	# Hacettepe University Computer Engineering Department **BBM479/480 End of Project Report**

Project Details

Title	Increasing the success rate of object detection by converting RGB images to thermal images
Supervisor	Hacer Yalım Keleş

Group Members

	Full Name	**Student ID**
1	Abdullah Atahan Türk	21827943
2	Yuşa Zorlu	21828054
3		
4		

Abstract of the Project (/ 10 Points)

With the development of computer vision, machine learning and artificial intelligence, object detection has developed and has become an important position today. Object detection with cameras is especially important in the defense industry and traffic. When we started our project, we set out to get better results in object detection.

First of all, in our research on the internet, we realized that with thermal images, object detection can be done with a higher success rate than RGB images. To verify this, we conducted experiments on the LLVIP dataset, which includes 12025 image pairs of the same image pairs taken with the RGB and thermal camera with Yolo v7 in the first semester. We also wanted to further consolidate the results we will obtain by using the FLIR Thermal Dataset. In these experiments we conducted with Yolo v7, we clearly saw that object detection, especially human detection, in thermal images can be achieved with a better success rate than RGB images. On top of that, in the second semester, we aimed to obtain thermal images from RGB images by using various GAN implementations, and thus to enable more successful object detection. We decided to use pix2pix and CycleGan for this. Again, using the same LLVIP dataset, we obtained thermal images from RGB images. We did our pix2pix and CycleGan experiments using Google Colab, as we did not have machines with the necessary equipment. For this, we used the paid pro version of Colab, but in this way, we could not train our dataset exactly as it should be, since Colab gives a certain usage limit. Although people usually appear clearer in the thermal images we obtained, the generality of other objects was not completely clear. With the thermal images we obtained, we again carried out object detection experiments in Yolo v7 and although we did not achieve very good results, we obtained promising results. We have seen that with the necessary hardware, we can produce thermal images with higher object detection success than RGB images with a larger dataset and more epoch numbers.

Introduction, Problem Definition & Literature Review (/ 20 Points)

Our project covers the fields of computer vision, machine learning and artificial intelligence. We focused on making object detection better. The problem we noticed was that it was sometimes difficult to detect objects on RGB images, especially in the defense industry and traffic. It was sometimes very difficult to detect objects with RGB images, especially when the sun was out and the lighting was low.

After detecting this problem, we performed our object detection experiments with Yolo v7 on a large dataset of identical image pairs to detect and confirm the accuracy of this problem. According to Yolo v7 results, object detection in thermal images was more successful than RGB images. The number of objects that could not be detected in thermal image could decrease to 4%. Based on the results we have obtained from these experiments and the research we have done, we have suggested that object detection on thermal images will yield better results than RGB images. However, the main problem in obtaining a thermal image was that thermal cameras were expensive and it was more difficult and costly to obtain a thermal image than to obtain a normal RGB image. That's why we deployed our project on this problem in the second semester. If we could obtain thermal images inexpensively and relatively easily, we could both enable better object detection and expand object detection with thermal images because we reduced the cost. Therefore, in the second semester, we did research on implementations for obtaining thermal images from RGB images. With the guidance of our supervisor, we learned that there are GANs we can use for this. We researched various GAN implementations and decided together with our supervisor that it would be appropriate to use pix2pix and CycleGan for our purposes.

We did our research on CycleGan and pix2pix and examined how these Gans work and their implementations.

Pix2Pix is an influential image-to-image translation method that harnesses the power of generative adversarial networks (GANs) to establish a mapping between input and output images. The core components of Pix2Pix are a generator, responsible for producing outputs that closely resemble the target images based on given inputs, and a discriminator, trained to distinguish between the generated and real target images. Through an adversarial training process, the generator strives to generate images that are indistinguishable from real ones, while the discriminator aims to improve its accuracy in discerning real from generated images.

Training Pix2Pix involves optimizing both the generator and discriminator using a loss function that simultaneously considers the realism of the generated images and their similarity to the target images. This dual objective allows the generator to generate visually plausible and aligned output images that closely resemble the desired transformation. Pix2Pix has demonstrated remarkable success in various image translation tasks, including converting sketches into realistic images, mapping aerial imagery, and transforming day-time visuals into night-time scenes. Its contributions have significantly impacted the field of computer vision research and have served as a catalyst for advancements in generative models. As a result, Pix2Pix continues to inspire further research and development in the realm of image-to-image translation.

CycleGAN is a type of generative adversarial network (GAN) architecture used for unsupervised image-to-image translation. It allows learning the mapping between two different image domains without paired examples. The architecture consists of generators and discriminators, and training involves a cycle consistency loss to ensure that images translated between the domains maintain important visual characteristics. CycleGAN has been successfully used for various tasks, such as transforming images of one type into another, without the need for paired training data. Its ability to perform unsupervised translation makes it valuable in scenarios where obtaining paired data is challenging. CycleGAN has made significant contributions to unsupervised image-to-image translation and remains an active area of research and development.

We experimented with our two image-to-image translation methods, and as a result of our experiments, we found that pix2pix, a supervised model, gave better results. The results we achieved fell short of our expectations, but were still promising. We think that much better results can be obtained if we reach larger datasets and make experiments on better hardware with better financial means.

Finally, our hypothesis is that: It is quite possible to obtain thermal images from RGB images by using GANs developed for image-to-image translation, according to the researches, experiments and results we have done throughout our entire project. If we can obtain a really large dataset of rgb-thermal image pairs and train them with pix2pix on powerful machines with a high number of epochs, we can obtain thermal images with higher object detection success than RGB images. In this way, we can avoid the high costs caused by thermal cameras and provide a faster and more practical thermal image. If this project can be developed, it can cause a real breakthrough for the defense industry and traffic safety.

Methodology (/ 25 Points)

In the beginning, we did a literature research and set up the necessary environments on our computers. For this, we installed YOLOv7 and Pytorch version with cuda support. Afterwards, we downloaded the LLVIP and FLIR RGB -Thermal images datasets that we will use in our experiments. We conducted experiments on the GPU using YOLOv7 on the LLVIP and FLIR datasets. The experiments were performed on a Windows operating system computer with 16 GB of ram, Nvidia GeForce GTX 1050 graphics card.

First of all, we wanted to directly measure the performance of YOLOv7 in RGB and thermal images without training LLVIP dataset. For this, we used YOLOv7's weigth previously trained on MS COCO dataset. The configuration settings we used in YOLOv7 while testing were as follows:

```
python test.py –img 640 –data person_detect.yaml –weights yolov7.pt –name untrained_visible
python test.py –img 640 –data person_detect.yaml –weights yolov7.pt –name untrained_infra
```

In these settings, --img specifies image size, --weights specifies the train weight to be used during the test, --data specifies the information of the images and labels in the dataset to be used.
To train YOLOv7 on RGB images in the LLVIP dataset and then testing on RGB images. The configuration settings we used while training were as follows:

```
python train.py --img 640 --batch 4 --epochs 5 --data person_detect.yaml --weights yolov7.pt
--name visible_trained_export
```

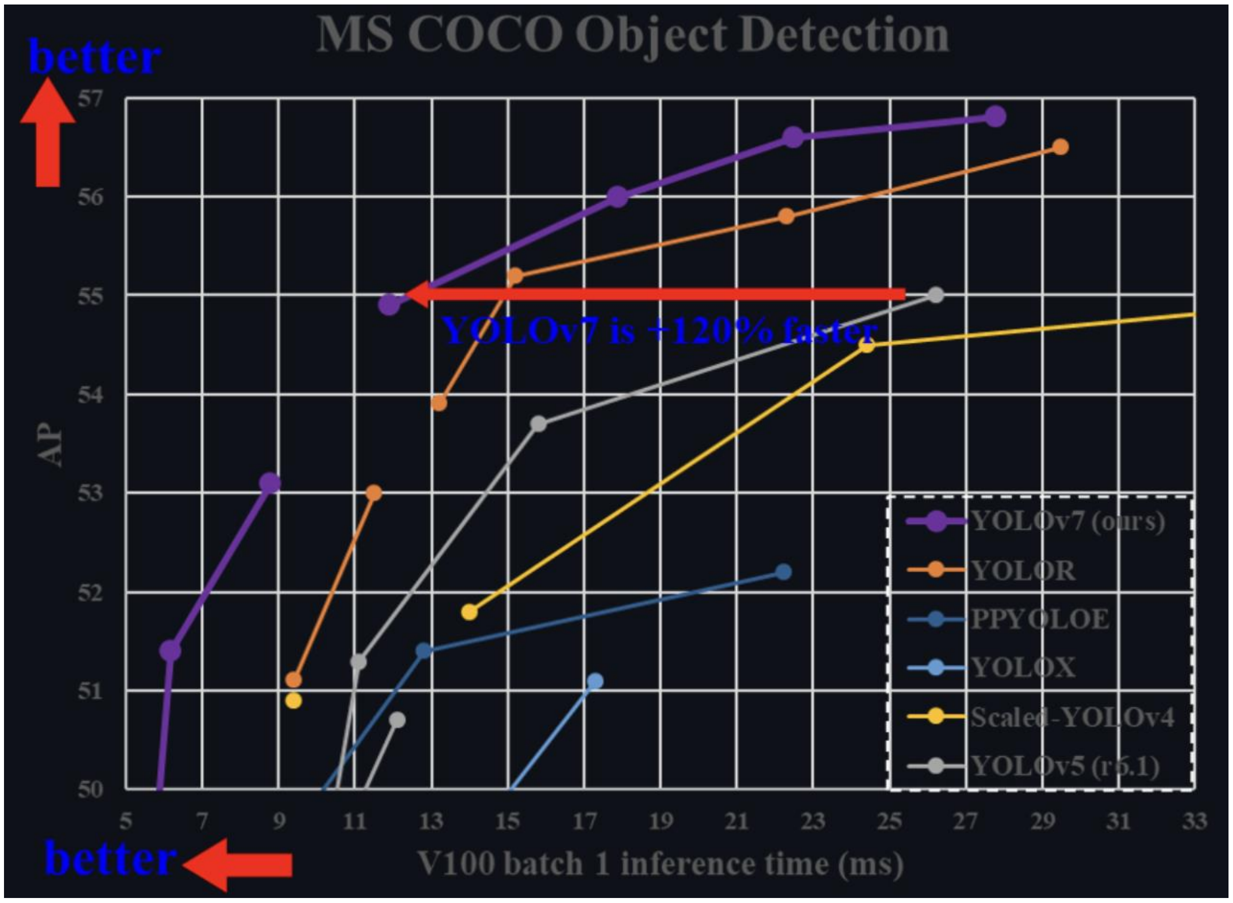
And here's how to test:

```
python test.py –img 640 –data person_detect.yaml –weights
.\YOLOV7\runs\train\visible_trained_export\weights\best.pt –name trained_visible
```

Finally, we trained with the thermal images in the LLVIP dataset and tested on the thermal images using the weight we obtained from here. Here are the configuration settings for training and testing:

```
python train.py --img 640 --batch 4 --epochs 5 --data person_detect.yaml --weights yolov7.pt
--name infra_trained_export
python test.py –img 640 –data person_detect.yaml –weights
.\YOLOV7\runs\train\infra_trained_export\weights\best.pt –name trained_infra_test
```

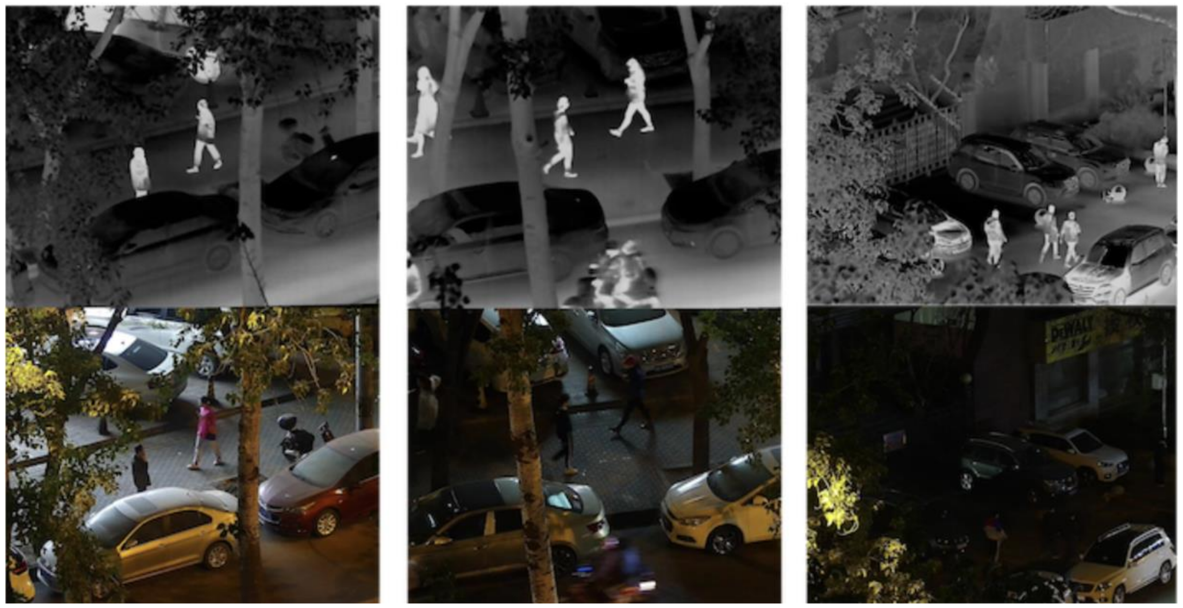
YOLOv7: We have used YOLOv7 for image detection on the first term which we focused on image detection from thermal images for people. We used YOLOv7 on LLVIP. YOLOv7 was the state of the art object detection model when we started our project which had highest accuracy with good performance compared to its accuracy metrics. MS COCO results from the paper is below.



LLVIP: A Visible-infrared Paired Dataset for Low-light Vision is the dataset we used for image detection. The dataset is very handy to use for both image detection and image generation with CycleGAN and Pix2pix. The reason its very handy for are:

- It has 15488 unique images for RGB and Thermal as pairs which in total makes 30976 which is high number of images with it being in pairs for pair requiring algorithms like pix2pix.
- Its images are mostly taken on evening and night on a active but not overly crowded street which are the ideal conditions for training an object detection model for detecting on low light for RGB and also very good to get most ideal results for detecting people on thermal level.
- Paper of dataset has accuracy rates which were useful what we have at start to improve or adapt for our project.

  3 example pairs below:



For second term We started with literature search for GAN models to implement for our project. Why we moved from image detection to image generation was because getting our own thermal data with a camera was unfeasible due to monetary costs and we wanted to improve our dataset with new unique images under low light that don't have thermal pair. GANs which is short for Generative Adversarial Networks are great tool for creating new images with style B from scratch using style A images. We choose CycleGAN and Pix2pix for generating images which are similar in some senses but Pix2pix needs pair images to train it and CycleGAN does not. After using both models for generating new images, we have tried to use those for testing YOLOv7 to detect generated "people" (which some seems like just random white pixels) there.

We used PyTorch implementations for both cycle gan and pix2pix. For both models we used LLVIP dataset to both train and test them. We then had used YOLOv7 to do some detection for generated results to show how much can it detect a person from generated images. For generated image quality test we used PSNR code below:

```
import os
import torch
from piqa import psnr
import torchvision.transforms as transforms
from PIL import Image

# Define the folder containing the image pairs
folder_path = "compare/"

# Define the transforms to be applied to the images
preprocess = transforms.Compose([
    transforms.Resize((256, 256)),  # Resize the image to (256, 256)
    transforms.ToTensor()  # Convert the image to a PyTorch tensor
])

# Initialize variables to store the total PSNR  values
total_psnr = 0.0
```

```
# Loop through all image pairs in the folder
for filename in os.listdir(folder_path):
    if filename.endswith("_fake.jpg"):

        # Load the generated image and ground truth image into PyTorch tensors
        generated_image_path = os.path.join(folder_path, filename)
        ground_truth_image_path = os.path.join(folder_path, filename.replace("_fake", ""))
        print(Image.open(generated_image_path).mode)
        generated_image = preprocess(Image.open(generated_image_path)).unsqueeze(0)

        ground_truth_image = 
preprocess(Image.open(ground_truth_image_path)).unsqueeze(0)

        # Calculate the PSNR between the generated image and ground truth image
        kernel = torch.tensor([[[[0.0448, 0.0512, 0.0448],
                              [0.0512, 0.0625, 0.0512],
                              [0.0448, 0.0512, 0.0448]]]])

        val_psnr = psnr.psnr(generated_image, ground_truth_image, 
kernel).mean().item()
        # Print the PSNR  values for the current image pair
        print("Image pair:", filename.replace("_fake.jpg", ""), " PSNR:", psnr)


        # Add the PSNR values to the running totals
        total_psnr += val_psnr

# Calculate the average PSNR across all image pairs
num_pairs = len(os.listdir(folder_path)) / 2
avg_psnr = total_psnr / num_pairs

# Print the average PSNR values
print("Average PSNR:", avg_psnr)
```

After calculating results with both eye and PSNR similarity, Pix2pix seemed to outperform CycleGan by good margin. We will explain CycleGan and Pix2pix implementation in sub-headings below because we divided the tasks for researching and implementation of two models one by one which is just like term plan (Abdullah Atahan Türk - Pix2pix, Yuşa Zorlu CycleGan)

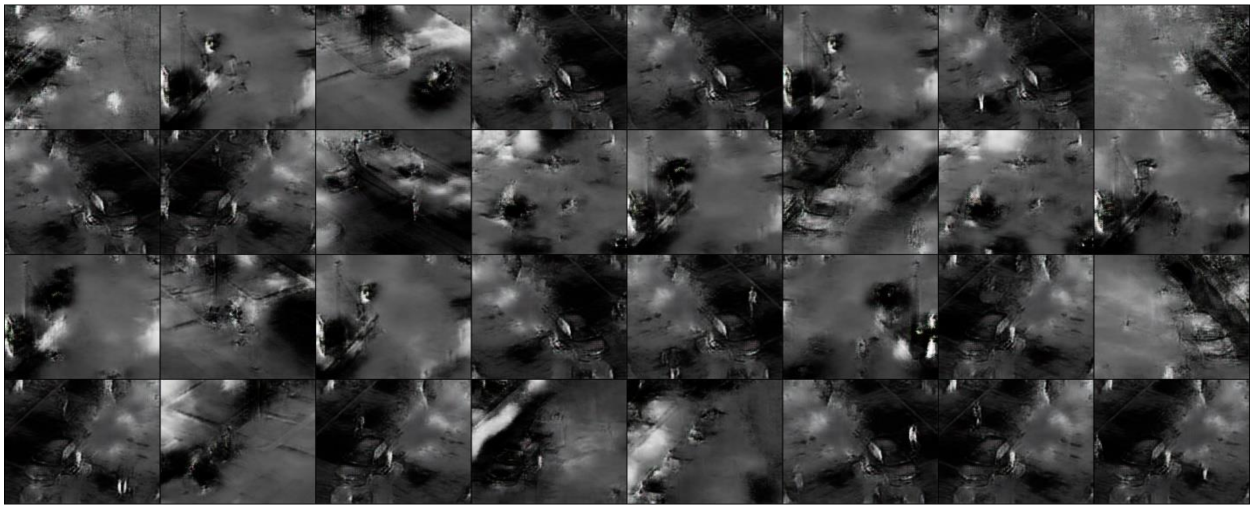
CycleGan: For cycleGan we created three files named “model.py” which is where generator and discriminator structure is stored. “train.py” where we trained our model and “test.py” where we tested our models with selected number of test images from LLVIP dataset which we didn’t trained our models with.

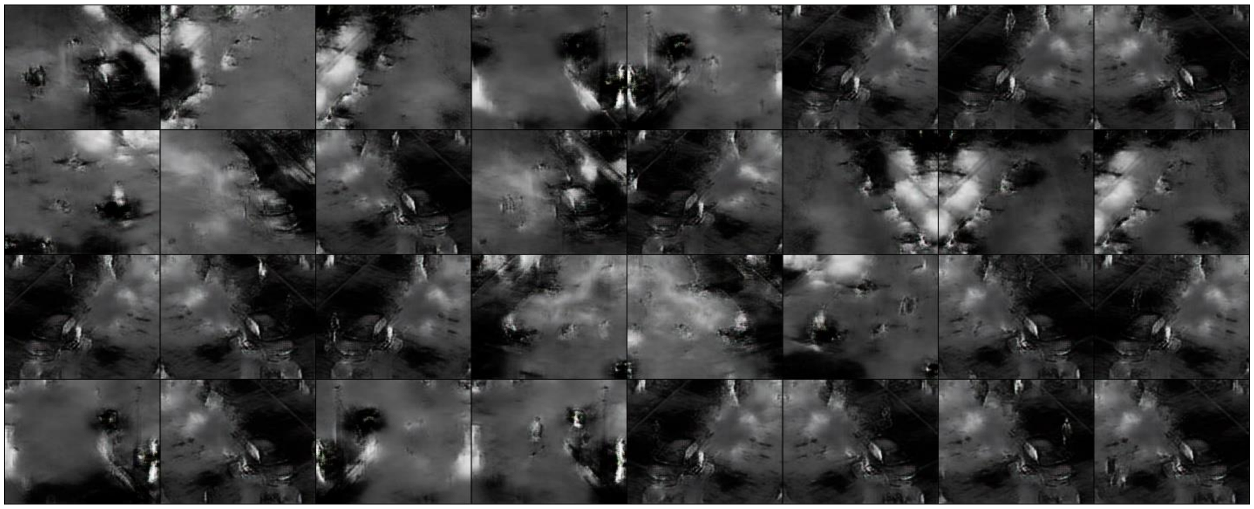
For generator and discriminator in model.py we used fairly simple implementation of what paper suggested because we lacked the processing time and power to implement more precise methodology.

For training configuration We used learning rates of lr_D = 0.00001 and lr_G = 0.00005 which D is for discriminator and G is generator. The 5 times difference is because of discriminator with same learning rate was catching generator too fast and 5 times difference at least gave some chance to improve itself. We also tried to implement improving discriminator for every two epochs instead of one but it did decreased image quality instead of increasing it. For image size for both test and train we used 256x256 image size which is the smallest size that can feasibly detect people from images.

For training time we decided 10 epochs is good enough to see our implementation is working or not. Batch size is decided to be 8 which was the highest that didn't cause problem on Google Colab and10 epochs with 1/3rd of LLVIP dataset gives us 4200*10 / 8 = 5500 ish batches which divided by 2 gives batch numbers for a to b and b to a.

For testing batch size is 32 by default to get less jpg files. But it can be changed to 1 to get single image file for each test input. Below there are some test outputs with default values for training and testing:





Pix2pix: For using pix2pix algorithm we used Google Colab, because pix2pix can only be using in Linux or OS X. For the experiments, we used an implementation that was faithful to what was written in the pix2pix paper. Pix2pix uses networks.GANLoss and torch.nn.L1Loss for loss functions in the generator. We also tried Perceptual Loss Function which is a function uses VGG16 to try to get better results but there is no significant change. Here is the sample code of the perceptal loss:

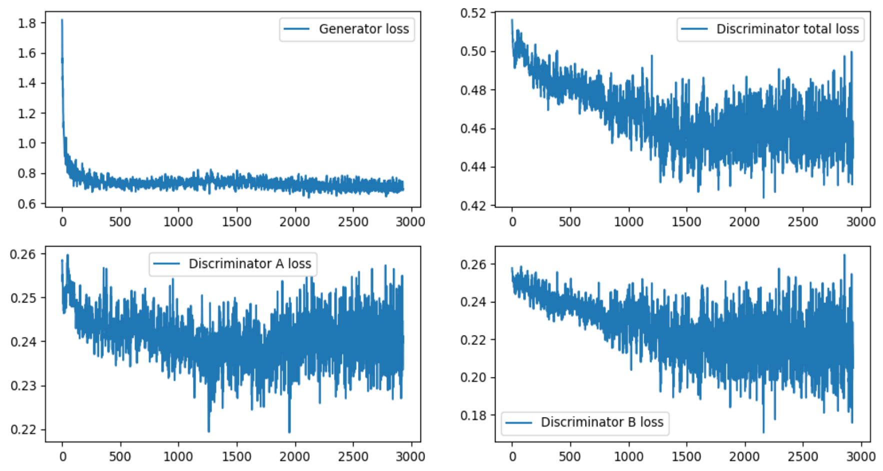
```
from torchvision.models import vgg16
class VGGPerceptualLoss(torch.nn.Module):
    def __init__(self):
        super().__init__()
        model = vgg16(pretrained=True, progress=False)
        self.feature_extractor = torch.nn.Sequential(*list(model.features)[:31]).eval() # till 4_3 layer
        for param in self.feature_extractor.parameters():
            param.requires_grad = False

    def forward(self, deconv_out, target):
        deconv_out_features = self.feature_extractor(deconv_out)
        target_features = self.feature_extractor(target)
        return F.l1_loss(deconv_out_features, target_features)
```

During pix2pix experiment we used 12025 image pairs LLVIP dataset, 32 batch size and 40 epochs. The learning rate of the algorithm was not static, it was optimized.

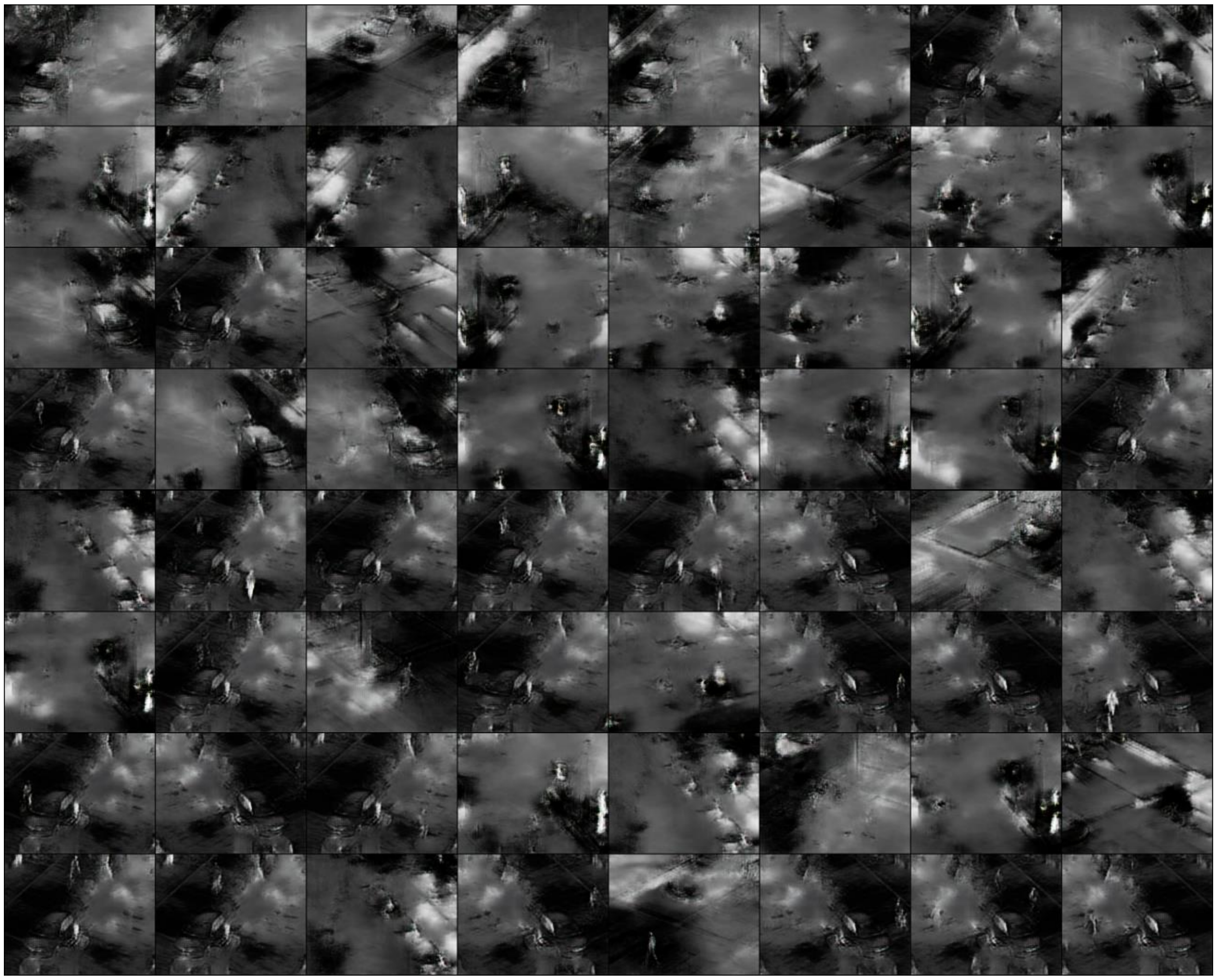
Results & Discussion (/ 30 Points)

For CycleGan our training results are looking like we have a problem that generator total loses are nearly always higher than discriminator loses which causes much less improvement than desired because when generator can't pass discriminator it can't know if it improves or not which in feedback loop makes also discriminator stale as it can be shown on table below. To resolve this problem we have tried to modify training process with saving on specific epochs and also with ratio of 5:1 generator should be better but it caused its own problems so we only implemented the ratio part for our code. Example image with suppressing discriminator for every second epoch is below which you can compare it with test results of default structure on git page.

Generator and Discriminator loss values for training default values below.



Test results of our cycleGan implementation for default values given on methodology part.



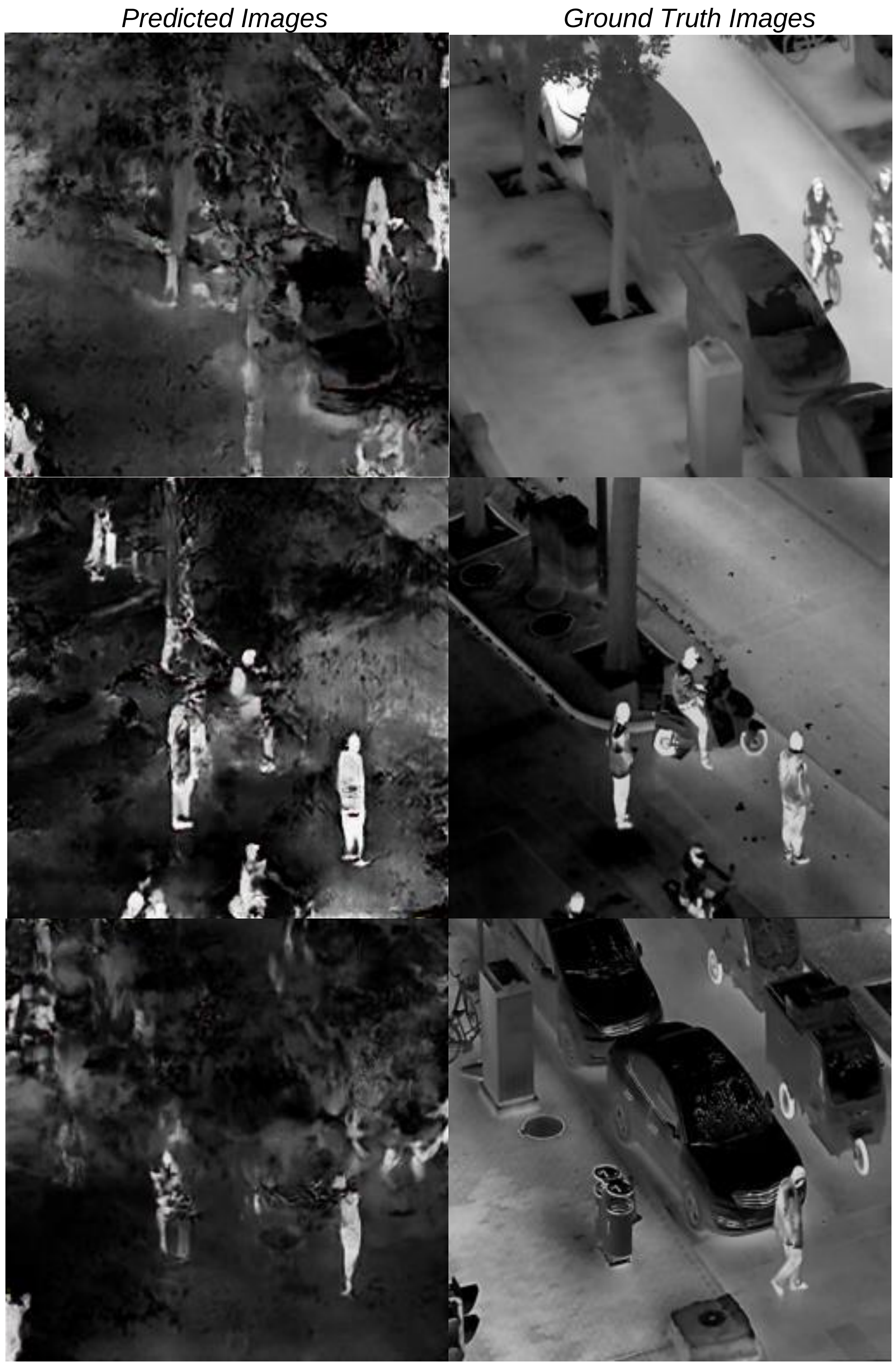
For Sota (state of the art) comparison, looking for generated images on LLVIP paper seems most logical for our project. There is a pix2pix generated image below to compare it against our results. In fact LLVIP dataset had 10.76 as PSNR score which We got 9.21 which is noticeably worse but for limited amount of training time its okay because cycleGan unlike pix2pix is un-supervised GAN model and our implementation of it was relatively simple than some codes that can be found on the internet.

They surely look like thermal images by color palette but for what is dark, what is light on thermal images aspect the cycleGan couldn't find out the characteristics of "glowing" objects under thermal camera. Some examples are below:

For pix2pix, firstly, since Linux or MacOs operating system is required to use the pix2pix algorithm, We conducted the experiments on Google Colab. Since we use a large dataset and the normal version of Colab can cause problems during training, we bought the Pro version. Before starting the training, we paired RGB and thermal images, since pix2pix works with 256x256 resolution image pairs.

Paired 256x256 image

We used 12025 image pairs during training with 40 epochs. At the end of the training, we tested on 3463 images and recorded all predicted images obtained by the pix2pix algorithm. Here are some predicted image results:



To be honest, we didn't get very heartwarming results, even if it wasn't too bad.
We think our dataset is large enough. But at least people on the thermal images are quite evident. Maybe better results can be achieved by increasing the number of epochs, but this poses a problem in Colab. we have 100 processing units per month in Colab Pro and as a result of this training and testing we have already spent 50 processing units.

We made measurements using some metrics for 3463 predicted images that we obtained with the pix2pix algorithm during the test phase. We used MSE, SSIM and PSNR. Since thermal images already have only black and white colors, we converted the images to grayscale during evaluation. Images have 8 bit depth.

PSNR:

PSNR (Peak Signal to Noise Ratio) is a commonly used metric for evaluating the quality of images in image processing and computer vision tasks. PSNR measures the difference between two images, typically the original image and the reconstructed image, and is commonly used to evaluate the quality of image generation models such as GANs and Pix2Pix.
In the context of Pix2Pix, PSNR can be used to evaluate the quality of the generated images relative to the ground truth images. The higher the PSNR value, the lower the distortion between the generated and ground truth images.
PSNR is calculated as follows:

$$PSNR = 20 * \log10(MAXp) - 10 * \log10(MSE)$$

where MAXp is the maximum possible pixel value of the image (for example, 255 for an 8-bit grayscale image), and MSE is the mean squared error between the generated and ground truth images.

PSNR Score: 27.91

From what we've researched online, values between 30 and 50 are generally considered good.
MSE:

MSE (Mean Squared Error) is a loss function used to measure the difference between two images, and is commonly used in image-to-image translation tasks such as pix2pix. The MSE value for pix2pix can vary widely depending on the specific dataset and problem being addressed. Lower values of MSE generally indicate better performance, as they indicate that the predicted image is closer to the ground truth image in terms of pixel values. Since we calculate the mse score with 8 bit depth grayscale images of 256x256 sizes, the value that mse can get is between 0 and 65025.

MSE Score: 5038.9814

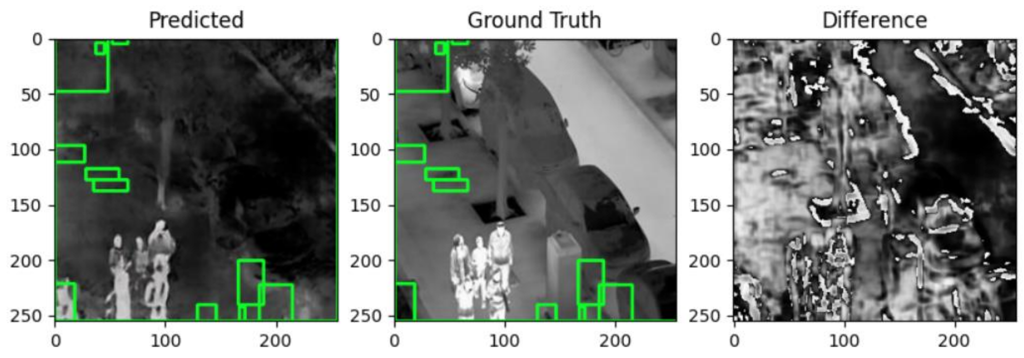
Actually, looking at the range, it seems like a not bad result, but as far as I've researched, it's actually not a good result.
SSIM:

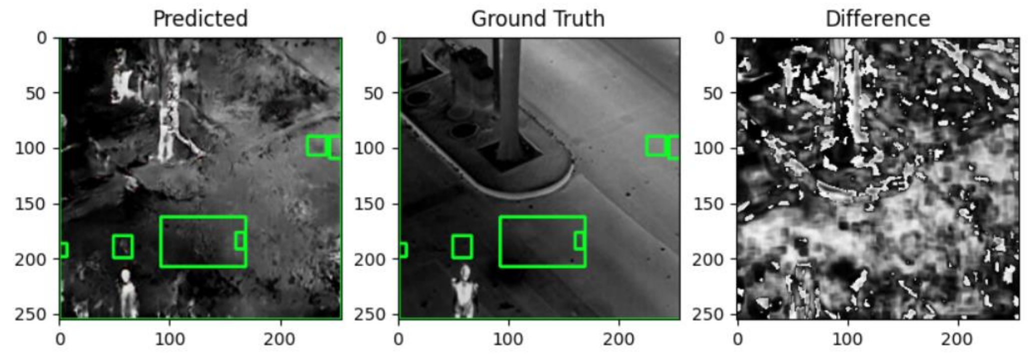
SSIM stands for Structural Similarity Index, which is a metric to measure the similarity between two images. In the context of image processing, it is commonly used to evaluate the performance of image restoration or image generation models, such as the pix2pix model. SSIM values range from -1 to 1, with 1 indicating a perfect match between two images and -1 indicating no similarity at all. Generally, a value greater than 0.9 is considered a good SSIM score for image restoration and generation tasks.

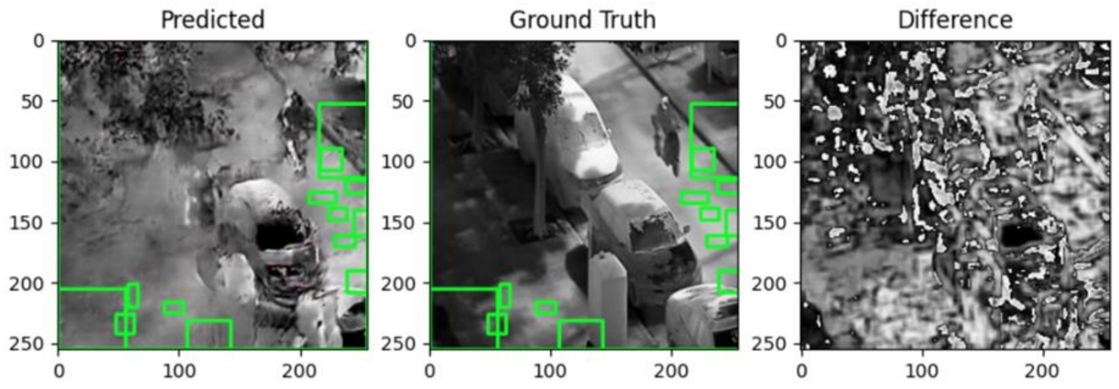
SSIM Score: 0.27157

So unfortunately, our SSIM score is bad.

We also made individual comparisons using SSIM on several images. By visualizing them, we revealed the differences and similarities between the two images. The green windows in the images show some obvious differences between the two images.



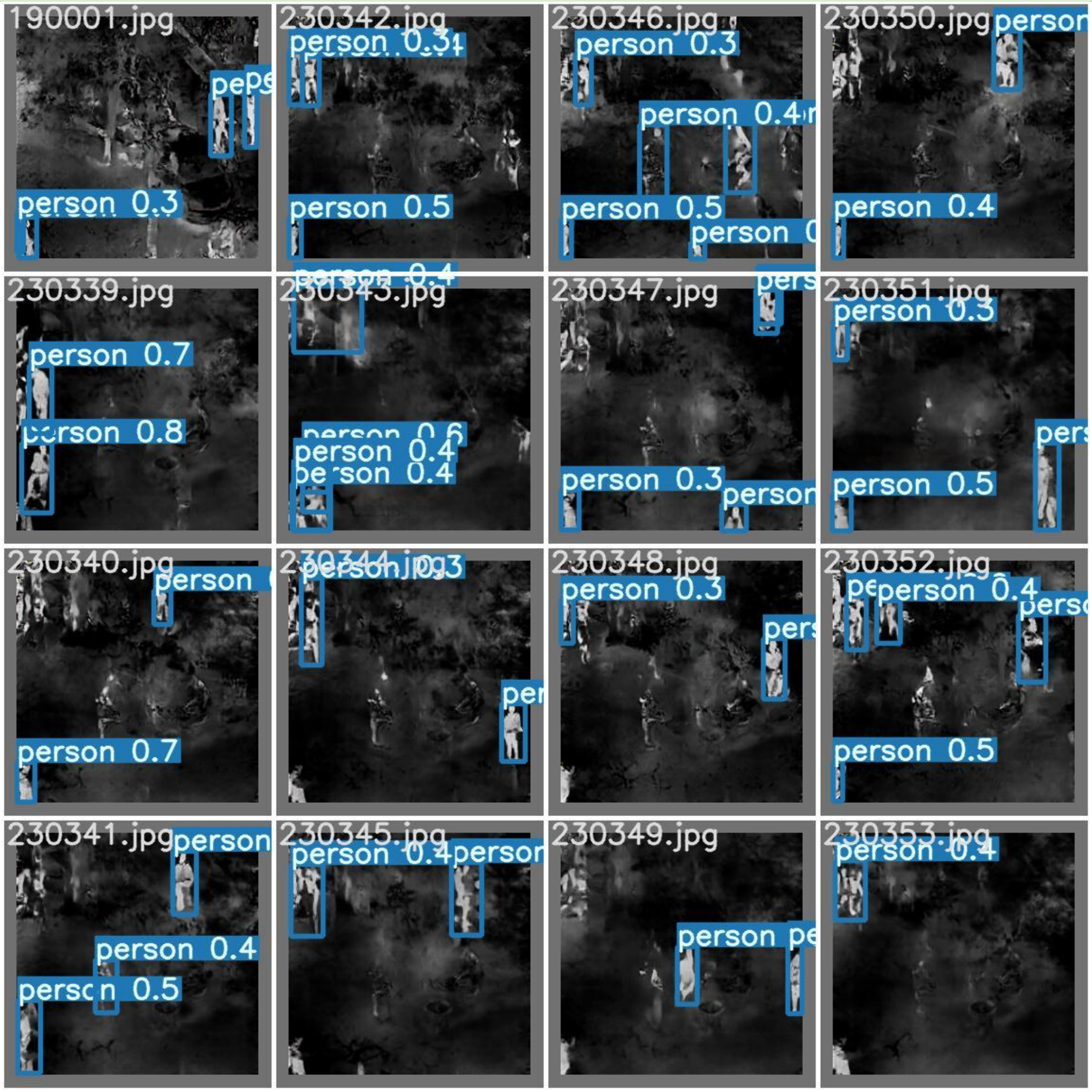




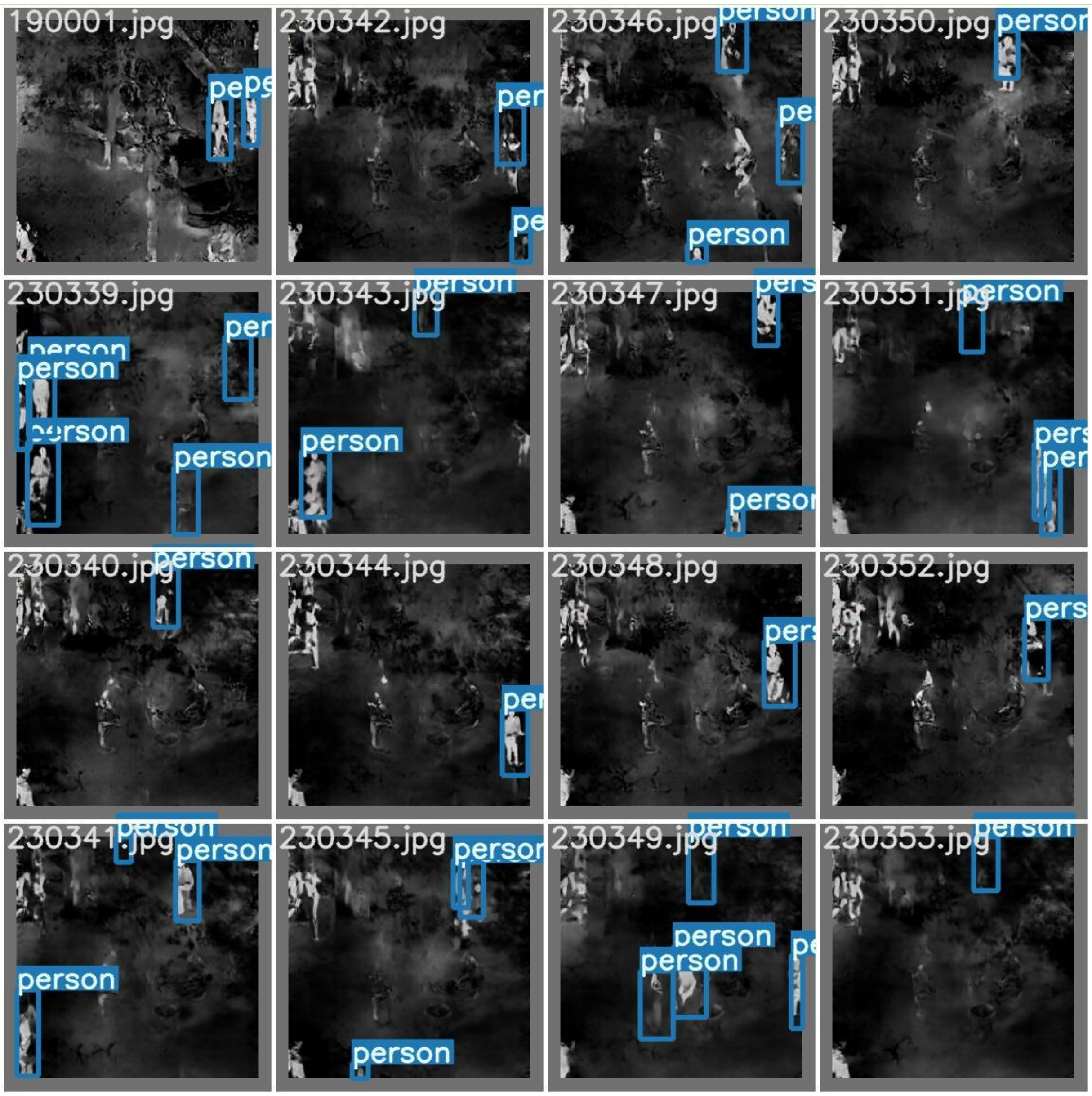
We thought it was a good indicator to do object detection using Yolov7 on the thermal images we created with pix2pix. Since pix2pix produces images in 256x256 sizes, the resolution of the images is low, which is a bit of a disadvantage for object detection. While doing object detection, we used the weight I obtained as a result of the training we made with the thermal images of LLVIP before. We have included the Yolov7 tests we have done on thermal images of LLVIP and FLIR before, in the table for comparison. If we compare it with the test on the thermal images of LLVIP, we got an extremely low result, but we can say that we got a result close to the test on the FLIR, which is somewhat promising. We also attach the images showing the object detection and actual labels by Yolov7 on the images produced by pix2pix.

METRICS ***DATASET***	**Precision**	**Recall**	**F1 Score**	**mAP@.5**	**mAP@.5:.95**
FLIR THERMAL	*0.581*	*0.495*	*0.38*	*0.382*	*0.159*
LLVIP THERMAL	*0.935*	*0.897*	*0.92*	*0.959*	*0.645*
Pix2pix THERMAL	*0.581*	*0.433*	*0.50*	*0.438*	*0.164*

Yolov7 scores of images



Yolov7 detections on pix2pix predicted images



Real labels of the images

The Impact and Future Directions (/ 15 Points)

Object detection, vital for various fields such as the defense industry and traffic safety, often faces challenges due to lighting conditions. Thermal cameras offer a solution as they work effectively in poor lighting, but their cost and complexities in handling thermal images raise concerns. This calls for a cost-effective solution like employing GANs (Generative Adversarial Networks), specifically image-to-image translation models like CycleGAN and Pix2Pix, to transform regular RGB images into thermal-like images.

Our proposed project, currently in its initial stages, suggests using these GANs to mimic thermal camera output. It's a more feasible solution, saving significant resources usually spent on thermal cameras and their associated data processing. The aim is to refine the output such that it becomes indistinguishable from actual thermal images.

The proposed project's potential success largely depends on better hardware and larger datasets. These enhancements can drive better training of the GAN models, leading to improved output image quality. It opens the possibility for further research, exploring more advanced GAN models that could potentially produce even clearer thermal image translations.

In essence, we're looking at a future where thermal camera costs can be significantly reduced or even eliminated. Such a development would greatly benefit areas requiring reliable object detection, like the defense industry or traffic safety initiatives. By converting RGB images to thermal images using advanced GAN models, we can achieve cost-effectiveness while maintaining, if not improving, the current standards of safety and security.

In other words, we think that this project has good potential and really useful results can be achieved if it receives the necessary financial support and focuses on it.