National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Khushbu Rajpara
X20175671

School of Computing
National College of Ireland

Supervisor: Prof. Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Khushbu Rajpara

Student ID: X20175671

Programme: MSc Data Analytics **Year:** 2021-22

Module: MSc Research Project

Lecturer: Prof. Vladimir Milosavljevic

Submission Due Date: 16th December, 2021

Project Title: Single Image Super Resolution using multiple Deep Convolution Neural Networks

Word Count: **1156** **Page Count: 14**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Khushbu Rajpara

Date: 16th December, 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Khushbu Rajpara
X20175671

1 Introduction

The given configuration manual interprets the requirements for implementing the system designed for enhancing the image resolution with the help of Deep neural network models. Moreover, the hardware and software requirements for successful implementation of the project will be explained meticulously in this manual.

2 System Configuration

Mentioned below are the hardware and software configurations used for project implementation.

2.1 Hardware Requirements:

Figure 1 Hardware Requirments

2.2 Software Requirements

This chapter provides requirements for the software used in this study as well as information about the software that was used to execute the model. The personal computer did not have enough resources to implement the project. That being the case, project is implemented on the Google Colab.

2.2.1 Google colaboratory

For the model's implementation, the Google infrastructure's computing, also known as Google colab, is utilized. All the necessary libraries have been installed, and the models have been created in Google Colab.

The dataset is uploaded to Google Drive and then linked to Google Colab using the code below. Apart from that, there are direct options for mounting our hard drive inside the Google Col

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Figure 2 Google Colab

This project is built with Google Colab Pro+, which runs the software in the background, allowing the GPU to stay connected for a longer period of time.

3 Project Implementation

The proposed research project is run on a GPU server on Google Colab with the aim of image super resolution. This study is divided into four sections, each of which performs trials using one of the four current models. This study examines all areas of data analytics, including data collection and cleansing, as well as model implementation and evaluation.

3.1 Data Collection:

The data that was utilized to conduct this research was obtained from Kaggle. There are three sub folders in the data and it contains high resolution images as well as low resolution images. The following image is showing the dataset in the system.

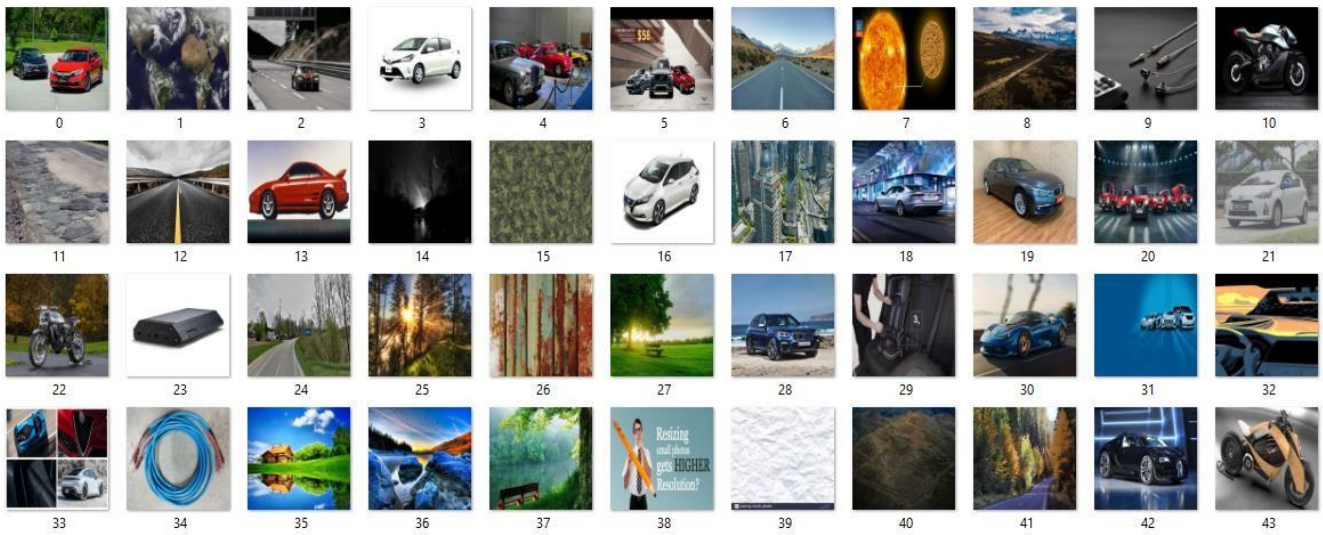

Figure 3 Data Collection

3.2 Data Preparation

As this dataset is relatively small compared to “DIV2k” although the data in this dataset is cleaned.

The below is the code for getting the file into the proper format. Firstly, the dataset is download from the Kaggle then it was uploaded into google drive.

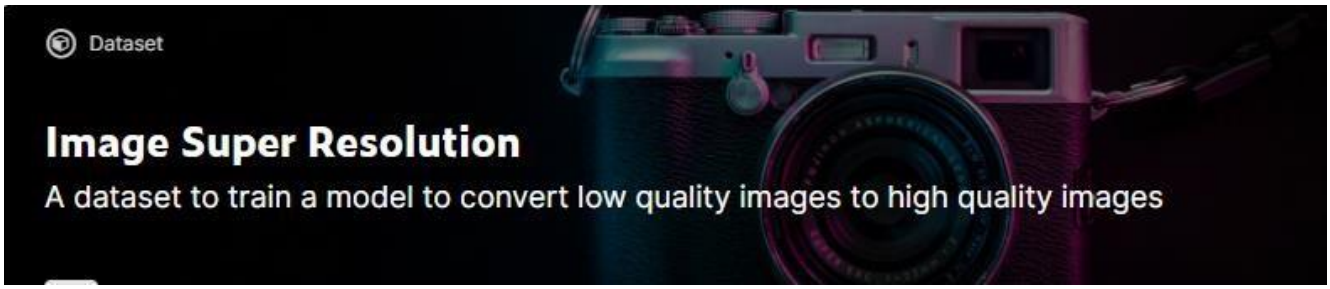

Figure 4 Dataset available on Kaggle

The following cell block is building a dataset in form of 2 arrays one consisting of High-resolution image and another of Low-resolution image. Then this array is used for splitting the dataset into 3 parts.

```
# function to get the files in proper alphanumeric order
def sort_file(data):
    convert_t_d = lambda text: int(text) if text.isdigit() else text.lower()
    alphanum_key = lambda key: [convert_t_d(c) for c in re.split('([0-9]+)',key)]
    return sorted(data,key = alphanum_key)

# defining the size of the image
SIZE = 256

#segregating high resolution image
h_res_img = []
#getting the dataset from given location
path = '/content/drive/MyDrive/FINAL_CODE/final_code_s/dataset/Raw Data/high_res'
files = os.listdir(path)
files = sort_file(files)
#Tqdm is a Python library used to display smart progress bars that show the progress of code execution.
for i in tqdm(files):
    if i == '855.jpg':
        break
    else:
        img = cv2.imread(path + '/'+i,1)
        # open cv reads images in BGR format so we have to convert_t_d it to RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        #resizing high resolution image into 256*256
        img = cv2.resize(img, (SIZE, SIZE))
        img = img.astype('float32') / 255.0
        #resizing image is append in the array which will be used for splitting
        h_res_img.append(img_to_array(img))

#segregating low resolution image
l_res_img = []
#getting the dataset from given location
path = '/content/drive/MyDrive/FINAL_CODE/final_code_s/dataset/Raw Data/low_res'
files = os.listdir(path)
#calling function
files = sort_file(files)
for i in tqdm(files):
    if i == '855.jpg':
        break
    else:
        img = cv2.imread(path + '/'+i,1)
        # open cv reads images in BGR format so we have to convert_t_d it to RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        #resizing image
        img = cv2.resize(img, (SIZE, SIZE))
        img = img.astype('float32') / 255.0
        l_res_img.append(img_to_array(img))
```

Figure 5 Data Preparation

3.3 Data Pre-Processing

The following cell is for visualizing the dataset. In that randomly four images select from the dataset and plot it as a output.

```
#plotting some images of the dataset
for i in range(4): # no. of images
    a = np.random.randint(0,855)
    plt.figure(figsize=(10,10))
    plt.subplot(1,2,1)
    plt.title('High Resolution Images', color = 'black', fontsize = 15)
    #h_res_img array which is defined in above cell
    plt.imshow(h_res_img[a])
    plt.axis('off')
    plt.subplot(1,2,2)
    plt.title('low Resolution Images ', color = 'black', fontsize = 15)
    plt.imshow(l_res_img[a])
    plt.axis('off')
```

Figure 6 Data Pre-processing

The below code splitting the dataset into training, Testing and validation. Brief explanation of used variable name is below.

TN_H_Img: high resolution images for training.

TN_L_Img: Low resolution images for training

V_H_image: high resolution images for validation

V_L_image: Low resolution images for validation

TS_H_Img: High resolution images for testing

TS_L_Img: Low resolution images for testing

This variable is used as input for the four model

3.4 Model Building

IMPORTING LIBRARIES

```
import os
import re
import shutil
import keras
import requests
import time
import tensorflow as tf
import cv2 as cv2
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
import numpy as np
np. random. seed(0)
from tqdm import tqdm
from scipy import ndimage, misc
from tensorflow.keras.preprocessing.image import img_to_array
from skimage.transform import resize, rescale
from tensorflow.keras.layers import Input, Dense ,Conv2D,MaxPooling2D ,Dropout
from tensorflow.keras.models import Model
from tensorflow.keras import regularizers
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Conv2DTranspose, UpSampling2D, add ,concatenate
from keras import layers
from keras.models import Sequential
from keras.preprocessing.image import img_to_array
from tensorflow.python.data.experimental import AUTOTUNE
```

Figure 8 Importing Libraries

3.4.1 CNN model

The layers are customized using up and down function, where down function creates a layer with Conv2D layer and given parameters such as numbers of filters, kernel size and whether to apply batch normalization, whereas the up function creates a layer using Conv2DTranspose and given parameters filters, kernel size and whether or not to add dropout layer. The model at

the end concatenates all layer to make a 2D convolutional layer model with Adam a s optimizer and loss is calculated using mean absolute error method.

```
#function that returns a sequential model with given configuaration of filters, kernel size and batch normalization
def down(filters , kernel_size, apply_batch_normalization = True):
    downsample = tf.keras.models.Sequential()
    downsample.add(layers.Conv2D(filters,kernel_size,padding = 'same', strides = 2))
    if apply_batch_normalization:
        downsample.add(layers.BatchNormalization())
    downsample.add(keras.layers.LeakyReLU())
    return downsample

#function that returns a sequential model with given filters, kernel size and droput
def up(filters, kernel_size, dropout = False):
    upsample = tf.keras.models.Sequential()
    upsample.add(layers.Conv2DTranspose(filters, kernel_size,padding = 'same', strides = 2))
    if dropout:
        upsample.dropout(0.2)
    upsample.add(keras.layers.LeakyReLU())
    return upsample
```

Figure 9 Functions of CNN Model

```
def model():
    #Downsampling
    #input size(256*256)
    inputs = layers.Input(shape= [SIZE,SIZE,3])
    d1 = down(128,(3,3),False)(inputs)#down the original size of input
    d2 = down(128,(3,3),False)(d1)
    d3 = down(256,(3,3),True)(d2)
    d4 = down(512,(3,3),True)(d3)
    d5 = down(512,(3,3),True)(d4)
    #Upsampling
    u1 = up(512,(3,3),False)(d5)#up the value of of original size
    u1 = layers.concatenate([u1,d4])
    u2 = up(256,(3,3),False)(u1)
    u2 = layers.concatenate([u2,d3])
    u3 = up(128,(3,3),False)(u2)
    u3 = layers.concatenate([u3,d2])
    u4 = up(128,(3,3),False)(u3)
    u4 = layers.concatenate([u4,d1])
    u5 = up(3,(3,3),False)(u4)
    u5 = layers.concatenate([u5,inputs])
    output = layers.Conv2D(3,(2,2),strides = 1, padding = 'same')(u5)
    return tf.keras.Model(inputs=inputs, outputs=output)
```

Figure 10 Down Sampling

```
Mycnn = model()
#print model summary in table form
Mycnn.summary()
#plotting model summary in flow diagram
plot_model(Mycnn, to_file ='/content/drive/MyDrive/FINAL_CODE/final_code_s/super_res.png',show_shapes=True)
#Compile the model using adam optimizer of keras library and loss variable print the loss of model during training and "acc" pri
nt the accuracy
Mycnn.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'mean_absolute_error', metrics = ['acc'])
```

Figure 11 CNN Model

```
start_time = time.time()
history=Mycnn.fit(TN_L_Img, TN_H_Img, epochs = 30, batch_size = 1,
          validation_data = (V_L_image,V_H_image))
total_time = round((time.time() - start_time), 4)
#print total training of the model
print("Computational time in seconds is " + str(total_time))
#store loss of model
training_loss = history.history['loss']
test_loss = history.history['val_loss']
#store accuracy of model
training_acc = history.history['acc']
testing_acc = history.history['val_acc']
#count the number of epoch
epoch_count = range(1, len(training_loss) + 1)

# Visualize loss history
plt.plot(epoch_count, training_loss, 'r-')
plt.plot(epoch_count, test_loss, 'b-')
plt.title('CNN model Loss')
plt.legend(['Train Loss', 'Validation Loss'], loc='upper left')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show();

#Visualize Accuracy history
plt.plot(epoch_count, training_acc, 'r-')
plt.plot(epoch_count, testing_acc, 'b-')
plt.title('CNN model Accuracy')
plt.legend(['Train Acuuracy', 'Validation Accuracy'], loc='upper left')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show();
```

Figure 12 Model Evaluation

```
#save model weights in drive
Mycnn.save_weights('/content/drive/MyDrive/FINAL_CODE/final_code_s/Mycnn.h5')
```

The above cell is for saving the model

Figure 13 Model Saving

```
#Supplementary functions used in comparing the original images to predicted images

#function to calculate Peak Signal to Noise Ratio of the predicted image
def PSNR(y_true,y_pred):
    mse=tf.reduce_mean( (y_true - y_pred) ** 2 )
    return 20 * log10(1/ (mse ** 0.5))

#function to convert decimal value to logarithmic value with base 10
def log10(x):
    numerator = tf.math.log(x)
    denominator = tf.math.log(tf.constant(10, dtype=numerator.dtype))
    return numerator / denominator

#function to calculate the mean square error for a given predicted image
def pixel_MSE(y_true,y_pred):
    return tf.reduce_mean( (y_true - y_pred) ** 2 )
```

The above cell is for calculating PSNR and MSE

Figure 14 Supplementary functions used in CNN

```
# function to plot images to see the comparision of the high resolutin image, low resolution image and predicted image respectiv
ely
def plot_images(high,low,predicted):
    plt.figure(figsize=(15,15))
    plt.subplot(1,3,1)
    plt.title('High Resolution Images', color = 'blue', fontsize = 10)
    plt.imshow(high)
    plt.subplot(1,3,2)
    plt.title('Low Resoltion Images ', color = 'blue', fontsize = 10)
    plt.imshow(low)
    plt.subplot(1,3,3)
    plt.title('CNN Predicted Image ', color = 'Red', fontsize = 10)
    plt.imshow(predicted)

    plt.show()

#plotting image comparisons of 3 images
for i in range(1,4):
    predicted = np.clip(Mycnn.predict(TS_L_Img[i].reshape(1,SIZE, SIZE,3)),0.0,1.0).reshape(SIZE, SIZE,3)
    plot_images(TS_H_Img[i],TS_L_Img[i],predicted)
    print('PSNR',PSNR(TS_H_Img[i],predicted),'dB')
```

Figure 15 Functions to plot images

3.4.2 Multi scale learning

Here the usage of a Multiscale model is shown. The Multiscale model can have numerous branches which all together end up giving result to the final layer and we can have high level of precision in prediction of the data. In the following code two branches are created of a network both have similar configuration. Each branch has 3 layers, and each layer consists of a sequence of layer such that 2D convolutional layer is connected to dropout layer (which helps in preventing overfitting of model) which is then down sampled using 2D Maxpooling layer. Each h layer of branch is different from previous layer as number of neurons or filters are doubled from the previous layer. Both these branches are then concatenated to form a single network w which is then up sampled (increased dimensions) and given to 2D Convolutional layer which pr edicts data, in our case the predicted image.

```
# Creating a input layer with fixed parameters
input_shape =Input(shape=(256,256,3))
batch_size = 16
kernel_size = 3
dropout = 0.4
n_filters = 64
```

```
#function that return an upsampled layer which improves the quality of data, it consists of 2D convolutional layer connected to
leakyrelu
def Upsample_block(x,ch=256, k_s=3, st=1):
    x = tf.keras.layers.Conv2D(ch,k_s, strides=(st,st), padding='same')(x)
    x = tf.nn.depth_to_space(x, 2) # Subpixel pixelshuffler
    x = tf.keras.layers.LeakyReLU()(x)
    return x
```

Figure 16 Global variables of function The below code is for methods of multi-layer model .

```
# left branch of Y network
left_inputs = Input(shape=(256,256,3))
x = left_inputs
filters = n_filters
# 1 layers of Conv2D-Dropout-MaxPooling2D
# number of filters doubles after each layer (32-64-128)
for i in range(1):
    x = Conv2D(filters=filters,
               kernel_size=kernel_size,
               padding='same',
               activation='relu')(x)
    x = Dropout(dropout)(x)
    x = MaxPooling2D()(x)
    filters *= 2

# right branch of Y network
right_inputs = Input(shape=(256,256,3))
y = right_inputs
filters = n_filters
# 3 layers of Conv2D-Dropout-MaxPooling2D
# number of filters doubles after each layer (32-64-128)
for i in range(1):
    y = Conv2D(filters=filters,
               kernel_size=kernel_size,
               padding='same',
               activation='relu')(y)
    y = Dropout(dropout)(y)
    y = MaxPooling2D()(y)
    filters *= 2

y = concatenate([x, y])
y=Upsample_block(y)
outputs=Conv2D (3,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(y)
```

Figure 17 Multi-CNN Model

```
# Compiling the model with Adam as optimization technique and mean absolute error as the loss metric
Multi_scale_learning= Model([left_inputs, right_inputs], outputs)
Multi_scale_learning.summary()
Multi_scale_learning.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'mean_absolute_error',
              metrics = ['acc'])
plot_model(Multi_scale_learning, to_file='/content/drive/MyDrive/FINAL_CODE/Multi_scale_learning.png', show_shapes=True)
```

Figure 18 Compiling the model

```
# Training the model
import time
start_time = time.time()
history = Multi_scale_learning.fit([TN_L_Img, TN_L_Img],
             TN_H_Img,
             validation_data=([V_L_image, V_L_image], V_H_image),
             epochs=30,
             batch_size=batch_size)
total_time = round((time.time() - start_time), 4)
print("Computational time in seconds is " + str(total_time))

training_loss = history.history['loss']
test_loss = history.history['val_loss']
training_acc = history.history['acc']
testing_acc = history.history['val_acc']
epoch_count = range(1, len(training_loss) + 1)

# Visualize loss history
plt.plot(epoch_count, training_loss, 'r-')
plt.plot(epoch_count, test_loss, 'b-')
plt.title('Multi Scale model Loss')
plt.legend(['Train Loss', 'Validation Loss'], loc='upper left')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show();

#Visualize Accuracy history
plt.plot(epoch_count, training_acc, 'g-')
plt.plot(epoch_count, testing_acc, 'b-')
plt.title('Multi Scaling model Accuracy')
plt.legend(['Train Acuuracy', 'Validation Accuracy'], loc='upper left')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show();
```

Figure 19 Model Training

```
def PSNR(y_true,y_pred):
    mse=tf.reduce_mean( (y_true - y_pred) ** 2 )
    return 20 * log10(1/ (mse ** 0.5))

def log10(x):
    numerator = tf.math.log(x)
    denominator = tf.math.log(tf.constant(10, dtype=numerator.dtype))
    return numerator / denominator

def pixel_MSE(y_true,y_pred):
    return tf.reduce_mean( (y_true - y_pred) ** 2 )
def plot_images(high,low,predicted):
    plt.figure(figsize=(15,15))
    plt.subplot(1,3,1)
    plt.title('High Resolution Image', color = 'black', fontsize = 20)
    plt.imshow(high)
    plt.subplot(1,3,2)
    plt.title('Low Resolution Image', color = 'black', fontsize = 20)
    plt.imshow(low)
    plt.subplot(1,3,3)
    plt.title('Predicted Image from Multi_scale', color = 'Red', fontsize = 20)
    plt.imshow(predicted)

    plt.show()

for i in range(9,16):
  predicted = np.clip(Multi_scale_learning.predict([TN_L_Img[i].reshape(1,SIZE, SIZE,3),TN_L_Img[i].reshape(1,SIZE, SIZE,3)]),0.
0,1.0).reshape(SIZE, SIZE,3)
  plot_images(TN_H_Img[i],TN_L_Img[i],predicted)
  print('PSNR Value is:',PSNR(TN_H_Img[i],predicted),'dB')
```

Figure 20 Model Evaluation

3.4.3 Auto-Encoder Model

The encoder part of the model has 15 layers in it. The model consists of 9 2D convolutional layers each with same parameters such as 2D strides and ReLU as activation function except for number of neurons of the layer. The model starts with 2D convolutional layers then Maxpooling layer is added to down sample the data, the same sequence of layers is added again with different parameters, further a residual block layer is added which helps in maintaining quality of data, then data is unsampled (increased in dimension) using an umsampling2D layer and at last the layers are added. The decoder part consists of just 1 2D convolutional layer and has same parameters as the convolutional layer present in the encoder part.

```
#function to generate a residual block comprising of different types of layers
def residual_block_gen(ch=64,k_s=3,st=1):
    model=tf.keras.Sequential([
    tf.keras.layers.Conv2D(ch,k_s,strides=(st,st),padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.LeakyReLU(),
    tf.keras.layers.Conv2D(ch,k_s,strides=(st,st),padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.LeakyReLU(),])
    return model
```

Figure 21 Function to generate residual block

```
#fix input size for model
input_img=Input(shape=(256,256,3))

#encoder layer
l1=Conv2D (64,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(input_img)
l2=Conv2D (64,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(l1)
l3=MaxPooling2D(padding='same')(l2)
l4=Conv2D (128,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(l3)
l5=Conv2D (128,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(l4)
l6=MaxPooling2D(padding='same')(l5)

l7=Conv2D (256,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(l6)
l7=residual_block_gen()(l7)
l8=UpSampling2D()(l7)
l9=Conv2D (128,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(l8)
l10=Conv2D (128,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(l9)
l11=add([l10,l5])
l12=UpSampling2D()(l11)
l13=Conv2D (64,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(l12)
l14=Conv2D (64,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(l13)
l15=add([l14,l2])

#decoder layer
decoder=Conv2D (3,(3,3) , padding='same' ,activation='relu',activity_regularizer=regularizers.l1(10e-10))(l15)
```

Figure 22 Auto-Encoder Model

```
# model call
Myauto_encoder_model=Model(input_img,decoder)
#display summary in table form
Myauto_encoder_model.summary()
#desplay summary in flow diagram
plot_model(Myauto_encoder_model, to_file ='/content/drive/MyDrive/FINAL_CODE/final_code_s/encoder.png',show_shapes=True)
#compile model
# Compiling the model with Adam as optimizer and mean absolute error as the loss metric
Myauto_encoder_model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'mean_absolute_error',
              metrics = ['acc'])
```

Figure 23 Model call

```
#training the model with 30 epochs

#save starting time in variable
start_time = time.time()
history=Myauto_encoder_model.fit(TN_L_Img, TN_H_Img, epochs = 30, batch_size = 1,
          validation_data = (V_L_image,V_H_image))
total_time = round((time.time() - start_time), 4)
#print training time of model
print("Computational time in seconds is " + str(total_time))

#Save history of the model in variable
training_loss = history.history['loss']
test_loss = history.history['val_loss']
training_acc = history.history['acc']
testing_acc = history.history['val_acc']
epoch_count = range(1, len(training_loss) + 1)

# Visualize loss history
plt.plot(epoch_count, training_loss, 'r-')
plt.plot(epoch_count, test_loss, 'b-')
plt.title('AutoEncoder model Loss')
plt.legend(['Train Loss', 'Validation Loss'], loc='upper left')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show();

#Visualize Accuracy history
plt.plot(epoch_count, training_acc, 'r-')
plt.plot(epoch_count, testing_acc, 'b-')
plt.title('AutoEncoder model Accuracy')
plt.legend(['Train Acuuracy', 'Validation Accuracy'], loc='upper left')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show();
```

Figure 24 Model Training

```
#Supplementary functions used in comparing the original images to predicted images

#function to calculate Peak Signal to Noise Ratio of the predicted image
def PSNR(y_true,y_pred):
    mse=tf.reduce_mean( (y_true - y_pred) ** 2 )
    return 20 * log10(1/ (mse ** 0.5))

#function to convert decimal value to logarithmic value with base 10
def log10(x):
    numerator = tf.math.log(x)
    denominator = tf.math.log(tf.constant(10, dtype=numerator.dtype))
    return numerator / denominator

#function to calculate the mean square error for a given predicted image
def pixel_MSE(y_true,y_pred):
    return tf.reduce_mean( (y_true - y_pred) ** 2 )
def plot_images(high,low,predicted):
    plt.figure(figsize=(15,15))
    plt.subplot(1,3,1)
    plt.title('High Resolution Image', color = 'black', fontsize = 20)
    plt.imshow(high)
    plt.subplot(1,3,2)
    plt.title('Low Resolution Image ', color = 'black', fontsize = 20)
    plt.imshow(low)
    plt.subplot(1,3,3)
    plt.title('Predicted Image for Auto-encoder', color = 'Red', fontsize = 20)
    plt.imshow(predicted)
    plt.show()

for i in range(5,9):

    predicted = np.clip(Myauto_encoder_model.predict(TN_L_Img[i].reshape(1,SIZE, SIZE,3)),0.0,1.0).reshape(SIZE, SIZE,3)
    plot_images(TN_H_Img[i],TN_L_Img[i],predicted)
    print('PSNR Value is',PSNR(TN_H_Img[i],predicted),'dB')
```

Figure 25 Supplementary Functions

3.4.4 EDSR Model

This code shows the use of an Enhanced Deep Super Resolution network. The input layer is of fixed size, the next layers are sequence of fixed set of layers.

Here the input layer is connected to the residual layer that we generate through a customized function, the residual layer is then unsampled (dataset dimension increased) using another customized function, the output of this layer is down sampled using the Maxpooling layer and at last 2D convolutional layer is added with 1D stride. This sequence of fixed layers is repeated for 5 times and output of one instance is input to the other hence a DNN is created. At last batch normalization is applied to standardize the inputs and output of this layer is then unsampled and given to Maxpooling which will give output to the 2D convolutional layer which will thereby predict the image.

```
# function to generate a residual block  which is used to skip connections in some layers so the quality of data persists
def residual_block_gen(ch=64,k_s=3,st=1):
    model=tf.keras.Sequential([
    tf.keras.layers.Conv2D(ch,k_s,strides=(st,st),padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.LeakyReLU(),
    tf.keras.layers.Conv2D(ch,k_s,strides=(st,st),padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.LeakyReLU(),])
    return model

#function to generate a upsample block which increases the quality of data
def Upsample_block(x,ch=256, k_s=3, st=1):
    x = tf.keras.layers.Conv2D(ch,k_s, strides=(st,st), padding='same')(x)
    x = tf.nn.depth_to_space(x, 2) # Subpixel pixelshuffler
    x = tf.keras.layers.LeakyReLU()(x)
    return x
lr=tf.keras.layers.Input(shape=(256,256,3))
input_conv=tf.keras.layers.Conv2D(64,9,padding='same')(lr)
input_conv=tf.keras.layers.LeakyReLU()(input_conv)
```

Figure 26 Functions to Generate Residual Blocks

```
#function to generate a upsample block which increases the quality of data
def Upsample_block(x,ch=256, k_s=3, st=1):
    x = tf.keras.layers.Conv2D(ch,k_s, strides=(st,st), padding='same')(x)
    x = tf.nn.depth_to_space(x, 2) # Subpixel pixelshuffler
    x = tf.keras.layers.LeakyReLU()(x)
    return x
lr=tf.keras.layers.Input(shape=(256,256,3))
input_conv=tf.keras.layers.Conv2D(64,9,padding='same')(lr)
input_conv=tf.keras.layers.LeakyReLU()(input_conv)

#function that generates a deep level neural network
def model():
    SRRes=input_conv
    for x in range(5):
        res_output=residual_block_gen()(SRRes)
        SRRes=Upsample_block(SRRes)
        SRRes=MaxPooling2D()(SRRes)
        SRRes=tf.keras.layers.Add()([SRRes,res_output])
        SRRes=tf.keras.layers.Conv2D(64,9,padding='same')(SRRes)
    SRRes=tf.keras.layers.BatchNormalization()(SRRes)
    SRRes=tf.keras.layers.Add()([SRRes,input_conv])
    SRRes=Upsample_block(SRRes)
    SRRes=MaxPooling2D()(SRRes)
    output_sr=tf.keras.layers.Conv2D(3,9,activation='sigmoid',padding='same')(SRRes)
    return tf.keras.models.Model(lr,output_sr)
```

Figure 27 Functions to generate deep neural network

```
# Building the model
MYEDSR = model()
MYEDSR.summary()
plot_model(MYEDSR, to_file ='/content/drive/MyDrive/FINAL_CODE/final_code_s/super_res.png',show_shapes=True)
# Compiling the model using Adam as optimizer and mean absolute error as loss metric.
MYEDSR.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'mean_absolute_error',
              metrics = ['acc'])
```

Figure 28 Model Building

```
#Training the model
start_time = time.time()
history=MYEDSR.fit(TN_L_Img, TN_H_Img, epochs = 30, batch_size = 1,
          validation_data = (V_L_image,V_H_image))
total_time = round((time.time() - start_time), 4)
#print total training of the model
print("Computational time in seconds is " + str(total_time))


training_loss = history.history['loss']
test_loss = history.history['val_loss']
training_acc = history.history['acc']
testing_acc = history.history['val_acc']
epoch_count = range(1, len(training_loss) + 1)

# Visualize loss history
plt.plot(epoch_count, training_loss, 'r-')
plt.plot(epoch_count, test_loss, 'b-')
plt.title('EDSR model Loss')
plt.legend(['Train Loss', 'Validation Loss'], loc='upper left')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show();

#Visualize Accuracy history
plt.plot(epoch_count, training_acc, 'r-')
plt.plot(epoch_count, testing_acc, 'b-')
plt.title('EDSR model Accuracy')
plt.legend(['Train Acuuracy', 'Validation Accuracy'], loc='upper left')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show();
```

Figure 29 Model Training

```
def PSNR(y_true,y_pred):
  mse=tf.reduce_mean( (y_true - y_pred) ** 2 )
  return 20 * log10(1 / (mse ** 0.5))

def log10(x):
  numerator = tf.math.log(x)
  denominator = tf.math.log(tf.constant(10, dtype=numerator.dtype))
  return numerator / denominator

def pixel_MSE(y_true,y_pred):
  return tf.reduce_mean( (y_true - y_pred) ** 2 )
def plot_images(high,low,predicted):
    plt.figure(figsize=(15,15))
    plt.subplot(1,3,1)
    plt.title('High resolution images', color = 'blue', fontsize = 15)
    plt.imshow(high)
    plt.subplot(1,3,2)
    plt.title('Low resolution images ', color = 'blue', fontsize = 15)
    plt.imshow(low)
    plt.subplot(1,3,3)
    plt.title('Predicted images ', color = 'Red', fontsize = 15)
    plt.imshow(predicted)
    plt.show()

for i in range(1,6):
    #Display predicting images of EDSR MODEL
    predicted = np.clip(MYEDSR.predict(TS_L_Img[i].reshape(1,SIZE, SIZE,3)),0.0,1.0).reshape(SIZE, SIZE,3)
    plot_images(TS_H_Img[i],TS_L_Img[i],predicted)
    print('PSNR value of EDSR :', PSNR(TS_H_Img[i],predicted))
```

Figure 30 Model Evaluation

The following is a simple explanation of the terms used while building the model.

Down-sampling:

When a given image is down sampled, the number of pixels in the image is reduced based on the frequency of sampled data. Consequently, image resolution and size are decreased.

Up-sampling:

Up-sampling techniques can be used to increase the number of pixels in a down-sampled image. As a result of up-sampling, the image has a higher resolution and a larger size.

Epoch:

The process of passing an entire dataset through the neural network one time is called an "epoch".

Adam:

Adam is a computationally efficient optimization solution for the Neural Network technique that takes little memory and is ideally suited for situations with a large dataset, parameters, or both.

Activation functions used for model building:

The activation function is a simple function that changes input values into output values based on a specified range.

- **ReLU**:
  In brief, if the input is positive, the rectified linear activation function (ReLU) outputs it directly; otherwise, it returns 0. Which means it ranged from 0 to infinity. All negative numbers are translated to zero in this situation, and the conversion rate is so quick that it cannot map to or fit into data properly, which creates an issue.
- **Leaky ReLU:**
  In ReLU, negative input values turn into zero very quickly. We don't make all negative inputs zero while using Leaky ReLU. Rather, we reduce them to a value close to zero. This resolves the main problem with ReLU activation.
- **Sigmoid:**
  A sigmoid activation function receives input and converts output values between 0 and 1 in different ways.

Kernel:

The kernels are simply filters that extract the features of a small portion of an image. They are utilized to extract features in convolutional layers.

Convo2D:

It stands for 2D convolution layer. The operation involves sliding a weighted matrix or kernel across a 2D dataset and multiplying the data under the kernel element by element.

Convo2DTranspose:

It applies convolution with a fractional stride. In simple terms, these convolutions compute the matrix transposition of a conventional convolutional layer, switching between the effects of the forward and reverse passes.

Learning rate:

This hyperparameter determines how quickly the neural network learns to cope with the challenge. Learning rate is usually between 0 and 1 and has a small positive value.