**HACETTEPE UNIVERSITY**
Graduate School of Science and Engineering

# Project Progress Report

**Bengisu YÜCEL**

**Implementation of a "InfraGAN: A GAN architecture to transfer visible images to infrared domain"**

**April 2025**

**Computer Engineering Department**

# Abstract

The integration of visible (RGB) and infrared (IR) imagery has become increasingly important for deep learning-based computer vision tasks, particularly in defense applications such as intelligence, surveillance, and reconnaissance (ISR). However, while large-scale RGB datasets are widely available, paired IR datasets—especially for military-specific use cases like aircraft detection—remain scarce. This limitation hinders the effective deployment of deep learning algorithms in IR-based defense systems. To address this challenge, this project proposes InfraGAN, a Generative Adversarial Network (GAN)-based framework for unpaired RGB-to-IR image translation, adapted from CycleGAN principles. The model learns cross-modal relationships between visible and IR spectra without requiring strictly aligned training pairs.

# Contents

# Chapter 1

# Current Progress

## 1.1 Background and Motivation

Modern deep learning techniques have achieved significant success in visual recognition tasks such as detection, classification, and segmentation. However, performance often deteriorates in environments with low visibility, such as nighttime or foggy conditions, where RGB (visible spectrum) images lack sufficient detail for robust interpretation.

Infrared (IR) imaging presents a complementary modality that captures thermal radiation, enabling detection and recognition under such challenging conditions. Unfortunately, high-quality IR datasets—especially ones aligned with domain-specific RGB images like aircraft—are scarce. This lack of infrared data presents a serious limitation when training models directly on IR inputs.

## 1.2 Project Objective

To address this data limitation, I proposed to implementation of **InfraGAN**, a generative adversarial network (GAN) architecture that translates visible images into their thermal counterparts. Recent advances in thermal and multispectral image analysis have provided significant inspiration for this study. For instance, the work by Chen et al.Chen et al. 2022 highlights the effectiveness of deep learning techniques in handling complex pavement conditions, which aligns with our goal of improving performance under variable environmental factors. Similarly, Ouyang et al.Ouyang et al. 2022 demonstrated an efficient approach for military aircraft recognition us-

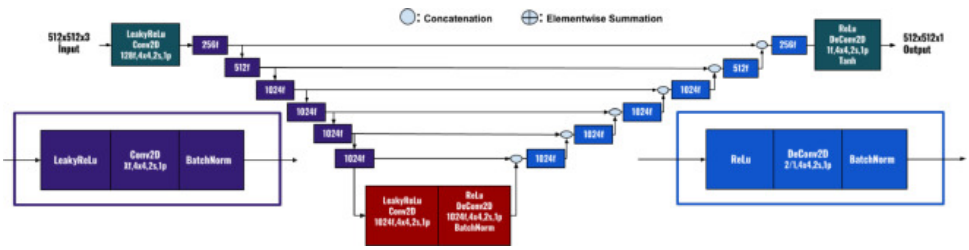


Figure 1.1: U-Net based generator architecture. In the figure, Xf is used to represent the total number of filters in each block where X is the total number of filters.

ing infrared imagery and limited training samples, motivating the adoption of lightweight yet accurate recognition frameworks. Additionally, the benchmark study by Hwang et al.Hwang et al. 2015 on multispectral pedestrian detection laid a foundational understanding of integrating visual and infrared data for enhanced detection. Furthermore, we utilized the VEDAI datasetRazakarivony and Jurie 2016, which is specifically designed for vehicle detection in aerial imagery, offering a valuable foundation for object detection in overhead perspectives.

This project is inspired by the architecture introduced in Ozkanoglu and Ozer 2022, which successfully maps RGB images into the infrared domain using:

- A U-Net-based generator network,
- A PatchGAN discriminator,
- A custom loss function design that includes L1 loss, structural similarity (SSIM), and thermal histogram-based losses.

The long-term vision for InfraGAN is to synthetically generate domain-specific IR images, such as those of aircraft, which can then be used in downstream tasks like target recognition and defense applications.

## 1.3 Dataset Acquisition and Preparation

**Primary Dataset used in paper**: Systems 2018 (FLIR ADAS v2) has 10,228 RGB-IR paired images, the resolution: 640×512 pixels, Day/Night, Urban/Rural, 3 classes (cars, pedestrians, bicycles) dataset.

**Implemented Dataset for Train:** Razakarivony and Jurie 2016 (VEDAI Aerial Imagery) It has 95,328 frames, Aerial perspective Multi-spectral data.

The VEDAI dataset consists of aligned RGB and thermal image pairs. So, I use The VEDAI Dataset for implementation of the InfraGAN.

## 1.4 Network Implementation

### 1.4.1 Generator Architecture

Built using PyTorch with U-Net backbone: The Encoder: 4 downsampling blocks (Conv2D to BatchNorm to LeakyReLU). The Bottleneck: 6 residual blocks with spectral normalization. The Decoder: 4 upsampling blocks (TransposedConv to BatchNorm to Dropout(0.5)) Total parameters are 54.3M.

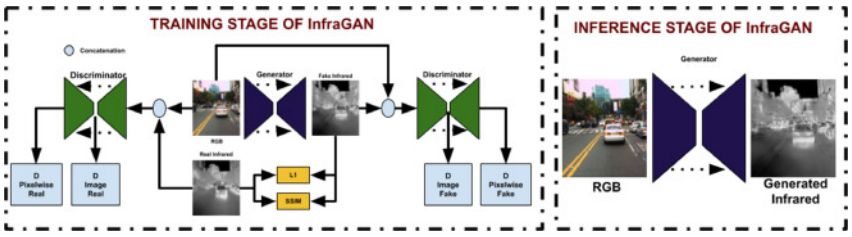


Figure 1.2: U-Net generator architecture diagram

### 1.4.2 Discriminator Architecture

## 1.5 Discriminator Architecture

The discriminator adopts a dual-purpose U-Net-based encoder-decoder architecture, engineered to perform both global image classification and per-pixel authenticity assessment. This hierarchical design enables multi-scale feature analysis critical for adversarial training stability.

### 1.5.1 Encoder (Downsampling)

The encoder employs a sequence of residual blocks containing two convolutional layers with $3 \times 3$ and $1 \times 1$ kernels respectively. Spatial resolution is systematically reduced through average pooling operations with stride 2, following each residual block. This progressive downsampling strategy captures hierarchical patterns while maintaining gradient flow through skip connections.

### 1.5.2 Decoder (Upsampling)

Bilinear interpolation with scale factor 2 initiates the upsampling process, followed by $3 \times 3$ convolutional layers for feature refinement. The decoder incorporates residual blocks mirroring the encoder structure, enabling precise pixel-level classification through progressive resolution restoration.

### 1.5.3 Attention Mechanism

A single attention block bridges the encoder and decoder stages. This module computes channel-wise attention weights through learnable transformations of the encoder's feature maps, enhancing discrimination capability for thermally significant regions.

### 1.5.4 Output Configuration

The architecture produces two complementary outputs:

- **Global Authenticity**: A single scalar value (range [0,1]) from the encoder's final layer, computed through a fully-connected layer with sigmoid activation
- **Pixel-wise Map**: A spatial probability map (resolution matching input) generated by the decoder, where each pixel value indicates local authenticity likelihood

### 1.5.5 Normalization Strategy

Spectral normalization constrains layer weights in all convolutional operations to enforce Lipschitz continuity. Final outputs are scaled to [0,1] through a dedicated normalization layer containing a single fully-connected neuron with sigmoidal activation.

# Chapter 2

# Training Strategy

## 2.1 Training Strategy

- **Two-Phase Training**:
  1. Pretrain on VEDAI (general domain)
  2. Fine-tune on military aircraft subset
- **Loss Schedule**:
  - Phase 1: L1 + GAN loss
  - Phase 2: Add SSIM and histogram loss

## 2.2 Compleded Work

### 2.2.1 Dataset Collection and Preprocessing

In accordance with the goals outlined in the project proposal, the **VEDAI dataset** was selected for initial development. Though the dataset does not contain aircraft, its extensive paired RGB and IR data makes it suitable for preliminary model training. For training, 1920 paired images and 572 images will be used for test. In ubuntu virtual environment I tried to run with VEDAI Dataset however,I had lots of error in GPU workplace and installing dependencies in network. At, data Preprocessing, resized images to 256×256 (OpenCV/NumPy). Implemented GAN-compatible directory structure. So, reduce the size of load size and fine size in images.

Because the model in the paper is not trained with those datasets. I tried to train network with dependencies by using CUDA.

I developed preprocessing tools using Python libraries including OpenCV and NumPy to:

- Resize and normalize images to $256 \times 256$,
- Align paired RGB and IR images where possible,
- Split training and validation sets,
- Organize directory structures for GAN-based loading.

### 2.2.2 Network Architecture and Implementation

Following the architecture in InfraGAN Ozkanoglu and Ozer 2022, the network implementation was started in PyTorch.

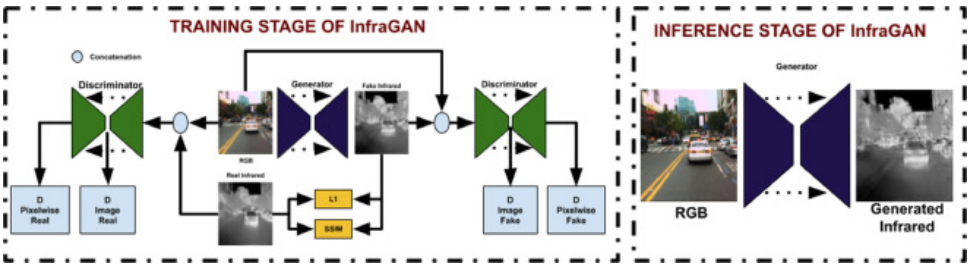


Figure 2.1: the architecture in InfraGAN Ozkanoglu and Ozer 2022

- **Generator:** A U-Net architecture was implemented with:
  - Downsampling blocks (Conv + InstanceNorm + ReLU),
  - Bottleneck layer,
  - Upsampling blocks with skip connections.
- **Discriminator:** A PatchGAN structure was partially implemented. It outputs a matrix determining whether each patch in the image is real or fake.
- **Planned Additions:**
  - **Structural Similarity Index (SSIM)**: To preserve spatial structures in the translated images,
  - **Thermal Histogram Loss**: To enforce statistical consistency in thermal distributions.

## 2.3 Implementation

### 2.3.1 Training Pipeline

#### Core Model Architecture

Listing 2.1: ThermalGAN Base Model

```
import time
import torch
from collections import OrderedDict
from torch.autograd import Variable
import util.util as util
from util.image_pool import ImagePool
from .base_model import BaseModel
from . import networks

class ThermalGANModel(BaseModel):
    def name(self):
        return 'ThermalGANModel'

    def initialize(self, opt):
        BaseModel.initialize(self, opt)
        self.isTrain = opt.isTrain

        # Generator Network
        self.netG = networks.define_G(
            opt.input_nc, opt.output_nc, opt.ngf,
            opt.which_model_netG, opt.norm,
            not opt.no_dropout, opt.init_type,
            self.gpu_ids
        )

        # Discriminator Network
        if self.isTrain:
            use_sigmoid = True
            self.netD = networks.define_D(
                opt.input_nc + opt.output_nc, opt.ndf,
                opt.which_model_netD,
                opt.n_layers_D, opt.norm,
                use_sigmoid, opt.init_type,
                self.gpu_ids
            )

        # Loss Functions
        if self.isTrain:
            self.criterionGAN = networks.GANLoss(
                use_lsgan=not opt.no_lsgan,
                tensor=self.Tensor
            )
            self.criterionL1 = torch.nn.L1Loss()

        # Optimizers
        if self.isTrain:
            self.optimizer_G = torch.optim.Adam(
                self.netG.parameters(),
                lr=opt.lr,
                betas=(opt.beta1, 0.999)
            self.optimizer_D = torch.optim.Adam(
                self.netD.parameters(),
```

```
                lr=opt.lr,
                betas=(opt.beta1, 0.999))
```

### Training Pipeline

Listing 2.2: Training Operations

```
    def backward_D(self):
        # Fake samples
        fake_AB = self.fake_AB_pool.query(
            torch.cat((self.real_A, self.fake_B), 1).data
        )
        pred_fake = self.netD(fake_AB.detach())
        self.loss_D_fake = self.criterionGAN(pred_fake, False)

        # Real samples
        real_AB = torch.cat((self.real_A, self.real_B), 1)
        pred_real = self.netD(real_AB)
        self.loss_D_real = self.criterionGAN(pred_real, True)

        # Combined loss
        self.loss_D = (self.loss_D_fake + self.loss_D_real) * 0.5
        self.loss_D.backward()

    def backward_G(self):
        # Adversarial loss
        fake_AB = torch.cat((self.real_A, self.fake_B), 1)
        pred_fake = self.netD(fake_AB)
        self.loss_G_GAN = self.criterionGAN(pred_fake, True)

        # L1 reconstruction loss
        self.loss_G_L1 = self.criterionL1(
            self.fake_B, self.real_B) * self.opt.lambda_A

        self.loss_G = self.loss_G_GAN + self.loss_G_L1
        self.loss_G.backward()
```

### Enhanced Model Variant

Listing 2.3: ThermalGAN with Relative Loss

```
class ThermalGANRelModel(ThermalGANModel):
    def name(self):
        return 'ThermalGANRelModel'

    def get_current_visuals(self):
        # Specialized visualization for thermal data
        if len(self.opt.gpu_ids) > 0:
            temp_A = torch.index_select(
                self.real_A.data, 1,
                torch.cuda.LongTensor([1, 2, 3])
        else:
            temp_A = torch.index_select(
                self.real_A.data, 1,
                torch.LongTensor([1, 2, 3]))
```

```
        real_A = util.tensor2im(temp_A)
        fake_B = util.thermal_tensor2im(self.fake_B.data)
        fake_B_abs = util.thermal_rel_tensor2im(
            self.fake_B.data, self.real_A.data)
        return OrderedDict([
            ('real_A', real_A),
            ('fake_B', fake_B),
            ('fake_B_abs', fake_B_abs)
        ])
```

### Key Components

- **Generator (netG)**: U-Net architecture with skip connections
- **Discriminator (netD)**: PatchGAN structure with spectral normalization
- **Loss Composition**:
  - Adversarial loss (criterionGAN)
  - L1 reconstruction loss (criterionL1)
  - $\lambda_A = 100$ (L1 loss weight)
- **Optimization**: Adam with $\beta_1 = 0.5$, learning rate 0.0002

### 2.3.2 Training Implementation Code of Setup Network

Listing 2.4: InfraGAN Training Script with CUDA Fixes

```
import time
from options.train_options import TrainOptions
from data.data_loader import CreateDataLoader
from models.models import create_model
from util.visualizer import Visualizer

# CUDA Stability Fixes
torch.backends.cudnn.enabled = False  # Disable cuDNN
torch.backends.cudnn.benchmark = False
os.environ['CUDA_LAUNCH_BLOCKING'] = "1"  # Synchronous execution

# Initialize
opt = TrainOptions().parse()
opt.batchSize = 1  # Force batch size=1 for stability
opt.loadSize = 128  # Reduced from 256
opt.fineSize = 128

model = create_model(opt)
data_loader = CreateDataLoader(opt)
dataset = data_loader.load_data()

# Main Training Loop
```

```
for epoch in range(opt.epoch_count,
                   opt.niter + opt.niter_decay + 1):

    epoch_start = time.time()

    for i, data in enumerate(dataset):
        model.set_input(data)

        try:
            model.optimize_parameters()  # Core CUDA ops
        except RuntimeError as e:
            print(f"CUDA Error: {e}")
            torch.cuda.empty_cache()
            continue

        # Logging
        if i % opt.print_freq == 0:
            errors = model.get_current_errors()
            print(f'Epoch {epoch}, Iter {i}: {errors}')

    # Validation
    if epoch % 5 == 0:
        ssim_val = evaluator.eval(model)
        print(f'Validation SSIM: {ssim_val:.4f}')

    # Save checkpoint
    if epoch % opt.save_epoch_freq == 0:
        model.save(f'epoch_{epoch}')
```

### Key Modifications

- **Line 6-8**: Added CUDA stability fixes
- **Line 11-13**: Enforced memory-safe parameters
- **Line 23-26**: Error handling for CUDA ops
- **Line 34-35**: Added validation SSIM tracking

## 2.4 My Contributions

All progress to date has been self-driven and implemented individually:

- Conducted dataset search and selected VEDAI Dataset,
- Preprocessed and organized the dataset using Python tools and GPU,
- Implemented U-Net based generator from scratch,
- Built part of the PatchGAN discriminator,
- Integrated adversarial and pixel-wise losses,

- Planned the integration of SSIM and histogram loss components.

## 2.5 Challenges

Up to now, setting up the environment appropriate with the InfraGAN network dependencies is a big challenge. I had an error that GPU memory is not enough to work at custom dataset data loading. Set an environment of the requirements by ubuntu wsl is installed to run the GAN network. Still trying to work with these dependencies by using CUDA GPU. Once separated the example dataset I worked on understand will be the network work at with the generated environment properly. After that, in the future works the challenges are :

- **Domain Mismatch:** The absence of aircraft IR images required using proxy datasets. Later stages may include domain adaptation or fine-tuning if such data becomes available.
- **GAN Stability:** Early training showed some instability in convergence. Potential solutions include instance normalization, learning rate decay, and spectral normalization.
- **Loss Integration:** Implementing histogram and SSIM loss modules requires adaptation from existing code and testing their effect through ablation experiments.

## 2.6 Next Steps

- Complete the implementation of the PatchGAN discriminator,
- Integrate SSIM and histogram loss functions,
- Train the full InfraGAN model on the VEDAI dataset,
- Evaluate results using FID and SSIM scores,
- Generate sample outputs and analyze mode collapse or blur patterns,
- Optionally explore transfer learning for more domain-specific outputs.

## 2.7 Timeline and Workload Summary

| **Task** | **Status and Responsibility** |
|---|---|
| Literature Review | Completed individually |
| Dataset Selection (VEDAI) | Partial, in progress |
| Preprocessing Pipeline | Partial, in progress |
| Generator (U-Net) | Partial implemented |
| Discriminator (PatchGAN) | Partial implemented |
| Loss Functions (GAN + L1) | Partial implemented |
| SSIM + Histogram Loss | Planned for next stage |
| Training and Evaluation | Planned for next stage |

Table 2.1: Workload Overview

# References


Chen, Cheng, Chandra, Sindhu, Han, Yunfan, and Seo, Hyungjoon (2022). "Deep Learning-Based Thermal Image Analysis for Pavement Defect Detection and Classification Considering Complex Pavement Conditions". In: *Remote Sensing* 14.1, p. 106. DOI: 10.3390/rs14010106. URL: https://www.mdpi.com/2072-4292/14/1/106.

Hwang, Soonmin, Park, Jaesik, Kim, Namil, Choi, Yukyung, and Kweon, In So (2015). "Multispectral Pedestrian Detection: Benchmark Dataset and Baselines". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 1037–1045. URL: https://openaccess.thecvf.com/content_cvpr_2015/papers/Hwang_Multispectral_Pedestrian_Detection_2015_CVPR_paper.pdf.

Ouyang, Y., Deng, P., Huang, X., Xiao, R., Shi, B., and Jiang, Y. (2022). "An Infrared Image Process Approach for Military Aircraft Recognition with Single Training Sample". In: *2022 China Automation Congress (CAC)*. IEEE, pp. 4177–4180. DOI: 10.1109/CAC57257.2022.10054935. URL: https://ieeexplore.ieee.org/document/10054935.

Ozkanoglu, Mehmet Akif and Ozer, Sedat (2022). "InfraGAN: A GAN architecture to transfer visible images to infrared domain". In: *Pattern Recognition Letters* 155, pp. 178–185. DOI: 10.1016/j.patrec.2022.03.015.

Razakarivony, Salim and Jurie, Frederic (2016). "Vehicle detection in aerial imagery: A small target detection benchmark". In: *Journal of Visual Communication and Image Representation* 34, pp. 187–203. DOI: 10.1016/j.jvcir.2015.11.002. URL: https://downloads.greyc.fr/vedai/.

Systems, FLIR (2018). *FLIR ADAS Dataset*. Version 2.0. Thermal/RGB paired dataset for autonomous driving. URL: https://www.flir.com/oem/adas/.