A thesis on

IMAGE PROCESSING USING GENERATIVE ADVERSARIAL NETWORK

Submitted in the partial fulfillment of the requirements for the degree of

BACHELOR'S OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

by

Sagar Jain (190060101094)
Shashank Singh (190060101101)
Shivani Sharma (190060101103)
Shreya Srivastava (190060101106)

Under the supervision of

Ass. Prof. Ms. Supriya Shukla
Computer Science Department,
College of Engineering Roorkee

Submitted to the
Department of Computer Science and Engineering
College of Engineering Roorkee (COER), Roorkee
Veer Madho Singh Bhandari Uttarakhand Technical University,Uttarakhand-248001
June, 2023

CANDIDATE'S DECLARATION

This is to certify that Thesis/Report entitled , "**Image processing using Generative adversarial network** " which is submitted by me in partial fulfillment of the requirement for the award of degree **B.Tech. in Computer Science & Engineering** to **College of Engineering Roorkee(COER),Veer Madho Singh Bhandari Uttarakhand Technical University, Dehradun, Uttarakhand** comprises only my original work and due acknowledgement has been made in the text to all other material used.

Date:

Sagar Jain (190060101094)
Shashank Singh (190060101101)
Shivani Sharma (190060101103)
Shreya Srivastava (190060101106)
College of Engineering Roorkee

Approved By:

Dr. Sumit Kumar
Head of the department
(Computer Science & Engineering)
College of Engineering Roorkee

CERTIFICATE

It is to certify that the Report entitled **"Image processing using Generative adversarial network"** which is being submitted by **Sagar Jain, Shashank Singh, Shivani Sharma, Shreya Srivastava** in partial fulfillment of the requirement for the award of degree **B.Tech. Computer Science & Engineering to College of Engineering Roorkee,** Veer Madho Singh Bhandari Uttarakhand Technical University, **Dehradun Uttarakhand** is a record of the candidate own work carried out by him under my/our supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Date:

Ms. Supriya Shukla
Assistant Professor of Computer Science Department
College of Engineering Roorkee (COER)

ACKNOWLEDGEMENT

I would like to take this opportunity to express my deep sense of gratitude to all who helped me directly or indirectly during this thesis work.

First of all, I would like to express my deepest gratitude to my thesis supervisor, **Ms. SUPRIYA SHUKLA** for his enormous help and advice and for providing inspiration which can not be expressed with words. I would not have accomplished this thesis without his patient care, understanding and encouragement. His advice, encouragement and critics are a source of innovative ideas, inspiration and causes behind the successful completion of this thesis work. The confidence shown in me by him was the biggest source of inspiration for me.

I am deeply thankful to **College of Engineering Roorkee, Roorkee** and **Veer Madho Singh Bhandari Uttarakhand Technical University, Uttarakhand-248001** for providing facilities for accomplishment of this dissertation

Date:

Sagar Jain (190060101094)

Shashank Singh (190060101101)

Shivani Sharma (190060101103)

Shreya Srivastava (190060101106)

College of Engineering Roorkee

PROJECT APPROVAL SHEET

This is to certify that the project titled

IMAGE PROCESSING USING GENERATIVE ADVERSARIAL NETWORK

By

Sagar Jain (190060101094)
Shashank Singh (190060101101)
Shivani Sharma (190060101103)
Shreya Srivastava (190060101106)

is approved for the degree of Bachelor of technology

Ms. Supriya Shukla Assistant Professor

Computer Science Department

College of Engineering Roorkee (COER), Roorkee

Internal examiner name & Signature :

External examiner name & Signature :

Date:

TABLE OF CONTENT

LIST OF FIGURES

ABSTRACT

The project aims to explore the capabilities of two advanced image processing algorithms, ESRGAN (Enhanced Super-Resolution Generative Adversarial Network) and DeOldify, and their combined application in enhancing and colorizing images. These algorithms leverage the power of deep learning and neural networks to improve image quality and breathe new life into historical or low-resolution images. DeOldify, an algorithm designed to add realistic colors to black and white or faded images. DeOldify utilizes deep learning techniques, including convolutional neural networks and self-attention mechanisms, to infer and apply color information to grayscale images. This process involves training the model on a dataset of colored and grayscale image pairs, enabling it to learn the relationship between grayscale inputs and their corresponding colors.ō

Chapter 1: INTRODUCTION

1.1 Overview of project

Image processing techniques have witnessed significant advancements in recent years, driven by the rise of deep learning and neural networks. Two prominent algorithms in this field are ESRGAN (Enhanced Super-Resolution Generative Adversarial Network) and DeOldify. ESRGAN focuses on upscaling low-resolution images to higher resolutions, while DeOldify specializes in adding realistic colors to grayscale or faded images. This project aims to explore the capabilities of both algorithms and their combined application in enhancing and colorizing images.

We will explore two popular GAN-based image processing techniques

- Image upscaling using ESRGAN

- Colorization using DeOldify

Image upscaling plays a crucial role in various domains, including digital restoration, computer vision, and visual content creation. ESRGAN represents a state-of-the-art approach in this area. By leveraging generative adversarial networks (GANs) and deep learning techniques, ESRGAN can generate high-resolution images that exhibit improved sharpness, textures, and overall visual quality. The algorithm's ability to learn complex mappings between low-resolution and high-resolution images allows it to restore missing details and enhance the overall appearance of images.

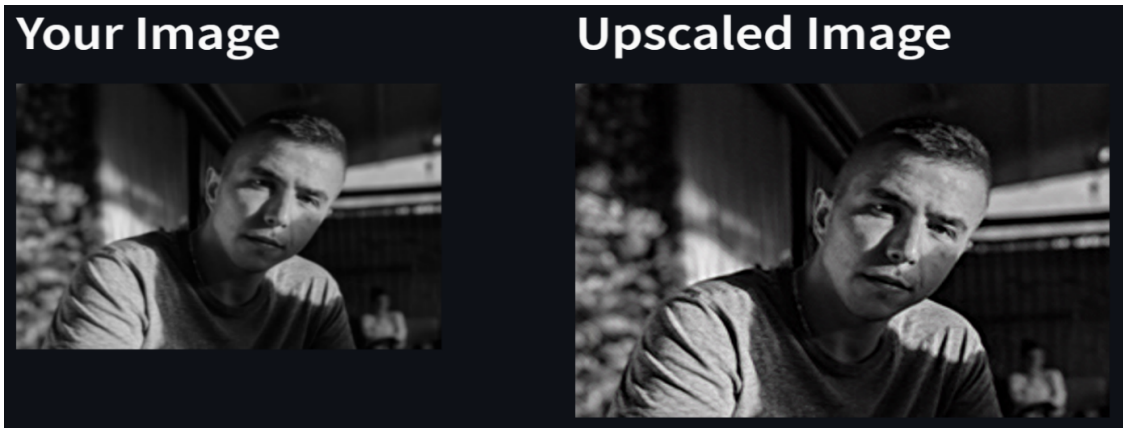


Fig no. 1.1 Upscaling of an image.

Fig no. 1.2 Upscaling of an image after zoom in.

On the other hand, DeOldify focuses on colorization, particularly targeting black and white or faded images. This algorithm harnesses the power of convolutional neural networks (CNNs) and self-attention mechanisms to infer and apply realistic colors to grayscale inputs. By training on a dataset of colored and grayscale image pairs, DeOldify learns the relationship between grayscale images and their corresponding colors. The algorithm preserves the coherence and historical context of images reviving old photographs and enhancing visual storytelling.

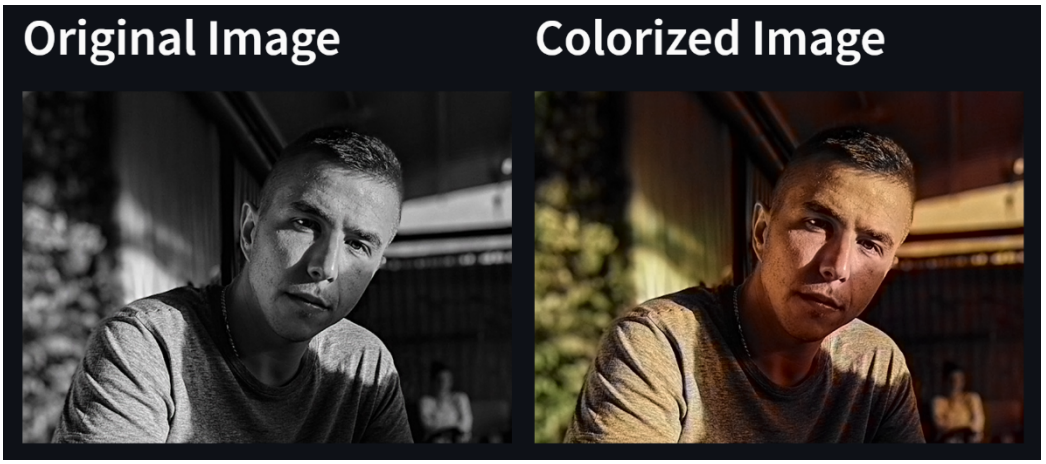


Fig no. 1.3 Colourization of an image

The objectives of this project include implementing ESRGAN and DeOldify algorithms, evaluating their individual performances, and exploring the effectiveness of their combination. The project aims to assess the quality of upscaled images using ESRGAN, measure the realism and accuracy of colorizations using DeOldify, and investigate the potential applications of the combined approach in various scenarios.

By studying and experimenting with these algorithms, this project aims to contribute to the advancement of image processing techniques and their practical applications. The outcomes of this research have the potential to find applications in digital restoration, cultural heritage preservation, entertainment, and other fields that rely on high-quality image enhancement and colorization.

In conclusion, the project focuses on exploring the capabilities of ESRGAN and DeOldify algorithms in image processing tasks. By combining the power of deep learning and neural networks, these algorithms offer the potential to upscale low-resolution images and apply realistic colorizations. The project aims to evaluate their performance and investigate the applicability of their combined approach in various domains, ultimately contributing to the advancement of image processing techniques and their practical implementations.

1.1 Problem Statement

The problem at hand is to develop an efficient and accurate system for upscaling low-resolution images and colorizing grayscale images. Upscaling an image refers to the process of increasing its size while preserving as much detail and clarity as possible. Colorization, on the other hand, involves adding color to grayscale images to make them visually appealing and realistic.

The challenge lies in creating a solution that can effectively upscale low-resolution images without significantly degrading the quality or introducing artifacts. Additionally, the colorization process should accurately predict appropriate colors based on the context and semantics of the grayscale image.

1. Upscaling Algorithm: Develop a robust and efficient algorithm capable of increasing the resolution of low-resolution images while maintaining sharpness and clarity. The algorithm should minimize the loss of important details and avoid introducing artifacts, such as blurriness or pixelation.
2. Colorization Model: Create a machine learning model that can accurately predict color information for grayscale images. The model should learn from a large dataset of color images to understand color patterns, textures, and contextual cues, enabling it to generate realistic and visually pleasing colorizations.
3. Computational Efficiency: Ensure that the proposed solution is computationally efficient, allowing for real-time or near-real-time processing of images. It should leverage optimized algorithms and techniques to minimize processing time while maintaining high-quality results.
4. Generalization and Adaptability: Develop a solution that can generalize well to different types of images, including various subjects, objects, and scenes. It should be able to handle images with different levels of complexity, such as landscapes, portraits, or abstract art.
5. Evaluation Metrics: Establish appropriate evaluation metrics to assess the performance of the system. Metrics such as peak signal-to-noise ratio (PSNR), structural similarity index (SSIM), or perceptual quality assessment can be utilized to quantify the accuracy and visual fidelity of the upscaled and colorized images

Chapter 2: Literature Survey

2.1 Primary Objective.

The primary objective of image processing using GANs is to generate high-quality synthetic images that are similar to real images. GANs are a type of deep learning neural network that consists of two networks, a generator and a discriminator. The generator network generates synthetic images from random noise, while the discriminator network evaluates the quality of the generated images and distinguishes them from real images.
The generator network's goal is to generate images that are realistic enough to fool the discriminator network, while the discriminator's goal is to correctly identify which images are real and which are synthetic. Through an iterative training process, the generator network learns to generate more realistic images, while the discriminator network becomes better at distinguishing between real and synthetic images.
Once trained, the GAN can generate high-quality synthetic images that can be used for a variety of purposes, such as upscaling low-resolution images, colorizing black and white images, and generating realistic images of people, animals, and objects.
The objective of image processing using GANs is to push the limits of what is possible in generating high-quality synthetic images. GANs have the potential to transform various industries, such as entertainment, fashion, and healthcare, by generating synthetic images that can be used for a wide range of applications, from creating realistic virtual worlds to aiding in medical diagnoses.

2.2 Technologies And Framework Used.

These libraries provide a range of functions and tools for building and training GANs for upscaling and colorization of images, making it easier for developers and researchers to experiment with and improve these techniques.

- *TensorFlow*
- *PyTorch*
- *Keras*
- *OpenCV*

2.2.1 TensorFlow:

TensorFlow is a popular open-source library for machine learning and deep learning developed by Google. It provides various tools and APIs for building and training neural networks, including GANs, and can be used for upscaling and colorization of images.

1. Data Preparation: The first step is to prepare the training data, which typically involves collecting a large dataset of low-resolution or black and white images and their corresponding high-resolution or color images.

2. Model Architecture: The next step is to define the architecture of the GAN model, which typically consists of a generator network that upscales or colorizes the input image, and a discriminator network that distinguishes between the generated and real images.
3. Training: The GAN model is then trained on the training dataset using backpropagation and stochastic gradient descent. During training, the generator network learns to generate high-quality images that can fool the discriminator network, while the discriminator network learns to distinguish between the generated and real images.
4. Evaluation: Once the GAN model has been trained, it can be evaluated on a separate validation dataset to assess its performance in generating high-quality, high-resolution color images.

2.2.2 PyTorch:

PyTorch is another popular open-source library for machine learning and deep learning developed by Facebook. It provides an efficient and flexible platform for building and training neural networks, including GANs, and can be used for upscaling and colorization of images.

1. Dataset Preparation:
   - Obtain a labeled dataset for image classification. The dataset should be divided into training and validation sets.
   - Use PyTorch's **torchvision.datasets** module to load popular datasets like CIFAR-10, MNIST, or ImageNet. Alternatively, you can create a custom dataset by subclassing **torch.utils.data.Dataset** and implementing the necessary methods.
2. Data Preprocessing:
   - Apply necessary preprocessing techniques to prepare the images for training. This may include resizing, normalization, and data augmentation (e.g., random cropping, flipping, rotation) to increase the diversity of training samples.
   - Use PyTorch's **torchvision.transforms** module to apply transformations to the dataset.
3. Model Definition:
   - Define your classification model architecture using PyTorch's **torch.nn** module. This typically involves creating a custom class that inherits from **torch.nn.Module** and implementing the **__init__** and **forward** methods.
   - Design your model by stacking various layers, such as convolutional layers (**torch.nn.Conv2d**), pooling layers (**torch.nn.MaxPool2d**), fully connected layers (**torch.nn.Linear**), and activation functions (**torch.nn.ReLU**).
4. Model Training:
   - Create an instance of your defined model.
   - Specify the loss function, such as cross-entropy loss (**torch.nn.CrossEntropyLoss**).
   - Choose an optimizer, such as stochastic gradient descent (**torch.optim.SGD**) or Adam (**torch.optim.Adam**).

- Iterate over the training dataset in batches and perform the following steps:
    1. Pass the batch of images through the model to obtain predicted class probabilities.
    2. Compute the loss between the predicted probabilities and the ground truth labels.
    3. Backpropagate the gradients and update the model parameters using the optimizer.
- Repeat the training process for multiple epochs, adjusting the learning rate and other hyperparameters as needed.

5. Model Evaluation:
    - Evaluate the trained model on the validation set or a separate test set.
    - Iterate over the validation/test dataset in batches and compute the model's predictions.
    - Calculate evaluation metrics such as accuracy, precision, recall, or F1-score to assess the model's performance.
6. Model Deployment:
    - Once you have a trained model, you can save it to disk using PyTorch's **torch.save** function.
      You can load the saved model using **torch.load** for later inference or deployment on new data.

2.2.3 Keras:

Keras is a high-level deep learning library that provides an intuitive interface for building and training neural networks, including GANs. It can be used for upscaling and colorization of images and is compatible with TensorFlow and Theano.

1. Dataset Preparation:
    - Obtain a labeled dataset for image classification. The dataset should be divided into training and validation sets.
    - Ensure that the images are organized in the appropriate directory structure (e.g., separate folders for each class).
    - You can use external libraries or tools to load and preprocess the images, such as OpenCV or scikit-image.

2. Data Preprocessing:
    - Apply necessary preprocessing techniques to prepare the images for training. This may include resizing, normalization, and data augmentation (e.g., random cropping, flipping, rotation) to increase the diversity of training samples.
    - You can use Keras' **ImageDataGenerator** class to perform data augmentation and create data iterators for efficient loading and preprocessing of images.

3. Model Definition
    - Import the necessary Keras modules: **from keras.models import Sequential** and **from keras.layers import ...** (where **...** represents

different layer types).

- Define your classification model architecture using the **Sequential** API. This involves creating a sequential stack of layers.
- Design your model by adding various layers, such as convolutional layers (**Conv2D**), pooling layers (**MaxPooling2D**), fully connected layers (**Dense**), and activation functions (**Activation**).
- Specify the input shape of your images in the first layer (e.g., **input_shape=(width, height, channels)**).

4. Model Compilation:
    - Compile your model by specifying the loss function, optimizer, and evaluation metrics.
    - Choose an appropriate loss function based on your classification problem, such as categorical cross-entropy (**loss='categorical_crossentropy'**) for multi-class classification.
    - Select an optimizer, such as stochastic gradient descent (**optimizer='sgd'**) or Adam (**optimizer='adam'**), and set its learning rate and other hyperparameters.
    - Specify evaluation metrics, such as accuracy (**metrics=['accuracy']**), to monitor the model's performance during training.

2.3.4 OpenCV

OpenCV is an open-source computer vision library that provides various tools and functions for image processing, including upscaling and colorization. It can be used in combination with GANs for image enhancement and manipulation.

1. Dataset Preparation:
    - Obtain a labeled dataset for image classification. The dataset should be divided into training and validation sets.
    - Ensure that the images are organized in the appropriate directory structure (e.g., separate folders for each class).

2. Feature Extraction:
    - Use OpenCV to extract relevant features from your images. Depending on the classification approach, you can extract handcrafted features or use pre-trained deep learning models for feature extraction.
    - For handcrafted features, you can utilize OpenCV's functions for image preprocessing, such as resizing, color conversion, or filtering. You can also apply techniques like Histogram of Oriented Gradients (HOG) or Local Binary Patterns (LBP) to extract texture or shape features.
    - Alternatively, you can use pre-trained deep learning models available in other libraries/frameworks (e.g., Keras, PyTorch) to extract features. OpenCV can be used to preprocess and resize the images before passing them through the pre-trained models.

2.3 Convolution Neural Network

Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3

dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions (number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class scores.

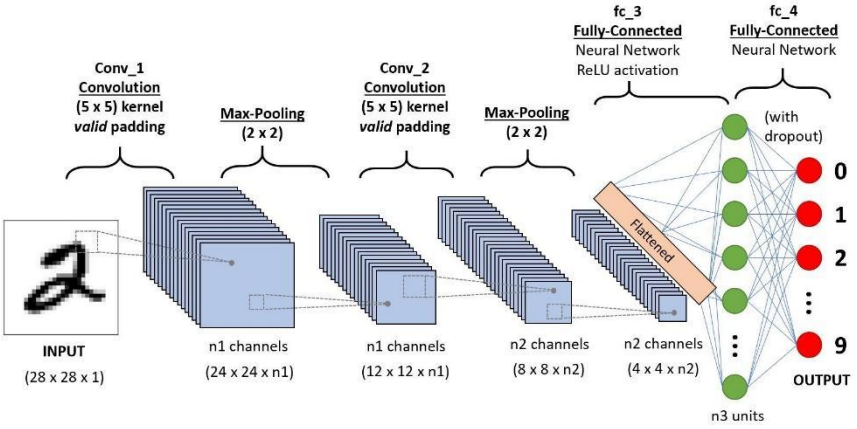


Fig no. 2.1 Convolutional neural network.

2.3.1 Convolution Layer

In convolution layer we take a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consists of learnable filters of window size. During every iteration we slide the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position. As we continue this process well create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color.

2.3.2 Pooling Layer

We use a pooling layer to decrease the size of the activation matrix and ultimately reduce the learnable parameters. There are two types of pooling:

a) **Max Pooling:** In max pooling we take a window size [for example window of size 2*2], and only take the maximum of 4 values. Well lid this window and continue this process, so well finally get an activation matrix half of its original Size.

b) **Average Pooling:** In average pooling we take the average of all values in a window.

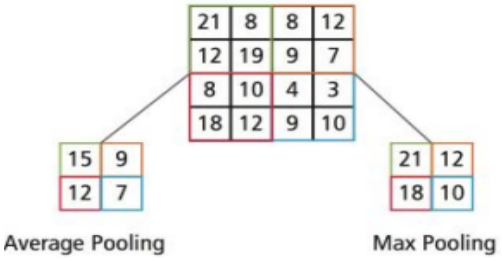


Fig no.2.2 Pooling and its types.

2.3.3 Fully Connected Layer

In convolution layers neurons are connected only to a local region, while in a fully connected region, well connect all the inputs to neurons.

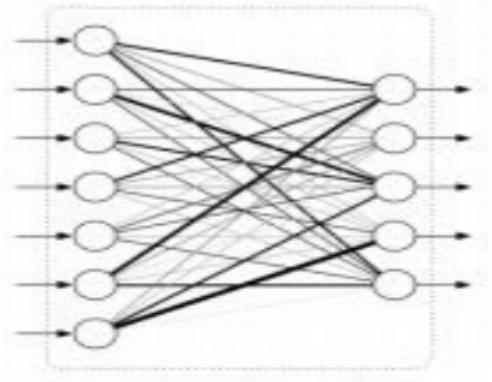


Fig no.2.3 Fully connected layer.

2.3.4 Final Output Layer

After getting values from a fully connected layer, we will connect them to the final layer of neurons [having count equal to total number of classes], that will predict the probability of each image to be in different classes.

2.4 GAN (GENERATIVE ADVERSARIAL NETWORK):

A generative adversarial network (GAN) is a machine learning (ML) model in which two neural networks compete with each other by using deep learning methods to become more accurate in their predictions. GANs typically run unsupervised and use a cooperative zero-sum game framework to learn, where one person's gain equals another person's loss.

The two neural networks that make up a GAN are referred to as the *generator* and the *discriminator*. The generator is a convolutional neural network and the discriminator is a deconvolutional neural network. The goal of the generator is to artificially manufacture outputs that could easily be mistaken for real data. The goal of the discriminator is to identify which of the outputs it receives have been artificially created.

A GAN typically takes the following steps:

1. The generator outputs an image after accepting random numbers.
2. The discriminator receives this created image in addition to a stream of photos from the real, ground-truth data set.
3. The discriminator inputs both real and fake images and outputs probabilities -- a value between 0 and 1 -- where 1 indicates a prediction of authenticity and 0 indicates a fake.

GANs are typically divided into the following three categories:
Generative. This describes how data is generated in terms of a probabilistic model.
Adversarial. A model is trained in an adversarial setting.
Networks. Deep neural networks can be used as artificial intelligence (AI) algorithms for training purposes.

2.4.1 ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks

In this, image super-resolution (SR) techniques reconstruct a higher-resolution (HR) image or sequence from the observed lower-resolution (LR) images, e.g. upscaling of four times an image(LR) .
One of the common approaches to solving this task is to use deep convolutional neural networks capable of recovering HR images from LR ones. And ESRGAN (Enhanced SRGAN) is one of them. Key points of ESRGAN:

- SRResNet-based architecture with residual-in-residual blocks.
- Mixture of context, perceptual, and adversarial losses. Context and perceptual losses are used for proper image upscaling, while adversarial loss pushes neural networks to the natural image manifold using a discriminator network that is trained to differentiate between the super-resolved images and original photo-realistic images.

2.4.2 DeOldify

It is using Generative Adversarial Networks with the iterative interplay between two Neural Networks Generator and Discriminator (like in ArtBreeder). But differently to the last model, the images in DeOldify aren't modified or generated in their form. Power of GAN brings colors — Generator applies colors to the recognized objects he's trained on, and Discriminator tries to criticize the color choice. DeOldify is based on the fast.ai library which brings more power and optimization for deep learning developers.
Flowchart for the colorization process using DeOldify. Here's a textual representation of the flowchart:

1. **Input**: The flowchart begins with the input of a grayscale or faded image that requires colorization.
2. **Preprocessing**: Preprocessing steps are applied to prepare the input image for colorization. This may include resizing the image, adjusting brightness/contrast, or any necessary preprocessing specific to the DeOldify algorithm.
3. **Colorization**: The DeOldify algorithm is applied to the preprocessed image. It

involves feeding the image through a deep learning model trained on a large dataset of colored and grayscale image pairs. The model predicts color information based on the grayscale input, utilizing convolutional neural networks (CNNs) and self-attention mechanisms.

4. **Postprocessing**: The colorized image goes through postprocessing steps to refine and enhance the output. This may involve adjusting color saturation, contrast, or other parameters to achieve the desired visual result.
5. **Output:** The final colorized image is obtained as the output of the process

2.5 Objective of project.

The objective of a project in image processing that involves upscaling using ESRGAN and colorization using DeOldify could be to enhance and transform images to improve their visual quality and appearance. The project might aim to achieve the following specific goals:

1. Upscaling using ESRGAN:
    - Develop or utilize an ESRGAN model to enhance the resolution and details of low-resolution images.
    - Explore and experiment with different ESRGAN architectures, loss functions, and training strategies to improve the quality of upscaled images.
    - Evaluate the performance of the ESRGAN model by comparing the upscaled images with their original low-resolution counterparts and other upscaling techniques.

2. Colorization using DeOldify:
    - Utilize the DeOldify technique to add color to black-and-white or faded images, restoring them to a more vibrant and realistic appearance.
    - Investigate and employ appropriate pre-trained DeOldify models or train custom models using relevant datasets.
    - Experiment with different colorization strategies, such as grayscale-guided or reference-based colorization, to achieve accurate and visually pleasing results.
    - Evaluate the colorization performance by comparing the colorized images with the original colored images or ground truth data, if available.

3. Integration and user experience:
    - Develop a user-friendly interface or application that allows users to upscale and colorize images using ESRGAN and DeOldify, respectively.
    - Provide options for users to adjust parameters, choose specific color palettes, or fine-tune the results according to their preferences.
    - Optimize the processing speed and efficiency to ensure a smooth and responsive user experience.

Overall, the objective of the project would be to leverage the capabilities of ESRGAN and DeOldify to enhance the resolution and color of images, resulting in visually appealing and high-quality outputs. The project may involve exploring different techniques, evaluating performance, and creating a user-friendly application for image upscaling and colorization.

2.6 Methodology

Image processing using Generative Adversarial Networks (GANs) involves training a GAN model to generate new images that possess desired characteristics or to enhance existing images. Here is a general methodology for image processing using GANs:

1. Define the objective: Determine the specific task or objective of your image processing project. For example, it could be image super-resolution, image inpainting, style transfer, or image-to-image translation. Clearly defining the goal will help guide the subsequent steps.

2. Gather and preprocess the dataset: Collect a dataset of images that are relevant to your objective. Ensure the dataset is diverse and representative of the images you want the GAN to generate or modify. Preprocess the images as necessary, such as resizing, cropping, or normalizing pixel values, to ensure consistency and ease of training.

3. Choose a GAN architecture: Select an appropriate GAN architecture for your image processing task. Popular GAN architectures include Vanilla GAN, Deep Convolutional GAN (DCGAN), Conditional GAN (cGAN), CycleGAN, Pix2Pix, and Progressive GAN (PGAN). Each architecture has its strengths and suitability for different tasks, so choose one that aligns with your objective.

4. Design the generator and discriminator networks: For most GAN architectures, you'll need to define the generator and discriminator networks. The generator generates new images, while the discriminator evaluates the authenticity of both real and generated images. Design the architecture of these networks based on the complexity and characteristics of your image processing task.

5. Train the GAN: Train the GAN model using the prepared dataset. During training, the generator and discriminator networks compete against each other in a min-max game. The generator tries to produce realistic images, while the discriminator aims to correctly classify real and generated images. This adversarial training process helps the generator improve its image generation quality.

6. Monitor and tune the training: Keep track of the training progress by monitoring metrics like generator and discriminator losses, image quality, and diversity. If the results are not satisfactory, consider adjusting hyperparameters such as learning rate, batch size, or network architecture. Experimentation and iteration are essential to achieve optimal results.

7. Evaluate and validate the generated images: After training, evaluate the quality and fidelity of the generated images. Use quantitative metrics (e.g., peak signal-to-noise ratio or structural similarity index) or qualitative assessments by human evaluators to gauge the effectiveness of the GAN model. Fine-tune or iterate on the model if necessary.

8. Apply the GAN for image processing: Once the GAN model is trained and validated, you can apply it to perform image processing tasks. Generate new

images with desired characteristics, enhance low-resolution images, inpaint missing regions, or perform other transformations based on the trained GAN's capabilities.

9. Post-process the output: Depending on the task, you may need to perform post-processing on the generated images. This could involve color adjustment, denoising, contrast enhancement, or any other techniques to refine the output and make it more visually appealing or suitable for your application.

Remember that the exact methodology and steps may vary depending on the specific GAN architecture and image processing task you are working on. It is essential to refer to the documentation and guidelines of the GAN architecture you choose and adapt the methodology accordingly.

2.7 BasicSR

The BasicSR library is an open-source library specifically designed for image super-resolution tasks. It provides a collection of state-of-the-art deep learning models and associated tools for various aspects of image super-resolution, including model training, evaluation, and deployment. The library is implemented in PyTorch and offers a comprehensive set of functionalities for researchers and practitioners working in the field of image super-resolution. Here are some key features and components of the BasicSR library:

1. **Model Architectures:** BasicSR includes several popular model architectures for image super-resolution, such as SRResNet, ESRGAN, EDVR, and RCAN. These architectures have achieved excellent performance in terms of improving the resolution and quality of low-resolution images.
2. **Data Loaders:** The library provides data loaders to handle common image super-resolution datasets, including DIV2K, Set5, Set14, and others. It also offers the flexibility to incorporate custom datasets.
3. **Loss Functions:** BasicSR offers various loss functions commonly used in image super-resolution, including pixel-wise losses like L1 loss and perceptual losses like VGG loss and adversarial losses like GAN loss. These loss functions enable effective training of the super-resolution models.
4. **Training Utilities:** The library provides utilities for training the image super-resolution models, including optimizer configurations, learning rate scheduling, checkpoint saving and loading, and training/validation logging. These tools help streamline the model training process.
5. **Evaluation Metrics:** BasicSR includes commonly used evaluation metrics for assessing the performance of image super-resolution models. These metrics include peak signal-to-noise ratio (PSNR), structural similarity index (SSIM), and others.
6. **Pretrained Models:** BasicSR offers pretrained models for different architectures, allowing users to directly use these models for image super-resolution tasks or as a starting point for further fine-tuning.
7. **Inference and Deployment:** The library provides tools and utilities to facilitate model inference and deployment. This includes scripts and functions for processing and generating high-resolution images using the trained models.
8. **Visualization and Analysis:** BasicSR offers functions and utilities for visualizing and analyzing the results of image super-resolution, such as comparing low-resolution and high-resolution images, generating visualizations of model predictions, and analyzing model performance

The BasicSR library aims to provide a comprehensive and accessible framework for image super-resolution research and applications. It allows researchers and practitioners to experiment with different model architectures, train models on custom datasets, and evaluate the performance of image super-resolution models using various metrics.

2.8 Important Libraries.

FaceXlib: facexlib aims at providing ready-to-use face-related functions based on current SOTA open-source methods.
Only PyTorch reference codes are available. For training or fine-tuning, please refer to their original repositories listed below.
Note that we just provide a collection of these algorithms. You need to refer to their original LICENCEs for your intended use.

GFPGAN: GFPGAN aims at developing a Practical Algorithm for Real-world Face Restoration.
It leverages rich and diverse priors encapsulated in a pretrained face GAN (e.g., StyleGAN2) for blind face restoration.

Streamlit: Streamlit is an open-source Python library that allows you to create interactive web applications for machine learning, data analysis, and other purposes with ease. It simplifies the process of building and sharing interactive web interfaces for your Python code, making it accessible to users without the need for web development skills.
Key features and benefits of Streamlit include:

1. Rapid Prototyping: Streamlit enables you to quickly prototype and iterate on your ideas by providing a simple API that allows you to turn your Python scripts into interactive apps.
2. Easy-to-Use Interface: With Streamlit, you can create user interfaces using familiar Python syntax. It provides a set of intuitive commands and functions to add various elements such as sliders, dropdowns, buttons, plots, and text to your app.
3. Live Code Editing: Streamlit automatically updates your app in real-time as you modify your code. This feature enables you to see the changes immediately, making the development process efficient.
4. Seamless Integration: Streamlit seamlessly integrates with popular Python libraries and frameworks such as pandas, matplotlib, Plotly, and TensorFlow. You can leverage the power of these libraries to analyze data, create visualizations, and incorporate machine learning models into your apps.
5. Sharing and Deployment: Streamlit makes it easy to share your apps with others. You can deploy your apps to a variety of platforms, including cloud services like Heroku or sharing them as standalone web applications.

Chapter 3 Materials and Method

3.1 Technologies

We have used deep learning (GAN) concepts to build this project.

3.1.1 Deep Learning:

Deep learning has become very popular in scientific computing, and businesses dealing with complex problems often use its techniques. All deep learning algorithms employ various types of neural networks in order to perform specific tasks. Artificial neural networks are used in deep learning to perform complex calculations on large amounts of data. It is a form of artificial intelligence based on the organization and function of the human brain. Machines are trained using deep learning algorithms by learning from examples. Deep learning is often used in areas such as healthcare, e-commerce, entertainment, and advertising.

Deep Learning Algorithms:

Some important algorithms of deep learning are listed below :

CNN- Convolutional Neural Networks.
LSTM- Long Short Term Memory Networks.
RNN- Residual Neural Networks.
GAN- Generative Adversarial Networks.
RBFN- Radial Basis Function Networks.
MLP- Multilayer perceptron.
SOM-Self-Organizing Maps.
DBM- Deep Belief Networks.
RBM- Restricted Boltzmann Devices.
Auto encoders.
Deep learning methods require a lot of data and computing power to address complex issues. They are capable of handling almost any kind of data.

3.1.1.1 Convolutional Neural Networks (CNNs):

The main applications of CNNs, also known as ConvNets, are object identification and image processing. Yann Le Can founded the original CNN in 1988, while it was still known as LeNet. This is used to distinguish between symbols like ZIP codes and numerals. CNNs are used for a variety of tasks, including anomaly detection, time series forecasting, medical imaging processing, and satellite image recognition. By using a number of layers CNNs analyze and also filter the features from data.

- **Convolution Layer**
  The convolution layer used by CNN contains a number of filters to carry out the convolution function. A Rectified Linear Unit (ReLU) layer, which is used to modify elements, is found in CNNs. It produces a revised feature map.

- **Layering Pools**
  The updated feature map is then applied to a layer of pooling. "Pooling" is a down sampling technique that reduces the size of the feature map. The

resultant two-dimensional arrays from the pooled feature map are then flattened by the pooling layer into a solitary, lengthy continuous linear vector. The pooling layer's flattened matrix serves as the input for creating a fully linked layer that classifies and identifier

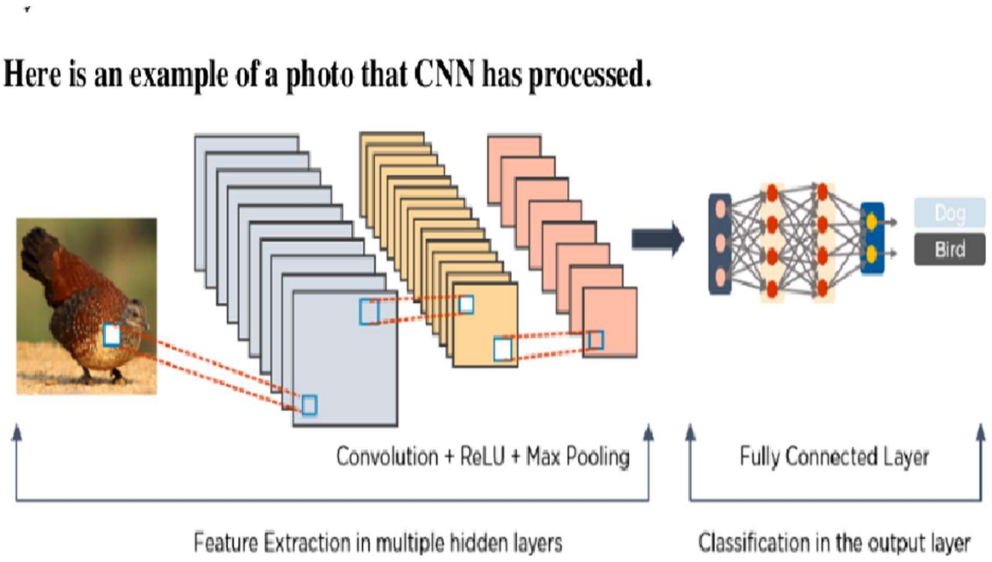


Fig 3.1 Processing of CNN

We have also used multiple frameworks and libraries to build the Drowsiness Detection System. The list of framework and libraries used are as follows:

1. Python
2. Yolov5
3. Pytorch
4. Cv2

Below is a brief introduction of all the technologies and frameworks that we used in our project.

3.1.2 Python:

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high- level programming language. Python is a dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python is very high in demand and all the major companies are looking for great Python Programmers to develop websites, software components, and applications or to work with Data Science, AI, and ML technologies. When we are developing this tutorial in 2022, there is a high shortage of Python Programmers whereas the market demands more number of Python Programmers due to its application in Machine Learning, Artificial Intelligence etc.

Today a Python Programmer with 3-5 years of experience is asking for around $150,000 annual package and this is the most demanding programming language in America. Though it can vary depending on the location of the Job. It's impossible to list all of the companies using Python, to name a few big companies are:

- Google
- Intel
- NASA
- PayPal
- Facebook
- IBM
- Amazon
- Netflix
- Pinterest
- Uber
- Many more...

Python is consistently rated as one of the world's most popular programming languages. Python is fairly easy to learn, so if you are starting to learn any programming language then Python could be your great choice. Today various Schools, Colleges and Universities are teaching Python as their primary programming language. There are many other good reasons which makes Python as the top choice of any programmer:

- Python is Open Source which means its available free of cost.
- Python is simple and so easy to learn
- Python is versatile and can be used to create many different things.

- Python has powerful development libraries include AI, ML etc.
- Python is much in demand and ensures high salary

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner- level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

3.1.2.1 Pytorch:

PyTorch is a machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, originally developed by Meta AI and now part of the Linux Foundation umbrella. It is free and open-source software released under the modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.
A number of pieces of deep learning software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers, PyTorch Lightning, and Catalyst.
PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU)
- Deep neural networks built on a tape-based automatic differentiation system

PyTorch provides Tensors that can live either on the CPU or the GPU and accelerates the computation by a huge amount.
We provide a wide variety of tensor routines to accelerate and fit your scientific computation needs such as slicing, indexing, mathematical operations, linear algebra, reductions. And they are fast!

3.1.2.2 Dynamic Neural Networks

Tape-Based Autograd PyTorch has a unique way of building neural networks: using and replaying a tape recorder.
Most frameworks such as TensorFlow, Theano, Caffe, and CNTK have a static view of the world. One has to build a neural network and reuse the same structure again and again. Changing the way the network behaves means that one has to start from scratch. With PyTorch, we use a technique called reverse-mode auto-differentiation, which allows you to change the way your network behaves arbitrarily with zero lag or overhead. Our inspiration comes

from several research papers on this topic, as well as current and past work such as torch-autograd, autograd, Chainer, etc.
While this technique is not unique to PyTorch, it's one of the fastest implementations of it to date. You get the best of speed and flexibility for your crazy research.

3.1.2.3 Python First

PyTorch is not a Python binding into a monolithic C++ framework. It is built to be deeply integrated into Python. You can use it naturally like you would use NumPy / SciPy / scikit-learn etc. You can write your new neural network layers in Python itself, using your favorite libraries and use packages such as Cython and Numba. Our goal is to not reinvent the wheel where appropriate.

3.1.2.4 Imperative Experiences

PyTorch is designed to be intuitive, linear in thought, and easy to use. When you execute a line of code, it gets executed. There isn't an asynchronous view of the world. When you drop into a debugger or receive error messages and stack traces, understanding them is straightforward. The stack trace points to exactly where your code was defined. We hope you never spend hours debugging your code because of bad stack traces or asynchronous and opaque execution engines.

3.1.2.5 Fast and Lean

PyTorch has minimal framework overhead. We integrate acceleration libraries such as Intel MKL and NVIDIA (cuDNN, NCCL) to maximize speed. At the core, its CPU and GPU Tensor and neural network backends are mature and have been tested for years.
Hence, PyTorch is quite fast – whether you run small or large neural networks.
The memory usage in PyTorch is extremely efficient compared to Torch or some of the alternatives. We've written custom memory allocators for the GPU to make sure that your deep learning models are maximally memory efficient. This enables you to train bigger deep learning models than before.

3.1.2.6 Extensions Without Pain

Writing new neural network modules, or interfacing with PyTorch's Tensor API was designed to be straightforward and with minimal abstractions.
You can write new neural network layers in Python using the torch API or your favorite NumPy-based libraries such as SciPy.
If you want to write your layers in C/C++, we provide a convenient extension API that is efficient and with minimal boilerplate. No wrapper code needs to be written

3.1.3 Cv2:

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

cv2.imread() method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.

All three types of flags are described below:

cv2.IMREAD_COLOR: It specifies to load a color image. Any transparency of image will be neglected. It is the default flag. Alternatively, we can pass integer value 1 for this flag.

cv2.IMREAD_GRAYSCALE: It specifies to load an image in grayscale mode. Alternatively, we can pass integer value 0 for this flag.

cv2.IMREAD_UNCHANGED: It specifies to load an image as such including alpha channel. Alternatively, we can pass integer value -1 for this flag.

3.1.4 What is Computer Vision?

The term Computer Vision (CV) is used and heard very often in artificial intelligence (AI) and deep learning (DL) applications. The term essentially means giving a computer the ability to see the world as we humans do.
Computer Vision is a field of study which enables computers to replicate the human visual system. As already mentioned above, it's a subset of artificial intelligence which collects information from digital images or videos and processes them to define the attributes. The entire process involves image acquiring, screening, analyzing, identifying and extracting information. This extensive processing helps computers to understand any visual content and act on it accordingly.
Computer vision projects translate digital visual content into explicit descriptions to gather multi-dimensional data. This data is then turned into a computer-readable language to aid the decision-making process. The main objective of this branch of artificial intelligence is to teach machines to collect information from pixels.

3.1.5 GAN (GENERATIVE ADVERSARIAl NETWORK):

A generative adversarial network (GAN) is a machine learning (ML) model in which two neural networks compete with each other by using deep learning methods to become more accurate in their predictions. GANs typically run unsupervised and use a cooperative zero-sum game framework to learn, where one person's gain equals another person's loss.

The two neural networks that make up a GAN are referred to as the *generator* and the *discriminator*. The generator is a convolutional neural network and the discriminator is a deconvolutional neural network. The goal of the generator is to artificially manufacture outputs that could easily be mistaken for real data. The goal of the discriminator is to identify which of the outputs it receives have been artificially created.

A GAN typically takes the following steps:

1. The generator outputs an image after accepting random numbers.
2. The discriminator receives this created image in addition to a stream of photos from the real, ground-truth data set.
3. The discriminator inputs both real and fake images and outputs probabilities -- a value between 0 and 1 -- where 1 indicates a prediction of authenticity and 0 indicates a fake.

GANs are typically divided into the following three categories:

Generative. This describes how data is generated in terms of a probabilistic model. Adversarial. A model is trained in an adversarial setting networks. Deep neural networks can be used as artificial intelligence (AI) algorithms for training purposes.

3.1.6 ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks

In this, image super-resolution (SR) techniques reconstruct a higher-resolution (HR) image or sequence from the observed lower-resolution (LR) images, e.g. upscaling of four times an image(LR) .
One of the common approaches to solving this task is to use deep convolutional neural networks capable of recovering HR images from LR ones. And ESRGAN (Enhanced SRGAN) is one of them. Key points of ESRGAN:

- SRResNet-based architecture with residual-in-residual blocks;
- Mixture of context, perceptual, and adversarial losses. Context and perceptual losses are used for proper image upscaling, while adversarial loss pushes neural networks to the natural image manifold using a discriminator network that is trained to differentiate between the super-resolved images and original photo-realistic images.

3.1.7 DeOldify:

DeOldigy is using Generative Adversarial Networks with the iterative interplay between two Neural Networks Generator and Discriminator (like in ArtBreeder). But differently to the last model, the images in DeOldify aren't modified or generated in their form. Power of GAN brings colors — Generator applies colors to the recognized objects he's trained on, and Discriminator tries to criticize the color choice. DeOldify is based on the fast.ai library which brings more power and optimization for deep learning developers.

Flowchart for the colorization process using DeOldify. Here's a textual representation of the flowchart:

1. 1.Input: The flowchart begins with the input of a grayscale or faded image that requires colorization.
2. Preprocessing: Preprocessing steps are applied to prepare the input image for colorization. This may include resizing the image, adjusting brightness/contrast, or any necessary preprocessing specific to the DeOldify algorithm.
3. Colorization: The DeOldify algorithm is applied to the preprocessed image. It involves feeding the image through a deep learning model trained on a large dataset of colored and grayscale image pairs. The model predicts color information based on the grayscale input, utilizing convolutional neural networks (CNNs) and self-attention mechanisms.
4. Postprocessing: The colorized image goes through postprocessing steps to refine and enhance the output. This may involve adjusting color saturation, contrast, or other parameters to achieve the desired visual result.
5. Output: The final colorized image is obtained as the output of the process

3.2 Material and Method

3.2.1 Datasets:

- Labeled image datasets: Obtain labeled datasets that include low-resolution images for upscaling and grayscale images for colorization. Popular datasets for image processing tasks include ImageNet, COCO, CIFAR-10, DIV2K, and CelebA.
- High-resolution images: Gather high-resolution images to evaluate the quality of the upscaling algorithms and colorization results.

3.2.2 Image Processing Libraries and Frameworks:

- OpenCV: OpenCV is a widely used open-source library for image processing. It provides a comprehensive set of functions and algorithms for image manipulation, preprocessing, and feature extraction.
- NumPy: NumPy is a fundamental library for scientific computing in Python. It provides efficient data structures and operations for numerical calculations, which are often used in image processing tasks.
- Scikit-image: Scikit-image is a Python library specifically designed for image processing tasks. It offers a wide range of functions for image enhancement, filtering, segmentation, and more.
- PyTorch or TensorFlow: Deep learning frameworks like PyTorch or TensorFlow can be utilized for building and training deep neural networks for upscaling and colorization tasks. They provide extensive support for image processing and deep learning operations.

3.2.3 Pretrained Models and Architectures:

- Pretrained models: Utilize pretrained models, such as VGG, ResNet, or U-Net, for image feature extraction or as a starting point for fine-tuning in upscaling and colorization tasks.
- GANs: Generative Adversarial Networks (GANs) have been successfully used for image-to-image translation tasks, including image super-resolution and colorization. Pretrained GAN models like SRGAN or Pix2Pix can be used or adapted for specific project needs.

3.2.4 Research Papers and Publications:

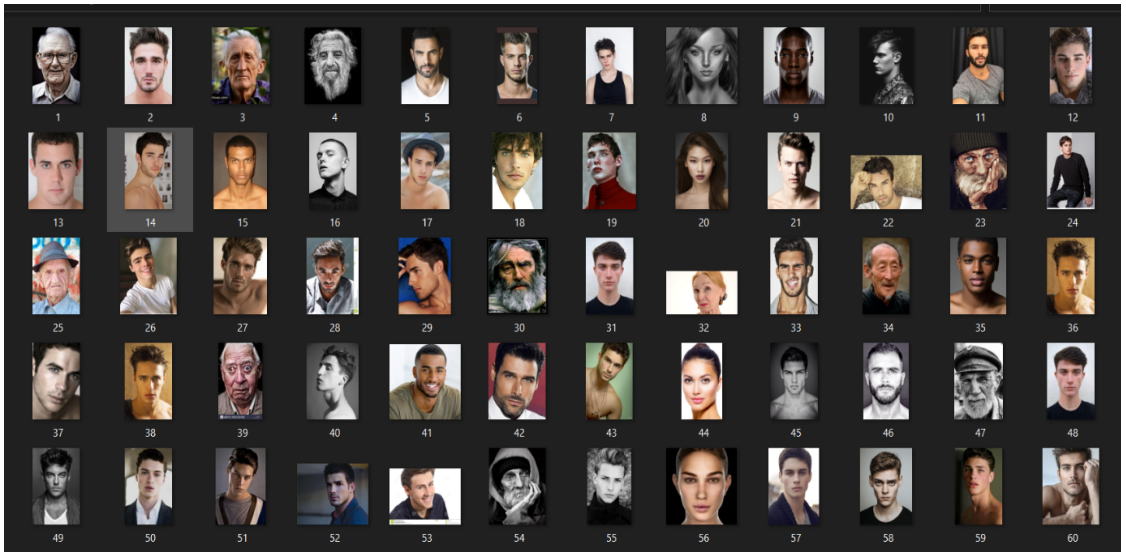
- Explore research papers and publications in the field of image processing, computer vision, and deep learning. These papers often introduce novel algorithms, techniques, or architectures specifically tailored for upscaling and colorization tasks. ArXiv, IEEE Xplore, and Google Scholar are valuable resources for finding relevant papers.

3.2.5 Programming Languages and Tools:

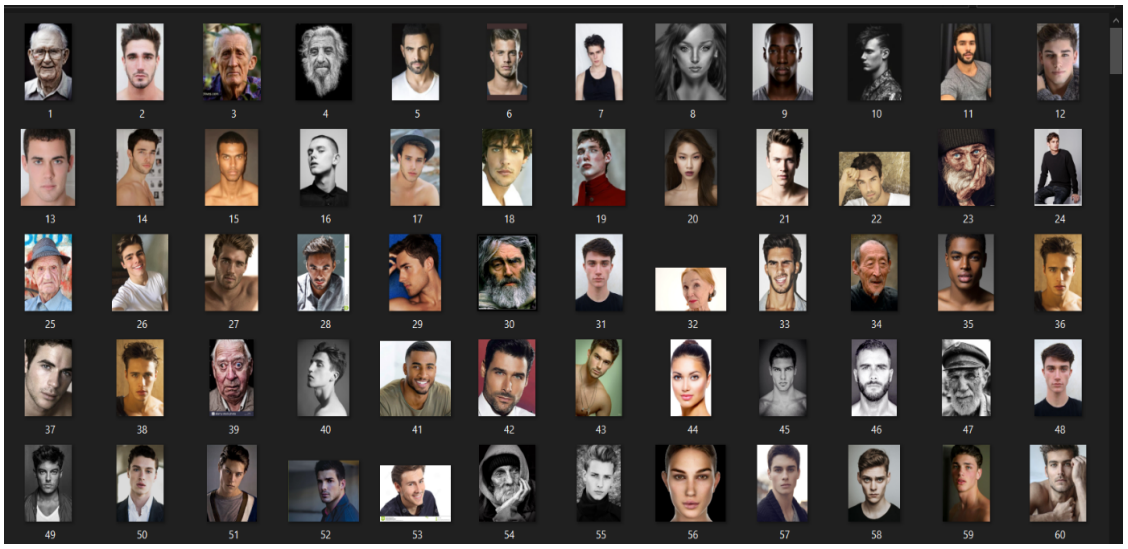
- Python: Python is a popular programming language for image processing tasks due to its extensive libraries and ease of use. Python libraries like NumPy, OpenCV, and scikit-image are commonly used for implementing image processing algorithms.
- Jupyter Notebooks: Jupyter Notebooks provide an interactive and collaborative

environment for prototyping and experimenting with image processing algorithms.

3.2.6 Our Dataset:



Low resolution images



High resolution image

3.2.7 Basic SR:

Basic SR, or Basic Super-Resolution, refers to a fundamental technique used in image processing to enhance the resolution or quality of low-resolution images. The goal of super-resolution is to generate a higher-resolution version of an image from its lower-resolution counterpart.
The basic SR approach typically involves the following steps:

1. Input Image: Begin with a low-resolution image that needs to be enhanced or upscaled.
2. Image Preprocessing: Apply any necessary preprocessing steps, such as noise reduction or image alignment, to prepare the image for super-resolution.
3. Learning or Interpolation: Perform a process called learning or interpolation to estimate the missing high-frequency details in the low-resolution image. There are various methods used for this step, including:
4. Nearest Neighbor: This method simply duplicates the existing pixels in the low-resolution image to increase its size. It is the simplest approach but may result in blocky appearance and lack of detail.
5. Bilinear Interpolation: This method takes the average of the surrounding pixels to estimate the values of new pixels. It provides smoother results compared to the nearest neighbor method but may still lack fine details.
6. Lanczos Interpolation: This method uses a more complex interpolation technique, such as the Lanczos kernel, to estimate the values of new pixels. It can produce sharper and more detailed results but may require more computational resources.
7. Other Interpolation Techniques: There are several other interpolation methods available, such as bicubic interpolation, spline interpolation, or Fourier-based interpolation, each with its own advantages and trade-offs.
8. Post-processing: Apply any necessary post-processing steps to refine the enhanced image. This may involve denoising, sharpening, or adjusting color and contrast to achieve the desired result.

It's important to note that basic SR methods may have limitations in terms of the level of detail and quality they can achieve compared to more advanced super-resolution techniques, such as deep learning-based approaches. Deep learning models, particularly convolutional neural networks (CNNs), have shown significant improvements in super-resolution tasks by learning complex mappings between low-resolution and high-resolution image pairs.

3.2.8 ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks

In this, image super-resolution (SR) techniques reconstruct a higher-resolution (HR) image or sequence from the observed lower-resolution (LR) images, e.g. upscaling of four times an image(LR) .
One of the common approaches to solving this task is to use deep convolutional neural networks capable of recovering HR images from LR ones. And ESRGAN (Enhanced SRGAN) is one of them. Key points of ESRGAN:

- SRResNet-based architecture with residual-in-residual blocks;
- Mixture of context, perceptual, and adversarial losses. Context and perceptual losses are used for proper image upscaling, while adversarial loss pushes neural networks to the natural image manifold using a discriminator network that is trained to differentiate between the super-resolved images and original photo-realistic images.

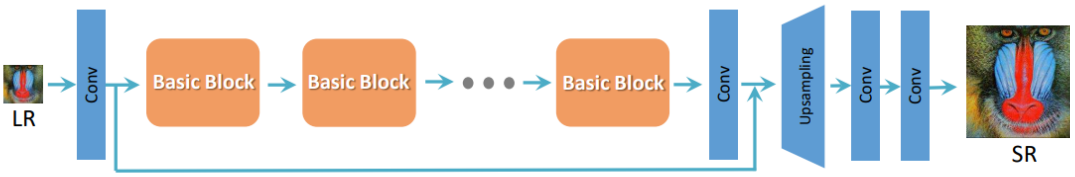


Fig no. 3.2 Enhanced Super-Resolution Generative Adversarial Networks graph

3.2.9 DeOldify:

DeOldify is using Generative Adversarial Networks with the iterative interplay between two Neural Networks Generator and Discriminator (like in ArtBreeder). But differently to the last model, the images in DeOldify aren't modified or generated in their form. Power of GAN brings colors — Generator applies colors to the recognized objects he's trained on, and Discriminator tries to criticize the color choice. DeOldify is based on the fast.ai library which brings more power and optimization for deep learning developers.

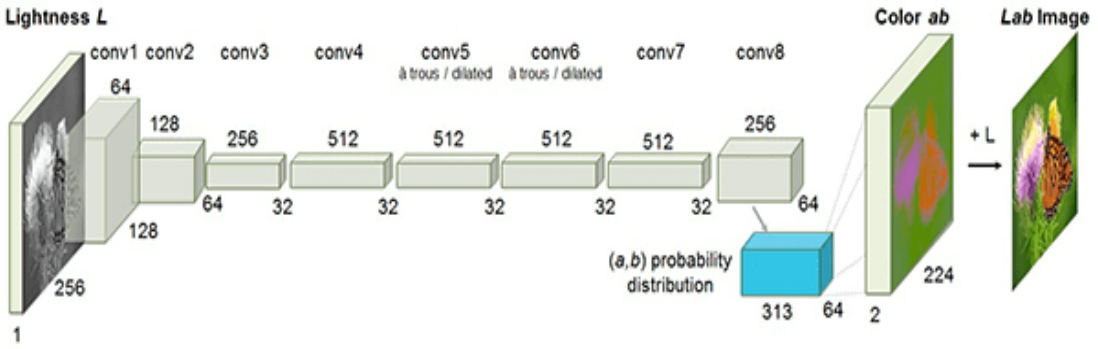


Fig no. 3.3 DeOldify Graph

*In this project we use the **STREAMLIT** library for interactive web applications and data visualizations.*

Some key features and benefits of Streamlit include:

1. **Easy-to-use**: Streamlit is designed to be beginner-friendly and allows you to quickly create interactive web applications with Python.
2. **Data visualization**: Streamlit provides built-in capabilities for creating visualizations, such as line plots, bar charts, maps, and more. You can easily display and explore your data in a visual format.
3. **Fast iteration**: With Streamlit, you can iterate rapidly on your app development. The instant feedback loop allows you to make changes in real-time, making the development process more efficient.
4. **Integration with machine learning libraries:** Streamlit seamlessly integrates with popular Python libraries for machine learning, such as TensorFlow, PyTorch, and scikit-learn. You can easily incorporate your models and visualize their results.
5. **Deployment flexibility:** Streamlit apps can be deployed locally or on various hosting platforms, such as Heroku, AWS, or Microsoft Azure. This enables you to share your interactive apps with others or deploy them for production use.

3.3 Code snippets

Upscaler :

C: > Users > IamSh > Desktop > deoldifywithcustomUpscale > pages > 1_upscale.py > ...

```python
import streamlit as st
import numpy as np
from numpy import asarray
from PIL import Image
from io import BytesIO

import cv2
import os
from realesrgan.archs.srvgg_arch import SRVGGNetCompact
from realesrgan import RealESRGANer


def upscale():

        model_path='models/customESRGAN.pth'

        my_model = SRVGGNetCompact(num_in_ch=3, num_out_ch=3, num_feat=64, num_conv=16, upscale=4, act_type='prelu')
        netscale = 4

        upsampler = RealESRGANer(
        scale=netscale,
        model_path=model_path,
        dni_weight=None,
        model=my_model,
        tile=0,
        tile_pad=10,
        pre_pad=0,
        half=True,
        gpu_id=0)



        if len(img.shape) == 3 and img.shape[2] == 4:
            img_mode = 'RGBA'
        else:
            img_mode = None

```

```
        if len(img.shape) == 3 and img.shape[2] == 4:
            img_mode = 'RGBA'
        else:
            img_mode = None

        try:
            output, _ = upsampler.enhance(img, outscale=4)
        except RuntimeError as error:
            print('Error', error)
        else:


            save_path = os.path.join('result_images/upscaled.jpg')
            output= cv2.GaussianBlur(output, (5, 5), 0)

            blurred_img = cv2.GaussianBlur(output, (0, 0), 7)
            output = cv2.addWeighted(output, 1.7, blurred_img, -0.7, 0)

            cv2.imwrite(save_path, output)


st.title("IMAGE UPSCALER")


img=st.file_uploader("")

col1,col2=st.columns(2)

if img:
    col1.header("Your Image")
    col2.header("Upscaled Image")
    col1.image(img)

    test_img= img

    with col2:
            with st.spinner():
```

```
C: > Users > IamSh > Desktop > deoldifywithcustomUpscale > pages > 1_upscale.py > ...
    with col2:
        with st.spinner():
            bytes_data = test_img.getvalue()
            img = np.array(Image.open(BytesIO(bytes_data)))
            img=np.flip(img, axis=-1)

            upscale()

            #img = img * 1.0 / 255



        col2.image("result_images/upscaled.jpg",width=test_img.width, use_column_width=True, clamp=True, channels='RGB')
        st.balloons()

        with open("result_images/upscaled.jpg") as file:
            btn = col2.download_button(
                    label="Download image",
                    data=file,
                    file_name="upscaled.jpg",
                    mime="image/png"
                )
```

C: > Users > IamSh > Desktop > deoldifywithcustomUpscale > realesrgan > archs > srvgg_arch.py > SRVGGNetCompact

```python
from basicsr.utils.registry import ARCH_REGISTRY
from torch import nn as nn
from torch.nn import functional as F


@ARCH_REGISTRY.register()
class SRVGGNetCompact(nn.Module):

    """A compact VGG-style network structure for super-resolution.

    It is a compact network structure, which performs upsampling in the last layer and no convolution is
    conducted on the HR feature space.

    Args:
        num_in_ch (int): Channel number of inputs. Default: 3.
        num_out_ch (int): Channel number of outputs. Default: 3.
        num_feat (int): Channel number of intermediate features. Default: 64.
        num_conv (int): Number of convolution layers in the body network. Default: 16.
        upscale (int): Upsampling factor. Default: 4.
        act_type (str): Activation type, options: 'relu', 'prelu', 'leakyrelu'. Default: prelu.
    """

    def __init__(self, num_in_ch=3, num_out_ch=3, num_feat=64, num_conv=16, upscale=4, act_type='prelu'):
        super(SRVGGNetCompact, self).__init__()
        self.num_in_ch = num_in_ch
        self.num_out_ch = num_out_ch
        self.num_feat = num_feat
        self.num_conv = num_conv
        self.upscale = upscale
        self.act_type = act_type

        self.body = nn.ModuleList()
        # the first conv
        self.body.append(nn.Conv2d(num_in_ch, num_feat, 3, 1, 1))
        # the first activation
        if act_type == 'relu':
            activation = nn.ReLU(inplace=True)
        elif act_type == 'prelu':
```

```
        self.body.append(nn.Conv2d(num_in_ch, num_feat, 3, 1, 1))
        # the first activation
        if act_type == 'relu':
            activation = nn.ReLU(inplace=True)
        elif act_type == 'prelu':
            activation = nn.PReLU(num_parameters=num_feat)
        elif act_type == 'leakyrelu':
            activation = nn.LeakyReLU(negative_slope=0.1, inplace=True)
        self.body.append(activation)

        # the body structure
        for _ in range(num_conv):
            self.body.append(nn.Conv2d(num_feat, num_feat, 3, 1, 1))
            # activation
            if act_type == 'relu':
                activation = nn.ReLU(inplace=True)
            elif act_type == 'prelu':
                activation = nn.PReLU(num_parameters=num_feat)
            elif act_type == 'leakyrelu':
                activation = nn.LeakyReLU(negative_slope=0.1, inplace=True)
            self.body.append(activation)

        # the last conv
        self.body.append(nn.Conv2d(num_feat, num_out_ch * upscale * upscale, 3, 1, 1))
        # upsample
        self.upsampler = nn.PixelShuffle(upscale)

    def forward(self, x):
        out = x
        for i in range(0, len(self.body)):
            out = self.body[i](out)

        out = self.upsampler(out)
        # add the nearest upsampled image, so that the network learns the residual
        base = F.interpolate(x, scale_factor=self.upscale, mode='nearest')
        out += base
        return out
```

C: > Users > IamSh > Desktop > deoldifywithcustomUpscale > pages > 2_Colorize.py > ...

```python

import streamlit as st
from PIL import Image
import numpy as np

import cv2
from io import BytesIO
import cv2


from deoldify import device
from deoldify.device_id import DeviceId

device.set(device=DeviceId.GPU0)

from deoldify.visualize import *
plt.style.use('dark_background')
torch.backends.cudnn.benchmark=True



colorizer = get_image_colorizer(artistic=True)


st.title("IMAGE COLORIZER")

img1=st.file_uploader("")

col1,col2=st.columns(2)



def color():
    with col2:
        with st.spinner():

            render_factor=20
```

⚠ 6

Chapter 4: Results and Declaration

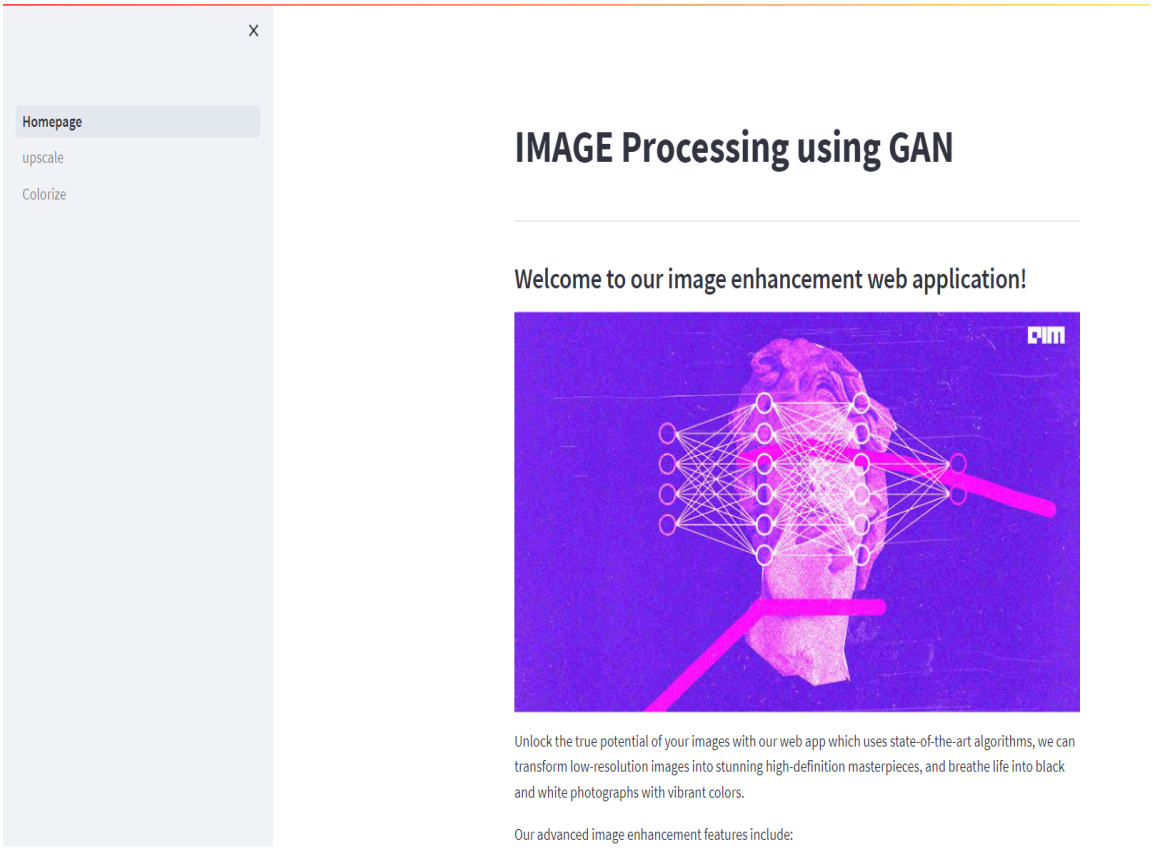
4.1 PSNR(Peak signal-to-noise ratio):

In [ ]:
```
import numpy as np
from numpy import asarray
from PIL import Image
from io import BytesIO
from basicsr.archs.rrdbnet_arch import RRDBNet
import cv2
import os
from realesrgan.archs.srvgg_arch import SRVGGNetCompact
from realesrgan import RealESRGANer
import math

import matplotlib.pyplot as plt



def psnr(img1, img2):
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))


def upscale(img,t):




        model_path='models/customESRGAN.pth'
        my_model = SRVGGNetCompact(num_in_ch=3, num_out_ch=3, num_feat=64, num_conv=16, upscale=4, act_type='prelu')

        netscale = 4



        upsampler = RealESRGANer(
        scale=netscale,
        model_path=model_path,
        dni_weight=None,
        model=my_model,
        tile=0,
        tile_pad=10,
        pre_pad=0,
        half=True,
        gpu_id=0)
```

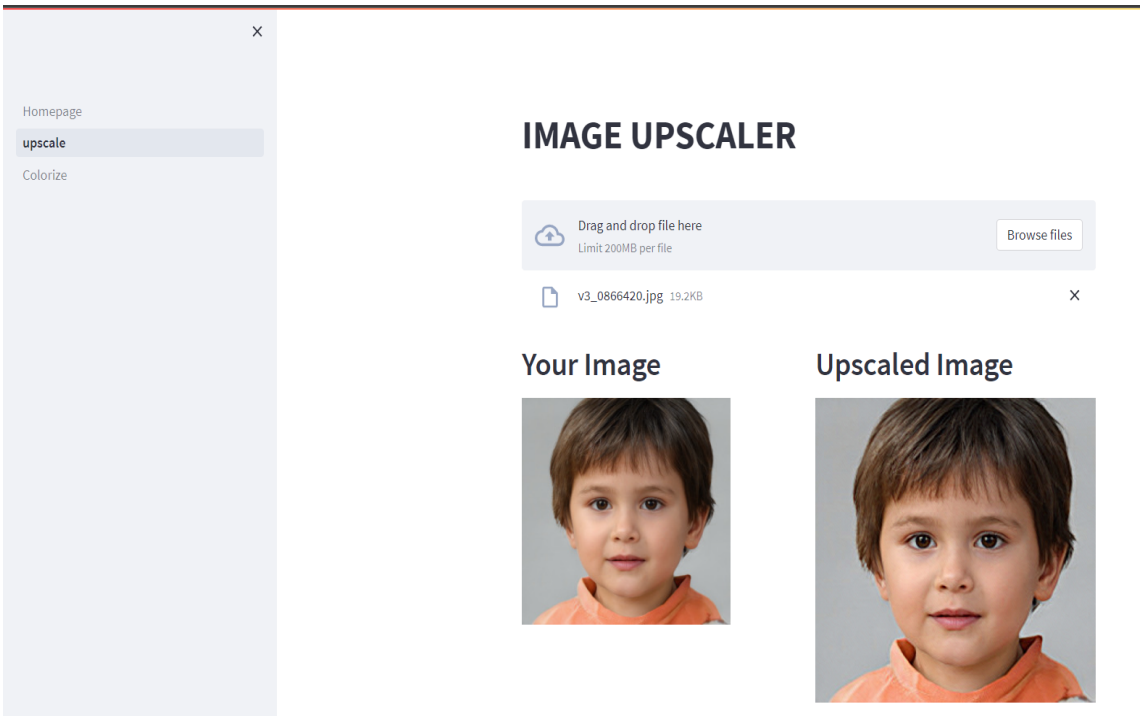
```
        tile_pad=10,
        pre_pad=0,
        half=True,
        gpu_id=0)


        if len(img.shape) == 3 and img.shape[2] == 4:
            img_mode = 'RGBA'
        else:
            img_mode = None

        try:
            output, _ = upsampler.enhance(img, outscale=4)
        except RuntimeError as error:
            print('Error', error)
        else:

            #blurred_img = cv2.GaussianBlur(output, (0, 0), 7)
            #output = cv2.addWeighted(output, 1.7, blurred_img, -0.7,0)
            #output=cv2.medianBlur(output, 3)

            return output

avgPSNR=0
c=0
hrpath=r'C:\Users\IamSh\Desktop\custom_model\images\val\hr'
lrpath=r'C:\Users\IamSh\Desktop\custom_model\images\val\lr'
for i,j in zip(os.listdir(hrpath),os.listdir(lrpath)):
    hr_path = os.path.join(hrpath,i)
    hr_image = cv2.imread(hr_path)

    lr_path = os.path.join(lrpath,i)
    lr_image = cv2.imread(lr_path)


    try:
        upscaled_img=upscale(lr_image)


    except Exception :
        continue

    psnrscore=psnr(hr_image,upscaled_img)
    #print(psnrscore)
    avgPSNR+=psnrscore
    c+=1

print(avgPSNR/c)
```

From execute above code we got avg PSNR value which is: **32.9**

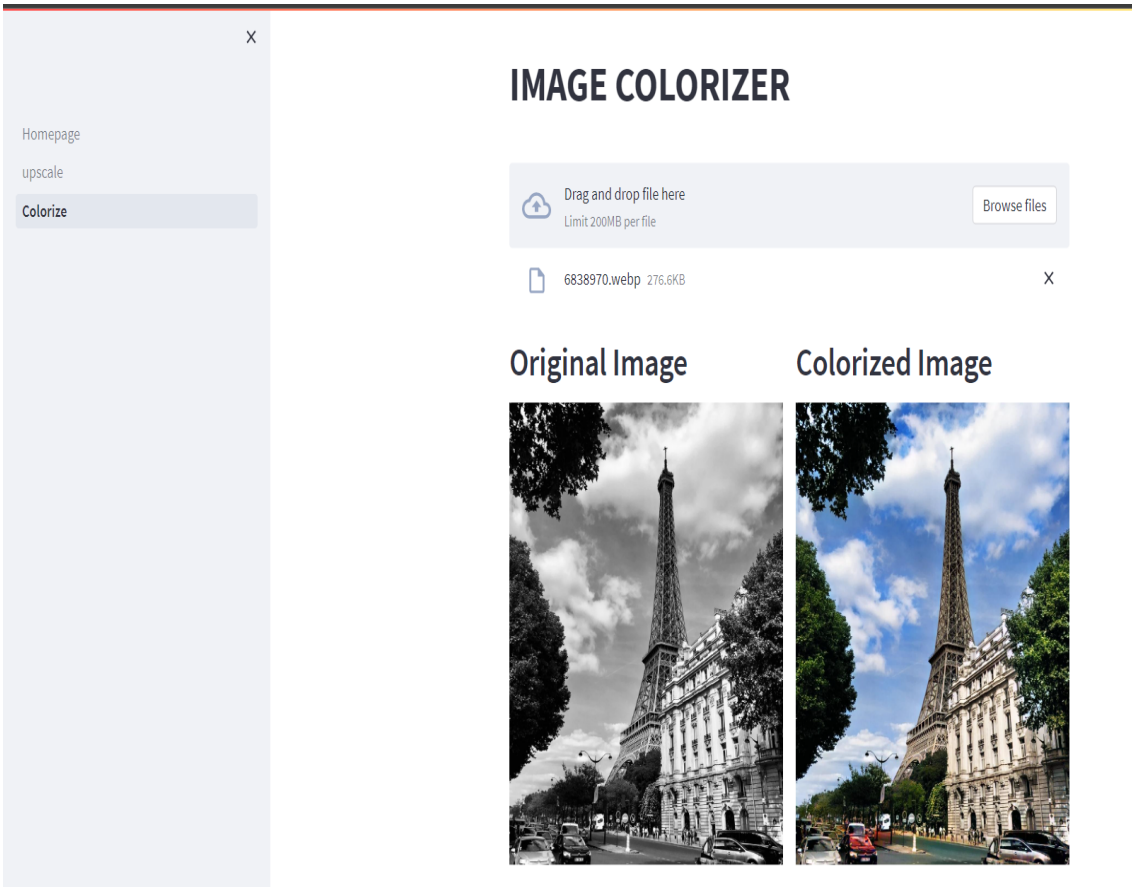
4.2 Homescreen:-



4.3 Upscaler:-



4.4 Colorizer:-



CHAPTER 5: CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

In conclusion, the project focused on two image enhancement techniques: ESRGAN for upscaling and DeOldify for colorization. ESRGAN, or Enhanced Super-Resolution Generative Adversarial Networks, was used to enhance the resolution and quality of low-resolution images. It achieved this by leveraging deep learning algorithms to generate high-resolution versions of the input images.

DeOldify, on the other hand, was employed for colorization. This technique aimed to add color to black-and-white or faded images, effectively bringing them to life. DeOldify utilized advanced algorithms and deep learning models to automatically predict and apply suitable colors to the grayscale images, resulting in vibrant and visually appealing colorizations.

Throughout the project, both ESRGAN and DeOldify demonstrated their effectiveness in enhancing and transforming images. The application of these techniques has the potential to greatly improve the visual quality and appeal of images, whether they are low-resolution or lacking in color. The project showcased the capabilities of these techniques and their potential for various applications in industries such as entertainment, digital preservation, and advertising.

It is important to note that the field of image enhancement and restoration is constantly evolving, and future advancements in deep learning and computational technologies may further enhance the performance and capabilities of ESRGAN and DeOldify. Continued research and development in this area can lead to even more impressive results and open up new possibilities for image enhancement in the future.

5.2 Limitations

ESRGAN (Enhanced Super-Resolution Generative Adversarial Networks) and DeOldify are both state-of-the-art deep learning models used for image enhancement tasks. While they have achieved impressive results, they do have certain limitations:

1. Training data limitations: ESRGAN and DeOldify require large amounts of high-quality training data to effectively learn image enhancement features. The availability of such data can be limited, especially for specific domains or rare scenarios. Insufficient or biased training data may result in suboptimal performance.

2. Generalization to unseen data: Although these models can produce remarkable enhancements for a wide range of images, they may struggle when presented with data significantly different from their training set. They might not accurately handle novel or unique image types, styles, or artifacts.

3. Over-smoothing or loss of details: In some cases, ESRGAN and DeOldify can over-smooth or soften the image, leading to a loss of fine details. This is a common challenge in image super-resolution tasks, as the models try to balance between removing noise and preserving image sharpness.

4. Artifacts and unrealistic textures: When pushed to extreme upsampling factors or when dealing with complex image structures, ESRGAN and DeOldify can sometimes generate artifacts or produce textures that appear unnatural. These artifacts can include checkerboard patterns, halo effects, or unrealistic color transitions.

5. Computational requirements: Both ESRGAN and DeOldify are computationally demanding models, requiring powerful hardware (e.g., GPUs) to achieve real-time or near-real-time performance. This limits their accessibility for users with limited computational resources.

5.3 Future scope

The future scope for upscaling resolution and colorization in image processing is promising and holds great potential for further advancements. As technology continues to evolve, we can expect improvements in algorithms and techniques that enhance the quality and realism of upscaling and colorization processes. The future may bring more sophisticated deep learning models that surpass the current state-of-the-art methods, enabling even higher-resolution upscaling and more accurate color representations. Additionally, the integration of domain-specific knowledge and context awareness can further enhance the preservation of historical and cultural aspects during colorization. Furthermore, the application of these techniques may extend to other media formats, such as videos, enabling the restoration and colorization of moving images. As research and development in the field of image processing continues, we can anticipate more efficient and user-friendly tools that empower users to upscale resolution and colorize images with ease, opening up new avenues for creative expression, digital restoration, and preserving our visual heritage.

1. **Image Synthesis and Augmentation:** GANs can be used to generate realistic and high-quality images, including photos, artwork, and textures. Future developments may focus on enhancing the realism, resolution, and diversity of generated images. GANs can also be employed for data augmentation, creating additional training samples to improve the performance of image classification and object detection models.

2. **Image Restoration and Super-Resolution:** GANs have been effective in restoring and enhancing low-quality or degraded images. Future research may explore more advanced techniques for image denoising, deblurring, inpainting, and super-resolution, enabling the recovery of finer details and enhancing image quality.

3. **Style Transfer and Image Editing:** GANs can be utilized for artistic style transfer, enabling the transformation of images into different artistic styles or mimicking the characteristics of specific artists. Future work may focus on refining the style transfer process and making it more controllable and customizable. GANs can also facilitate intuitive image editing by allowing users to manipulate specific attributes (e.g., change hair color, add or remove objects) while preserving overall image consistency.

4. **Medical Imaging and Healthcare:** GANs have demonstrated potential in medical imaging applications, such as generating synthetic medical images, synthesizing missing modalities, and enhancing image quality for diagnosis. Future advancements may involve refining GAN models to generate more accurate and realistic medical images, aiding in disease detection, treatment planning, and surgical simulations.

5. **Video and Motion Analysis:** GANs can be extended to process and manipulate video sequences and motion data. Future research may explore techniques for video generation, object tracking, video prediction, and action recognition using GANs. These advancements could have applications in video editing, computer animation, and surveillance systems.

REFERENCES

[1] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, Xiaoou Tang (2018, Sep), “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks” Computer vision and Recognition, Available: https://arxiv.org/pdf/1809.00219.pdf

[2] Chao Dong, Chen Change Loy, Kaiming He, Xiaoou Tang (2015, July), “Image Super-Resolution Using Deep Convolutional Networks” Computer vision and Pattern Recognition, Available: https://arxiv.org/abs/1501.00092

[3] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi (2017, May), “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network” Computer vision and Pattern Recognition, Available: https://arxiv.org/abs/1609.04802

[4] Christoph Vogler, Claas Abert, Florian Bruckner, Dieter Suess, Dirk Praetorius (2016, April), “Basic noise mechanisms of heat-assisted-magnetic recording” Materials science, Available: https://arxiv.org/abs/1605.00067

[5] Omar Abdulwahhab Othman ,Betül Uzbaş and Sait Ali Uymaz (2019, November), “Automatic Black & White Images colorization using Convolutional neural network” Academic Perspective Procedia, Available:https://www.researchgate.net/publication/337686473_Automatic_Black_White_Images_colorization_using_Convolutional_neural_network

[6] T. Sharmila, L.Megalan Leo (2016, April), “Image upscaling based convolutional neural network for better reconstruction quality” Computer vision and Pattern Recognition, Available:https://www.researchgate.net/publication/337686473_Automatic_Black_White_Images_colorization_using_Convolutional_neural_network

[7] Github. (2023, April) DeOldify Available: https://github.com/jantic/DeOldify

[8] Github. (2021, July) ESRGAN Available: https://github.com/xinntao/ESRGAN