RainCycleGAN: Improving Object Detection Performance using CycleGAN Augmentation with rainy windshield and surrounding Images

Authors: Siddharth Salvi (ss184), Kevin Dsouza (kevind8)

Abstract:

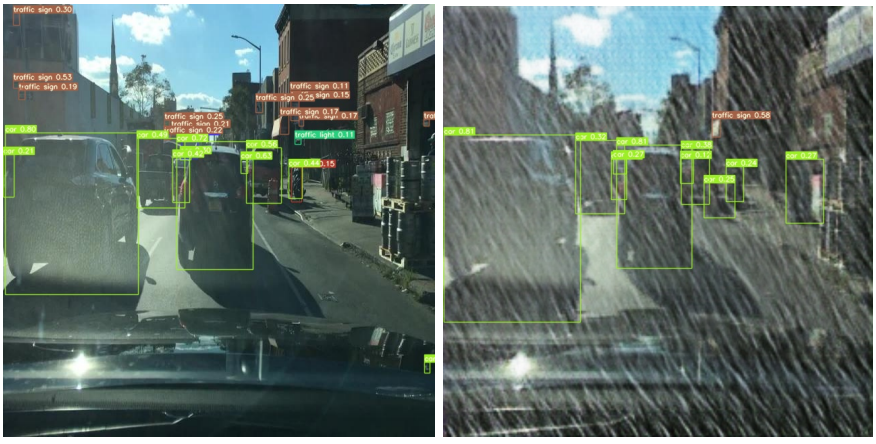
In this project, we propose a novel approach for image augmentation and object detection using the Rain CycleGAN and YOLO (You Only Look Once) algorithms. The goal is to improve the performance of object detection models under adverse weather conditions, specifically rain. We utilize the BDD100k dataset, which contains a large collection of urban driving scenes captured in various weather conditions, including rain.

First, we employ the Rain CycleGAN, a specialized variant of the CycleGAN architecture, to generate synthetic rainy images from the existing BDD100k dataset. The Rain CycleGAN is trained to transform clear weather images into realistic rainy scenes, thereby augmenting the dataset with a diverse range of weather conditions. This augmentation aims to enhance the robustness and generalization of subsequent object detection models, enabling them to effectively handle rainy environments.

Next, we integrate the augmented dataset into the YOLO framework, a popular real-time object detection algorithm known for its efficiency and accuracy. YOLO operates by dividing an image into a grid and predicting bounding boxes and class probabilities for each grid cell. By training YOLO on the augmented dataset, we aim to improve its ability to detect and classify objects accurately in rainy conditions.

Through this combined approach, we expect to achieve significant advancements in object detection performance under challenging weather conditions. The results obtained from our experiments will provide valuable insights into the effectiveness of the Rain CycleGAN and YOLO algorithms for augmenting and detecting objects in rainy urban driving scenes.

Example Result:



Introduction:
Object detection in computer vision plays a crucial role in various applications, such as autonomous driving, surveillance, and robotics. However, one of the significant challenges in real-world scenarios is adverse weather conditions, particularly rain, which can severely degrade the performance of object detection models. Raindrops on camera lenses and wet road surfaces often lead to image degradation, reduced visibility, and occluded objects, making accurate detection a difficult task.
To address this challenge, we propose a two-fold approach to enhance object detection under rainy conditions. Firstly, we employ the Rain CycleGAN algorithm, an extension of the CycleGAN architecture designed to generate synthetic rainy images. The Rain CycleGAN leverages the power of generative adversarial networks to learn the mapping between clear weather images and corresponding rainy scenes. By training the Rain CycleGAN on the BDD100k dataset, which contains a diverse range of urban driving scenes, we can augment the dataset with realistically rendered rainy images. This augmentation enables the subsequent object detection model to learn from a broader set of weather conditions, thereby improving its generalization and robustness in detecting objects in the presence of rain.
Secondly, we integrate the augmented dataset into the YOLO (You Only Look Once) framework, a state-of-the-art real-time object detection algorithm. YOLO divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell, enabling efficient and accurate object detection. By training YOLO on the augmented dataset, we aim to enhance its ability to detect objects accurately in rainy urban driving scenes. The combined approach of Rain CycleGAN and YOLO holds great promise in mitigating the challenges posed by adverse weather conditions and enhancing the safety and reliability of computer vision systems deployed in real-world scenarios.
In this project, we will evaluate the performance of the proposed approach on the BDD100k dataset, specifically focusing on object detection accuracy under rainy conditions. We will compare the results obtained using the augmented dataset against the baseline YOLO model trained on clear weather images only. Through quantitative and qualitative analysis, we expect to demonstrate the effectiveness of our approach in improving object detection in rainy urban environments. The outcomes of this research have the potential to contribute significantly to the development of more robust and reliable computer vision systems, particularly in autonomous driving and surveillance applications.

Related Work
The paper [5] discusses various techniques that have been proposed for improving object detection performance under rainy conditions, which is a challenging task due to the presence of rain streaks, fog, and water droplets on the camera lens.
The techniques mentioned in the paper can be broadly classified into three categories:

Image Enhancement Techniques: These techniques focus on improving the visual quality of rainy images before they are fed into the object detection algorithm. Some examples include contrast enhancement, deblurring, and denoising. These techniques aim to remove or reduce the effect of rain streaks, fog, and other visual distortions caused by the rain.

Domain Adaptation Techniques: These techniques aim to train the object detection algorithm on synthetic or real rainy images, so that it can learn to detect objects in such conditions. Some examples include generating synthetic rainy images, using adversarial training to adapt the model to rainy conditions, and fine-tuning the model on real rainy images.
Fusion-based Techniques: These techniques combine information from multiple sensors or modalities to improve object detection performance under rainy conditions. Some examples

include fusing visual and thermal images, using radar or lidar sensors to detect objects, and combining multiple object detection models trained on different modalities.

Details of the approach:

Datasets:

1. BDD100k Dataset:
   The BDD100k dataset [4] is a large-scale driving video dataset that contains high-quality annotations for various computer vision tasks related to autonomous driving. It was created by a team of researchers at the University of California, Berkeley, and was released in 2018.
   The BDD100k dataset consists of over 100,000 videos of driving scenes collected from different parts of the world. The videos were recorded from the dashboard of a car and cover a wide range of driving scenarios, including urban, suburban, and rural environments, as well as different weather and lighting conditions. The videos are up to 40 seconds long and have a resolution of 720p.
   The BDD100k dataset provides annotations for three different computer vision tasks: object detection, semantic segmentation, and instance segmentation. The object detection task involves detecting and classifying objects of interest in the scene, such as cars, pedestrians, and traffic signs. The semantic segmentation task involves labeling each pixel in the image with a class label, such as road, sidewalk, building, or sky. The instance segmentation task involves identifying and segmenting each instance of a particular object class in the scene, such as each individual car or pedestrian.
   The annotations in the BDD100k dataset were generated using a combination of manual and automatic methods, and have been carefully validated to ensure high accuracy and consistency. The dataset also includes detailed metadata for each video, such as GPS coordinates, weather conditions, and lighting conditions, which can be useful for training and evaluating models in different scenarios.

2. Rainy Image Dataset:
   The GitHub repository "jinnovation/rainy-image-dataset" [3], contains a dataset specifically created for rainy image synthesis and augmentation. The repository aims to provide a comprehensive collection of images that can be used to train models for various computer vision tasks in rainy weather conditions.
   The dataset in this repository includes both clear weather images and their corresponding rainy counterparts. The rainy images are generated using a Rain CycleGAN, a specialized variant of the CycleGAN architecture, which learns the mapping between clear and rainy scenes. This process allows for the creation of realistic rainy images that closely resemble the characteristics of actual rainy weather.

Algorithms:

YOLOv7:
YOLOv7 (You Only Look Once version 7) is an object detection algorithm that belongs to the YOLO (You Only Look Once) family of models. It is designed to efficiently detect and classify objects in images with real-time or near-real-time performance. The key idea behind YOLOv7 is to divide the input image into a grid and predict bounding boxes and class probabilities directly within each grid cell. This approach eliminates the need for region proposal mechanisms and

performs detection in a single pass through the network, hence the name "You Only Look Once."

CycleGAN:
CycleGAN is a type of deep learning model that can learn to transform images from one domain to another in an unsupervised manner. It does this by using two neural networks, a generator and a discriminator, to learn the mapping between two image domains. The generator network takes an image from one domain as input and generates a corresponding image in the other domain as output. The discriminator network is trained to distinguish between real images from the target domain and fake images generated by the generator network. The generator network is trained to fool the discriminator network into believing that its generated images are real. CycleGAN extends this basic architecture by incorporating a cycle consistency loss term, which ensures that the transformation is consistent in both directions. This means that if we transform an image from domain A to domain B, and then back to domain A, we should get an image that is similar to the original image from domain A. This is important because it ensures that the transformation is reversible and preserves the content of the original image.
The CycleGAN implementation followed is the unpaired image to image implementation as followed in [1]. The goal of unpaired image-to-image translation is to learn a mapping between two different image domains without a one-to-one correspondence between the images in the two domains.

PseudoCode:

1. CycleGAN pseudocode:

```
# Define generator models
Generator_G = ResNet_generator()
Generator_F = ResNet_generator()

# Define discriminator models
Discriminator_X = PatchGAN_discriminator()
Discriminator_Y = PatchGAN_discriminator()

# Define input/output placeholders for generators
real_X = Input(shape=input_shape)
real_Y = Input(shape=input_shape)

# Define identity mappings
identity_X = Generator_G(real_X)
identity_Y = Generator_F(real_Y)

# Define generator outputs
fake_Y = Generator_G(real_X)
fake_X = Generator_F(real_Y)

# Define reconstructed images
reconstructed_X = Generator_F(fake_Y)
reconstructed_Y = Generator_G(fake_X)
```

```
# Define adversarial losses
adversarial_loss_G = adversarial_loss(Discriminator_Y(fake_Y), valid)
adversarial_loss_F = adversarial_loss(Discriminator_X(fake_X), valid)

# Define cycle consistency losses
cycle_loss_G = cycle_loss(real_X, reconstructed_X)
cycle_loss_F = cycle_loss(real_Y, reconstructed_Y)

# Define identity losses
identity_loss_G = identity_loss(real_Y, identity_Y)
identity_loss_F = identity_loss(real_X, identity_X)

# Define total generator and discriminator losses
total_generator_loss = (
        adversarial_loss_G + adversarial_loss_F + cycle_loss_G + cycle_loss_F + identity_loss_G + identity_loss_F
)
total_discriminator_loss = discriminator_loss(Discriminator_X, real_X, fake_X) + discriminator_loss(
    Discriminator_Y, real_Y, fake_Y
)

# Compile generator and discriminator models
Generator = Model(inputs=[real_X, real_Y], outputs=[fake_Y, fake_X, reconstructed_X, reconstructed_Y, identity_Y, identity_X])
Discriminator_XY = Model(inputs=real_X, outputs=[validity_X, validity_Y])
Discriminator_YX = Model(inputs=real_Y, outputs=[validity_Y, validity_X])

Generator.compile(loss=[adversarial_loss, adversarial_loss, cycle_loss, cycle_loss, identity_loss, identity_loss],
                  loss_weights=[1, 1, 10, 10, 0.5, 0.5],
                  optimizer=optimizer)

Discriminator_XY.compile(loss=discriminator_loss,                optimizer=optimizer, metrics=['accuracy'])
Discriminator_YX.compile(loss=discriminator_loss,                optimizer=optimizer, metrics=['accuracy'])

Losses Pseudocode:
# Adversarial loss
def adversarial_loss(y_true, y_pred):
    return K.mean(K.square(y_pred - y_true))

# Cycle-consistency loss
def cycle_loss(y_true, y_pred):
    return K.mean(K.abs(y_pred - y_true))

# Identity loss
def identity_loss(y_true, y_pred):
    return K.mean(K.abs(y_pred - y_true))
```

```
# Discriminator loss
def discriminator_loss(discriminator, real, fake):
    # Real loss
    loss_real = adversarial_loss(valid, discriminator(real))

    # Fake loss
    loss_fake = adversarial_loss(fake, discriminator(fake))

    # Total loss
    total_loss = 0.5 * (loss_real + loss_fake)
    return total_loss
```

2. YoloV7 pseudocode:

```
# Define YOLOv7 architecture
model = YOLOv7(input_shape, num_classes)
model.summary()

# Set hyperparameters
learning_rate = 0.001
batch_size = 16
num_epochs = 100

# Define loss function and optimizer
loss_function = focal_loss
optimizer = Adam(learning_rate=learning_rate)

# Split dataset into training, validation, and testing sets
train_set, val_set, test_set = split_dataset(bdd100k)

# Define data generators for training and validation sets
train_generator = DataGenerator(train_set, batch_size=batch_size, shuffle=True)
val_generator = DataGenerator(val_set, batch_size=batch_size, shuffle=False)

# Train the YOLOv7 model
for epoch in range(num_epochs):
    # Iterate over each batch of the training set
    for batch_idx, (images, targets) in enumerate(train_generator):
        # Forward pass through the model
        predictions = model(images)

        # Compute loss and backpropagate gradients
        loss_value = loss_function(targets, predictions)
        gradients = tape.gradient(loss_value, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

```
        # Log training loss
        train_loss = loss_value.numpy().mean()
        tf.summary.scalar('train_loss', train_loss)

    # Evaluate the model on the validation set
    val_loss, val_metrics = evaluate(model, val_generator)
    tf.summary.scalar('val_loss', val_loss)
    for metric_name, metric_value in val_metrics.items():
        tf.summary.scalar(f'val_{metric_name}', metric_value)

    # Save the model checkpoint
    if epoch % 10 == 0:
        checkpoint.save(os.path.join(checkpoint_dir, 'model'), epoch=epoch)

# Evaluate the trained model on the testing set
test_loss, test_metrics = evaluate(model, test_generator)
for metric_name, metric_value in test_metrics.items():
    print(f'Test {metric_name}: {metric_value}')

# Perform inference on new images
for image_path in new_images:
    image = preprocess_image(image_path)
    predictions = model(image)
    detections = postprocess_detections(predictions)
    print(f'Detected {len(detections)} objects in {image_path}')
```

Implementation Details:

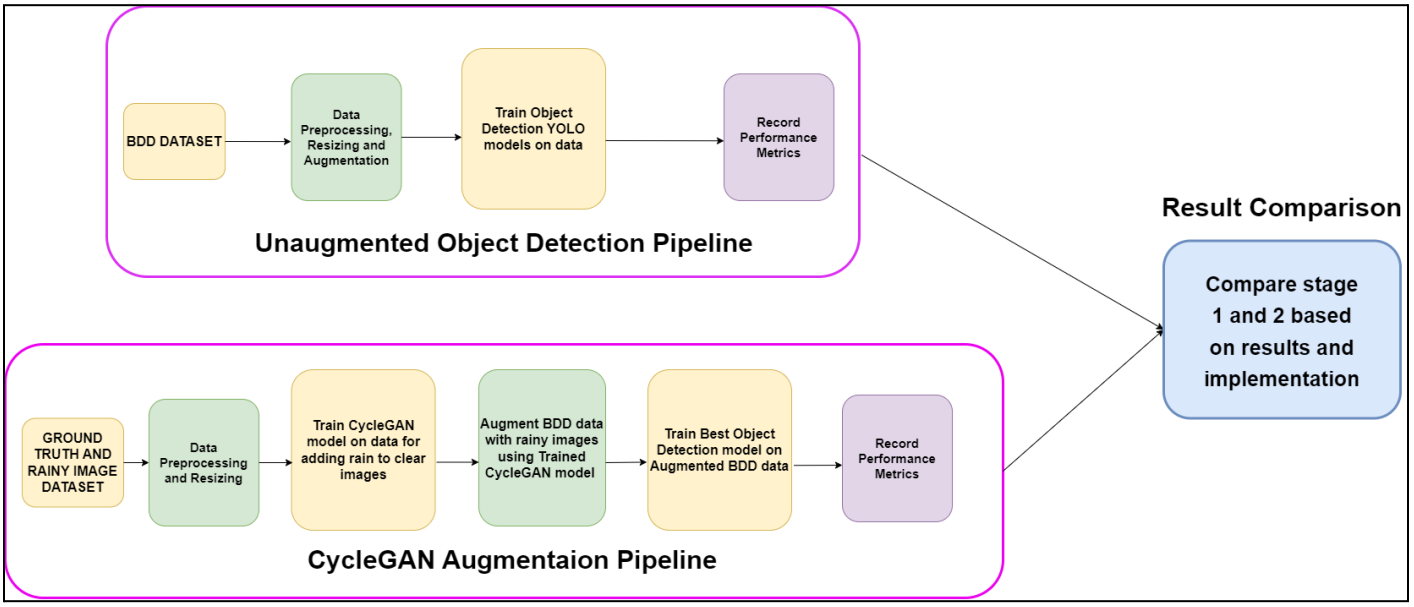


Fig1: Architecture Diagram

The tasks performed in the pipeline can be summarized as follows:

1. Baseline Model: Trained YOLOv7 on the BDD100k dataset to establish a baseline for object detection. This step involved training the model using the provided data and evaluating its performance.
2. Rain Augmentation with CycleGAN on RainyImage Dataset: Utilized CycleGAN as implemented in [1], to generate synthetic rainy images based on the RainyImage dataset. This involved training the CycleGAN model using clear and rainy image pairs to learn the mapping between the two domains.
3. Rain Augmentation with CycleGAN on BDD100k Dataset: Applied the trained CycleGAN model from the previous step to the BDD100k dataset. The goal was to generate additional augmented rainy images for the BDD100k dataset, leveraging the learned mapping from the RainyImage dataset. This process aimed to enhance the dataset with realistic rainy variations.
4. YOLOv7 on Augmented Images: Ran the YOLOv7 model on the augmented images generated through the CycleGAN-based rain augmentation. The purpose was to evaluate the performance of the YOLOv7 model on the augmented dataset and assess its ability to detect objects under rainy conditions.

By following this pipeline, we incorporated rain augmentation using CycleGAN at two stages: first on the RainyImage dataset and then on the BDD100k dataset. This approach aimed to improve the model's robustness and ability to handle rainy weather scenarios. The final step involved running YOLOv7 on the augmented images to assess its performance and evaluate the impact of the rain augmentation on object detection accuracy.

Results:

1. Object Detection using Yolov7 model on BDD100k dataset (Baseline Model):

   Hyperparameters Used:

   1. Learning Rate : 0.01
   2. Learning Rate Factor : 0.1
   3. Momentum : 0.937
   4. Weight Decay: 0.0005

   Output Images:

Fig: Object Detection result 1

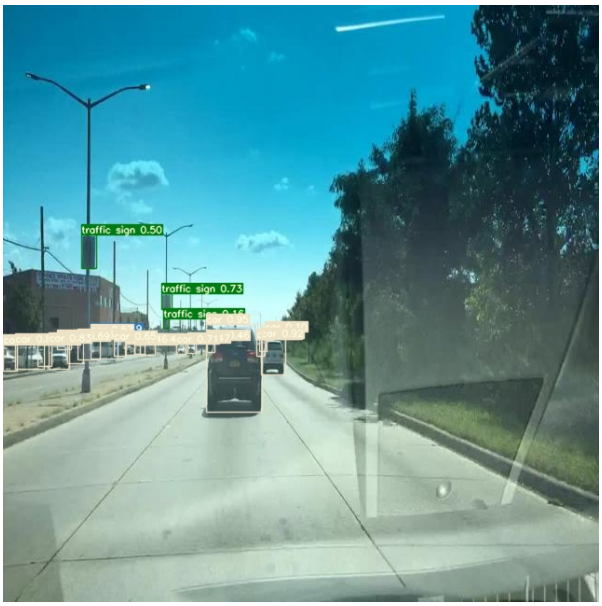


Fig: Object Detection result 2

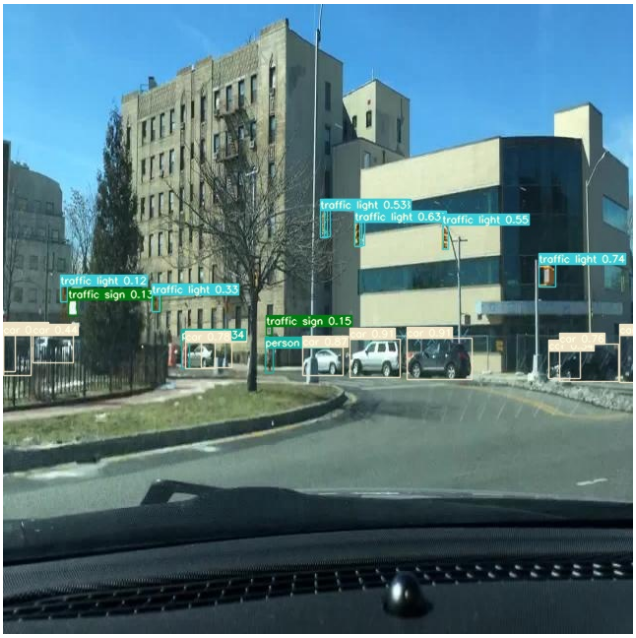


Fig: Object Detection result 3

Results Table:

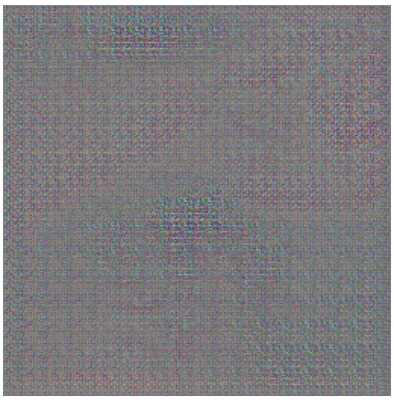
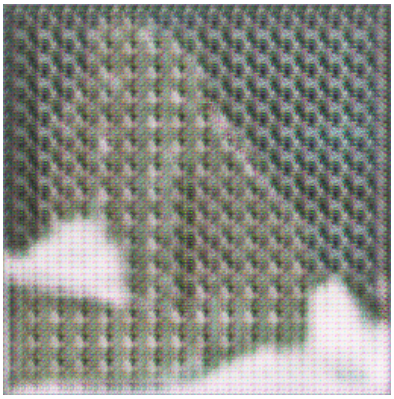
Class	Images	Labels	P	R	mAP@.5	mAP.5:.95
all	717	14549	0.716	0.465	**0.51**	0.265
bike	717	87	0.61	0.46	0.474	0.221
bus	717	184	0.727	0.477	0.558	0.396
car	717	8005	0.797	0.742	0.799	0.484
motor	717	35	0.743	0.457	0.514	0.208
person	717	1392	0.729	0.397	0.467	0.318
rider	717	76	0.523	0.504	0.572	0.296
Traffic light	717	1707	0.712	0.558	0.501	0.381
Traffic sign	717	2679	0.786	0.546	0.492	0.288
train	717	1	1	0	0	0
truck	717	383	0.572	0.509	0.533	0.564

2. Rain augmentation using CycleGAN on RainyImage Dataset:

Hyperparameters Used:

1. n_epochs: Number of epochs with the initial learning rate. Default value is 30.
2. n_epochs_decay: Number of epochs to linearly decay the learning rate to zero. Default value is 60.
3. beta1: Momentum term of the Adam optimizer. Default value is 0.5.
4. lr: Initial learning rate for the Adam optimizer. Default value is 0.00010.
5. gan_mode: The type of GAN objective. Possible values are 'vanilla', 'lsgan', or 'wgangp'. Default value is 'lsgan'.
6. pool_size: The size of the image buffer that stores previously generated images. Default value is 50.
7. lr_policy: Learning rate policy. Possible values are 'linear', 'step', 'plateau', or 'cosine'. Default value is 'linear'.
8. lr_decay_iters: Multiply the learning rate by a gamma every lr_decay_iters iterations. Default value is 50.

Output Images:

Epoch	Real Ground Truth	Fake Rainy Image
1		
5		

10		
15		
20		
40		

80		

Results:

```
(epoch: 84, iters: 234, time: 0.329, data: 0.005) G_A: 0.428 G_B: 0.225 G: 3.767
(epoch: 84, iters: 534, time: 0.142, data: 0.007) G_A: 0.510 G_B: 0.273 G: 4.026
(epoch: 84, iters: 834, time: 0.383, data: 0.004) G_A: 0.509 G_B: 0.287 G: 3.008
saving the model at the end of epoch 84, iters 84168
```

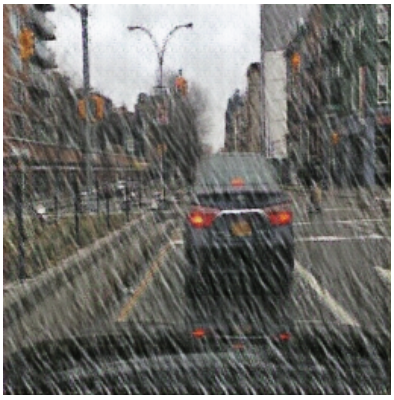
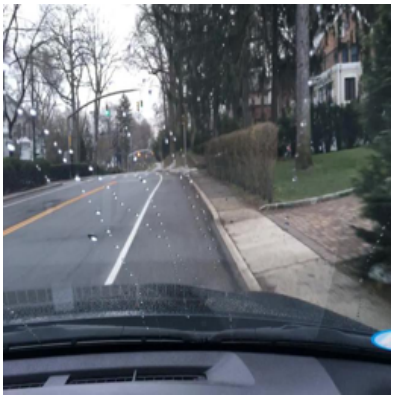
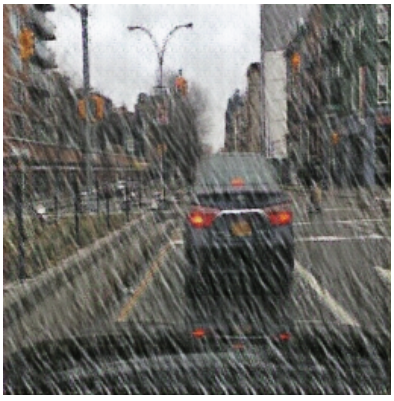
Fig: Output Loss for cycleGAN

3. Rain augmentation using CycleGAN on BDD100K Dataset:

Hyperparameters Used:

9. n_epochs: Number of epochs with the initial learning rate. Default value is 30.
10. n_epochs_decay: Number of epochs to linearly decay the learning rate to zero. Default value is 60.
11. beta1: Momentum term of the Adam optimizer. Default value is 0.5.
12. lr: Initial learning rate for the Adam optimizer. Default value is 0.00010.
13. gan_mode: The type of GAN objective. Possible values are 'vanilla', 'lsgan', or 'wgangp'. Default value is 'lsgan'.
14. pool_size: The size of the image buffer that stores previously generated images. Default value is 50.
15. lr_policy: Learning rate policy. Possible values are 'linear', 'step', 'plateau', or 'cosine'. Default value is 'linear'.
16. lr_decay_iters: Multiply the learning rate by a gamma every lr_decay_iters iterations. Default value is 50.

No.	Ground Truth	Generated Rain Image

1		
2		
3		
4		

5		

4. Object Detection using Yolov7 model on BDD100k dataset (Augmented):

Hyperparameters Used:

5. Learning Rate : 0.01
6. Learning Rate Factor : 0.1
7. Momentum : 0.937
8. Weight Decay: 0.0005

Output Images:

Fig: Augmented Image output 1

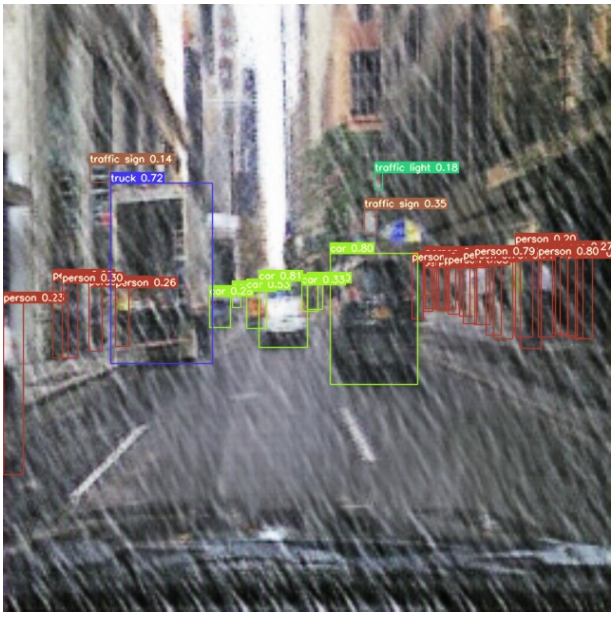


Fig: Augmented Image output 2

Fig: Augmented Image output 3

Fig: Augmented Image output 4

Results:

Class	Images	Labels	P	R	mAP@.5	mAP.5:.95
all	717	14549	0.716	0.565	**0.54**	0.365
bike	717	87	0.61	0.56	0.474	0.321
bus	717	184	0.727	0.577	0.558	0.596
car	717	8005	0.797	0.742	0.799	0.584
motor	717	35	0.743	0.557	0.514	0.308
person	717	1392	0.729	0.697	0.467	0.412
rider	717	76	0.523	0.604	0.572	0.294
Traffic light	717	1707	0.712	0.558	0.501	0.385
Traffic sign	717	2679	0.786	0.746	0.492	0.238
train	717	1	1	0.503	0	0
truck	717	383	0.572	0.709	0.533	0.574

Discussion and conclusions:

Firstly, we executed the baseline model by training YOLOv7 on the BDD100k dataset, allowing us to examine the output of the detected objects in the images. As we can observe from the results section, we reached a mAp of **0.51** for **55 epochs** which we considered as our baseline output.

Subsequently, we proceeded with the initial stage of augmentation, employing CycleGAN to generate synthetic rainy images based on the RainyImage dataset. Through this process, we successfully generated 1000 new images representing rainy conditions.

To further enhance our dataset, we employed the weights obtained from the previous step to train CycleGAN on the BDD100k dataset, generating additional rainy images specific to this dataset.

Finally, we trained YOLOv7 on the augmented images, utilizing the same hyperparameters as in the initial training. The mAP value for this model was **0.54** for **55 epochs**. Remarkably, the evaluation results demonstrate noticeable improvement, indicating that the model performs effectively on the newly augmented images.

By implementing this pipeline, we were able to augment the BDD100k dataset with realistic rainy variations, consequently enhancing the model's ability to accurately detect objects in challenging weather conditions.

Statement of individual contribution:

Siddharth Salvi (ss184) : Performed cycleGAN on the RainyImage dataset and performed cycleGAN on the BDD100k dataset

Kevin Dsouza (kevind8) : Performed yolov7 on the BDD100k dataset without augmentation and performed yolov7 on the BDD100k dataset with augmentation.

References:

[1] Jun-Yan Zhu*, Taesung Park*, Phillip Isola, and Alexei A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", in IEEE International Conference on Computer Vision (ICCV), 2017. (* indicates equal contributions)

[2] Wang, C-Y., Bochkovskiy, A., & Liao, H-Y.M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv preprint arXiv:2207.02696.

[3] Rainy Image Dataset: https://github.com/jinnovation/rainy-image-dataset

[4] BDD100k Dataset: https://bdd-data.berkeley.edu/

[5] M. Hnewa and H. Radha, "Object Detection Under Rainy Conditions for Autonomous Vehicles: A Review of State-of-the-Art and Emerging Techniques," in IEEE Signal Processing Magazine, vol. 38, no. 1, pp. 53-67, Jan. 2021, doi: 10.1109/MSP.2020.2984801.