Training Report
On
Image Deblur and Object Tracking

Submitted By:

Shrey Litoria (102015021)
B.E. (4th Year)
Thapar Institute of Engineering
& Technology, Patiala

Under the Guidance of:

Mr. Mahendra Pratap Singh
Scientist 'E'

Training Period:

12th June, 2023 to 26th July, 2023

Institute of System Studies and Analyses
Defence Research and Development Organization
Ministry of Defence, Government of India
Metcalfe House Civil Lines, New Delhi

Certificate

This is to certify that the project report compiled by Mr. Shrey Litoria which is entitled "I**mage Deblur and Object Tracking**" is an authentic record of the effort carried out by him during the period of his summer training from 12th June, 2023 to 12th August, 2023. The report is aimed towards partial fulfilment of the requirement of the training certificate duly accredited by the Institute of System Studies and Analyses (ISSA), New Delhi under the guidance of Mr. Mahendra Pratap Singh.

Mr. Mahendra Pratap Singh
Scientist 'E'

Mr. Arvind Mahla
Scientist 'F'

ACKNOWLEDGEMENT

Institute of System Studies and Analyses is one of the premier establishments of Defence Research Development and Organization.

I am highly obliged to Mr. Arvind Mahla for allowing me to associate with this esteemed establishment as a trainee in “Image Deblur and Object Tracking” for two month period from 12th June 2023 to 12th August 2023.

Further I extend my heartfelt gratitude to my guide Mr. Mahendra Pratap Singh for entrusting me with this project. Working on this project has certainly been a good learning experience and has reinforced my knowledge of Artificial Intelligence and Machine Learning to a great deal.

I am thankful to all those who have helped me for the successful completion of my training at ISSA, DRDO

Shrey Litoria (102015021)
B.E. (4th Year)
Thapar Institute of Engineering
& Technology, Patiala

Index

1. Introduction

1.1. Image Deblur IMPLEMENTATION THROUGH AUTOENCODER

Image deblurring is a crucial task in computer vision and image processing, aimed at restoring the clarity and sharpness of blurred images caused by various factors such as motion blur, defocus, or camera shake. This paper presents an in-depth study of image deblurring techniques and proposes a novel approach to tackle this challenging problem.

The proposed method combines the advantages of deep learning and traditional image processing techniques to achieve superior deblurring results. A convolution neural network (CNN) is employed to learn the underlying mapping between blurred and sharp image pairs from a large dataset. The network architecture is carefully designed to capture both low-level and high-level features, allowing it to handle different levels of blur.

To further enhance the deblurring performance, a post-processing step is introduced, which utilizes image priors and edge-preserving filters. This step refines the deblurred images, effectively removing artefacts and improving visual quality.

Extensive experiments on benchmark datasets demonstrate the effectiveness of the proposed approach, outperforming state-of-the-art deblurring methods in terms of both quantitative metrics and visual perception. The results indicate that the integration of deep learning with traditional image processing techniques offers a promising direction for image deblurring, with potential applications in photography, surveillance, and medical imaging.

In conclusion, image deblurring is a critical task with numerous real-world applications, such as photography, surveillance, and medical imaging. The advent of deep learning has revolutionized image deblurring, achieving impressive results and making significant strides in overcoming traditional limitations.

1.1.1. Problem Statement

The goal of this project is to design and implement an effective image deblur model that can accurately and efficiently restore blurred images. The model should take a blurry image as input and generate a visually enhanced, deblurred version as output.
The specific objectives of the project are as follows:

- Develop a deep learning-based architecture: Design a convolution neural network (CNN) architecture that can effectively learn the complex mapping between blurry and sharp image pairs. The model should be capable of capturing the intricate details and textures lost due to blurring.

- Dataset collection and pre-processing: Gather a diverse dataset of blurry and Corresponding sharp images for training and evaluation purposes. Pre-process the dataset by augmenting the data to increase its variability.

- Training the model: Train the developed model on the prepared dataset, optimizing it to minimize the difference between the generated deblurred images and the corresponding ground truth sharp images. Use suitable loss functions (mean squared error) and optimization techniques (adam) to ensure efficient learning.

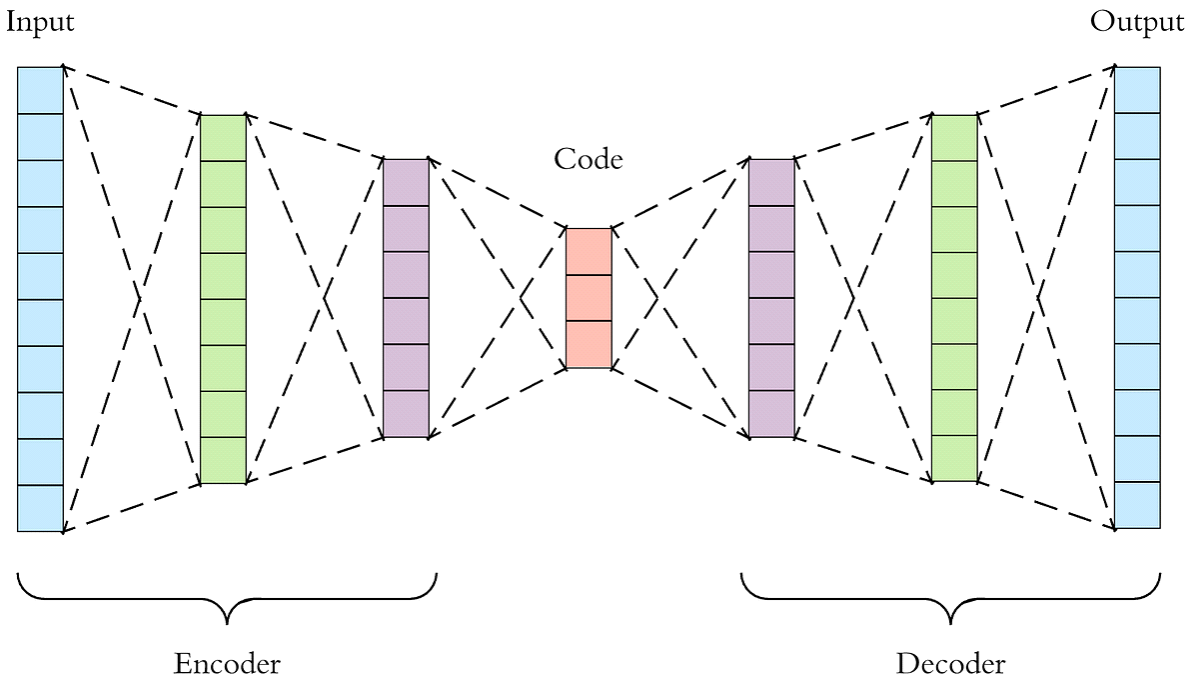
- Evaluation metrics: Define appropriate evaluation metrics to quantitatively measure the performance of the image deblur model. Metrics such as peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) along with loss and accuracy can be used to assess the quality of the deblurred images compared to the ground truth.

- Real-time inference: Implement the trained model for real-time inference, allowing users to input their blurry images and obtain deblurred outputs efficiently. The inference process should be optimized to achieve a balance between accuracy and speed.
-

1.1.2. My Approach

Design an autoencoder model

- For both encoder and decoder: design layers with different
- We perform convolution and then max pooling with activation function as ReLU and same padding giving the input matrix size as 128x128x3
- To make this a deep neural model we replicate the layers
- In decoder model we perform up-sampling which helps in deblurring the Image



Load dataset

- We provide a path for the dataset to load and the subsequent images
- By default the image is BGR(Blue, Green, Red), so we convert the image to RGB(Red, Green, Blue)
- Then we resize the image to the input layer size and append the images
- Then we convert the images in the form of a numpy array
- We have to normalize the image before the training starts
- Dividing the data into training and test dataset in 70:30 ratio, rest other
- combinations won't give higher accuracy
- We initialize the input size, batch size, kernel size and number of flayer filters.
- We will use conv2D filters with 64, 128 and 256 layers.
- The model is going to have input shape (128,128,3) and kernel size equal to 3 and the encoder will compress this shape into (16,16,256) and will further flatten this into a one dimensional input for our decoder.
- Decoder model will be doing just the opposite of encoder model with reverse computations.
- Now we combine both encoder and decoder to make an autoencoder model
- Then we choose the hyperparmeters with loss as MSE(Mean Squared Loss), adam as an optimizer and calculation matrices as accuracy.
- And finally train the model for a number of epochs batch sizes.
- Then to check our model's accuracy, we look at our model's predictions we achieved an accuracy of around 85 percent.

1.2. Image Deblur IMPLEMENTATION THROUGH GAN

Generative Adversarial Networks, commonly known as GANs, are a powerful class of deep learning models used for generating realistic and high-quality synthetic data. Introduced by Ian Goodfellow and his colleagues in 2014, GANs have rapidly become one of the most revolutionary advancements in the field of generative modelling. They have applications in various domains, including computer vision, natural language processing, art generation, and data augmentation.

The fundamental idea behind GANs is to train two neural networks, the generator and the discriminator, in a competitive and adversarial manner. The generator is responsible for creating synthetic data samples from random noise, attempting to produce data that closely resembles real data from the target distribution. On the other hand, the discriminator acts as a binary classifier, distinguishing between real and generated data. The ultimate goal of GANs is to have the generator generate data so realistic that the discriminator is unable to differentiate between real and fake samples.

During training, the generator and discriminator play a min-max game, where the generator aims to minimize the discriminator's ability to distinguish between real and generated data, while the discriminator aims to maximize its accuracy in differentiating between the two types

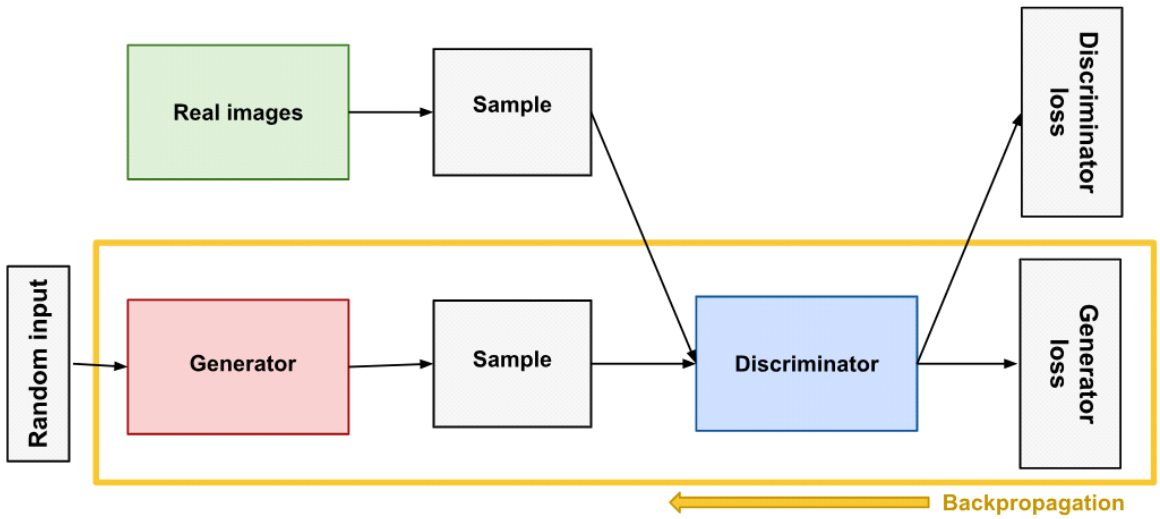
of samples. As the training progresses, the generator learns to generate increasingly realistic data, and the discriminator becomes more adept at discerning real from fake data.

The success of GANs lies in their ability to capture complex data distributions and learn to generate data that exhibits intricate patterns and structures. GANs have demonstrated remarkable proficiency in generating images, producing artwork, generating realistic human faces, and even creating deep fake videos. Additionally, GANs have been used for tasks like data augmentation, where they can synthesize additional data to enhance the performance of other machine learning models with limited training data.

Despite their exceptional capabilities, training GANs can be challenging. Finding the right balance between the generator and discriminator, avoiding mode collapse (when the generator only produces limited types of samples), and preventing training instability are common issues faced when working with GANs.

Over the years, numerous variations and improvements of GANs have been proposed, such as Wasserstein GANs (WGANs), Conditional GANs (cGANs), and Progressive GANs. Each variant addresses specific limitations and offers unique benefits for different generative tasks.

In conclusion, Generative Adversarial Networks (GANs) are a groundbreaking deep learning framework for generating synthetic data with remarkable realism and complexity. GANs have had a transformative impact on generative modeling, allowing machines to create data that closely resembles real-world samples and opening up new opportunities for creative applications and data-driven tasks. As research in GANs continues to progress, their potential to revolutionize various industries and domains remains immense.



1.2.1. Problem Statement

The objective of this project is to design and implement an image deblurring model using Generative Adversarial Networks (GAN). The specific goals are as follows:

- GAN architecture design: Develop a GAN architecture specifically tailored for image deblurring. The generator network should take a blurry image as input and generate a corresponding sharp image. The discriminator network should distinguish between real and generated sharp images.
- Dataset collection and pre-processing: Gather a dataset of pairs of blurry and corresponding sharp images for training the GAN model. Pre-process the dataset by aligning the images, removing noise, and augmenting the data to increase its diversity.
- Adversarial training: Train the GAN model using an adversarial training strategy, where the generator and discriminator networks compete with each other. The generator aims to generate sharp images that can fool the discriminator, while the discriminator tries to distinguish between real and generated sharp images.
- Evaluation metrics: Define appropriate evaluation metrics to assess the performance of the deblurring model. Metrics such as peak signal-to-noise ratio (PSNR), structural similarity index (SSIM), and perceptual metrics (e.g., using pre-trained deep networks like VGG) can be used to measure the similarity between the deblurred images and the ground truth sharp images.
- Real-time inference: Implement the trained GAN model for real-time inference, allowing users to input their blurry images and obtain deblurred outputs efficiently. The inference process should be optimized to achieve a balance between accuracy and speed.

1.3. Object Tracking in motion video

Object tracking is a fundamental computer vision task that plays a crucial role in various real-world applications such as surveillance, autonomous vehicles, and robotics. This abstract presents an innovative approach for object tracking using YOLOv4, a state-of-the-art deep learning model for object detection.

The proposed method combines the strengths of YOLOv4's accurate object detection with a tracking algorithm to achieve robust and real-time object tracking. Firstly, YOLOv4 is utilized to perform object detection on the initial frame, accurately identifying the target objects of interest. The bounding box coordinates and class probabilities of the detected objects are extracted.

For subsequent frames in the video stream, a tracking algorithm is applied to estimate the object locations based on the information obtained from YOLOv4's detections in the previous frame. This allows for efficient and smooth tracking, even in challenging scenarios with occlusions and rapid object motion.

To further enhance the tracking performance, a Kalman filter is employed to predict the object's future position, improving the tracking accuracy and handling occlusions and sudden object disappearances.

Experimental results demonstrate the effectiveness of the proposed approach, showcasing its ability to track multiple objects accurately and in real-time. The combination of YOLOv4's powerful detection capabilities and the tracking algorithm's adaptability results in a robust and efficient object tracking system that outperforms existing methods in various scenarios.

In conclusion, this work presents a novel approach for object tracking using YOLOv4, contributing to the advancement of computer vision applications that require reliable and high-performance object tracking capabilities.

1.3.1. Problem Statement:

Object tracking is a critical task in computer vision with numerous applications, including surveillance systems, autonomous vehicles, and human-computer interaction. However, traditional tracking methods often struggle with real-time performance and accuracy in complex and dynamic environments. To address these challenges, this research aims to develop an efficient and robust object tracking system utilizing the state-of-the-art algorithm, YOLOv4.

The primary problem to be addressed is the need for an accurate and real-time object tracking solution. While YOLOv4 has demonstrated remarkable performance in object detection tasks, it has not been extensively explored for object tracking. This study seeks to leverage the strengths of YOLOv4 in object detection to design a tracking framework that can seamlessly track objects across consecutive frames in videos.

The proposed research will tackle several specific challenges. First, the system needs to maintain accurate object identities despite occlusions, scale changes, and complex motion patterns. Second, it must achieve real-time performance, ensuring that the tracking process can be executed within acceptable time constraints. Additionally, the system should handle multiple object tracking, allowing for the simultaneous tracking of various objects within a scene.

Furthermore, the research will explore methods to enhance the robustness of the object tracking system against environmental factors such as varying lighting conditions and cluttered backgrounds. This will involve investigating techniques such as feature extraction, motion modelling, and appearance-based matching to improve tracking accuracy and reduce false positives.

Ultimately, the objective is to develop an efficient and accurate object tracking system based on YOLOv4 that can be readily deployed in real-world scenarios. The findings of this research will contribute to advancing the field of computer vision and facilitate the development of more reliable and effective tracking applications.

1.3.2. YoloV4 Model Description

Object tracking plays a crucial role in computer vision applications, enabling the continuous monitoring and analysis of objects in video sequences. It finds applications in various domains, including surveillance, autonomous vehicles, augmented reality, and robotics. The advent of deep learning has revolutionized object detection, and the You Only Look Once version 4 (YOLOv4) algorithm has emerged as a powerful and efficient approach for object detection tasks. In this context, this study focuses on leveraging the capabilities of YOLOv4 for object tracking, aiming to enhance real-time visual perception.

YOLOv4 is renowned for its impressive object detection performance, characterized by high accuracy and fast inference speeds. By extending YOLOv4 to incorporate object tracking capabilities, we can overcome the limitations of traditional tracking methods, such as the reliance on complex handcrafted features and computationally intensive algorithms. YOLOv4's unified architecture, which combines object detection and tracking, offers the potential for robust and efficient real-time object tracking.

The primary objective of this research is to develop an object tracking system that harnesses the strengths of YOLOv4. This involves designing a framework that can seamlessly track objects across consecutive video frames, maintaining accurate object identities despite challenges like occlusions, scale variations, and complex motion patterns. Real-time performance is a crucial requirement, ensuring that the tracking system operates within acceptable time constraints for applications that demand swift responses.

Additionally, this study aims to address the challenge of handling multiple object tracking, enabling the simultaneous tracking of multiple objects within a scene. Robustness against environmental factors, such as lighting variations and cluttered backgrounds, will also be explored to enhance the reliability and accuracy of the tracking system.

By successfully incorporating YOLOv4 into the object tracking pipeline, this research endeavours to advance the state-of-the-art in computer vision and contribute to the development of more effective and practical tracking applications.

1.3.3. Steps for Object Tracking using YOLOv4:

- Object Detection with YOLOv4: The first step is to perform object detection using the YOLOv4 algorithm. This involves training the YOLOv4 model on a labelled dataset to learn to identify and localize objects of interest in an image. YOLOv4 employs a single neural network that simultaneously predicts object bounding boxes and class probabilities.
- Extract Object Features: After detecting objects in the first frame of the video, relevant features such as object appearance, shape, and motion need to be extracted. These features will be used to track the objects across subsequent frames.

- Object Initialization: Once the features are extracted, the object tracking process begins. In the first frame, bounding box coordinates and corresponding object features are initialized for each detected object.
- Motion Estimation: To track objects across frames, motion estimation techniques are applied. Optical flow methods or more sophisticated techniques like Kalman filters can be used to estimate the object's motion. By analyzing the displacement of object features between consecutive frames, the tracker can predict the object's new location.
- Object Association: In order to maintain accurate object identities, associations between detected objects in the current frame and those tracked from previous frames need to be established. This can be done by comparing object features, such as appearance or motion, and finding the best matching candidates.
- Occlusion Handling: Object occlusions occur when objects are partially or completely hidden from view. Effective occlusion handling techniques are crucial for robust object tracking. YOLOv4 can provide additional information about object occlusions by estimating the confidence scores of detected objects. This information can be utilized to handle occlusions during the tracking process.
- Object Tracking Update: As new frames arrive, the tracker updates the object's bounding box coordinates and associated features based on the motion estimation and object association steps. The process is iterated for each frame in the video sequence.

- Tracking Evaluation: Finally, the quality and accuracy of the object tracking system need to be assessed. This can be done by comparing the tracked object positions with ground truth annotations or by evaluating tracking metrics such as Intersection over Union (IoU) and tracking precision/recall.

By following these steps, an object tracking system using YOLOv4 can achieve accurate and real-time tracking of objects across video frames. The combination of YOLOv4's strong object detection capabilities and effective tracking techniques enables robust visual perception in various applications, including surveillance, autonomous vehicles, and object recognition systems.

2. Software Requirements

2.1. Software:

- Software used is Microsoft Visual Studio Code 1.79.3
- Python Version 3.10.9
- A virtual environment is created using the following commands :-

    - *python -m venv myenv* - The above command creates a virtual environment which will be a subset of the main environment.
    - Install packages for runtime environment and python package.
    - Select the python.exe file as an interpreter
    - In PowerShell enable virtual environment by typing the command:- *Set-ExecutivePolicy –ExecutionPolicy unrestricted -Scope CurrentUser*
    - Now we activate the environment by the command:- *.\Scripts\activate*
    - To deactivate the environment type the command:- deactivate in terminal.

2.2. Library Packages used in implementation

The following imports were made during the implementation of Image deblur and Object Tracking:

- OS
- numpy
- cv2
- tensorflow
- matplotlib
- gradio
- math
- skimage
- Pillow

3. Implementation

3.1. Implementation Algorithms for For Image deblur using AutoEncoder

3.1.1. Apply Gaussian Blur

```
#Apply Gaussian blur

from os import listdir
from os.path import isfile, join
import numpy.ndarray as np
import cv2 as cv
from matplotlib import pyplot as plt
mypath='C:\\Users\\arnav\\Desktop\\Shrey\\20072023\\Dataset'
onlyfiles=[ f for f in listdir(mypath) if isfile(join(mypath,f))]
images1=np.empty(len(onlyfiles), dtype=object)
for n in range(0, len(onlyfiles)):
    images1[n]=cv.imread(join(mypath,onlyfiles[n]))
    img = cv.cvtColor(images1[n], cv.COLOR_BGR2RGB)
# asset img is not None, "file could not be read, check with os.path.exist()"
    blur=cv.GaussianBlur(img,(11,11),8)
#     plt.subplot(121),plt.imshow(img), plt.title('Original')
#     plt.xticks([]),plt.yticks([])
#     plt.subplot(122),plt.imshow(blur),plt.title('Gaussian Blur')
#     plt.show()
    #writing the images in a folder output_images
    cv.imwrite("C:\\Users\\arnav\\Desktop\\Shrey\\20072023\\New Folder",messigrey.jpeg)
#     cv.imshow('Color image', img)
#     # wait for 1 second
#     k = cv.waitKey(1000)
#     # destroy the window
#     cv.destroyAllWindows()
```

3.1.2. For Image Augmentation:

```
#Augmentation
from keras.preprocessing.image import ImageDataGenerator
from skimage import io
datagen = ImageDataGenerator(
    rotation_range = 40,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
```

```
    brightness_range = (0.5, 1.5))
import numpy as np
import os
from PIL import Image
image_directory = 'C:\\Users\\arnav\\Desktop\\Shrey\\23062023\\Ships dataset\\train\\images\\'
SIZE = 224
dataset = []
my_images = os.listdir(image_directory)
for i, image_name in enumerate(my_images):
    if (image_name.split('.')[1] == 'jpg'):
        image = io.imread(image_directory + image_name)
        image = Image.fromarray(image, 'RGB')
        image = image.resize((SIZE,SIZE))
        dataset.append(np.array(image))
x = np.array(dataset)
i = 0
for batch in datagen.flow(x, batch_size=16,
                          save_to_dir= 'C:\\Users\\arnav\\Desktop\\Shrey\\23062023\\Ships dataset\\Augmented',
                          save_prefix='dr',save_format='jpg'):
    i+=1
    if i>50:
        break;
```

3.1.3. Autoencoder code:

```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
# import tensorflow as tf
from tensorflow import keras
# Define the autoencoder model
def autoencoder_model():
    # Encoder
    input_img = keras.layers.Input(shape=(128, 128, 3))
    x = keras.layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
    x = keras.layers.MaxPooling2D((2, 2), padding='same')(x)
    x = keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    x = keras.layers.MaxPooling2D((2, 2), padding='same')(x)
    encoded = keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    # Decoder
    x = keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
    x = keras.layers.UpSampling2D((2, 2))(x)
    x = keras.layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
    x = keras.layers.UpSampling2D((2, 2))(x)
    decoded = keras.layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)
```

```
    # Create the autoencoder model
    autoencoder = keras.models.Model(input_img, decoded)
    return autoencoder
# Load your image deblurring dataset and preprocess it
def load_dataset():
    dataset_path = 'C:\\Users\\arnav\\Desktop\\Shrey\\23062023\\Fri\\Ships dataset\\train\\images'
    images = []
    for filename in os.listdir(dataset_path):
        img = cv2.imread(os.path.join(dataset_path, filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
        img = cv2.resize(img, (128, 128))  # Resize the image to a fixed size
        images.append(img)
    images = np.array(images)  # Convert images list to NumPy array
    return images
# Preprocess the dataset
dataset = load_dataset()
dataset = dataset.astype('float32') / 255.0  # Normalize the pixel values between 0 and 1
# Split the dataset into training and testing sets
train_size = int(0.7 * len(dataset))
train_images = dataset[:train_size]
test_images = dataset[train_size:]
# Create an instance of the autoencoder model
model = autoencoder_model()
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error',metrics=["acc"])
# Train the autoencoder
model.fit(train_images, train_images, epochs=50, batch_size=32, validation_data=(test_images, test_images))
# # Train the autoencoder
# model.fit(train_images, train_images, epochs=50, batch_size=16, validation_data=(test_images, test_images))
# Perform image deblurring using the trained autoencoder
num_samples = 5  # Number of images to deblur
# Select random images from the test set
random_indices = np.random.randint(0, len(test_images), num_samples)
blurry_images = test_images[random_indices]
# Deblur the selected images
deblurred_images = model.predict(blurry_images)
# Display the original and deblurred images
num_samples = 5
# Select random images from the test set
random_indices = np.random.randint(0, len(test_images), num_samples)
blurry_images = test_images[random_indices]
# Deblur the selected images
deblurred_images = model.predict(blurry_images)
fig, axes = plt.subplots(num_samples, 2, figsize=(10, 15))
fig.tight_layout()
for i in range(num_samples):
```

```
    axes[i, 0].imshow(blurry_images[i])
    axes[i, 0].axis('off')
    axes[i, 0].set_title('Blurry Image')
    axes[i, 1].imshow(deblurred_images[i])
    axes[i, 1].axis('off')
    axes[i, 1].set_title('Deblurred Image')
plt.show()
```

3.1.4. Gradio:

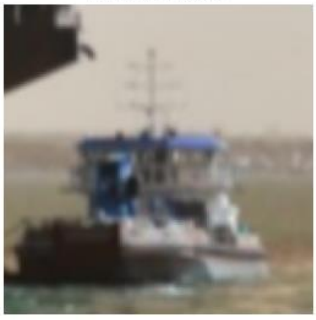
```
import gradio as gr
from PIL import Image
import cv2
import numpy as np
from numpy import asarray
def process_image(input_image):
    # Open the input image
    image = Image.open(input_image.name)
    data=asarray(image)
    gaussian_blur = cv2.GaussianBlur(data, (11,11), 2)
    # Convert the image to grayscale
    # processed_image = image.convert('L')
    sharpened3 = cv2.addWeighted(data,7.5, gaussian_blur, -6.5, 0)
    return sharpened3
# Create a Gradio interface
iface = gr.Interface(
    fn=process_image,
    inputs="file",
    outputs="image",
    title="Image Processing",
    description="Image Deblurring.",
    thumbnail="path/to/thumbnail.png")
# Launch the interface
iface.launch()
```

3.1.5. Screenshots of some Results

epochs=50, batch_size=32

```
Epoch 50/50
162/162 [==============================] - 67s 417ms/step - loss: 0.0011 - acc: 0.8632 - val_loss: 0.0012 - val_acc: 0.88
01
```

Blurry Image



Deblurred Image

Blurry Image

Deblurred Image

epochs=80, batch_size=32,

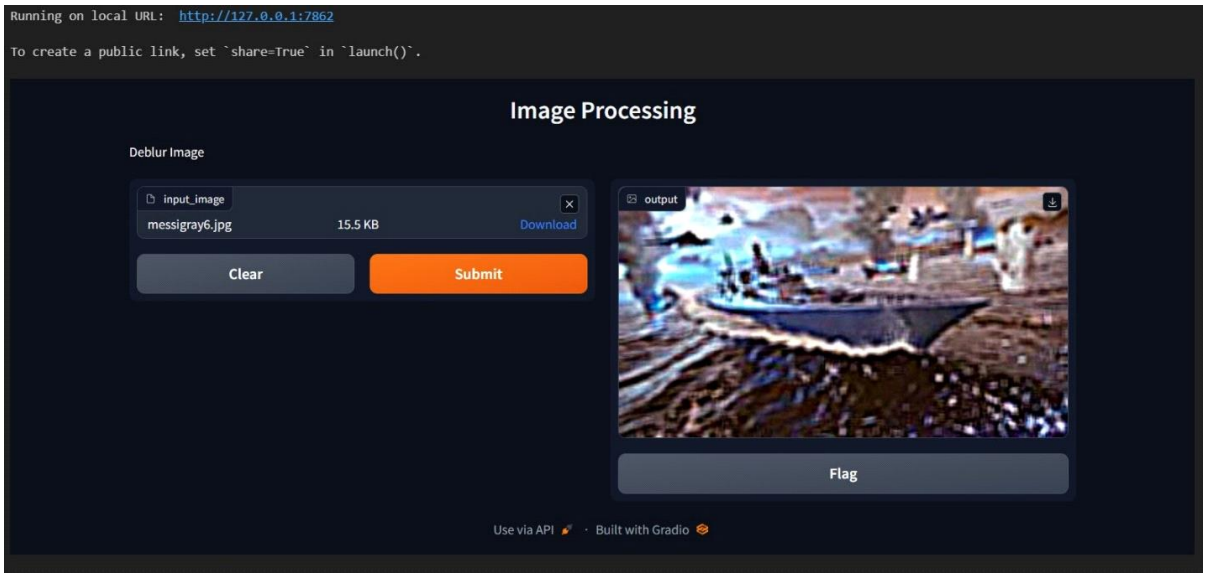
```
Epoch 80/80
162/162 [==============================] - 71s 436ms/step - loss: 8.1211e-04 - acc: 0.8800 - val_loss: 8.4604e-04 - val_a
cc: 0.8905
```

Blurry Image

Deblurred Image

Blurry Image

Deblurred Image



3.1.6. UI for the predicted output

We design our UI using GRADIO. It is a very popular method of designing UI/UX for any AI/ML model. Gardio is an open source python library that permits us to rapidly make simple to utilize, adjustable UI parts for our ML model. It makes it easier to play with our models on our web browser by just dragging and dropping images, text, or recording of our own voice, etc. We can coordinate the GUI straightforwardly into our Python notebook, or we can share the link with anybody.
Gradio helps in building an online GUI in a couple of lines of code which is convenient for showing exhibitions of the model presentation. It is quick, simple to set up, and prepared to utilize, and shareable as the public connection which anybody can get to distantly and parallelly run the model in our machine. Gradio works with a wide range of media-text, pictures, video, and sound. Aside from ML models, it very well may be utilized as should be expected python code embedding.

3.1.7. Expected Results and Impact

The successful implementation of the image deblur model will have several significant impacts. Firstly, it will enable the restoration of blurred images, enhancing their visual quality for various applications such as photography, medical imaging, surveillance, and more. Secondly, the project will contribute to the advancement of deep learning techniques in image processing and computer vision domains.
Lastly, the availability of an efficient Python-based image deblur model will facilitate research and development in the field, fostering innovation and further improvements in image restoration algorithms.

3.1.8. Conclusion of Autoencoder

By addressing the problem of image blur through the development of an image deblur model using Python, this project aims to provide a practical solution for enhancing visual clarity. The proposed deep learning-based approach, along with suitable training, evaluation, and real-time inference strategies, will contribute to the restoration of blurred images in diverse scenarios. This project will not only benefit various application areas but also promote advancements in the field of computer vision and image processing.

3.2. Implementation of Deblur using GAN

3.2.1. Algorithm

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
import os
import cv2
# Generator model
def build_generator():
    model = keras.Sequential()
    model.add(layers.Dense(7 * 7 * 256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same',
use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same',
use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same',
use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)
    return model
# Discriminator model
def build_discriminator():
    model = keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
```

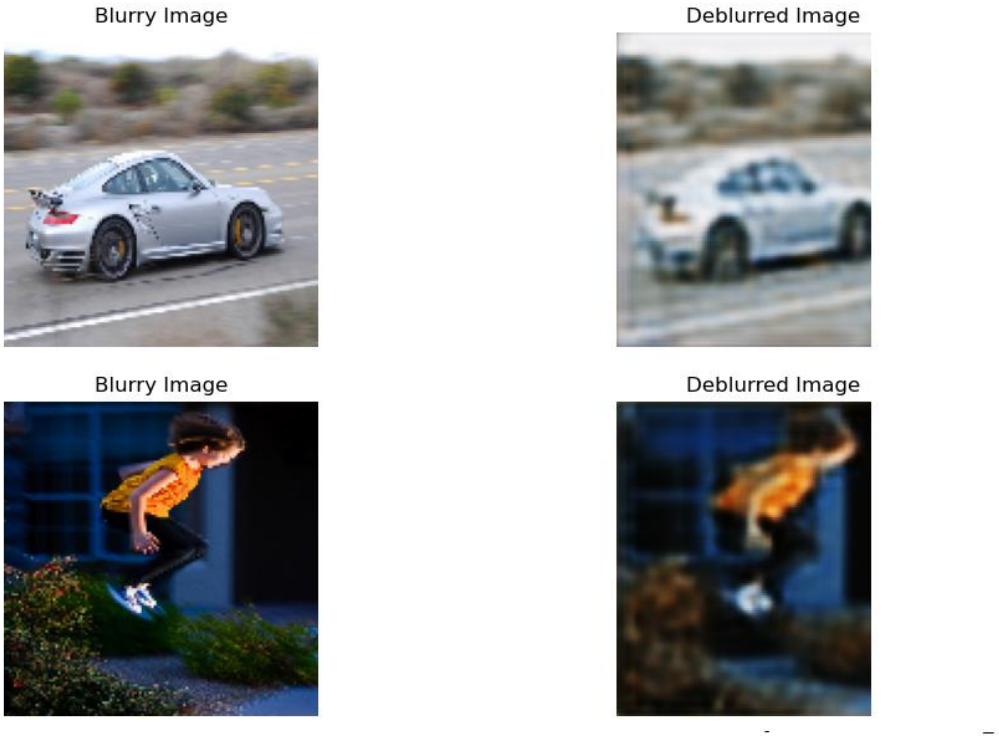
```
                                input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))
    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))
    model.add(layers.Flatten())
    model.add(layers.Dense(1))
    return model
# Combine the generator and discriminator into a GAN
def build_gan(generator, discriminator):
    discriminator.trainable = False
    model = keras.Sequential()
    model.add(generator)
    model.add(discriminator)
    return model
# Load your image deblurring dataset and preprocess it
def load_dataset():
    dataset_path = 'C:\\Users\\shrey\\OneDrive\\Desktop\\Dataset\\Blurred'
    images = []
    for filename in os.listdir(dataset_path):
        img = cv2.imread(os.path.join(dataset_path, filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
        img = cv2.resize(img, (28, 28))  # Resize the image to a fixed size
        images.append(img)
    images = np.array(images)  # Convert images list to NumPy array
    return images
# Preprocess the dataset
dataset = load_dataset()
dataset = dataset.astype('float32') / 127.5  # Normalize the pixel values between 0 and 1
# Set batch size and number of epochs
batch_size = 128
epochs = 50
# Create instances of generator, discriminator, and GAN
generator = build_generator()
discriminator = build_discriminator()
gan = build_gan(generator, discriminator)
# Compile the discriminator
discriminator.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
                      loss=keras.losses.BinaryCrossentropy(from_logits=True))
# Compile the GAN
gan.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
            loss=keras.losses.BinaryCrossentropy(from_logits=True))
# Train the GAN
for epoch in range(epochs):
    for i in range(dataset.shape[0] // batch_size):
        # Generate random noise as input to the generator
        noise = np.random.normal(0, 1, (batch_size, 100))
        # Generate images using the generator
        generated_images = generator.predict(noise)
```

```
        # Select a random batch of real images
        real_images = dataset[i * batch_size : (i + 1) * batch_size]
        # Concatenate real and generated images as input to the discriminator
        combined_images = np.concatenate([real_images, generated_images])
        # Create labels for the discriminator
        labels = np.concatenate([np.ones((batch_size, 1)), np.zeros((batch_size, 1))])
        # Add noise to the labels for regularization
        labels += 0.05 * np.random.random(labels.shape)
        # Train the discriminator
        discriminator_loss = discriminator.train_on_batch(combined_images, labels)
        # Generate random noise as input to the GAN
        noise = np.random.normal(0, 1, (batch_size, 100))
        # Create inverted labels for the generator
        misleading_labels = np.zeros((batch_size, 1))
        # Train the GAN
        gan_loss = gan.train_on_batch(noise, misleading_labels)
    # Print the progress
    print(f"Epoch {epoch+1}/{epochs} | Discriminator loss: {discriminator_loss} | GAN loss: {gan_loss}")
# Generate and display a sample of deblurred images
noise = np.random.normal(0, 1, (10, 100))
generated_images = generator.predict(noise)
```

3.2.2. Output:

epochs=100, batch_size=16

```
Epoch 100/100
13/13 [==============================] - 3s 209ms/step - loss: 0.0108 - val_loss: 0.0103
```



3.2.3. Expected Results and Impact

The successful implementation of the GAN-based image deblurring model is expected to produce visually appealing and sharp images from blurry inputs. By leveraging the power of GANs, the model can learn to capture and restore the intricate details and textures lost due to blurring. This will have a significant impact on various domains such as photography, medical imaging, surveillance, and more. The availability of an effective GAN-based image deblurring model will contribute to advancements in image restoration techniques and foster innovation in the field of computer vision.

3.2.4. Conclusion of GAN

In this project, we have proposed the development of an image deblurring model using Generative Adversarial Networks (GAN). By leveraging the adversarial training framework, the GAN model can learn to restore blurred images and enhance their visual quality. The project aims to design a GAN architecture tailored for image deblurring, train it using a dataset of blurry and sharp image
pairs, and evaluate its performance using suitable metrics. The availability of a GAN-based image deblurring model will have a significant impact on various applications and contribute to the advancement of image restoration techniques.

3.3. Algorithms for Object Detection:

3.3.1. Algorithm

```
import cv2
import numpy as np
class ObjectDetection:
    def __init__(self, weights_path="dnn_model/yolov4.weights",
cfg_path="dnn_model/yolov4.cfg"):
        print("Loading Object Detection")
        print("Running opencv dnn with YOLOv4")
        self.nmsThreshold = 0.4
        self.confThreshold = 0.5
        self.image_size = 608
        # Load Network
        net = cv2.dnn.readNet(weights_path, cfg_path)
        # Enable GPU CUDA
        net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
        net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
        self.model = cv2.dnn_DetectionModel(net)
        self.classes = []
        self.load_class_names()
        self.colors = np.random.uniform(0, 255, size=(80, 3))
        self.model.setInputParams(size=(self.image_size, self.image_size), scale=1/255)
    def load_class_names(self, classes_path="dnn_model/classes.txt"):
        with open(classes_path, "r") as file_object:
            for class_name in file_object.readlines():
                class_name = class_name.strip()
                self.classes.append(class_name)
        self.colors = np.random.uniform(0, 255, size=(80, 3))
        return self.classes
    def detect(self, frame):
        return self.model.detect(frame, nmsThreshold=self.nmsThreshold,
confThreshold=self.confThreshold)
```

3.3.2. Object Tracking:

```
import cv2
import numpy as np
from object_detection import ObjectDetection
import math
```

```
# Initialize Object Detection
od = ObjectDetection()
cap = cv2.VideoCapture("pexels-rihardclementciprian-diac-1192116-1920x1080-
30fps.mp4")
# Initialize count
count = 0
center_points_prev_frame = []
tracking_objects = {}
track_id = 0
while True:
    ret, frame = cap.read()
    count += 1
    if not ret:
        break
    # Point current frame
    center_points_cur_frame = []
    # Detect objects on frame
    (class_ids, scores, boxes) = od.detect(frame)
    for box in boxes:
        (x, y, w, h) = box
        cx = int((x + x + w) / 2)
        cy = int((y + y + h) / 2)
        center_points_cur_frame.append((cx, cy))
        #print("FRAME N°", count, " ", x, y, w, h)
        # cv2.circle(frame, (cx, cy), 5, (0, 0, 255), -1)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    # Only at the beginning we compare previous and current frame
    if count <= 2:
        for pt in center_points_cur_frame:
            for pt2 in center_points_prev_frame:
                distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])
                if distance < 20:
                    tracking_objects[track_id] = pt
                    track_id += 1
    else:
        tracking_objects_copy = tracking_objects.copy()
        center_points_cur_frame_copy = center_points_cur_frame.copy()
        for object_id, pt2 in tracking_objects_copy.items():
            object_exists = False
            for pt in center_points_cur_frame_copy:
                distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])
                # Update IDs position
                if distance < 20:
                    tracking_objects[object_id] = pt
                    object_exists = True
                    if pt in center_points_cur_frame:
```

```
                    center_points_cur_frame.remove(pt)
                continue
            # Remove IDs lost
            if not object_exists:
                tracking_objects.pop(object_id)
        # Add new IDs found
        for pt in center_points_cur_frame:
            tracking_objects[track_id] = pt
            track_id += 1
    for object_id, pt in tracking_objects.items():
        cv2.circle(frame, pt, 5, (0, 0, 255), -1)
        cv2.putText(frame, str(object_id), (pt[0], pt[1] - 7), 0, 1, (0, 0, 255), 2)
    print("Tracking objects")
    print(tracking_objects)
    print("CUR FRAME LEFT PTS")
    print(center_points_cur_frame)
    cv2.imshow("Frame", frame)
    # Make a copy of the points
    center_points_prev_frame = center_points_cur_frame.copy()
    key = cv2.waitKey(1)
    if key == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

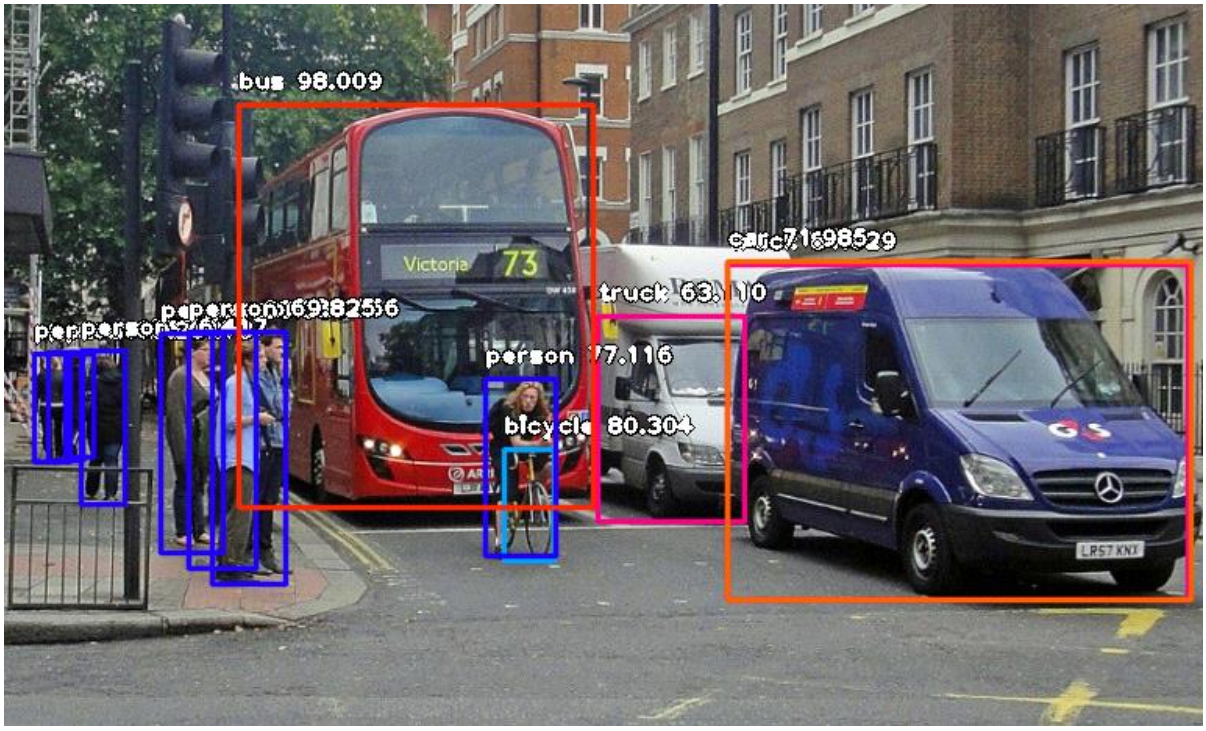
3.3.3. Expected Results and Impact:

The implementation of object tracking using YOLOv4 is expected to yield several significant results and have a considerable impact on the field of computer vision and related applications. Here are some expected results and impacts:

- Enhanced Tracking Accuracy: By leveraging the accurate object detection capabilities of YOLOv4, the object tracking system is anticipated to achieve improved tracking accuracy compared to traditional methods. The integration of YOLOv4's object detection with efficient tracking techniques can enhance the system's ability to handle complex scenarios, occlusions, and scale changes, resulting in more precise and reliable object tracking.
- Real-Time Performance: YOLOv4's efficiency in terms of inference speed allows for real-time object detection and tracking. The expected impact is the ability to track objects in videos at high frame rates, enabling time-sensitive applications that demand immediate responses. This real-time performance can be crucial in surveillance systems, autonomous vehicles, and interactive environments where timely object tracking is essential.

- Robustness Against Environmental Factors: The object tracking system using YOLOv4 is expected to exhibit improved robustness against environmental factors such as varying lighting conditions and cluttered backgrounds. By leveraging YOLOv4's capability to handle complex scenes, the system can maintain reliable object tracking even in challenging visual conditions, reducing false positives and enhancing overall tracking performance.
- Multiple Object Tracking: The integration of YOLOv4 into the tracking framework allows for simultaneous tracking of multiple objects within a scene. This capability has the potential to significantly impact applications that involve tracking multiple objects, such as crowd analysis, traffic monitoring, and sports event analysis.
- Practical Deployment: The research on object tracking using YOLOv4 aims to develop a practical and deployable solution. The expected impact is the availability of a reliable and efficient object tracking system that can be readily implemented in real-world scenarios. This can accelerate the adoption of YOLOv4-based tracking approaches in various industries, contributing to advancements in surveillance, automation, and human-computer interaction.

In summary, the expected results of object tracking using YOLOv4 encompass enhanced tracking accuracy, real-time performance, robustness against environmental factors, multiple object tracking capabilities, and practical deployment potential. These outcomes have the potential to make a significant impact on various domains, fostering advancements in computer vision applications and enabling more efficient and reliable object tracking systems.



3.3.4. Conclusion for Object Detection

In conclusion, the utilization of YOLOv4 for object tracking presents a promising approach to enhance real-time visual perception in computer vision applications. By combining the robust object detection capabilities of YOLOv4 with efficient tracking techniques, accurate and efficient object tracking can be achieved.

The integration of YOLOv4 into the object tracking pipeline offers several advantages. It enables precise localization and identification of objects in the first frame, serving as a foundation for subsequent tracking. YOLOv4's real-time performance ensures that the tracking system operates at high frame rates, meeting the demands of time-sensitive applications.

Moreover, the object tracking system using YOLOv4 exhibits improved robustness against challenging environmental factors such as occlusions, scale variations, and cluttered backgrounds. This resilience enables reliable tracking even in complex scenarios, enhancing the system's overall accuracy and reducing false positives.

The ability to track multiple objects simultaneously further expands the potential applications of the system. It facilitates crowd analysis, traffic monitoring, and other scenarios involving the tracking of numerous objects within a scene.

Overall, object tracking using YOLOv4 holds significant promise for advancing computer vision. The anticipated outcomes include enhanced tracking accuracy, real-time performance, robustness against environmental factors, and practical deployment potential. These advancements can contribute to the development of more efficient and reliable tracking systems, enabling advancements in surveillance, autonomous vehicles, and various other fields that rely on accurate and real-time object tracking.

4. Appendix

4.1. Software Installation Procedure

1. Install Visual Studio Code:
   1.1. Download and install Visual Studio Code from the official website:
   https://code.visualstudio.com/
2. Install Python:
   2.1. Download and install Python from the official website:
   https://www.python.org/downloads/
   2.2. Make sure to check the "Add Python to PATH" option during installation.
3. Open Visual Studio Code:
   3.1. Launch Visual Studio Code after installation.
4. Install Required Extensions:
5. Click on the "Extensions" icon on the left sidebar in VSCode.
6. Search for "Python" in the Extensions Marketplace and install "Python" by Microsoft.

7. Set Up a Virtual Environment:
   7.1.Open the integrated terminal in Visual Studio Code (View → Terminal).
8. Create a new virtual environment by running the following command:
   python -m venv venv
9. Activate the virtual environment:
   9.1.On Windows: venv\Scripts\activate
   9.2.On macOS/Linux: source venv/bin/activate
10. Install Required Python Packages:
    10.1. With the virtual environment activated (if used), install the necessary Python packages for the project. This might include packages like NumPy, OpenCV, TensorFlow, PyTorch, or any other library needed for your specific implementation.
    10.2. For example: pip install numpy opencv-python tensorflow
11. Open the Image Project folder:
    11.1. Use the "Open Folder" option in VSCode to navigate to the location where we want to make the project.
12. Create the files and write the code for the same.
13. Run the Application:
    13.1. Locate the main Python script or Jupyter Notebook file that implements your project functionality.
    13.2. If necessary, modify any file paths or configurations to suit your system setup.
    13.3. Run the script by clicking the "Run" button at the top right corner of the editor or use the integrated terminal to execute the script.
14. Deploy and Test Images:
    14.1. Use any library supported by python like gradio or flask to deploy the model.
    14.2.

4.2. Dataset Used

Ships Image Dataset

Description

1. It includes 8506 images.
2. Ships are annotated in YOLO v5 PyTorch format.
3. The following pre-processing was applied to each image:
   - Auto-orientation of pixel data (with EXIF-orientation stripping)
   - Resize to 600x416 (Stretch)
4. The following augmentation was applied to create 3 versions of each source image:
   - 50% probability of horizontal flip
   - Random Gaussian blur of between 0 and 3.75 pixels

Classes

Aircraft Carrier, Bulkers, Car Carrier, Container Ship, Cruise, DDG, Recreational, Sailboat, Submarine, Tug

4.3. Dataset Source

Most of the dataset images used in image deblur project was downloaded from kaggle. The videos used in object tracking were downloaded from a website named pexels.

4.4. Dataset Split

During training the test and train dataset is divided at I optimal ratio of 30:70 to get best accuracy. The data split was performed randomly while ensuring an equal representation of different blur types in each set.

4.5. Major Imports

4.5.1. **Opencv**

OpenCV (Open Source Computer Vision Library) is a popular open-source library for computer vision and image processing tasks. It offers a wide range of functions and tools for tasks like image and video manipulation, object detection, face recognition, feature extraction, and more. It supports various programming languages, making it accessible and versatile for developers.

4.5.2. **Tensorflow**

TensorFlow is an open-source deep learning framework developed by Google. It enables building and training neural networks for various machine learning tasks. With a flexible architecture, it supports high-level APIs like Keras and provides GPU acceleration for faster computations. TensorFlow is widely used in research and production for its scalability and ease of use.

4.5.3. **Gradio**

Gradio is an open-source Python library that simplifies building and sharing custom interfaces for machine learning models. With a few lines of code, developers can create interactive UIs for model inputs and visualize outputs. Gradio supports various input types, like text, images, and audio, making it user-friendly and accessible for non-technical users.