Innovative Image Depixelizer using an Efficient Sub - Pixel CNN

Ankit Singh
Student of Computing Science
Department of computer
and engineering (cse)
Galgotias University
Greater Noida
as1230641@gmail.com

Vaishnavi Chauhan
Student of Computing Science
and engineering(cse)
Galgotias University
Greater Noida
chauhanvaishnavi2316@gmail.com

Mrs Sonia Rani
Department of computer
Science, Galgotias University
Greater Noida
soniarani@galgotiasuniversity.edu.in

Mr Brijesh Singh
Department of computer science,
Galgotias University,
Greater noida
brijeshsingh@galgotiasuniversity.edu.in

Abstract- Computer vision faces challenges when reconstructing images. in the proposed idea, we tried an attempt to enhance image quality by providing guidance to a deep residual convolutional neural network using pre-existing datasets. Our method leverages an efficient sub-pixel convolutional neural network (ESPCNN) [1] algorithm to turn the low-resolution (LR) images into high-resolution (HR) or super-resolution (SR) images. The ESPCNN algorithm is designed to enhance and enlarge images, resulting in high-resolution images that retain the original data. We employ a convolutional recurrent neural network (CRNN) [2]approach to reconstruct high-resolution images from under-sampled data, which can be useful in applications such as medical imaging, satellite imagery, and surveillance systems. The method we proposed addresses some of the limitations of techniques that use convolutional neural networks, such as FSRCNN[3] and SRCNN[4]. Through research, we determine the validity of our proposed model.

Keywords— ESPCNN, image-resolution, convolution neural network, pixelization.

I. INTRODUCTION

In image processing, super resolution is a crucial task that involves creating high-resolution (HR) images from low-resolution (LR) images. This is a difficult problem, particularly for single image super-resolution (SISR), where the objective is to enhance the quality of a single LR image. The applications of this technique are diverse, ranging from improving face identification and detection from images obtained from low-resolution camera surveillance systems to reducing server costs in media content by sending lower resolution media and upscaling it in the process. Additionally, it has several other applications in the fields of medical and satellite imaging.

The accuracy and performance of the reconstructed super-resolution (SR) images have improved significantly as a result of the numerous super-resolution (SR) models and potent deep learning algorithms created by researchers in previous years. Because low-resolution images have little high-frequency selective information, single-image super-resolution (SISR) is still a challenging job. We present a technique for reconstructing high-resolution pictures from under-sampled data using an effective sub-pixel convolutional neural network (ESPCNN)-based image depixelizer in order to address this problem. This method is based on the ESPCNN algorithm, which takes a low-resolution image as input and enhances it using several functions like enlargement and c, without affecting the original data, thus converting it into super and high-resolution images. The embedded image depixelizer is robust and can handle diverse low-resolution images, providing high-quality resolution outcomes.
Our paper provides a thorough analysis of our proposed image depixelizer method and evaluates its effectiveness against other advanced single image super-resolution (SISR) techniques.

II. LITERATURE SURVEY

In Image Depixelizer Using Enhanced Deep Residual Network, Nagothi Moulika [5] stated in her work that she made the image which needs to be depixelized as a constraint related problem and tried to solve it using CNN algorithm with resulting output as a super resolution image however the CNN algorithm is not memory efficient as it interpolate the image in the very first layer of CNN, another similar work done as a collaboration [6] aims to convert a low resolution or dipixelated image as super resolution image using CNN proposes a very homogenous result along with a same problem of memory efficiency. In Image super-resolution with sparse neighbor embedding [7] they proposed a sparse neighbor selection scheme for super resolution image construction in which they first determined a huge amount of neighbors as their targeted candidates and formulated a SL0 Algorithm. In a research paper titled as deep network cascade for super resolution[8] they proposed a new model named deep cascade network which increases or scales the images layer by layer. In going deeper in resolution [9] they proposed a system using CNN algorithm however they managed to increase the width and height of the network for optimal quality while keeping computational budget constant thrust increasing the overall output as against image depixelizer [5] where they used CNN with smaller height and width. In an efficient study Very Deep Convolutional Networks for Large-Scale Image Recognition [10] they worked upon learning the effects upon a neural network by increasing the network depth to 16-19 layers.on the prior studies and CNN models.

III. NETWORK STRUCTURE

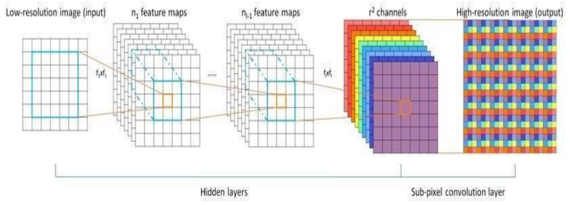


Fig. 1 Structure of neural network

Any deep learning model, including the efficient sub-pixel convolutional neural network (ESPCNN) used in the picture depixelizer, must have a strong network structure. As shown in the figure, the network is made up of L layers, the first of which is a convolutional layer that takes feature maps from the input image's low resolution.

The feature maps generated in the first layer are then passed to the subsequent layers, which help to refine and enhance the extracted features. After the feature extraction process, the output image is reconstructed using the efficient sub-pixel convolutional layer in the last layer of the network. The upscaling factor specified beforehand is used to adjust the size of the output image, and the sub-pixel convolutional layer ensures that the output image has the desired resolution and quality.

The efficient sub pixel convolutional layer is the key element in the ESPCNN architecture, as it enables the network to perform super-resolution by upscaling the image without requiring the use of an interpolation method. The sub pixel convolutional layer is applied at the bottom-most layer of the network, and it works to upscale the image in the final step. The utilisation of smaller image and filter sizes for feature extraction allows for a significant reduction in computational complexity and memory cost within the network. This results in improved overall efficiency.

The ESPCNN network's structure, which is utilised in the image depixelizer, is specifically designed to maximise the super-resolution process and enhance the clarity of the high-resolution images produced.

Dataset

The proposed ESPCNN model for image super-resolution was trained and evaluated in this research using publicly accessible datasets. Both training and testing were conducted using the Timofte dataset, which is frequently used in SISR research. It consists of 91 pictures and two test datasets, SET5 and SET14, each of which contains 5 and 14 images. The Berkeley segmentation datasets BSD300 and BSD500, which offered 100 and 200 images, respectively, for experimentation, were also used. Additionally, we used the super texture dataset, which consists of 136 texture images, to ensure a thorough assessment of the suggested model under different conditions.

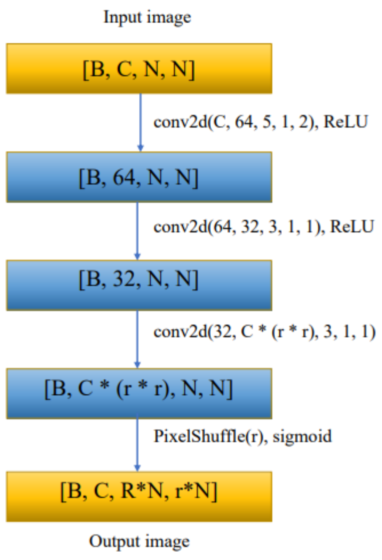


Fig. 2 proposed system

To further explain the proposed system, let's break down each layer and its role in the network.

Convolutional filters, which make up the first layer, accept LR (low-resolution) images with dimensions of [B, C, N, N] and apply 64 filters of 5x5 size to extract features from the picture. This layer's output passes through a Rectified Linear Unit (ReLU) activation function, which adds nonlinearity to the network.
A second convolutional layer with 32 3x3-sized filters makes up the second layer. This layer aids the network in learning more complex representations of the picture by also extracting features from the LR image. The result of this layer is once more subjected to a ReLU activation function.

The third layer of the network has a kernel dimension of 3x3 and a fixed number of output channels Crr, where r is the upscaling factor. A sub-pixel shuffle function, which rearranges the output channels to create the HR picture, comes after this layer, making it special. The output picture produced by the sub-pixel shuffle function has the following dimensions: [B, C, rN, rN], increasing the resolution of the image by a factor of r.

In summary, the proposed system efficiently enhances the resolution of LR images by utilising the sub-pixel convolutional layer. This approach reduces the network's computational complexity and memory requirements, improving its efficiency.

4.1 Sub Pixel Convolution

The sub-pixel concept is utilised in the proposed model through the pixel shuffle function, which is used to increase the resolution of the input image. The pixel shuffle function rearranges the pixels in the low-resolution image in such a way that they are grouped into larger blocks, where each block contains a group of sub-pixels that correspond to a single pixel in the high-resolution image. This grouping of sub-pixels allows for a more accurate reconstruction of the high-resolution image.
The pixel shuffle function is implemented in the final layer of the proposed model, which is a convolutional layer with a fixed count of output channels Crr also 33 kernel size. The output of this layer is reshaped using the pixel shuffle function therefore the resulting super-resolution image has the dimensions [B, C, rN, r*N], B is the batch size, C is the number of channels, and N is the size of the input picture. The resulting high-resolution image has a resolution r times higher than the input image, and is obtained without the need for any interpolation techniques.

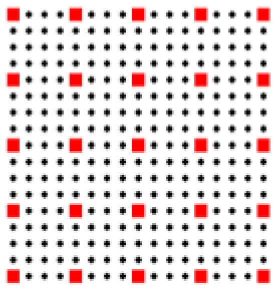


Fig. 3
Visualisation of Pixels

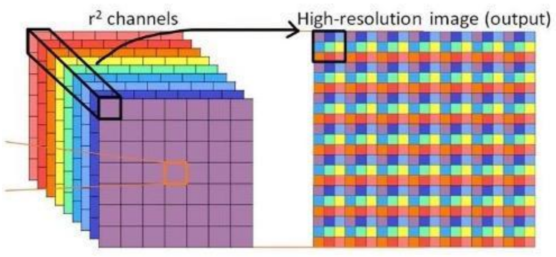


Fig. 4 Interpolation of camera's image

The figure above illustrates the pixels captured by the camera's imaging plane, where the red squares represent the physical pixels, and the dark(black) points represent sub-pixels. The accuracy of sub-pixels can be altered through interpolation, which influences the mapping from small to large square areas. This enables the implementation of sub-pixel interpolation to achieve improved image resolution.

The sub-pixel convolution process is a key component of the proposed SPCNN method for super resolution. It involves two main operations: a standard convolution operation and a pixel shuffling operation. The convolution operation is performed first, and the output channel of the final layer must be C x r x r in order to make sure that all pixel elements are consistent according to the desired high-resolution output image.

Unlike other methods that use deconvolution, the sub-pixel convolution process does not require padding. In contrast, the output image combines individual pixels from multiple channel feature maps into a single r x r square area. This allows individual pixels on feature maps to be treated as sub-pixels in the generated output image.

The interpolation method is implicitly learned by the network through the convolutional layers, making it more efficient than other methods. To further improve efficiency, the convolution operations are conducted on smaller LR images. Overall, the sub-pixel convolution process is an important part of the SPCNN method for super resolution.

4.2 Loss Function

The pixel-wise mean squared error (MSE) loss function is commonly used for measuring the difference among the generated super resolution pictures and the corresponding ground truth higher resolution pictures. It can be mathematically expressed as: MSE = 1/N * sum_i=1^N (x_i - y_i)^2
where N is the total number of pixels in the image, x_i is the pixel value of the generated super resolution image at position i, and y_i is the corresponding pixel value of the ground truth higher resolution image at position i.

During training, the network learns to minimise this loss function by adjusting its weights and biases to produce more accurate super resolution images.

$$\ell(W_{1:L}, b_{1:L}) = \frac{1}{r^2HW} \sum_{x=1}^{rH} \sum_{x=1}^{rW} \left(\mathbf{I}_{x,y}^{HR} - f_{x,y}^{L}(\mathbf{I}^{LR})\right)^2$$

Fig 4

Loss function of ESPCN

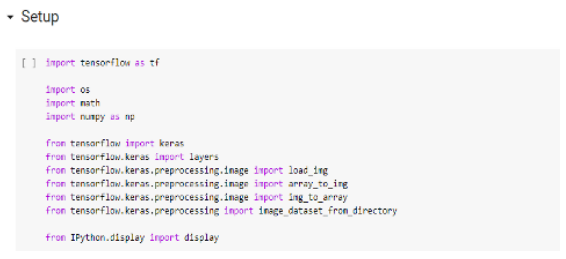
To clarify, the equation you provided is a mathematical expression that represents the pixel-wise mean squared error loss function of the network. This function is used to evaluate the similarity between the super resolution images generated by the network and the corresponding high resolution ground truth images.

In this equation, I(HR) denotes the high resolution ground truth image and I(LR) represents the low resolution input image. The variable r represents the desired upscaling factor, where H and W represent the height and width of the picture, respectively. The terms W(1:L) and b(1:L) refer to the learnable weights and biases of the network, respectively.

The objective of the training process is to minimise the MSE loss function in accordance with the network parameters W and b. By doing so, the network will be able to generate high quality super resolution images with improved performance.

Implementation and Output

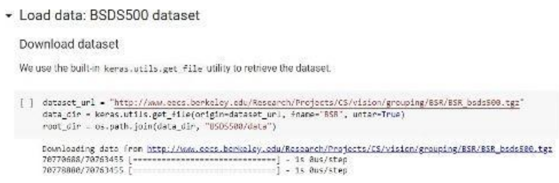
Importing the libraries and creating a setup to perform the desired algorithm



Setup

```
[ ] import tensorflow as tf

    import os
    import math
    import numpy as np

    from tensorflow import keras
    from tensorflow.keras import layers
    from tensorflow.keras.preprocessing.image import load_img
    from tensorflow.keras.preprocessing.image import array_to_img
    from tensorflow.keras.preprocessing.image import img_to_array
    from tensorflow.keras.preprocessing import image_dataset_from_directory

    from IPython.display import display
```

In this we are using the dataset BSDS500 and we are using the keras.utils.get_file utility to redeem the dataset.



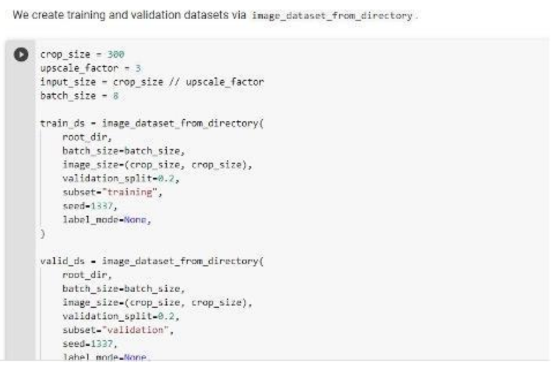
Load data: BSDS500 dataset

Download dataset

We use the built-in `keras.utils.get_file` utility to retrieve the dataset.

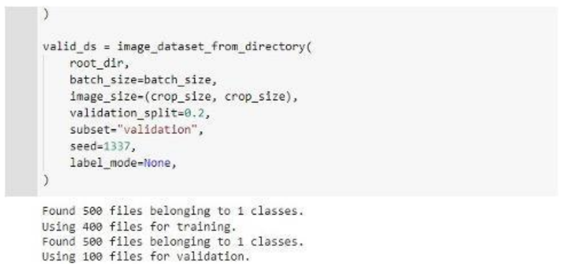
```
[ ] dataset_url = "http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/BSR/BSR_bsds500.tgz"
    data_dir = keras.utils.get_file(origin=dataset_url, fname="BSR", untar=True)
    root_dir = os.path.join(data_dir, "BSDS500/data")

    Downloading data from http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/BSR/BSR_bsds500.tgz
    70770688/70763455 [==============================] - 1s 0us/step
    70778880/70763455 [==============================] - 1s 0us/step
```

Generating and dividing the dataset into training and validation datasets with the help of image_dataset_from_directory.



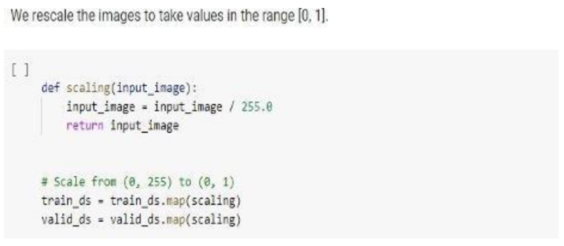
We create training and validation datasets via `image_dataset_from_directory`.

```
crop_size = 300
upscale_factor = 3
input_size = crop_size // upscale_factor
batch_size = 8

train_ds = image_dataset_from_directory(
    root_dir,
    batch_size=batch_size,
    image_size=(crop_size, crop_size),
    validation_split=0.2,
    subset="training",
    seed=1337,
    label_mode=None,
)

valid_ds = image_dataset_from_directory(
    root_dir,
    batch_size=batch_size,
    image_size=(crop_size, crop_size),
    validation_split=0.2,
    subset="validation",
    seed=1337,
    label_mode=None,
```



```
)

valid_ds = image_dataset_from_directory(
    root_dir,
    batch_size=batch_size,
    image_size=(crop_size, crop_size),
    validation_split=0.2,
    subset="validation",
    seed=1337,
    label_mode=None,
)

Found 500 files belonging to 1 classes.
Using 400 files for training.
Found 500 files belonging to 1 classes.
Using 100 files for validation.
```

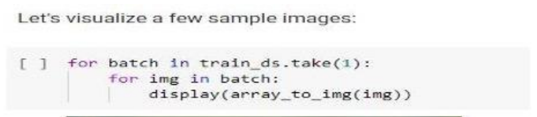
Now we are rescaling the pictures to get the val. in the range [0, 1]



We rescale the images to take values in the range [0, 1].

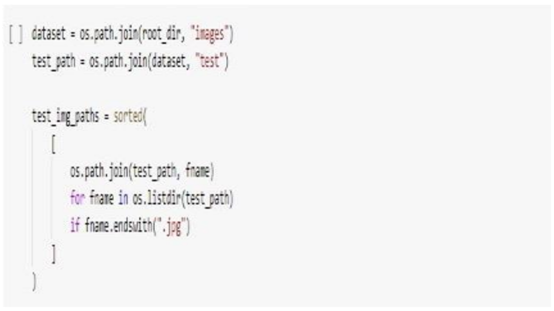
```
[ ] def scaling(input_image):
        input_image = input_image / 255.0
        return input_image


    # Scale from (0, 255) to (0, 1)
    train_ds = train_ds.map(scaling)
    valid_ds = valid_ds.map(scaling)
```

Lets visualize some sample images

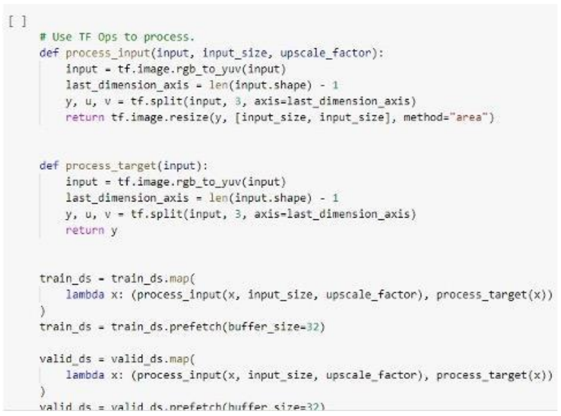


Let's visualize a few sample images:

```
[ ] for batch in train_ds.take(1):
        for img in batch:
            display(array_to_img(img))
```

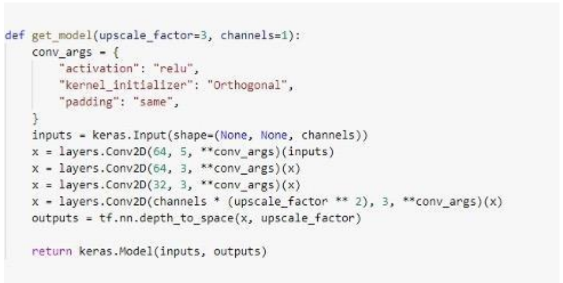


```
[ ] dataset = os.path.join(root_dir, "images")
    test_path = os.path.join(dataset, "test")

    test_img_paths = sorted(
        [
            os.path.join(test_path, fname)
            for fname in os.listdir(test_path)
            if fname.endswith(".jpg")
        ]
    )
```



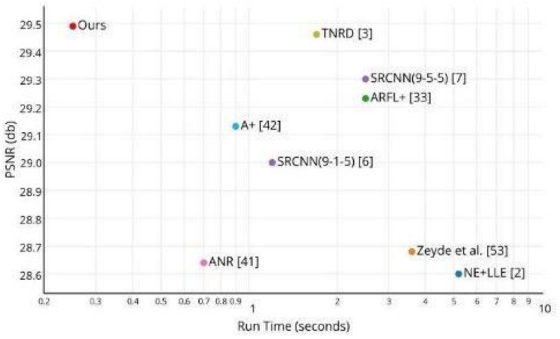
```
[ ]
    # Use TF Ops to process.
    def process_input(input, input_size, upscale_factor):
        input = tf.image.rgb_to_yuv(input)
        last_dimension_axis = len(input.shape) - 1
        y, u, v = tf.split(input, 3, axis=last_dimension_axis)
        return tf.image.resize(y, [input_size, input_size], method="area")


    def process_target(input):
        input = tf.image.rgb_to_yuv(input)
        last_dimension_axis = len(input.shape) - 1
        y, u, v = tf.split(input, 3, axis=last_dimension_axis)
        return y


    train_ds = train_ds.map(
        lambda x: (process_input(x, input_size, upscale_factor), process_target(x))
    )
    train_ds = train_ds.prefetch(buffer_size=32)

    valid_ds = valid_ds.map(
        lambda x: (process_input(x, input_size, upscale_factor), process_target(x))
    )
    valid_ds = valid_ds.prefetch(buffer_size=32)
```

We are using the ReLu as an activation function and building the model according to that.



```
def get_model(upscale_factor=3, channels=1):
    conv_args = {
        "activation": "relu",
        "kernel_initializer": "Orthogonal",
        "padding": "same",
    }
    inputs = keras.Input(shape=(None, None, channels))
    x = layers.Conv2D(64, 5, **conv_args)(inputs)
    x = layers.Conv2D(64, 3, **conv_args)(x)
    x = layers.Conv2D(32, 3, **conv_args)(x)
    x = layers.Conv2D(channels * (upscale_factor ** 2), 3, **conv_args)(x)
    outputs = tf.nn.depth_to_space(x, upscale_factor)

    return keras.Model(inputs, outputs)
```

This efficient subpixel CNN model can be compared to other super resolution models on the basis of their run time and speed.
Plotting a graph between these model to compare the speed of various super resolution models.



V. RESULTS

Picture super-resolution outcomes

The experiments demonstrated that the ESPCN model outperformed both the SRCNN and standard 9-1-5 models, with an average peak signal-to-noise ratio (PSNR) of 27.76 decibels (dB). The use of the hyperbolic tangent (tanh) activation function did not show a significant improvement in performance. The authors further evaluated the model's performance on various datasets, including Timofte, SET5, SET14, BSD300, BSD500, and the super texture dataset. Their model achieved competitive results compared to the best existing methods on these datasets.

VI. CONCLUSION

In this paper we demonstrate that the lower resolution (LR) pictures may be reconstructed to higher resolution pictures with the help of the convolutional neural network. We have various CNN methods like SRCNN, FSRCNN and VDSR all these approaches can be used to upscale an image resolution to a specific factor. In this paper we used ESPCNN to reconstruct the images resolutions and convert the lower resolution image into the higher resolution images. We use various python libraries that are provided by python, to import these libraries we use virtual studio code (VS Code) Which provides a suitable interface to work for the project and also very convenient to use. The dataset used in the project are available on the internet free of cost. The dataset are also imported using the libraries. These datasets are divided into training and validation sets. In this project ReLu is used as an activation function to build the model we define some utility functions. The model is efficient in manner of time and complexity compared to other CNN approaches and produces better results.

REFERENCES

1. Nay, Dam Ja Ba, Single Image Super Resolution using ESPCN – With SSIM Loss (January 10, 2021). Available at SSRN

2. Elsevier "Physica D: Nonlinear Phenomena" journal, Volume 404, March 2020: Special Issue on Machine Learning and Dynamical Systems; doi:10.1016/j.physd.2019.132306

3. Chao Dong, Chen Change Loy, Kaiming He, Xiaoou Tang. Learning a Deep Convolutional Network for Image Super-Resolution, in Proceedings of European Conference on Computer Vision (ECCV), 2014
4. Jinlu Zhang; Mingliang Liu; Xiaohang Wang; Chengcheng Cao,INSPEC Accession Number: 21225774,DOI: 10.23919/CCC52363.2021.9550205 Publisher: IEEE, Conference Location: Shanghai, China

5. Nagothi Moulika, M.Tech Scholar Dept. of CS&SE, Andhra University College of Engineering, Visakhapatnam, A.P 2021 JETIR October 2021, Volume 8.

6. P. Vasanthi, K. Umasaisirisha Department of , K. Hari Swetha, M. Ratna Shivani, Dr. P. Vijaya Bharati.Image Deipixelizer, International Journal of Innovations in Engineering and Technology (IJIET).

7. X. Gao, K. Zhang, D. Tao, and X. Li. Image super-resolution with sparse neighbour embedding. IEEE Transactions on Image Processing, 21(7):3194–3205, 2012.

8. Z. Cui, H. Chang, S. Shan, B. Zhong, and X. Chen. Deep network cascade for image super-resolution. In European Conference on Computer Vision (ECCV), pages 49–64. Springer, 2014.

9. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In CVPR 2015, 2015.

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