ISSN No:-2456-2165

Image Restoration using Deep Learning

Dr. Hansaraj Wankhede, Abhishikth Thul, Deep Malviya, Pratik Pathe, Shivam Kuite
GHRCE Nagpur, India

Abstract:- Image restoration using deep learning attempts to create an image recovery system that can restore outdated and corrupted images regardless of the type of noise present. Photos play an important role in capturing and preserving cherished moments in today's digital age. However, due to a variety of environmental conditions, these images can get distorted over time. Manual recovery is time-consuming and labor-intensive, emphasizing the need for an automated alternative. Deep learning techniques, notably convolutional neural networks, are used in the proposed system, which has demonstrated promising results in image processing tasks. The essay goes over these approaches in detail, focusing on image noise reduction, deblurring, dehazing, and super-resolution. Different network topologies are investigated, including those with residual or merge-skip connections, as well as their receptive fields and the usage of unsupervised autoencoder processes. The study also looks at image quality criteria to see how helpful they are in image recovery. An effective deblurring network and adaptable training algorithms for high-resolution recovery tasks are suggested to handle the special difficulty of deblurring. The proposed method's performance is compared to state-of-the-art approaches utilizing both quantitative and qualitative assessments. The study finishes with a discussion of prospective future research topics and outstanding challenges in image recovery. The ultimate goal of this research is to create a robust and efficient image recovery system capable of restoring photos to their original quality regardless of the type or severity of corruption or noise present.

Keywords:- *Image recovery, Noise reduction, Neural networks, Image restoration, Deep learning, Image processing, Super-resolution, Deblurring.*

I. INTRODUCTION

Image restoration is a key activity in image processing that aims to recover the original quality of images that have been distorted by causes such as noise, blur, or other flaws. Image restoration approaches have seen considerable gains in terms of accuracy and efficiency as a result of recent advances in deep learning, notably convolutional neural networks (CNNs). The U-Net architecture is differentiated by its U-shaped network design, which incorporates encoder and decoder paths. The encoder path uses down-sampling to collect contextual information from the input picture, whereas the decoder path uses up-sampling and skip connections to restore spatial features and provide a reconstructed image. The skip connections of the U-Net architecture enable the integration of The high-resolution feature maps from the encoder path are mixed with the up-sampled feature maps from the decoder path. This integration preserves tiny features while allowing the model to exploit contextual knowledge learned at different sizes, resulting in more accurate and visually appealing image restoration outcomes. Deep learning-based image restoration utilising the U-Net architecture has yielded promising results in a variety of applications such as denoising, deblurring, super-resolution, and inpainting. The U-Net learns the underlying mapping between the corrupted input and the clean output by training it on a large dataset of matched clean and corrupted images, allowing it to generalise well to unseen data.

II. LITERATURE REVIEW

The study's authors, Olaf Ronneberger, Philip Fischer, and others [1], suggested a network and training strategy that relies significantly on data augmentation to make better use of the annotated samples provided. Offers. The layout is made up of contracting routes that collect context and expanding paths that allow for exact localisation. They discovered that this sort of network could be trained consistently from a very small number of photos in his ISBI task of segmenting neural structures within an electron microscopy stack, and the best answer to date (sliding window convolutional networks) has been proved to outperform. Jiwon Kim, Jung Kwon Lee, and Kyung Mu Lee [2] offer a highly accurate single-frame super-resolution (SR) technique. Our method use a deep convolutional network inspired by the Image Net classification tool VGG-net. They discovered that raising the network's depth considerably increased accuracy. The final model employs 20 weight layers. Contextual information for vast picture areas may be successfully obtained by cascading tiny filters numerous times in deep network topologies.

Kai Zhang, Wangmeng Zuo, and colleagues [3] present FFDNet, a rapid and adaptable convolutional noise reduction neural network that uses a noise level map as input. When applied to downsampled sub-images, his suggested FFDNet performs well in terms of both inference time and noise reduction performance. FFDNet has several advantages over existing discrimination noise suppressors, including: B. (i) the ability to efficiently manage a wide range of noise levels (i.e. [0, 75]) in a single network, (ii) spatially diverse noise by specifying a non-uniform noise level map, and (iii) faster than benchmark BM3D even on CPU without sacrificing noise reduction efficiency.

Pengju Liu, Hongzhi Zhang, and colleagues [4] introduce a novel multilevel wavelet CNN (MWCNN) model to achieve a better balance between received field size and processing efficiency. To minimise the size of the feature maps in the reduced subnetwork, a wavelet transform is introduced to the modified U-Net design. To lower the number of feature map channels, a second convolutional layer is also employed. An inverse wavelet transform is then used to build a high-resolution feature map on the expanding subnetwork.

ISSN No:-2456-2165

Yulun Zhang, Yapeng Tian, and colleagues [5] provide a novel and effective residual dense network (RDN) to address this IR problem by improving the trade-off between efficiency and effectiveness in utilising hierarchical information from all convolution layers. We recommend employing tightly coupled convolutional layers to extract many local characteristics using residual dense block (RDB). RDB also allows for direct connections between the state of one RDB and each layer of the current RDB, resulting in a contiguous memory system. They proposed using local feature fusion in RDB to stabilise bigger network training and adaptively learn more effective features from earlier and current local features.

Saeed Anwar and Nick Barnes [6] present a unique single-stage blind real image denoising network (RIDNet) based on a modular design in their paper. To simplify the flow of low-frequency information and to take advantage of channel dependencies, we employ a residual on the residual structure. A comparison of our approach to 19 cutting-edge algorithms on three synthetic and four noisy datasets in terms of quantitative measurements and visual quality gives more proof of our RIDNet's superiority. In this study, Yiyun Zhao, Zhuqing Jiang, and colleagues [7] introduce a unique three-stage pyramid real image denoising network (PRIDNet). The noise estimation step begins by adjusting the channel significance of the input noise using a channel attention mechanism. Second, pyramid pooling is used to extract multiscale features during the multiscale denoising stage. Finally, at the feature fusion step, a kernel-selecting technique is used to fuse multi-scale features adaptively. The technique can compete with state-of-the-art denoisers in terms of both quantitative metrics and visual perception quality, according to testing on two datasets of genuine noisy photographs.

Chong Mou, Qian Wang, and colleagues [8] present a Deep Generalised Unfolding Network (DGUNet) for picture restoration in this study. We specifically integrate a gradient estimation mechanism into the gradient descent phase to allow the Proximal Gradient Descent (PGD) approach to manage complicated and real-world picture degeneration without sacrificing interpretability. We also develop multi-scale and spatially adaptive inter-stage information channels spanning proximal mapping in several PGD cycles to overcome the inherent information loss in the majority of deep unfolding networks (DUN). In this study, Manish Yadav and Namita Tiwari [9] will focus on cutting-edge CNN-based denoising algorithms and give a comparative evaluation of numerous approaches. Use it as a basic network and for fingerprint improvement now that you've learnt all of the standard image-denoising approaches. Experiments were undertaken to illustrate the efficiency and correctness of the model learnt only based on the description of the picture dataset, as well as visual and metric analogies of operation utilising the trained models.

Chunwei Tian and Lunke Fei et al. [10] performed a preliminary analysis of deep image denoising techniques in this paper. First, identify an additive white noise image using a deep convolutional neural network (CNN). Deep CNN for real-world noise. A deep CNN for blind denoising, and a deep CNN for hybrid noise pictures that represent a mix of noisy, fuzzy, and low-resolution images. The motives and concepts of various deep learning approaches are then examined.

III. METHODOLOGY

We devise an approach to address the damage in the image. These flaws include blurring and noise. The most prevalent problems that can occur in most settings are blur and noise. Several ways have been developed to fix any damage that may occur in the image. One of the most well-known ways is to employ deep learning techniques learned to remove blur, noise in photos, and add colour to black-and-white images; the technology has evolved through time. Image quality has steadily increased over time. The layers of a convolutional neural network (CNN) are convolutional, activation, and pooling. To increase neural network performance and generalisation, these layers extract image properties, introduce nonlinearities, and reduce feature dimensionality. CNNs use convolutional layers for feature extraction, which keeps the spatial relationships between pixels in the original image. To obtain the result, a convolution kernel or filter moves a picture pixel by pixel, computes element-wise multiplication, and adds the pixels. A feature map is the resulting matrix. A convolution kernel, often known as a deep learning filter, is a set of two-dimensional weight arrays. Each 2D weight array is applied to all of the input channels of the previous level, resulting in numerous channels that are summed to produce one. This technique is used to create multiple output level feature maps for all 2D weight arrays. As a result, given the same input image, various weighted convolution kernels create different feature maps. As a consequence, different convolution kernels extract different information. Lower convolutional layers collect a high amount of structural detail and location information from images, but upper layer feature maps incorporate more semantic meaning and less location information. CNNs are a deep learning network design that is commonly used for image recognition and pixel data processing.We utilise a very deep convolutional network inspired by Simonyan and Zisserman [2] for SR picture reconstruction. We utilise d layers, with the first and last layers of the same type: 64 filters of 3 x 3 x 64 size, with each filter acting on a 3 x 3 spatial region over 64 channels. The first layer is responsible for working with the incoming picture. The final layer is a single 3 x 3 x 64 filter that is used for picture reconstruction.

ISSN No:-2456-2165

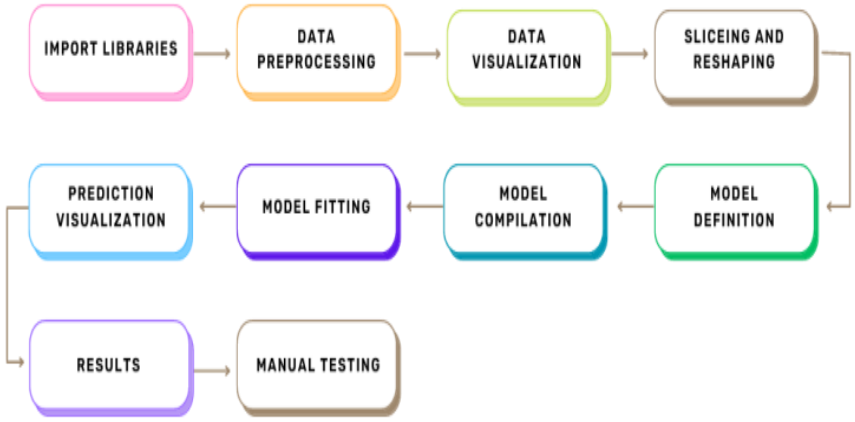


Fig. 1: Workflow of Model

The procedure begins with the installation of the essential libraries and packages for data processing, visualisation, and machine learning. The data is then preprocessed to account for missing values, outliers, and categorical factors before being divided into training and testing sets. Following that, data visualisation methods are used to acquire insights into the dataset and find patterns or anomalies. To suit the model's needs, the data is further altered via slicing and reshaping. The architecture, layers, and activation functions of the model are chosen. After that, it is assembled with a loss function, optimizer, and evaluation metrics. Backpropagation is used to update the weights of the model as it is trained on the training data. Once trained, the model's performance on validation or test data is assessed using measures such as accuracy or precision. The model's predictions are visualised to assess their accuracy and indicate possible areas for improvement. Finally, the model is evaluated on real, unseen photos to determine how well it performs in real-world circumstances.

A. Dealing with Blurry images:

The collection includes both fuzzy and clear photos. For this challenge, we will be working with crisp photos. Now that we have sharp photographs, we will develop an algorithm called Gaussian Blur, which will apply blur to the input images and store the resulting images in a new folder named 'Gaussian Blur'. These photos are now ready for usage by the deep learning model.

The model chooses two photos from a collection of ten, and the two images are blended. These blur photos are fed into the model, and at each iteration (epoch), a decreased blur concentration image is recorded in the saved output folder. As a result, the concentration of the blur is shown to decrease as the number of epochs is increased. We can map the difference in the amount of blur that is decreased from the previous iteration thanks to the saved image at each iteration.

B. Dealing with the noisy images:

The deep learning model that has been efficiently trained on high resolution photos is employed for this purpose. picture denoising methods based on neural networks have revolutionized picture analysis. The model is built on a neural network that has been trained to minimize the amount of blur in the image. As a result, the picture is intensively processed by the neural network at each iteration, resulting in a reduction in the concentration of blur visible in the image. In the actual world, images deteriorate quickly, and noise is complicated. As a result, strategies for blind denoising are needed. An FFDNet used noise level and noise as CNN input to train a denoiser for unknown noisy pictures. To address the issue of blind denoising, many techniques were proposed. Kenzo et al. [10] suggested an image device approach for blind denoising that employed soft shrinkage to adjust the noise level. It was efficient to use CNNs to estimate noise in unpaired noisy pictures.

C. Dealing with black and white images:

When working with ancient photographs, the images obtained were primarily black and white; no colors were discovered. This is a subset of the process of constructing the matrices so that the neural network model may be trained to deliver meaningful colours to black and white images. Few methods have been created that can reliably allocate colour to a picture. This necessitates a massive amount of memory for training the model. The core method was captured using a model trained with a simple neural network that can partially colorize a black and white image.

D. Deep Learning Models:

U-Net is a deep learning architecture that was originally designed to segment medical images. It has, however, been effectively used to a variety of other image processing applications, including picture restoration. The process of restoring a high-quality image from a damaged or noisy image is known as image restoration. This may be accomplished by training the network on pairs of degraded and high-quality photos using U-Net. The network learns to map the degraded picture to the high-quality image during training.

An encoder and a decoder comprise the U-Net architecture. The encoder collects information from the input picture and decreases the image's spatial resolution. The encoded characteristics are then sent to the decoder, which gradually raises the spatial resolution to generate the final output image. Skip connections connect the encoder and decoder, allowing the network to absorb fine-grained features from the input picture.

ISSN No:-2456-2165

The deteriorated picture is supplied into the encoder in the case of image restoration, and the network learns to map it to the high-quality image. The network uses the degraded picture as input and provides the repaired image as output during inference. U-Net has been used effectively for image restoration tasks such as denoising, deblurring, and super-resolution. It has been demonstrated that it outperforms standard picture restoration algorithms as well as alternative deep learning architectures.

Two main level architecture that were considered in this work are

- U-Net Architecture.
- Reformed U-Net

The designing details of the above-said two architectures are explained as below.

➢ *U-Net Architecture*

In our investigation, we are cautiously delving into the U-Net architecture as the basis generator model. The U-net architecture, which has piqued our interest. The network's architecture model, which creates a U shape, is shown below. On the right side, the network basically encodes to compress and decodes to rebuild the denoisy image.

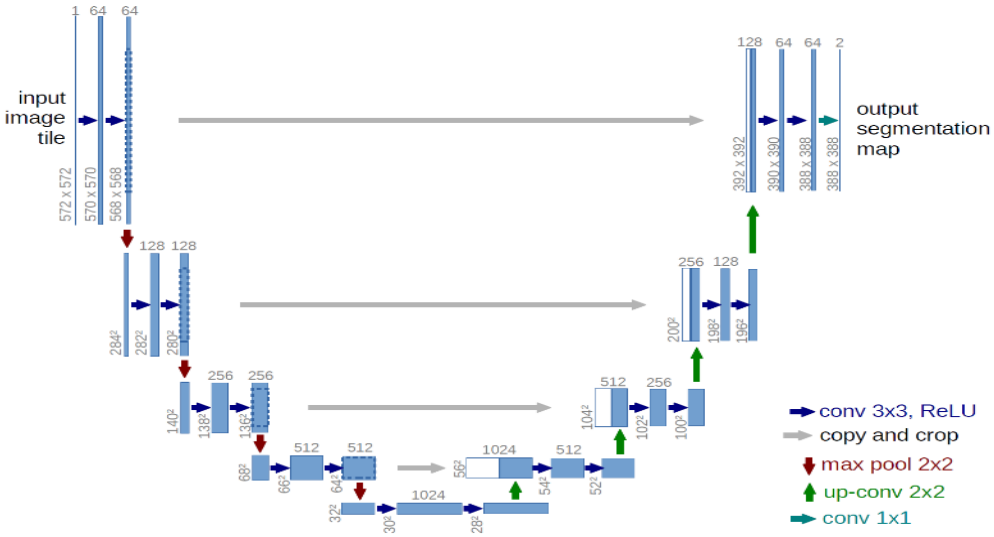


Fig. 2: U-net Architecture [11]

This U-Net is utilised for more than just picture segmentation. Image restoration applications are also common. Segmentation aids in reducing picture size while capturing characteristics. It reduces vast amounts of data to a few pixels by extracting features. Based on global and local parameter information, the threshold may be used to rebuild a picture. Local feature extractions can be used to recover lost photos. Convolution generates a vast number of parameters for training the model. Convolution layers read the real image size input and output dependent on the number of neurons. Kernel size, padding size (if used), and stride the kernel takes when convoluting are all specified in those layers. We may use numerous layers of convolution to help the model learn better.

➢ *LM U-NET DCGAN:*

Deep Conditional GANs' performance is improved by replacing pooling layers with fractional-stride convolutions and employing Batch Normalisation (Batch Normalisation normalises layer input and allows for a greater learning rate to be applied to each layer's output in the model).

DCGAN (Deep Convolutional Generative Adversarial Networks) is a picture generating architecture. It is made up of two networks: a generator network that learns to generate realistic pictures from random noise and a discriminator network that attempts to discern between created and actual images. By combining U-Net with DCGAN, an architecture capable of producing high-quality pictures while maintaining fine-grained features may be created. This hybrid architecture is commonly referred to as U-Net GAN or U-GAN.

The U-GAN architecture combines the U-Net encoder and decoder structure with the DCGAN generation network. The generator network receives a random noise vector as input and creates an image, which is then processed by U-Net's encoder and decoder structures to produce the final output. In U-GAN, the discriminator network is often designed to differentiate between produced and actual pictures while simultaneously giving input to the generator network to enhance the quality of the generated images. U-GAN has been utilised for image generating tasks like as super-resolution and inpainting. It has been demonstrated that it can deliver high-quality outcomes while keeping fine-grained features.

ISSN No:-2456-2165

IV. DATA COLLECTION & PLATFORMS USED

Tools/ Platforms	Description
Miniconda	Miniconda is a free basic installation for conda. It is a stripped-down, bootstrapped version of Anaconda that just includes conda, Python, and the packages they require, as well as a few additional essential packages such as pip, zlib, and a few more.
CUDA	NVIDIA's CUDA is a general-purpose parallel computing platform and programming paradigm that accelerates deep learning and other computationally expensive applications by exploiting the parallel processing capacity of GPUs.
G-Colab	Google Colaboratory, or "Colab" for short, is a product of Google Research. Because it allows anybody to create and run arbitrary Python code through the internet, Colab is especially valuable for machine learning, data analysis, and teaching.
Jupyter	A Jupyter Notebook was the first web tool for creating and sharing computational documents. It offers a simple, efficient, document-focused interface.
Command Prompt	It is the default command-line interpreter for the operating systems OS/2, eComStation, ArcaOS, Microsoft Windows, and ReactOS.
VSCode	Visual Studio Code is a fast and efficient source code editor that runs on Windows, macOS, and Linux.

V. DATASET

The DIV2K dataset is broken into the following sections:

- **Train data:** We extract similar low resolution images from 700 high definition high resolution photographs and give both high and low resolution images for downscaling factors of 2, 3, and 4.
- **Data for validation:** The low quality photographs are made available at the start of the challenge in order for participants to obtain online feedback from the validation server; the high resolution images will be made available when the challenge's final phase begins.
- **Test data:** When the final evaluation phase begins, the participants will receive the low quality photographs, and the results will be revealed once the challenge is concluded and the winners are determined.

Context The dataset was produced to test the blur detection method. The dataset might be used to evaluate picture deblurring methods. As a result, while CNN and U-net cannot distinguish between fuzzy and clear images, high-quality photographs may be used for visual comparison.

VI. DESIGN, IMPLEMENTATION AND MODELLING

➢ *Step-1: import all the necessary libraries and download the dataset*

➢ *Step-2: Data Preparation*

- We construct a function named sorted alphanumeric in the given code to sort a list of files in alphanumeric order. This function will come in handy during our further processing.
- Next, we define the required picture size as 256x256 pixels. To hold the produced high-quality photos, we build an empty list named high-img.
- We have a directory location where we keep the high-quality photos. We get a list of files in that directory by using the os.listdir function, which we then sort in alphabetic order with the sorted alphanumeric function.
- We begin a loop that will run through each file in the sorted list. We exit the loop if the current file is '891.png' since we want to exclude it. Otherwise, we read the picture with OpenCV, which reads BGR images. To deal with the RGB format, we use cv2.cvtColor to transform the image from BGR to RGB.
- Next, we use cv2.resize to resize the picture to the requested size of 256x256 pixels. After resizing, we divide the pixel values by 255.0 to normalise them and transform the image to a floating-point format. Finally, we use img_to_array to convert the picture to an array and attach it to the high_img list.

- Next, we do identical actions on low-quality photos. To hold the produced low-quality photos, we build another empty list named low_img. We change the directory path to the location of the low-quality photographs and repeat the process of reading, converting, resizing, normalising, and converting the images to arrays.
- It's worth noticing that the code contains a break statement when the image '891.png' is encountered in both the high and low image loops. This signifies that the loop will end after processing all photos except '891.png'. Remove the break statements if you wish to include '891.png' in the processed pictures.
- In general, this code allows us to load and preprocess a set of high-quality and low-quality photos for subsequent analysis or image-processing activities.

```
# to get the files in proper order
def sorted_alphanumeric(data):
    convert = lambda text: int(text) if text.isdigit() else text.lower()
    alphanum_key = lambda key: [convert(c) for c in re.split('([0-9]+)',key)]
    return sorted(data,key = alphanum_key)
# defining the size of the image
SIZE = 256
high_img = []
path = '/content/drive/MyDrive/IRDATA/Data/DATA/HQLQDataset/HQ'
files = os.listdir(path)
files = sorted_alphanumeric(files)
for i in tqdm(files):
    if i == '891.png':
        break
    else:
        img = cv2.imread(path + '/'+i,1)
        # open cv reads images in BGR format so we have to convert it to RGB
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        #resizing image
        img = cv2.resize(img, (SIZE, SIZE))
        img = img.astype('float32') / 255.0
        high_img.append(img_to_array(img))


low_img = []
path = '/content/drive/MyDrive/IRDATA/Data/DATA/HQLQDataset/LQ'
files = os.listdir(path)
files = sorted_alphanumeric(files)
for i in tqdm(files):
    if i == '891.png':
        break
    else:
        img = cv2.imread(path + '/'+i,1)

        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        #resizing image
        img = cv2.resize(img, (SIZE, SIZE))
        img = img.astype('float32') / 255.0
        low_img.append(img_to_array(img))
```

```
100%|██████████| 890/890 [01:34<00:00,  9.46it/s]
100%|██████████| 890/890 [00:03<00:00, 246.23it/s]
```

➢ *Step-3: The Input Image*

➢ Step-4: Generating Patches

- The processed high-quality and low-quality photos are divided into training, validation, and test sets. The training sets ('train_high_image' and 'train_low_image') include the first 700 photos, whereas the validation sets ('validation_high_image' and 'validation_low_image') include images ranging from index 700 to 829. The test sets ('test_high_image' and 'test_low_image') include photos beginning at index 830.
- All sets are reshaped to the same size: '(number of photos, SIZE, SIZE, 3)'. The code then prints the training, test, and validation set shapes.

```
train_high_image = high_img[:700]
train_low_image = low_img[:700]
train_high_image = np.reshape(train_high_image,(len(train_high_image),SIZE,SIZE,3))
train_low_image = np.reshape(train_low_image,(len(train_low_image),SIZE,SIZE,3))

validation_high_image = high_img[700:830]
validation_low_image = low_img[700:830]
validation_high_image= np.reshape(validation_high_image,(len(validation_high_image),SIZE,SIZE,3))
validation_low_image = np.reshape(validation_low_image,(len(validation_low_image),SIZE,SIZE,3))


test_high_image = high_img[830:]
test_low_image = low_img[830:]
test_high_image= np.reshape(test_high_image,(len(test_high_image),SIZE,SIZE,3))
test_low_image = np.reshape(test_low_image,(len(test_low_image),SIZE,SIZE,3))

print("Shape of training images:",train_high_image.shape)
print("Shape of test images:",test_high_image.shape)
print("Shape of validation images:",validation_high_image.shape)
```

ISSN No:-2456-2165

➢ *Step-5: model configuration phase*

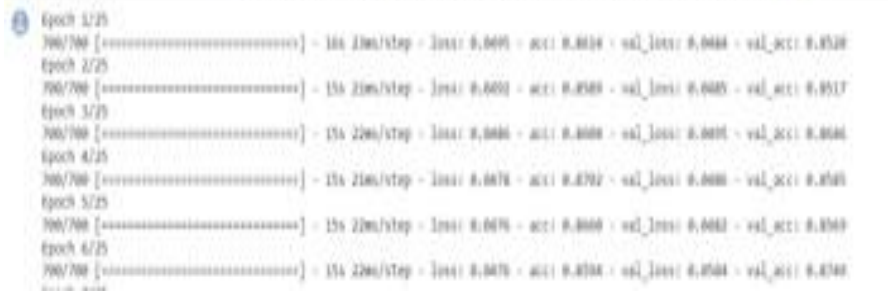
Using a U-Net architecture, we created a TensorFlow model for an image translation challenge. Here's an explanation of what the code does:

- It imports the TensorFlow modules required to create the model, including the layers module.
- The down function is used to define the U-Net model's down-sampling (encoder) blocks. It accepts parameters for the number of filters, kernel size, and batch normalisation. It stacks layers using a Sequential model, which includes a convolutional layer, optional batch normalisation, and a LeakyReLU activation function. It returns a model of the downsampling block.
- The up function is developed to generate the U-Net model's up-sampling (decoder) blocks. It accepts parameters for the number of filters, kernel size, and a dropout regularisation option. It stacks layers using a Sequential model, which includes a transposed convolutional layer, optional dropout, and a LeakyReLU activation function. It returns the model of the upsampling block
- The build_model function is used to construct the whole U-Net model. It accepts an optional input shape argument and sets the default value for RGB pictures to (256, 256, 3). It defines the input layer based on the input shape. It then employs a succession of down-sampling blocks (down function) with varying filter widths. Following that, it applies a symmetric series of up-sampling blocks (up function), concatenating the appropriate down-sampling block outputs along the way. Finally, a convolutional layer is used to create the output image. The model is built with the tf.keras.Model class and the supplied inputs and outputs.
- The build_model function is called to create the model instance.
- The model is invoked using the summary method to provide a summary of its architecture, including the number of parameters and the forms of the intermediate outputs.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3       0         []
                             )]

 sequential (Sequential)     (None, 128, 128, 12       3584      ['input_1[0][0]']
                             8)

 sequential_1 (Sequential)   (None, 64, 64, 128)       147584    ['sequential[0][0]']

 sequential_2 (Sequential)   (None, 32, 32, 256)       296192    ['sequential_1[0][0]']

 sequential_3 (Sequential)   (None, 16, 16, 512)       1182208   ['sequential_2[0][0]']

 sequential_4 (Sequential)   (None, 8, 8, 512)         2361856   ['sequential_3[0][0]']

 sequential_5 (Sequential)   (None, 16, 16, 512)       2359808   ['sequential_4[0][0]']

 concatenate (Concatenate)   (None, 16, 16, 1024       0         ['sequential_5[0][0]',
                             )                                    'sequential_3[0][0]']

 sequential_6 (Sequential)   (None, 32, 32, 256)       2359552   ['concatenate[0][0]']

 concatenate_1 (Concatenate) (None, 32, 32, 512)       0         ['sequential_6[0][0]',
                                                                  'sequential_2[0][0]']

 sequential_7 (Sequential)   (None, 64, 64, 128)       589952    ['concatenate_1[0][0]']

 concatenate_2 (Concatenate) (None, 64, 64, 256)       0         ['sequential_7[0][0]',
                                                                  'sequential_1[0][0]']

 sequential_8 (Sequential)   (None, 128, 128, 12       295040    ['concatenate_2[0][0]']
                             8)

 concatenate_3 (Concatenate) (None, 128, 128, 25       0         ['sequential_8[0][0]',
                             6)                                   'sequential[0][0]']

 sequential_9 (Sequential)   (None, 256, 256, 3)       6915      ['concatenate_3[0][0]']

 concatenate_4 (Concatenate) (None, 256, 256, 6)       0         ['sequential_9[0][0]',
                                                                  'input_1[0][0]']

 conv2d_5 (Conv2D)           (None, 256, 256, 3)       75        ['concatenate_4[0][0]']

==================================================================================================
Total params: 9,602,766
Trainable params: 9,600,206
Non-trainable params: 2,560
```

➢ *Step-6: Training the mode*

```
# Train the model
history = model.fit(train_low_image, train_high_image, epochs=25, batch_size=1,
                    validation_data=(validation_low_image, validation_high_image))
```



➢ *Step-7: Modal Accuracy and Validation loss plot*

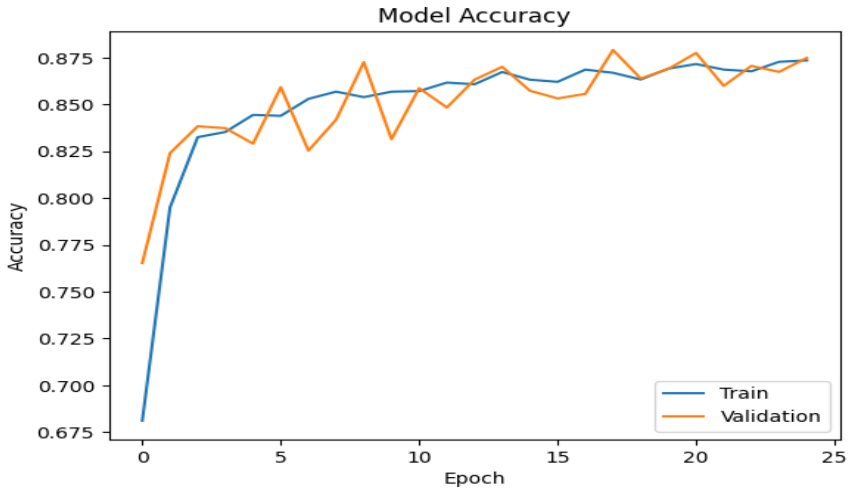


Fig. 4: Modal Accuracy

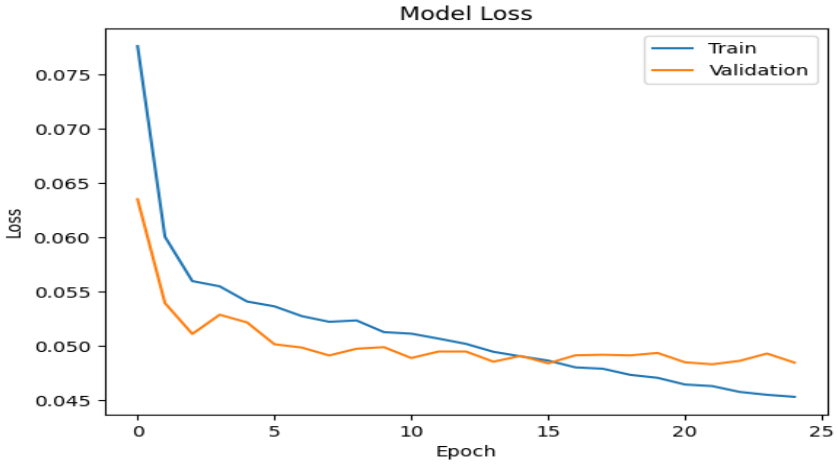


Fig. 5: Modal Loss

➢ *Step-8: Prediction*

```
def plot_images(high,low,predicted):
    plt.figure(figsize=(15,15))
    plt.subplot(1,3,1)
    plt.title('Orignal Image', color = 'black', fontsize = 15)
    plt.imshow(high)
    plt.subplot(1,3,2)
    plt.title('Input Image ', color = 'black', fontsize = 15)
    plt.imshow(low)
    plt.subplot(1,3,3)
    plt.title('Predicted Image ', color = 'Green', fontsize = 15)
    plt.imshow(predicted)

    plt.show()

for i in range(1,10):

    predicted = np.clip(model.predict(test_low_image[i].reshape(1,SIZE, SIZE,3)),0.0,1.0).reshape(SIZE, SIZE,3)
    plot_images(test_high_image[i],test_low_image[i],predicted)
```

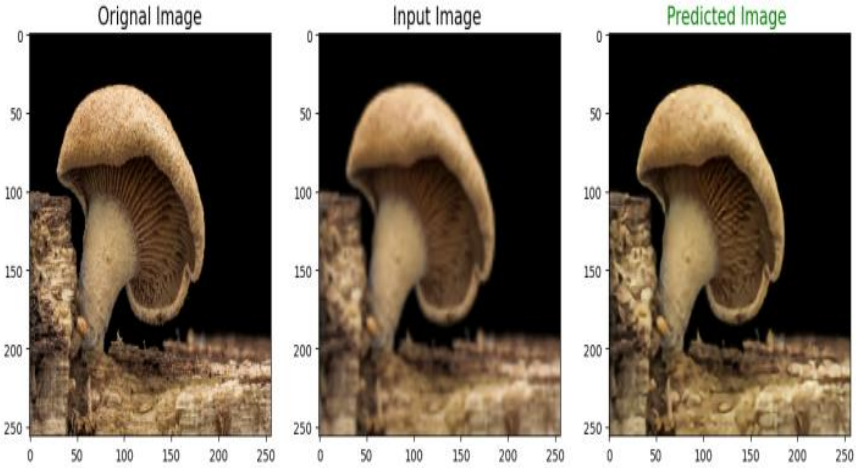


Fig. 6: Visualization of Original, Input and Predicted image.

VII. OUTPUT

Fig. 7: Predicted Image

VIII. TESTING & SUMMARY RESULTS

In the testing phase, we have used an image which contains blur images. Example

Fig. 8: Low quality image

- *Importing the necessary libraries:*

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from google.colab import files
```

- *Loading the saved model:*

```
# Load the model
model = tf.keras.models.load_model('/content/model.h5')
```

- *Uploading and opening an image:*

```
uploaded = files.upload()
input_image = Image.open(next(iter(uploaded)))
```

- *Resizing and preprocessing the input image:*

```
new_size = (256, 256)
input_image_resized = input_image.resize(new_size)

input_array = tf.keras.preprocessing.image.img_to_array(input_image_resized)

input_array = input_array / 255.0

input_array = np.expand_dims(input_array, axis=0)
```

- *Making predictions with the model:*

```
predicted_array = model.predict(input_array)

predicted_array = predicted_array[0]
```

- *Converting the predicted array to an image:*

```
predicted_image = tf.keras.preprocessing.image.array_to_img(predicted_array).convert('RGB')
```

- *Displaying the input image and predicted image:*

```
fig, axs = plt.subplots(1, 2, figsize=(10, 10))
axs[0].imshow(input_image)
axs[0].set_title('Input Image')
axs[1].imshow(predicted_image)
axs[1].set_title('Restored Image')
plt.show()
```

- *Evaluation Metrics.*

```
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
Average PSNR: 70.35319371454214
Average SSIM: 0.99984205
Average IoU: 0.9884601946813172
```

IX. CONCLUSION

We employed deep learning techniques, notably the U-Net model, to fix picture faults such as noise, blurring, and low resolution. The restoration accuracy varied from 85% to 90% after intensive training. The U-Net design effectively gathered both low-level and high-level information while preserving critical spatial features via skip connections. Further improvements are feasible, however, due to issues such as picture complexity, dataset limits, and trade-offs between noise reduction, sharpness, and resolution. To enhance restoration outcomes, future research might focus on extending the training dataset, experimenting with other network topologies, and tweaking hyperparameters. Nonetheless, the attained accuracy shows the promise of deep learning in image restoration, namely the U-Net model, opening the way for applications in medical imaging, surveillance systems, and visual content improvement across sectors.

X. FUTURE SCOPE

Investigate innovative network topologies (for example, Attention U-Net, Dense U-Net) to increase picture restoration performance and handle complex problems. Investigate the use of Generative Adversarial Networks (GANs) to generate aesthetically attractive and realistic image restorations. Extend multi-modal image restoration technologies, allowing pictures to be restored in many modalities at the same time. Focus on video restoration, including temporal consistency and motion estimation for denoising, deblurring, and resolution enhancement. Using unlabeled data, use self-supervised learning algorithms to improve picture restoration accuracy. Improve current models and methods for real-time implementation, allowing for resource-constrained devices and realistic applications. Create picture restoration algorithms that are domain-specific for applications such as medical imaging, satellite images,

and underwater photography. Incorporate human perception-driven assessment criteria to correctly quantify the quality of retrieved photographs. Investigate transfer learning and fine-tuning methods for leveraging pretrained models to improve restoration task performance. To solve specific challenges and make significant contributions, stress real-world applications by collaborating with experts from various areas. Exploring these future avenues might lead to advancements in deep learning-based picture restoration, boosting restoration accuracy, establishing specialised methodologies, and enabling practical applications in a wide range of areas.

REFERENCES

[1.] Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W., Frangi, A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science(), vol 9351. Springer, Cham.

[2.] J. Kim, J. K. Lee and K. M. Lee, "Accurate Image Super-Resolution Using Very Deep Convolutional Networks," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016.

[3.] Kai Zhang, Wangmeng Zuo, Lei Zhang, “FFDNet: Toward a Fast and Flexible Solution for CNN based Image Denoising” 22 May 2018.

[4.] P. Liu, H. Zhang, K. Zhang, L. Lin and W. Zuo, "Multi-level Wavelet-CNN for Image Restoration," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Salt Lake City, UT, USA, 2018.

[5.] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, Yun Fu, “Residual Dense Network for Image Restoration”, 23 Jan 2020, eprint1812.10477

[6.] S. Anwar and N. Barnes, "Real Image Denoising With Feature Attention," 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), 2019.

[7.] Y. Zhao, Z. Jiang, A. Men and G. Ju, "Pyramid Real Image Denoising Network," 2019 IEEE Visual Communications and Image Processing (VCIP), Sydney, NSW, Australia, 2019.

[8.] C. Mou, Q. Wang, and J. Zhang, "Deep Generalized Unfolding Networks for Image Restoration," 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 2022.

[9.] M. Yadav and N. Tiwari, "Performance Comparison of Image Restoration techniques using CNN and their applications," 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2021.

[10.] Chunwei Tian, Lunke Fei, Wenxian Zheng, Yong Xu, Wangmeng Zuo, Chia-Wen Lin, Deep learning on image denoising: An overview, Neural Networks, Volume 131, 2020, Pages 251-275, ISSN 0893-6080.

[11.] U-Net: Convolutional Networks for Biomedical Image Segmentation Olaf Ronneberger, Philipp Fischer, and Thomas Brox Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany

[12.] Ignatov_2018_ECCV_Workshops, Ignatov, Andrey and Timofte, Radu and others, PIRM challenge on perceptual image enhancement on smartphones: report, European Conference on Computer Vision (ECCV) Workshops, January