HAW HAMBURG

BACHELORTHESIS
Seifeldin Abdelwahab

Realistic Shadow Generation for Image Composition Using Deep Learning

FACULTY OF COMPUTER SCIENCE AND ENGINEERING
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

HAMBURG UNIVERSITY
OF APPLIED SCIENCES

Hochschule für Angewandte
Wissenschaften Hamburg

Seifeldin Abdelwahab

Realistic Shadow Generation for Image Composition Using Deep Learning

Bachelor Thesis based on the examination and study regulations
for the Bachelor of Engineering degree programme
Bachelor of Science Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg
Supervising examiner: Prof. Dr. Ulrike Herster
Second examiner: Prof. Dr. Marc Hensel

Day of delivery: 06. June 2025

Seifeldin Abdelwahab

Title of Thesis

Realistic Shadow Generation for Image Composition Using Deep Learning

Keywords

Artificial Intelligence, Deep Learning, Computer Vision, Shadow generation, Image composition

Abstract

Realistic shadows are key to making virtual objects appear naturally integrated into real-world scenes. However, many deep learning models fall short when handling high-resolution images, often producing inconsistent or low-quality results. This thesis proposes RRSGNet(Resolution-Robust Shadow Generation Network), which introduces a resolution-robust solution by using Fast Fourier Convolutions, allowing the model to adapt to different image sizes without sacrificing visual quality. Experiments show that RRSGNet generates consistent and high-quality shadows across a wide range of resolutions, which makes it well suited for applications like augmented reality and visual effects.

Seifeldin Abdelwahab

Thema der Arbeit

Realistische Schattengenerierung für Bildkomposition mit Deep Learning

Stichworte

Künstliche Intelligenz, Deep Learning, Computer Vision, Schattengenerierung, Bildkomposition

Kurzzusammenfassung

Realistische Schatten sind entscheidend, damit virtuelle Objekte glaubwürdig in reale Szenen integriert wirken. Viele Deep-Learning-Modelle stoßen dabei an ihre Grenzen, insbesondere bei hochauflösenden Bildern, und erzeugen häufig uneinheitliche oder qualitativ minderwertige Ergebnisse. RRSGNet bietet eine auflösungsrobuste Lösung durch den Einsatz von Fast Fourier Convolutions, wodurch das Modell flexibel mit unterschiedlich großen Bildern arbeiten kann, ohne an Bildqualität zu verlieren. Zusätzlich werden kontextbezogene Informationen wie Hintergrund- und Schattenbereiche automatisch erkannt, was den manuellen Aufwand reduziert und die Benutzerfreundlichkeit erhöht. Experimente zeigen, dass RRSGNet über verschiedene Auflösungen hinweg konsistente und hochwertige Schatten erzeugt und sich somit ideal für Anwendungen wie Augmented Reality und visuelle Effekte eignet.

List of Abbreviations

AI Artificial Intelligence

AR Augmented Reality

BCE Binary Cross-Entropy

BER Balanced Error Rate

CIoU Complete Intersection over Union

CNN Convolutional Neural Network

CPU Central Processing Unit

DESOBA DEshadowed Shadow-OBject Association (dataset)

DMASNet Decomposed Mask Prediction and Attentive Shadow Filling Network

FC Fully Connected (Layer)

FCN Fully Convolutional Network

FFC Fast Fourier Convolution

FFT Fast Fourier Transform

FID Fréchet Inception Distance

FPN Feature Pyramid Network

FR Functional Requirement

FU Fourier Unit

GAN Generative Adversarial Network

GPU Graphics Processing Unit

HD High Definition

IoU Intersection over Union

LaMa Large Mask Inpainting (model)

LFU Local Fourier Unit

LPIPS Learned Perceptual Image Patch Similarity

Mbo Mask of background objects

Mbs Mask of background shadows

Mfo Mask of foreground object

Mfs Mask of foreground shadow

NFR Non-Functional Requirement

PSNR Peak Signal-to-Noise Ratio

RdSOBA Rendered Shadow-Object Association (dataset)

ReLU Rectified Linear Unit

ResNet Residual Network

RGB Red, Green, Blue

RMSE Root Mean Square Error

RRSGNet Resolution-Robust Shadow Generation Network

SE Squeeze-and-Excitation

SGRNet Shadow Generation and Refinement Network

SMART Specific, Measurable, Achievable, Relevant, Time-bound

SOBA Shadow-Object Association (dataset)

SSIM Structural Similarity Index Measure

SSIS Single-Stage Instance Shadow Detection

TPU Tensor Processing Unit

UC Use Case

VGG Visual Geometry Group (network architecture)

VRAM Video Random Access Memory

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Image composition, the process of inserting a foreground object copied from an image or a virtual object into a background image or scene, is a fundamental operation with wide-ranging applications. It is used in creative arts, film postproduction and visual effects, advertising, virtual product placement, data augmentation for training computer vision models, and emerging fields such as augmented and virtual reality [54, 52]. The primary objective of image composition is typically photorealism – creating a final image where the inserted object appears naturally integrated within the background scene [31]. However, simply pasting the object in the background scene often leads to visual inconsistencies, like the difference in illumination between object and background scene [31], lack of foreground shadow or reflection [15, 44] and so on, making it look weird and non-realistic to the eye.

The accurate rendering of shadows cast by the inserted foreground object is particularly critical for achieving a convincing composite image[15, 44]. Shadows are essential for blending objects within their environment, as they provide clues about the shape of surfaces, and provide information about the lighting [32]. An image lacking plausible shadows, or one where the shadow's characteristics (e.g., softness, direction, color) are inconsistent with the background scene, immediately appears artificial and non-believable [15].

Creating these shadows automatically is a challenge. The appearance of a shadow depends on many complex factors: the 3D shape of the object that casts the shadow, the shape of the surface the shadow falls on, and the properties of the light source(s) [54].Traditional computer graphics methods can render physically accurate shadows but require explicit 3D scene models and lighting setups, which are rarely available [32]. Although manual shadow creation by digital artists is a common practice, it is a very time-consuming process that requires a lot of skill and effort.

Deep learning techniques have recently shown great potential in synthesizing visual elements, including shadows, directly from image data without the need to explicit 3D models or lightning information. These methods aim to learn the complex mapping from scene appearance to shadow appearance implicitly. [28, 29, 44, 15, 54]. These methods try to learn a connection between how a scene looks and how its shadows should look. However, state-of-the-art shadow generation models often face practical limitations. A key issue, seen in the recent DMASNet model [44] and many others, is the restriction to fixed input resolutions (typically 256x256 pixels). This means high-resolution images, that are most common today, have to be down sampled, limiting the final shadow detail and making them less suitable for professional applications using high-resolution imagery. Furthermore, DMASNet and similar approaches [15] frequently require additional manually annotated inputs beyond the composite image and foreground object mask. Specifically, they often need masks that identify background objects (Mbo) and existing background shadows (Mbs) [44, 15]. While these masks help guide the neural network, they are not commonly available in real-world scenarios, limiting automation and ease of use.
This thesis is motivated bythe need to address these specific limitations in current deep learning-based shadow generation. We aim to develop a deep learning approach for realistic foreground shadow generation that is (1) capable of operating on variable input resolutions without being restricted to a fixed training size, and (2) reduces the reliance on hard-to-obtain input masks like Mbo and Mbs by automating their prediction within the pipeline. This moves towards a more flexible, automated, and broadly applicable solution for realistic image composition.

1.2 Company & Project Overview

This thesis project is carried out in collaboration with portrix.net GmbH. The project focuses on the use of a drone of some sort to capture high-resolution images or videos of landscape areas. The core objective is to augment these scenes by digitally inserting a virtual object or building and, crucially, to cast realistic shadows of the inserted object into the captured environment.

Background

portrix.net GmbH is an independent software vendor known for developing high-performance and scalable software solutions tailored to corporate customers. Since its establishment in 2001, the company has emphasized process standardization to guarantee quality and reliability in every project.

Core Services

- **Custom Software Development:** Delivering tailor-made solutions for diverse client needs.
- **IT Consulting and Project Management:** Providing comprehensive requirement analysis and strategic oversight.
- **AI-Driven Innovations:** Recently, the company has also started to utilize artificial intelligence (AI) to further enhance its processes and offerings.

1.3 Initial Situation

Recent advancements in deep learning have led to several promising approaches to automate shadow generation in image composition. Models like ShadowGAN [54] and AR-ShadowGAN [27] utilize generative adversarial networks to synthesize shadows. More recently, SGRNet [15] introduced a two-stage approach involving mask prediction and illumination-based filling, while DMASNet [44] further refined mask prediction using decomposition and employed attentive filling strategies, achieving state-of-the-art results on standard benchmarks.

Despite this progress, significant limitations hinder the practical applicability of these methods. Critically, DMASNet, which represents one of the current state-of-the-art methods and serves as a baseline for this work, is restricted to operating on fixed resolution inputs (256x256 pixels) [44]. This restricts its use with common high-resolution images. Furthermore, DMASNet requires pre-defined background object (Mbo) and background shadow (Mbs) masks as essential inputs to guide its network.[44]. The need for

these auxiliary masks complicates the workflow and prevents full automation. These specific constraints regarding resolution flexibility and input requirements form the primary motive for the novel approach developed in this thesis.

1.4 Goals and Contributions

The primary objectives of this thesis are defined using the SMART (Specific, Measurable, Achievable, Relevant, Time-bound) criteria:

- **Goal 1: Replication and Baseline Establishment**
  - **S (Specific):** Replicate the DMASNet architecture and training procedure as described in the original paper [44], including its decomposed mask prediction and attentive shadow filling stages, using the DESOBA and RdSOBA datasets.
  - **M (Measurable):** Achieve numerical performance metrics (RMSE, PSNR, BER) on the DESOBA test set that are comparable to those reported in the DMASNet paper. The visual appearance of generated shadows should resemble the shadow quality and characteristics presented in the DMASNet paper [44].
  - **A (Achievable):** The DMASNet paper provides sufficient architectural and training details. Standard deep learning frameworks (PyTorch/TensorFlow) and datasets (DESOBA/RdSOBA) are accessible.
  - **R (Relevant):** This replication is crucial for deeply understanding the model, identifying its practical limitations, and establishing a verified performance baseline to evaluate the proposed novel model against.
  - **T (Time-bound):** Complete the implementation, training, and evaluation for the DMASNet replication the 31st of March, 2025
- **Goal 2: Design and Implementation of RRSGNet for Variable Resolution**

- **S (Specific):** Design and implement a novel shadow generation network that incorporates Fast Fourier Convolutions (FFCs), inspired by LaMa [42], into a DMASNet-like architecture to enable processing of variable input image resolutions.
- **M (Measurable):** RRSGNet must successfully process images of varying input resolutions (e.g., ranging from 128x128, 256x256, 512x512, up to 1920x1080) without requiring architectural changes or retraining. producing visually plausible shadows assessed on DESOBA test images across different resolutions mentioned above. Performance on fixed-size (256x256) DESOBA benchmarks should be the same with the replicated DMASNet on key metrics.
- **A (Achievable):** FFC principles are documented [42, 6], and the basic blocks are implemented and available. The core logic builds upon the replicated DMASNet.
- **R (Relevant):** Directly addresses the critical limitation of fixed resolution identified in DMASNet, significantly increasing the practicality of the shadow generation model.
- **T (Time-bound):** Complete the design, implementation, and initial training/testing of the variable-resolution RRSGNet module by the 30th of April 2025.

- **Goal 3: Integration of Automated Background Mask Prediction**
  - **S (Specific):** Integrate a pre-trained object/shadow detection model based on SSIS [49] into the RRSGNet pipeline to automatically predict background object (Mbo) and background shadow (Mbs) masks, removing the need for manual input.
  - **M (Measurable):** RRSGNet, using *predicted* Mbo/Mbs masks, should generate shadows that achieve quantitative scores on the DESOBA 256x256 test set that are the same as RRSGNet performance using ground-truth Mbo/Mbs masks. Visual assessment on test images should show successful shadow generation guided by the predicted masks.
  - **A (Achievable):** Pre-trained models for related detection tasks exist. Integration involves adding a pre-processing step to the pipeline to feed the RRSGNet shadow generation module.

  - **R (Relevant):** Addresses the second major limitation of DMASNet (manual Mbo/Mbs input), further enhancing the automation and practicality of the proposed system.
  - **T (Time-bound):** Complete the integration and testing of the automated mask prediction pipeline by the 10th of May 2025

- **Goal 4: Visual Evaluation and User Preference Study**
  - **S (Specific):** Conduct a user study comparing the visual realism of shadows generated by the final RRSGNet model against the replicated DMASNet and other baselines on a set of real composite images.
  - **M (Measurable):** Show through user testing that RRSGNet's shadows are chosen as more realistic compared to other baselines like DMASnet, SGRnet, ARShadowGAN, etc. Collect preference data from at least 20 participants on 10 real composite image examples. RRSGNet should receive a clear majority of preference votes.
  - **A (Achievable):** Using an existing online survey tool, in particular google forms.
  - **R (Relevant):** Provides crucial validation of the practical effectiveness and perceived quality improvement of the proposed model, by assessing visual realism as perceived by humans.
  - **T (Time-bound):** Design, execute, and analyze the user study by 30th of May 2025.

Based on these goals, the main contributions of this work are:

1. A successful replication and analysis of the DMASNet model, providing a verified baseline.
2. The design and implementation of RRSGNet, a novel shadow generation architecture incorporating FFCs for robust handling of variable image resolutions.
3. The integration of an automated background mask prediction module into the RRSGNet pipeline, reducing manual input requirements.

4. Comprehensive quantitative and visual evaluation, including a user study, demonstrating the effectiveness and improved practicality of the proposed RRSGNet compared to the other baselines.

1.5 Thesis Outline

Chapter 1 covers the problem of realistic shadow generation and project goals. Chapter 2 outlines requirements and stakeholder needs. Chapter 3 reviews background on image composition, shadow techniques, DMASNet and Deep learing fundamentals. Chapter 4 introduces the RRSGNet method with Fast Fourier Convolutions and mask automation. Chapter 5 details the implementation of both DMASNet and RRSGNet. Chapter 6 presents experiments and compares results. Chapter 7 discusses the findings, limitations, and related work. Chapter 8 concludes and suggests future directions.

2 Requirement Analysis

This chapter outlines what is needed to develop a shadow generation model that works well in augmented reality (AR) applications. In AR, making virtual objects look like they truly belong in the real world is a big challenge, and realistic lighting and shadows are essential to achieve that illusion.

To understand what the system should do, this chapter looks at who will use it, how it will be used, and what technical and design goals it needs to meet. The sections that follow describe the key stakeholders, typical use cases, and both the functional and non-functional requirements of the project.

2.1 Stakeholders

This project involves several key stakeholders, each with different needs and expectations:

- **Portrix.net GmbH**
  Portrix.net GmbH sponsors this project with the aim of exploring practical machine learning solutions for augmented reality. Their interest lies in lightweight, deployable AI model that will enhance the realism of AR content, particularly on constrained hardware platforms such as the Raspberry Pi.
- **The Developer**
  The person designing, building, and optimizing the model. The focus of the developer is to create an overall working solution and to achieve this within the thesis time frame.
- **AR Application Teams**
  This group includes engineers and product designers who will use the model within AR systems. They care about visual realism that the shadows should look natural

and the model should fit smoothly into the AR workflow, without needing extra manual steps and that the model works with various image sizes.

- **End Users**
  These are the people who use AR applications. They may not know how the model works, but they will notice if the shadows look wrong or out of place. For them, convincing shadows are the key to a believable AR experience.

2.2 Use Cases

The following use cases describe interactions with the shadow generation system.

2.2.1 UC-01: Generate Shadow for a Single Object

- **Actors:** End User, AR Application
- **Preconditions:**
  - A background image and a single foreground object, with a transparent background, are available.
- **Flow:**
  1. The user submits the background image and the image of the object to the system.
  2. The user places the objects onto the background image.
  3. The system infers generates intermediate masks to pass to the shadow generation model.
  4. The system generates a realistic, correctly oriented shadow for the inserted object .
  5. The system returns the final image at the same resolution and aspect ratio as the input.

2.2.2 UC-02: Generate Shadows for Multiple Objects

- **Actors:** End User
- **Preconditions:**
  - A background image and two or more foreground objects, with transparent backgrounds, are provided.
- **Flow:**
  1. The user or application submits the background image the images of the objects.
  2. The user places the objects onto the background image.
  3. The system infers generates intermediate masks to pass to the shadow generation model.
  4. The system generates a realistic, correctly oriented shadow for all inserted objects.
  5. The system returns the final image at the same resolution and aspect ratio as the input.

2.2.3 UC-03: Embedded Deployment on Raspberry Pi

- **Actors:** Developer, Embedded System
- **Preconditions:**
  - The model and its dependencies are installed on the Raspberry Pi.
- **Flow:**
  1. The developer loads an image–object pair onto the device.
  2. The device runs inference on CPU-only hardware.
  3. The system generates and composites the shadow as in UC-01.
  4. The result is saved or streamed back to the user interface.

2.2.4 UC-04: Generate Shadow in AR Application

- **Actors:** AR Application, End User
- **Preconditions:**
  - A background scene and a foreground object, with a transparent background, or a 3D object model are available.
- **Flow:**
  1. The user places the objects onto the background scene.
  2. The system infers generates intermediate masks to pass to the shadow generation model.
  3. The system generates a realistic, correctly oriented shadow for the inserted object .
  4. The system returns the final image with the generated shadow to the AR application.

2.3 Requirements

Based on the project goals, stakeholder needs, and identified use cases, the following functional and non-functional requirements are defined for the shadow generation system.

2.3.1 Functional Requirements

- **FR-1:** The system must generate a shadow for a given foreground object, placed onto a background image.
- **FR-2:** The system must accept input images of various resolutions and aspect ratios, and output a same sized image.
- **FR-3:** The system should generate shadows without needing masks for background objects or existing shadows.

- **FR-4:** The system must output a composite image including the background, inserted object, and its generated shadow.
- **FR-5:** The system could generate shadows for multiple inserted objects in the same image.

2.3.2 Non-Functional Requirements

These describe how well the system should perform, not just what it does.

- **NFR-1:** The model should run fast enough on cpu only devices, such as small embedded devices like a Raspberry Pi.
  - For a 256×256 image, inference time should be less than 10 seconds.
  - For an HD image, inference time should be less than 120 seconds.
- **NFR-2:** Shadows must look realistic and better or at least as good as existing models like DMASNet.
  - This will be tested using both user studies and image similarity metrics (e.g., LPIPS, SSIM, PSNR).
- **NFR-3:** The system should be easy to use with minimal setup. Users should not need to tweak many settings to get good results.
- **NFR-4:** The system should handle a variety of input conditions well:
  - Different lighting and background scenes.
  - Different object sizes and shapes.
- **NFR-6:** The code should be clean, well documented, and easy to extend for future development or research.

2.4 Traceability Matrix

Requirement	**UC-01**	**UC-02**	**UC-03**	**UC-04**
FR-1: Generate single shadow	X	X	X	X
FR-2: Preserve input resolution	X	X	X	X
FR-3: No manual masks required	X	X	X	X
FR-4: Output composite image	X	X	X	X
FR-5: Multiple-object support		X		
NFR-1: Embedded (Raspberry Pi) time			X	
NFR-3: Minimal setup	X	X		

3 Background

This chapter provides the necessary background information and reviews prior research relevant to the task of realistic shadow generation for image composition using deep learning. We begin by defining image composition and its associated challenges, then delve into specific techniques for shadow generation, relevant deep learning concepts like Fast Fourier Convolutions, the baseline DMASNet model, and finally, object/shadow detection methods.

3.1 Image compostion

Image composition refers to the process of selecting and extracting a foreground object or region from a source image (or creating a virtual object) and integrating it seamlessly into a target background image or scene [31]. This fundamental operation allows for the creation of new visual content by combining elements from different contexts and this process employed across many different fields, including visual effects, advertising, art, data augmentation, and augmented/virtual reality[5**?**].

Achieving this seamless integration is not that simple. Simply overlaying the foreground element onto the background often results in visually disturbing artifacts because the foreground may not be consistent with the background scene in several crucial aspects [31]. These inconsistencies are major obstacles to realism and can be broadly classified as follows:

- **Geometric Inconsistencies:** These happen when the inserted object does not match the scene in terms of position, size, or perspective. For example, the object might look too big or too small compared to its surroundings, be placed at an unrealistic angle, or not follow the 3D structure of the background properly [31]

- **Semantic Inconsistencies:** These occur when the inserted object does not make sense in the context of the scene, even if it is placed correctly in terms of geometry. For example, adding a deep-sea diver into a desert landscape would feel out of place because it does not fit the environment or story of the image. [31].
- **Appearance Inconsistencies:** These refer to visual mismatches between the inserted object and the background, especially in terms of lighting, color, and overall atmosphere. Common types include:
  - *Illumination Contradiction:* When the lighting on the foreground object doesn't match the background, such as differences in light direction, intensity, or color temperature, it becomes immediately noticeable. Image harmonization methods aim to correct these mismatches by adjusting the color and tone of the inserted object to better blend with the scene [45, 7, 8].
  - *Boundary Artifacts:* These happen when the edges around the object are too sharp or do not blend well, making it obvious that the object was added later. Blending and matting techniques are used to smooth the edges and make the transition look more natural [37].
  - *Shadow and Lighting Effect Inconsistency:* One of the most important parts of realism is how light and shadows interact. If the object doesn't cast the right shadow or if the background does not affect the lighting of the object, it breaks the illusion. This also includes reflections and how the light bounces around the scene [15, 44, 47].

Among all of the possible inconsistencies in a composite image, shadows play one of the most important roles in making the scene look realistic [15, 44, 47]. Shadows provide strong visual cues that anchor the inserted object within the scene, show its shape and how it interacts with surfaces, and provide valuable clues about the direction and quality of the light in the environment [32]. When a shadow is missing or does not match the lighting in the scene, whether it is in the wrong direction, too sharp, too soft, or the wrong color, it can quickly break the illusion. That's why this thesis focuses on creating methods that can automatically generate realistic and consistent shadows for inserted objects, so they fit naturally into the scene.

3.2 Shadow Generation Techniques

There are two main approaches to generating realistic shadows for inserted objects. The first relies on traditional computer graphics rendering techniques that require detailed information about the scene. The second uses deep learning, where models learn to predict shadows directly from image data without needing explicit scene geometry.

Rendering Based Shadow Generation

Traditional methods for generating shadows are based on the physical principles of light and how it behaves in the real world, and computer graphics rendering [9]. These approaches aim to produce physically accurate shadows, but they need a lot of detailed information about the scene, including:

1. **3D Geometry:** Detailed 3D models of both the inserted object and the background surfaces onto which shadows will be cast.[22, 9].
2. **Material Properties:** Information about how different surfaces reflect light.
3. **Lighting Conditions:** A precise understanding of the scene's lighting setup, including the position, size, intensity, and color of light sources; for example, sunlight, sky light, or indoor lamps [9].

With this information, rendering techniques like ray tracing or shadow mapping can simulate how light is blocked or scattered to generate realistic shadows. These methods can produce both sharp and soft shadows that look physically accurate in the scene [10].

While rendering based methods can produce highly realistic results, one of their main disadvantages in image composition is the challenge of obtaining the necessary information from real world scenes using only 2D images [22]. It is extremely difficult, and often impossible, to accurately estimate detailed 3D geometry, material properties, and the full lighting information of a scene from a single image, which is a process known as inverse rendering [3]. Because of this, many of the practical methods used in rendering require a lot of manual work, like defining geometry or estimating lighting conditions by hand [22]. Sometimes, they also need additional data, like environment maps [9], but that is not always feasible. Although differential rendering [9] can help insert objects with minimal geometry, it still relies on known lighting and has trouble handling complex shadow

interactions.

Deep Learning Based Shadow Generation

To overcome the need for detailed 3D information, recent research has turned to deep learning, particularly using image-to-image translation models like [18]. These methods aim to learn how to add realistic shadows to a composite image without the need for detailed 3D data. Instead, the model is trained on datasets of images paired with corresponding ground-truth images that include realistic shadows. By learning from these pairs, the model can generate plausible shadows for inserted objects, even when working with just a 2D composite image.

A key challenge for these supervised methods is gathering enough suitable training data. It is very hard to collect large-scale, real-world image pairs that perfectly align, with one version having foreground shadows and the other without, all under the same lighting conditions [15, 44]. As a result, researchers often turn to:

- **Synthetic Data:** This involves generating training pairs using 3D rendering engines [54, 27]. While this approach is scalable, it can introduce a gap between the synthetic data used for training and the real-world images the model is tested on.
- **Real-World Data Synthesis Pipelines:** In this case, researchers create paired data from real images with object-shadow pairs. For example, DESOBA [15] and its extension, DESOBAv2 [29], involve manually removing shadows of objects or using inpainting models to remove the shadow [29] to generate shadow-free versions of images.
- **Unpaired or Weakly Supervised Methods:** Getting perfectly matched image pairs, one with a shadow and one without a shadow, is somewhat of a hard task. So, some methods skip that pairing. For example, CycleGAN [57] just needs two sets of images: some with shadows, some without. It tries to learn what makes them different, even if the images do not match one-to-one. Other approaches borrow ideas from shadow removal like Mask-ShadowGAN [54] to help the model understand how shadows behave [27].

Even though deep learning models have made a lot of progress in generating shadows, they still face some common challenges. For example, advanced methods like SGRNet and DMASNet [15, 44] work at low, fixed resolutions, which makes it hard to capture fine shadow details like soft edges or smooth changes in intensity.

3.3 Foundational Deep Learning Concepts for Image Processing

The deep learning-based shadow generation methods discussed previously rely on several fundamental concepts and architectures commonly used in computer vision and image processing. Understanding these principles and building blocks is essential to analyze the design and capabilities of models such as DMASNet and the proposed RRSGNet. This section provides an overview of key building blocks such as Convolutional Neural Networks (CNNs) , Residual Networks (ResNets), Fast Fourier Convolutions (FFC) and Fully Convolutional Networks (FCNs) . These concepts help explain how models can learn to understand both local details and the broader structure of a scene, something that is especially important when generating realistic shadows.

3.3.1 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are a class of deep neural networks and it was originally developed for image recognition [51]. The typical architecture of it consists of an input image layer, followed by multiple convolutional layers, each convolutional layer is often followed directly by a pooling layer, and then ending with one or more fully connected layers for classification or regression. The way convolutional layers work, is that they apply layers of filters that detect patterns like edges, shapes, textures, and then the filters begin to combine those simple features into more complex structures, such as eyes, noses, or wheels or even entire objects like faces or cars [51]. For example, the LeNet-5 model by LeCun *et al.* (1998) was an early CNN that achieved high accuracy on digit recognition tasks [25]. CNNs have since become the most used approach for many computer vision tasks [26].

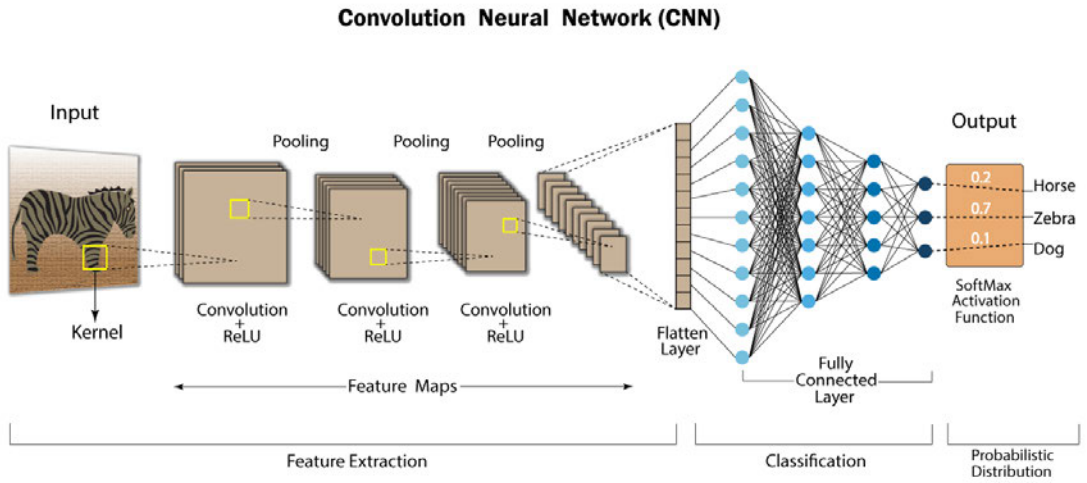


Figure 3.1: An Example of the architecture of a CNN. [43]

Convolution Operation

A convolutional layer basically applies a set of filters also known as kernels, that the model learns, across the input image to produce output feature maps that represents learned visual patterns. Each filter is a small matrix that slides (convolves) over the width and height of the input image to compute the dot product between the filter weights and the corresponding input patch. The result of this dot product is a single number and so sliding the filter over all the positions in the input produces a 2D matrix, referred to as a feature map, that highlights the learned patterns across the input [33]. Multiple filters can be used, their outputs are then stacked as separate channels in the layer's output, where each stack corresponds to the response of a specific filter. This way, early layers can learn simple patterns such as edges or colors, and deeper layers can combine them into complex features such as textures or object parts [33].

Figure 3.2: Hierarchy of layers, showing learned patterns from early to deeper layers. [16]

In convolutional neural networks (CNNs), several key concepts make the convolution operation efficient:

- **Local receptive fields**: Each neuron in a convolutional layer connects only to a small local region of the input,its receptive field. The size of this region is the filter size, and as layers stack, the effective receptive field grows [33].
- **Channel depth**: Each filter looks at the entire depth of the input for example, all three RGB channels of an image. If multiple filters are used, the output will have one feature map per filter, so convolutional layer with K filters will produce K output channels.
- **Parameter sharing**: The wieghts of each filter are reused and shared across all the input positions when the filter is sliding, which greatly reduces the number of parameters relative to a fully connected layer for example [33].

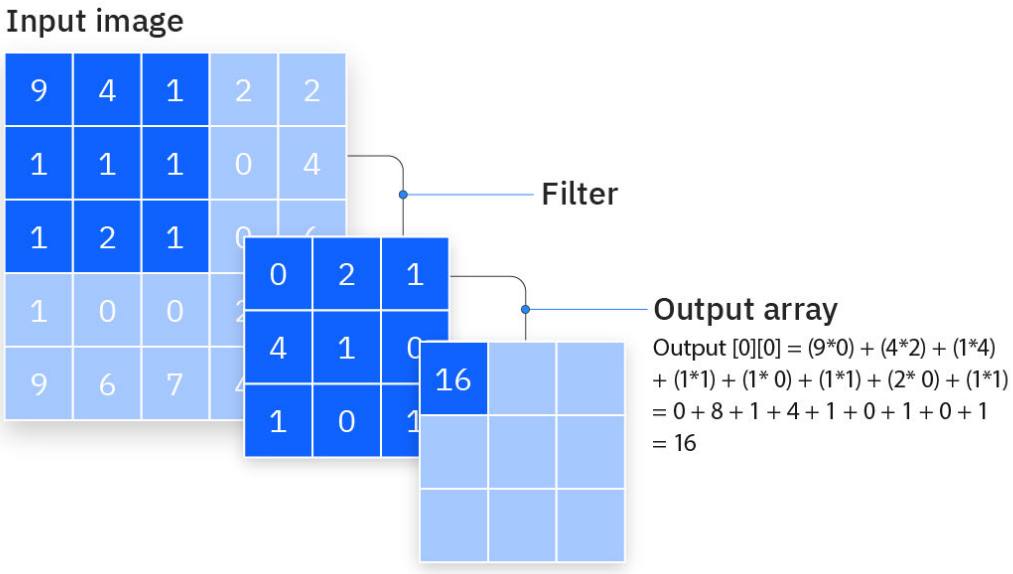


Figure 3.3: Illustration of a convolution operation. [16]

Padding

When a convolution is applied, the output usually becomes smaller because the filter can't fully slide over the edges of the input. Padding is a way to fix this. It adds extra pixels, usually zeros, around the border of the input so the filter can go over every part, including the edges.

- Valid padding: No padding; output size = (input size – filter size + 1).
- Same padding: Pad enough zeros (typically $\lfloor F/2 \rfloor$ on each side) so that output size matches input size (for stride 1).

Padding helps:

- Keep the output size the same as the input which is useful in deep networks.
- Avoid losing information at the edges.
- Make sure features align well across layers.

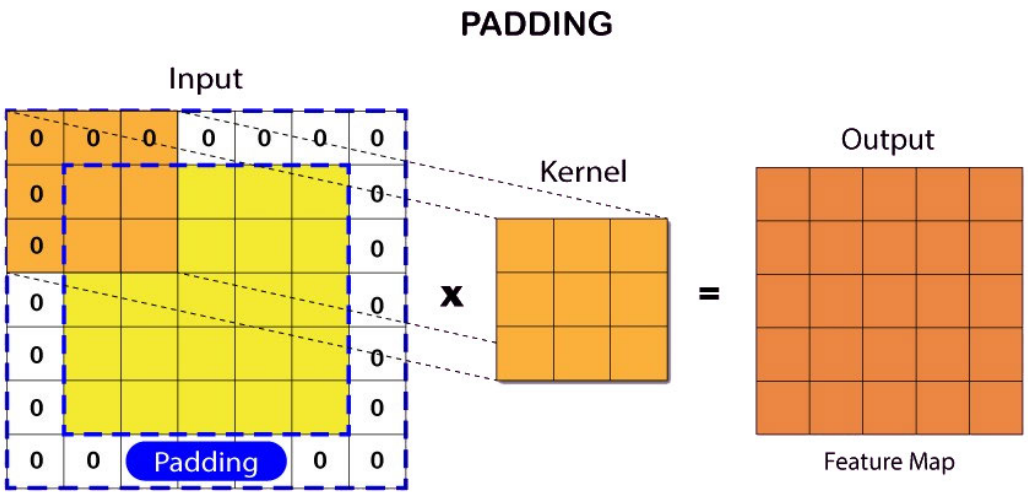


Figure 3.4: Example of zero padding in a convolutional layer. [16]

Stride

The stride of a convolution is the step size of how much the filter window moves across the input. A stride of 1, which is the default, means the filter shifts one pixel at a time. A larger stride moves the filter by multiple pixels, effectively downsampling the output, making the output smaller [33]. For example, using a 3×3 filter with stride 2 on a 224×224 image "with padding" produces an output of about 112×112 which is half the size in each dimension. Larger strides reduce the spatial resolution more quickly, but may lose detail. They are often used instead of pooling layers to reduce the size of the feature map in some network architectures.

- **Stride** s: Number of pixels the filter takes in each step, both horizontally and vertically.
- **Effect**: Bigger strides results in smaller outputs, it downsamples by a factor of s.

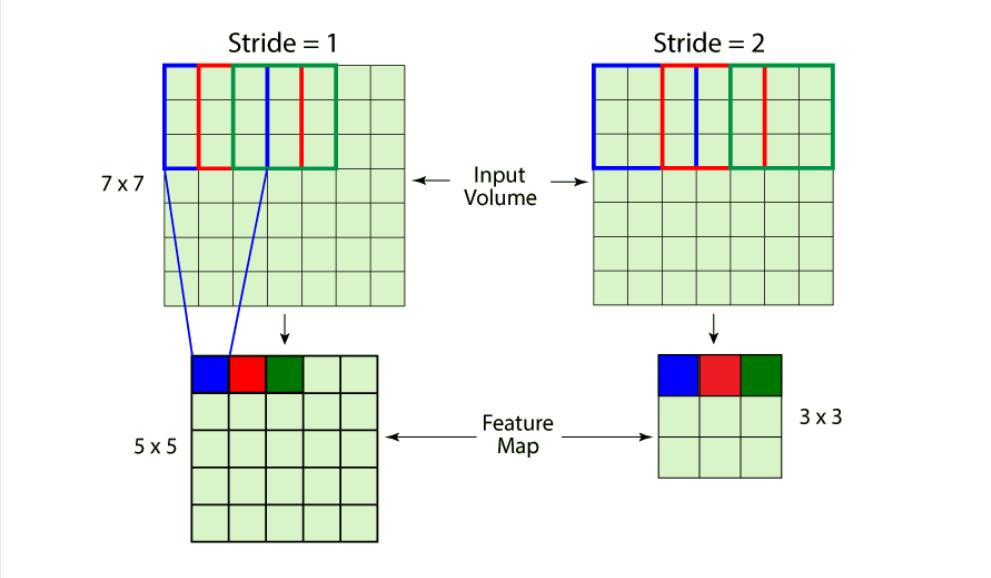


Figure 3.5: Example of stride in a convolutional layer. [43]

Activation Functions

After each convolution, an activation function is applied to the output of that convolution to introduce non-linearity. This helps the network learn complex patterns instead of just linear ones. The most common activation in CNNs currently is the Rectified Linear Unit (ReLU). ReLU is defined as:

$$f(x) = \max(0, x)$$

This function is simple yet effective, it sets negative values to zero while keeping positive values unchanged. This makes the network faster to train and helps avoid issues like vanishing gradients (a situation where gradients become very small as they are back-propagated through many layers, causing the weights to update very slowly or not at all) [30]. Nair and Hinton (2010) showed that replacing traditional activation units with ReLUs in deep networks leads to improved learning and better feature representations [30]. Before, earlier networks used functions like sigmoid or tanh, but these can slow training because they saturate, flatten out, for large inputs [30].

- **ReLU**: $f(x) = \max(0, x)$. Fast, simple, and works well in deep networks.[30]
- **Sigmoid/Tanh**: Older choices, smooth but can slow training due to vanishing gradients.[30]
- **Softmax**: Turns raw scores into probabilities, used in the output layer for multi-class classification.[36]

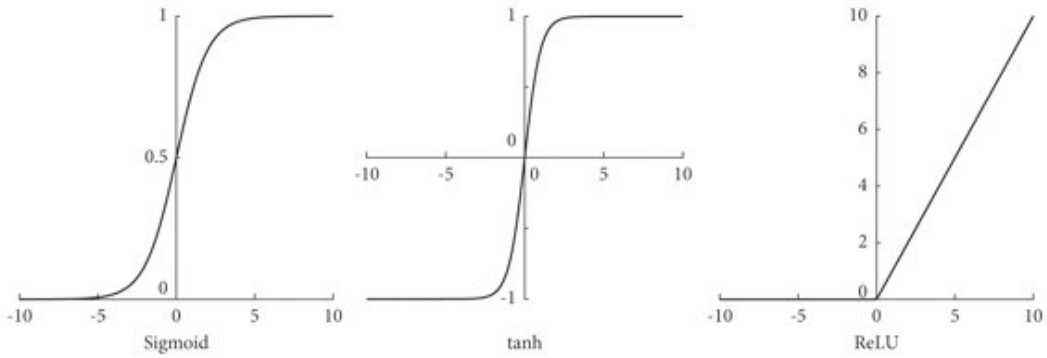


Figure 3.6: Comparison between Sigmoid, Tanh and ReLU. [20]

Pooling Layers

Pooling layers reduce the spatial dimensions, height and width, of feature maps while retaining important information. This helps make the network more efficient and provides a degree of translational invariance. The two most common types are max pooling and average pooling [33]. Max pooling (e.g., with a 2×2 window and stride 2) takes the maximum value in each non-overlapping region of the input, effectively summarizing the most prominent feature in that region. Average pooling takes the mean of the values in the region instead of the maximum value. By reducing the height and width of the representation, pooling layers make models more efficient and help prevent overfitting by reducing sensitivity to exact feature locations [33]. For example, the ImageNet architecture interleaved max pooling layers between convolutional layers to progressively reduce resolution while preserving critical features [24].

- **Max pooling**: Outputs the maximum value from each region.
- **Average pooling**: Outputs the mean value from each region.

- **Effect**: If a 2×2 max-pool is applied, the output dimensions are halved.

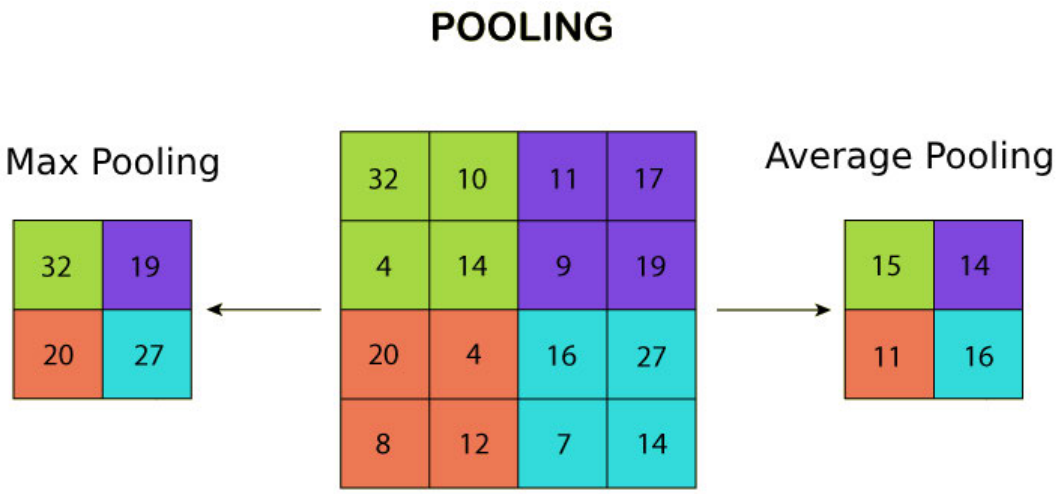


Figure 3.7: Example of max and average pooling. [43]

Normalization Layers (Batch and Instance)

Normalization layers are used between convolution and activation to improve training speed and stability. They help standardize the inputs to each layer, which reduces training difficulties and speeds up convergence.
Batch Normalization (BatchNorm) is a widely adopted method, introduced by Ioffe and Szegedy (2015) [17], normalizes each channel's activations to have zero mean and unit variance, computed over a small batch. This helps reduce internal covariate shift, which is the change in input distribution across layers during training, and allow the use for higher learning rates and results in faster training [17]. BatchNorm also includes learnable scale and shift parameters to keep the flexibility after normalization [17].
Instance Normalization (InstanceNorm), on the other hand, normalizes each feature channel for each input sample individually, across its dimensions. Initially proposed for style transfer networks [46], it has been found effective in generative models, especially when image appearance or style is more important than keeping the original visual details of each input [46].

- **Batch Normalization**: Normalizes each feature channel using the mean and variance over the mini-batch, improves training speed and stability [17].

- **Instance Normalization**: Normalizes each feature channel per sample, independently of the batch, often used in image generation and style transfer [46].

3.3.2 Residual Networks

Residual Networks (ResNets) are a type of deep convolutional neural network architecture designed to solve the challenges that come with training very deep networks. Simply stacking many layers on top of each other, makes it harder for the network to learn, gradients can also vanish, and training accuracy may even get worse as it goes deeper into the network. To solve this, He et al.(2015) introduced the concept of residual learning [13].
What residual learning means is basically that instead of making the network try to learn the full mapping $H(x)$(the function or the relationship that connects "maps" the input to the output) directly, each block in a ResNet learns a residual function defined as:

$$F(x) = H(x) - x$$

[13] This can be rearranged as:

$$H(x) = F(x) + x$$

So basically, residual networks add the input x directly to the output of a few layers using a shortcut connection. This helps the network learn more easily by focusing on the difference between the input and the desired output. If the best solution is to leave the input unchanged, it is easier for the network to learn to do nothing than to force several layers to copy the input exactly [13].

This approach allowed very deep ResNets, up to 152 layers, to be trained more effectively and much faster, achieving better accuracy than traditional deep networks [13]. ResNets were the first models to successfully train a model with over 100 layers on the ImageNet dataset[24] and achieved state-of-the-art results, winning the ImageNet Large Scale Visual Recognition Challenge 2015 competition [13]. Overall, the introduction of skip connections in residual networks significantly reduces training difficulties in very deep architectures.

Skip Connections and Identity Mapping

One of the most important features of ResNets is the use of skip connections, also known as shortcut connections. These shortcuts allow the input x to skip one or more convolutional layers and be added directly to the output of those layers. So basically, the shortcut simply copies the input, this is called an *identity mapping*. [13] For example, if x is an input to a residual block and $F(x)$ is the function learned by the stacked convolutional layers, the output is then:

$$x + F(x)$$

[13] Most importantly is that these shortcuts do not introduce extra weights or computation and simply pass the input data through. This structure helps information and gradients flow more easily through the network because the shortcut adds the input directly to the output [13].

To summarize, residual networks use skip connections to preserve input information and help gradients flow, which leads to faster training and better performance.

ResNet-34 Architecture

ResNet-34 is a type of deep convolutional neural network that uses *residual learning* to train effectively, even with many layers. It was introduced as part of the ResNet family by He et al. [13], and is specifically designed to reduce the training problems found in very deep networks, such as vanishing gradients. ResNet-34 has 34 layers with learnable weights, including convolutional layers, and is known for its good balance of accuracy and efficiency [13]. The architecture begins with an initial convolutional layer that employs a 7×7 kernel with 64 filters and a stride of 2. This initial convolution is followed by a 3×3 max pooling layer with a stride of 2, which reduces the spatial dimensions of the input image. [13]Following the initial layers, the network consists of four stages, each containing a sequence of residual blocks:

- **Conv2_x**: 3 residual blocks, each with two 3×3 convolutional layers, 64 filters [13].
- **Conv3_x**: 4 residual blocks, each with two 3×3 convolutional layers, 128 filters. The first block uses stride 2 for downsampling [13].

- **Conv4_x**: 6 residual blocks, each with two 3×3 convolutional layers, 256 filters. The first block uses stride 2 for downsampling [13].
- **Conv5_x**: 3 residual blocks, each with two 3×3 convolutional layers, 512 filters. The first block uses stride 2 for downsampling [13].

Each residual block includes a skip connection that adds the input of the block to its output, enabling the network to learn residual functions with reference to the layer inputs. After the stacks of the residual blocks, the network includes:

- **Global Average Pooling**: Reduces each feature map to a single value.
- **Fully Connected Layer**: Outputs class scores for classification.

This configuration results in a total of 34 layers with learnable parameters. The use of residual connections allows for efficient training of deep networks by mitigating issues such as vanishing gradients [13].

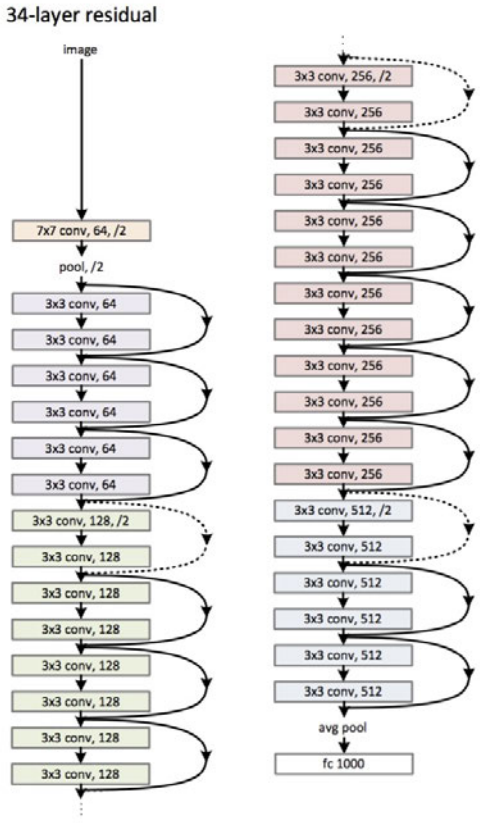


Figure 3.8: ResNet-34 Architecture Diagram. [13]

Training Deep Networks with Residuals

Residual learning makes it much easier to train very deep neural networks. The key idea is to use these identity shortcut connections, also known as skip connections, that create shortcuts for gradients to flow backward during training. These shortcuts help prevent problems like vanishing or exploding gradients, which commonly occur in very deep networks [13]. In the ResNet paper [13], it was shown that simply making a plain non-residual network deeper often made training worse. For example, a 56-layer plain network performed worse than a 20-layer one on CIFAR-10[23, 13]. Where on the other hand, a 56-layer ResNet trained successfully and even improved accuracy [13].

The reason is that identity shortcuts allow gradients to pass through the network more directly, without being blocked or for them to fade. So, if both the residual function and the shortcut use identity mappings, the output of each block is just the input plus a small learned change. This makes it easier for the network to update itself during training [13]. Thanks to this design, adding more layers no longer makes training harder. As long as overfitting is managed, deeper ResNets generally achieve better accuracy. While techniques like Batch Normalization are also important, the residual structure is the main reason ResNets can successfully train networks with tens or even hundreds of layers [13].

Relevance for Segmentation and Generation Tasks

ResNets are not only useful for image classification, they are also widely used in other computer vision tasks like image segmentation and image generation. As ResNets work well as feature extractors because they can capture both local and global context. ResNet blocks are also popular in encoder-decoder models like U-Net [39], where they are used in the encoder to extract meaningful features from input images. These features are then used by the decoder to predict pixels for the output image.

So to summarize, residual networks, such as ResNet-34, offer a powerful and reliable architecture for deep learning in computer vision. The key innovations are residual learning and identity skip connections which make it possible to train much deeper networks effectively by preserving information flow and allowing gradients to pass through layers more easily [14][13]. Because of their flexibility and strong performance, ResNets have become

the standard backbones in a variety of visual tasks, including image classification, object detection, semantic segmentation, and image generation [34].

3.3.3 Fast Fourier Convolutions (FFCs) for Global Context and Resolution Robustness

The field of image processing is continually advancing, driven by an increasing demand for models capable of understanding the very details of visual data while also needing robustness to variations in image resolution. Many modern vision tasks, such as image inpainting, semantic segmentation, and shadow generation, require an understanding of global relationships and details across the entire image. For example, inpainting large missing regions or generating realistic shadows requires information about lighting and geometry that can extend throughout the full image. Also, segmentation of distant objects or large structures benefits from global context in the image.

Traditional CNNs struggle with this because they use small filters for example, a 3×3 filter, that only look at local parts of the image where they are applied. To see the whole image, they must stack many layers [6]. Because of this limitation, they have trouble connecting distant regions. In addition, when applied to images with different resolutions from training data, the performance of the model is much worse [42][6].

In short, regular CNNs struggle to understand the overall structure of an image and are not naturally robust to changes in image size.

Classical signal processing offers a solution to handle global image context more efficiently. According to the convolution theorem, a convolution in the spatial domain is the same as multiplying in the frequency domain. This means that a small change in the frequency domain can affect the entire image when transformed back.

Mathematically speaking, if $\mathcal{F}$ is the 2D Fourier transform, then applying it to a feature map X gives a frequency representation $\hat{X}$. By multiplying this with a learned function $W(\mathbf{k})$ in the frequency domain, we get $\hat{Y}$, and the inverse Fourier transform gives us the output Y in which every pixel in Y depends on all pixels in X.

In other words, a simple operation in the frequency domain can influence the whole image, making it useful for capturing global relationships [6]. The Fast Fourier Transform (FFT) makes this efficient, running in $O(N \log(N))$ time ,where N is equal to the total number

of pixels , which is often faster than using large convolutional filters or stacking many layers [6].

Fast Fourier Convolution Architecture

The Fast Fourier Convolution (FFC) block [6] combines standard convolutions with global processing using the Fourier Transform. It splits the feature channels of an input tensor X into two parts:

- X_l: the **local branch**, which handles features with normal small kernel convolutions
- X_g: the **global branch**, which uses the Fourier Transform to capture image-wide context

The split is controlled by a ratio α_{in} so that:

$$X = \{X_l, X_g\}, \quad \text{with} \quad X_l \in \mathbb{R}^{H \times W \times (1-\alpha_{\text{in}})C}, \quad X_g \in \mathbb{R}^{H \times W \times \alpha_{\text{in}} C}$$

Where:

- H is the height of the image or feature map (number of rows),
- W is the width (number of columns),
- C is the number of channels.

The outputs are also split as $Y = \{Y_l, Y_g\}$, possibly with a different ratio α_{out}. Each part in the FFC block then computes:

- **Local-to-local:** $Y_{l \to l} = f_l(X_l)$ via standard convolution
- **Global-to-global:** $Y_{g \to g} = f_g(X_g)$ using a spectral transformer
- **Cross terms:** Local and global branches influence each other through:

$$Y_{g \to l} = f_{g \to l}(X_g), \quad Y_{l \to g} = f_{l \to g}(X_l)$$

The final outputs are sums of these contributions:

$$Y_l = f_l(X_l) + f_{g\to l}(X_g), \quad Y_g = f_g(X_g) + f_{l\to g}(X_l)$$

The key innovation in the global branch is the **Fourier Unit (FU)**. It includes:

1. A 1×1 convolution to reduce channel size
2. Two sub-units:
    - A **Fourier Unit (FU)** applied to all channels for global context
    - A **Local Fourier Unit (LFU)** applied to a small subset (typically $1/4$) for semi-global context
3. A skip connection that adds the input back
4. A final 1×1 convolution to restore full channel size

So basically, the local branch processes the local channels with normal convolutions, while the global branch transforms the frequency channels via FFT-based operations. Cross branch convolutions ensure that local features can influence the global output and vice versa.

The multibranch structure of the FFC gives each layer access to both fine local details and broad global context. Because operations in the spectral branch affect the entire image at once, an FFC layer has a global receptive field, even in the early layers. As noted by Suvorov et al.[42], "FFCs allow for a receptive field that covers an entire image even in the early layers of the network." [42] This global reasoning improves the efficiency and quality of the output of the model, especially in tasks that depend on understanding the whole scene.

FFC in inpainting Inspiration

An existing proof of the efficiency of ffcs is the model LaMa, a high-resolution inpainting network that incorporates Fast Fourier Convolutions into a fully convolutional encoder-decoder architecture [42]. Each residual block in LaMa, except the first, replaces standard convolutions with FFC layers, enabling early access to global context through spectral

processing [42]. The generator downscales the input image, applies several FFC-based residual blocks, and then upsamples the output [42].

The authors highlight two key benefits of FFCs in LaMa:

- **Global consistency**: Because spectral convolutions have image-wide receptive fields, LaMa can reconstruct large-scale patterns such as brick walls, fences, or windows more accurately and consistently than conventional CNNs. This is particularly important for inpainting structured textures or repeating elements [42].
- **Resolution robustness**: Unlike traditional CNNs, which often fail when applied to images larger than those seen during training, LaMa generalizes well to much higher resolutions, from 512×512 to 2048×2048 [42]. This robustness is because of the frequency domain processing of FFC, which naturally works well with various sizes.

Despite using only about 45M parameters, the LaMa-Fourier model outperforms larger CNN-based models like CoModGAN [55], which has 109M parameters, in high resolution inpainting tasks [42]. It also offered greater parameter efficiency and showed fewer artifacts when tested on images with resolutions higher than those used in training [42]. These results showed that FFC-based architectures are very good at capturing large-scale and periodic patterns and generalizing beyond trained resolutions.

Comparison Between Standard Convolutional Layers and Fast Fourier Convolutions

Table 3.1: Key differences between standard convolutions and Fast Fourier Convolutions (FFC).

Property	**Standard Convolution**	**Fast Fourier Convolution**
Receptive field	Local (limited by kernel size; grows with depth)	Global (covers entire image from early layers)
Architecture	Single spatial branch	Dual branches: local + global (spectral)
Frequency-domain use	Not used	Yes (FFT $\rightarrow$ spectral conv $\rightarrow$ iFFT)
Kernel size	$K \times K$ (e.g., 3×3)	Effectively global (via FFT, using 1×1 conv)
Context awareness	Needs many layers	Directly captures broad context
Resolution robustness	No	Yes

Summary

To summarize, FFCs overcome the limited receptive field of standard convolutional layers by combining local convolutions with global frequency domain processing [6, 42]. and unlike traditional CNNs, which need deep stacks to capture long range dependencies, FFCs offer full image context from the early layers. Also, their spectral branch makes the model naturally better at handling images of different sizes, since frequency based features generalize well across scales [42].

3.3.4 Deep Learning Frameworks

Deep learning today is powered by widely used frameworks like **TensorFlow** and **PyTorch**, which make it easier to build and train neural networks. **TensorFlow** is developed by Google Brain and released in 2015 and it uses a static computation graph to represent operations. This approach allows the framework to optimize the entire computation ahead of time, which is useful for scaling up models and deploying them across

different devices like CPUs, GPUs, and TPUs [1, 2]. That is why TensorFlow has become a popular choice for production environments, especially at Google.

On the other hand, **PyTorch**, released by Facebook AI Research in 2016, takes a more flexible approach. It uses dynamic computation graphs and eager execution, which means code runs immediately and is easier to write and debug similar to working with NumPy but with GPU support [35]. PyTorch builds on the older Torch library, but makes everything accessible in Python, which has helped it become a favorite in the research community [35].

In short, TensorFlow offers a more structured system that is great for large scale deployment, while PyTorch focuses on simplicity and flexibility, making it ideal for experimentation and rapid development.

3.3.5 Loss Functions in Deep Learning

Loss functions are a key part of supervised learning. They measure how far the predictions of a model are from the actual targets and guide the learning process during training. Here are some commonly used loss functions in deep learning, with their formulas and explanations.

L1 and L2 Loss

The L1 (mean absolute error) and L2 (mean squared error) losses are standard for regression tasks:

$$\mathcal{L}_{L1} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|, \qquad \mathcal{L}_{L2} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{3.1}$$

where $\hat{y}_i$ is the prediction and y_i is the groundtruth value. L2 loss penalizes larger errors more heavily and assumes normally distributed noise [19]. L1 loss works better when there are unexpected values in the data and usually makes the model focus on the most important parts [19].

Binary Cross-Entropy (BCE)

The binary cross-entropy loss , also know as the logistic loss, measures the difference between predicted probabilities $\hat{p} \in (0, 1)$ and groundtruth labels $y \in 0, 1$. For a single example, the loss is

$$\mathcal{L}_{\text{BCE}} = -[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})]. \tag{3.2}$$

[38]

In practice, this is averaged or summed over all examples. BCE is widely used for binary classification tasks, logistic regression, and for binary segmentation masks in computer vision problems.

Perceptual Loss

Perceptual loss compares high-level features from a pretrained network, like VGG [41], instead of comparing raw pixels like in other losses [21]. Perceptual loss is typically used in image generation tasks to produce outputs that look visually similar to human observers, even if the pixel-wise differences are large [21].

Learned Perceptual Image Patch Similarity (LPIPS)

LPIPS is an advanced, commonly used, perceptual similarity metric between images [53]. It calculates the distance between two image patches x and $\hat{x}$ by using learned weights on deep network features [53]. It is often used as a loss or evaluation metric in generative models where human perception quality is important. And it is better at capturing differences in images like the human eye.

Complete Intersection over Union (CIoU)

CIoU loss is used in object detection to compare bounding boxes. For a predicted bounding box $\mathbf{b}$ and a ground truth box $\mathbf{b}^{gt}$, the Intersection over Union (IoU) measures how much the two boxes overlap, divided by the total area covered by both [56]. The Complete IoU (CIoU) improves upon this by also penalizing differences in the distance between box centers and in their aspect ratios [56].

3.3.6 Evaluation Metrics

Metrics are important for knowing how well the model performs on unseen data, and for benchmarking the model for it to be compared with other baselines.

Root mean square error (RMSE)

RMSE is a simple metric that calculates the average difference between pixel values in the generated image and groundtruth image [4]. It is the square root of the average of the squared pixel-wise differences. A low RMSE means that the generated image is more similar to the groundtruth [4]. RMSE is measured in the same units as the pixel values, intensity, so it offers an insight of the average error per pixel. This makes RMSE especially useful in tasks such as image restoration, where precise pixel values matter.

Peak Signal-to-Noise Ratio (PSNR)

PSNR is closely related to RMSE but expressed on a logarithmic scale [40]. It uses the maximum possible pixel value as the signal and compares it with the mean squared error, which is considered the noise [40]. A higher PSNR indicates less noise and a higher quality image. Since PSNR is derived from RMSE, they are closely related so, as RMSE decreases, PSNR increases. It is widely used because its ability to normalize error across different image scales [40].

Balanced Error Rate (BER)

BER is a way to measure how well a model performs in a classification task where there are two classes, like shadow and non-shadow. BER looks at the error rate for each class separately, instead of just counting how many total predictions were correct [12]. It then takes the average of those two error rates. This gives a fair evaluation by treating both classes equally, no matter how many pixels belong to each one [12]. A BER of 0 means that the model correctly classified all pixels and a higher BER means the model made more mistakes [12]. Because of this balanced approach, BER is often used in tasks like shadow detection, where the shadow regions are usually much smaller than the non-shadow ones.

3.4 DMASNet: Decomposed Mask Prediction and Attentive Shadow Filling

DMASNet is a two-stage convolutional neural network designed to generate realistic shadows for inserted foreground objects in a composite image [44]. It is designed to separate the task of generating shadow into two parts, predicting the shape and position of the shadow, and filling in the shadow. This division of tasks helps produce accurate, clean shadow shapes and ensures that their appearance matches the lighting of the background scene.

3.4.1 Two-Stage Architecture Overview

DMASNet works in two stages. In the first stage, called mask prediction, the model guesses where and what shape the missing shadow should be. In the second stage, called shadow filling, it adjusts the appearance of the shadow so it blends naturally with the lighting of the background scene.

This method has been shown to produce more realistic shadows than earlier approaches and works well even on real-world composite images [44].

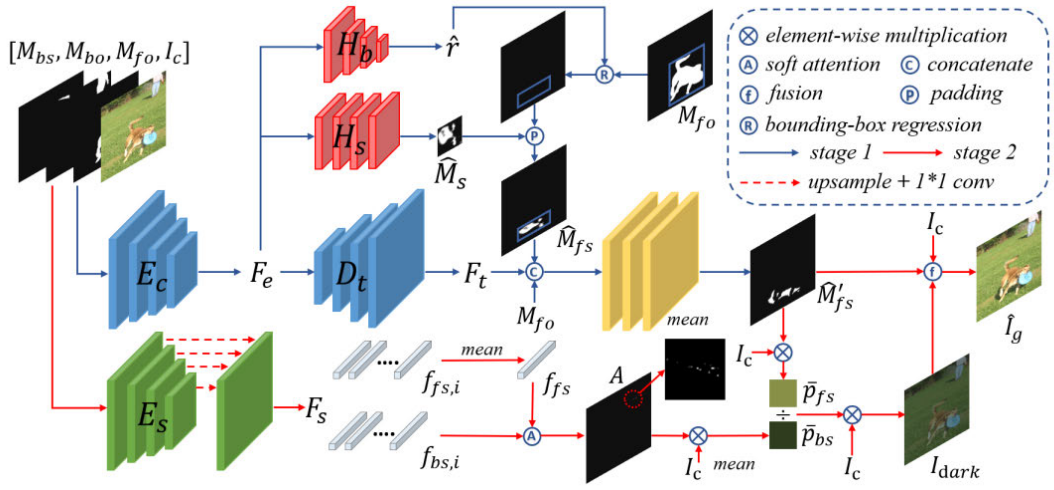


Figure 3.9: DMASNet Architecture diagram. [44]

Mask Prediction Stage

This is the first stage of DMASNet, it focuses on predicting a binary shadow mask for the foreground object. This task of predicting the shadow is also split into two subtasks, bounding-box regression and shape prediction.

A CNN backbon extracts multi-scale features from the input. This input consists of the composite RGB image along with masks for the foreground object and background objects and shadows. These features are passed to two separate branches of the network:

1. Box head (bounding-box regression): This branch predicts a set of four values $(\Delta x, \Delta y, \Delta w, \Delta h)$, which are used to adjust predict a bounding box, a place for the shadow to be placed at, based on the location of the inserted object [44]. The network first computes a bounding box around the foreground object, then learns how to shift and resize it to match the expected position and size of the to be generated shadow [44]. This transformation is trained using a Complete IoU (CIoU) loss, which encourages the predicted shadow box to align well with the ground truth [44].

2. Shape head (mask prediction): The shape head is responsible for predicting what the shape of the shadow. For simplicity, DMASNet always predicts the shadow shape within a fixed-size square region, specifically, a 64×64 pixel area. To achieve this, the model first crops the part of the input image that contains the foreground object. This cropped region is then resized to 64×64 pixels [44]. The shape head takes this resized patch and outputs a 64×64 binary mask, which represents the shape of the shadow within that region. During training, the groundtruth shadow mask is processed the same way, cropped, resized, and compared to the predicted mask [44]. This allows the model to focus on learning realistic shadow shapes, without needing to worry about their position in the full image [44]. After both the bounding box and the shape mask are predicted, the 64×64 shadow mask is upscaled and placed inside the predicted shadow box. This creates a rough version of the shadow that is in the right place and has the right general shape, but may still lack finer details.

After that the shadow is refined using a small decoder network. It takes the features from the CNN backbone and combines them with the rough shadow mask. They are then passed together through several convolutional layers that upsample and enhance the mask, producing a final high-resolution, refined version. This refined mask is trained using an L1 loss, which helps it capture finer details like soft edges and smooth transitions [44].

In summary, the output of the first stage is a binary shadow mask M_{fs} that defines location and shape of the shadow.

Attentive Shadow Filling Stage

Once the shadow mask is predicted in stage 1, the second stage of DMASNet adjusts the intensity of the shadow to make it blend naturally into the bachground scene. The idea is that the generated shadow should have a similar appearance to the already presented background shadows in the image [44]. To achieve this, DMASNet uses an attention-based mechanism that uses shading information from background shadows to fill the predicted foreground shadow [44].

Feature Extraction: The network uses the input composite image and input masks, and processes them through a second encoder to get feature maps at multiple scales [44].

How it works:

- The network looks at features, visual details, from the predicted shadow area and from all background shadows.
- It then calculates how similar the predicted shadow is to each background shadow.
- Based on these similarities, it gives more weight to background shadows that are more alike.

The model then computes the average darkness of similar background shadows using this weighted attention. After that it adjusts the predicted intensity of the shadow to match this average, making the new shadow blend naturally with the image [44].

3.4.2 Strengths and Limitations

DMASNet has a lot of strengths for example, its two-step mask prediction, bounding box and shape, helps generate clean, realistic shadows with an accurate placement. The attentive shadow filling even improves realism more by adapting the intensity of the shadow to match the scene. In experiments, DMASNet outperformed previous methods like SGRNet and generalized well to real-world images. However, it works with fixed input sizes only, 256×256 to be exact, and predicts shadows at a fixed resolution of

64×64, which limits scalability. Finally, its shading approach assumes uniform shadows, that all of the shadow has the same darnkness or intensity, which may not handle complex lighting well. Despite these limitations, DMASNet provides a structured and effective solution for realistic shadow generation in image compositing.

3.5 Single-Stage Instance Shadow Detection (SSIS)

SSIS is an AI model that predicts object and shadow pairs.

Traditional approaches handled this task in two separate steps:

1. Detect object masks and shadow masks.
2. Apply a post-processing step to match each shadow to its corresponding object.

SSIS: A Unified Approach

SSIS introduces a simpler, end-to-end approach. It uses a single convolutional neural network that takes an entire image as input and directly outputs all object and shadow masks, along with their correct pairings. By learning both detection and relationships together, SSIS avoids post-processing and improves both efficiency and accuracy.

Figure 3.10: An example output of SSIS. [48]

Architecture and Key Features

The model is built upon **Detectron2**, which is a high-performance object detection model developed by Facebook AI Research [49]. So it is built on top of the architecture used in detectron2, which creates custom convolution filters for each object or shadow, and this allows it to produce detailed masks more quickly and handle complicated shapes, like irregular shadows, more effectively [49] [48].

A key feature of SSIS is its *bidirectional relation learning* [49]. This part of the network learns the locational relationships between shadows and their objects by predicting the direction and distance as offset vectors from each shadow to its object, and the other way around. This helps the model correctly match pairs even when objects and shadows are far apart or partly hidden [49]. To increase accuracy even more, SSIS uses a *deformable MaskIoU head*, which checks how well each predicted mask matches the ground truth. It acts like a quality filter, helping to remove bad predictions and keeping only the most reliable ones [49].

In short, SSIS offers a smart and efficient way to understand how objects and shadows relate in an image. Its object-shadow matching and fast architecture make it an ideal fit for modern tools that work with complex scenes in real world.

4 Concept and Methodology

This chapter will detail the technical concepts and methods used to design the shadow generation system, building directly on the requirements from chapter 2. It describes how the selected techniques, from chapter 3, are applied to satisfy those requirements.

4.1 Technology Stack and Tools

The system is implemented in **Python 3.9.21**, a widely adopted language in the field of machine learning and computer vision. Python's simplicity, extensive community support, and the availability of high-performance libraries make it an ideal choice for prototyping and experimentation.

4.1.1 Programming Language and Core Libraries

- **Python 3.9.21:** Chosen for its popularity and support in scientific computing and deep learning workflows.
- **Pytorch:** Chosen for its flexibility in model design and debugging, its strong support for GPU acceleration, and for its huge community support.
- **NumPy:** Used for efficient numerical computations and array manipulations.
- **OpenCV:** Employed for image input/output, preprocessing, and transformations such as resizing, cropping, and mask processing.
- **Matplotlib:** Supports visualization of training metrics, intermediate outputs, and final results, aiding in debugging and visual analysis.
- **TensorBoard:** Utilized for logging training metrics, visualizations, and experiment tracking.

4.2 Dataset Selection and Preparation

This section outlines the datasets used for training and evaluating the shadow generation model, as well as the reason for their selection. A combination of synthetic and real-world datasets was used to ensure both scalability and real-world generalization.

4.2.1 Synthetic Dataset: RdSOBA

The primary dataset used for initial training is the Rendered Shadow-Object Association (RdSOBA) dataset [44], a large scale synthetic collection designed for shadow realated tasks, created using a unity game engine. RdSOBA consists of [44]:

- 30 distinct 3D scenes.
- 788 3D object models.
- 4 super-categories for foreground objects, containing people, animals, vehicles, plants.
- Approximately 600 synthetic 2D scenes rendered from different viewpoints of the 3D scenes.
- About 80,000 object-shadow image pairs.
- Total of 280,000 example tuples.

In this dataset, each object is rendered with and without shadows, and the differences are used to generate the shadow and object masks [44]. Figure 4.2 illustrates example scenes from RdSOBA. For each example, the dataset provides [44]:

- The composite image with a foreground object (I_{c}).
- The foreground object mask (M_{fo}).
- The foreground shadow mask (M_{fs}).
- The background object mask (M_{bo}).
- The background shadow mask (M_{bs}).
- The groundtruth image (I_{g}).

This large dataset offers diversity and scale, it ensures the model learns a broad range of shadow patterns before ever seeing real photos, satisfying our requirement for handling different conditions.

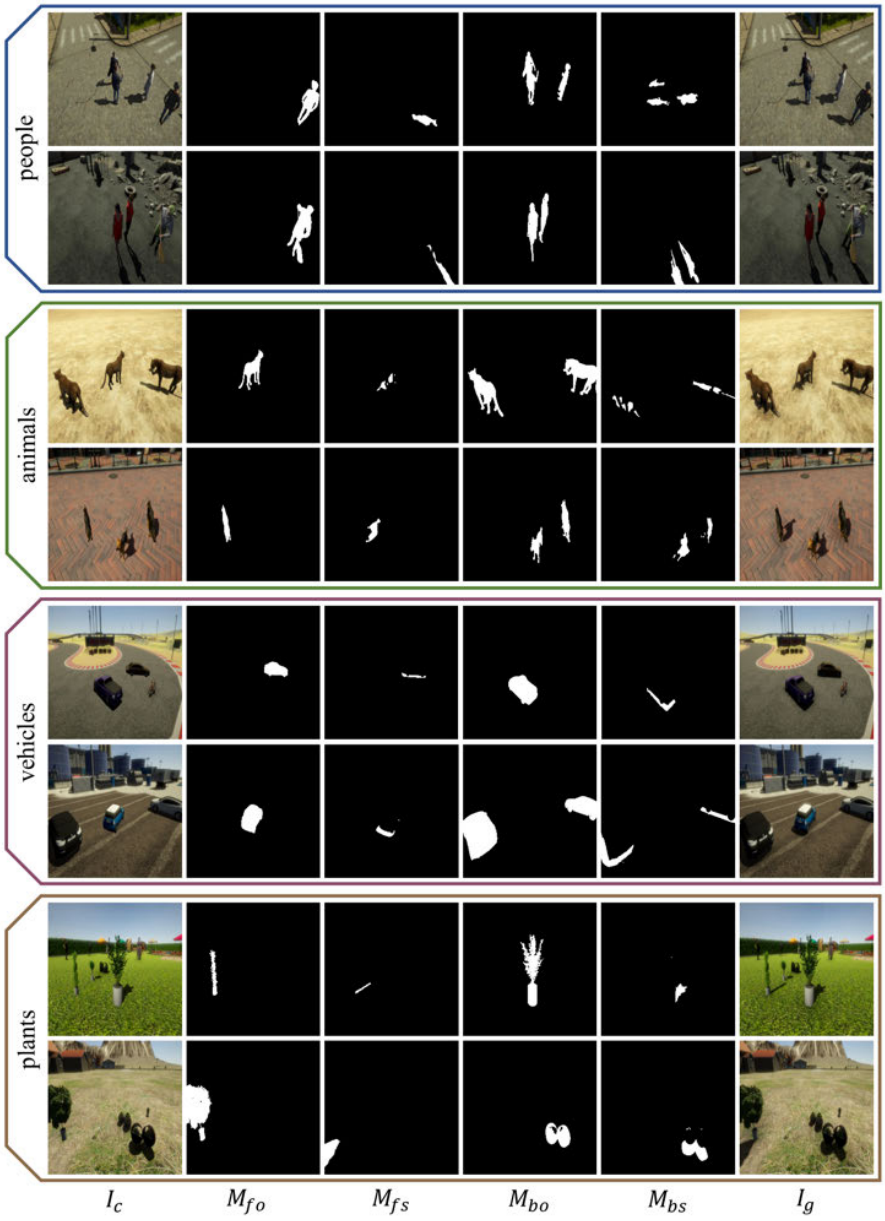


Figure 4.1: Example of a tuple in the RdSOBA dataset [44].

4.2.2 Real-World Datasets: DESOBA and DESOBAv2

To improve real-world generalization, the model was fine-tuned and evaluated using two datasets based on real-world images: DESOBA and DESOBAv2.

DESOBA

The DEshadowed Shadow-OBject Association (DESOBA) dataset [15] is built on the basis of the Shadow-Object Association dataset (SOBA) [50] collection of outdoor scenes. Shadows are manually removed from these images to produce realistic composite-shadow pairs. It consists of:

- 581 test image tuples, with available background objects.
- 31 test image tuples, without background objects.
- Total of 11509 training example tuples.

DESOBA offers high quality, groundtruth shadows while preserving realistic lighting and textures.

DESOBA is used for further fine-tuning and validation because its shadows and shadow masks are derived from real photographs, which tests the ability of the model to handle real textures.

DESOBAv2

DESOBAv2 [28] expands on DESOBA. It includes:

- 28,573 image tuples.

For each image, a shadow is removed using a pretrained inpainting model to generate a shadow-free composite. This resulted in accurate training pairs where the only variation is the presence or absence of a foreground shadow.

The model will be fine tuned using the combined DESOBA and DESOBAv2 datasets, to improve and encourage the model to generalize well to real-world data, and evaluated on the test split in the DESOBA dataset.

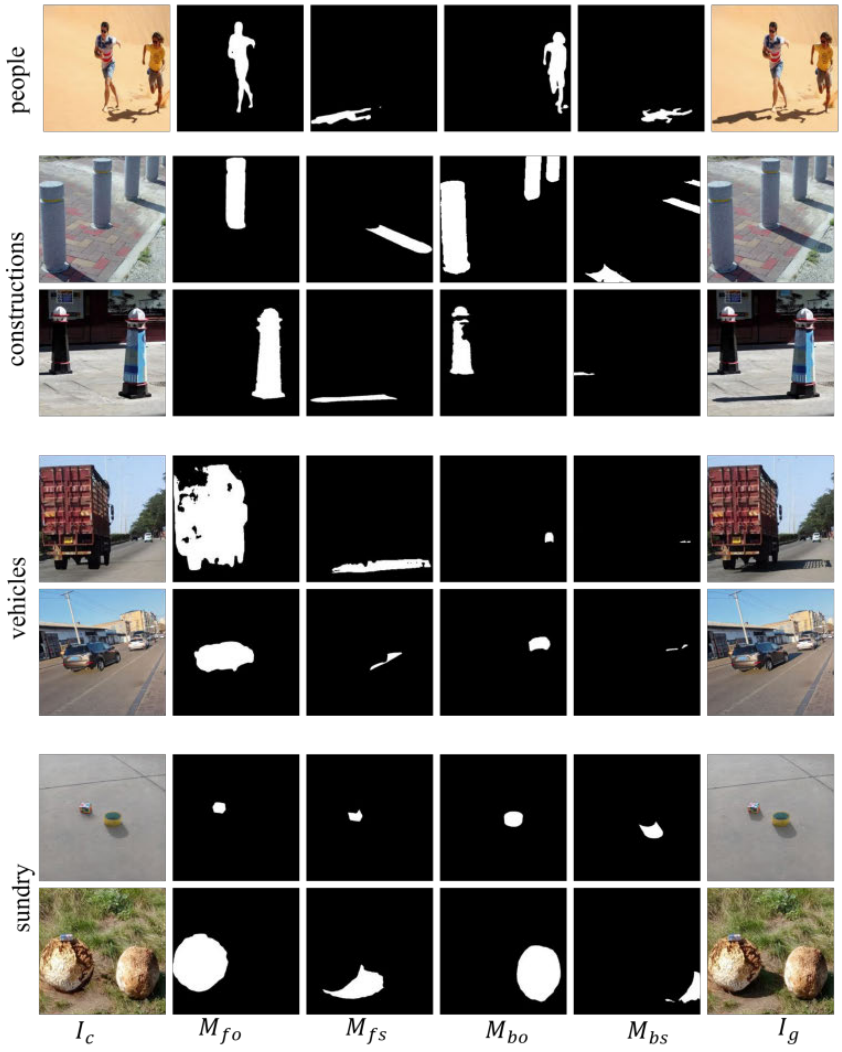


Figure 4.2: Example of a tuple in the DESOBA dataset [44].

4.2.3 Data Preprocessing

All training and validation images are resized to 256×256 pixels. This resolution was chosen as a compromise between maintaining enough detail for shadow quality and keeping computation and memory requirements reasonable. The model is then tested on various resolutions to assess its resolution robustness.

4.2.4 Summary

To summarize, this dataset preparation takes advantage of the scale of synthetic data (RdSOBA) and the authenticity of real data (DESOBA, DESOBAv2) to meet the system requirements for both performance and realism.

4.3 Baseline Architecture: DMASNet

DMASNet was chosen as the baseline, and the ground to build upon, because it is one of the most recent and state-of-the-art approaches to shadow generation. It divides the main task into two smaller clearer ones, mask prediction and shadow filling, making the training process more stable and the architecture easier to understand and extend. Also, when it is compared to other models for shadow generation, it achieves significantly better results in both accuracy and visual realism and appearance, when compared on the real-world dataset DESOBA. Its use of convolutional neural networks (CNNs) also makes it highly efficient and suitable for being used or deployed on CPU-only embedded devices, unlike heavier models such as Stable Diffusion which require GPU acceleration, which fits the requirements discussed in chapter 2. The design of the model makes each part of the model able to focus on a specific task, making the whole system easier to understand and more effective across different scenes and conditions.

4.3.1 Replication Strategy

Since the official implemntaion of DMASNet was not published by the authors, the first part of this research will be replicating the DMASNet architecture and training pipeline, as described in the original paper [44].

This includes:

- Implementing the ResNet-34-based encoders (E_c and E_s) and other modules such as the Box Head, Shape Head, and the decoder D_t.
- Reproducing the original loss functions: Complete Intersection over Union (CIoU) for bounding box prediction, and L_1 losses for mask supervision.

- Training the model on the DESOBA [15] and RdSOBA [44] datasets, following the original training protocols.

This replication serves three main purposes:

1. Ensure a full understanding of the design and training pipeline of DMASNet.
2. Establish a strong and comparable performance baseline.
3. Create an implementation that can be extended to the proposed RRSGNet .

This replication lays the groundwork for the architecture of RRSGNet.

4.4 RRSGNet: Architectural Concept and Design Choices

The proposed **Resolution-Robust Shadow Generation Network (RRSGNet)** builds directly on the strengths of DMASNet while trying to address its limitations discussed in 3.4.2. RRSGNet is designed to offer a more flexible and resolution-robust shadow generation method. To achieve this, some architectural design changes are introduced, and the most significant change is the use of **Fast Fourier Convolutions (FFCs)**. These changes focus on making the model work in real-world scenarios, where images vary in size and scene complexity.

4.4.1 Using FFCs for Resolution Robustness and Global Context

To overcome the fixed size input limitation in existing methods, RRSGNet integrates FFCs into its feature extraction modules. The use of FFCs instead of noraml Convoloutianal layers is mainly inspired by the inpainting model LaMa [42]. As discussed in 3.3.3, FFCs offer two important benefits relevant to this task. First, they enable the network to access global context information early in the network because of their use of the full receptive field in the frequency domain, where in normal CNNs this happens deeper in the network as seen in Figure 3.2. Second, FFCs are naturally robust to changes in input resolution, which makes them particularly suitable for tasks involving varying image sizes.

Based on that, RRSGNet modifies the architecture of DMASNet by either replacing or enhancing some of its normal convolutional layers with FFC layers. More specifically,

these changes are applied to the main encoder modules E_c and E_s, and to the decoder D_t. The use of FFCs in these parts of the network makes RRSGNet learn more adaptable features that are not limited by a fixed image resolution. As a result, the network can more easily handle input images of different sizes without needing to be retrained for each new resolution. Also, FFCs can significantly improve computational efficiency, especially on larger images, by performing convolutions as multiplications in the frequency domain rather than in the spatial domain. This frequency domain operation becomes more efficient as image size increases, offering a huge advantage in scenarios involving high-resolution input images.

Also, due to that the wide receptive field introduces global awareness, that helps the network predict shadow masks more accurately and produce shadows that appear natural and consistent.

Despite these changes, RRSGNet still keeps the core two-stage design of DMASNet, where the first stage focuses on predicting the shadow mask and the second stage fills in the shadowed areas.

4.4.2 Stage 1: FFC-Enhanced Decomposed Mask Prediction

This is the first stage of the RRSGNet architecture, it focuses on predicting the shape and size of the shadow that should be cast by the inserted object. In this stage the model predicts a bounding box and a relative shadow shape. After that a final refinement then brings these predictions together. In this work, the original architecture has been significantly enhanced using Fast Fourier Convolutions (FFCs) to improve resolution robustness and increase the model's ability to understand broader contextual information.

FFC-based Encoder E_c

- **Purpose:** This encoder is designed to extract the visual features from a the input which consists of, the composite image, a mask of the inserted foreground object relative to the background image, mask of existing background objects and a mask for their shadows. They form a 6-channel tensor, 3 for the RGB composite image and one for each mask. By using this input structure, the model can learn how objects interact within the scene, including how their locations and shapes influence can shadow.

- **Architecture Overview:** The encoder has a structure like that of ResNet34, which known for its strength in extracting hierarchical features. However, the standard convolutional layers are replaced with FFC-based blocks to improve context awareness.
- **Layer Details:**
  1. **Initial Layer:** Begins with a wide (7x7) convolution using FFCs, padded to avoid artifacts around the edges. The large 7x7 kernel is to capture features in the image across a wider area. This layer splits output into local and global streams, preparing the features for further FFC layers.
  2. **Subsequent Layers:** These are three layers that follow other, each including several FFC-based residual blocks:
     - The first layer maintains original resolution while deepening the feature representation, allowing the network to learn more complex patterns by combining with the initial layer features.
     - The second and third layers progressively reduce resolution dimensions by factors of 2, while doubling the number of feature channels. Each downsampling step is followed by deeper FFC processing to extract more features at this reduced resolution. This hierarchical downsampling creates a feature pyramid, allowing these layers to learn features at different scales.
  3. **Final Processing:** A final downsampling stage brings features to their lowest spatial resolution, 1/8 of the original size, which is common for ResNet-like backbones, to provide a significant reduction in resolution dimensions while keeping essential information from the input. The output from local and global branches is then merged into a single, combined feature map.
- **Output:** Is a feature map that summarizes both detailed local structure and the global scene context. This output serves as the main input for shadow shape and position prediction steps.

Bounding Box Prediction Head

- **Purpose:** This module predicts the geometric transformations needed to shift from the bounding box of the inserted object to the bounding box of the predicted shadow. It effectively estimates where and how the shadow should fall in the scene.
- **Architecture:** The feature map from the encoder is first refined using a channel attention mechanism, which helps the network focus on the most important, informative features. These refined features are then processed by several convolutional layers, followed by global pooling to create a fixed-size feature vector. Finally, a fully connected layer outputs four values that define the offset and scale of the shadow relative to its object.
- **Output:** A 4-dimensional vector containing the predicted shift and size changes for the bounding box of the shadow.

Shape Prediction Head

- **Purpose:** This part of the network focuses on predicting the actual shape of the shadow. The shape is predicted at a lower resolution and will be scaled up and refined later.
- **Approach:** The same encoder feature map is processed through several convolutional layers to predict the shape mask. A final convolution maps this to a single channel output, and a Tanh activation is applied to ensure the values of the output pixels are in a range between -1 and 1, where 1 is the highest probability that this is a shadow pixel and -1 is the least probability.
- **Output:** A low-resolution, normalized mask representing the general structure of the shadow.

Rough Shadow Mask Construction

- This step combines the outputs of the bounding box and shape prediction heads to form an initial rough version of the mask.

- The shadow shape predicted in the shape head is geometrically warped to fit into the predicted bounding box using affine transformation. This means each pixel here is traced back to where it came from in the lower-resolution shadow shape, predicted in the shape head. This process adjusts the shape so that it is it is properly scaled and positioned within the scene.
- **Output:** A full-resolution shadow mask that roughly approximates the final shadow location and shape.

FFC-Hybrid Decoder for Mask Refinement

- **Purpose:** The main purpose is to produce a final shadow mask that looks realistic and matches the scene precisely.
- **Structure:** This decoder combines the strengths of the global context of FFCs with traditional convolutions:
  1. **Initial Processing:** The output of the encoder is processed at its original resolution using FFCs. This captures any remaining global context needed for refinement.
  2. **Upsampling:** The feature maps are then gradually upsampled using a series of standard convolution layers to recover full spatial resolution.
  3. **Final Composition:** After upsampling, the features are combined with the inserted object mask and the rough shadow mask from the previous step. This combined input is then passed through standard convolutional layers to sharpen and finalize the shadow.
- **Output:** A final high-resolution, refined shadow mask that aligns well with the shape and geometry of the object and fits naturally into the scene.

4.4.3 Stage 2: FFC-Enhanced Shadow Filling

After obtaining the final shadow mask from Stage 1, the next step is to generate the visual appearance of the shadow. This is done by predicting pixel intensities for each pixel in the predicted shadow mask to match the lighting conditions of the scene.

Flexible FFC-Based Encoder with FPN E_S

The goal of this encoder is to learn a multi-scale representation of the lighting conditions of the scenes, textures, and the existing shadows and objects. This information is used to help guide how the new shadow should look. The encoder is based on Fast Fourier Convolution (FFC), which provides strong global context, and it integrates a Feature Pyramid Network (FPN) to capture details at multiple resolutions.

- **Input:** The encoder takes as input the composite image, foreground and background object masks, and the background shadow mask, resulting in a six-channel input. The shadow and object masks provide information that helps the model understand how shadows naturally behave in the scene.
- **Architecture:** The encoder begins with a stem layer that does an initial feature extraction and downsampling. This early reduction in resolution allows the network to quickly capture broader higher-level context information. The input is then passed through several layers, each designed to extract increasingly meaningful features at different levels of detail. Each layer works on a smaller version of the image but learns more meaningful and high-level information, helping the network understand both fine details and the overall structure of the scene.
- **Feature Pyramid Network:** The FPN structure improves the encoder by combining detailed high-resolution features with abstract low-resolution ones. It uses a top-down path and side connections to merge features at different scales, helping the model capture both fine textures and overall lighting.
- **Output:** The result is a multi-scale feature map that preserves important details and context.

Additive Shadow Filling Module

After obtaining the feature map from the previous step, the next step is to generate the final image with the shadow applied.

- **Purpose:** This module takes the refined shadow mask, features from the encoder E_S, and the original composite image as inputs. It then predicts a darkening factor for each pixel in the predicted shadowed areas. This lets the model control shadow

intensity based on both local details and the overall scene context and allows for different light intensity across the shadow

- **Process:**
  - The feature map is first upsampled to match the original image size for proper alignment.
  - These features are combined with the image and the refined shadow mask and used as input to a small convolutional network.
  - The network outputs a per-pixel darkening factor between 0 and 1, where 0 means no darkening and 1 represents the maximum allowed darkening.
- **Applying the Shadow:** The predicted darkening factor is used to compute a pixel-wise offset that darkens the image. This offset is applied only within the shadow region, leaving the rest of the image unchanged. The darkened shadow region is then blended into the original image to produce a natural looking shadow effect.
- **Output:** The final output is the final image with the shadow generated. And because the model adjusts the shadow strength at each pixel, it can make the shadow look right for different surfaces and lighting, making it more realistic.

This two-part setup allows the model to create realistic shadows by combining a broad understanding of the scene, using FFC and FPN, with a flexible method that adjusts shadow pixels based on lighting and surface details.

4.5 Automation of Additional Mask Inputs

To improve the practicality and scalability of RRSGNet, the system incorporates an automated task for predicting the background object mask (M_{bo}) and the background shadow mask (M_{bs}), to remove the need for manual annotation.

Proposed Solution: Using SSIS SSIS , is employed to generate these masks. This model is selected based on the following considerations:

- **End-to-End Detection:** SSIS detects both objects and their corresponding shadows in a single stage, offering a much efficient approach than than deticting the objects alone with a segmentaion model and then detecting the shadow with a shadow detection model.
- **Robust Framework:** It is built on Detectron2, which make it have a strong object detection and instance segmentation backbone, supporting reliable mask generation.
- **Output Compatibility:** The instance masks produced by SSIS can be easily adjusted to fit the input structure required by RRSGNet.

Integration into RRSGNet SSIS will be used as a pre-processing step within the RRSGNet pipeline. It takes the original background image, prior to the insertion of the new object, as input and produces M_{bo} and M_{bs}. These masks are then used by RRSGNet generate shadows.

Expected Outcome Using automated mask prediction allows for a more user-friendly shadow generation system. Although the quality of generated shadows may be affected by possible inaccuracies in SSIS predictions, this tradeoff is acceptable given the significant improvement in usability.

4.6 Training details

The training process for RRSGNet is fully supervised, relying on multiple loss functions that guide the network towards generating realistic shadows. Each loss is chosen to target a specific part of the task.

4.6.1 Loss Functions

- **Reconstruction Loss (L1)**: This ensures that the generated image matches the ground truth closely. L1 loss is used because it produces sharper results than L2 and is more robust to outliers, making it suitable for image generation tasks.
- **Mask Loss (Weighted L1)**: This guides the accuracy of the predicted shadow mask. A weighted version of L1 is applied to give more importance to shadow pixels, which are often not as much compared to non-shadow areas. This helps the model focus on correctly placing the shadow.
- **Perceptual Loss (LPIPS)**: This encourages the output to look more realistic from a human eye perspective. LPIPS compares features from deep networks rather than raw pixels, making it better suited for evaluating the texture of the image and the overall visual quality.
- **Box Loss (CIoU)**: This ensures the predicted bounding box for the shadow is well aligned. CIoU is used because it takes into account overlap, center alignment, and shape consistency, making it more effective than simpler box losses.
- **Shape Loss (L1)**: This helps the network learn the relative shape of the shadow within its bounding box. The loss of L1 is effective here to capture the outline without introducing blur.

4.6.2 Optimization Strategy

The Adam optimizer is used due to its strong performance in deep learning tasks. It adapts the learning rate for each parameter, helping the model to converge quickly and stably.

4.7 Evaluation Strategy

To evaluate RRSGNet and compare it with the baselines, a well-rounded evaluation strategy is used. This includes both objective, numerical metrics and visual assessments to measure shadow quality, mask accuracy, and perceptual realism.

4.7.1 Numerical Metrics: Measuring Performance

The following metrics will be used to evaluate specific aspects of the output of the model:

Image Reconstruction Quality

- **Root Mean Square Error (RMSE)** It measures the average difference between predicted and real pixel values. Chosen because it provides a clear sense of how far off the prediction is per pixel.
- **Peak Signal-to-Noise Ratio (PSNR)** It measures the quality of an image by comparing the signal (the image) to the noise (the error). Chosen because it is widely accepted for image quality tasks.

Both metrics will be calculated over the full image and also just for the shadow region. This helps evaluate both overall quality and how well the shadowed area is rendered.

Shadow Mask Accuracy

- **Balanced Error Rate (BER)** It measures errors in both shadow and non-shadow areas equally. Which is important because shadow regions are much smaller, and without a balanced metric, the result might be good because the effect of the shadow pixels is smaller.

Also calculated for the entire mask and specifically for the shadow region as well, to focus on how well shadow pixels are predicted.

Perceptual Image Quality

- **LPIPS** It measures similarity based on learned features from pre-trained deep networks. It is chosen because it better reflects what looks realistic to the human eye compared to the pixel-wise metrics.

5 Implementation

This chapter details the development environment, step-by-step module implementation, model assembly, and training and evaluation pipeline setup. It also highlights challenges encountered along the way and how initial designs were adapted overcome them.

5.1 Development Environment

5.1.1 Hardware Configurations

There were two main experiments for training RRSGNet on different setups.

Setup 1:

- **GPU:** NVIDIA GeForce RTX 4090
- **VRAM:** 24 GB

Used for the main training with a large dataset.

Setup 2:

- **GPU:** NVIDIA RTX 2000 Ada Generation Laptop GPU
- **VRAM:** 16 GB

Used to train on a smaller dataset and for running fast experiments to test different modules and losses.

5.2 DMASNet Replication

The first step in this project was to replicate the DMASNet architecture as a starting point. This replication served two main purposes, to develop a deep understanding of two-stage architecture of dmasnet and to build a codebase that could be extended into the proposed RRSGNet architecture.

5.2.1 Initial Implementation Steps

The codebase was set up with separate files for each major component such as encoders, decoders, prediction heads, shadow filler, and utility tools like data loading, loss functions, and logging. Then the key parts of the DMASNet architecture were implemented.

- **Stage 1 – Mask Prediction:**
  - Encoder E_c for feature extraction.
  - The Box Head H_b for bounding box regression.
  - The Shape Head H_s for relative shadow shape prediction.
  - The Decoder D_t for refining the coarse shadow mask.
- **Stage 2 – Shadow Filling:**
  - Encoder E_s for extracting features relevant to shadow appearance.
  - The attentive shadow filling mechanism as described in the original paper.

Each part of the network was implemented and tested alone to make sure everything worked as expected. For each part a small test script was written to make sure that the layers were connected correctly and that the outputs had the expected shapes. By checking each part of the network this way, shape mismatches and other small bugs were caught early, making it easier to assemble the full model without unexpected issues.

Data loaders were also implemented to handle the RSOBADataset and DESOBADataset strucure, also loss functions as mentioned in the paper, logging and visualization scripts for inspecting intermediate and final outputs.

5.2.2 Challenges and Adaptations during DMASNet Replication

Problem with the Attentive Shadow Filler: One of the first major issues was from the shadow filling module. According to the DMASNet paper, this component should darken specific regions of the image to simulate shadows by predicting RGB pixel values. However, in practice, the results often looked strange, so instead of simply darkening the masked area, the module would introduce odd colored pixels, in green, red, or blue. Figure 5.1 shows an example of this issue.

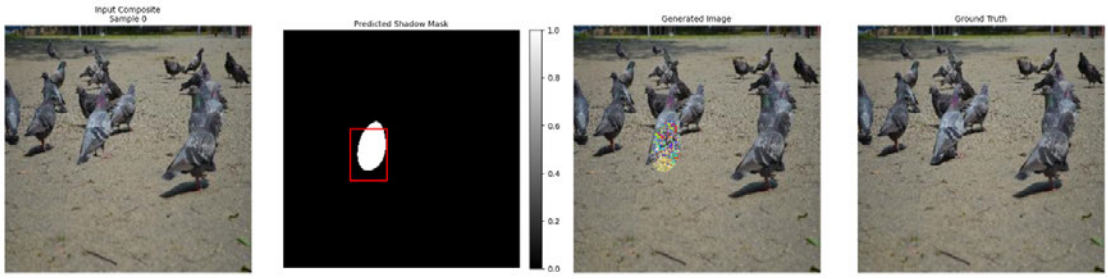


Figure 5.1: Example of the output after 1 epoch.

It did not make sense to have it predict RGB values for the shadow because it just needs to be darker so there is need to mess with the colors of the pixels. So to fix this, a new version of the filler was designed, an *additive shadow filler*. So, instead of predicting full RGB pixel values, this new version learns how much to subtract from the existing pixel brightness. Basically, how much to darken each pixel, without changing the color itself. The new shadow filler predicts a darkening intensity for each pixel and this this is multiplied by a constant *Maximum darkening factor*. This approach kept the shadows consistent with the original scene and looked much more natural Figure 5.3.

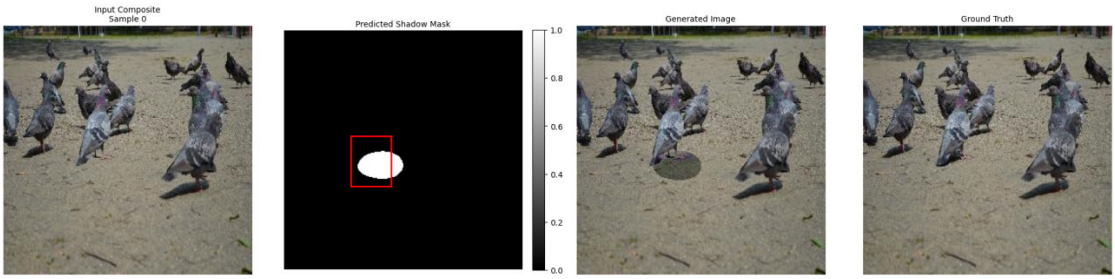


Figure 5.2: Example of the output after 1 epoch, with max_darkening_factor $= -0.7$.

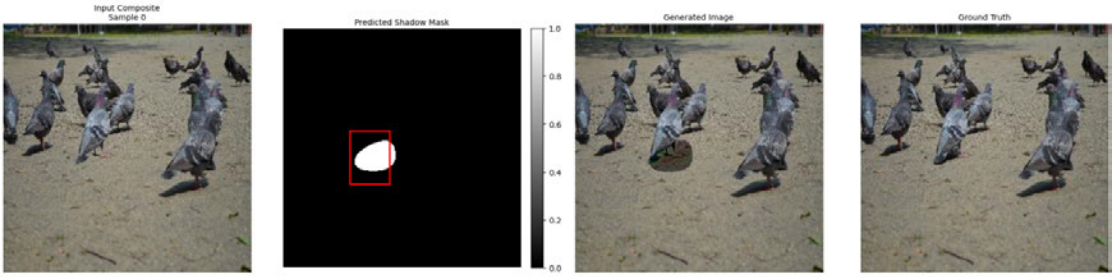


Figure 5.3: Example of the output after 1 epoch, with max_darkening_factor = −0.9.

After testing by overfitting for around 50 epochs, it was decided to keep using the maximum darkening factor as -0.9, to avoid lighter shadow when it should be darker, and -0.9 showed to work better with all shadow intensities.

Limitations of Standard L1 Loss for mask loss: Another issue came from the loss function for the shadow mask. DMASNet originally used a standard L1 loss, but during experiments it was shown that it had a problem. The model could just avoid predicting shadows altogether and still get a decent loss score, so after one epoch it will just predict a completely black shadow mask.

To solve this, a *weighted L1 loss* was implemented. This version places more emphasis on false negatives, which are the pixels that should be shadow but were not predicted as shadow. This made the model more sensitive to actual shadow regions and led to more accurate mask predictions.

5.2.3 Training Setup and Limited Evaluation

The training setup built for DMASNet was also the base for RRSGNet's pipeline. It included:

- Mixed precision training for faster convergence and lower GPU memory usage ,resulting in a much faster training.
- A custom PyTorch dataset class to load the specific structure of the datasets.
- Visualizations of intermediate outputs, shadow masks, and predictions.

- Standard training utilities such as checkpointing, logging, and TensorBoard integration.

Due to the primary focus on developing RRSGNet and limited time, the replicated DMASNet was not fully trained on the entire dataset. Instead, it was tested through batch overfitting on the test dataset. These experiments were enough to confirm that the model worked and that the changes made to the filler and loss were meaningful.

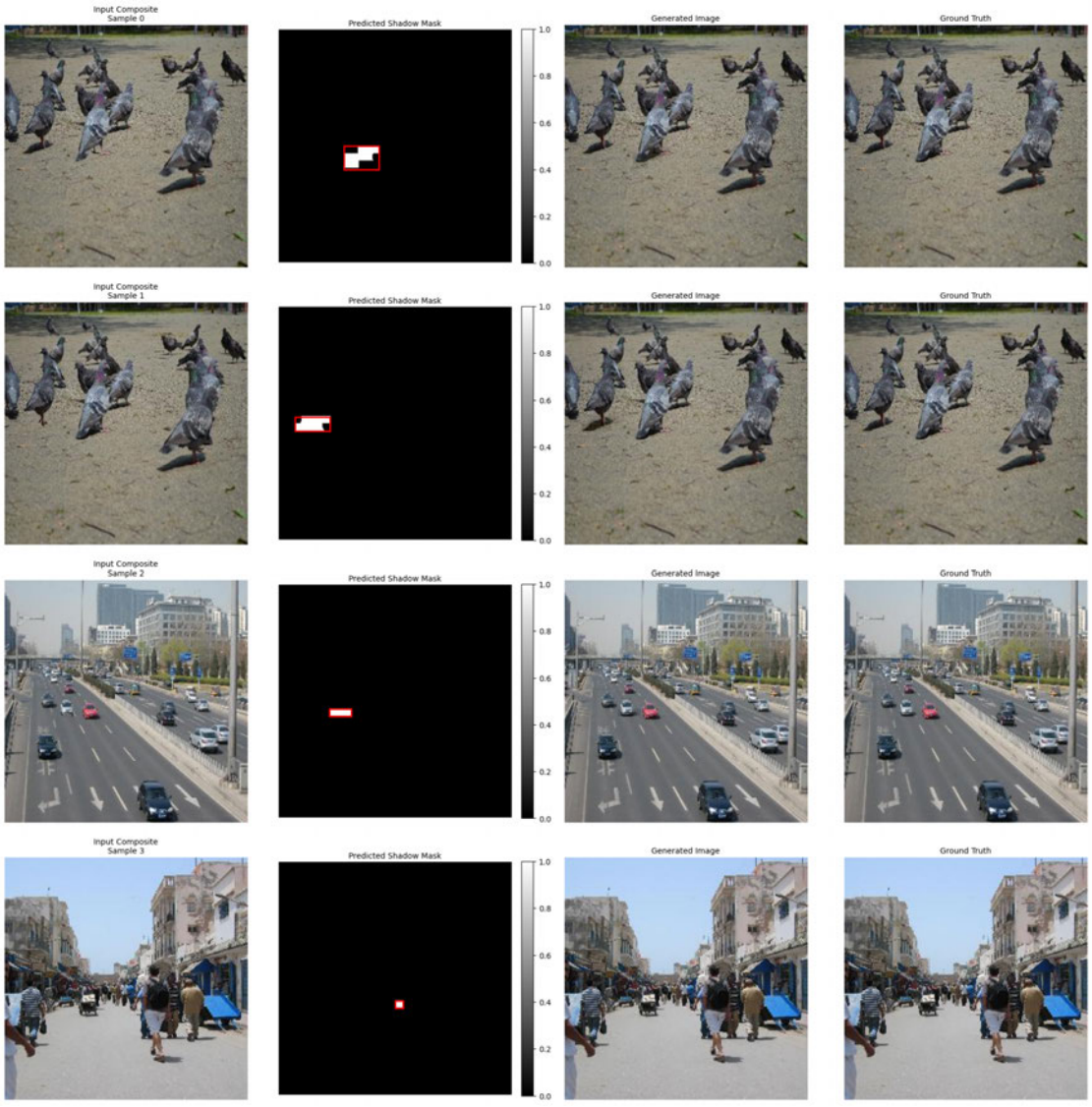


Figure 5.4: Example of DMASNet's output after overfitting for 280 epochs.

5.2.4 Outcome of DMASNet Replication

The DMASNet replication phase, despite its limited training scope, was a crucial step. It resulted in:

1. A functional, but adapted, implementation of the DMASNet architecture.
2. Identification of practical limitations in the original attentive shadow filler and L1 mask loss, leading to effective solutions.
3. A solid codebase for core model components, data loading, training loops, and utility functions, which helped speed up the development of RRSGNet.
4. A deeper understanding of the challenges involved in realistic shadow generation, which directly influenced the design of RRSGNet's FFC-based modules and overall structure.

This adapted DMASNet implementation provided the foundation for developing RRSGNet. It helped guide the design choices and served as a baseline for evaluating the improvements introduced in the new model.

5.3 RRSGNet Implementation

This section explains how the main components of RRSGNet was implemented, based on the design in the previous chapter. Each part of the network was designed to work with inputs of varying resolution and uses FFCs or standard convolutions as needed.

5.3.1 FFC Encoder for Mask Prediction

This encoder replaces DMASNet's CNN with an FFC-based design to generate resolution-robust features F_e from the input tuple (I_c, M_{fo}, M_{bo}, M_{bs}).

5.4 RRSGNet Components Implementation

Following the adaptations made to DMASNet, the key modules of the Resolution-Robust Shadow Generation Network (RRSGNet) were developed. These components are designed to support variable input resolutions and improve shadow prediction performance by integrating Fast Fourier Convolutions . The following subsections outline each major module, its role, and notable implementation details.

5.4.1 FFC Encoder for Mask Prediction

This module serves as the main feature extractor for the shadow mask prediction stage.

- **Implementation:**
  - Based on a ResNet-like structure, using FFCResnetBlocks to process both local and global features.
  - The FFC blocks were configured with specific `ratio_gin` and `ratio_gout` parameters, set as 0.5, to manage the split between local and global features. Local Fourier Unit and gating, which are parts of the FFC block in the existing library, were disabled for simplicity.
  - Downsampling between stages is achieved using `FFC_BN_ACT` layers with a stride of 2, which is an FFC layer followed by a Batch Normalization layer and then an identity activation layer.
  - The encoder is designed to produce an output feature map F_e at a resolution of Height/8 x Width/8 relative to the input.
  - A final ConcatTupleLayer is used to merge the local and global features outputs from the last FFC block into a single tensor output F_e.
- **Notes:** Careful handling of the tuple structure across FFC layers was essential for stable training and correct feature merging.

5.4.2 Box Head

- **Implementation:**
  - Uses a series of 3×3 convolution layers with Instance Normalization and ReLU.
  - A Squeeze-and-Excitation (SE) block is used for channel-wise attention.
  - Global pooling is applied before the final fully connected layer, allowing the head to handle variable feature sizes.

5.4.3 Shape Head

- **Implementation:**
  - Consists of four 3x3 convolutional blocks, which consist of standard convolutional layer, Instance normalization and ReLU.
  - A final 3x3 convolutional layer reduces the feature map to a single channel, producing the final mask where each pixel represents the predicted likelihood of it being shadow.
  - A `Tanh()` activation function is applied to scale the output mask to the range [-1, 1].

5.4.4 FFC-Hybrid Decoder

- **Implementation:**
  - The decoder begins by processing the features F_e using a series of FFCResnetBlocks.
  - Upsampling is performed using standard 2D convolutional layers instead of FFC-based upsampling. This choice helps reduce computational cost while maintaining sufficient spatial detail.
  - After upsampling, a convolutional refinement block combines the decoder's feature map with the rough shadow mask M_{fs_rough} and the foreground object mask M_{fo} to refine the final prediction.

- Before applying the final `Tanh()` activation, a fixed bias derived from M_{fs_rough} is added. This bias subtly boosts the confidence in regions already predicted as shadow, helping the network focus on likely shadow areas.

5.4.5 FFC Encoder for Shadow Filling

- **Implementation:**
  - The core architecture uses FFCResnetBlock and FFC_BN_ACT layers similar to FFCEncoderEc.
  - Then four stages of FFC blocks further process features, with downsampling between each stage, resulting in features at H/32 x W/32 in the deepest FFC layer.
  - A Feature Pyramid Network (FPN) structure is implemented on top of the FFC backbone.
  - Top-down upsampling combines coarse and fine features.
  - Final 3×3 convolution produces the final feature map F_s at H/4 resolution.

5.4.6 Flexible Additive Shadow Filling

- **Implementation:**
  - Features F_s are upsampled to match the resolution of I_c.
  - A small CNN predicts a per-pixel darkening factor α (0 to 1) using F_s, I_c, and M_{fs}.
  - An additive offset is calculated as $\alpha \times$ `max_dark_offset`, which is -0.9, and added to I_c resulting in I_{dark}.
  - The final output is then I_c without the masked area added to it is I_{dark} of only the masked area.

5.5 RRSGNet Model Assembly

5.5.1 Overview

- **Stage 1: Shadow Mask Prediction.** The input image I_c is combined with object and background masks (M_{fo}, M_{bo}, M_{bs}) and passed through the encoder E_c to extract features F_e. The two heads then predict a rough shadow shape M_s and a bounding box. These are used to construct a rough shadow mask M_{fs_rough} using geometric warping to fit the shadow in the box. A decoder then refines this into the final predicted shadow mask M'_{fs}.
- **Stage 2: Attentive Shadow Filling.** The same inputs are encoded again to produce the features F_s, which, together with M'_{fs} and I_c, are used to calculate an additive darkening offset. This offset is applied to shadow regions in I_c to generate the final image I_g.

5.5.2 Forward Pass Summary

Given an input I_c and masks (M_{fo}, M_{bo}, M_{bs}), the network does the following:

1. Merge the inputs together and encode them through E_c to produce feature map F_e.
2. Predict the bounding box r and the shadow shape M_s, through passing F_e to the box head and the shaper head respectively.
3. Generate M_{fs_rough} by placing and wrapping the predicted shadow shape M_s into the bounding box.
4. Then this is passed to the decoder to refine M_{fs_rough} to produce the final shadow mask M_{fs}.
5. The inputs are encoded again through E_s to produce F_s.
6. Use F_s, M_{fs}, and I_c to calculate and apply darkening for the predicted shadow area, resulting in a fila image I_g with a show casted for the insterted object.

5.6 Utilities

A range of utilities were implemented to support the training, evaluation, and visualization.

5.6.1 Data Loading and Preprocessing

This handles the loading of RdSOBA and DESOBA datasets, loading tuples of images and masks: $(I_c, M_{fo}, M_{bo}, M_{bs}, I_g, M_{fs})$. This includes:

- **Resizing:** All images and masks were resized to a specific resolution for training.
- **Normalization:** Input images were normalized to the range $[-1, 1]$, suitable for dealing with Tanh activation functions.

5.6.2 Experiment Management and Logging

- **Logger):** The Logger class tracked experiments via TensorBoard, also logging scalar metrics such as losses, scores and the visual outputs. It also handled the basic model checkpointing, saving both the latest model state and the best one with the highest score in chosen metric, and supported resuming from checkpoints with the complete state of the training.
- **Checkpoint Manager:** Was developed after the logger to save more checkpoints if needed. It saves only the top-k checkpoints based on a chosen validation metric such as LPIPS, PSNR, etc. This is done using a min or a max heap, depending on the metric, whether the higher the better or if the lower the better.

5.6.3 Metrics Calculation

The metrics mentioned in the Concept chapter were implemnted here and variations were made for the specific shadow patches, and these are s_psnr, s_rmse and s_ber.

5.6.4 Visualization

This is to visualize the model performance througout the traing. It includes:

- **Comparison Grids:** A side by side visualizations of inputs, predicted outputs, and ground truths, with the bounding boxes overlayed where relevant.
- **Training Curves:** Plotted trends of key metrics across epochs, helping to track the training process. These images would update each epoch without the need to create extra images.
- **Mask Detail:** For debuggin to check the shadow prediction process, this displays intermediate mask predictions, decoder features, and refined masks for given sample.
- **Metrics Archiving:** Logs all training metrics to a JSON files, that gets updated each epoch as well.

5.7 Training Pipeline Implementation

The core logic is structured around a training loop for epochs. Each epoch includes:

1. **Training Phase:** The model is set to training mode. Then for each batch in the epoch:
    - Input tensors ($I_c, M_{fo}, M_{bo}, M_{bs}$) and ground truth targets (I_{g_gt}, M_{fs_gt}) are loaded to the appropriate device (CPU/GPU).
    - A forward pass through RRSGNet produces the predictions most importantly the generated image (I_g), final shadow mask (M_{fs}) and the predicted bounding box (B_{s_pred})
    - The Loss module calculates the total loss based on these outputs and the targets.
    - Gradients are calculated using backpropagation.
    - Model parameters are updated using the Adam optimizer.
    - Batch loss values are then logged to TensorBoard for real-time monitoring.

Training is stabilized and accelerated using:

- **Automatic Mixed Precision:** For memory efficient and faster training.
- **Gradient Accumulation:** To effectively simulate larger batch sizes if needed. This means The weights will not get updated until a chosen number batches has been seen by the model.
- The training script also supports training on multiple GPUs if available.

2. **Validation Phase:** Validation runs are triggered at intervals defined in the configuration file. During this phase:
   - The model is set to evaluation mode, so gradient computation is disabled.
   - Predictions are generated for each batch from the validation dataset.
   - Evaluation metrics, including RMSE, PSNR, BER and LPIPS, are computed using the Metrics module.
   - An average of these metrics across the epoch is calculated and logged.
   - A sample batch is saved for visualization.

After each validation cycle, model checkpointing is managed by the logger and the checkpoint Manager.

The Visualization utility is then used to generate visualizations of the model outputs, including comparison grids and detailed mask progression views. These visual assets are logged to TensorBoard and saved locally for inspection.

Finally, training curves and a complete history of metrics are exported to a JSON file, providing a detailed and a summary of the training run.

5.7.1 Configuration Management

All major hyperparameters and settings for training and evaluation are in a single Python module. This module defines a Config class, which acts as a global configuration object that other modules across the codebase can import and use. The following is managed by the config file:

- **Data Paths:** Specifies directories for datasets, for training and validation.
- **Experiment Settings:** Includes setting a name for the experiment, setting a path for the log, and option for setting a path for resuming training from a specific checkpoint.
- **Training Parameters:** Defines important training parameters such as the number of epochs, batch sizes, learning rates , and optimizer beta values. Also choosing a specific size to resize images for training and validation and setting the gradient accumulation steps.
- **Loss Weights:** Setting weights for each loss component.
- **Checkpointing:** The number of checkpoints to be saved and based on which metric.
- **Logging and Visualization:** The frequency of running validation, and generating visual outputs.
- **Performance and Hardware Settings:** Setting the number of CPU workers for loading the data, and flags to control GPU usage, multi-GPU support, model compilation and mixed precision.

5.8 Inference Pipeline Implentation

This covers the implementation of the inference pipeline with automated background masks detection.

5.8.1 Overview

- A shadow segmentation model (SSIS) to analyze the background and extract relevant shadow and object masks.
- The proposed RRSGNet model to generate realistic shadows for newly placed foreground objects.

This setup allows a user to load a background image, place a new object anywhere in the scene, and then generate a composite image with plausible shadows.

5.8.2 Background Analysis with SSIS

When a background image is first loaded, it is processed using a pre-trained SSIS model. This model predicts two important binary masks:

- **Mbo** – binary mask for object shadows already present in the scene.
- **Mbs** – binary mask shadows of the shadows of the background objects.

5.8.3 Placing a Foreground Object

While the background image is being analyzed, a user can place an object anywhere on the image. The object is then represented by a binary mask (`Mfo`) that marks its location. This mask is the same size as the background image (`Ic`), making it easy to align and combine the inputs.

5.8.4 Shadow Generation with RRSGNet

With the composite image, inserted object mask, and background objects and shadows available, the pipeline forwards the data to RRSGNet. Before the model processes the inputs, all tensors are resized and padded if needed to ensure compatibility with the network's input dimensions, meaning that resolution is divisible by 32, so there would not be a mismatch after downsampling and the upsampling.

RRSGNet then generates a realistic shadow corresponding to the object placement.

5.8.5 Usage Summary

The full process involves three steps:

1. Load and process the background image using SSIS to extract `Mbo`, and `Mbs`.
2. Add a foreground object then the mask `Mfo` and *Ic* will be generated.
3. Generate a shadow with RRSGNet, resulting in an output that can be saved, or edited.

6 Experiments and Results

This chapter details the experimental setup, training configurations, dataset splits, and results for the DMASNet replication and the proposed RRSGNet model. Experiments were conducted in several phases to iteratively validate design decisions and measure performance under different constraints.

6.1 DMASNet Replication: Overfitting

To validate the experimental setup and ensure correct implementation, DMASNet was replicated and run in a batch overfitting using the DESOBA test set. This type of test is used to confirm whether a model can perfectly memorize a small amount of data, indicating that the training pipeline and loss computations are functioning correctly.

The overfitting was successful, showing that the model was implemented correctly. No further training of DMASNet was conducted beyond this due to time constraints.

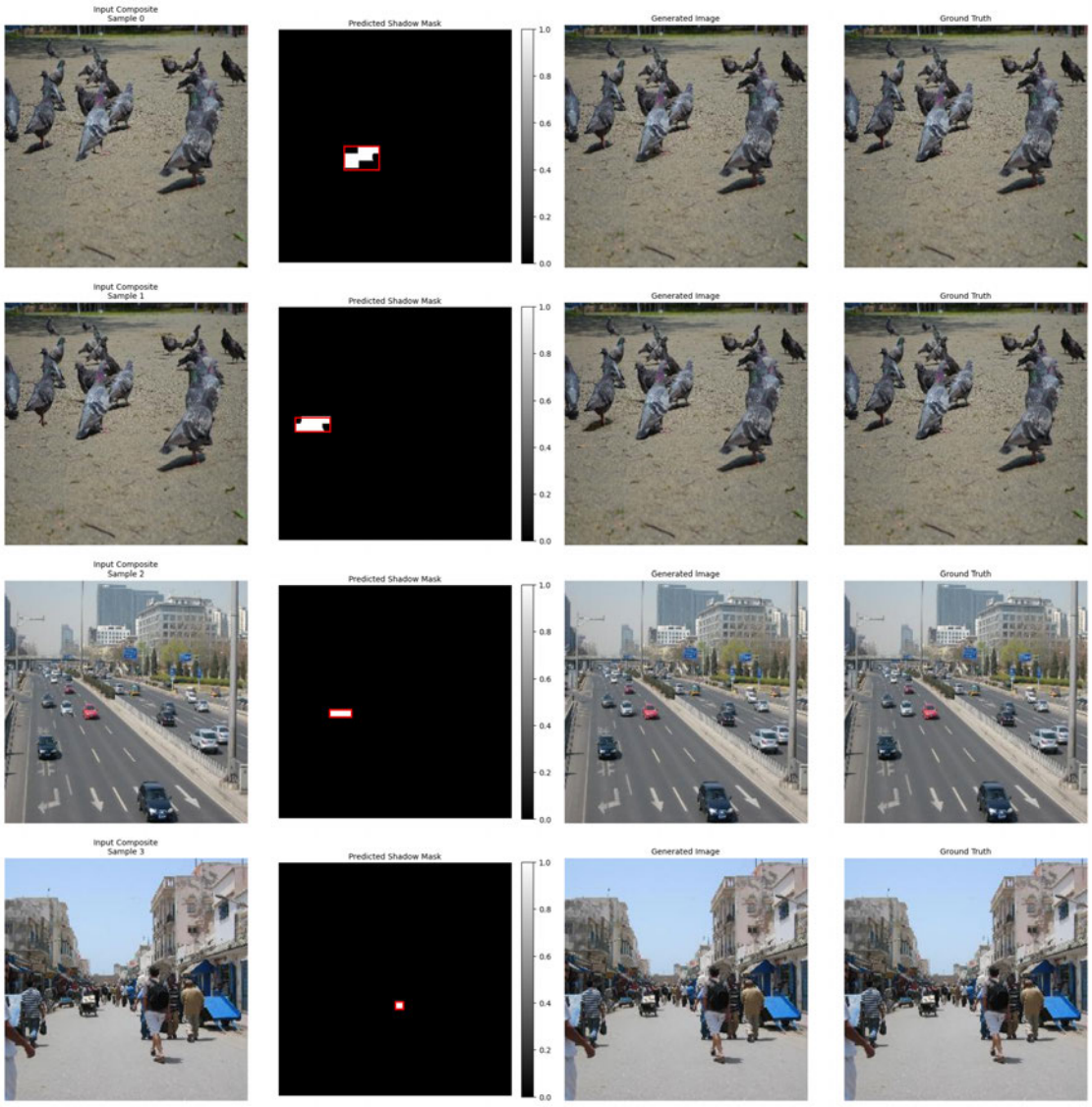


Figure 6.1: Example of DMASNet after overfitting for 280 epochs.

6.2 RRSGNet Experiments

The RRSGNet was trained in two different setups with different conditions.

6.2.1 First Experiment

In the first experiment, training was done on an RTX 2000 GPU with a batch size of 10. The dataset used was a combined version of **DESOBA v1 and v2**, while evaluation was performed on the DESOBA test set. Under these conditions, the model showed minimal improvement and poor learning, confirming the limitation of using such a small dataset for this task. These results highlighted the need for the larger training set and how much of a difference it makes.

Figure 6.2: Example of RRSGNet (Experiment 1) after 270 epochs.

6.2.2 Main Training Experiment

This experiment was conducted on an RTX 4090 which allowed for handling larger data and faster training. This experiment is divided into two parts, first is pretraining the

model using the synthetic dataset RdSOBA[44], and the second is fine-tuning the model on real-world data using the combined dataset of DESOBA v1 and v2[15][28].

The original DMASNet's paper pretrained their model for 50 epochs on the RdSOBA dataset and fine-tuned it further for 1000 epochs with only DESOBA v1. Since DESOBA v1 is approximately 10000 images, and the combined dataset of DESOBA v1 and v2 is approximately 40000 images, so it was estimated for RRSGNet to acheive comparable results to DMASNet after 250 epochs of fine-tuning on DESOBA v1 and v2.

Pretraining on RdSOBA

- **Dataset:** RdSOBA
- **Epochs:** 50
- **Batch Size:** 16
- **Precision:** Mixed precision for faster training.
- **Loss weights (λ):** All set to 10, to avoid early convergence.
- **Learning rate:** 0.0001 common practica and recommended by DMASNet's paper[44]

PSNR	BER	RMSE	LPIPS
31.54	18.59	7.39	0.011

Table 6.1: Average results of the metrics on validation set after pretraining.

Fine-Tuning on Desoba Real-World Data

- **Dataset:** Desoba v1 and v2
- **Epochs:** 500
- **Batch Size:** 10
- **Precision:** Mixed precision disabled.
- **Loss weights (λ):** All set to 10
- **Learning rate:** 0.00003

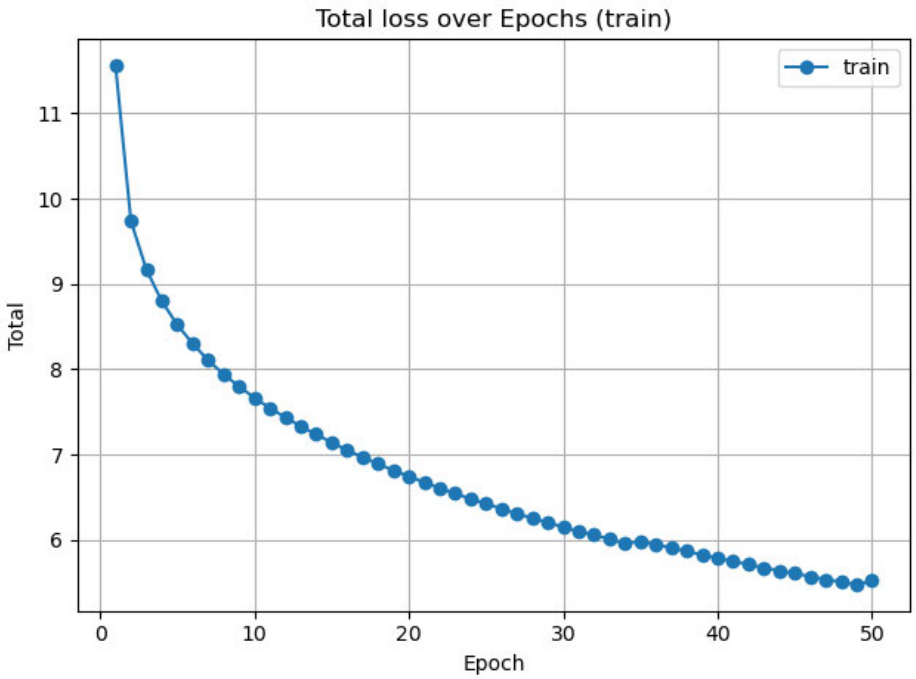


Figure 6.3: Loss curve from pretraining on the RdSOBA dataset for 50 epochs

Mixed precision had to be disabled in this session due to frequent instability. This is because the mixed percision support for fast fourier transform, which is used in the FFC blocks is currently experimental in PyTorch. It was also unstable in the pretraining run but would eventually work when resuming or restarting the training process, but in the fine-tuning run it would not work properly at all, causing GPU memory leaks and therefore leading to out of memory errors.

Using full precision requires more memory that is why the batch size here is set to 10 according to the equation $\mathrm{LR}_{\mathrm{new}} = \mathrm{LR}_{\mathrm{old}} \times \frac{\mathrm{Batch}_{\mathrm{new}}}{\mathrm{Batch}_{\mathrm{old}}}$ [11].

Checkpoint selection was made at **epoch 441**, where the model's outputs showed the best perceptual quality and realism.

PSNR↑	**BER↓**	**RMSE↓**	**LPIPS↓**
34.88	16.81	5.79	0.0076

Table 6.2: Average results of the metrics on validation set (256x256) after fine-tuning at epoch 441.

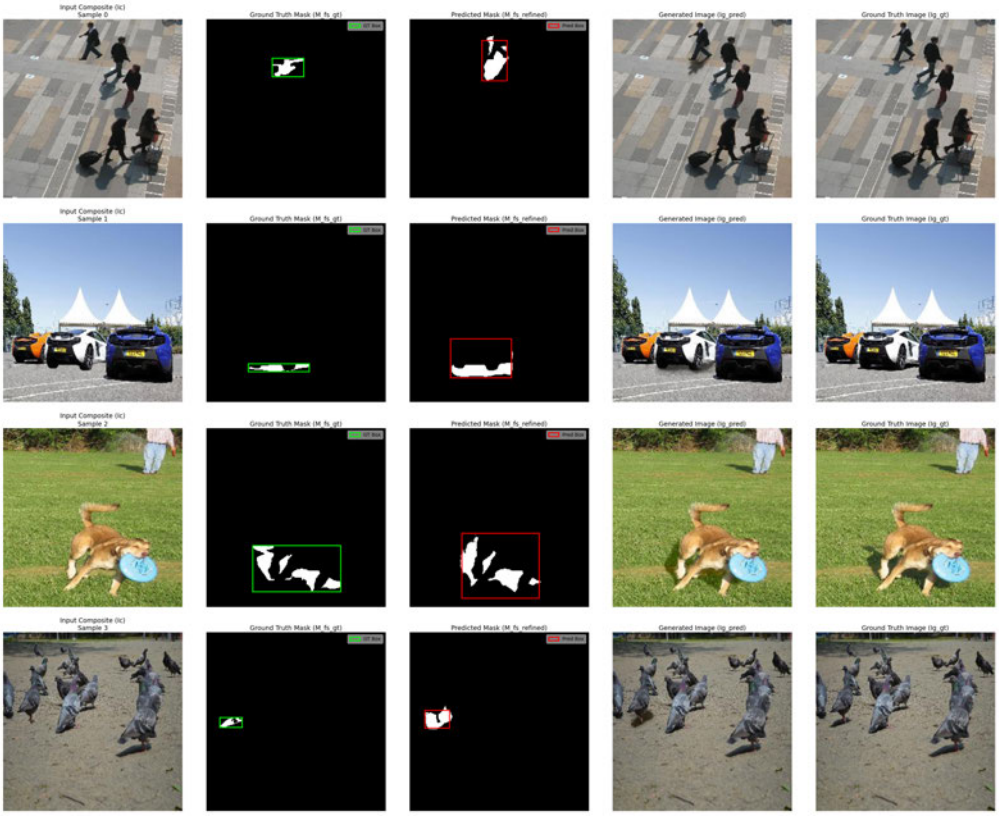


Figure 6.4: Sample predictions from RRSGNet after pretraining

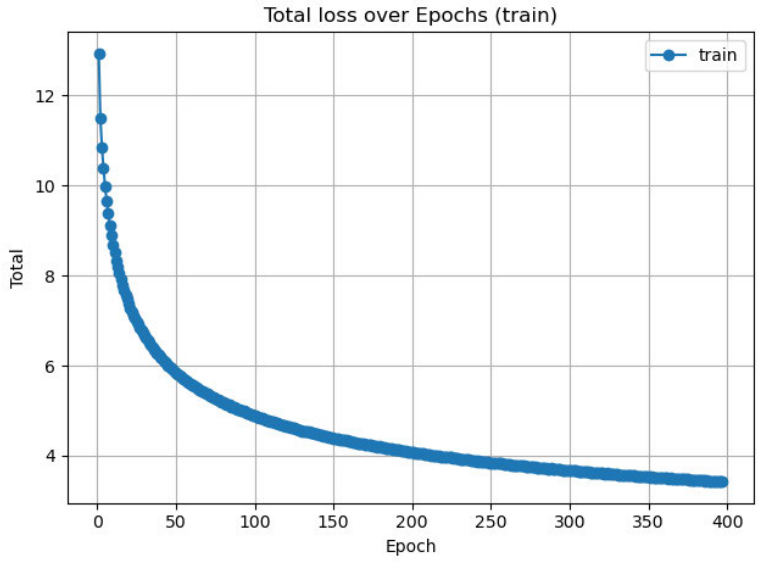


Figure 6.5: Loss curve after fine-tuning on the DESOBAv1v2 dataset for 447 epochs

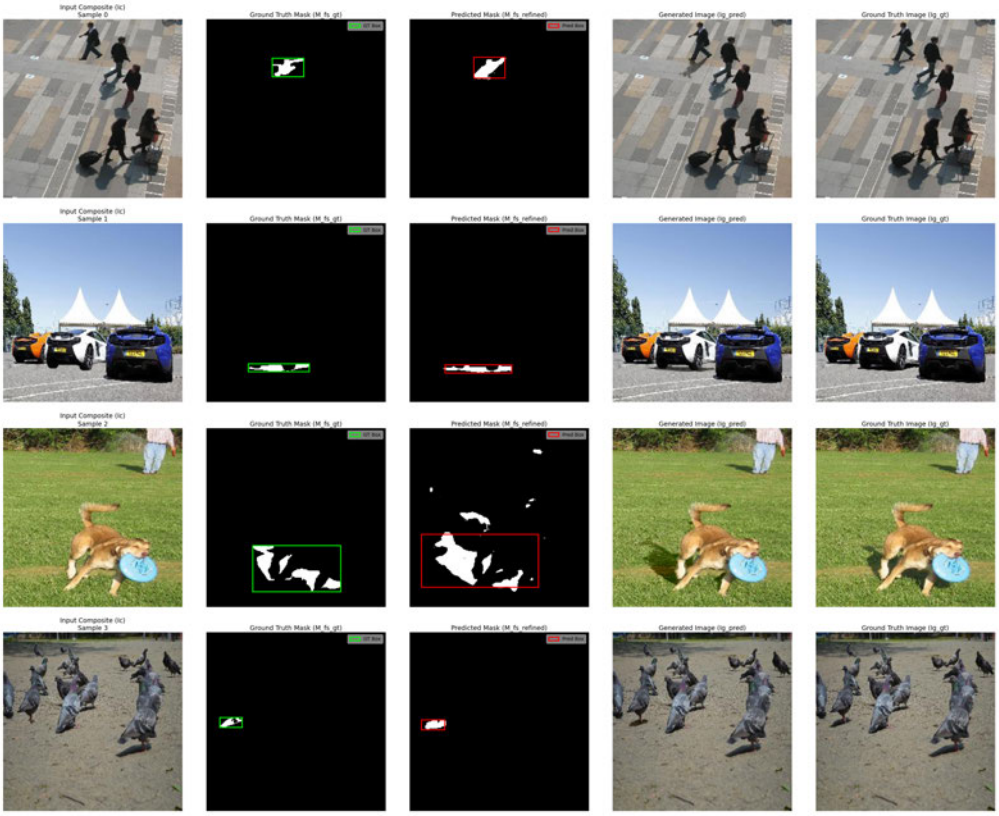


Figure 6.6: Sample predictions from RRSGNet at epoch 441

6.3 Evaluation and Comparison

To measure performance, several standard image quality metrics were calculated for RRSGNet and compared to other baselines. All models were evaluated on the Desoba test set using PSNR, SSIM, FID, and LPIPS.

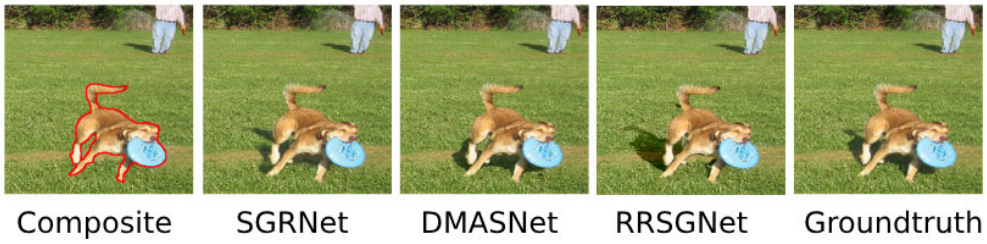


Figure 6.7: Example results of different models

Model	RMSE↓	PSNR↑	BER↓
RRSGNet	5.79	34.88	16.81
SGRNet	4.676	36.898	27.233
DMASNet	4.703	37.149	24.295

Table 6.3: Performance comparison on DESOBA test set (256x256)

6.4 Testing at Different Input Resolutions

To assess the robustness of the model to different resolutions, it was tested across a range of input resolutions. This is crucial for applications where the image dimensions may not match the training setup.

Model	RMSE↓	PSNR↑	BER↓	LPIPS↓
128×128	5.68	35.28	25.89	0.0090
256×256	5.79	34.88	16.81	0.0076
512×512	5.81	35.10	22.44	0.0083
1280×720	6.14	34.47	26.29	0.0088
1920×1080	5.86	35.63	27.34	0.0093

Table 6.4: RRSGNet performance across different resolutions

The results show that the model maintains consistent output quality across most scales, with only minor degradation observed at the highest resolutions. This shows strong generalization capabilities beyond the resolution used during training.

7 Discussion

7.1 Replication of DMASNet

Replicating the DMASNet architecture [44] on a small subset of the DESOBA dataset confirmed the correct implementation and established a functional training pipeline. This step provided essential insight into the architecture's mechanics and served as a foundation for the proposed RRSGNet.

7.2 RRSGNet Performance Evaluation

Initial RRSGNet training using only the DESOBA v1 and v2 datasets, under limited computational resources, showed minimal progress. This highlighted the insufficiency of small datasets for training deep networks in shadow generation and motivated the pre-training approach using the larger, more diverse RdSOBA dataset.

7.2.1 Pre-training and Fine-tuning

Pre-training on RdSOBA allowed RRSGNet to learn a broad representation of shadow patterns. Loss curves showed consistent improvement, and early visual results showed reasonable shadow formation, although synthetic in appearance. Fine-tuning then on DESOBA v1 and v2 significantly improved realism and generalization for real-world data, as shown in the visual outputs and the improved evaluation metrics. This has validated the training strategy and emphasized the importance of large-scale pre-training followed by domain-specific fine-tuning.

7.2.2 Comparison with Baseline Models

RRSGNet delivered mixed results compared to DMASNet and SGRNet. Although RRSGNet did not surpass them in RMSE and PSNR, it performed better in BER, indicating stronger accuracy in mask placement. The FFC-based architecture contributed to this improvement due to its enhanced global context capabilities. LPIPS scores further supported the perceptual quality of RRSGNet's outputs, although a direct comparison for this metric with other models was not available as it was not published.

7.2.3 Generalization Across Resolutions

The main design goal was to maintain performance across varying image resolutions without retraining. Evaluation showed that RRSGNet remained stable from low resolutions (128×128) to full HD (1920×1080). LPIPS remained nearly constant, while RMSE and PSNR fluctuated slightly, and BER showed some changes at extreme resolutions. These results confirmed the advantage of using FFCs to achieve resolution robustness, especially when compared to traditional CNN-based models.

7.2.4 Automated Mask Prediction

RRSGNet was tested using both ground-truth and predicted masks (M_{bo} and M_{bs}). Results suggested that the model maintained reasonable performance with predicted masks, though slight degradation was observed. This indicates a degree of robustness to auxiliary mask errors, an important trait for practical deployment without ground-truth data.

7.3 Limitations

Despite its strengths, RRSGNet faces several limitations:

- **Dependence on Auxiliary Masks:** The quality of predicted M_{bo} and M_{bs} masks affects the final output. Inaccurate inputs can badly affect shadow placement and realism.

- **Pixel-level Accuracy:** Although RRSGNet achieves high perceptual quality, it does not outperform baselines in pixel-wise metrics such as RMSE or PSNR. This trade-off is typical when optimizing for realism rather than strict pixel correspondence.
- **Sensitivity to Shadow Orientation:** Most training data featured objects casting shadows downward. As a result, when tested on bird's-eye images where shadows should appear to the side or above, the model still tends to place shadows underneath. This bias reduces generalization across viewing angles. A potential remedy involves augmenting the dataset with rotated versions of each image to expose the model to all shadow directions.
- **Increased Training Requirements:** The FFC blocks, while offering greater global context awareness and resolution generalization, are architecturally more complex than standard convolutional layers. As a result, RRSGNet may require longer training durations or larger datasets to converge effectively, potentially explaining why DMASNet and SGRNet outperform it on some low-level metrics despite simpler designs.
- **Failure Cases in Complex Scenes:** In certain challenging scenarios, such as complicated structures, or bad lighting, the generated shadows can contain artifacts. Further refinement and a more diverse dataset may help address these issues.

7.4 Relation to Existing Work

RRSGNet builds upon DMASNet's two-stage shadow generation pipeline but replaces core components with FFCs to enable resolution robustness. Unlike DMASNet, which is limited to a fixed input size, RRSGNet processes varied resolutions with minimal degradation. In addition, the improved BER indicates improved mask prediction, probably due to the global context provided by the FFCs.

The use of FFCs is inspired by their success in LaMa [42], where they contributed to resolution-robust inpainting. RRSGNet extends this architectural principle to shadow synthesis, a more structurally complex task. By incorporating FFCs into a structured generation pipeline, RRSGNet bridges concepts from inpainting and compositional image understanding.

Overall, RRSGNet contributes a resolution robust and visually strong approach that offers practical benefits for real-world image editing and augmented reality applications.

8 Conclusion and Future Work

This chapter summarizes the key outcomes of the thesis and outlines possible directions for future research and development based on the limitations and observations discussed in Chapter 7.

8.1 Conclusion

This thesis introduced **RRSGNet**, a novel model for generating realistic shadows in composite images that operates effectively across a wide range of input resolutions by integrating Fast Fourier Convolution (FFC) blocks. These blocks enable the model to capture both local details and global context, enhancing flexibility and robustness compared to traditional convolution-based methods like DMASNet.

The project addressed four main goals:

1. **Replication of DMASNet:** The baseline replication was completed, although it took approximately six weeks instead of the four planned. This delay was due to the complexity of the DMASNet architecture and practical challenges with its shadow filling mechanism and loss functions, necessitating adaptations such as the additive shadow filler and weighted L1 loss. This replication provided essential understanding and a solid foundation for the project.

2. **Design and Implementation of RRSGNet:** Building on the DMASNet framework, RRSGNet was implemented efficiently within about two weeks by replacing traditional convolution blocks with FFCs to enable resolution robustness. The model was pre-trained on the large synthetic RdSOBA dataset and fine-tuned on real-world DESOBA and DESOBAv2 datasets. It produced visually realistic shadows and performed competitively, especially on structural metrics like BER. Most

importantly, RRSGNet handled varying input resolutions without retraining or resizing, fulfilling a key practical requirement.

3. **Integration of Automated Mask Prediction:** The system included automated background mask prediction using a pre-trained SSIS model, reducing reliance on manual mask inputs and enhancing usability. RRSGNet showed reasonable performance using these automatically predicted masks.

4. **Visual Evaluation:** Due to extended time spent on DMASNet replication and RRSGNet training, the planned user study could not be completed within the project's timeframe.

RRSGNet's ability to process images at different resolutions without retraining presents a significant advantage for real-world applications. Nonetheless, challenges remain, such as sensitivity to the quality of predicted masks and not generalizing well to images where the shadow should not be beneath the object. These areas suggest promising directions for future work.

Overall, this thesis establishes RRSGNet as a strong foundation for resolution-robust shadow generation.

8.2 Future Work

There are several ways this work can be expanded and improved:

- **Data Augmentation for Better Shadow Direction:** The training datasets mostly contain objects with shadows cast directly beneath them. This has caused the model to consistently cast shadows downward, even in situations where a different direction would be more realistic, such as in a top-down view. Rotating the training data in different directions could help the model learn a wider variety of shadow placements.

- **Reducing Dependence on Input Masks:** The quality of predicted M_{bo} and M_{bs} masks can affect how realistic the final shadow looks. A useful improvement would be training the model to work without these masks, relying instead only on the composite image and the foreground object mask (M_{fo}). This could make the model more robust and easier to use in practice.

- **Larger and More Diverse Training Sets:** Since FFCs allow the model to work across different resolutions, training on a larger and more varied dataset could help it generalize even better. This might also improve pixel-level performance metrics while keeping the benefits seen in perceptual quality.

Overall, RRSGNet lays the groundwork for more flexible and realistic shadow generation. With further refinement in training methods and data preparation, it could become a more powerful tool for applications in image editing, augmented reality, and beyond.

Bibliography

[1] Abadi, Martín ; Barham, Paul ; Chen, Jianmin ; Chen, Zhifeng ; Davis, Andy ; Dean, Jeffrey ; Devin, Matthieu ; Ghemawat, Sanjay ; Irving, Geoffrey ; Isard, Michael ; Kudlur, Manjunath ; Levenberg, Josh ; Monga, Rajat ; Moore, Sherry ; Murray, Derek G. ; Steiner, Benoit ; Tucker, Paul ; Vasudevan, Vijay ; Warden, Pete ; Wicke, Martin ; Yu, Yuan ; Zheng, Xiaoqiang: *TensorFlow: A system for large-scale machine learning*. 2016. – URL https://arxiv.org/abs/1605.08695

[2] Alvi, Farooq: PyTorch vs TensorFlow in 2025: A Comparative Guide of AI Frameworks. In: *AI Careers* (2024). – URL https://opencv.org/blog/pytorch-vs-tensorflow/

[3] Barron, Jonathan T. ; Malik, Jitendra: *Shape, Illumination, and Reflectance from Shading*. 2020. – URL https://arxiv.org/abs/2010.03592

[4] Chai, Tianfeng ; Draxler, R.R.: Root mean square error (RMSE) or mean absolute error (MAE)?– Arguments against avoiding RMSE in the literature. In: *Geoscientific Model Development* 7 (2014), 06, S. 1247–1250

[5] Chen, Bi-Shiun ; Kae, Arcot: Toward realistic image compositing with adversarial learning. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, S. 8415–8424

[6] Chi, Lu ; Jiang, Borui ; Mu, Yadong: Fast Fourier Convolution. In: Larochelle, H. (Hrsg.) ; Ranzato, M. (Hrsg.) ; Hadsell, R. (Hrsg.) ; Balcan, M.F. (Hrsg.) ; Lin, H. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 33, Curran Associates, Inc., 2020, S. 4479–4488. – URL https://proceedings.neurips.cc/paper_files/paper/2020/file/2fd5d41ec6cfab47e32164d5624269b1-Paper.pdf

[7] Cong, Wenyan ; Zhang, Jianfu ; Niu, Li ; Liu, Liu ; Ling, Zhixin ; Li, Weiyuan ; Zhang, Liqing: *DoveNet: Deep Image Harmonization via Domain Verification*. 2020. – URL https://arxiv.org/abs/1911.13239

[8] Cun, Xiaodong ; Pun, Chi-Man: Improving the Harmony of the Composite Image by Spatial-Separated Attention Module. In: *IEEE Transactions on Image Processing* 29 (2020), S. 4759–4771. – URL http://dx.doi.org/10.1109/TIP.2020.2975979. – ISSN 1941-0042

[9] Debevec, Paul: Rendering synthetic objects into real scenes, 08 2008, S. 1

[10] Donnelly, William ; Lauritzen, Andrew: Variance shadow maps, 01 2006, S. 161–165

[11] Goyal, Priya ; Dollár, Piotr ; Girshick, Ross ; Noordhuis, Pieter ; Wesolowski, Lukasz ; Kyrola, Aapo ; Tulloch, Andrew ; Jia, Yangqing ; He, Kaiming: *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2018. – URL https://arxiv.org/abs/1706.02677

[12] Guan, Huankang ; Xu, Ke ; Lau, Rynson W. H.: *Delving into Dark Regions for Robust Shadow Detection*. 2024. – URL https://arxiv.org/abs/2402.13631

[13] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: *Deep Residual Learning for Image Recognition*. 2015. – URL https://arxiv.org/abs/1512.03385

[14] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: *Identity Mappings in Deep Residual Networks*. 2016. – URL https://arxiv.org/abs/1603.05027

[15] Hong, Yan ; Niu, Li ; Zhang, Jing: Shadow generation for composite image in real-world scenes. In: *Proceedings of the AAAI Conference on Artificial Intelligence* Bd. 36, 2022

[16] IBM: What are convolutional neural networks? In: *IBM* (2020). – URL https://www.ibm.com/think/topics/convolutional-neural-networks

[17] Ioffe, Sergey ; Szegedy, Christian: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: Bach, Francis (Hrsg.) ; Blei, David (Hrsg.): *Proceedings of the 32nd International Conference on Machine Learning* Bd. 37. Lille, France : PMLR, 07–09 Jul 2015, S. 448–456. – URL https://proceedings.mlr.press/v37/ioffe15.html

[18] Isola, Phillip ; Zhu, Jun-Yan ; Zhou, Tinghui ; Efros, Alexei A.: *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. – URL https://arxiv.org/abs/1611.07004

[19] Janocha, Katarzyna ; Czarnecki, Wojciech M.: *On Loss Functions for Deep Neural Networks in Classification*. 2017. – URL https://arxiv.org/abs/1702.05659

[20] Jin, Rui ; Niu, Qiang: Automatic Fabric Defect Detection Based on an Improved YOLOv5. In: *Mathematical Problems in Engineering* 2021 (2021), 09, S. 1–13

[21] Johnson, Justin ; Alahi, Alexandre ; Fei-Fei, Li: *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. 2016. – URL https://arxiv.org/abs/1603.08155

[22] Karsch, Kevin ; Sunkavalli, Kalyan ; Hadap, Sunil ; Carr, Nathan ; Jin, Hailin ; Fonte, Rafael ; Sittig, Michael: *Automatic Scene Inference for 3D Object Compositing*. 2019. – URL https://arxiv.org/abs/1912.12297

[23] Krizhevsky, Alex: Learning Multiple Layers of Features from Tiny Images. In: *University of Toronto* (2012), 05

[24] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey: ImageNet Classification with Deep Convolutional Neural Networks. In: *Neural Information Processing Systems* 25 (2012), 01

[25] Lecun, Y. ; Bottou, L. ; Bengio, Y. ; Haffner, P.: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86 (1998), Nr. 11, S. 2278–2324

[26] LeCun, Yann ; Bengio, Y. ; Hinton, Geoffrey: Deep Learning. In: *Nature* 521 (2015), 05, S. 436–44

[27] Liu, Daquan ; Long, Chengjiang ; Zhang, Hongpan ; Yu, Hanning ; Dong, Xinzhi ; Xiao, Chunxia: ARShadowGAN: Shadow Generative Adversarial Network for Augmented Reality in Single Light Scenes. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020

[28] Liu, Qingyang ; You, Junqi ; Wang, Jianting ; Tao, Xinhao ; Zhang, Bo ; Niu, Li: *Shadow Generation for Composite Image Using Diffusion model*. 2024. – URL https://arxiv.org/abs/2403.15234

[29] Liu, Qingyang ; You, Junqi ; Wang, Jianting ; Tao, Xinhao ; Zhang, Bo ; Niu, Li: *Shadow Generation for Composite Image Using Diffusion model.* 2024. – URL https://arxiv.org/abs/2403.15234

[30] Nair, Vinod ; Hinton, Geoffrey: Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair, 06 2010

[31] Niu, Li ; Cong, Wenyan ; Liu, Liu ; Hong, Yan ; Zhang, Bo ; Liang, Jing ; Zhang, Liqing: *Making Images Real Again: A Comprehensive Survey on Deep Image Composition.* 2025. – URL https://arxiv.org/abs/2106.14490

[32] O'Brien, James ; Farid, Hany: Exposing Photo Manipulation with Inconsistent Shadows. In: *ACM Transactions on Graphics - TOG* 32 (2012), 05, S. 1–11

[33] O'Shea, Keiron ; Nash, Ryan: *An Introduction to Convolutional Neural Networks.* 2015. – URL https://arxiv.org/abs/1511.08458

[34] Park, Taesung ; Efros, Alexei A. ; Zhang, Richard ; Zhu, Jun-Yan: *Contrastive Learning for Unpaired Image-to-Image Translation.* 2020. – URL https://arxiv.org/abs/2007.15651

[35] Paszke, Adam ; Gross, Sam ; Massa, Francisco ; Lerer, Adam ; Bradbury, James ; Chanan, Gregory ; Killeen, Trevor ; Lin, Zeming ; Gimelshein, Natalia ; Antiga, Luca ; Desmaison, Alban ; Köpf, Andreas ; Yang, Edward ; DeVito, Zach ; Raison, Martin ; Tejani, Alykhan ; Chilamkurthy, Sasank ; Steiner, Benoit ; Fang, Lu ; Bai, Junjie ; Chintala, Soumith: *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* 2019. – URL https://arxiv.org/abs/1912.01703

[36] Pearce, Tim ; Brintrup, Alexandra ; Zhu, Jun: *Understanding Softmax Confidence and Uncertainty.* 2021. – URL https://arxiv.org/abs/2106.04972

[37] Perez, P. ; Gangnet, M. ; Blake, A.: Poisson Image Editing. In: *Proceedings of the ACM SIGGRAPH 2003 Conference*, 2003, S. 313–318

[38] PyTorch: BCE Loss. In: *Pytorch Documentation.* – URL https://docs.pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

[39] Ronneberger, Olaf ; Fischer, Philipp ; Brox, Thomas: *U-Net: Convolutional Networks for Biomedical Image Segmentation.* 2015. – URL https://arxiv.org/abs/1505.04597

[40] Sara, Akter M. ; Uddin, M.: Image Quality Assessment through FSIM, SSIM, MSE and PSNR—A Comparative Study. In: *Journal of Computer and Communications* (2019). – URL 10.4236/jcc.2019.73002.

[41] Simonyan, Karen ; Zisserman, Andrew: *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. – URL https://arxiv.org/abs/1409.1556

[42] Suvorov, Roman ; Logacheva, Elizaveta ; Mashikhin, Anton ; Remizova, Anastasia ; Ashukha, Arsenii ; Silvestrov, Aleksei ; Kong, Naejin ; Goka, Harshith ; Park, Kiwoong ; Lempitsky, Victor: *Resolution-robust Large Mask Inpainting with Fourier Convolutions*. 2021. – URL https://arxiv.org/abs/2109.07161

[43] Swapna: Convolutional Neural Network | Deep Learning. In: *Developers Breach* (2020). – URL https://developersbreach.com/convolution-neural-network-deep-learning

[44] Tao, Xinhao ; Cao, Junyan ; Hong, Yan ; Niu, Li: Shadow Generation with Decomposed Mask Prediction and Attentive Shadow Filling. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024

[45] Tsai, Y.-H. ; Chen, Y.-Y. ; Hsieh, J.-W. ; Yang, M.-H.: Deep Image Harmonization. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, URL https://openaccess.thecvf.com/content_cvpr_2017/html/Tsai_Deep_Image_Harmonization_CVPR_2017_paper.html, 2017, S. 1–9

[46] Ulyanov, Dmitry ; Vedaldi, Andrea ; Lempitsky, Victor: *Instance Normalization: The Missing Ingredient for Fast Stylization*. 2017. – URL https://arxiv.org/abs/1607.08022

[47] Valença, Lucas ; Zhang, Jinsong ; Gharbi, Michaël ; Hold-Geoffroy, Yannick ; Lalonde, Jean-François: Shadow Harmonization for Realistic Compositing. In: *ACM SIGGRAPH Asia 2023 Conference Proceedings*, 2023

[48] Wang, Tianyu ; Hu, Xiaowei ; Fu, Chi-Wing ; Heng, Pheng-Ann: Single-Stage Instance Shadow Detection With Bidirectional Relation Learning. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, S. 1–11

[49] Wang, Tianyu ; Hu, Xiaowei ; Heng, Pheng-Ann ; Fu, Chi-Wing: Instance Shadow Detection with A Single-Stage Detector. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022), S. 1–14

[50] Wang, Tianyu ; Hu, Xiaowei ; Wang, Qiong ; Heng, Pheng-Ann ; Fu, Chi-Wing: Instance Shadow Detection. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020

[51] Yamashita, Rikiya ; Nishio, Mizuho ; Do, Richard ; Togashi, Kaori: Convolutional neural networks: an overview and application in radiology. In: *Insights into Imaging* 9 (2018), 06

[52] Zhang, Bo ; Duan, Yuxuan ; Lan, Jun ; Hong, Yan ; Zhu, Huijia ; Wang, Weiqiang ; Niu, Li: *ControlCom: Controllable Image Composition using Diffusion Model*. 2023. – URL https://arxiv.org/abs/2308.10040

[53] Zhang, Richard ; Isola, Phillip ; Efros, Alexei A. ; Shechtman, Eli ; Wang, Oliver: *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*. 2018. – URL https://arxiv.org/abs/1801.03924

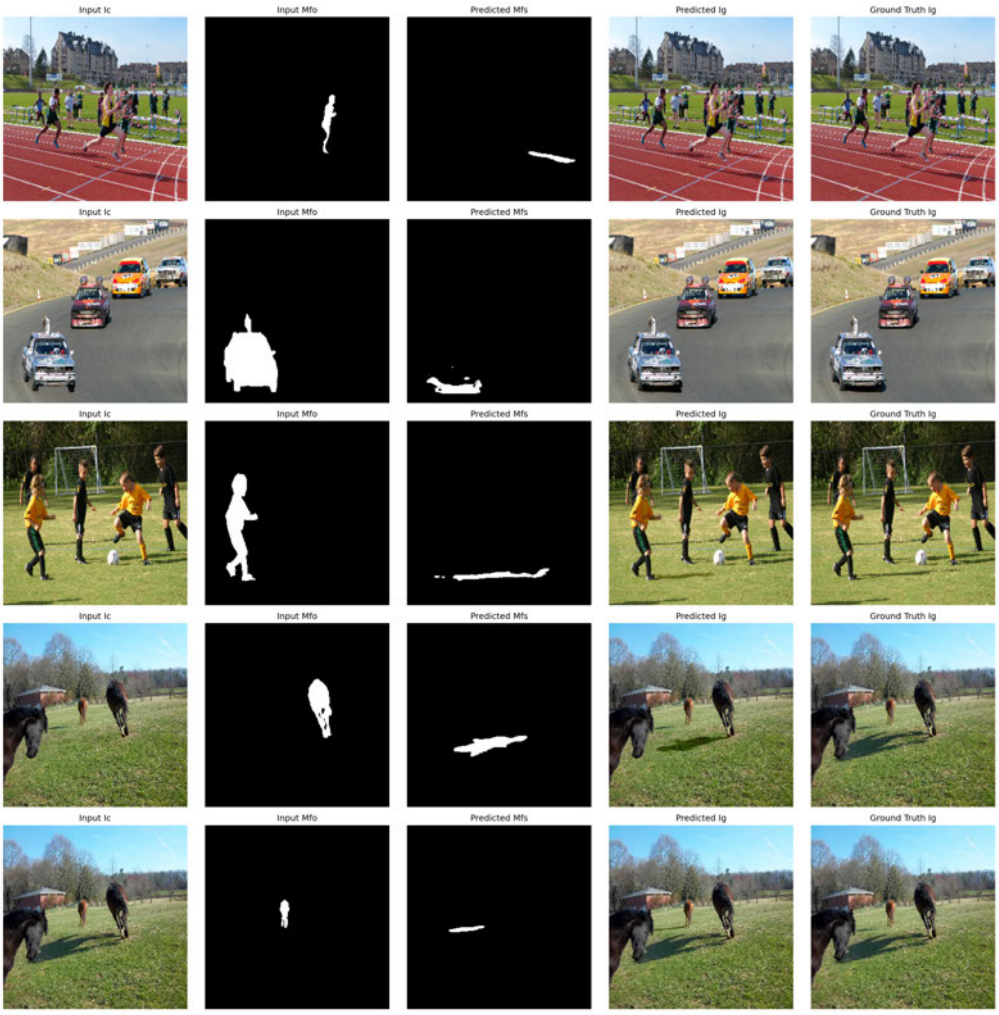
[54] Zhang, Shuyang ; Liang, Runze ; Wang, Miao: ShadowGAN: Shadow synthesis for virtual objects with conditional adversarial networks. In: *Computational Visual Media* 5 (2019), 03, S. 105–115

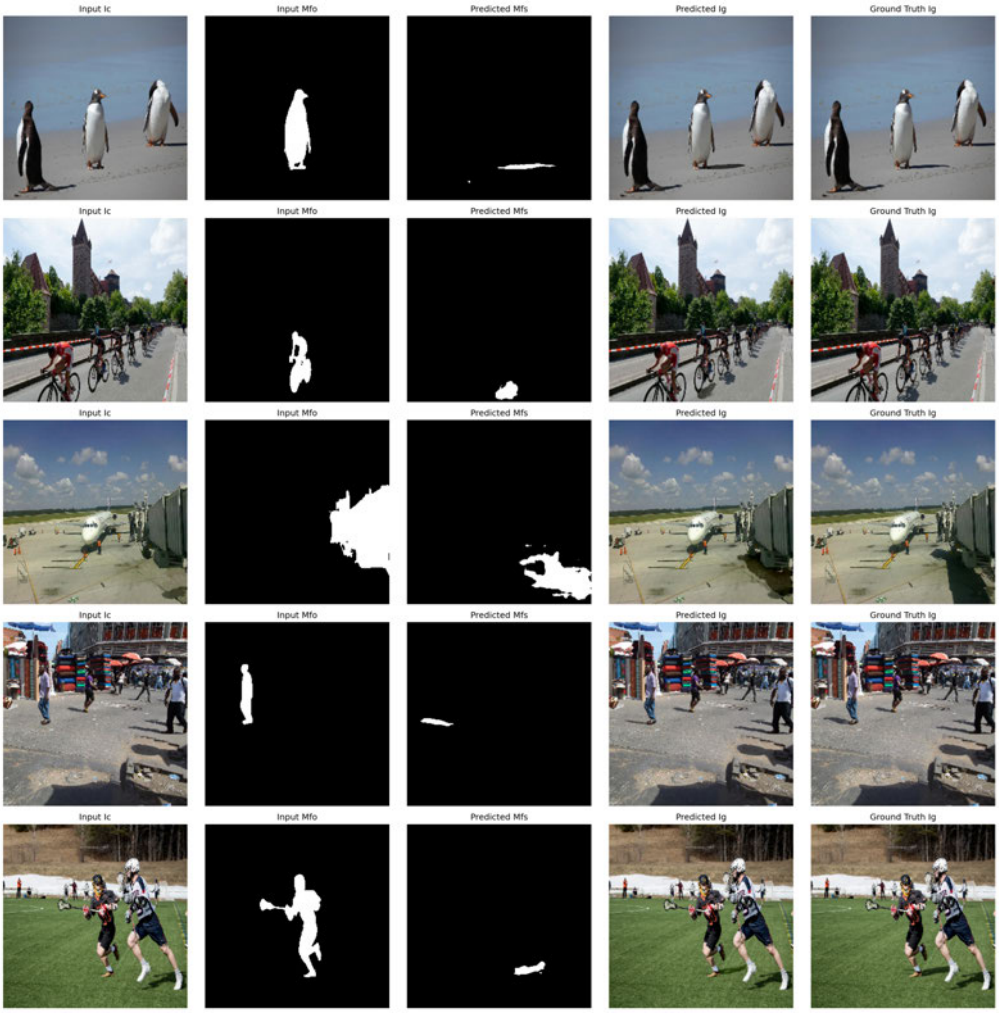
[55] Zhao, Shengyu ; Cui, Jonathan ; Sheng, Yilun ; Dong, Yue ; Liang, Xiao ; Chang, Eric I. ; Xu, Yan: *Large Scale Image Completion via Co-Modulated Generative Adversarial Networks*. 2021. – URL https://arxiv.org/abs/2103.10428

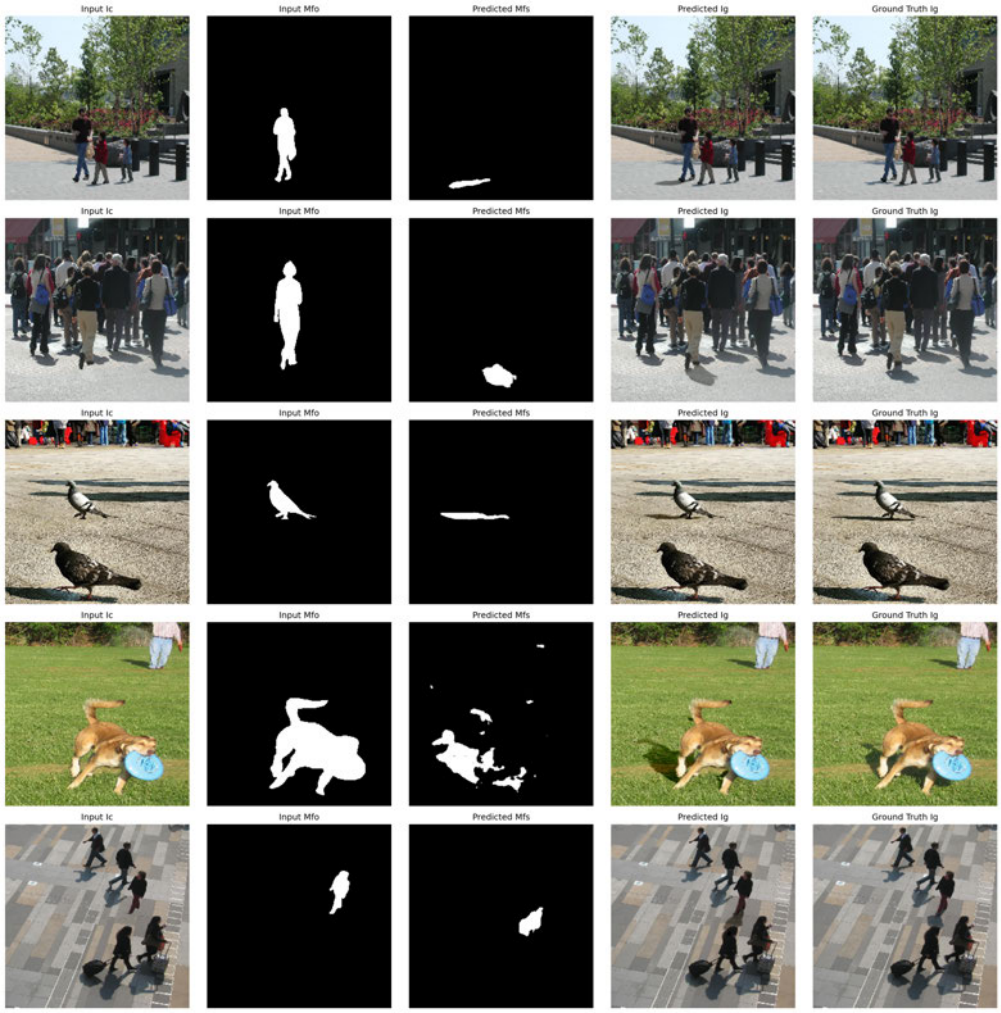
[56] Zheng, Zhaohui ; Wang, Ping ; Liu, Wei ; Li, Jinze ; Ye, Rongguang ; Ren, Dongwei: *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression*. 2019. – URL https://arxiv.org/abs/1911.08287

[57] Zhu, Jun-Yan ; Park, Taesung ; Isola, Phillip ; Efros, Alexei A.: *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2020. – URL https://arxiv.org/abs/1703.10593

A Appendix


Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig


Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig

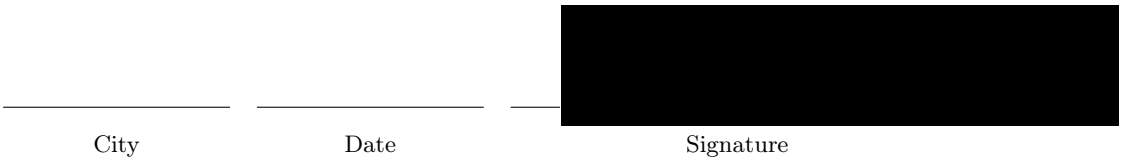

Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig


Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig
Input Ic
Input Mfo
Predicted Mfs
Predicted Ig
Ground Truth Ig

The rest of the appendix to the thesis, including source code and other metrics values, is on CD and can be obtained from the first examiner.

Declaration

I declare that this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used.



City Date Signature