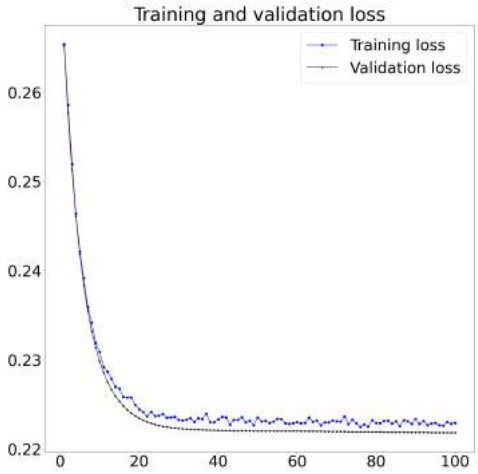

Figure 5.11: Training and Validation Loss

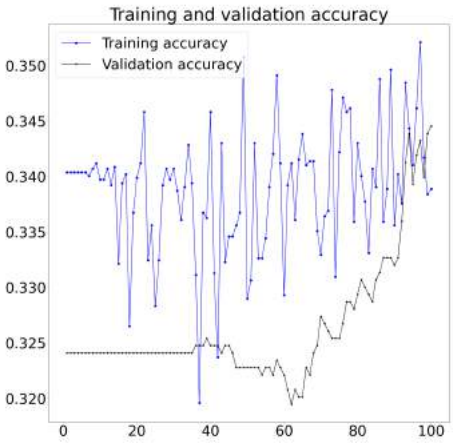

Figure 5.12: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	32%	88%	47%
Over Expsoed	00%	00%	00%
Under Expsoed	51%	20%	29%
Accuracy			35%
Macro Avg	28%	36%	25%
Weighted Avg	28%	35%	25%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	367	1	49
Over Expsoed	403	0	45
Under Expsoed	376	0	96

Table 5.6: Performance metrics and confusion matrix

5.2.2 Fully Fine-tuned VGG-16 model

Figure 5.13 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.14 illustrates the training and validation accuracy of the model. Both training accuracy and validation accuracy is increasing. Validation accuracy is slightly lower than training accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.7.

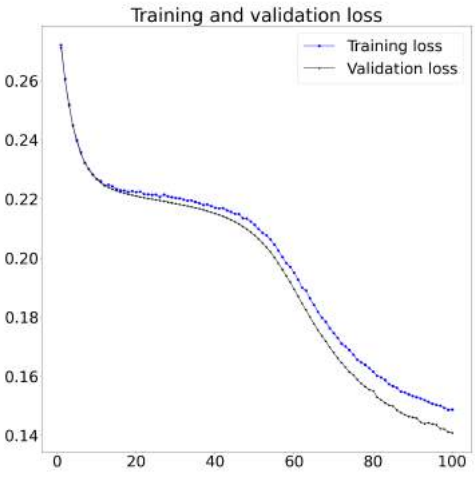

Figure 5.13: Training and Validation Loss

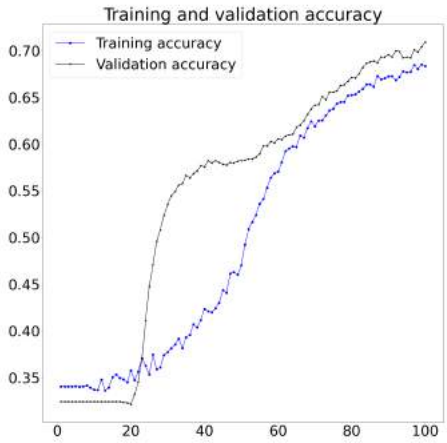

Figure 5.14: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	54%	45%	49%
Over Expsoed	77%	74%	76%
Under Expsoed	70%	83%	76%
Accuracy			68%
Macro Avg	67%	67%	67%
Weighted Avg	67%	68%	67%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	186	95	136
Over Expsoed	82	333	33
Under Expsoed	77	5	390

Table 5.7: Performance metrics and confusion matrix of trainable VGG16

5.2.3 FCNN model

Figure 5.15 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well at around 200

epochs but after that training loss decrease but validation loss don't. Figure 5.16 illustrates the training and validation accuracy of the model. Training accuracy is increasing aand validation accuracy is also increasing till 100 epochs after that validation accuracy doesn't increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.8.

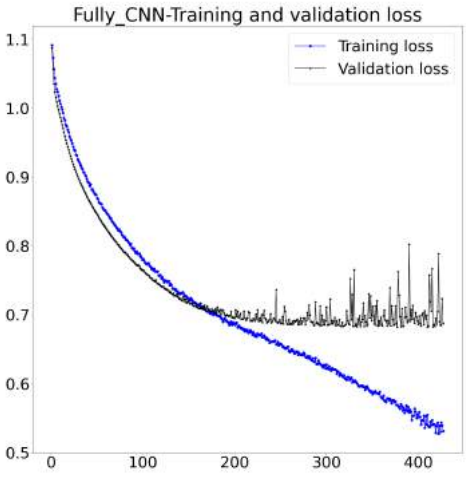

Figure 5.15: Training and Validation Loss

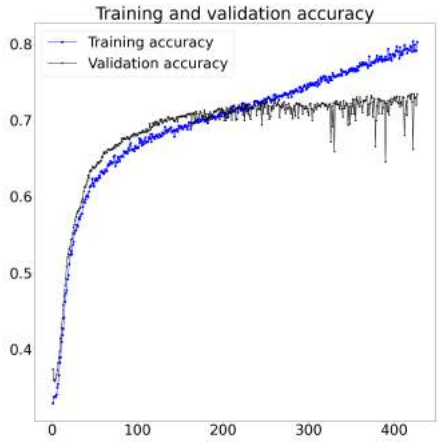

Figure 5.16: Training and validation Accuracy

Performance metrics			
Class	**Accuracy**	**Precision**	**Recall**
Normal	60%	52%	56%
Over Expsoed	80%	73%	76%
Under Expsoed	72%	86%	78%
Accuracy			71%
Macro Avg	71%	70%	70%
Weighted Avg	71%	71%	71%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	218	77	122
Over Expsoed	85	328	35
Under Expsoed	60	7	405

Table 5.8: Performance metrics and confusion matrix

5.2.4 CNN-1 model

Figure 5.17 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing till 80 epochs but after that both losses doesn't decrease as expected. Figure 5.18 illustrates the training and validation accuracy of the model. This model doesn't perform well. Both validation accuracy and training accuracy is increasing till around 100 epochs but after that both accuracy don't increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.9

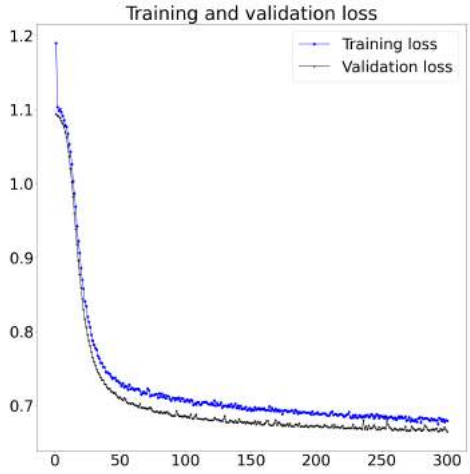

Figure 5.17: Training and Validation Loss

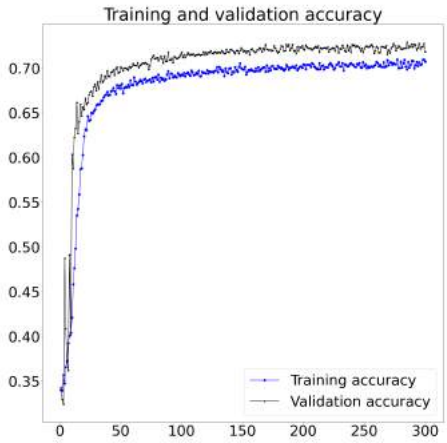

Figure 5.18: Training and validation Accuracy.

Performance metrics			
Class	**Accuracy**	**Precision**	**Recall**
Normal	55%	46%	50%
Over Expsoed	76%	79%	77%
Under Expsoed	72%	79%	75%
Accuracy			69%
Macro Avg	67%	68%	68%
Weighted Avg	68%	69%	68%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	193	107	117
Over Expsoed	67	353	28
Under Expsoed	93	6	373

Table 5.9: Performance metrics and confusion matrix

5.2.5 CNN-2 model

Figure 5.19 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.20 illustrates the training and validation accuracy of the model. This model does not perform well. Validation accuracy is increasing but training accuracy does not increasing as validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.10.

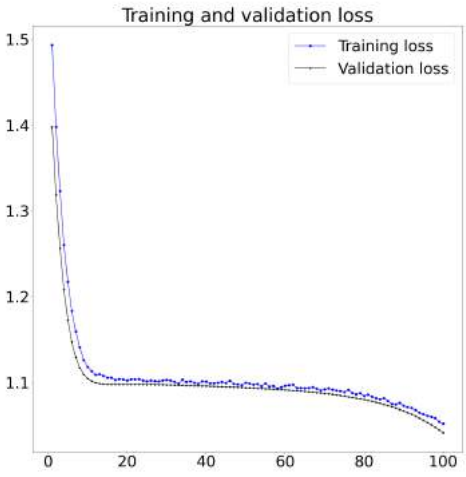

Figure 5.19: Training and Validation Loss

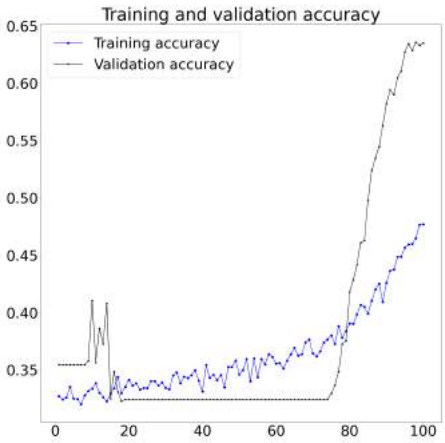

Figure 5.20: Large Size CNN Model Training and validation Accuracy.

Performance metrics			
Class	**Accuracy**	**Precision**	**Recall**
Normal	57%	44%	50%
Over Expsoed	77%	78%	77%
Under Expsoed	70%	82%	76%
Accuracy			69%
Macro Avg	68%	68%	68%
Weighted Avg	68%	69%	68%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	185	99	133
Over Expsoed	64	350	34
Under Expsoed	76	7	389

Table 5.10: Performance metrics and confusion matrix

5.3 Single Channel Images (Green)

5.3.1 Partially fine-tuned VGG-16 model

The Figure 5.21 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. The Figure 5.22 illustrates the training and validation accuracy of the model.This model doesn't perform well. Validation accuracy is increasing but training accuracy does not increasing as validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.11

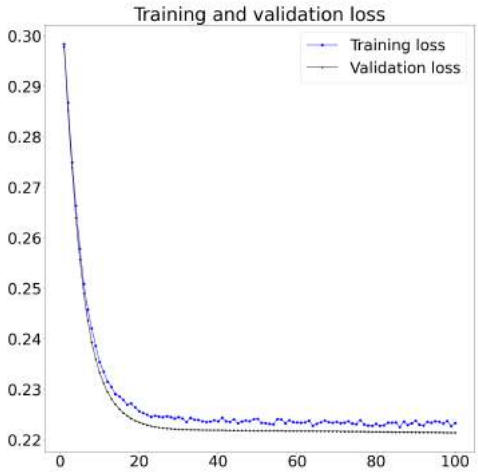

Figure 5.21: Training and Validation Loss

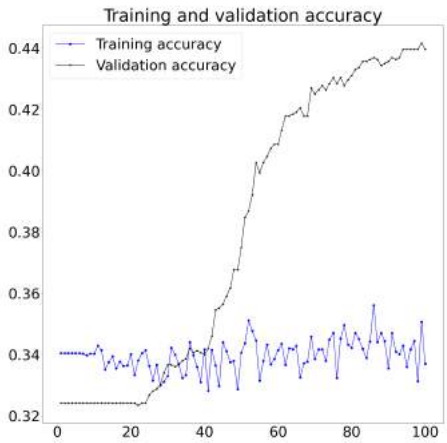

Figure 5.22: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	37%	53%	43%
Over Expsoed	60%	16%	25%
Under Expsoed	48%	62%	54%
Accuracy			44%
Macro Avg	48%	44%	41%
Weighted Avg	49%	44%	41%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	222	28	167
Over Expsoed	224	70	154
Under Expsoed	160	18	294

Table 5.11: Performance metrics and confusion matrix

5.3.2 Fully Fine-tuned VGG-16 model

The Figure 5.23 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. The Figure 5.24 illustrates the training and validation accuracy of the model. Both validation and training accuracy is increasing. Training accuracy is slightly lower than validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.12

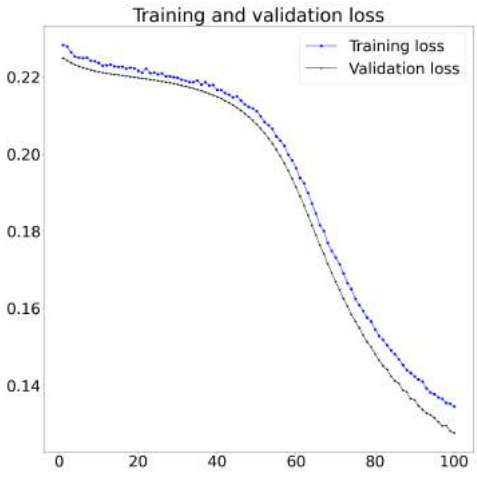

Figure 5.23: Training and Validation Loss

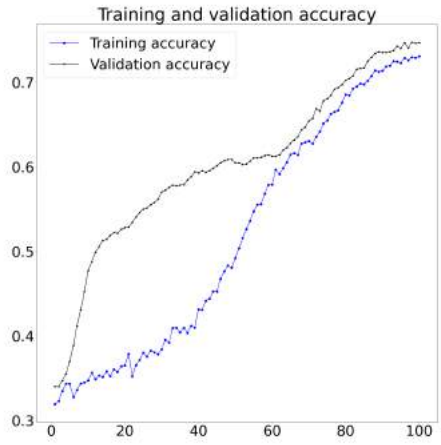

Figure 5.24: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	61%	54%	57%
Over Expsoed	78%	76%	77%
Under Expsoed	78%	87%	82%
Accuracy			73%
Macro Avg	72%	72%	72%
Weighted Avg	73%	73%	73%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	226	92	99
Over Expsoed	89	340	19
Under Expsoed	56	4	412

Table 5.12: Performance metrics and confusion matrix

5.3.3 FCNN model

The Figure 5.25 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well at around

200 epochs but after that training loss decrease but validation loss don't. The Figure 5.26 illustrates the training and validation accuracy of the model. Training accuracy is increasing aand validation accuracy is also increasing till 150 epochs after that validation accuracy doesn't increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.13

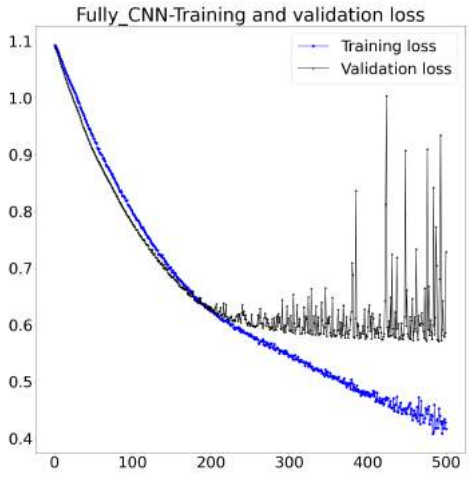

Figure 5.25: Training and Validation Loss

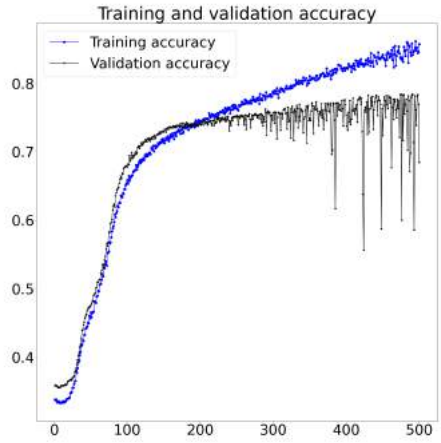

Figure 5.26: Training and validation Accuracy

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	52%	65%	57%
Over Expsoed	89%	46%	61%
Under Expsoed	75%	92%	82%
Accuracy			68%
Macro Avg	72%	68%	67%
Weighted Avg	72%	68%	67%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	269	24	124
Over Expsoed	216	208	24
Under Expsoed	36	3	433

Table 5.13: Performance metrics and confusion matrix

5.3.4 CNN-1 model

The Figure 5.27 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing till 100 epochs but after that both losses doesn't decrease as expected. The Figure 5.28 illustrates the training and validation accuracy of the model. Both validation accuracy and training accuracy is increasing till around 100 epochs but after that both accuracy don't increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.14

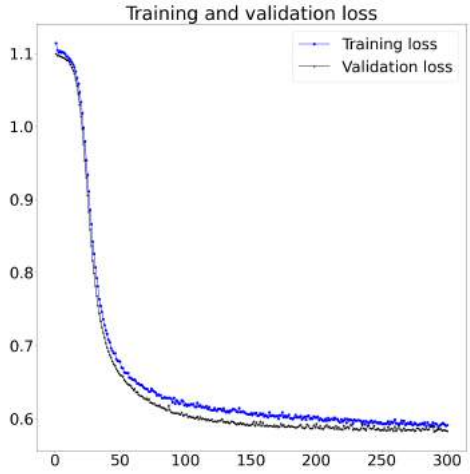

Figure 5.27: Training and Validation Loss

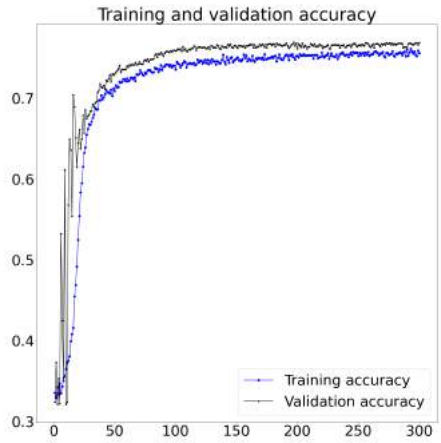

Figure 5.28: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	59%	58%	58%
Over Expsoed	80%	77%	79%
Under Expsoed	79%	82%	81%
Accuracy			73%
Macro Avg	72%	72%	72%
Weighted Avg	73%	73%	73%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	241	83	93
Over Expsoed	90	346	12
Under Expsoed	80	3	389

Table 5.14: Performance metrics and confusion matrix

5.3.5 CNN-2 model

The Figure 5.29 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. The Figure 5.30 illustrates the training and validation accuracy of the model. Validation accuracy is increasing but training accuracy does not increasing as validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.15

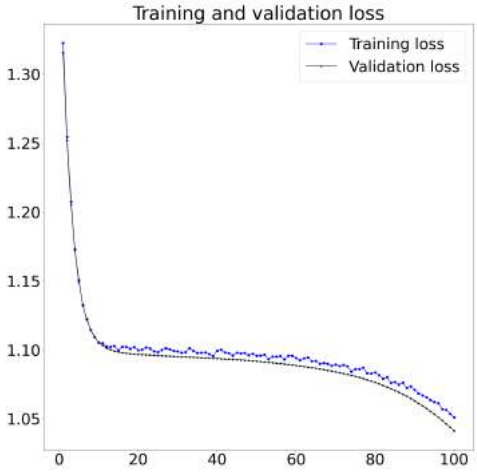

Figure 5.29: Training and Validation Loss

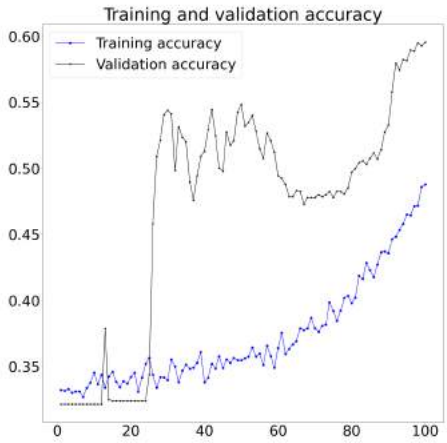

Figure 5.30: Large Size CNN Model Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	36%	28%	32%
Over Expsoed	57%	94%	71%
Under Expsoed	88%	53%	66%
Accuracy			59%
Macro Avg	60%	58%	56%
Weighted Avg	62%	59%	57%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	117	269	31
Over Expsoed	25	419	4
Under Expsoed	181	41	250

Table 5.15: Performance metrics and confusion matrix

5.4 Single Channel Images (Blue)

5.4.1 Partially fine-tuned VGG-16 model

Figure 5.31 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.32 illustrates the training and validation accuracy of the model.This model doesn't perform well. Validation accuracy is increasing but training accuracy does not increasing as validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.16.

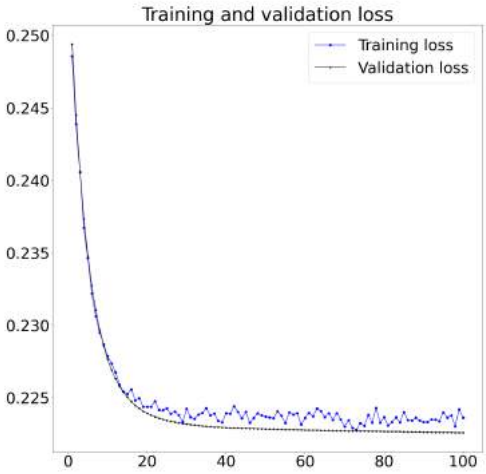

Figure 5.31: Training and Validation Loss

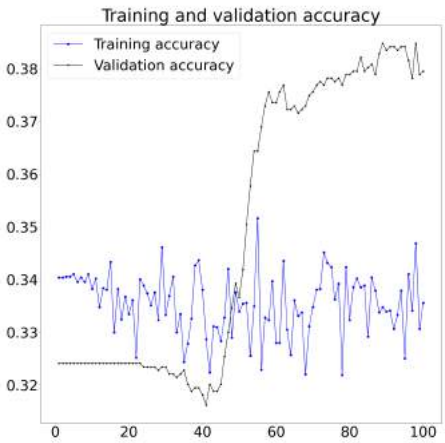

Figure 5.32: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	32%	43%	37%
Over Exposed	14%	0%	1%
Under Exposed	43%	69%	53%
Accuracy			38%
Macro Avg	30%	38%	30%
Weighted Avg	30%	38%	31%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	181	8	288
OverExposed	247	2	199
UnderExposed	141	4	327

Table 5.16: Performance metrics and confusion matrix

5.4.2 Fully Fine-tuned VGG-16 model

The Figure 5.33 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. The Figure 5.34 illustrates the training and validation accuracy of the model. Both validation and training accuracy is increasing. Training accuracy is slightly lower than validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.17.

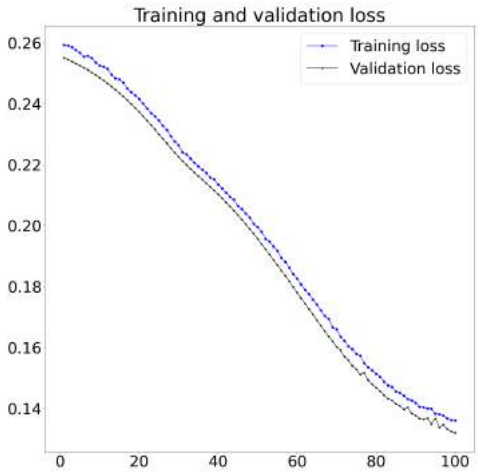

Figure 5.33: Training and Validation Loss

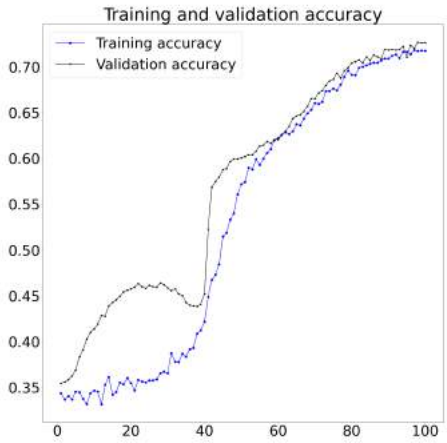

Figure 5.34: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	60%	56%	58%
Over Exposed	77%	76%	76%
Under Exposed	79%	84%	81%
Accuracy			73%
Macro Avg	72%	72%	72%
Weighted Avg	72%	73%	72%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	234	95	88
OverExposed	89	340	19
UnderExposed	69	6	397

Table 5.17: Performance metrics and confusion matrix

5.4.3 FCNN model

Figure 5.35 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well at around 200 epochs

but after that training loss decrease but validation loss don't. The Figure 5.36 illustrates the training and validation accuracy of the model. Training accuracy is increasing and validation accuracy is also increasing till 150 epochs after that validation accuracy doesn't increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.18.

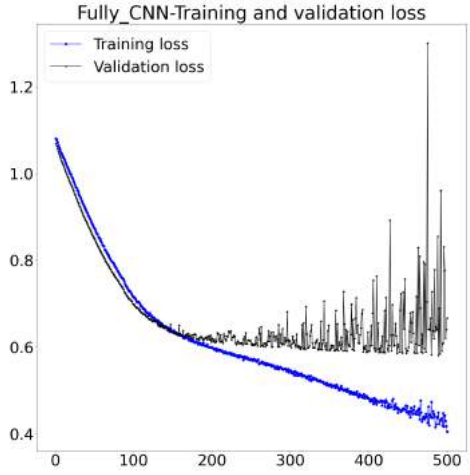

Figure 5.35: Training and Validation Loss

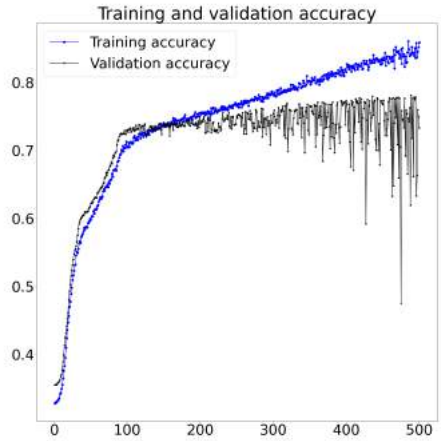

Figure 5.36: Training and validation Accuracy

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	56%	66%	60%
Over Exposed	90%	56%	69%
Under Exposed	76%	92%	83%
Accuracy			72%
Macro Avg	74%	71%	71%
Weighted Avg	74%	72%	71%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	274	26	117
OverExposed	180	249	19
UnderExposed	37	2	433

Table 5.18: Performance metrics and confusion matrix

5.4.4 CNN-1 model

The Figure 5.37 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing till 100 epochs but after that both losses does not decrease as expected. The Figure 5.38 illustrates the training and validation accuracy of the model. Both validation accuracy and training accuracy is increasing till around 100 epochs but after that both accuracy do not increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.19.

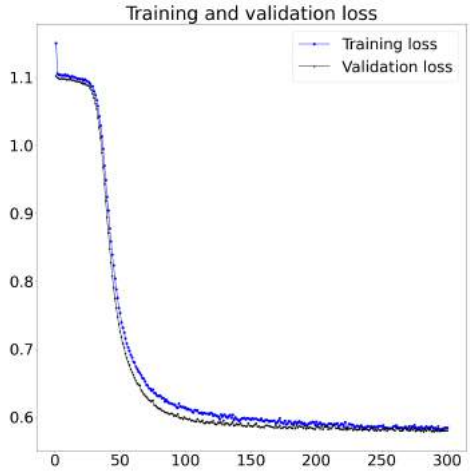

Figure 5.37: Training and Validation Loss

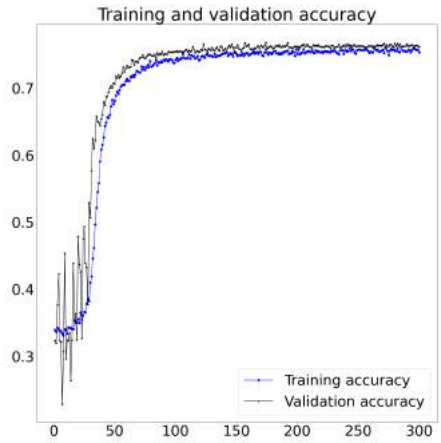

Figure 5.38: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	58%	61%	59%
Over Exposed	81%	73%	77%
Under Exposed	79%	83%	81%
Accuracy			73%
Macro Avg	73%	72%	72%
Weighted Avg	73%	73%	73%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	254	74	89
OverExposed	106	327	15
UnderExposed	79	3	390

Table 5.19: Performance metrics and confusion matrix

5.4.5 CNN-2 model

The Figure 5.29 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. The Figure 5.30 illustrates the training and validation accuracy of the model. Both accuracy are increasing quite well. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.20.

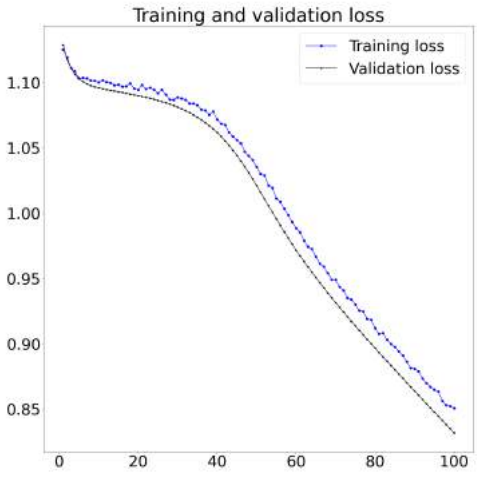

Figure 5.39: Training and Validation Loss

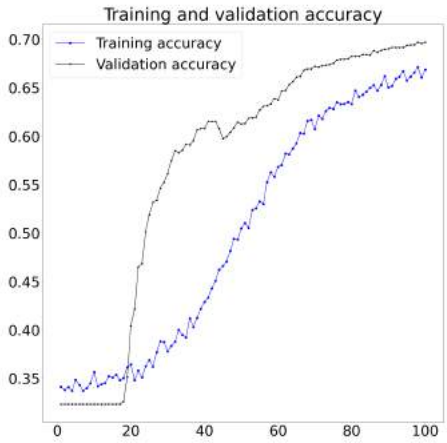

Figure 5.40: Large Size CNN Model Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	60%	30%	40%
Over Exposed	67%	89%	76%
Under Exposed	76%	87%	81%
Accuracy			70%
Macro Avg	68%	68%	66%
Weighted Avg	68%	70%	67%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	125	182	110
OverExposed	34	398	16
UnderExposed	50	13	409

Table 5.20: Performance metrics and confusion matrix

5.5 Image Histogram(Gray Scale)

5.5.1 Partially fine-tuned VGG-16 model

The Figure 5.41 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. The Figure 5.42 illustrates the training and validation accuracy of the model.This model doesn't perform well. Validation accuracy is increasing but training accuracy does not increasing as validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.21.

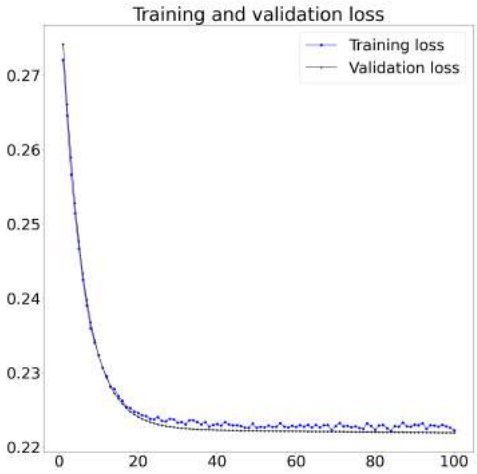

Figure 5.41: Training and Validation Loss

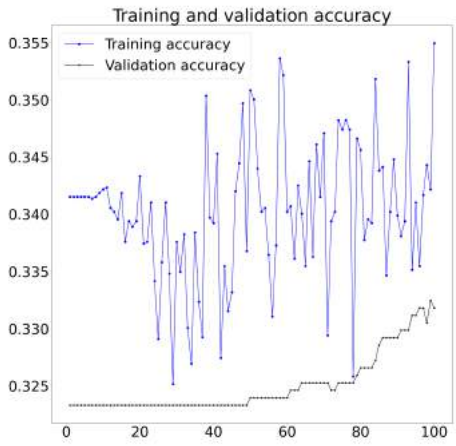

Figure 5.42: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	32%	99%	48%
Over Expsoed	50%	00%	00%
Under Expsoed	52%	03%	06%
Accuracy			32%
Macro Avg	44%	34%	18%
Weighted Avg	45%	32%	17%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	416	1	4
Over Expsoed	442	1	9
Under Expsoed	461	0	14

Table 5.21: Performance metrics and confusion matrix

5.5.2 Fully Fine-tuned VGG-16 model

The Figure 5.43 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. The Figure 5.44 illustrates the training and validation accuracy of the model. Both validation and training accuracy is increasing. Training accuracy is slightly lower than validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.22.

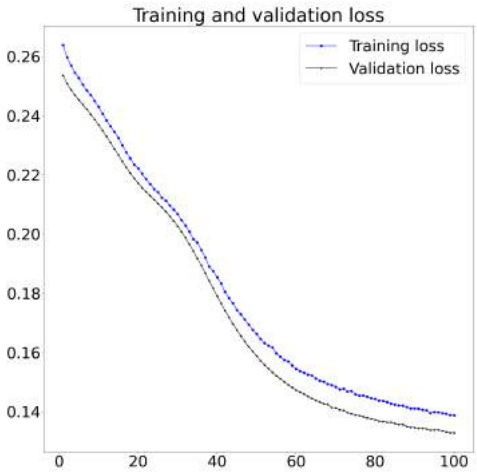

Figure 5.43: Training and Validation Loss

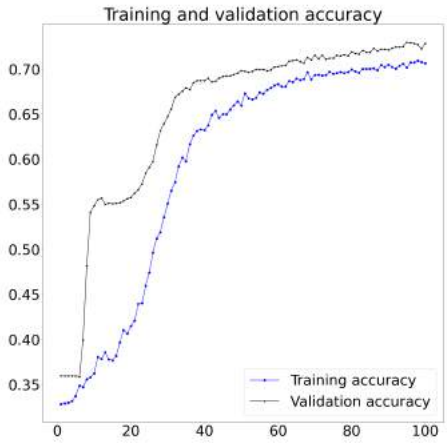

Figure 5.44: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	60%	59%	59%
Over Expsoed	80%	75%	77%
Under Expsoed	80%	85%	82%
Accuracy			74%
Macro Avg	73%	73%	73%
Weighted Avg	73%	74%	73%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	248	85	88
Over Expsoed	99	339	14
Under Expsoed	68	2	405

Table 5.22: Performance metrics and confusion matrix

5.5.3 FCNN model

The Figure 5.45 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well at around

120 epochs but after that training loss decrease but validation loss don't. The Figure 5.46 illustrates the training and validation accuracy of the model. Training accuracy is increasing and validation accuracy is also increasing till 100 epochs after that validation accuracy doesn't increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.23.

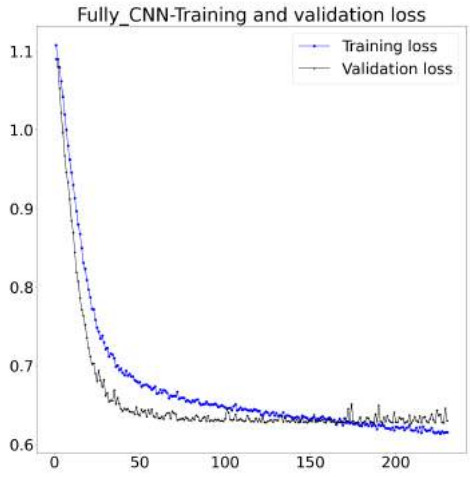

Figure 5.45: Training and Validation Loss

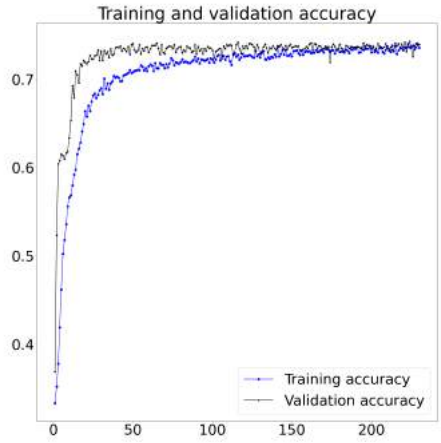

Figure 5.46: Training and validation Accuracy

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	67%	59%	62%
Over Expsoed	83%	83%	83%
Under Expsoed	79%	87%	83%
Accuracy			77%
Macro Avg	76%	76%	76%
Weighted Avg	76%	77%	76%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	247	78	96
Over Expsoed	64	374	14
Under Expsoed	60	1	414

Table 5.23: Performance metrics and confusion matrix

5.5.4 CNN-1 model

The Figure 5.47 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing till 150 epochs but after that both losses doesn't decrease as expected. The Figure 5.48 illustrates the training and validation accuracy of the model. Both validation accuracy and training accuracy is increasing till around 120 epochs but after that both accuracy don't increase as expected. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.24.

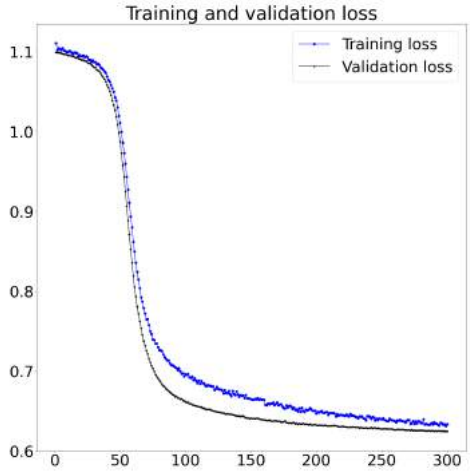

Figure 5.47: Training and Validation Loss

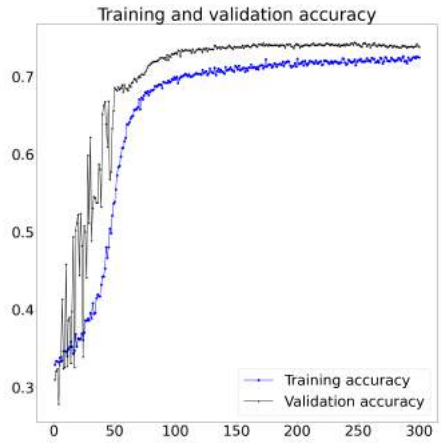

Figure 5.48: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	65%	64%	65%
Over Expsoed	84%	81%	82%
Under Expsoed	81%	84%	83%
Accuracy			77%
Macro Avg	77%	77%	77%
Weighted Avg	77%	77%	77%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	271	71	79
Over Expsoed	74	365	13
Under Expsoed	74	0	401

Table 5.24: Performance metrics and confusion matrix

5.5.5 CNN-2 model

The Figure 5.49 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. The Figure 5.50 illustrates the training and validation accuracy of the model. In this mode validation accuracy is increasing but training accuracy does not increasing as validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.25.

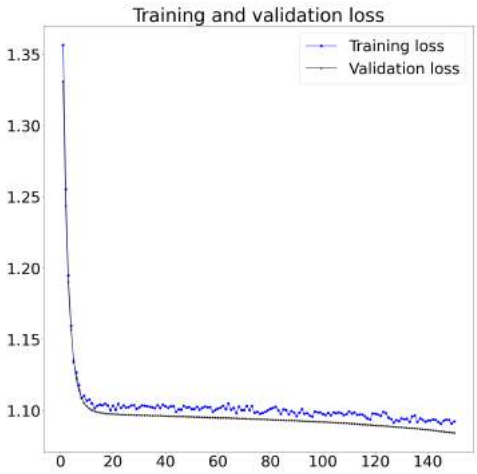

Figure 5.49: Training and Validation Loss

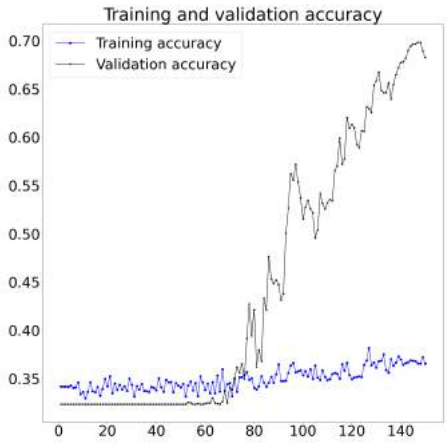

Figure 5.50: Large Size CNN Model Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	57%	64%	60%
Over Expsoed	84%	64%	72%
Under Expsoed	72%	81%	76%
Accuracy			70%
Macro Avg	71%	69%	70%
Weighted Avg	71%	70%	70%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	269	52	100
Over Expsoed	116	288	48
Under Expsoed	88	3	384

Table 5.25: Performance metrics and confusion matrix

5.6 Negative Images

5.6.1 Partially fine-tuned VGG-16 model

Figure 5.51 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.52 illustrates the training and validation accuracy of the model.This model doesn't perform well. Validation accuracy is increasing at first but some epochs later it start to decrease and again start to increase. Training accuracy does not increase as validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.26.

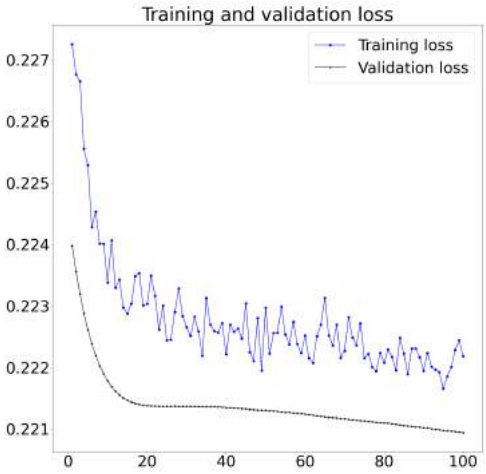

Figure 5.51: Training and Validation Loss

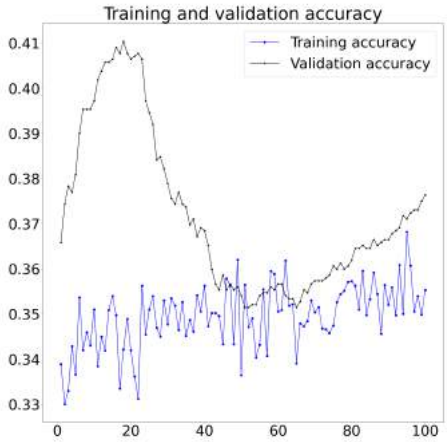

Figure 5.52: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	34%	59%	44%
Over Expsoed	42%	34%	38%
Under Expsoed	63%	35%	45%
Accuracy			42%
Macro Avg	47%	43%	42%
Weighted Avg	47%	42%	42%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	248	112	61
Over Expsoed	262	155	35
Under Expsoed	209	100	166

Table 5.26: Performance metrics and confusion matrix

5.6.2 Fully Fine-tuned VGG-16 model

Figure 5.53 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.54 illustrates the training and validation accuracy of the model. Validation accuracy is increasing at first but some epochs later it start to decrease and again start to increase. Training accuracy does not increase as validation accuracy. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.27.

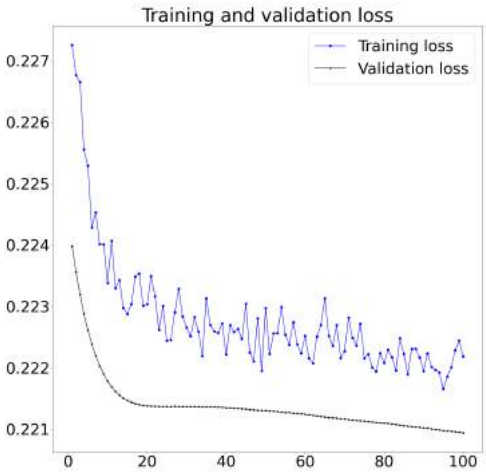

Figure 5.53: Training and Validation Loss

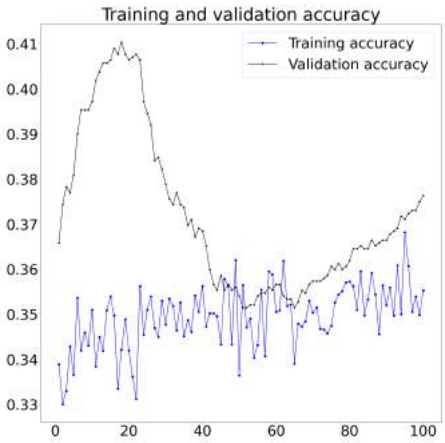

Figure 5.54: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	50%	27%	35%
Over Expsoed	68%	78%	72%
Under Expsoed	69%	87%	77%
Accuracy			65%
Macro Avg	62%	64%	61%
Weighted Avg	63%	65%	62%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	113	161	147
Over Expsoed	59	352	41
Under Expsoed	53	8	414

Table 5.27: Performance metrics and confusion matrix

5.6.3 FCNN model

Figure 5.55 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well till around 100

epochs but after that training loss decrease but validation loss do not. Figure 5.56 illustrates the training and validation accuracy of the model. Training accuracy is increasing and validation accuracy is also increasing till 150 epochs after that validation accuracy does not increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.28.

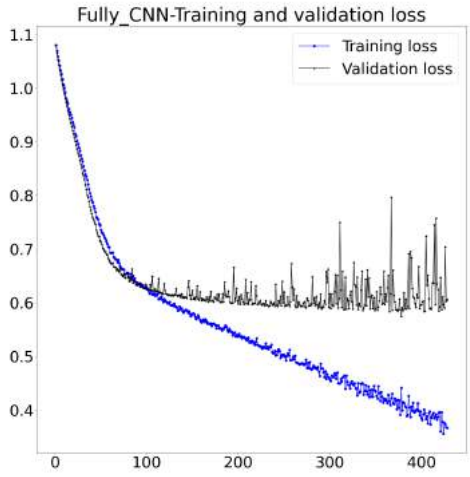

Figure 5.55: Training and Validation Loss

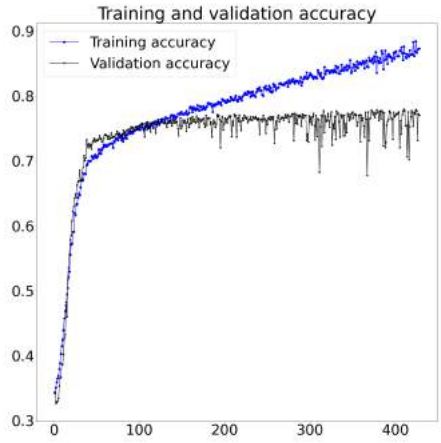

Figure 5.56: Training and validation Accuracy

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	70%	64%	67%
Over Expsoed	87%	78%	82%
Under Expsoed	79%	93%	85%
Accuracy			79%
Macro Avg	79%	78%	78%
Weighted Avg	79%	79%	79%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	271	51	99
Over Expsoed	83	351	18
Under Expsoed	34	1	440

Table 5.28: Performance metrics and confusion matrix

5.6.4 CNN-1 model

The Figure 5.57 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing till 100 epochs but after that both losses doesn't decrease as expected. The Figure 5.58 illustrates the training and validation accuracy of the model. Both validation accuracy and training accuracy is increasing till around 100 epochs but after that both accuracy don't increase. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.29.

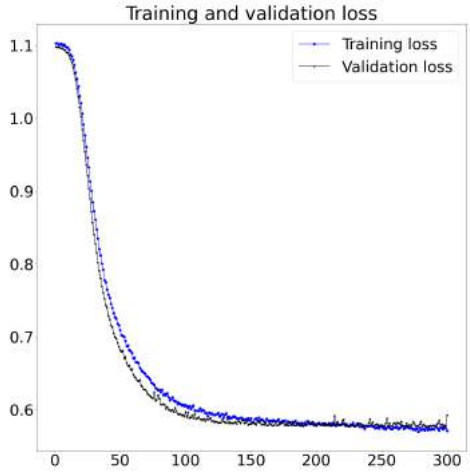

Figure 5.57: Training and Validation Loss

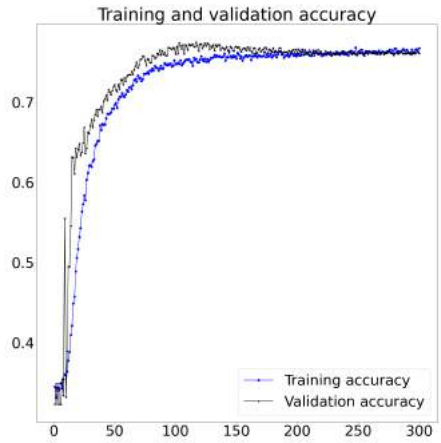

Figure 5.58: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	67%	65%	66%
Over Expsoed	86%	76%	81%
Under Expsoed	79%	91%	85%
Accuracy			78%
Macro Avg	78%	77%	77%
Weighted Avg	78%	78%	78%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	272	54	95
Over Expsoed	89	344	19
Under Expsoed	42	1	432

Table 5.29: Performance metrics and confusion matrix

5.6.5 CNN-2 model

The Figure 5.59 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. The Figure 5.60 illustrates the training and validation accuracy of the model. Both accuracy are increasing quite good. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.30.

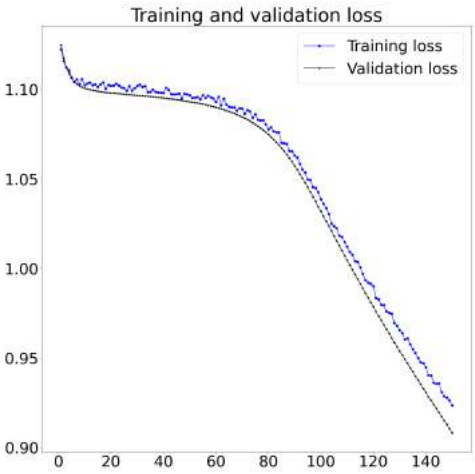

Figure 5.59: Training and Validation Loss

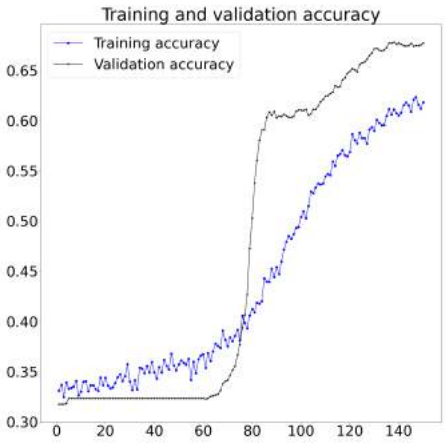

Figure 5.60: Large Size CNN Model Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	63%	48%	54%
Over Expsoed	88%	73%	80%
Under Expsoed	69%	96%	80%
Accuracy			73%
Macro Avg	73%	72%	71%
Weighted Avg	74%	73%	72%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	200	44	177
Over Expsoed	99	328	25
Under Expsoed	18	1	456

Table 5.30: Performance metrics and confusion matrix

5.7 Images with Increased and Decreased Exposure

5.7.1 Partially fine-tuned VGG-16 model

Figure 5.61 illustrates the training loss and validation loss of the model. The training loss and validation loss both decrease in first few epoch but later both are not decreasing as expected. The Figure 5.62 illustrates the training and validation accuracy of the model. This model does not perform well. Training accuracy is increasing at very little rate. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.31.

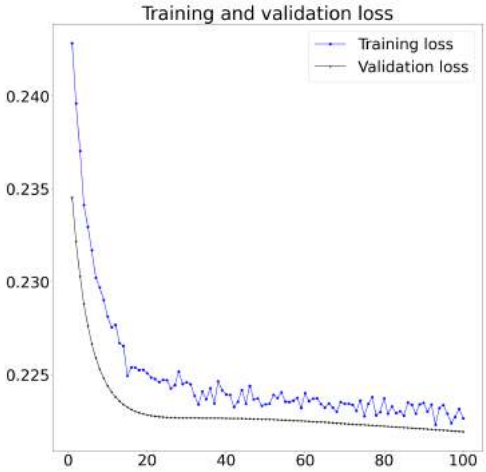

Figure 5.61: Training and Validation Loss

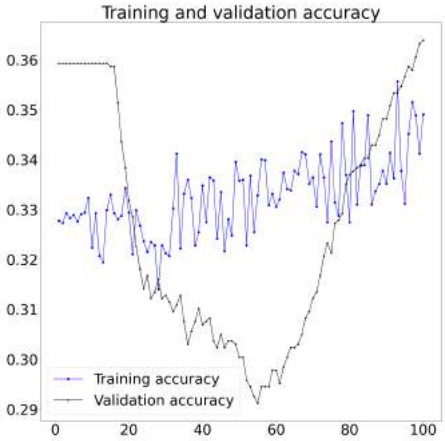

Figure 5.62: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	32%	59%	42%
Over Expsoed	38%	06%	11%
Under Expsoed	39%	42%	40%
Accuracy			35%
Macro Avg	37%	36%	31%
Weighted Avg	37%	35%	31%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	248	41	132
Over Expsoed	243	28	181
Under Expsoed	273	4	198

Table 5.31: Performance metrics and confusion matrix

5.7.2 Fully Fine-tuned VGG-16 model

Figure 5.63 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.64 illustrates the training and validation accuracy of the model. Both validation and training accuracy is increasing with time. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.32.

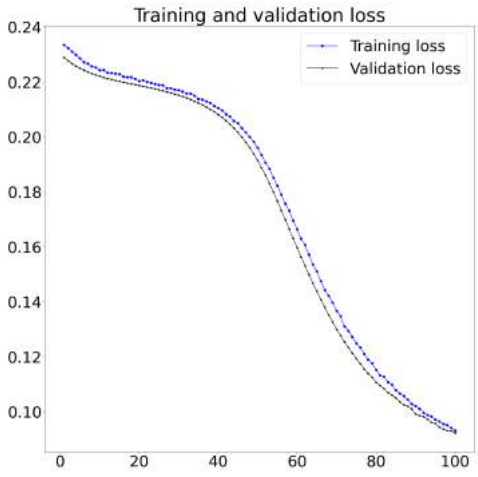

Figure 5.63: Training and Validation Loss

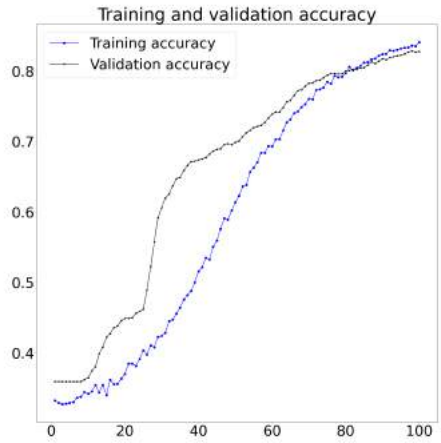

Figure 5.64: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	74%	75%	74%
Over Expsoed	87%	80%	84%
Under Expsoed	88%	94%	91%
Accuracy			83%
Macro Avg	83%	83%	83%
Weighted Avg	83%	83%	83%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	314	53	54
Over Expsoed	83	363	6
Under Expsoed	29	0	446

Table 5.32: Performance metrics and confusion matrix

5.7.3 FCNN model

Figure 5.65 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite very well till 150 epochs.

The Figure 5.66 illustrates the training and validation accuracy of the model. This model performs very well. Both validation accuracy and training accuracy is increasing. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.33.

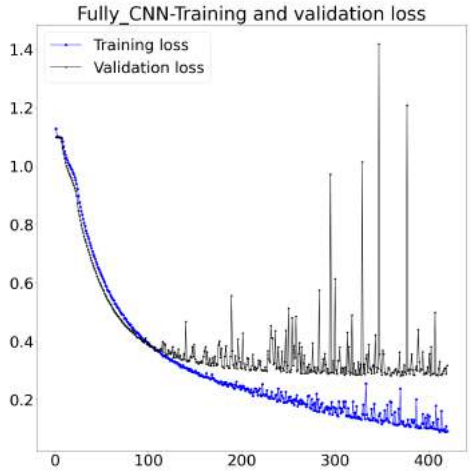

Figure 5.65: Training and Validation Loss

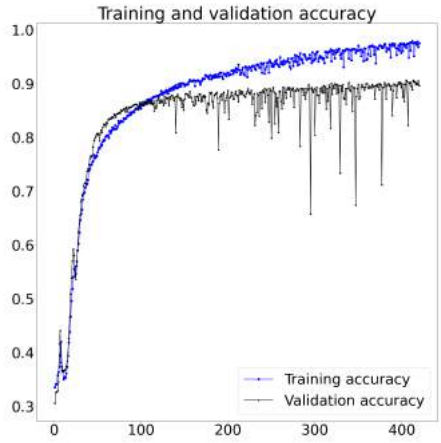

Figure 5.66: Training and validation Accuracy

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	86%	84%	85%
Over Expsoed	96%	86%	91%
Under Expsoed	89%	00%	94%
Accuracy			90%
Macro Avg	90%	90%	90%
Weighted Avg	90%	90%	90%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	354	16	51
Over Expsoed	57	390	5
Under Expsoed	2	0	473

Table 5.33: Performance metrics and confusion matrix

5.7.4 CNN-1 model

The Figure 5.65 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite very well till 150 epochs. The Figure 5.66 illustrates the training and validation accuracy of the model. This model performs very well. Both validation accuracy and training accuracy is increasing. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.34.

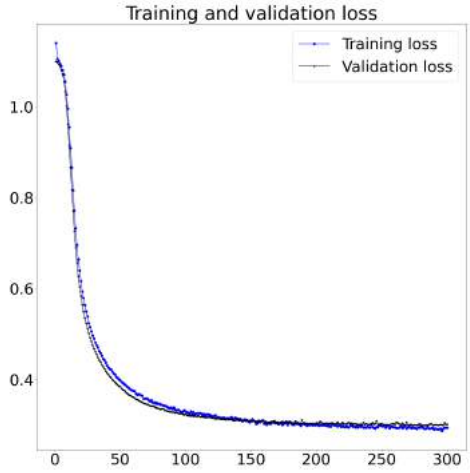

Figure 5.67: Training and Validation Loss

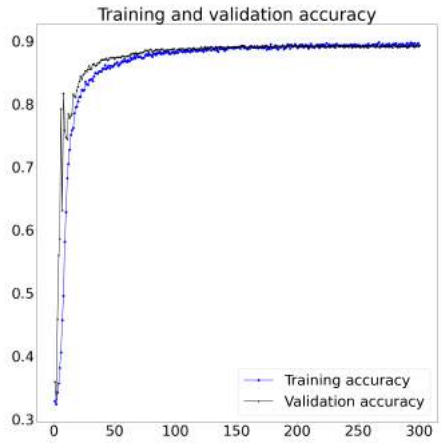

Figure 5.68: Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	85%	84%	85%
Over Expsoed	95%	86%	90%
Under Expsoed	89%	00%	94%
Accuracy			89%
Macro Avg	91%	90%	90%
Weighted Avg	91%	91%	91%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	354	16	51
Over Expsoed	57	384	11
Under Expsoed	467	1	461

Table 5.34: Performance metrics and confusion matrix

5.7.5 CNN-2 model

Figure 5.69 illustrates the training loss and validation loss of the model. The training loss and validation loss both are decreasing quite well. Figure 5.70 illustrates the training and validation accuracy of the model. The performance of the classifier model and the confusion matrix for the model is shown in Table 5.35.

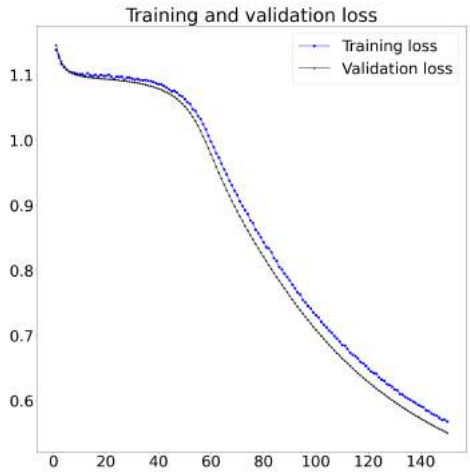

Figure 5.69: Training and Validation Loss

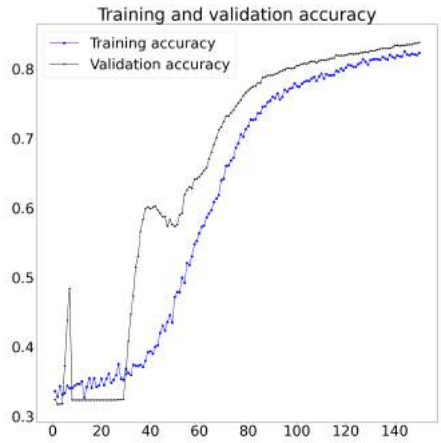

Figure 5.70: Large Size CNN Model Training and validation Accuracy.

Performance metrics			
Class	**Precision**	**Recall**	**F1-Score**
Normal	84%	65%	73%
Over Expsoed	79%	90%	84%
Under Expsoed	91%	97%	94%
Accuracy			85%
Macro Avg	85%	84%	84%
Weighted Avg	85%	85%	84%

Confusion Matrix			
	Normal	OverExposed	UnderExposed
Normal	275	108	38
Over Expsoed	38	408	6
Under Expsoed	16	0	459

Table 5.35: Performance metrics and confusion matrix

5.8 Performance Comparisons of Models

We have five models, among these models partially fine-tuned VGG16 performance is very poor, accuracy of this model on different forms of data set (RGB image, Red Green Blue channel of image, image histogram, inverted image, changed exposure image set) are 31%, 35%, 44%, 38%, 32%, 42%, 35%. We figure out the reason for low accuracy in this model, our data set is not drastically different in context from the dataset which the pre-trained VGG-16 model is trained on, but VGG-16 is trained to classify object in image where our classification totally different because we do not need object identification. We only need exposure information which is totally different from object classification, that is why trained parameter of pre-trained VGG-16 model is not working properly in our classification.

Our FCNN and CNN-1 model perform very well. Accuracy of FCNN are 81%, 71%, 68%, 72%, 77%, 79%, 90%. And accuracy of CNN-1 model are 78%, 69%, 73%, 73%, 77%, 78%, 89%. Performance of other two model (fully fine-tuned VGG16 and our CNN-2) are average. Specifically most of the model give best performance on our increased and decreased exposure image set. Performance of FCNN model is 90% and performance of CNN-1 model is 89% on our changed exposure image set. Here is the accuracy comparison table:

Model Accuracy							
Model	**RGB**	**Red**	**Green**	**Blue**	**Histogram**	**Inverted**	**Exposure change**
Partially fine-tuned VGG-16	31%	35%	44%	38%	32%	42%	35%
Fine-tuned VGG-16	75%	68%	73%	73%	74%	65%	83%
FCNN	81%	71%	68%	72%	77%	79%	90%
CNN-1	78%	69%	73%	73%	77%	78%	89%
CNN-2	76%	69%	59%	70%	70%	73%	85%

Table 5.36: Accuracy Comparison Table

5.9 Image in Frequency Domain

We convert MIT-Adobe FiveK dataset to frequency domain image. Then we apply all of our model on imageset of frequency domain, but we get extremely poor performance in classification by all models. That is why we do not include Image in frequency domain technique.

5.10 Skewness

Skewness take vital rule in image exposure. We calculate skewness all of normal, over-expoosed and underexposed images. Red,Green,Blue and gray channel skewness value of normal image given in fig 5.71.

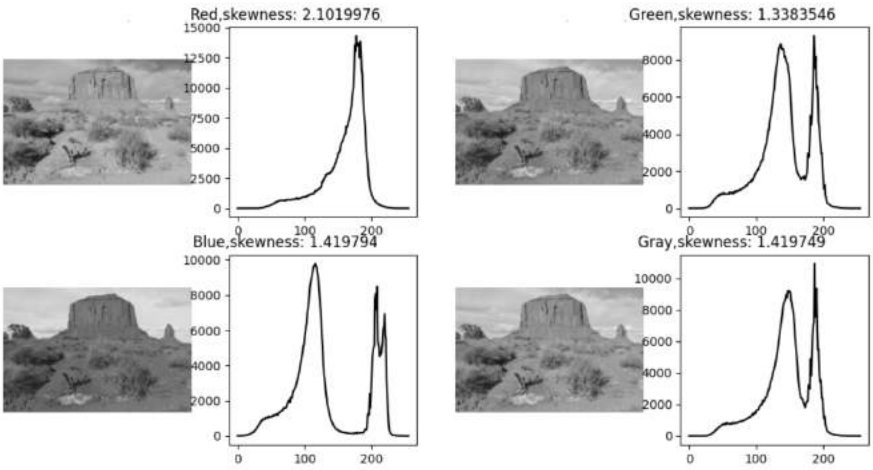

Figure 5.71: Red,Green,Blue and gray channel skewness value of normal image

Red,Green,Blue and gray channel skewness value of overexposed image given in fig 5.72.

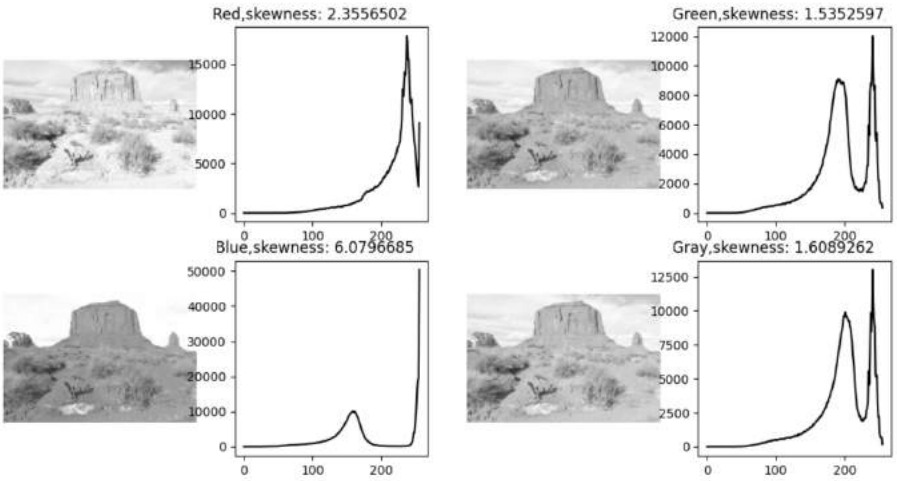

Figure 5.72: Red,Green,Blue and gray channel skewness value of overexposed image

Red,Green,Blue and gray channel skewness value of under-exposed image given in Fig 5.73

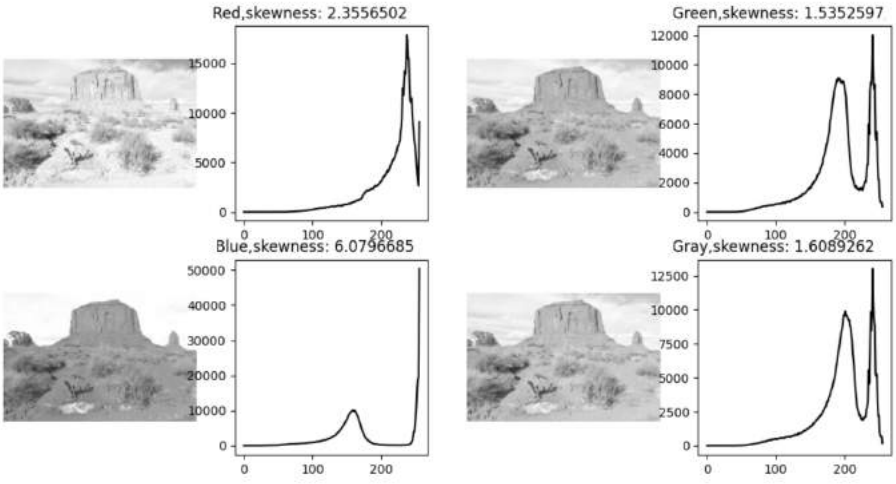

Figure 5.73: Red,Green,Blue and gray channel skewness value of under-exposed image

From fig 5.71 ,fig 5.72 and fig 5.73 we show that normal, overexposed and underexposed images red channel, blue channel, green channel and gray scale image skewness value are different. Then we try to identify what the relation between noraml, overexposed and underexposed images. Given below the red channel image skewness value count graph on fig 5.74. From fig 5.74 we show, in the range 0.25 to 0.3 skewness value number of normal image 160, number of overexposed image 170 and number of underexposed image 242. Similarly we check every range and count how much data in the range. From this graph we don't get enough information that can be useful to differentiate between normal, overexposed and underexposed image.

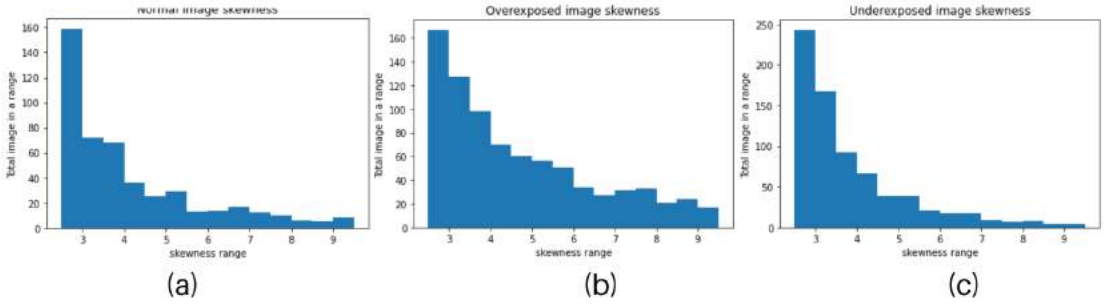

Figure 5.74: Normal, overexposed and underexposed images skewness count, x-axis represent the skewness value and y-axis represent how much data in a range. 0.5 subsequent range calcuated the number of image

5.11 Reason for Low Accuracy

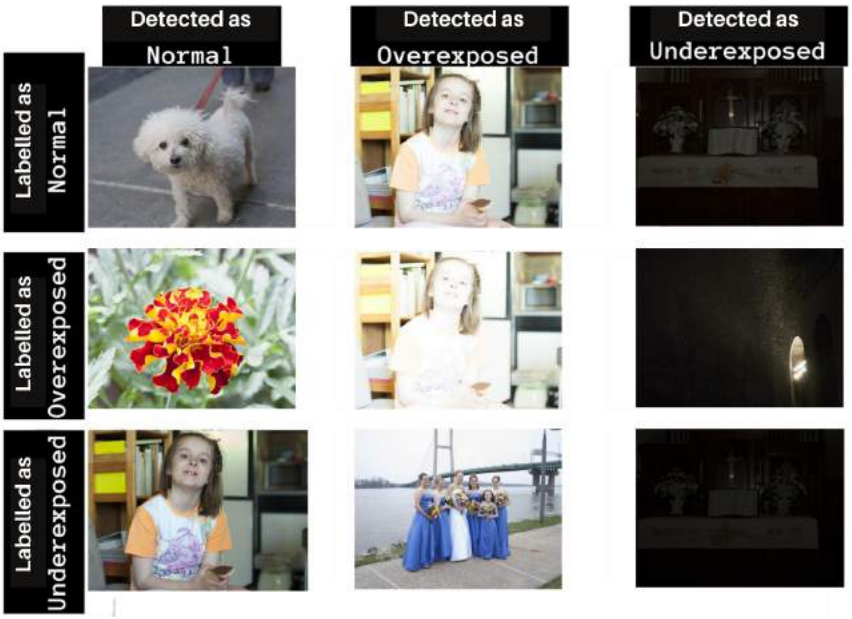

Figure 5.75: Some examples where our classifier's decisions contradicts with the data set provided labels.

The underlying reasons behind the mis-classifications can be better understood from Fig. 5.75. In this figure, the confusion matrix is shown with a few randomly selected sample images. First, it can be seen that most of the misclassified over-exposed images taken in low light and underexposed images taken in high light. Pictures taken in low light environment contains less exposure compared to normal image(which is taken in proper light environment) that is why sometimes model misclassifies the image by predicting as underexposed image.

In our data set, there are many images which are very difficult to correctly classify even by human. Many of our over-exposed labelled image have low exposure which are more likely normal image. And many of our normal labelled image have not correct exposure that is why some of them looked as over-exposed image and also some of them looked as under-exposed image. Just like over-exposed and normal image, many of our under-exposed image looks like properly exposed image.

Figure 5.76: We take these three images from our dataset to show that in our dataset there have some image which can not be detected even by naked eye(these image labelled as normal in dataset but if we look we can understand that these image don't look like normal image)

Figure 5.77: We take these three images from our dataset to show that in our dataset there have some image which can not be detected even by naked eye(these image labelled as overexposed in dataset but if we look we can understand that these image don't look like overexposed image)

Image-1 Image-2 Image-3

Figure 5.78: We take these three images from our dataset to show that in our dataset there have some image which can not be detected even by naked eye(these image labelled as under-exposed in dataset but if we look we can understand that these image don't look like under-exposed image)

Chapter 6

Conclusion

In this work, we have developed deep learning-based classifiers to decide whether an image is underexposed, overexposed, or normal. We have investigated the performance of a fully connected neural network (FCNN) and four convolutional neural networks (CNNs). Among four CNNs, two CNNs have been developed by applying a transfer learning approach. For this, we have fine-tuned parameters of a pre-trained CNN named VGG-16. The VGG16 is a popular classical CNN trained by the ImageNet data set containing 1.2 million images to classify 1,000 different classes. We have conducted experiments on a publicly available data set named MIT-Adobe FiveK. We have investigated the performance of all models by different kinds of pre-processed images. As evaluation metrics we have used *accuracy*, *precision* and *recall*. We have achieved the best performance when we have increased exposure and used the FCNN. We have achieved 90.94% accuracy.

Bibliography

[1] L. Medsker, *RECURRENT NEURAL NETWORKS*. ETD2000 International Conference in 1995, 1 ed., 1994.

[2] W. Tao, M. Al-Amin, H. Chen, M. C. Leu, Z. Yin, and R. Qin, "Real-time assembly operation recognition with fog computing and transfer learning for human-centered intelligent manufacturing," *Procedia Manufacturing*, vol. 48, pp. 926–931, 2020.

[3] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of pharmaceutical and biomedical analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[4] J.-S. Li, I.-H. Liu, C.-J. Tsai, Z.-Y. Su, C.-F. Li, and C.-G. Liu, "Secure content-based image retrieval in the cloud with key confidentiality," *IEEE Access*, vol. PP, pp. 1–1, 06 2020.

[5] B. Peterson, *Understanding exposure: how to shoot great photographs with any camera*. AmPhoto books, 2016.

[6] H. C. Karaimer and M. S. Brown, "A software platform for manipulating the camera imaging pipeline," in *European Conference on Computer Vision*, pp. 429–444, Springer, 2016.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] Z. Kolar, H. Chen, and X. Luo, "Transfer learning and deep convolutional neural networks for safety guardrail detection in 2d images," *Automation in Construction*, vol. 89, pp. 58–70, 2018.

[9] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[13] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-r. Mohamed, G. Dahl, and B. Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," *Neural networks*, vol. 64, pp. 39–48, 2015.

[14] F. Schnekenburger, M. Scharffenberg, M. Wülker, U. Hochberg, and K. Dorer, "Detection and localization of features on a soccer field with feedforward fully convolutional neural networks (fcnn) for the adultsize humanoid robot sweaty," in *Proceedings of the 12th workshop on humanoid soccer robots, IEEE-RAS international conference on humanoid robots, Birmingham*, sn, 2017.

[15] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128–3137, 2015.

[16] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[18] J. Turian, J. Bergstra, and Y. Bengio, "Quadratic features and deep architectures for chunking," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pp. 245–248, 2009.

[19] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[20] Y. Bengio, Y. LeCun, *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[21] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[22] X. Guo, Y. Li, and H. Ling, "Lime: Low-light image enhancement via illumination map estimation," *IEEE Transactions on Image Processing*, vol. 26, no. 2, pp. 982–993, 2017.

[23] G. Chunle, C. Li, J. Guo, C. C. Loy, J. Hou, S. Kwong, and R. Cong, "Zero-reference deep curve estimation for low-light image enhancement," 01 2020.

[24] R. Wang, Q. Zhang, C.-W. Fu, X. Shen, W.-S. Zheng, and J. Jia, "Underexposed photo enhancement using deep illumination estimation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6842–6850, 2019.

[25] C. Wei, W. Wang, W. Yang, and J. Liu, "Deep retinex decomposition for low-light enhancement," 08 2018.

[26] Q. Zhang, G. Yuan, C. Xiao, L. Zhu, and W.-S. Zheng, "High-quality exposure correction of underexposed photos," pp. 582–590, 10 2018.

[27] Y. Zhang, J. Zhang, and X. Guo, "Kindling the darkness: A practical low-light image enhancer," 05 2019.

[28] Y.-S. Chen, Y.-C. Wang, M.-H. Kao, and Y.-Y. Chuang, "Deep photo enhancer: Unpaired learning for image enhancement from photographs with gans," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6306–6314, 2018.

[29] M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff, and F. Durand, "Deep bilateral learning for real-time image enhancement," *ACM Transactions on Graphics (TOG)*, vol. 36, pp. 1 – 12, 2017.

[30] M. Afifi, K. G. Derpanis, B. Ommer, and M. S. Brown, "Learning to correct overexposed and underexposed photos," *arXiv preprint arXiv:2003.11596*, vol. 13, 2020.

[31] T. Celik and T. Tjahjadi, "Contextual and variational contrast enhancement," *Image Processing, IEEE Transactions on*, vol. 20, pp. 3431 – 3441, 01 2012.

[32] R. C. Gonzalez and R. E. Woods., *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., 3rd ed., 2001.

[33] C. Lee, C. Lee, and C.-S. Kim, "Contrast enhancement based on layered difference representation of 2d histograms," *IEEE Transactions on Image Processing*, vol. 22, no. 12, pp. 5372–5384, 2013.

[34] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer Vision, Graphics, and Image Processing*, vol. 39, no. 3, pp. 355–368, 1987.

[35] L. Yuan and J. Sun, "Automatic exposure correction of consumer photographs."

[36] G. Chunle, C. Li, J. Guo, C. C. Loy, J. Hou, S. Kwong, and R. Cong, "Zero-reference deep curve estimation for low-light image enhancement," 01 2020.

[37] S. Moran, P. Marza, S. McDonagh, S. Parisot, and G. Slabaugh, "Deeplpf: Deep local parametric filters for image enhancement," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12823–12832, 2020.

[38] J. Park, J.-Y. Lee, D. Yoo, and S.-O. Kweon, "Distort-and-recover: Color enhancement using deep reinforcement learning," pp. 5928–5936, 06 2018.

[39] E. H. Land, "The retinex theory of color vision scientific american," vol. 237(6), p. 108129, 2009.

[40] M. Shirvaikar, "An optimal measure for camera focus and exposure," in *Thirty-Sixth Southeastern Symposium on System Theory, 2004. Proceedings of the*, pp. 472–475, 2004.

[41] V. Bychkovsky, S. Paris, E. Chan, and F. Durand, "Learning photographic global tonal adjustment with a database of input / output image pairs," in *The Twenty-Fourth IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

Appendix A

Code

Listing A.1: Program for converting RGB imageset to red, green, blue channel image

```
import cv2
import os
import numpy as np

DIR = './Main Dataset/'

total_img = 2970

def prepare_image_array(imgDir, imgW, imgH):
    imgList = os.listdir(imgDir)
    # print(imgList)
    n = len(imgList)

    imgSet = []
    for i in range(total_img):
        imgPath = imgDir + imgList[i]
        if (os.path.exists(imgPath)):
            # print(imgPath)
            img = cv2.imread(imgPath)
            resizedImg = cv2.resize(img, (imgW, imgH))
            rgbImg = cv2.cvtColor(resizedImg, cv2.COLOR_BGR2RGB)
```

```
22              imgSet.append(rgbImg)
23          else:
24              print("It is not a valid image path.")
25
26      print("total image "+str(len(imgSet)))
27      imgSet = np.array(imgSet, dtype=np.uint8)
28      print("total shape "+str(imgSet.shape))
29
30      return imgSet
31
32  imgDir = DIR+'Normal/'
33  imgSet1 = prepare_image_array(imgDir, 512, 512)
34  m = imgSet1.shape[0]
35  imgDir = DIR+'OverExposed/'
36  imgSet2 = prepare_image_array(imgDir, 512, 512)
37  n = imgSet2.shape[0]
38
39  imgDir = imgDir = DIR+'UnderExposed/'
40  imgSet3 = prepare_image_array(imgDir, 512, 512)
41  o = imgSet3.shape[0]
42
43  imgSet = np.concatenate([imgSet1, imgSet2, imgSet3]);
44
45  print("toal image ====== " + str(imgSet.shape))
46
47
48  blue_normal = os.path.join('./blue-channel/', 'Normal')
49  blue_UnderExposed = os.path.join('./blue-channel/', 'UnderExposed')
50  blue_OverExposed = os.path.join('./blue-channel/', 'OverExposed')
51  if(os.path.exists(blue_normal) == False):
52      os.mkdir(blue_normal)
53  if(os.path.exists(blue_UnderExposed) == False):
54      os.mkdir(blue_UnderExposed)
55  if(os.path.exists(blue_OverExposed) == False):
```

```
56      os.mkdir(blue_OverExposed)
57
58  green_normal = os.path.join('./green-channel/', 'Normal')
59  green_UnderExposed = os.path.join('./green-channel/', 'UnderExposed')
60  green_OverExposed = os.path.join('./green-channel/', 'OverExposed')
61  if(os.path.exists(green_normal) == False):
62      os.mkdir(green_normal)
63  if(os.path.exists(green_UnderExposed) == False):
64      os.mkdir(green_UnderExposed)
65  if(os.path.exists(green_OverExposed) == False):
66      os.mkdir(green_OverExposed)
67
68  red_normal = os.path.join('./red-channel/', 'Normal')
69  red_UnderExposed = os.path.join('./red-channel/', 'UnderExposed')
70  red_OverExposed = os.path.join('./red-channel/', 'OverExposed')
71  if(os.path.exists(red_normal) == False):
72      os.mkdir(red_normal)
73  if(os.path.exists(red_UnderExposed) == False):
74      os.mkdir(red_UnderExposed)
75  if(os.path.exists(red_OverExposed) == False):
76      os.mkdir(red_OverExposed)
77
78
79
80
81  count=0
82  for  i in range(0, len(imgSet)):
83
84      # #write red channel to greyscale image
85
86      if(count<total_img):
87          red_channel = imgSet[i][:,:,2]
88          red_img = np.zeros(imgSet[i].shape)
89          red_img[:,:,2] = red_channel
```

```
90          cv2.imwrite(red_normal+'/image - %s.jpg'%count,red_img)
91
92          green_channel = imgSet[i][:,:,1]
93          green_img = np.zeros(imgSet[i].shape)
94          green_img[:,:,1] = green_channel
95          cv2.imwrite(green_normal+'/image - %s.jpg'%count,green_img)
96
97          blue_channel = imgSet[i][:,:,0]
98          blue_img = np.zeros(imgSet[i].shape)
99          blue_img[:,:,0] = blue_channel
100         cv2.imwrite(blue_normal+'/image - %s.jpg'%count,blue_img)
101
102     elif(count>= total_img and count<2*total_img):
103         red_channel = imgSet[i][:,:,2]
104         red_img = np.zeros(imgSet[i].shape)
105         red_img[:,:,2] = red_channel
106         cv2.imwrite(red_OverExposed+'/image - %s.jpg'%count,red_img)
107
108         green_channel = imgSet[i][:,:,1]
109         green_img = np.zeros(imgSet[i].shape)
110         green_img[:,:,1] = green_channel
111         cv2.imwrite(green_OverExposed+'/image - %s.jpg'%count,green_img)
112
113         blue_channel = imgSet[i][:,:,0]
114         blue_img = np.zeros(imgSet[i].shape)
115         blue_img[:,:,0] = blue_channel
116         cv2.imwrite(blue_OverExposed+'/image - %s.jpg'%count,blue_img)
117
118     else:
119         red_channel = imgSet[i][:,:,2]
120         red_img = np.zeros(imgSet[i].shape)
121         red_img[:,:,2] = red_channel
122         cv2.imwrite(red_UnderExposed+'/image - %s.jpg'%count,red_img)
123
```

```
        green_channel = imgSet[i][:,:,1]
        green_img = np.zeros(imgSet[i].shape)
        green_img[:,:,1] = green_channel
        cv2.imwrite(green_UnderExposed+'/image - %s.jpg'%count,green_img)

        blue_channel = imgSet[i][:,:,0]
        blue_img = np.zeros(imgSet[i].shape)
        blue_img[:,:,0] = blue_channel
        cv2.imwrite(blue_UnderExposed+'/image - %s.jpg'%count,blue_img)
    count+=1
    print(count)
```

Listing A.2: Program for increasing and decreasing exposure of image

```
from PIL import Image, ImageEnhance
import os

DIR = './Main Database/'

overexposed = DIR+'OverExposed/'
underexposed = DIR+'UnderExposed/'
overexposed_dir = './increased and decrased exposure/OverExposed/'
underexposed_dir = './increased and decrased exposure/UnderExposed/'

#read the image



def ChangeExposure(dir,destination_dir,factor):
    imgList = os.listdir(dir)
    failed_img = 0
    for i in range(len(imgList)):
        try:
            im = Image.open(dir+imgList[i])
            #image brightness enhancer
```

```
            enhancer = ImageEnhance.Brightness(im)
            im_output = enhancer.enhance(factor)
            im_output.save(destination_dir+'%s'%imgList[i])

        except Exception:
            failed_img+=1
            print("%s =====> not work properly\n"%imgList[i])
    print("%s failed total = %s"%(dir,failed_img))


ChangeExposure(overexposed,overexposed_dir,factor = 1.3)
ChangeExposure(underexposed,underexposed_dir,factor = 0.7)
```

Listing A.3: Program to convert RGB image to gray scale histogram

```
import os
import cv2
from scipy.stats import skew
import numpy as np
import matplotlib.pyplot as plt
DIR = './Main Database/'
destination = "./histogram/"
count =0

def main():
    name = "Normal/"
    imgDir = DIR + name
    imgList = os.listdir(imgDir)
    for imgFile in imgList:
        imgPath = imgDir + imgFile
        imgSet = load_img2(imgPath,name)
    name = "OverExposed/"
    imgDir = DIR + name
    imgList = os.listdir(imgDir)
    for imgFile in imgList:
```

```
        imgPath = imgDir + imgFile
        imgSet = load_img2(imgPath,name)
    name = "UnderExposed/"
    imgDir = DIR + name
    imgList = os.listdir(imgDir)
    for imgFile in imgList:
        imgPath = imgDir + imgFile
        imgSet = load_img2(imgPath,name)
def load_img2(imgPath,name):
    global count
    print(count)
    bgrImg = cv2.imread(imgPath)
    grayImg = cv2.cvtColor(bgrImg, cv2.COLOR_BGR2GRAY)
    plt.hist(grayImg.ravel(), bins=256, range=(0, 256), color = 'k')
        #calculating histogram
    plt.savefig(destination + name
        +imgPath.split('/')[-1].split('.')[0] + '.jpg')
    count+=1
    plt.clf()


if __name__ == '__main__':
    main()
```

Listing A.4: Program for image inversion

```
from PIL import Image
import PIL.ImageOps
import os


DIR = './Main Database/'
normal = DIR+'Normal/'
overexposed = DIR+'OverExposed/'
underexposed = DIR+'UnderExposed/'
```

```
normal_dir = './invert-image/Normal/'
overexposed_dir = './invert-image/OverExposed/'
underexposed_dir = './invert-image/UnderExposed/'


def TransferFrequencyDomain(dir,destination_dir):
    imgList = os.listdir(dir)
    failed_img = 0
    for i in range(len(imgList)):
        try:
            image = Image.open(dir+imgList[i])
            inverted_image = PIL.ImageOps.invert(image)
            inverted_image.save(destination_dir+'%s'%imgList[i])
        except Exception:
            failed_img+=1
            print("%s =====> not work properly\n"%imgList[i])
    print("%s failed total = %s"%(dir,failed_img))



TransferFrequencyDomain(normal,normal_dir)
TransferFrequencyDomain(overexposed,overexposed_dir)
TransferFrequencyDomain(underexposed,underexposed_dir)
```

Listing A.5: Program for estimation of skewness and Entropy of main dataset

```
from audioop import avg
from distutils.command.build_scripts import first_line_re
import os
import cv2
from scipy.stats import skew
import numpy as np
import matplotlib.pyplot as plt
import csv

```

```
# open the file in the write mode
f = open('./Estimate_SkewnessEntropy_of_main_dataset.csv', 'w')

# create the csv writer
writer = csv.writer(f)

# write a row to the csv file
first_line =
    [['image-id'],['Normal']*8,['OverExposed']*8,['UnderExposed']*8,['Red']*4,['Green'
first_row = []
for i in first_line:
    for kk in i:
        first_row.append(kk)

second_line =
    [['image-id'],['Red','Red','Green','Green','Blue','Blue','Gray','Gray']*3,[['Skewn
second_row = []
for i in second_line:
    for kk in i:
        if len(kk)==2:
            for jj in kk:
                second_row.append(jj)
        else:
            second_row.append(kk)

third_line = [['image-id'],['Skewness','Entropy']*12 ,
    ['average','standard deviation']*8]
third_row = []
for i in third_line:
    for kk in i:
        third_row.append(kk)


writer.writerow(first_row)
```

```
41 writer.writerow(second_row)
42 writer.writerow(third_row)
43
44 DIR = './Main Dataset/'
45
46
47 def main():
48     Normal = os.listdir(DIR + '/Normal')
49     OverExposed = os.listdir(DIR + '/OverExposed')
50     UnderExposed = os.listdir(DIR + '/UnderExposed')
51     Normal.sort()
52     OverExposed.sort()
53     UnderExposed.sort()
54
55     j = 0
56     k = 0
57     for i in range(len(Normal)):
58         normal = Normal[i].split('-')[0][1:]
59         over = 0
60         under = 0
61         if j< len(OverExposed):
62             over = OverExposed[j].split('-')[0][1:]
63             if (over < normal):
64                 while (over < normal):
65                     j += 1
66                     if j< len(OverExposed):
67                         over = OverExposed[j].split('-')[0][1:]
68                         if over==normal:
69                             j+=1
70                             break
71                     else:
72                         over = 0
73                         break
74             else:
```

```
75                  if normal == over:
76                      j += 1
77
78
79          if k< len(UnderExposed):
80              under = UnderExposed[k].split('-')[0][1:]
81              if( under<normal):
82                  while under<normal:
83                      k+=1
84                      if k< len(UnderExposed):
85                          under = UnderExposed[k].split('-')[0][1:]
86                          if under==normal:
87                              k+=1
88                              break
89                      else:
90                          under = 0
91                          break
92              else:
93                  if normal==under:
94                      k+=1
95
96          if normal == over and normal == under:
97              head = []
98              head.append(Normal[i])
99
100             entropyDict = {'Red': [], 'Green': [], 'Blue': [], 'Gray':
                    []}
101             skewnessDict = {'Red': [], 'Green': [], 'Blue': [], 'Gray':
                    []}
102
103             Label = ['/Normal/', '/OverExposed/', '/UnderExposed/']
104             data =
                    [str(Normal[i]),str(OverExposed[j-1]),str(UnderExposed[k-1])]
105             all_en_skew = []
```

```
106             for kk in range(len(Label)):
107                 path = DIR+Label[kk]+data[kk]
108                 chImgSet, titleSet, histSet = different_label_image(path)
109                 # plot_img(chImgSet, titleSet, histSet,Label[kk])
110 
111                 for ts in titleSet:
112                     all_en_skew.append(ts)
113 
114 
115             for aes in all_en_skew:
116                 head.append(str(aes.split(',')[1]))
117                 head.append(str(aes.split(',')[2]))
118                 skewnessDict[aes.split(',')[0]].append(float(aes.split(',')[1]))
119                 entropyDict[aes.split(',')[0]].append(float(aes.split(',')[2]))
120 
121 
122             for ch in ['Red', 'Green', 'Blue', 'Gray']:
123                 skewness_avg,skewness_std =
                        estimate_avg_std_performance(skewnessDict[ch])
124                 entropy_avg,entropy_std =
                        estimate_avg_std_performance(entropyDict[ch])
125                 head.append(skewness_avg)
126                 head.append(skewness_std)
127                 head.append(entropy_avg)
128                 head.append(entropy_std)
129             writer.writerow(head)
130 
131 
132 
133 
134 def different_label_image(imgPath):
135     chImgSet = []
136     titleSet = []
137     histSet = []
```

```
138
139     imgSet = load_img(imgPath)
140
141
142     for ch in ['Red', 'Green', 'Blue', 'Gray']:
143         hist, entropy, skewness = estimate_skewness_entropy(imgSet[ch])
144         msg = ch + ',' + str(skewness) + ',' + str(entropy)
145         titleSet.append(msg)
146         chImgSet.append(imgSet[ch])
147         histSet.append(hist)
148
149     return chImgSet, titleSet, histSet
150
151
152
153 def plot_img(imgSet, titleSet, histSet,img_type):
154     plt.figure(figsize = (20, 20))
155     j = 1
156     for i in range(4):
157         plt.subplot(2, 4, j)
158         j += 1
159         plt.imshow(imgSet[i], cmap = 'gray')
160         plt.axis('off')
161         plt.subplot(2, 4, j)
162         j += 1
163         plt.plot(histSet[i], color = 'k')
164         plt.title(img_type + titleSet[i])
165     plt.show()
166     plt.close()
167
168 def estimate_skewness_entropy(img):
169     hist = cv2.calcHist([img],[0], None, [256], [0,256])
170     skewness = skew(hist)[0]
171     normalizedData = hist / hist.sum() + 0.0000001
```

```
172     entropy = -(normalizedData * np.log(normalizedData)).sum()
173     print('Skewness: {}, Entropy: {}'.format(skewness, entropy))
174 
175     return hist, entropy, skewness
176 
177 def estimate_avg_std_performance(performanceList):
178     avg = np.average(np.array(performanceList))
179     std = np.std(np.array(performanceList))
180 
181     return avg,std
182 
183 def load_img(imgPath):
184     bgrImg = cv2.imread(imgPath)
185     grayImg = cv2.cvtColor(bgrImg, cv2.COLOR_BGR2GRAY)
186     blueImg, greenImg, redImg = cv2.split(bgrImg)
187     imgSet = {'Red': redImg, 'Green': greenImg, 'Blue': blueImg,
            'Gray': grayImg}
188     return imgSet
189 
190 
191 if __name__ == '__main__':
192     main()
```

Appendix B

Image source

We collect some of images from different website. Link of those website given below with figure number:

Figure 1.1 : https://www.studiobinder.com/blog/underexposed-photography-technique/
Figure 2.1 : https://capturetheatlas.com/what-is-exposure-in-photography/
Figure 2.2 : https://www.studiobinder.com/blog/what-is-overexposure-in-photography/
Figure 2.3 : https://vanceai.com/posts/fix-underexposed-and-overexposed-photos/
Figure 2.4 : https://photographylife.com/underexposure-and-overexposure-in-photography
Figure 2.5: https://viso.ai/deep-learning/deep-neural-network-three-popular-types/
Figure 2.7: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/
Figure 2.8: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/
Figure 2.13: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253
Figure 2.15: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
Figure 2.5: https://viso.ai/deep-learning/deep-neural-network-three-popular-types/