A Project Report
on

IMAGE DEBLURRING

submitted in partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

17WH1A0582	**Ms. S. SIRISHA**
17WH1A0593	**Ms. K. SAI SINDHU**
17WH1A05B7	**Ms. S. SREEYA**

under the esteemed guidance of

Ms. E. G. PADMAVATI
Assistant Professor

Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTU, Hyderabad)
Bachupally, Hyderabad – 500090

June, 2021

DECLARATION

We hereby declare that the work presented in this project entitled **"IMAGE DEBLURRING"** submitted towards completion of Project Work in IV year of B.Tech CSE at 'BVRIT HYDERABAD College of Engineering For Women', Hyderabad is an authentic record of our original work carried out under the guidance of Ms. E. G. Padmavati, Assistant Professor, Department of CSE.

Sign. with date:
Ms. S.SIRISHA
(17WH1A0582)

Sign. with date:
Ms. K. SAI SINDHU
(17WH1A0593)

Sign. with date:
Ms. S. SREEYA
(17WH1A05B7)

BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTU, Hyderabad)
Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering

Certificate

This is to certify that the Project Work report on "**IMAGE DEBLURRING**" is a bonafide work carried out by Ms. S.SIRISHA (17WH1A0582) ; Ms. K. SAI SINDHU (17WH1A0593) ; Ms. S. SREEYA (17WH1A05B7) in the partial fulfillment for the award of B.Tech. degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department
Dr. K. Srinivasa Reddy
Professor and HoD,
Department of CSE

Guide
Ms. E. G. Padmavati
Assistant Professor

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal**, **BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. K. Srinivasa Reddy, Professor and HoD** Department of CSE, **BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Ms. E. G. Padmavati, Assistant Professor**, Department of CSE**, BVRIT HYDERABAD College of Engineering for Women** for her constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of **CSE** Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Ms. S.SIRISHA

(17WH1A0582)

Ms. K. SAI SINDHU

(17WH1A0593)

Ms. S. SREEYA

(17WH1A05B7)

ABSTRACT

Image Deblurring aims to recover the clear image solely from an input blurry image, and is a challenging ill-posed problem. This is done using a novel ranking convolutional neural network (Ranking-CNN). In Ranking-CNN, a novel ranking layer is proposed to extend the structure of CNN so that the statistical and structural attributes of blurry images can be simultaneously captured. By training Ranking-CNN in a well-designed manner, powerful blurr-relevant features can be automatically learned from massive blurry image patches.Based on these features, blurry effect can be effectively removed by using a blurr density prediction model trained through the random forest regression.

To facilitate the training process, a two-stage training scheme is adopted. That is, first the problem is converted to a 10-category classification problem and trained using a Ranking-CNN model for classification. After that, the output layer is discarded and the output of the second fully-connected layer is used as features for training a random forest regression to predict the transmission. In this manner, the training process of the Ranking-CNN is easier and the learned features,when combined with the random forest regressor, still have impressive performance in image deblurring.

LIST OF FIGURES

TABLE OF CONTENTS

1. INTRODUCION

Image deblurring is the process where the old images and the low-quality images or the images which are destroyed can be recycled and made high quality by using different types of software. This is very much helpful in our modern world.

1.1 Objective

Deblurring an image is a challenging ill-posed problem. This can be done using deep learning and machine learning algorithms. The system will allow the user to upload a blur image and give the clear image as an output. This is done using Ranking Convolutional neural networks and Random forest Regression algorithms to learn both statistical and structural attributes of an image.

1.2 Methodology

To train the model a large dataset of blur images is required. In this section methodology followed is discussed in detail

1.2.1 Dataset

Proper and large dataset of blur images is needed to train the model. The dataset for the experiment is downloaded from kaggle. The dataset used is Syn Dataset(Synthetic dataset of low fidelity images) which contains 12000 blur images generated using 1000 clear images. These images are used to train the model.

1.2.2 The proposed Ranking CNN Model

CNN Architectures vary with the type of problem at hand. The proposed model consists of ten different layers. The first layer is the input layer, which includes the RGB channels of a color image patch with resolution 20 × 20. The second layer is a convolutional layer, where the R, G, B maps are convolved with 5 × 5 convolutional kernels to generate 32 feature maps with resolution 16 × 16. The third layer is a max pooling layer that sub-samples the input feature map over each 2 × 2 non-overlapping window. The fourth layer is the ranking layer, which operates separately on each input feature map. It sorts all the elements in an input feature map and outputs a ranked feature map with the same dimension. Note that the elements in the ranked feature map are in ascending order from left-top to right-bottom. The fifth layer is a convolutional layer, which includes 32 feature maps and the convolutional kernel size is 3 × 3.The sixth layer is also a convolutional layer same with the fifth layer. The seventh layer is another max pooling layer which is the same as the third layer. After this layer, finally obtain 32 feature maps of size 2 × 2. The eighth layer, ninth layer (each with 64

features) and the output layer (with 10 output values) are all fully-connected layers. In this Ranking-CNN rectified linear unit (ReLU) activation function is used for all convolutional layers and the first fully-connected layer.After that, the output layer is discarded and the output of the second fully-connected layer is used as features for training a random forest regressor to predict the transmission. In this manner, the training process of the Ranking-CNN is easier and the learned features, when combined with the random forest regressor, still have impressive performance in image deblurring.

There are five CNN algorithm steps

Convolution:

The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information. In the case of a CNN, the convolution is performed on the input data with the use of a filter or kernel to then produce a feature map.

Here are the three elements that enter into the convolution operation:

- Input image
- Feature detector
- Feature map

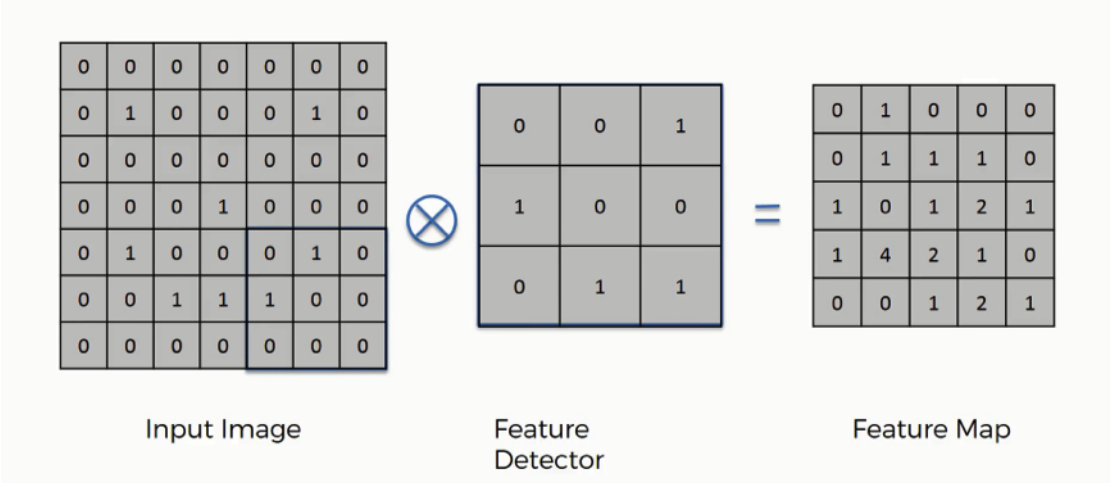

Fig 1.2.2.1 Convolution in CNN

Flattening:

Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector.

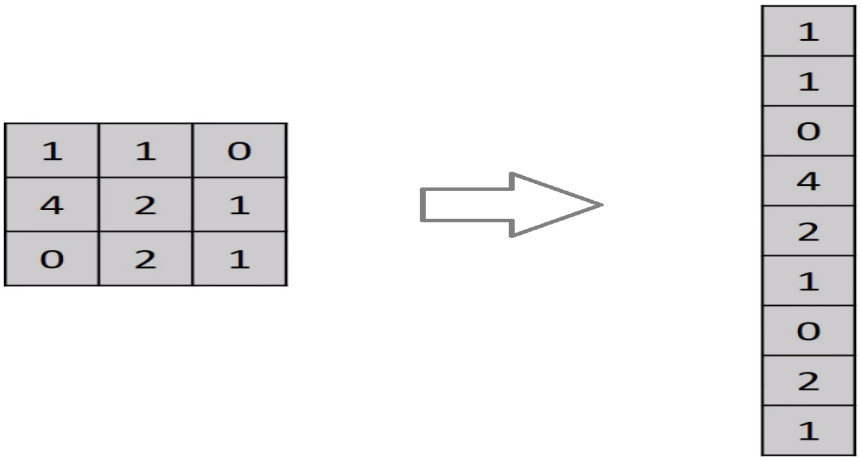

Fig 1.2.2.2 Flattening in CNN

Max Pooling:

Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.),reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned**.**

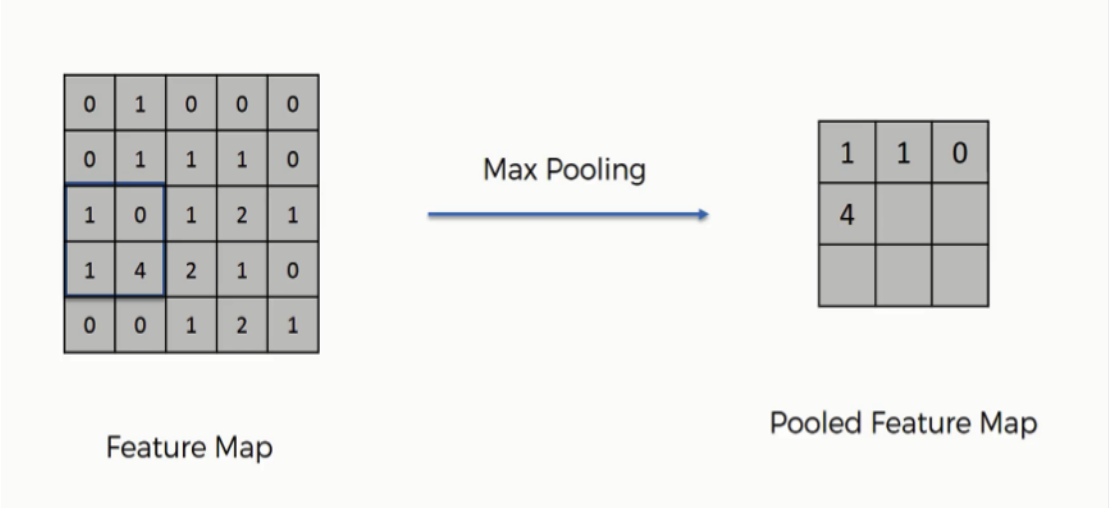

Fig 1.2.2.3 Max Pooling in CNN

Ranking layer:

The ranking layer operates separately on each input feature map. Feature map is actually a 2D matrix which is turned into a 1D vector by sampling elements column-wise so as to provide a better viewing experience.It sorts all the elements in

an input feature map and outputs a ranked feature map with the same dimension. The elements in the ranked feature map are in ascending order from left-top to right-bottom.

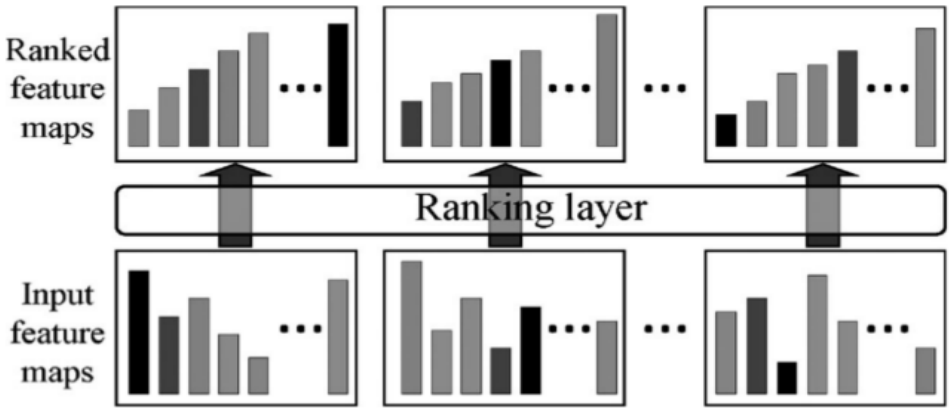

Fig 1.2.2.4 Ranking Layer

Full Connection:

At the end of a CNN, the output of the last Pooling Layer acts as an input to the so-called Fully Connected Layer. There can be one or more of these layers ("fully connected" means that every node in the first layer is connected to every node in the second layer). There are three layers in the full connection step:

- Input layer
- Fully-connected layer
- Output layer

2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

2.1 Requirements Gathering

2.1.1 Software Requirements

Programming Language : Python 3.6

Graphical user interface : Tkinter

Dataset : Syn Dataset(Synthetic dataset of low fidelity images)

Packages : Numpy, Tensorflow, Scikit-image, Matplotlib

Tool : Python IDE

2.1.2 Hardware Requirements

Operating System : Ubuntu 16.04 / Windows 10

Processor : Intel Core i3-2348M

CPU Speed : 2.30GHz

Memory : 2GB (RAM)

2.2 Technologies Description

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development,

The ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Tkinter:

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Syn Dataset:

The dataset for this experiment is downloaded from kaggle. It is a Synthetic dataset of low fidelity images. It consists of 12000 blur images generated from 1000 clear images.

Tensorflow:

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

Numpy:

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Scikit-image:

Scikit-image is an open-source image processing library for the Python programming language. It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

3. DESIGN

3.1 Introduction

In real world scenarios, small particles suspended in the atmosphere (e.g droplets and dust) often scatter the light. As a consequence, the clarity of an image would be seriously degraded, which may decrease the performance of many multimedia processing systems, e.g., content-based image retrieval. Image enhancement methods can only alleviate this problem slightly. It is still helpful to develop effective deblurring methods to recover the clear image from an input blur image. It is widely acknowledged that a blur image can be regarded as a convex combination of scene radiance and atmospheric light. The combination coefficient is often called the transmission. As a result, the task of image deblurring can be formulated as recovering the scene radiance from a blur image by estimating the atmospheric light and the transmission.

3.2 Architecture diagram

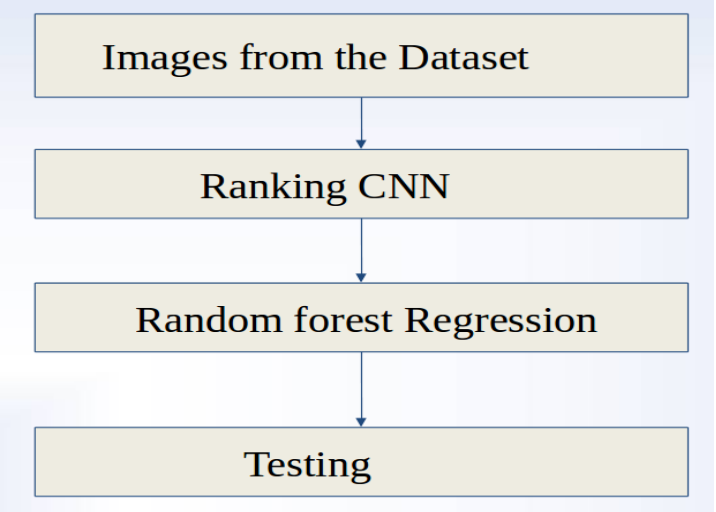

Fig 3.2.1 Architecture Diagram

3.2.1 Dataset

The dataset used for this experiment is Syn Dataset(Synthetic dataset of low fidelity images). The dataset is downloaded from kaggle. It consists of 12000 blur images. These blur images are generated using 1000 clear images. This dataset is used to train the model and to learn the features.

3.2.2 Ranking CNN

There are two types of attributes that influence the performance of transmission estimation, including statistical attributes (e.g., dark channel prior and color-lines prior) and structural attributes (e.g., boundaries).CNN performs impressively on capturing the structural attributes due to the usage of convolutional layers, while it often lacks the ability to extract statistical attributes. To enhance the ability in extracting blur-relevant features, a ranking layer is added to the classical CNN so as to construct a novel Ranking-CNN.Ranking-CNN has ten layers. The first layer is the input layer, which includes the RGB channels of a color image patch with resolution 20 × 20. The second layer is a convolutional layer, where the R, G, B maps are convolved with 5 × 5 convolutional kernels to generate 32 feature maps with resolution 16 × 16. The third layer is a max pooling layer that sub-samples the input feature map over each 2 × 2 non-overlapping window. The fourth layer is the ranking layer, which operates separately on each input feature map. It sorts all the elements in an input feature map and outputs a ranked feature map with the same dimension. The elements in the ranked feature map are in ascending order from left-top to right-bottom. The fifth layer is a convolutional layer, which includes 32 feature maps and the convolutional kernel size is 3 × 3. The sixth layer is also a convolutional layer same with the fifth layer. The seventh layer is another max pooling layer which is the same as the third layer. After this layer, 32 feature maps of size 2 × 2 are obtained. The eighth layer, ninth layer (each with 64 features) and the output layer (with 10 output values) are all fully-connected layers. In Ranking-CNN, rectified linear unit (ReLU) activation function is used for all convolutional layers and the first fully-connected layer.

3.2.3 Random Forest Regression

The model can be trained in an end-to-end network that predicts the transmission by replacing the output layer with a linear regression layer. However, since the output of the linear regression layer is only a variable varying between (0, 1], it is difficult to effectively train the deep network. To facilitate the training process, a two-stage training scheme is adopted. That is, first convert the problem to a 10-category classification problem and train a Ranking-CNN model for classification. After that, the output layer is discarded and the output of the second fully connected layer is used as features for training a random forest regressor to predict the transmission. In

this manner, the training process of the Ranking-CNN is easier and the learned features, when combined with the random forest regressor, still have impressive performance in image dehazing. Random Forest is used to learn a regression model between the transmission t and the blur-relevant features. Random forest model has 200 trees and each tree randomly selects 1/3 feature dimensions. For efficiency, 1/100 blur image patches to train the regression model.

3.2.4 Testing

After building the model it is tested by uploading a blur image from the graphical user interface then it is converted into a clear image.

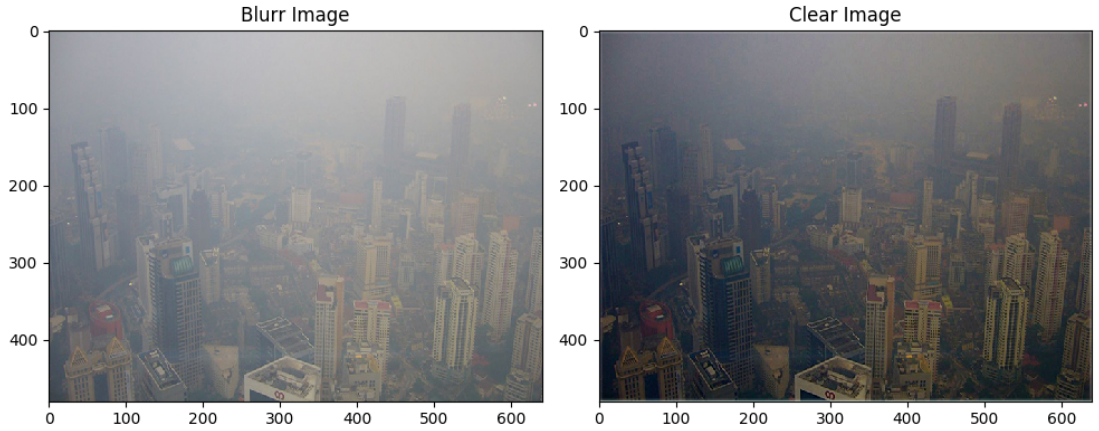

Fig 3.2.4.1 Sample input and output

4. IMPLEMENTATION

4.1 Code

CNNDeblurr.py

```
from tkinter import *
import tkinter
from tkinter import filedialog
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename
import tensorflow as tf
import numpy as np
from tensorflow.keras.layers import *
from PIL import Image
import matplotlib.pyplot as plt
from DataReader import DataReader
import cv2
from math import log10, sqrt
import os
from tensorflow.python.framework import ops
main = tkinter.Tk()
main.title("Image Deblurring")
main.geometry("1200x1200")



global deblur_RGB
global saver
global RGB
global MAX



global filename

def generateCNNModel(RGB):
```

```
    cnn1 =
Conv2D(3,1,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(RGB)#layer 1
    cnn2 =
Conv2D(3,3,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(cnn1)#layer 2
    encoder1 = tf.concat([cnn1,cnn2],axis=-1) #concatenate layer1 and layer2 to from
residual network
    cnn3 =
Conv2D(3,5,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(encoder1)
    encoder2 = tf.concat([cnn2,cnn3],axis=-1)#concatenate layer2 and layer3 to from
residual network
    cnn4 =
Conv2D(3,7,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(encoder2)
    decoder = tf.concat([cnn1,cnn2,cnn3,cnn4],axis=-1)
    cnn5 =
Conv2D(3,3,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(decoder)
    MAX = cnn5 #max layer
    dehaze_cnn = ReLU(max_value=1.0)(tf.math.multiply(MAX,RGB) - MAX + 1.0)
#replace pixels intensity
    return dehaze_cnn

def loadModel():
    global dehazed_RGB
    global saver
    global RGB
```

```
    global MAX
    dr = DataReader()  #class to read training images
    ops.reset_default_graph() #reset tensorflow graph
    trainImages, testImages =
dr.readImages("/home/goutham/Desktop/Datset/data/orig_images/","/home/goutham/
Desktop/Datset/data/hazy_images/") #reading normal and haze image to generate
tensorflow CNN object
    trainData, testData, itr = dr.generateTrainTestImages(trainImages,testImages)
    print("cnn dehaze ", trainData)
    next_element = itr.get_next()
    RGB = tf.placeholder(shape=(None,480, 640,3),dtype=tf.float32)
    MAX = tf.placeholder(shape=(None,480, 640,3),dtype=tf.float32)
    deblur_RGB = generateCNNModel(RGB) #loading and generating model

    trainingLoss = tf.reduce_mean(tf.square(deblur_RGB-MAX)) #optimizations
    optimizerRate = tf.train.AdamOptimizer(1e-4)
    trainVariables = tf.trainable_variables()
    gradient = optimizerRate.compute_gradients(trainingLoss,trainVariables)
    clippedGradients = [(tf.clip_by_norm(gradients,0.1),var1) for gradients,var1 in
gradient]
    optimize = optimizerRate.apply_gradients(gradient)

    saver = tf.train.Saver()
    #pathlabel.config(text='CNN model loaded')

def PSNR(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if(mse == 0):
        return 100
    max_pixel = 255.0
    psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr,mse

def SNR(a, axis=None, ddof=0):
```

```
    a = np.asanyarray(a)
    m = a.mean(axis)
    sd = a.std(axis=None, ddof=ddof)
    return np.where(sd == 0, 0, m/sd)

#function to allow user to upload images directory
def uploadImage():
    #text.delete('1.0', END)
    global filename
    filename = askopenfilename(initialdir = "TestImages")
    #pathlabel.config(text=filename)
    with tf.Session() as session:
        saver.restore(session,'/home/goutham/Desktop/Deblurr/CNNData/data')
        img = Image.open(filename)
        img = img.resize((640, 480))
        img = np.asarray(img) / 255.0
        img = img.reshape((1,) + img.shape)
        deblurImage = session.run(dehazed_RGB,feed_dict={RGB:img,MAX:img})
        print(deblur[0])
        orig = cv2.imread(filename)
        orig = cv2.resize(orig,(640, 480),interpolation = cv2.INTER_CUBIC)
        psnr,mse = PSNR(orig,deblur[0].astype('float32') * 255)
        snr = SNR(deblur[0].astype('float32') * 255)
        #text.insert(END,'CNN PSNR : '+str(psnr)+"\n")
        #text.insert(END,'CNN SNR : '+str(snr)+"\n")
        #text.insert(END,'CNN MSE : '+str(mse)+"\n")
        fname = os.path.basename(filename)
        f = open("values.txt", "w")
        f.write(fname+","+str(psnr)+","+str(snr)+","+str(mse))
        f.close()
        figure, axis = plt.subplots(nrows=1, ncols=2,figsize=(10,10))
        axis[0].set_title("Blur Image")
        axis[1].set_title("Clear Image")
        axis[0].imshow(img[0])
```

```
        axis[1].imshow(deblurImage[0])
        figure.tight_layout()
        plt.show()




def close():
    main.destroy()

font = ('times', 30, 'bold')
f1 = ('ties', 25)
title = Label(main, text='IMAGE DEBLURRING')
title.config(bg='antiquewhite', fg='black')
title.config(font=font)
title.config(height=5, width=90)
title.place(x=50,y=5)
#title.config(anchor = CENTER)
canvas = Canvas(main, width = 500, height = 450)
canvas.pack()
img = PhotoImage(file="/home/goutham/Downloads/empire_state.png/")
canvas.create_image(10,50, anchor=NW, image=img)
canvas.place(x = 250, y = 400)
t = Text(main, height = 3, width = 95)
t.place(x = 190, y = 200)
t.config(font = f1)
t.config(bg = 'papayawhip')
te = " Deblurring is the process of removing blurring artifacts from images. It is used
in medical \n imaging, remote sensing etc."
t.insert(END,te)
font1 = ('times', 20, 'bold')
upload = Button(main, text="Build Model", command=loadModel)
upload.place(x=1100,y=500)
upload.config(font=font1)
```

```
#pathlabel = Label(main)
#pathlabel.config(bg='white', fg='black')
#pathlabel.config(font=font1)
#pathlabel.place(x=380,y=100)

dcpButton = Button(main, text="Upload Blur Image", command=uploadImage)
dcpButton.place(x=1100,y=600)
dcpButton.config(font=font1)

exitButton = Button(main, text="Exit", command=close)
exitButton.place(x=1100,y=700)
exitButton.config(font=font1)

#font1 = ('times', 12, 'bold')
#text=Text(main,height=10,width=150)
#scroll=Scrollbar(text)
#text.configure(yscrollcommand=scroll.set)
#text.place(x=10,y=350)
#text.config(font=font1)



main.config(bg='antiquewhite')
main.mainloop()
loadModel()
```

CNNModel.py

```
from DataReader import DataReader
import tensorflow as tf
import numpy as np
import os, shutil
from tensorflow.keras.layers import *
import glob
import random
```

```
from PIL import Image
import cv2

import matplotlib.pyplot as plt
import matplotlib as mpl

n_epochs = 20
batch_size = 8
learning_rate = 1e-4
weight_decay = 1e-4

def showImage(x):
  x = np.asarray(x*255,dtype=np.int32)
  plt.figure()
  plt.imshow(x)
  plt.show()

def generateCNNModel(RGB):
    cnn1 =
Conv2D(3,1,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(RGB)#layer 1
    cnn2 =
Conv2D(3,3,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(cnn1)#layer 2
    dense1 = tf.concat([cnn1,cnn2],axis=-1) #concatenate layer1 and layer2 to from
residual network
    cnn3 =
Conv2D(3,5,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(dense1)
    dense2 = tf.concat([cnn2,cnn3],axis=-1)#concatenate layer2 and layer3 to from
residual network
```

```
    cnn4 =
Conv2D(3,7,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(dense2)
    decoder = tf.concat([cnn1,cnn2,cnn3,cnn4],axis=-1)
    cnn5 =
Conv2D(3,3,1,padding="same",activation="relu",use_bias=True,kernel_initializer=tf.
initializers.random_normal(stddev=0.02),
                kernel_regularizer=tf.keras.regularizers.l2(1e-4))(decoder)
    MAX = cnn5 #max layer
    deblur_cnn = ReLU(max_value=1.0)(tf.math.multiply(MAX,RGB) - MAX + 1.0)
#replace pixels intensity
    return deblur_cnn

dr = DataReader()
np.random.seed(9999)
tf.reset_default_graph()
hr_train_data,lr_val_data =
dr.readImages('/home/goutham/Desktop/Datset/data/orig_images/','/home/goutham/D
esktop/Datset/data/hazy_images/')
#lr_val_data = dr.readImages('data/hazy_images')
train_init_op, val_init_op, iterator =
dr.generateTrainTestImages(hr_train_data,lr_val_data)
next_element = iterator.get_next()

X = tf.placeholder(shape=(None,64,64,3),dtype=tf.float32)
Y = tf.placeholder(shape=(None,64,64,3),dtype=tf.float32)
reconstruct_X = generateCNNModel(X)

loss = tf.reduce_mean(tf.square(reconstruct_X-Y))
optimizer = tf.train.AdamOptimizer(learning_rate)
trainable_variables = tf.trainable_variables()
gradients_and_vars = optimizer.compute_gradients(loss,trainable_variables)
```

```
clipped_gradients = [(tf.clip_by_norm(gradient,0.1),var) for gradient,var in
gradients_and_vars]
optimizer = optimizer.apply_gradients(gradients_and_vars)

saver = tf.train.Saver()
load_path = None

with tf.device('/cpu:0'):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for epoch in range(n_epochs):
            sess.run(train_init_op)
            batches = len(hr_train_data) // batch_size
            epoch_loss = 0.0
            for batch in range(batches):
                batch_data = sess.run(next_element)
                batch_loss, _ =
sess.run([loss,optimizer],feed_dict={X:batch_data[0],Y:batch_data[1]})
                epoch_loss += batch_loss / float(batches)
                if batch % 1000 == 0:
                  print("Training loss at batch %d: %f\n"%(batch,batch_loss))
            train_loss = epoch_loss

            sess.run(val_init_op)
            batches= len(lr_val_data) // batch_size
            epoch_loss = 0.0
            for batch in range(batches):
                batch_data = sess.run(next_element)
                batch_loss = sess.run(loss,feed_dict={X:batch_data[0], Y:batch_data[1]})
                epoch_loss += batch_loss / float(batches)
                if batch % 100 == 0:
                    print("Validation loss at batch %d: %f\n"%(batch,batch_loss))
                    for j in range(-2 + batch_size//2):
                        x = batch_data[0][j].reshape((1,)+batch_data[0][j].shape)
```

```
                y = batch_data[1][j].reshape((1,)+batch_data[1][j].shape)
                reconstruct_x = sess.run(reconstruct_X,feed_dict={X:x,Y:y})
                print("Image Number: %d\n"%(j))
                print(str(reconstruct_x[0]))
                #showImage(x[0])
                #showImage(y[0])
                #showImage(dehazed_x[0])
        val_loss = epoch_loss

        saver.save(sess,'./Extension_models/model_checkpoint_' + str(epoch) + '.ckpt')

        print("Epoch %d\nTraining loss: %f\nValidation loss:
%f\n\n"%(epoch,train_loss,val_loss))
```

DataReader.py

```
import os
import glob
import random
import tensorflow as tf
from tensorflow.python.data.ops.iterator_ops import Iterator as Iterator
class DataReader:

   def readimage(self,img_path):
      img = tf.io.read_file(str(img_path))
      img = tf.image.decode_jpeg(img,channels=3)
      img = tf.image.resize(img,(480,640))
      img = img / 255.0
      return img

   def readimage1(self,img_path):
      img = tf.io.read_file(img_path)
      img = tf.image.decode_jpeg(img,channels=3)
      img = tf.image.resize(img,(64,64))
      img = img / 255.0
```

```
        return img

    def generateTrainTestImages(self,trainImages,testImages):
        trainblur = tf.data.Dataset.from_tensor_slices([img[0] for img in
trainImages]).map(lambda name:self.readimage(name))
        trainNormal = tf.data.Dataset.from_tensor_slices([img[1] for img in
trainImages]).map(lambda name:self.readimage(name))
        trainData =
tf.data.Dataset.zip((trainblur,trainNormal)).shuffle(100).repeat().batch(8)
        testblur = tf.data.Dataset.from_tensor_slices([img[0] for img in
testImages]).map(lambda name:self.readimage(name))
        testNormal = tf.data.Dataset.from_tensor_slices([img[1] for img in
testImages]).map(lambda name: self.readimage(name))
        testData =
tf.data.Dataset.zip((testblur,testNormal)).shuffle(100).repeat().batch(8)
        itr =
Iterator.from_structure(tf.compat.v1.data.get_output_types(trainData),tf.compat.v1.da
ta.get_output_shapes(trainData))
        #print("train data", trainData)
        training = itr.make_initializer(trainData)
        testing = itr.make_initializer(testData, name = None)
        print("training", training)
        print("traindata", trainData)
        return training, testing, itr

    def generateExtensionTrainTestImages(self,trainImages,testImages):
        trainblur = tf.data.Dataset.from_tensor_slices([img[0] for img in
trainImages]).map(lambda name: self.readimage1(name))
        trainNormal = tf.data.Dataset.from_tensor_slices([img[1] for img in
trainImages]).map(lambda name: self.readimage1(name))
        trainData =
tf.data.Dataset.zip((trainblur,trainNormal)).shuffle(100).repeat().batch(8)
        testblur = tf.data.Dataset.from_tensor_slices([img[0] for img in
testImages]).map(lambda name: self.readimage1(name))
```

```
        testNormal = tf.data.Dataset.from_tensor_slices([img[1] for img in
testImages]).map(lambda name: self.readimage1(name))
        testData =
tf.data.Dataset.zip((testblur,testNormal)).shuffle(100).repeat().batch(8)
        itr =
tf.data.Iterator.from_structure(trainData.output_types,trainData.output_shapes)
        training = itr.make_initializer(trainData)
        #print(training)
        testing = itr.make_initializer(testData)
        #print(testing)
        return training, testing, itr

    def readImages(self,train_images,test_images):
        training_path = glob.glob(train_images + "/*.jpg")
        total_images = len(training_path)
        random.shuffle(training_path)
        trainingKeys = training_path[:int(0.90*total_images)]
        testingKeys = training_path[int(0.90*total_images):]
        imageDict = {}
        for name in trainingKeys:
            imageDict[name] = 'training'
        for name in testingKeys:
            imageDict[name] = 'testing'
        trainImages = []
        testImages = []
        #print(imageDict)
        test_path = glob.glob(train_images + "/*.jpg")
        #print(test_path)
        for name in test_path:
            #print(name)
            imgName = name.split('/')[-1]
            img = os.path.splitext(os.path.basename(name))[0]
            imgpath = train_images + img+".jpg"
            if(imageDict[imgpath] == 'training'):
```

```
                trainImages.append([name,imgpath])
            else:
                testImages.append([name,imgpath])
        return trainImages, testImages
#d = DataReader()
#trainImages, testImages =
d.readImages("/home/goutham/Desktop/Dataset/data/orig_images/","/home/goutham/
Desktop/Dataset/data/hazy_images/")
#a,b =
d.generateTrainTestImages("/home/goutham/Desktop/Dataset/data/orig_images/","/ho
me/goutham/Desktop/Dataset/data/hazy_images/")
#a,b,c = d.generateTrainTestImages(trainImages, testImages)
```

4.2 Output

Fig 4.2.1 GUI to upload the image

Initially in Graphical user interface a window appears to upload the blur image. Initially the model is built by clicking the build model button.Later the image can be uploaded by clicking the Upload Blur Image button. After uploading the blur image we get a new window showing both the clear image and blur image uploaded.

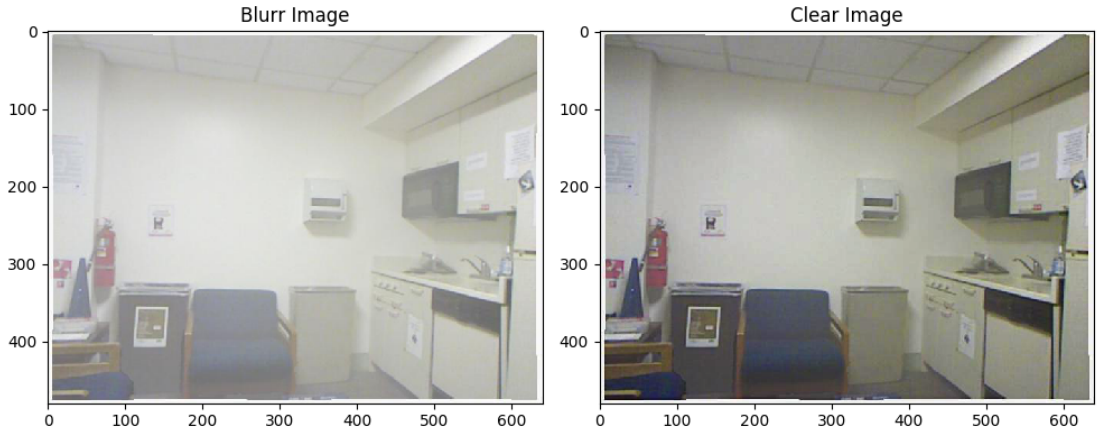

Fig 4.2.2 Output

5. CONCLUSION AND FUTURE SCOPE

Conclusion:

Equipped with the novel ranking layer, Ranking-CNN captures the structural and statistical features simultaneously. Based on the learned features, a regression model is further trained to predict haze density for effective haze removal. Experimental results show that Ranking-CNN features are effective.

Future scope:

The design can be improved for reducing the time taken for execution of code and improved PSNR and MSE.This project can be extended to deblur the videos of cctv cameras.

6. REFERENCES

[1] Mahdi S. Hosseini, and Konstantinos N. Plataniotis “Convolutional Deblurring for Natural Imaging” (Base paper).

[2] J. Sun, W. Cao, Z. Xu, and J. Ponce, “Learning a convolutional neural network for non-uniform motion blur removal,” in Proc. CVPR, 2015, pp. 769–777.

[3] S. M. C. Nascimento, K. Amano, and D. H. Foster, “Spatial distributions of local illumination color in natural scenes,” Vis. Res., vol. 120, pp. 39–44, Mar. 2016.

[4] W. Ren, S. Liu, H. Zhang, J. Pan, X. Cao, and M.-H. Yang, “Single image dehazing via multi-scale convolutional neural networks,” in Proc. Eur. Conf. Comput. Vis. Amsterdam, The Netherlands: Springer, 2016, pp. 154–169.

[5] S. M. C. Nascimento, K. Amano, and D. H. Foster, “Spatial distributions of local illumination color in natural scenes,” Vis. Res., vol. 120, pp. 39–44, Mar. 2016.

[6] O. Whyte, J. Sivic, and A. Zisserman, “Deblurring shaken and partially saturated images,” Int. J. Comput. Vis., vol. 110, no. 2, pp. 185–201, 2014.

[7] C.-T. Shen, W.-L. Hwang, and S.-C. Pei, “Spatially-varying out-of- focus image deblurring with L1-2 optimization and a guided blur map,” in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), Mar. 2012, pp. 1069–1072

[8] X. Zhu and P. Milanfar, “Removing atmospheric turbulence via space invariant deconvolution,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 35, no. 1, pp. 157–170, Jan. 2013.