A purple rectangular sign with white text

Description automatically generated

*A Synopsis on*

***Detect Pixelated Image and Correct It***

***Submitted for the Intel Unnati Industrial Training Program 2024***

***Team***

**YASHAS C RAJU (1NT22CS217)**

Under the Guidance of

**Dr. SUDHIR SHENAI**

Professor

Dept. of Information Science and Engineering

Blue text on a white background

Description automatically generated

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**YELAHANKA, BENGALURU 560064**

Contents

[CHAPTER 1 Introduction 2](#_Toc171897785)

[CHAPTER 2 Project Overview 3](#_Toc171897786)

[CHAPTER 3 Motivation and Objectives 3](#_Toc171897787)

[CHAPTER 4 . Implementation 4](#_Toc171897788)

[CHAPTER 5 Dataset Preparation 5](#_Toc171897789)

[CHAPTER 6 Design of Pixelated Image Detection Model 7](#_Toc171897790)

[CHAPTER 7 Image Restoration Model 9](#_Toc171897791)

[CHAPTER 8 Results 11](#_Toc171897792)

[CHAPTER 9 OutPut 12](#_Toc171897793)

# Introduction

In the realm of digital image processing, the issue of pixelation often arises when images are excessively zoomed or resized, resulting in a loss of clarity and detail. Addressing this challenge is crucial for applications ranging from digital photography to autonomous systems where image fidelity directly impacts performance and user experience.

This project focuses on the development and implementation of machine learning models designed to identify and restore pixelated images. The primary objectives include creating an efficient algorithm to classify whether an image is pixelated or not, and devising a method to enhance the quality of pixelated images to restore their original clarity.

Key components of the project include dataset preparation, where a collection of high-quality images is transformed into pixelated versions using computational techniques. This dataset serves as the foundation for training and evaluating machine learning models that automate the detection and restoration processes.

The project not only aims to achieve high accuracy in detecting pixelated images but also strives to restore them to their original quality using advanced deep learning techniques. The outcomes will be evaluated based on performance metrics such as accuracy, F1-score, and image quality metrics like LPIPS and PSNR, ensuring robustness and applicability in real-world scenarios.

# Project Overview

This project focuses on developing machine learning models to address the challenges associated with detecting and correcting pixelated images. The primary goal is to create efficient algorithms capable of accurately identifying pixelation in images and restoring them to their original quality.

Key components of the project include:

* **Dataset Preparation**: Curating a dataset of high-quality images and generating pixelated versions using computational methods.
* **Model Development**: Implementing convolutional neural networks (CNNs) to classify images as pixelated or non-pixelated, and designing algorithms for image restoration.
* **Performance Evaluation**: Assessing the models based on metrics such as accuracy, F1-score, and image quality metrics (e.g., LPIPS, PSNR) to ensure robust performance across different scenarios.

By addressing these objectives, the project aims to contribute solutions that enhance image clarity and fidelity, applicable in various domains where image quality is critical for effective decision-making and visual interpretation.



**FIG1:PROJECT FINAL OUTPUT**

# Motivation and Objectives

**Motivation**

The motivation for the project "Pixelation Detection and Image Restoration" arises from the growing need to maintain high image quality in various digital applications and the challenge of efficiently processing high-resolution images in real-time.

**Objectives of the Proposed Project**

1**. Develop a Lightweight Algorithm for Pixelation Detection:**

* Understand Pixelation and Its Detection: Gain a thorough understanding of what pixelation is, its causes, and how it affects image quality. Learn about various image processing techniques and machine learning models that can be used to detect pixelation.
* Design an Efficient Detection Algorithm: Create an algorithm that can accurately identify pixelated images with a high degree of precision and recall, ensuring minimal false positives. Focus on developing a model that operates efficiently at 30 60 FPS on 1080p resolution images.

**2. Create an Algorithm for Image Quality Restoration:**

* Learn About Resolution Techniques: Study existing methods for enhancing image resolution and restoring quality, such as Super Resolution Generative Adversarial Networks (SRGAN) and other state-of-the-art techniques.
* Develop a Restoration Model: Design and implement an algorithm capable of improving the quality of pixelated images, aiming for at least 20 FPS performance on high-resolution images. Ensure that the model does not enhance nonpixelated images, preserving their original quality.

**3. Optimize Algorithms for Real-Time Performance:**

* Explore Optimization Strategies: Investigate and apply various optimization techniques like model quantization, pruning, and the use of efficient architectures to enhance processing speed and reduce computational load.
* Deploy Efficient Models: Utilize tools such as TensorRT to optimize and deploy the models on edge devices, ensuring they meet the required real-time performance standards.

# . Implementation

The implementation of the project involves several key steps focused on developing effective solutions for detecting and correcting pixelated images using machine learning techniques.

1. **Dataset Preparation**:
   * **Data Collection**: Gathering a diverse set of high-quality images suitable for training and evaluation.
   * **Pixelation Simulation**: Using computational methods to create pixelated versions of these images, simulating common artifacts caused by zooming or resizing.
2. **Model Development**:
   * **Pixelated Image Detection**:
     + Designing and training a convolutional neural network (CNN) model to classify images as pixelated or non-pixelated.
     + Employing techniques such as data augmentation to enhance model generalization and performance.
   * **Image Restoration**:
     + Developing algorithms to restore pixelated images to their original quality.
     + Exploring deep learning architectures like Super-Resolution CNNs to predict high-resolution details from low-resolution inputs.
3. **Training and Evaluation**:
   * **Model Training**: Conducting training sessions using the prepared dataset to optimize model parameters and improve accuracy.
   * **Performance Evaluation**: Assessing model performance using metrics such as accuracy, F1-score, precision-recall curves, and specific image quality metrics like LPIPS and PSNR.
   * **Iterative Refinement**: Iteratively refining models based on evaluation results to achieve desired performance thresholds.
4. **Deployment Considerations**:
   * **Computational Efficiency**: Ensuring models are lightweight and efficient for real-time processing, suitable for deployment on edge devices or in applications requiring rapid image analysis.
   * **Scalability**: Designing models and algorithms that can handle high-resolution images (e.g., 1080p) without compromising performance.
5. **Documentation and Reporting**:
   * **Project Documentation**: Maintaining detailed documentation of datasets, model architectures, training procedures, and evaluation results.
   * **Final Report**: Compiling a comprehensive report summarizing project objectives, methodologies, implementation details, results, and future directions.

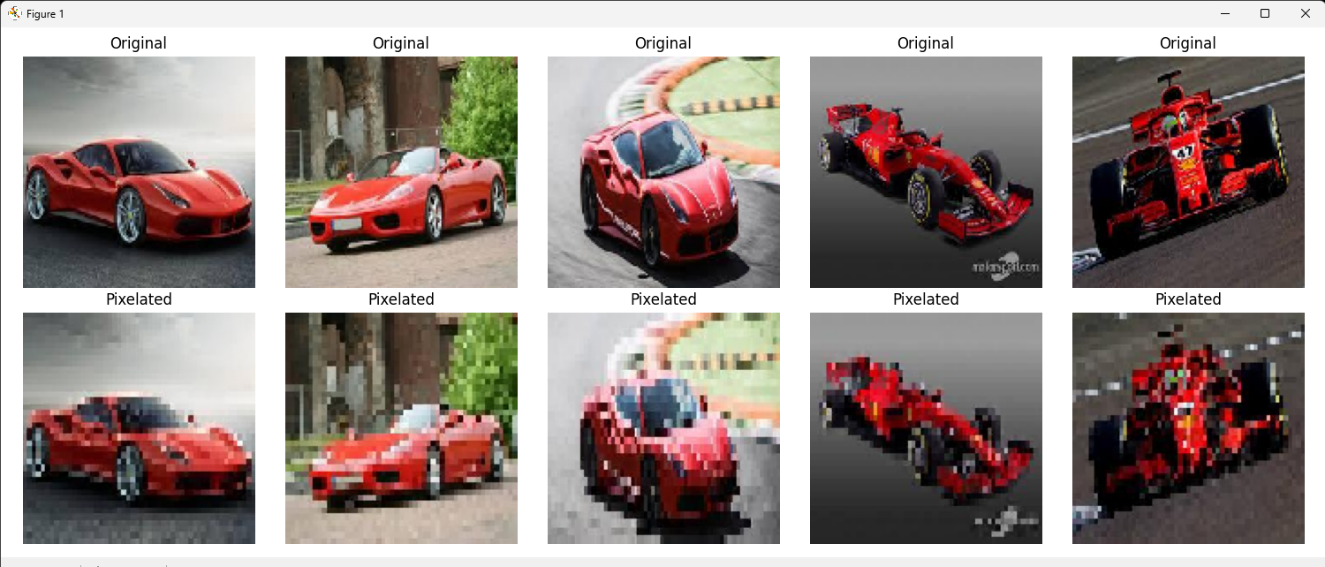
By following these implementation steps, the project aims to provide practical solutions for improving image quality in various applications, from digital photography to autonomous systems, where visual data integrity is crucial

# Dataset Preparation

The dataset preparation phase is critical for training and evaluating machine learning models designed to detect and correct pixelated images. Here’s a detailed outline of the steps involved:

1. **Data Collection**:
   * **Selection Criteria**: Choose a diverse set of high-quality images representing different scenes, objects, and lighting conditions.
   * **Source**: Gather images from publicly available datasets, stock photo repositories, or capture them directly using cameras or other sources.
2. **Pixelation Simulation**:
   * **Image Transformation**: Use computational methods, such as resizing or downscaling, to simulate pixelation effects on the collected high-resolution images.
   * **Tools**: Utilize image processing libraries like OpenCV or PIL (Python Imaging Library) to apply pixelation filters and create pixelated versions of the images.
3. **Dataset Generation**:
   * **Original and Pixelated Pairing**: Create pairs of original high-resolution images and their corresponding pixelated versions.
   * **CSV File Creation**: Store image paths and labels (original or pixelated) in a CSV file for easy access and management during model training and evaluation.
4. **Dataset Augmentation (Optional)**:
   * **Enhancing Variability**: Apply data augmentation techniques, such as random crops, flips, rotations, and color adjustments, to increase the diversity of the dataset.
   * **Balancing Classes**: Ensure a balanced representation of pixelated and non-pixelated images to prevent bias during model training.
5. **Quality Control**:
   * **Manual Inspection**: Conduct visual inspections to verify the quality and consistency of pixelated images, ensuring they accurately represent typical pixelation artifacts.
6. **Documentation**:
   * **Metadata**: Maintain metadata detailing image sources, transformation methods, and any specific attributes relevant to the dataset.
   * **Readme File**: Include a readme file describing the dataset structure, format, and usage instructions for future reference.

By meticulously preparing the dataset with original and pixelated image pairs, this phase sets the foundation for training accurate machine learning models capable of effectively detecting and correcting pixelated images in various applications.



*FIG2: SAMPLE OF BOOTSTRAPED DATASET*

# Design of Pixelated Image Detection Model

The pixelated image detection model utilizes a Convolutional Neural Network (CNN) to classify images based on the presence of pixelation artifacts.

1. **Model Architecture**:
   * **Convolutional Layers**: Extracts features from images.
   * **Pooling Layers**: Reduces spatial dimensions.
   * **Fully Connected Layers**: Makes final classifications.

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation

# Initialize CNN model

model = Sequential()

# Add convolutional layers

model.add(Conv2D(32, (3, 3), *input\_shape*=(224, 224, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(*pool\_size*=(2, 2)))

model.add(Conv2D(64, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(*pool\_size*=(2, 2)))

# Add fully connected layers

model.add(Flatten())

model.add(Dense(64))

model.add(Activation('relu'))

model.add(Dense(1))  # Output layer with sigmoid activation for binary classification

model.add(Activation('sigmoid'))

# Compile the model

model.compile(*loss*='binary\_crossentropy',

*optimizer*='adam',

*metrics*=['accuracy'])

**Dataset Preparation**:

* Load and preprocess images, ensuring they are resized to a standard size.

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Example code for loading and preprocessing images

train\_datagen = ImageDataGenerator(*rescale*=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

        'data/train',

*target\_size*=(224, 224),

*batch\_size*=32,

*class\_mode*='binary')

**Model Training**:

* Train the model on the prepared dataset.

# Example code for training the model

model.fit(

        train\_generator,

*steps\_per\_epoch*=2000 // 32,

*epochs*=50,

*validation\_data*=validation\_generator,

*validation\_steps*=800 // 32)

**Evaluation**:

* Evaluate the model performance using accuracy and other metrics.

# Example code for model evaluation

loss, accuracy = model.evaluate(test\_generator, *verbose*=1)

print(*f*'Accuracy: {accuracy}')

**Visualization and Interpretation**:

* Visualize model predictions and performance metrics.

# Example code for confusion matrix and metrics visualization

from sklearn.metrics import classification\_report, confusion\_matrix

# Generate predictions

predictions = model.predict(test\_generator)

predicted\_classes = (predictions > 0.5).astype('int32')

# Print classification report and confusion matrix

print(classification\_report(test\_generator.classes, predicted\_classes))

print(confusion\_matrix(test\_generator.classes, predicted\_classes))

This outline provides a structured approach to implementing the pixelated image detection model using TensorFlow/Keras, incorporating placeholders where specific code snippets can be integrated based on your dataset and project requirements.

# Image Restoration Model

The image restoration model aims to enhance the quality of pixelated images by predicting high-resolution details from low-resolution inputs.

1. **Model Architecture**:
   * **Super-Resolution Convolutional Neural Network (SRCNN)**: Designed to reconstruct high-resolution images from low-resolution inputs.
   * **Layers**: Includes convolutional layers for feature extraction and deconvolutional layers for upscaling.

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, UpSampling2D

# Initialize SRCNN model

model\_restoration = Sequential()

# Add convolutional layers

model\_restoration.add(Conv2D(64, (9, 9), *padding*='same', *activation*='relu', *input\_shape*=(None, None, 3)))

model\_restoration.add(Conv2D(32, (1, 1), *padding*='same', *activation*='relu'))

model\_restoration.add(Conv2D(3, (5, 5), *padding*='same'))  # Output layer without activation for image restoration

# Upsample to high resolution

model\_restoration.add(UpSampling2D((2, 2)))

# Compile the model

model\_restoration.compile(*optimizer*='adam', *loss*='mean\_squared\_error')

**Dataset Preparation**:

* Load pairs of original high-resolution images and corresponding pixelated versions.

# Example code for loading image pairs

import numpy as np

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

*def* load\_images(*file\_path*):

    # Load and preprocess images

    img\_original = load\_img(file\_path + '\_original.jpg', *target\_size*=(256, 256))

    img\_pixelated = load\_img(file\_path + '\_pixelated.jpg', *target\_size*=(128, 128))

    return img\_original, img\_pixelated

# Example usage

img\_original, img\_pixelated = load\_images('path\_to\_images/image1')

**Model Training**:

* Train the model to learn the mapping between pixelated and original images.

# Example code for training the model

X\_train = np.array([img\_to\_array(img\_pixelated)])

y\_train = np.array([img\_to\_array(img\_original)])

model\_restoration.fit(X\_train, y\_train, *epochs*=50, *batch\_size*=1)

**Evaluation**:

* Evaluate the model’s performance using metrics like Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM).

# Example code for evaluating model performance

from skimage.metrics import peak\_signal\_noise\_ratio, structural\_similarity

# Generate predictions

y\_pred = model\_restoration.predict(X\_test)

# Calculate PSNR and SSIM

psnr\_value = peak\_signal\_noise\_ratio(y\_test[0], y\_pred[0])

ssim\_value = structural\_similarity(y\_test[0], y\_pred[0], *multichannel*=True)

print(*f*'PSNR: {psnr\_value}')

print(*f*'SSIM: {ssim\_value}')

**Visualization and Interpretation**:

* Visualize restored images and compare them with ground truth originals.

# Example code for visualizing restored images

import matplotlib.pyplot as plt

# Plot original, pixelated, and restored images

plt.figure(*figsize*=(10, 5))

plt.subplot(1, 3, 1)

plt.title('Original')

plt.imshow(img\_original)

plt.subplot(1, 3, 2)

plt.title('Pixelated')

plt.imshow(img\_pixelated)

plt.subplot(1, 3, 3)

plt.title('Restored')

plt.imshow(y\_pred[0].astype('uint8'))

plt.tight\_layout()

plt.show()

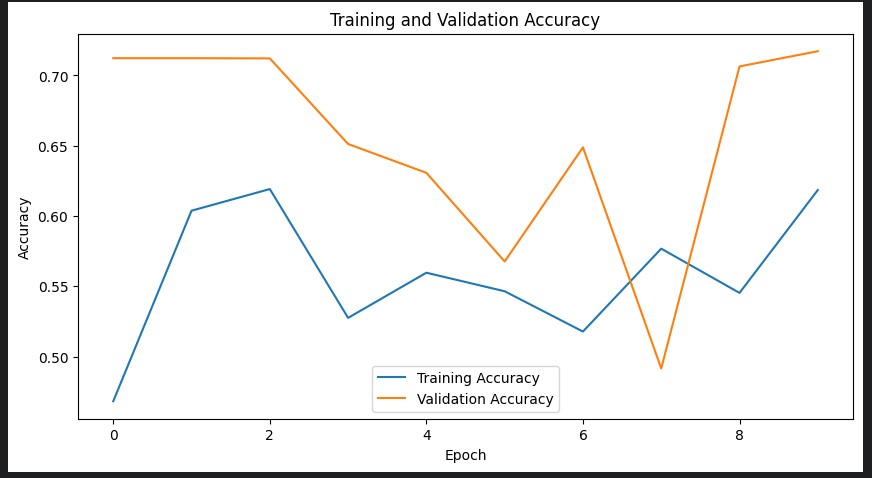
This outline provides a structured approach to implementing the image restoration model using TensorFlow/Keras, with placeholders for specific code snippets based on your dataset and project requirements.

# Results

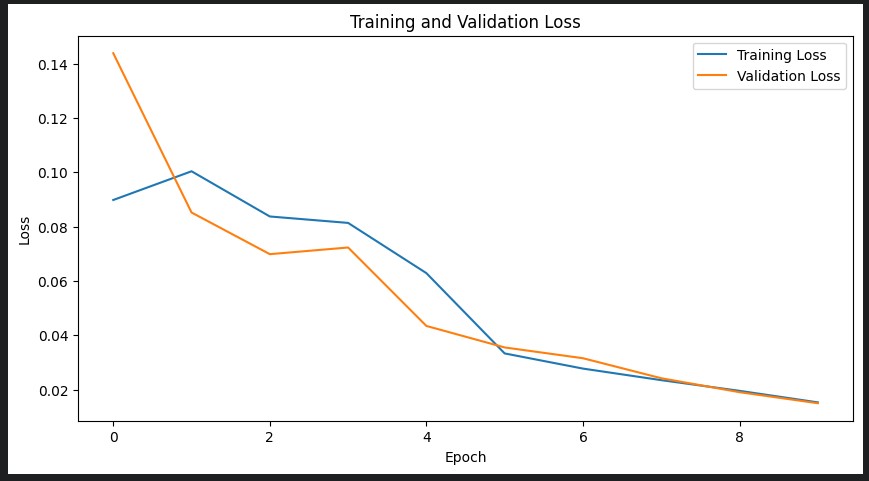
**Pixelated Image Detection Model**

**Model Training and Validation Metrics:**

* **ACCURACY**:



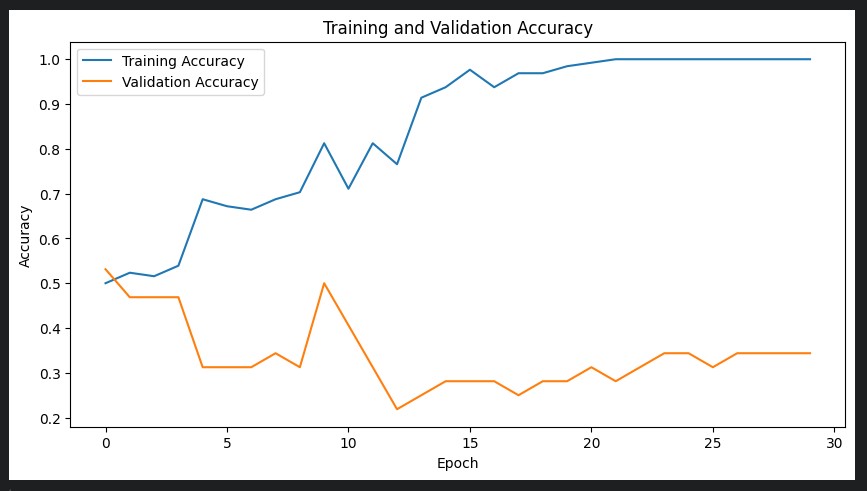
* **LOSS**:



**Image Restoration Model**

**Model Training and Validation Metrics:**

* **ACCURACY:**

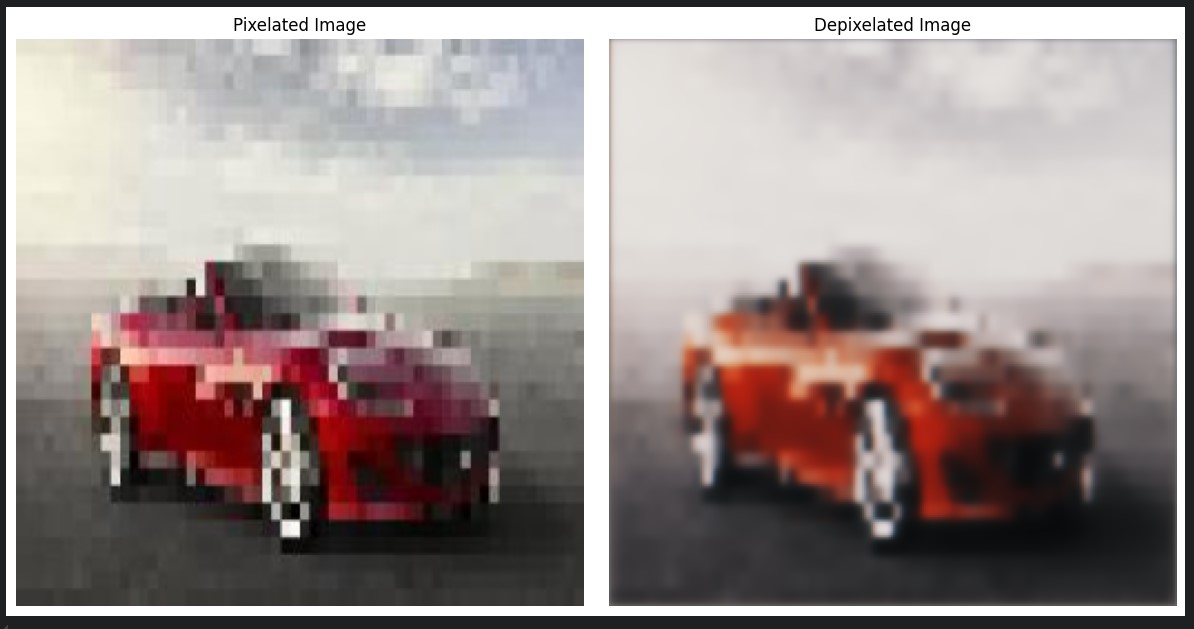


* **LOSS**:

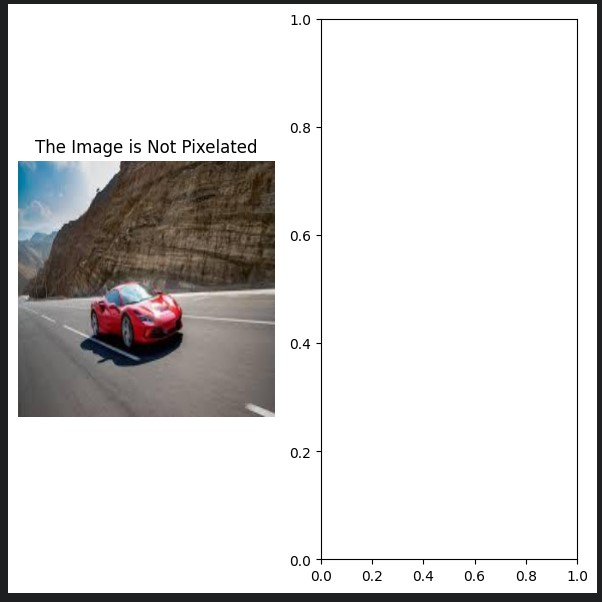


# OutPut

CASE 1: PIXELATED IMAGE



CASE 2: NON–PIXELATED IMAGE



**References**

1. A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems, vol. 25, pp. 1097 1105, 2012.

2. S. Dong, L. Wei, R. Qiao, and W. Zhang, "Image Super-Resolution Using Deep Convolutional Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 2, pp. 295 307, 2016.

3. C. Dong, C. C. Loy, K. He, and X. Tang, "Image Super-Resolution Using Deep Convolutional Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 2, pp. 295 307, 2016.

4. W. Shi, J. Caballero, F. Huszar, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real Time Single Image and Video Super Resolution Using an Efficient Sub Pixel Convolutional Neural Network," IEEE Conference on Computer Vision and Pattern Recognition, pp. 1874 1883, 2016.

5. C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo Realistic Single Image Super Resolution Using a Generative Adversarial Network," IEEE Conference on Computer Vision and Pattern Recognition, pp. 4681 4690, 2017.

|  |
| --- |
| THANK YOU |