MonoDepthDeFocus

CS492 Project

MDL17CS012 CSU172/05 Albert B George
MDL17CS045 CSU172/23 Chris Davis
MDL17CS046 CSU172/24 Christy Thambi
MDL17CS092 CSU172/48 Sachin Sajith

B. Tech Computer Science & Engineering

Department of Computer Engineering
Model Engineering College
Thrikkakara, Kochi 682021
Phone: +91.484.2575370
http://www.mec.ac.in
hodcs@mec.ac.in

June 2021

Model Engineering College Thrikkakara
Department of Computer Engineering

CERTIFICATE

This is to certify that, this report titled ***MonoDepthDeFocus*** is a bonafide record of the work done by

MDL17CS012 CSU172/05 Albert B George
MDL17CS045 CSU172/23 Chris Davis
MDL17CS046 CSU172/24 Christy Thambi
MDL17CS092 CSU172/48 Sachin Sajith

Eighth Semester B. Tech. Computer Science & Engineering

students, for the course work in **CS492 Project**, which is the second part of the two semester project work, under our guidance and supervision, in partial fulfillment of the requirements for the award of the degree, B. Tech. Computer Science & Engineering of **APJ Abdul Kalam Technological University**.

Guide

Sheena KV
Assistant Professor
Computer Engineering

Coordinator

Dr. Preetha Theresa Joy
Professor
Computer Engineering

Head of the Department

Dr. Preetha Theresa Joy
Professor
Computer Engineering

June 24, 2021

Acknowledgements

This project would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to all of them.

First of all, We would like to thank our esteemed Principal, Prof. Dr. Vinu Thomas, for his guidance and support in maintaining a calm and refreshing environment to work in and also for providing the facilities that this work demanded.

We are highly indebted to our Project Coordinator and Head of the Department, Dr. Preetha Theresa Joy, for her guidance, support and constant supervision throughout the duration of the work as well as for providing all the necessary information and facilities that this work demanded.

We would like to thank our Project Guide, Ms.Sheena KV for her support and valuable insights and also for helping me out in correcting any mistakes that were made during the course of the work.

We offer our sincere gratitude to all our friends and peers for their support and encouragement that helped me get through the tough phases during the course of this work.

Last but not the least, we thank the Almighty God for guiding me through and enabling us to complete the work within the specified time.

`Albert B George`

`Chris Davis`

`Christy Thambi`

`Sachin Sajith`

Abstract

Depth estimation plays a key role in understanding any given scene. In particular, monocular depth estimation is interesting from a practical point of view, since using a single camera is cheaper than many other options and avoids the need for continuous calibration strategies as required by stereo-vision approaches.The project aims to enable the graded blurring of an image based on how far a point is from focus using just a single image with no extra data. This can be achieved by generating a depth map for the image using machine learning. Blurred versions of the image can be created in levels based on how far a point is from the depth of focus. Stitching the different blurred images will create the final image with the selected point fully focussed and points further away become more blurred.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many a times a person might take a photo which is somewhat flat and wish the subject stood out a bit more or maybe the background of the image had less emphasis on the objects that are unappealing or if rest of the image was gradually blurred with respect to the subject for artistic purposes. All of these issues can be fixed with one simple thing, a shallow depth of field. This is a great way to bring attention to a specific part of an image, and the effect is similar to how things would look if the photo had been shot with a camera having wide aperture lens. But such hardware is not always easily available or usable to the casual user.

To overcome this problem we propose a software which converts a simple input image taken from a conventional camera, into a synthetically defocussed image, where the image is blurred in a graded manner, with the blurring increasing from minimum at the point of focus selected by the user , to maximum at the farthest point from the focus.

1.1 Proposed Project

1.1.1 Problem Statement

To produce a synthetically blurred image based on how far a point is from point of focus, without using any extra data but a single image and performing monocular depth estimation on the image.

1.1.2 Proposed Solution

Graded blurring of an image based on how far a point is from focus using just a single image with no extra data. To achieve this we generate a depth map for the image using machine learning. Create blurred versions of the image in levels based on how far a point is from the depth of focus. Stitch the different blurred images to create the final image with the selected point fully focussed and points further away become more blurred.

Chapter 2

System Study Report

2.1 Literature Survey

1.Learning 3-D Scene Structure from a Single Still Image [1]
Ashutosh Saxena et al. proposed a paper regarding 3D depth estimation from a single still image. Here, a training set of monocular images is collected. Unlike triangulation-based algorithms which are widely used, such as stereopsis and structure from motion, an algorithm has been developed that makes use of a large orthogonal set of monocular cues. The proposed model uses a hierarchical, multiscale Markov Random Field (MRF) that uses such cues to estimate depth from a single still image. It also estimates the depth and the relation between depths at di fferent points in the image. It has been shown that even on unstructured scenes, this algorithm is repeatedly used to recover reasonably accurate depth maps.

2.Shape Reconstruction from Two Color Images using Photometric Stereo combined with Segmentation and Stereopsis [2]
Osamu Ikeda considered the problem of estimating the depth of each pixel in a scene from a single monocular image. First, a semantic segmentation of the scene is performed and the semantic labels are used to show the 3D reconstruction. The photometric stereo with color segmentation and the binocular stereopsis have been combined to reconstruct accurate shapes from two color images and also to reconstruct the scene. Finally, the existing approach depends on accurate ground truth data for learning the parameters of the linear regression model. It aims at extending the model in order to learn from richer data sources together with synthetic data.

3.Depth Perception from a Single Still Image [3]
Min Sun and Andrew Y. Ng et al. stated that the problem of depth perception is essential to computer vision. For any image, depth and shape perception are mapped as they are important elements for many applications like object recognition, grasping, navigation, image composing and video retrieval. Monocular depth perception has an ability to improve all of these applications, provided only a single image scene is available. In a still image we have many monocular cues like texture variations, gradients, defocus, color which are used for depth perception.

4.Single Image Depth Estimation From Predicted Semantic Labels [4]
Beyang Liu et al. proposed an approach where, semantic segmentation is performed and used

to show 3d reconstruction. Several advantages can be achieved using this approach. Depth and geometry constraints can easily imposed by knowing semantic class of a pixel or region. In this approach firstly, the foreground class is divided into subdivisions which allows addition of some more modeling constraints. Second, geometry currently makes strong assumptions about position of supporting pixels. Finally, it depends on accurate ground data for learning parameters of our linear regression model.

5.A threshold selection method from gray level histograms [5]
N.Otsu presented a non-parametric method of automatic threshold selection for picture segmentation. A discriminant criterion is used to select an optimal threshold, in order to make best use of the separability of the resultant classes in gray level. The method is rather common as it covers a wide scope of unsupervised decision procedure.

6.A flexible new technique for camera calibration [6]
Zhengyou Zhang introduced a new technique to easily calibrate a camera. It only requires the camera to examine a planar pattern that is shown at a few different orientations. If pixels are square, the minimum number of orientations is two. However, it is suggested that four or five different orientations should be used to improve quality. Both computer simulation and real data have been used to test the proposed technique and the results obtained have been quite accurate.

2.2 Proposed System

The system provides the graded blurring of an image based on how far a point is from the focus using just a single image. To achieve this we generate a depth map for the image using machine learning. Create blurred versions of the image in levels based on how far a point is from the depth of focus. Stitch the different blurred images to create the final image with the selected point fully focused and points further away become more blurred.

We have used the method based on Unsupervised Monocular Depth Estimation with Left- Right Consistency.

In simple terms the model is trained on a large set of stereo(left-right) images to generate a right image from a given left image. Generating the right image, the model is learning internally about the depth of various points in the image.

Chapter 3

Software Requirement Specification

3.1 Introduction

3.1.1 Purpose

The purpose of the Software Requirements Specification document is to maintain all the functions and the specifications of monocular depth estimation with left- right consistancy and graded blurring of the image using the estimated depth. Besides, it contains detailed descriptions of all the hardware, software, functional and non-functional requirements specified.

3.1.2 Document Conventions

This document follows MLA (Modern Language Association) Format. Bold-faced text has been used to emphasize section and sub-section headings. Highlighting is to point out words in the glossary and italicized text is used to label and recognize diagram.

3.1.3 Intended Audience and Reading Suggestions

This document contains software functionality, software and hardware requirements and user documentation.

• Developer: The developer who wants to read, change, modify or add new requirements into the existing program may need first to consult this document and update the requirements in an appropriate manner so as not to change the actual purpose of the system or make the system inconsistent.

• User: The user of this program reviews the diagram and the specification provided in the document and check to determine whether the software has all the suitable requirements and if the software developer has implemented all of them. He/She can also consult the user guide in the event of any confusion for clarifications.

• Tester: The tester needs this document to prepare his test cases to validate that the initial requirements of this project is actually implemented in the deliverable.

This document need not be read sequentially; users are encouraged to jump to any section they find relevant.

3.1.4 Project Scope

The project named **MonoDepthDeFocus** aims at producing a synthetically blurred version of simple an input image , where the input image can be taken from a conventional camera by a casual photographer, and does not require expensive camera hardware or professional photographers. It achieves this by estimating the depth of each pixel of the image using machine learning techniques. The resultant depth map is used for blurring the image in a graded manner depending on the distances of the various points in the image.

3.1.5 Overview of Developer's Responsibilities

The developer should:

- Maintain appropriate coding standards and design.
- Contribute to technical design documentation
- Contribution in accordance with the software development life cycle.
- Producing detailed specifications and writing program codes.
- Testing the product in controlled, real situations before going live.
- Preparation of training manuals for users.
- Maintaining the systems once they are up and running.

3.2 Overall Description

3.2.1 Product Perspective

The project is a self contained application that simplifies the process of graded blurring of an image based on how far a point is from focus using just a single image with no extra data. To achieve this we generate a depth map for the image using machine learning. Create blurred versions of the image in levels based on how far a point is from the depth of focus. Stitch the different blurred images to create the final image with the selected point fully focussed and points further away become more blurred. We have used the method based on the paper Unsupervised Monocular Depth Estimation with Left-Right Consistency. In simple terms the model is trained on a large set of stereo(left-right) images to generate a right image from a given left image. Generating the right image, the model is learning internally about the depth of various points in the image.

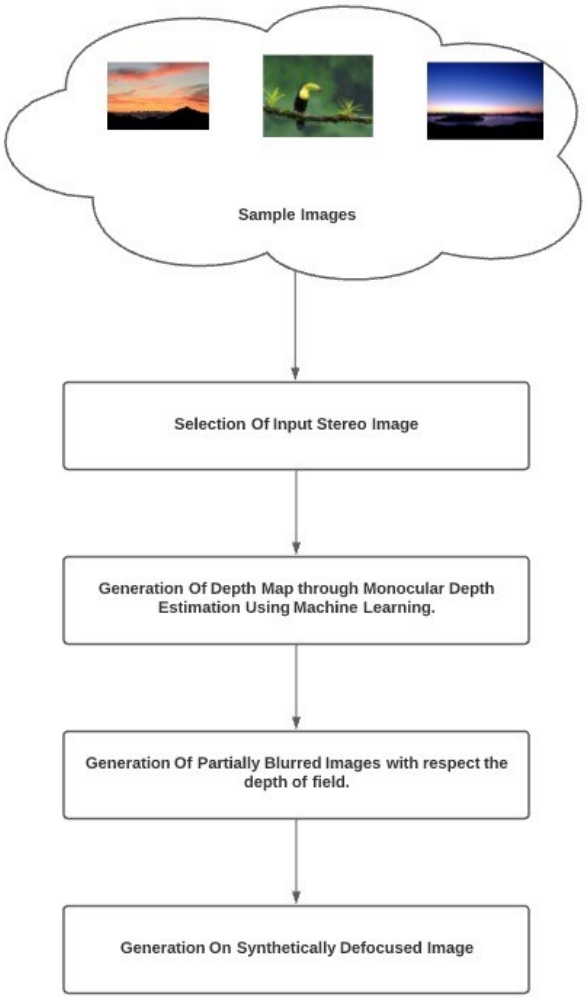


Figure 3.1: Block Diagram

3.2.2 Product Functions

The project aims at making depth estimation estimation from a single image easy and efficient and also using the estimated depth it helps produce a syntheticallydefocussed image where the point of maximum focus is the region where the cursor lies and the other regions are blurred in a gradient with respect to the distance of those regions from the point of focus.

Main functions of the project are:

- Accept single input image taken from a conventional camera or otherwise.
- Create a depth map using the input image through machine learning techniques.
- Create a synthetically defocussed image from the depth map.

3.2.3 User Classes and Characteristics

Casual users with regular camera module ,experts and researchers with proffessional camara hardware.

3.2.4 Operating Environment

This is a gpu intensive system and hence will require a computer which has a appropriately powered graphical processing unit, (recommened Nvidea 1050ti with 4gb Vram and above). CPU also requires to be of x86 architechture so as to be compatible with all the various software used.

3.2.5 Design and Implementation Constraints

- **Regulatory Policies**: The availability of input image depends entirely on the camera hardware , and regulations in them will affect the data collection process.
- **Hardware Limitations**: This is a gpu intensive system and hence will require a computer which has a appropriately powered graphical processing unit.
- **Audit Functions**: NA
- **Control Functions**: NA
- **Safety and Security Considerations**: Error messages in time of exceptions occured so as to protect the system from harm.
- **Reliability Requirements**: Total number of bugs in the system shall not exceed 1 percent of the total line number of code, except connection reliability, which is out of range.
- **Criticality of the Application**: The application shall be available 365 days as the requirements

will be hosted on a server.

3.2.6 User Documentation

All documentation will be made in accordance with requirements pertaining to open source software under the GNU General Purpose License. Additionally, user documentation will be available with the concerning paper to determine proper use.

3.2.7 General Constraints

Availability of a device with necessary graphical processing capability, and requirement of a physical camera module for the procurement of input images.

3.2.8 Assumptions and Dependencies

- API's used in the system are available freely.
- Camera Module to take input images is available.

Dependencies:

- python
- openCV
- TensorFlow
- matplotib
- numpy

3.3 External Interface Requirements

3.3.1 User Interfaces

• Front-end User Interface using Python extension.

3.3.2 Hardware Interfaces

• Device should have a GPU.

3.3.3 Software Interfaces

• **Python API for TensorFlow** : TensorFlow is a free and open-source software library for machine learning. This API helps in accessing and using TensorFlow.

• **Pyplot API**: An API that makes Matplotlib work like MATLAB.

• **PyGObject API** : PyGObject is an API which provides bindings for GObject based libraries such as GTK.

3.4 Hardware and Software Requirements

3.4.1 Hardware Requirements

The project requires the following hardware for functioning:

• **Graphics Processing Unit**: A Graphics Processing Unit (GPU) is a chip or electronic circuit capable of rendering graphics for display on an electronic device. A GPU equivalent or higher to Nvidea 1050ti with 4 GB VRAM is recommended.

3.4.2 Software Requirements

This project requires the following software for functioning:

• **Operating System**: Linux based OS.

• **Python**: Python is an interpreted, high-level and general-purpose programming language.

• **OpenCV-Python**: OpenCV is a library of programming functions mainly aimed at real-time computer vision. OpenCV-Python is a library of Python bindings for OpenCV.

• **Matplotlib**: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

• **Numpy**: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

• **Pycairo**: Pycario is a Python module providing bindings for the cairo graphics library.

• **TensorFlow**: TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

3.5 Functional Requirements

This section gives the details of the features and functions this system provides for various users and additional information on how each module works.

3.5.1 Selection Of Image as the Input

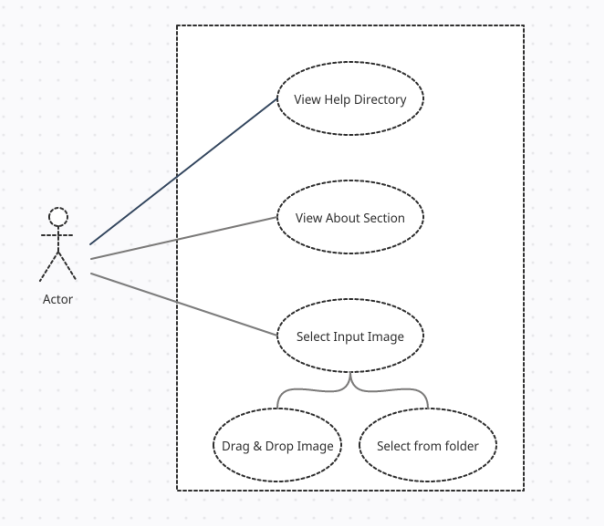


Figure 3.2: Use case diagram for input image selection

Description: The system allows the user to select an input image which has been taken from a conventional camera system, which acts as the input data for the system over which various analysis and functions are conducted in order to obtain the required output.The functions in this section are :
• Enable the user to view help directory, which consists of the detailed instruction of using the system.

• Enable the user to view the about section, which consists of a general description of the system and its functions.

• Enable the user to select Input Image by two different methods:

1. To drag and drop the image from a folder into the system using a cursor.

2. To select the image directly from the file system.

Actors: The actors include both amateur users using general purpose camera hardware, to experts and researchers requiring accurate depth analysis and defocusing.

Precondition: The user should have camera hardware to take images, and should use a computer with recommended system configuration of 8gb ram or higher, intel core i5 10 th gen or higher, and an Nvidea GPU 1050 or higher with 4gb Vram or higher.

Main Flow of Events:
• The user reads the help directory and about section to familiarize with the functions in the system and how to access them.

• User selects an input image stored in the computer which has been taken from a camera module, and either drags and drops the picture from the source to destination , or directly accesses the picture through the system.

Alternate Flow of Events: In case of failure to select the image, or if the image file format is not supported, an error message is displayed on the screen.

Post Condition: The selected image is accepted by the system, and is ready for further processes.

3.5.2 Depth Estimation Generation of Depth Map

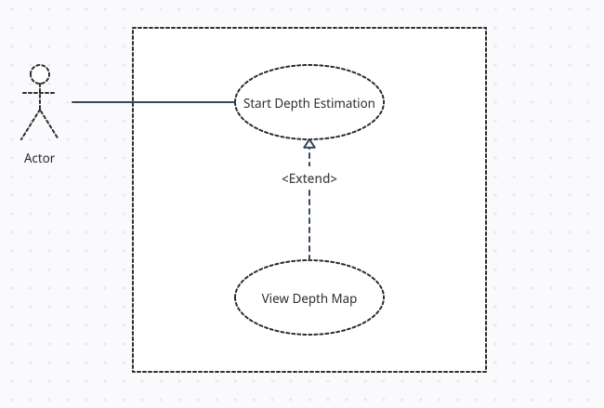


Figure 3.3: Use case diagram for depth map generation.

Description: The system alows the generate the a depth map of the input image which has been selected by the user in the previous step. The depth estimation is conducted through the help of a trained machine learning model. The model is trained on alarge set of stereo(left-right) images to generate a right image from a given left image. Generating the right image, the model is learning internally about the depth of various points in the image.Once the model is trained. We can give it a simple image and it will be able to generate a depth map for that image.

The functions in this section are:

• To enable the user to generate a depth map using the input image which had been initially selected.

Actors: The actors include both amateur users using general purpose camera hardware, to experts and researchers requiring accurate depth analysis and defocusing.

Precondition: The user should have camera harware to take images, and should use a computer with recommended system configuration of 8gb ram or higher, intel core i5 10 th gen or higher, and an Nvidea GPU 1050 or higher with 4gb Vram or higher.

Main Flow of Events:
• Once the input image is selected in the previous section, then the user clicks on the next button to start the depth estimation process.

• After the completion of the process a depth map is produced and is displayed to the user, this can be used by the user as is, or can be further used as the input to for graded blurring of the input image based on how far a point is from focus.

Alternate Flow of Events: In case of failure of depth map generation due to failure in depth estimation or otherwise , an error message is displayed on the screen.

Post Condition:The depth map generated from the selected input image is accepted by the system, and is ready for further processes.

3.5.3 Graded Blurring of the Image

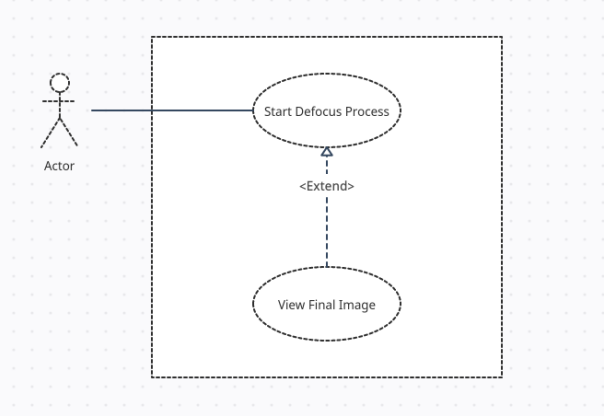


Figure 3.4: Use case diagram for defocussed image generation

Description: The system allows graded blurring of an image based on how far a point is from focus using just a single input image with no extra data. To achieve this a depth map for the image is generated using machine learning. Blurred versions of the image in levels based on how far a point is from the depth of focus are created.The different blurred images are stitched together to create the final image with the selected point fully focused and points further away become more blurred.

The functions in this section are:

• To enable the user to generate a synthetically defocused image using a single input image which has been initially selected.

Actors: The actors include both amateur users using general purpose camera hardware, to experts and researchers requiring accurate depth analysis and defocusing.

Precondition: The user should have camera harware to take images, and should use a computer with recommended system configuration of 8gb ram or higher, intel core i5 10 th gen or higher, and an Nvidea GPU 1050 or higher with 4gb Vram or higher.

Main Flow of Events:
• Once the depth map from the input image is generated in the previous section, then the user clicks on the next button to production of numerous differently blurred images of the input image with respect to depth.

• After the completion of the process a complete synthetically defocussed image of the input image is produced and is displayed to the user, here the region pointed by the cursor is at maximum focus and the regions farther away from the point of focus are defoccused gradually with respect to the distance between the region and the point of maximum focus/ where the cursor is situated.

Alternate Flow of Events: In case of failure to produce the defocussed image, an error message is displayed on the screen.

Post Condition: The synthetically defocussed image is displayed to the user and can be stored in the system memory.

3.6 Non-functional Requirements

3.6.1 Performance Requirements

Major performance requirements are:

• Should be adaptable to all API structures and download data from them efficiently.

• Cross platform mapping should be fast and conflict free.

• Making depth estimation from a single image is easy and efficient; using the estimated depth it helps produce a synthetically defocussed image where the point of maximum focus is the region where the cursor lies and the other regions are blurred in a gradient with respect to the distance of those regions from the point of focus.

• Create a synthetically defocussed image from the depth map.

• Defocussed image generation should be adaptable to all formats.

• Requires a system which has appropriately powered graphical processing unit, (recommended nVidia 1050ti with 4GB VRAM and above). CPU also requires to be of x86 architecture so as to be compatible with all the various software used.

• Availability of a device with necessary graphical processing capability, and requirement of modules for the procurement of input images.

• Another performance requirement is the storage space. Higher storage space means more dataset can be trained, better the accuracy.

3.6.2 Safety Requirements

Security Requirements include:

• **Data Protection**: Collected data must not be corrupted during the process.

• **User Authentication**: Only the root user of the system can access the application.

• **Device Protection**: If device doesn't meet min requirements the application doesn't work so as to not cause harm to the system.

3.6.3 Software Quality Attributes

• **Correctness**: The software must correctly map the attributes of input image and give proper output image with accurate defocus.

• **Reliability**: The software should be able to work with different kinds of input images taken from diffrent kinds of camera modules and of diffrent image format and give accurate reports.

• **Robustness**: The software should be able to handle a lot of data, store them and perform aggregate operations on them without error and failure.

• **Usability**: The software must be usable to all from content creators to businesses/companies.

• **Learnability**: The Software must have a simple and interactive UI/UX. A linear style of movement through the website and a one-click solution for all the user needs.

• **Portability**: The software service must be available over all operating systems and devices.

• **Testability**: The software must be modularized and have a proper structure in order to conduct tests.

• **Extensibility**: Required modification at specific locations can be made without the software failing.

• **Scalability**: Software must be able to handle different image formats and newer camera harware that emerge in the future and be capable of supporting them.

• **Portability**: Software should be lightweight so that it can run on a machine with slow internet connection.

3.7 Other Requirements

Permissions: Sufficient permissions must be obtained from the users for the access of their photos.

Chapter 4

System Design

4.1 System Architecture

System architecture shows the various modules of the system and the interaction between them.The architecture of the proposed solution is as follows.

4.1.1 Depth Estimation Module

The Depth Estimation module allows the generation of a depth map of the input image which has been selected by the user. The depth estimation is conducted through the help of a trained machine learning model. The model is trained on a large set of stereo(left-right) images to generate a right image from a given left image. Generating the right image, the model is learning internally about the depth of various points in the image, once the model is trained. We can give it a simple image and it will be able to generate a depth map for that image.

4.1.2 Defocus Module

The system allows graded blurring of an image based on how far a point is from focus using just a single input image with no extra data. To achieve this a depth map for the image is generated using machine learning through the depth estimation module. Blurred versions of the image in levels based on how far a point is from the depth of focus are created.The different blurred images are stitched together to create the final image with the selected point fully focused and points further away become more blurred.

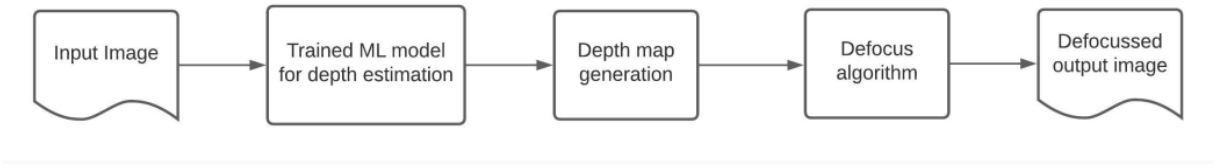


Figure 4.1: System Architecture

4.2 Input Design

In the system, users can select a picture or drag and drop that picture from its folder into the graphical user interface of the MonoDepthDefocus application. Once the picture is accepted , the software will display a depth map of the input image and will also defocus the input image with respect to the depth of focus provided as input through a left mouse-button click on the desired point in the picture.

4.3 Libraries and Packages Used

4.3.1 Libraries

The python standard library: Modules used

• **TensorFlow**:

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Tensorflow is a symbolic math library based on dataflow and differentiable programming.It is an open source artificial intelligence library, using data flow graphs to build models. It allows developers to create large-scale neural networks with many layers. TensorFlow is mainly used for: Classification, Perception, Understanding, Discovering, Prediction and Creation.

• **Open-CV**:

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source Apache 2 License.
OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source Apache 2 License.

• **Matplotlib**:

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

4.3.2 Packages

• **NumPy**:

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

• **Tkinter**:

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python. The name Tkinter comes from Tk interface

• **Six**:

Six provides simple utilities for wrapping over differences between Python 2 and Python 3. It is intended to support codebases that work on both Python 2 and 3 without modification. Six consists of only one Python file, so it is painless to copy into a project.

4.4 Usecase Diagram

A use case diagram in the Unified Modelling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use case) and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for each actor. Roles of the actors in the system can be depicted.

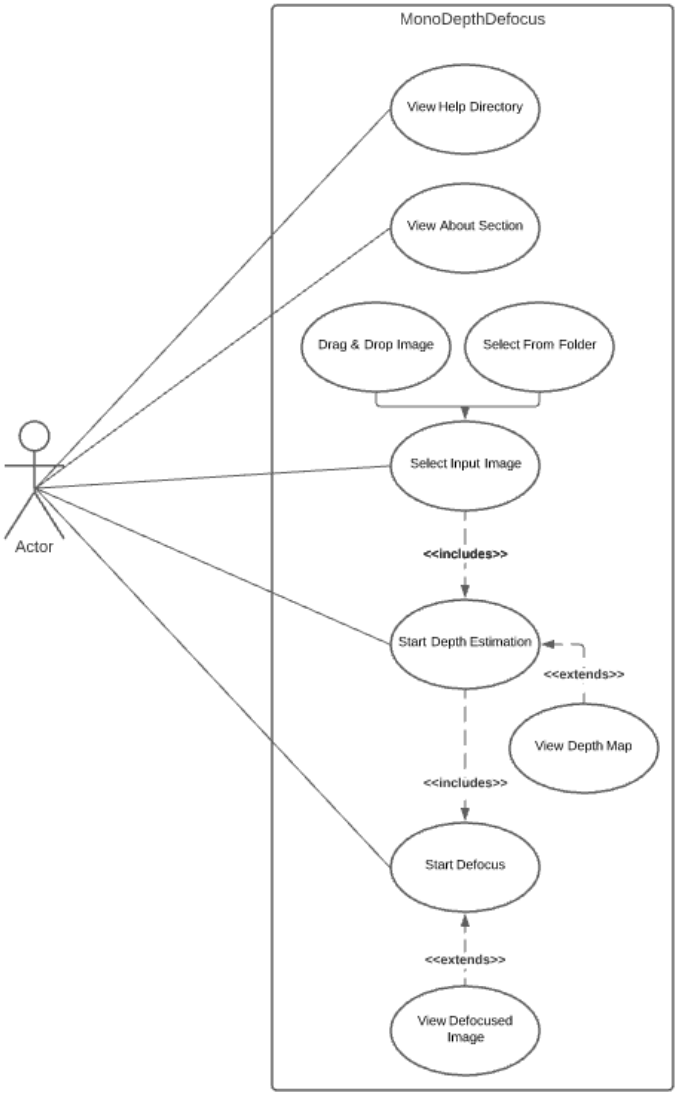


Figure 4.2: Usecase diagram of the entire system

4.5 Activity Diagram

Activity diagrams are graphical representations of work-flows of step wise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes. Activity diagrams show the overall flow of control.

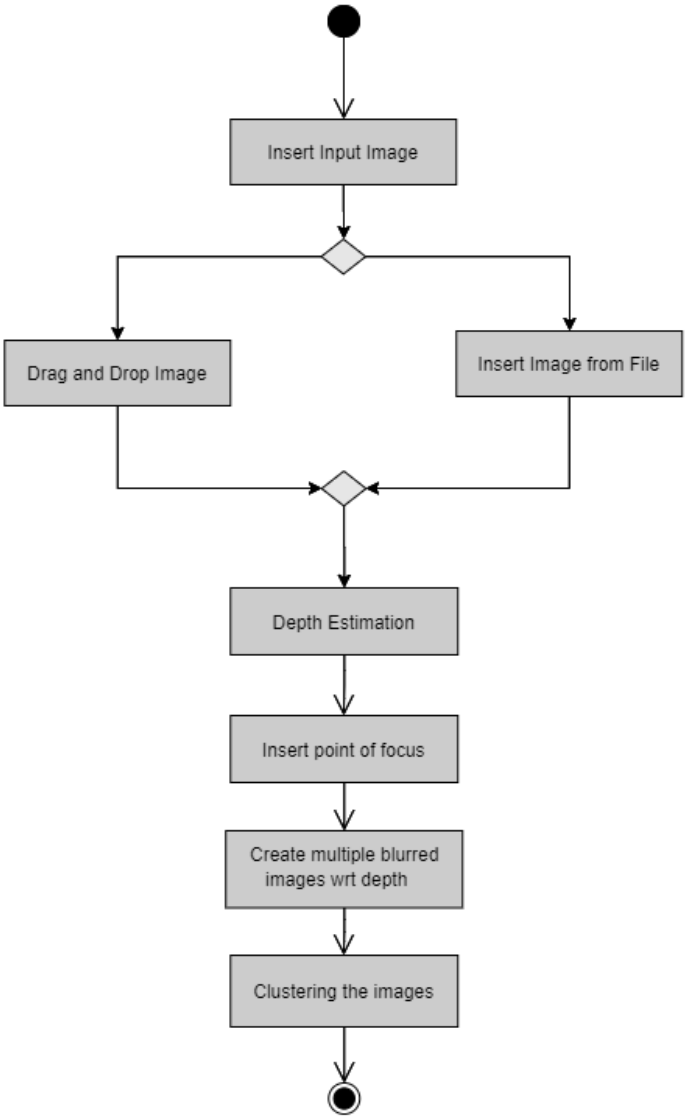


Figure 4.3: Activity Diagram

Chapter 5

Data Flow Diagram

Data flow diagram is a graphical representation of the flow of data through an information system, modelling processed aspects. Often they are a preliminary step used to create an overview of the system which can be later elaborated.

5.1 Level 0 DFD

Highest abstraction level DFD is known as level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details.

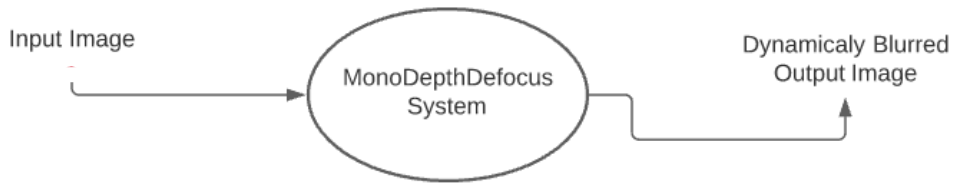


Figure 5.1: Level 0

5.2 Level 1 DFD

Level 0 DFD is broken down into more specific level 1 DFD. Level 1 DFD depicts basic modules in the system and the flow of data in various modules.

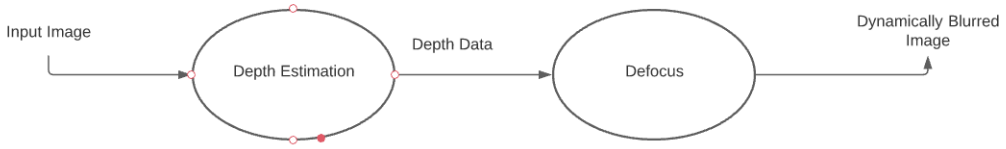


Figure 5.2: Level 1

5.3 Level 2 DFD

In this level DFD shows how data flows inside the modules mentioned in level 1

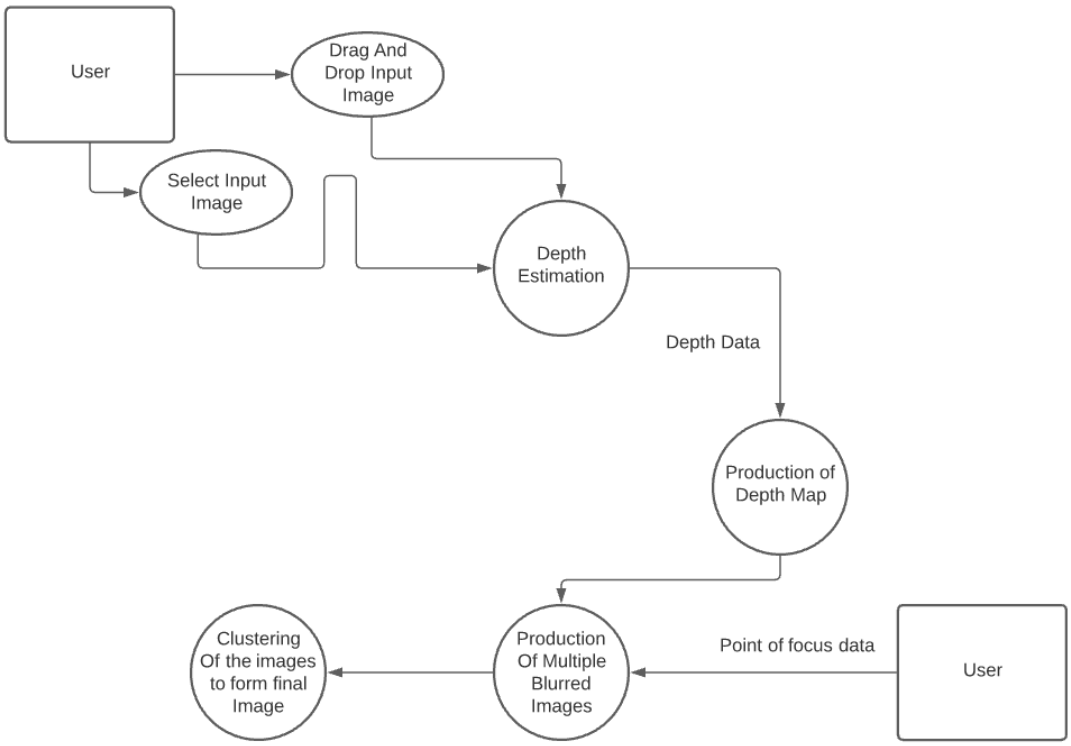


Figure 5.3: Level 2

Chapter 6

Implementation

6.1 Algorithms

There is a set of basic algorithms that aids in the realization of the MonoDepthFocus system.

6.2 Defocus Algorithm

The Defocus algoritm , helps in the graded blurring of the input image with respect to the depth of focus and the depth data of the diffrent pixels from the depth of focus, which is selected by the user using a cursor.

Input = Sample Image , Depth Map
Output = Defocused Image

1. Start
2. Class DefocuserObject() holds functions that handle defocusing of an image using a depth map
3. It takes in the path to the image to be defocused and the blur method to be used for defocusing
4. It requires the depth map to be saved in the same folder as the image.
5. Parameters:
   image_path: Path to the image which is to be defocused
   blur_method: The blur function to be used for blurring.
   Available values are: gaussian, avg_blur, median, bilateral
6. Lambda functions to call the appropriate blur function according to set method.
7. All the functions takes 2 arguments: the image and the kernel size to be used in the function.
8. As the kernel size increases the amount of blur increases.
9. The blurred versions of the images can be precalculated.

10. Normalization of the depth values based on the depth of focus
11. The depth of focus is saved in point_of_focus.
12. norm_depth_data holds the final normalized depth of focus.
13. 0 value corresponds to the point which is focused.
14. 1 value corresponds to the point furthest from the point of focus.
15. Mouse callback function, The point of the click is taken as the point of focus and defocusing is performed around it.
16. The depth map is normalized around the point of focus after which the defocusing of the image is done by sectioning and masking.
17. Generated different blurred versions of the original image.
18. The blurred versions are stored in the list blur_imgs.
19. Creates a window to display the original image.
20. The callback function is attached to this window.
21. End

6.3 Development Tools

6.3.1 VS Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C, Java, Python, PHP, Go) and runtimes (such as .NET and Unity). Visual Studio Code features a lightning-fast source code editor. Visual Studio Code includes built-in support for IntelliSense code completion, rich semantic code understanding and navigation, and code refactoring. Visual Studio Code includes a public extensibility model that lets developers build and use extensions, and richly customize their edit-build-debug experience.VS Code also integrates with build and scripting tools to perform common tasks making everyday workflows faster. VS Code has support for Git.

6.3.2 Google Colaboratory

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs. Colaboratory notebooks are stored in Google Drive and can be shared just as you would

with Google Docs or Sheets. Colaboratory is free to use. The main features provided by colaboratory includes :

• Colaboratory allows you to integrate PyTorch, TensorFlow, Keras, OpenCV

• Colaboratory includes widely used libraries like matplotlib, simplifying visualization

• Free Cloud service with free GPU

• Colab documents the code that supports mathematical equations

• All Colab notebooks are stored in the open-source Jupyter notebook format (.ipynb). So it's easy to download and for further usage.

6.3.3 Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Compared to other version control systems, Git is responsive, easy to use, and inexpensive (free, actually). Branching allows you to create independent local branches in your code.

Chapter 7

Testing

7.1 Testing Methodologies

Software testing methodologies are the various strategies or approaches used to test an application to ensure it behaves and looks as expected. The idea of using various testing methodologies in the development process is to ensure that the software can successfully work in multiple environments and different platforms. Broadly, this is broken down into functional and non-functional testing. In functional testing, the application is tested against business requirements. Non-functional testing methods incorporate all test types focused on the operational aspects of a piece of software. We have performed functional testing part which includes unit testing, integration testing and system testing, each of which is described below.

7.2 Unit Testing

Unit testing refers to the testing of individual software modules or components that make up an application or system. It validates that each module of the software performs as designed.

7.2.1 Depth Module

• Conditions:

1. Module accepts a image in .png file format.

2. Converts the image in .png file format to a _disp.npy depth map.

• Steps:

1. Input .png image file

2. Stop

Test Case Id	Input	Expected Output	Actual Output	Test Status
1	sample0.png	sample0_disp.npy	sample0_disp.npy	Pass
2	sample1.png	sample1_disp.npy	sample1_disp.npy	Pass
3	sample2.png	sample2_disp.npy	sample2_disp.npy	Pass
4	sample3.png	sample3_disp.npy	sample3_disp.npy	Pass
5	sample4.png	sample4_disp.npy	sample4_disp.npy	Pass

Table 7.1: Unit Testing : Depth Module

7.2.2 Defocus Module

• Conditions:

1. Module accepts a image in .png file format.

2. Module accepts the corresponding _disp.npy depth map.

3. Module produces the defocused image as _defocus.png.

• Steps:

1. Input .png image file.

2. Input the corresponding _disp.npy file.

3. Stop

Test Case Id	Input 1	Input 2	Expected Output	Actual Output	Test Status
1	sample0.png	sample0_disp.npy	sample0_defocus.png	sample0_defocus.png	Pass
2	sample1.png	sample1_disp.npy	sample1_defocus.png	sample1_defocus.png	Pass
3	sample2.png	sample2_disp.npy	sample2_defocus.png	sample2_defocus.png	Pass
4	sample3.png	sample3_disp.npy	sample3_defocus.png	sample3_defocus.png	Pass
5	sample4.png	sample4_disp.npy	sample4_defocus.png	sample4_defocus.png	Pass

Table 7.2: Unit Testing : Defocus Module

7.3 Integration Testing

In Integration Testing, different modules were combined and tested to see if the modules interact properly and produce the correct output.

• Conditions:

1. Depth Module accepts a image in .png file format.

2. Defocus Module accepts the corresponding _disp.npy depth map (the output of depth module) and input image.

3. Module produces the defocused image as _defocus.png.

• Steps:

1. Input .png image file.

3. Stop

Test Case Id	Input 1	Output1/Input 2	Expected Output	Actual Output	Test Status
1	sample0.png	sample0_disp.npy	sample0_defocus.png	sample0_defocus.png	Pass
2	sample1.png	sample1_disp.npy	sample1_defocus.png	sample1_defocus.png	Pass
3	sample2.png	sample2_disp.npy	sample2_defocus.png	sample2_defocus.png	Pass
4	sample3.png	sample3_disp.npy	sample3_defocus.png	sample3_defocus.png	Pass
5	sample4.png	sample4_disp.npy	sample4_defocus.png	sample4_defocus.png	Pass

Table 7.3: Integration Testing

7.4 System Testing

In system testing all the modules are integrated in order of execution. Testing is carried out over the whole system.

• Conditions:

1. Module accepts a image in .png file format.

• Steps:

1. Input .png image file.

3. Stop

Test Case Id	Input 1	Expected Output	Actual Output	Test Status
1	sample0.png	sample0_defocus.png	sample0_defocus.png	Pass
2	sample1.png	sample1_defocus.png	sample1_defocus.png	Pass
3	sample2.png	sample2_defocus.png	sample2_defocus.png	Pass
4	sample3.png	sample3_defocus.png	sample3_defocus.png	Pass
5	sample4.png	sample4_defocus.png	sample4_defocus.png	Pass

Table 7.4: System Testing

Chapter 8

Graphical User Interface

The graphical user interface is a user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notations, instead of text-based user interfaces, typed command labels or text navigation.GUIs were introduced in reaction to the perceived steep learning curve of command-interfaces which require a command to be typed on a computer keyboard.

Designing the visual composition and temporal behaviour of a GUI is an important part of software application programming in the area of human-computer interaction. Its goal is to enhance the efficiency and use case for the underlying logical design of a stored program, a designed discipline name usability. Methods for user-centred design are used to ensure that the visual language introduced in the design is well-tailored to the tasks.

For several decades GUIs were controlled exclusively by a mouse and keyboard. While these types of input devices are sufficient for desktop computers, they do not work as well for mobile devices, such as smartphones and tablets. Therefore, mobile operating systems are designed to use a touch screen. Many mobile devices can now be controlled by spoken commands as well.

Because there are now many types of digital devices available, GUIs must be designed for the appropriate type of input. For example, a desktop operating system, such as OS X, includes a menu bar and windows with small icons that can be easily navigated using a mouse. A mobile OS, like iOS, includes large icons and supports touch commands like swiping and pinching to zoom in or zoom out. Automotive interfaces are often designed to be controlled with knobs and buttons and TV interfaces are built to work with remote control. Regardless of the type of input each of these interfaces are considered GUIs since they include graphical elements.

8.1 GUI Overview

A GUI uses a combination of technologies and devices to provide a platform that users can interact with, for the task of gathering and producing information. Traditionally, browsers acted themselves as independent and isolated software. All the features that were necessary for the browsers

to perform, were implemented in browsers itself. Limitation of this method was that the users had no control over the performance of the browser, or at least they could not do anything to improve its performance.

Nowadays, browser performance can be improved by users, especially as the user demands. Users can either develop or get modules such as apps, extensions, plug-ins which can work within the browser. These modules are usually optional and can be installed to improve the user experience of the browser. These modules perform a specific task, which it is designed to do when the user asks to. For instance, there are browser apps which help to give users some reminder, browser extensions which can fill forms automatically, plugins which improve the visual effect of the browser.

For the proposed system, the GUI is developed using GTK toolkit, which is a is a free and open-source cross-platform widget toolkit for creating graphical user interfaces. It is licensed under the terms of the GNU Lesser General Public License, allowing both free and proprietary software to use it. It is one of the most popular toolkits for the Wayland and X11 windowing systems.

We use the gtk tool Glade which is a RAD tool to enable quick and easy development of user interfaces for the GTK toolkit and the GNOME desktop environment.

The user interfaces designed in Glade are saved as XML, and by using the GtkBuilder GTK object these can be loaded by applications dynamically as needed.

8.2 Main GUI Components

Figure 8.1: Home Page of the application

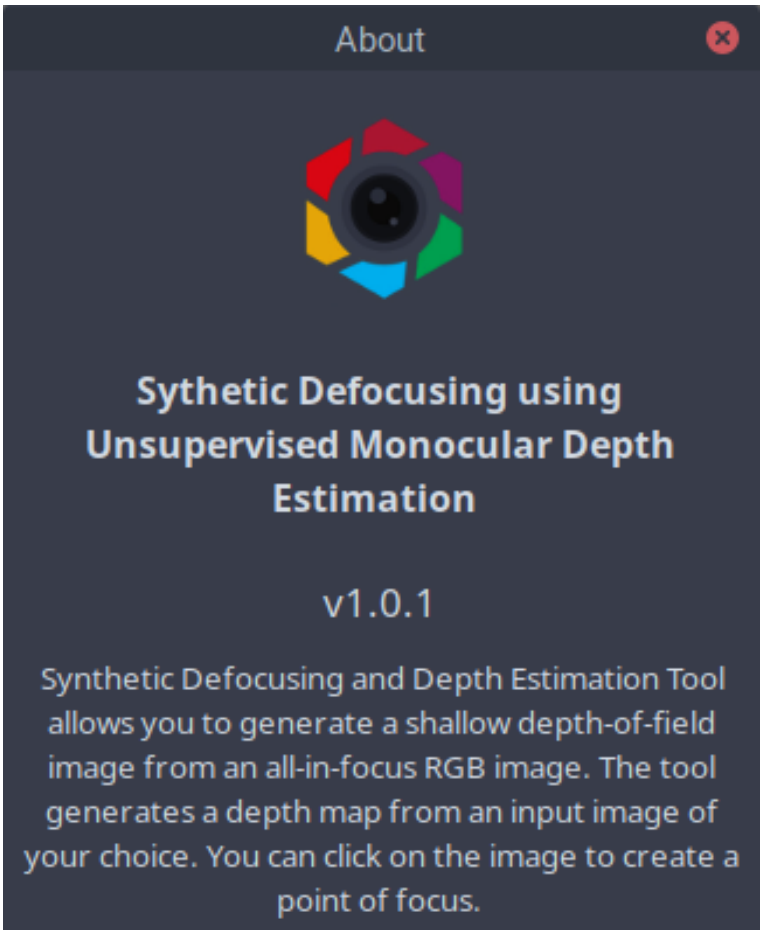


Figure 8.2: About Page , which describes what the application is about in easy to understand words.

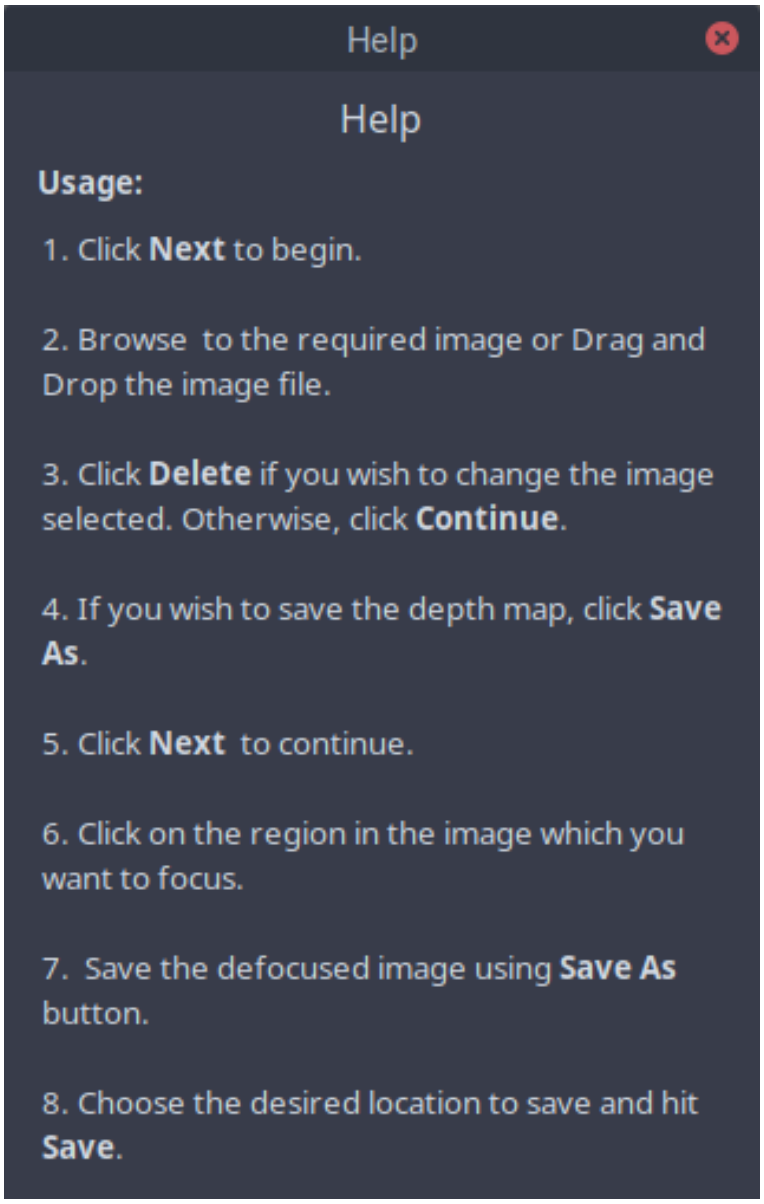


Figure 8.3: Help Page , which helps users by explaining the steps to use the application in simple point by point basis.

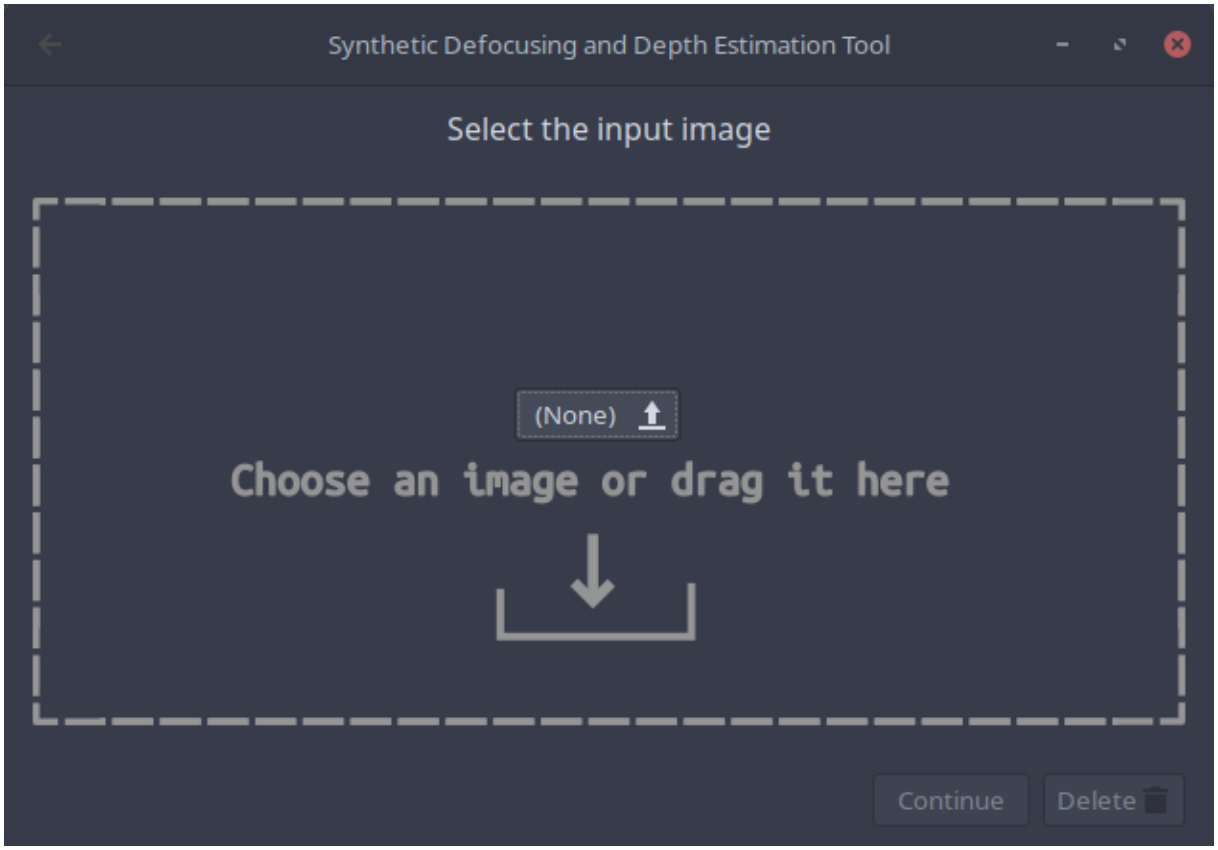


Figure 8.4: The input image can be selected from a folder or dragged and dropped into the application.

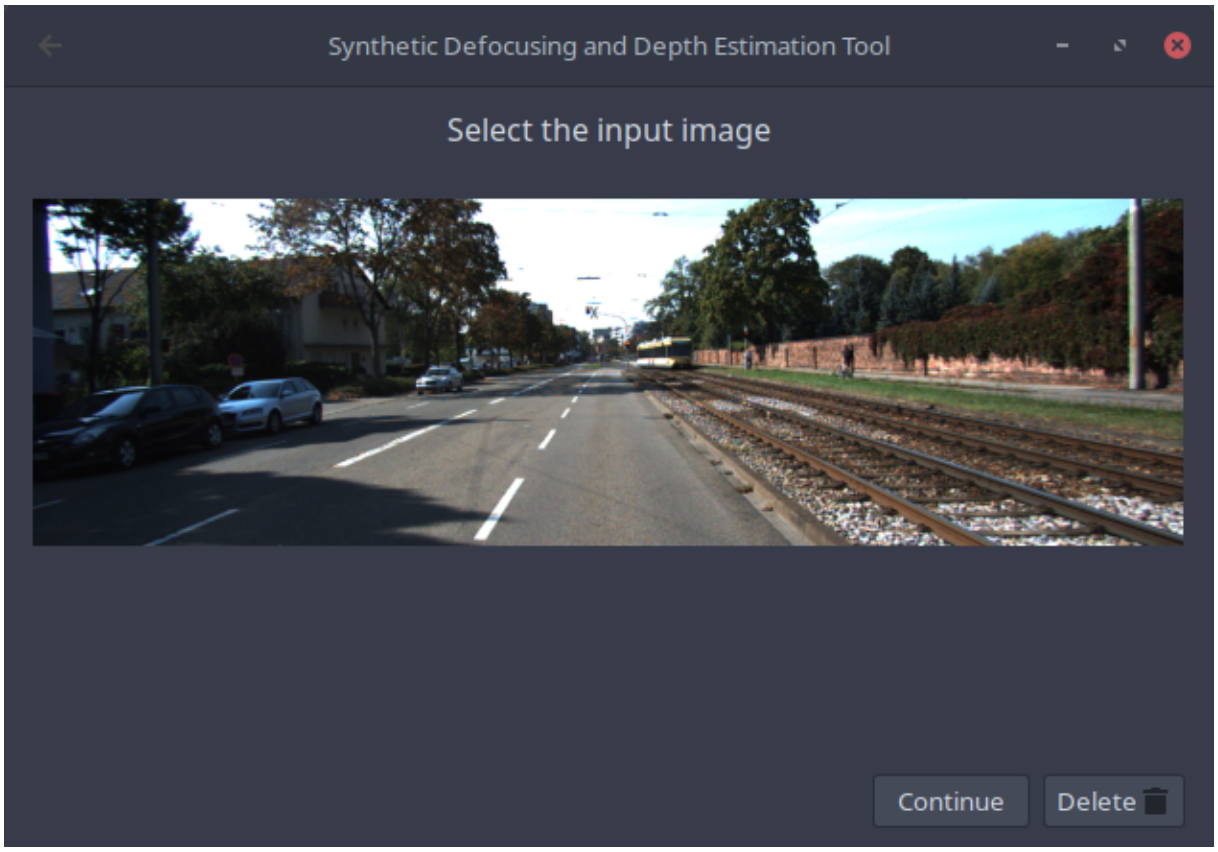


Figure 8.5: The selected image is displayed to the user.

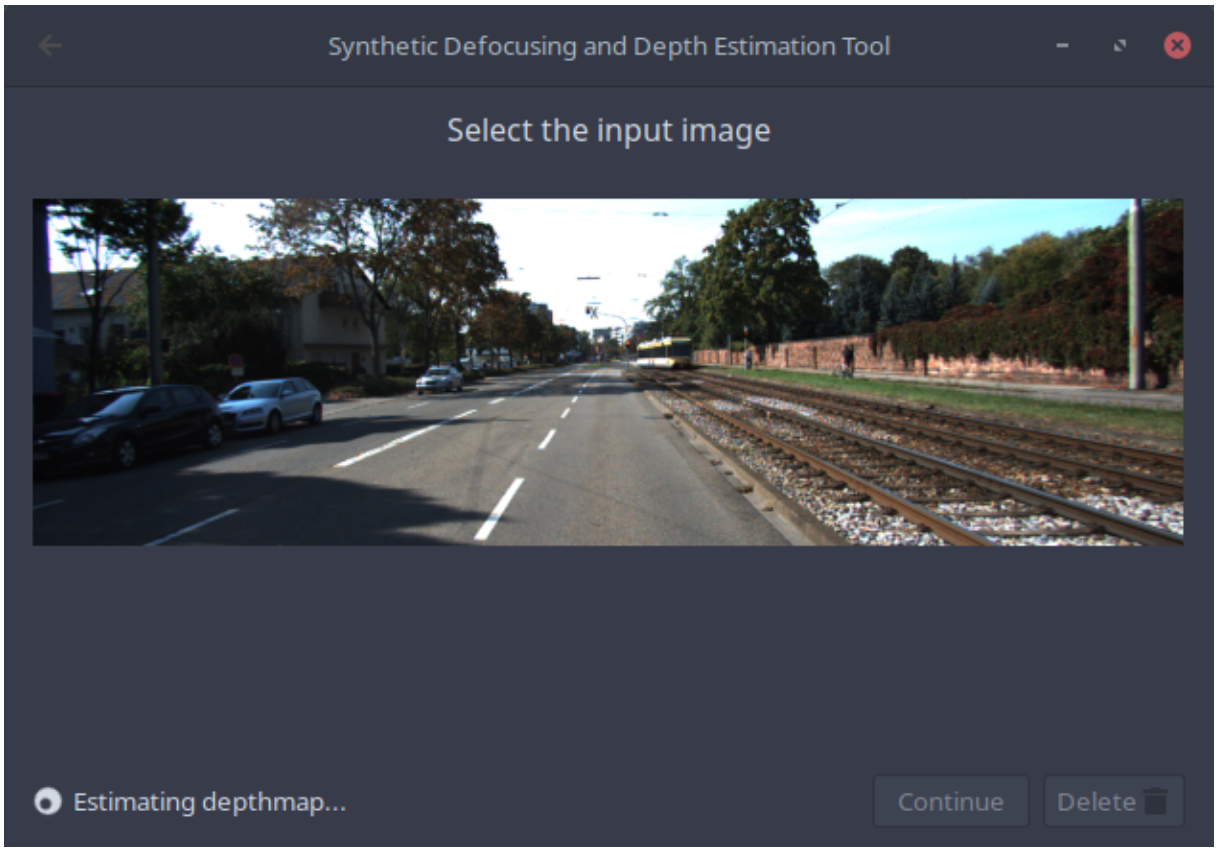


Figure 8.6: The depth map is being estimated.

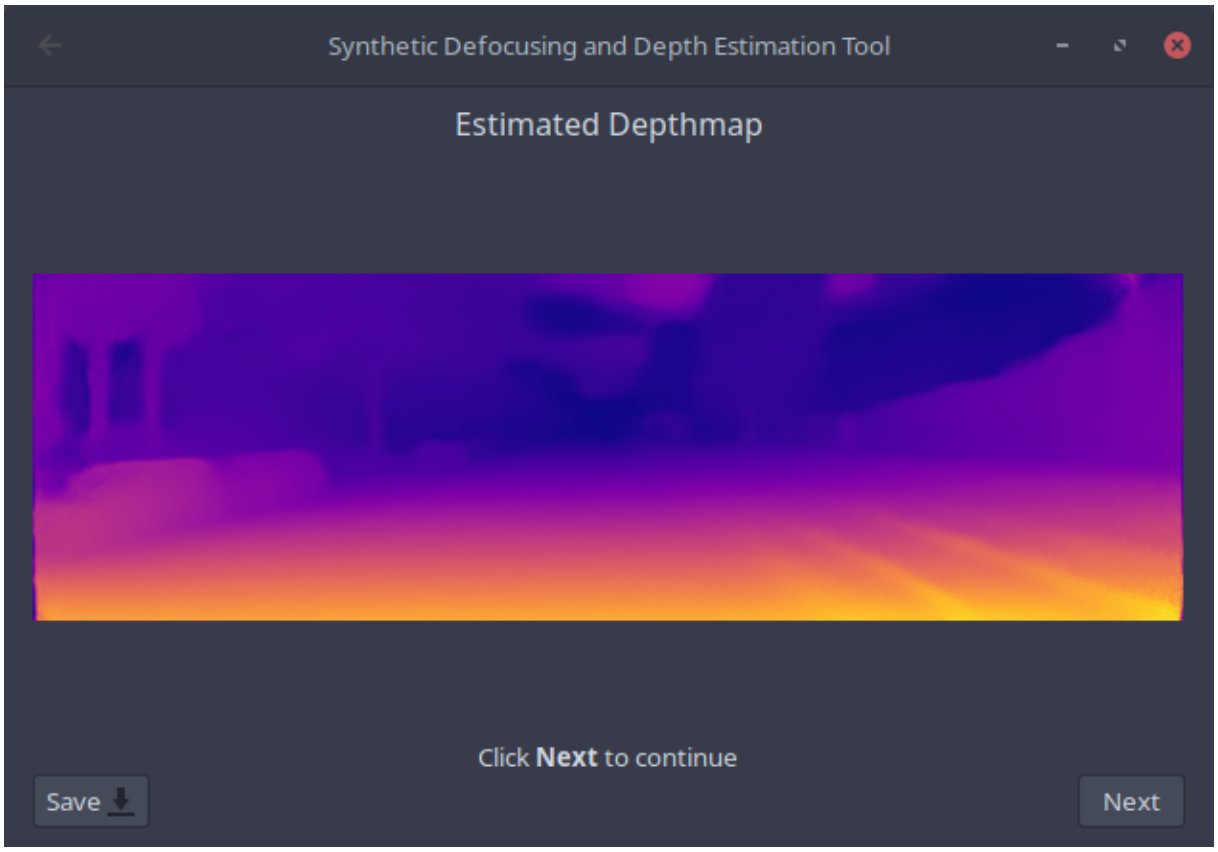


Figure 8.7: The depth map is displayed to the user.

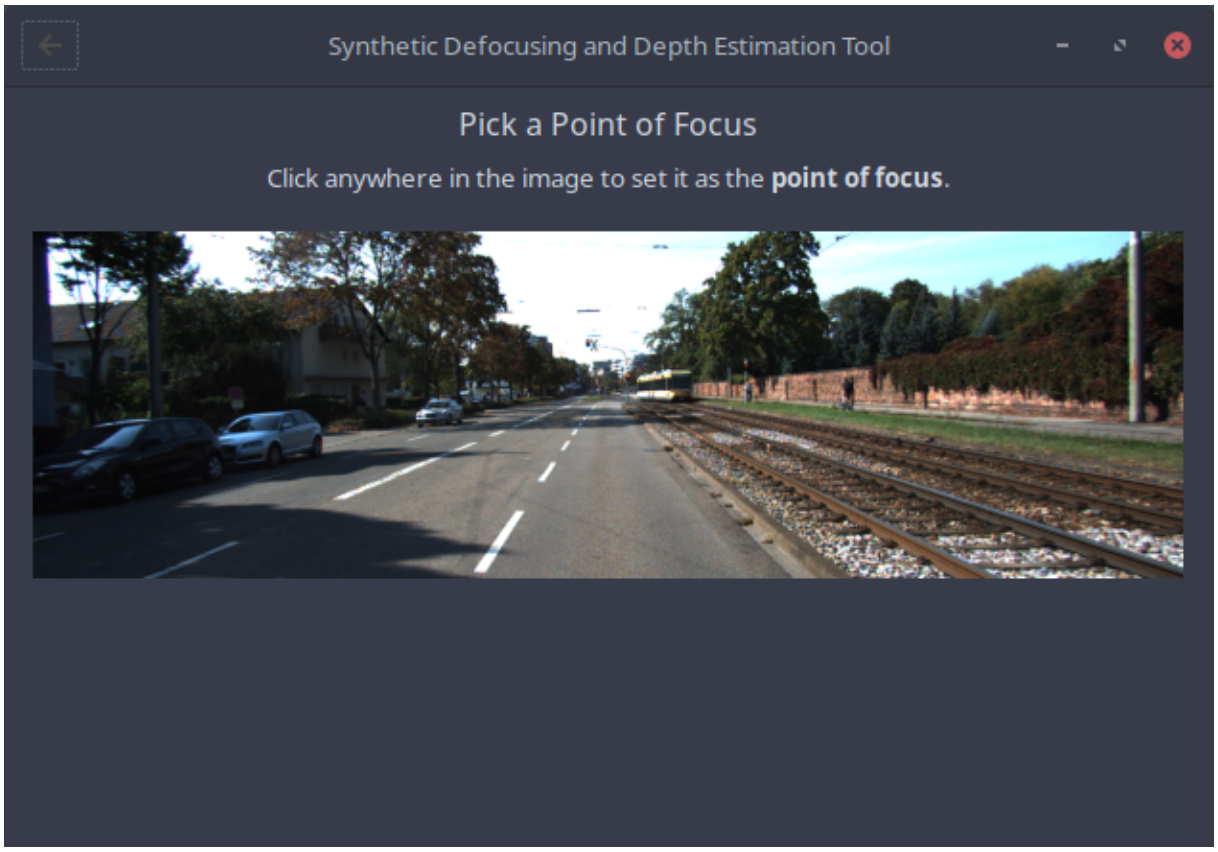


Figure 8.8: The point of focus is to be selected by the user using a mouse left-button click.

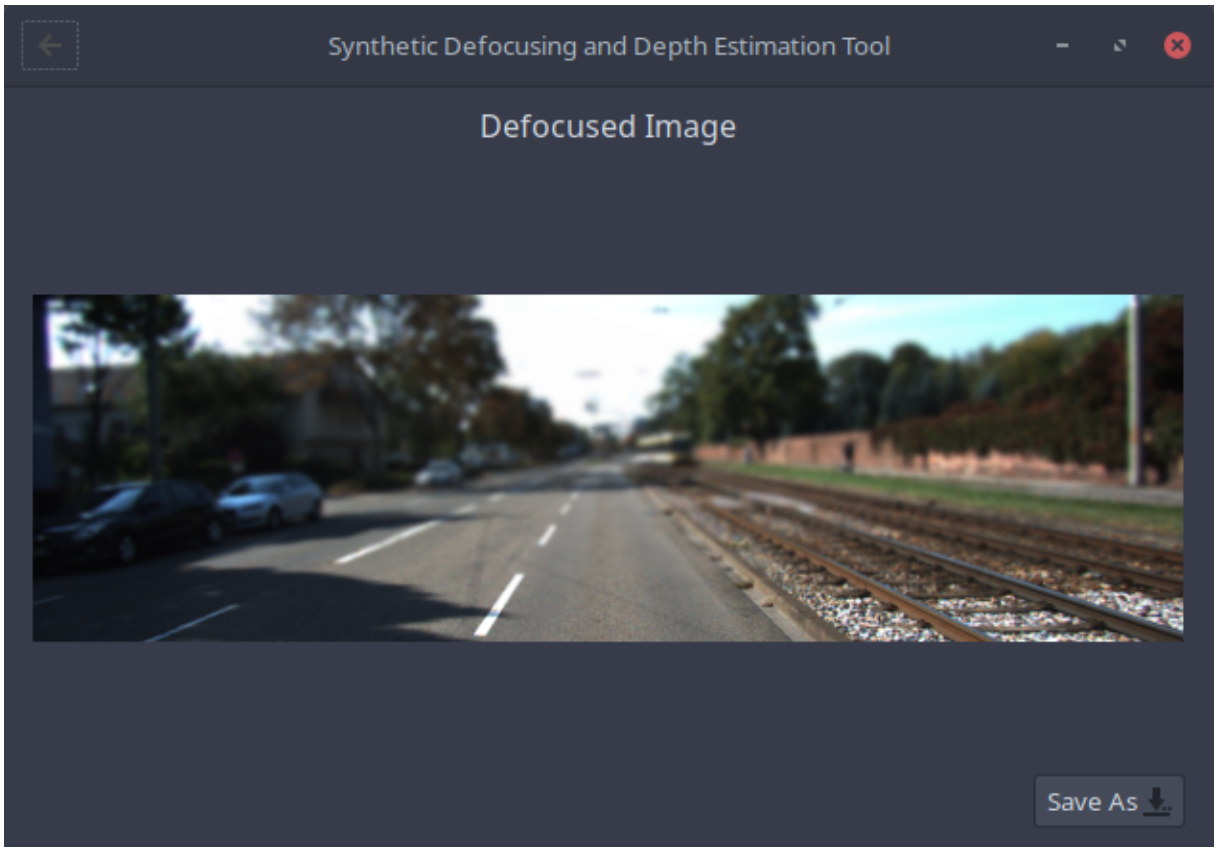


Figure 8.9: The defocused image is displayed to the user.

Chapter 9

Results

The application allows the users to input an image either by directly selecting image from the folder, or by dragging and dropping the image into the gui application.

Figure 9.1: The selected image is displayed to the user.

At the backend the depth estimation module produces a corresponding depth map for the input image, and displays it to the user as a png file which can be saved.

Figure 9.2: The depth map is displayed to the user.

Then the defocus module accepts the generated depth map as well as the input image to produce a sythetically defocused image where the user can select the point of focus using a mouse button click and the defocussed image is displayed where the image is blurred in a graded manner with respect to depth of focus selected by the user.

Figure 9.3: The defocused image is displayed to the user.

Chapter 10

Conclusion

MonoDeptDefocus is a scalable solution to producing a synthetically defocused images, using just a single input image and no extra information.The current system for this requires users to have multiple shots of the same image or expensive camera module for depth estimation or are less detailed and time consuming.The developed system accepts input image and displays the depth map and the defocussed image to the user in less than 2 seconds. It acts as an assistive tool for photo editors who are professionals as well as casual users. The process involves the generation of a depth map and then using the generated depth data along with the initial input image to produces a synthetically defocused image.

After manual testing on 5000 random test images from the KITTI dataset , the system has been found to efficiently produce a result on approximately 98% of the images.

Chapter 11

Future Scope

Currently the system only allows 4 different blurring option which are average blur, median blur, guassian blur and bilateral filtering. In the future there can be additional blurring methods as well as mixed blurring methods where the blurring method changes with respect to depth from point of focus.

Once the accuracy of the depth estimation increases with new methodologies in the future , the system can be used to produce accurate enough defocused images, which could be used in various domains including the military, scientific research, advertising industry etc.

The system can also be upgraded to synthetically defocus a video in real time which can be then used for video editing in the film and TV industry.

References

[1] Ashutosh Saxena, Min Sun and Andrew Y. Ng, "Learning 3-D Scene Structure from a Single Still Image", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), pp.1-8, 2008.

[2] Osamu Ikeda, "Shape Reconstruction from Two Color Images using Photometric Stereo combined with Segmentation and Stereopsis", IEEE - AV SS, pp. 434-438, 2005.

[3] Make3D: Depth Perception from a Single Still Image Ashutosh Saxena, Min Sun and Andrew Y. Ng.

[4] Beyang Liu, Stephen Gould, Daphne Koller, "Single Image Depth Estimation From Predicted Semantic Labels", IEEE conference on Computer Vision and Pattern Recognition: pp 1253-1260,2010.

[5] N.Otsu, "A threshold selection method from gray level histograms", IEEE Trans. Syst. Man Cybern. SMC-9, 62-66 (1979).

[6] Zhengyou Zhang, "A flexible new technique for camera calibration", IEEE Transactions on Pattern Analysis and Machine Intelligence 22(11) : 1330-1334 (2000).

[7] Clément Godard , Oisin Mac Aodha , Gabriel J. Brostow, "Unsupervised Monocular Depth Estimation with Left-Right Consistency", CVPR, 2017

Unsupervised Monocular Depth Estimation with Left-Right Consistency

Clément Godard Oisin Mac Aodha Gabriel J. Brostow
University College London
http://visual.cs.ucl.ac.uk/pubs/monoDepth/

Abstract

Learning based methods have shown very promising results for the task of depth estimation in single images. However, most existing approaches treat depth prediction as a supervised regression problem and as a result, require vast quantities of corresponding ground truth depth data for training. Just recording quality depth data in a range of environments is a challenging problem. In this paper, we innovate beyond existing approaches, replacing the use of explicit depth data during training with easier-to-obtain binocular stereo footage.

We propose a novel training objective that enables our convolutional neural network to learn to perform single image depth estimation, despite the absence of ground truth depth data. Exploiting epipolar geometry constraints, we generate disparity images by training our network with an image reconstruction loss. We show that solving for image reconstruction alone results in poor quality depth images. To overcome this problem, we propose a novel training loss that enforces consistency between the disparities produced relative to both the left and right images, leading to improved performance and robustness compared to existing approaches. Our method produces state of the art results for monocular depth estimation on the KITTI driving dataset, even outperforming supervised methods that have been trained with ground truth depth.

1. Introduction

Depth estimation from images has a long history in computer vision. Fruitful approaches have relied on structure from motion, shape-from-X, binocular, and multi-view stereo. However, most of these techniques rely on the assumption that multiple observations of the scene of interest are available. These can come in the form of multiple viewpoints, or observations of the scene under different lighting conditions. To overcome this limitation, there has recently been a surge in the number of works that pose the task of monocular depth estimation as a supervised learning problem [32, 10, 36]. These methods attempt to directly predict the depth of each pixel in an image using models that have been trained offline on large collections of ground truth depth data. While these methods have enjoyed great success, to date they have been restricted to scenes where large image collections and their corresponding pixel depths are available.

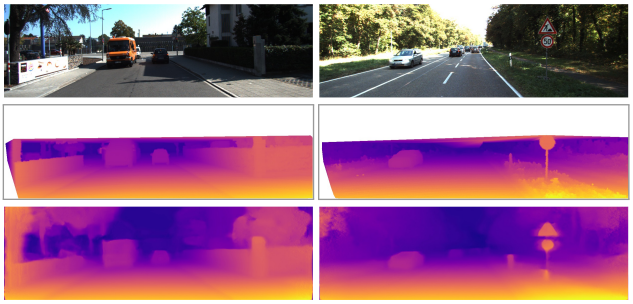


Figure 1. Our depth prediction results on KITTI 2015. Top to bottom: input image, ground truth disparities, and our result. Our method is able to estimate depth for thin structures such as street signs and poles.

Understanding the shape of a scene from a single image, independent of its appearance, is a fundamental problem in machine perception. There are many applications such as synthetic object insertion in computer graphics [29], synthetic depth of field in computational photography [3], grasping in robotics [34], using depth as a cue in human body pose estimation [48], robot assisted surgery [49], and automatic 2D to 3D conversion in film [53]. Accurate depth data from one or more cameras is also crucial for self-driving cars, where expensive laser-based systems are often used.

Humans perform well at monocular depth estimation by exploiting cues such as perspective, scaling relative to the known size of familiar objects, appearance in the form of lighting and shading and occlusion [24]. This combination of both top-down and bottom-up cues appears to link full scene understanding with our ability to accurately estimate depth. In this work, we take an alternative approach and treat automatic depth estimation as an image reconstruction problem during training. Our fully convolutional model does not require any depth data, and is instead trained to synthesize depth as an intermediate. It learns to predict the pixel-level correspondence between pairs of rectified stereo images that have a known camera baseline. There are some existing methods that also address the same problem, but with several limitations. For example they are not fully differentiable, making training suboptimal [16], or have image formation models that do

not scale to large output resolutions [53]. We improve upon these methods with a novel training objective and enhanced network architecture that significantly increases the quality of our final results. An example result from our algorithm is illustrated in Fig. 1. Our method is fast and only takes on the order of 35 milliseconds to predict a dense depth map for a 512×256 image on a modern GPU. Specifically, we propose the following contributions:

1) A network architecture that performs end-to-end unsupervised monocular depth estimation with a novel training loss that enforces left-right depth consistency inside the network.

2) An evaluation of several training losses and image formation models highlighting the effectiveness of our approach.

3) In addition to showing state of the art results on a challenging driving dataset, we also show that our model generalizes to three different datasets, including a new outdoor urban dataset that we have collected ourselves, which we make openly available.

2. Related Work

There is a large body of work that focuses on depth estimation from images, either using pairs [46], several overlapping images captured from different viewpoints [14], temporal sequences [44], or assuming a fixed camera, static scene, and changing lighting [52, 2]. These approaches are typically only applicable when there is more than one input image available of the scene of interest. Here we focus on works related to monocular depth estimation, where there is only a single input image, and no assumptions about the scene geometry or types of objects present are made.

Learning-Based Stereo

The vast majority of stereo estimation algorithms have a data term which computes the similarity between each pixel in the first image and every other pixel in the second image. Typically the stereo pair is rectified and thus the problem of disparity (*i.e.* scaled inverse depth) estimation can be posed as a 1D search problem for each pixel. Recently, it has been shown that instead of using hand defined similarity measures, treating the matching as a supervised learning problem and training a function to predict the correspondences produces far superior results [54, 31]. It has also been shown that posing this binocular correspondence search as a multi-class classification problem has advantages both in terms of quality of results and speed [38]. Instead of just learning the matching function, Mayer et al. [39] introduced a fully convolutional [47] deep network called DispNet that directly computes the correspondence field between two images. At training time, they attempt to directly predict the disparity for each pixel by minimizing a regression training loss. DispNet has a similar architecture to their previous end-to-end deep optical flow network [12].

The above methods rely on having large amounts of accurate ground truth disparity data and stereo image pairs at training time. This type of data can be difficult to obtain for real world scenes, so these approaches typically use synthetic data for training. Synthetic data is becoming more realistic, *e.g.* [15], but still requires the manual creation of new content for every new application scenario.

Supervised Single Image Depth Estimation

Single-view, or monocular, depth estimation refers to the problem setup where only a single image is available at test time. Saxena et al. [45] proposed a patch-based model known as Make3D that first over-segments the input image into patches and then estimates the 3D location and orientation of local planes to explain each patch. The predictions of the plane parameters are made using a linear model trained offline on a dataset of laser scans, and the predictions are then combined together using an MRF. The disadvantage of this method, and other planar based approximations, *e.g.* [22], is that they can have difficulty modeling thin structures and, as predictions are made locally, lack the global context required to generate realistic outputs. Instead of hand-tuning the unary and pairwise terms, Liu et al. [36] use a convolutional neural network (CNN) to learn them. In another local approach, Ladicky et al. [32] incorporate semantics into their model to improve their per pixel depth estimation. Karsch et al. [28] attempt to produce more consistent image level predictions by copying whole depth images from a training set. A drawback of this approach is that it requires the entire training set to be available at test time.

Eigen et al. [10, 9] showed that it was possible to produce dense pixel depth estimates using a two scale deep network trained on images and their corresponding depth values. Unlike most other previous work in single image depth estimation, they do not rely on hand crafted features or an initial over-segmentation and instead learn a representation directly from the raw pixel values. Several works have built upon the success of this approach using techniques such as CRFs to improve accuracy [35], changing the loss from regression to classification [5], using other more robust loss functions [33], and incorporating strong scene priors in the case of the related problem of surface normal estimation [50]. Again, like the previous stereo methods, these approaches rely on having high quality, pixel aligned, ground truth depth at training time. We too perform single depth image estimation, but train with an added binocular color image, instead of requiring ground truth depth.

Unsupervised Depth Estimation

Recently, a small number of deep network based methods for novel view synthesis and depth estimation have been proposed, which do not require ground truth depth at training time. Flynn et al. [13] introduced a novel image synthesis network called DeepStereo that generates new views by selecting pixels from nearby images. During training, the relative pose of multiple cameras is used to predict the appearance of a held-out nearby image. Then the most appropriate depths are selected to sample colors from the neighboring images, based on plane sweep volumes.

At test time, image synthesis is performed on small overlapping patches. As it requires several nearby posed images at test time DeepStereo is not suitable for monocular depth estimation.

The Deep3D network of Xie et al. [53] also addresses the problem of novel view synthesis, where their goal is to generate the corresponding right view from an input left image (*i.e.* the source image) in the context of binocular pairs. Again using an image reconstruction loss, their method produces a distribution over all the possible disparities for each pixel. The resulting synthesized right image pixel values are a combination of the pixels on the same scan line from the left image, weighted by the probability of each disparity. The disadvantage of their image formation model is that increasing the number of candidate disparity values greatly increases the memory consumption of the algorithm, making it difficult to scale their approach to bigger output resolutions. In this work, we perform a comparison to the Deep3D image formation model, and show that our algorithm produces superior results.

Closest to our model in spirit is the concurrent work of Garg et al. [16]. Like Deep3D and our method, they train a network for monocular depth estimation using an image reconstruction loss. However, their image formation model is not fully differentiable. To compensate, they perform a Taylor approximation to linearize their loss resulting in an objective that is more challenging to optimize. Similar to other recent work, *e.g.* [43, 56, 57], our model overcomes this problem by using bilinear sampling [27] to generate images, resulting in a fully (sub-)differentiable training loss.

We propose a fully convolutional deep neural network loosely inspired by the supervised DispNet architecture of Mayer et al. [39]. By posing monocular depth estimation as an image reconstruction problem, we can solve for the disparity field without requiring ground truth depth. However, only minimizing a photometric loss can result in good quality image reconstructions but poor quality depth. Among other terms, our fully differentiable training loss includes a left-right consistency check to improve the quality of our synthesized depth images. This type of consistency check is commonly used as a post-processing step in many stereo methods, *e.g.* [54], but we incorporate it directly into our network.

3. Method

This section describes our single image depth prediction network. We introduce a novel depth estimation training loss, featuring an inbuilt left-right consistency check, which enables us to train on image pairs without requiring supervision in the form of ground truth depth.

3.1. Depth Estimation as Image Reconstruction

Given a single image I at test time, our goal is to learn a function f that can predict the per-pixel scene depth, $\hat{d}=f(I)$. Most existing learning based approaches treat this as a supervised learning problem, where they have color input images and their corresponding target depth values at training. It is presently not practical to acquire such ground truth depth data for a large variety of scenes. Even expensive hardware, such as laser scanners, can be imprecise in natural scenes featuring movement and reflections. As an alternative, we instead pose depth estimation as an image reconstruction problem during training. The intuition here is that, given a calibrated pair of binocular cameras, if we can learn a function that is able to reconstruct one image from the other, then we have learned something about the 3D shape of the scene that is being imaged.

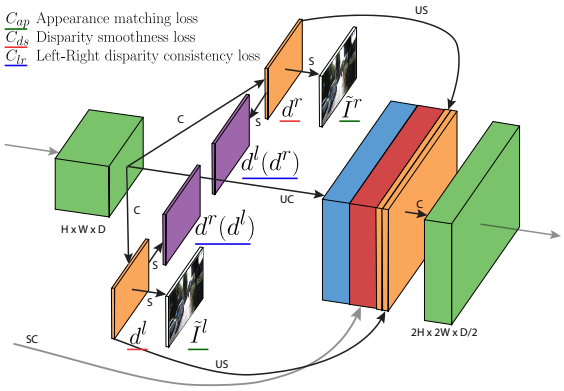


Figure 2. Our loss module outputs left and right disparity maps, d^l and d^r. The loss combines smoothness, reconstruction, and left-right disparity consistency terms. This same module is repeated at each of the four different output scales. C: Convolution, UC: Up-Convolution, S: Bilinear Sampling, US: Up-Sampling, SC: Skip Connection.

Specifically, at training time, we have access to two images I^l and I^r, corresponding to the left and right color images from a calibrated stereo pair, captured at the same moment in time. Instead of trying to directly predict the depth, we attempt to find the dense correspondence field d^r that, when applied to the left image, would enable us to reconstruct the right image. We will refer to the reconstructed image $I^l(d^r)$ as $\tilde{I}^r$. Similarly, we can also estimate the left image given the right one, $\tilde{I}^l=I^r(d^l)$. Assuming that the images are rectified [19], d corresponds to the image disparity - a scalar value per pixel that our model will learn to predict. Given the baseline distance b between the cameras and the camera focal length f, we can then trivially recover the depth $\hat{d}$ from the predicted disparity, $\hat{d}=bf/d$.

3.2. Depth Estimation Network

At a high level, our network estimates depth by inferring the disparities that warp the left image to match the right one. The key insight of our method is that we can simultaneously infer both disparities (left-to-right and right-to-left), using only the left input image, and obtain better depths by enforcing them to be consistent with each other.

Our network generates the predicted image with backward mapping using a bilinear sampler, resulting in a fully differentiable image formation model. As illustrated in Fig. 3, naïvely learning to generate the right image by sampling from the left

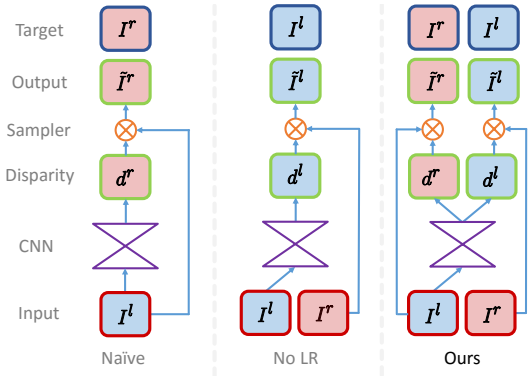


Figure 3. Sampling strategies for backward mapping. With naïve sampling the CNN produces a disparity map aligned with the target instead of the input. No LR corrects for this, but suffers from artifacts. Our approach uses the left image to produce disparities for both images, improving quality by enforcing mutual consistency.

one will produce disparities aligned with the right image (target). However, we want the output disparity map to align with the input left image, meaning the network has to sample from the right image. We could instead train the network to generate the left view by sampling from the right image, thus creating a left view aligned disparity map (**No LR** in Fig. 3). While this alone works, the inferred disparities exhibit 'texture-copy' artifacts and errors at depth discontinuities as seen in Fig. 5. We solve this by training the network to predict the disparity maps for both views by sampling from the opposite input images. This still only requires a single left image as input to the convolutional layers and the right image is only used during training (**Ours** in Fig. 3). Enforcing consistency between both disparity maps using this novel left-right consistency cost leads to more accurate results.

Our fully convolutional architecture is inspired by DispNet [39], but features several important modifications that enable us to train without requiring ground truth depth. Our network, is composed of two main parts - an encoder (from cnv1 to cnv7b) and decoder (from upcnv7), please see the supplementary material for a detailed description. The decoder uses skip connections [47] from the encoder's activation blocks, enabling it to resolve higher resolution details. We output disparity predictions at four different scales (disp4 to disp1), which double in spatial resolution at each of the subsequent scales. Even though it only takes a single image as input, our network predicts two disparity maps at each output scale - left-to-right and right-to-left.

3.3. Training Loss

We define a loss C_s at each output scale s, forming the total loss as the sum $C = \sum_{s=1}^{4} C_s$. Our loss module (Fig. 2) computes C_s as a combination of three main terms,

$$C_s = \alpha_{ap}(C_{ap}^l + C_{ap}^r) + \alpha_{ds}(C_{ds}^l + C_{ds}^r) + \alpha_{lr}(C_{lr}^l + C_{lr}^r), \tag{1}$$

where C_{ap} encourages the reconstructed image to appear similar to the corresponding training input, C_{ds} enforces smooth disparities, and C_{lr} prefers the predicted left and right disparities to be consistent. Each of the main terms contains both a left and a right image variant, but only the left image is fed through the convolutional layers.

Next, we present each component of our loss in terms of the left image (*e.g.* C_{ap}^l). The right image versions, *e.g.* C_{ap}^r, require to swap left for right and to sample in the opposite direction.

Appearance Matching Loss During training, the network learns to generate an image by sampling pixels from the opposite stereo image. Our image formation model uses the image sampler from the spatial transformer network (STN) [27] to sample the input image using a disparity map. The STN uses bilinear sampling where the output pixel is the weighted sum of four input pixels. In contrast to alternative approaches [16, 53], the bilinear sampler used is locally fully differentiable and integrates seamlessly into our fully convolutional architecture. This means that we do not require any simplification or approximation of our cost function.

Inspired by [55], we use a combination of an $L1$ and single scale SSIM [51] term as our photometric image reconstruction cost C_{ap}, which compares the input image I_{ij}^l and its reconstruction $\tilde{I}_{ij}^l$, where N is the number of pixels,

$$C_{ap}^l = \frac{1}{N}\sum_{i,j} \alpha \frac{1 - \text{SSIM}(I_{ij}^l, \tilde{I}_{ij}^l)}{2} + (1-\alpha)\left\| I_{ij}^l - \tilde{I}_{ij}^l \right\|. \tag{2}$$

Here, we use a simplified SSIM with a 3×3 block filter instead of a Gaussian, and set $\alpha = 0.85$.

Disparity Smoothness Loss We encourage disparities to be locally smooth with an $L1$ penalty on the disparity gradients ∂d. As depth discontinuities often occur at image gradients, similar to [21], we weight this cost with an edge-aware term using the image gradients ∂I,

$$C_{ds}^l = \frac{1}{N}\sum_{i,j} |\partial_x d_{ij}^l| e^{-\|\partial_x I_{ij}^l\|} + |\partial_y d_{ij}^l| e^{-\|\partial_y I_{ij}^l\|}. \tag{3}$$

Left-Right Disparity Consistency Loss To produce more accurate disparity maps, we train our network to predict both the left and right image disparities, while only being given the left view as input to the convolutional part of the network. To ensure coherence, we introduce an $L1$ left-right disparity consistency penalty as part of our model. This cost attempts to make the left-view disparity map be equal to the *projected* right-view disparity map,

$$C_{lr}^l = \frac{1}{N}\sum_{i,j} \left| d_{ij}^l - d_{ij + d_{ij}^l}^r \right|. \tag{4}$$

Like all the other terms, this cost is mirrored for the right-view disparity map and is evaluated at all of the output scales.

Method	Dataset	Abs Rel	Sq Rel	RMSE	RMSE log	*D1-all*	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Ours with Deep3D [53]	K	0.412	16.37	13.693	0.512	66.85	0.690	0.833	0.891
Ours with Deep3Ds [53]	K	0.151	1.312	6.344	0.239	59.64	0.781	0.931	0.976
Ours No LR	K	0.123	1.417	6.315	0.220	30.318	0.841	0.937	0.973
Ours	K	0.124	1.388	6.125	0.217	30.272	0.841	0.936	0.975
Ours	CS	0.699	10.060	14.445	0.542	94.757	0.053	0.326	0.862
Ours	CS + K	0.104	1.070	5.417	0.188	25.523	0.875	0.956	0.983
Ours pp	CS + K	0.100	0.934	5.141	0.178	25.077	0.878	0.961	**0.986**
Ours resnet pp	CS + K	**0.097**	**0.896**	**5.093**	**0.176**	**23.811**	**0.879**	**0.962**	**0.986**
Ours Stereo	K	0.068	0.835	4.392	0.146	9.194	0.942	0.978	0.989

Lower is better

Higher is better

Table 1. Comparison of different image formation models. Results on the KITTI 2015 stereo 200 training set disparity images [17]. For training, K is the KITTI dataset [17] and CS is Cityscapes [8]. Our model with left-right consistency performs the best, and is further improved with the addition of the Cityscapes data. The last row shows the result of our model trained *and tested* with two input images instead of one (see Sec. 4.3).

At test time, our network predicts the disparity at the finest scale level for the left image d^l, which has the same resolution as the input image. Using the known camera baseline and focal length from the training set, we then convert from the disparity map to a depth map. While we also estimate the right disparity d^r during training, it is not used at test time.

4. Results

Here we compare the performance of our approach to both supervised and unsupervised single view depth estimation methods. We train on rectified stereo image pairs, and do not require any supervision in the form of ground truth depth. Existing single image datasets, such as [41, 45], that lack stereo pairs, are not suitable for evaluation. Instead we evaluate our approach using the popular KITTI 2015 [17] dataset. To evaluate our image formation model, we compare to a variant of our algorithm that uses the original Deep3D [53] image formation model and a modified one, Deep3Ds, with an added smoothness constraint. We also evaluate our approach with and without the left-right consistency constraint.

4.1. Implementation Details

The network which is implemented in TensorFlow [1] contains 31 million trainable parameters, and takes on the order of 25 hours to train using a single Titan X GPU on a dataset of 30 thousand images for 50 epochs. Inference is fast and takes less than 35 ms, or more than 28 frames per second, for a 512×256 image, including transfer times to and from the GPU. Please see the supplementary material and our code[1] for more details.

During optimization, we set the weighting of the different loss components to $\alpha_{ap} = 1$ and $\alpha_{lr} = 1$. The possible output disparities are constrained to be between 0 and d_{max} using a scaled sigmoid non-linearity, where $d_{max} = 0.3\times$ the image width at a given output scale. As a result of our multi-scale output, the typical disparity of neighboring pixels will differ by a factor of two between each scale (as we are upsampling the output by a factor of two). To correct for this, we scale the disparity smoothness term α_{ds} with r for each scale to get equivalent smoothing at each level. Thus $\alpha_{ds} = 0.1/r$, where r is the downscaling factor of the corresponding layer with respect to the resolution of the input image that is passed into the network.

For the non-linearities in the network, we used exponential linear units [7] instead of the commonly used rectified liner units (ReLU) [40]. We found that ReLUs tended to prematurely fix the predicted disparities at intermediate scales to a single value, making subsequent improvement difficult. Following [42], we replaced the usual deconvolutions with a nearest neighbor upsampling followed by a convolutions. We trained our model from scratch for 50 epochs, with a batch size of 8 using Adam [30], where $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We used an initial learning rate of $\lambda = 10^{-4}$ which we kept constant for the first 30 epochs before halving it every 10 epochs until the end. We initially experimented with progressive update schedules, as in [39], where lower resolution image scales were optimized first. However, we found that optimizing all four scales at once led to more stable convergence. Similarly, we use an identical weighting for the loss of each scale as we found that weighting them differently led to unstable convergence. We experimented with batch normalization [26], but found that it did not produce a significant improvement, and ultimately excluded it.

Data augmentation is performed on the fly. We flip the input images horizontally with a 50% chance, taking care to also swap both images so they are in the correct position relative to each other. We also added color augmentations, with a 50% chance, where we performed random gamma, brightness, and color shifts by sampling from uniform distributions in the ranges $[0.8, 1.2]$ for gamma, $[0.5, 2.0]$ for brightness, and $[0.8, 1.2]$ for each color channel separately.

Resnet50 For the sake of completeness, and similar to [33], we also show a variant of our model using Resnet50 [20] as the encoder, the rest of the architecture, parameters and training procedure staying identical. This variant contains 48 million trainable parameters and is indicated by **resnet** in result tables.

Post-processing In order to reduce the effect of stereo disocclusions which create disparity ramps on both the left side of the image and of the occluders, a final post-processing step is performed on the output. For an input image I at test time, we also

[1] Available at https://github.com/mrharicot/monodepth

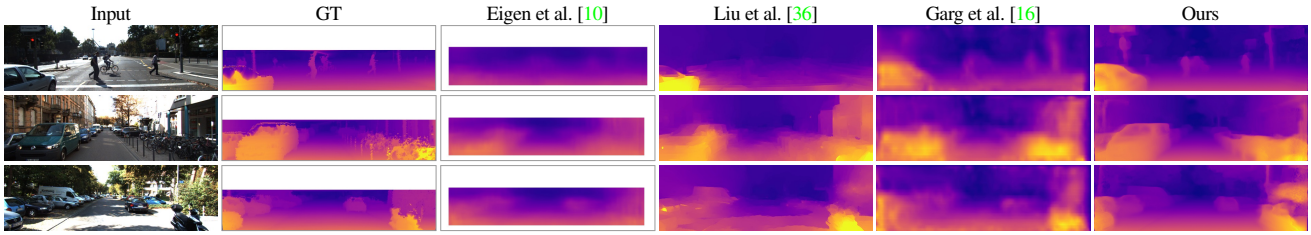


Figure 4. Qualitative results on the KITTI Eigen Split. The ground truth velodyne depth being very sparse, we interpolate it for visualization purposes. Our method does better at resolving small objects such as the pedestrians and poles.

compute the disparity map d_l' for its horizontally flipped image I'. By flipping back this disparity map we obtain a disparity map d_l'', which aligns with d_l but where the disparity ramps are located on the right of occluders as well as on the right side of the image. We combine both disparity maps to form the final result by assigning the first 5% on the left of the image using d_l'' and the last 5% on the right to the disparities from d_l. The central part of the final disparity map is the average of d_l and d_l'. This final post-processing step leads to both better accuracy and less visual artifacts at the expense of doubling the amount of test time computation. We indicate such results using **pp** in result tables.

4.2. KITTI

We present results for the KITTI dataset [17] using two different test splits, to enable comparison to existing works. In its raw form, the dataset contains 42,382 rectified stereo pairs from 61 scenes, with a typical image being 1242×375 pixels in size.

KITTI Split First we compare different variants of our method including different image formation models and different training sets. We evaluate on the 200 high quality disparity images provided as part of the official KITTI training set, which covers a total of 28 scenes. The remaining 33 scenes contain 30,159 images from which we keep 29,000 for training and the rest for evaluation. While these disparity images are much better quality than the reprojected velodyne laser depth values, they have CAD models inserted in place of moving cars. These CAD models result in ambiguous disparity values on transparent surfaces such as car windows, and issues at object boundaries where the CAD models do not perfectly align with the images. In addition, the maximum depth present in the KITTI dataset is on the order of 80 meters, and we cap the maximum predictions of all networks to this value. Results are computed using the depth metrics from [10] along with the *D1-all* disparity error from KITTI [17]. The metrics from [10] measure error in both meters from the ground truth and the percentage of depths that are within some threshold from the correct value. It is important to note that measuring the error in depth space while the ground truth is given in disparities leads to precision issues. In particular, the non-thresholded measures can be sensitive to the large errors in depth caused by prediction errors at small disparity values.

In Table 1, we see that in addition to having poor scaling properties (in terms of both resolution and the number of disparities it can represent), when trained from scratch with the same network architecture as ours, the Deep3D [53] image formation model performs poorly. From Fig. 6 we can see that Deep3D produces plausible image reconstructions but the output disparities are inferior to ours. Our loss outperforms both the Deep3D baselines and the addition of the left-right consistency check increases performance in all measures. In Fig. 5 we illustrate some zoomed in comparisons, clearly showing that the inclusion of the left-right check improves the visual quality of the results. Our results are further improved by first pre-training our model with additional training data from the Cityscapes dataset [8] containing 22,973 training stereo pairs captured in various cities across Germany. This dataset brings higher resolution, image quality, and variety compared to KITTI, while having a similar setting. We cropped the input images to only keep the top 80% of the image, removing the very reflective car hoods from the input. Interestingly, our model trained on Cityscapes alone does not perform very well numerically. This is likely due to the difference in camera calibration between the two datasets, but there is a clear advantage to fine-tuning on data that is related to the test set.

Eigen Split To be able to compare to existing work, we also use the test split of 697 images as proposed by [10] which covers a total of 29 scenes. The remaining 32 scenes contain 23,488 images from which we keep 22,600 for training and the rest for evaluation, similarly to [16]. To generate the ground truth depth images, we reproject the 3D points viewed from the velodyne laser into the left input color camera. Aside from only producing depth values for less than 5% of the pixels in the input image, errors are also introduced because of the rotation

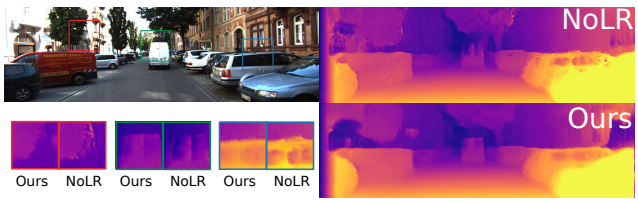


Figure 5. Comparison between our method with and without the left-right consistency. Our consistency term produces superior results on the object boundaries. Both results are shown without post-processing.

Method	Supervised	Dataset	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Train set mean	No	K	0.361	4.826	8.102	0.377	0.638	0.804	0.894
Eigen et al. [10] Coarse °	Yes	K	0.214	1.605	6.563	0.292	0.673	0.884	0.957
Eigen et al. [10] Fine °	Yes	K	0.203	1.548	6.307	0.282	0.702	0.890	0.958
Liu et al. [36] DCNF-FCSP FT *	Yes	K	0.201	1.584	6.471	0.273	0.68	0.898	0.967
Ours No LR	No	K	0.152	1.528	6.098	0.252	0.801	0.922	0.963
Ours	No	K	0.148	1.344	5.927	0.247	0.803	0.922	0.964
Ours	No	CS + K	0.124	1.076	5.311	0.219	0.847	0.942	0.973
Ours pp	No	CS + K	0.118	0.923	5.015	0.210	0.854	0.947	**0.976**
Ours resnet pp	No	CS + K	**0.114**	**0.898**	**4.935**	**0.206**	**0.861**	**0.949**	**0.976**
Garg et al. [16] L12 Aug 8× cap 50m	No	K	0.169	1.080	5.104	0.273	0.740	0.904	0.962
Ours cap 50m	No	K	0.140	0.976	4.471	0.232	0.818	0.931	0.969
Ours cap 50m	No	CS + K	0.117	0.762	3.972	0.206	0.860	0.948	0.976
Ours pp cap 50m	No	CS + K	0.112	0.680	3.810	0.198	0.866	0.953	**0.979**
Ours resnet pp cap 50m	No	CS + K	**0.108**	**0.657**	**3.729**	**0.194**	**0.873**	**0.954**	**0.979**
Our pp uncropped	No	CS + K	0.134	1.261	5.336	0.230	0.835	0.938	0.971
Ours resnet pp uncropped	No	CS + K	0.130	1.197	5.222	0.226	0.843	0.940	0.971

Lower is better

Higher is better

Table 2. Results on KITTI 2015 [17] using the split of Eigen et al. [10]. For training, K is the KITTI dataset [17] and CS is Cityscapes [8]. The predictions of Liu et al. [36]* are generated on a mix of the left and right images instead of just the left input images. For a fair comparison, we compute their results relative to the correct image. As in the provided source code, Eigen et al. [10]° results are computed relative to the velodyne instead of the camera. Garg et al. [16] results are taken directly from their paper. All results, except [10], use the crop from [16]. We also show our results with the same crop and maximum evaluation distance. The last two rows are computed on the uncropped ground truth.

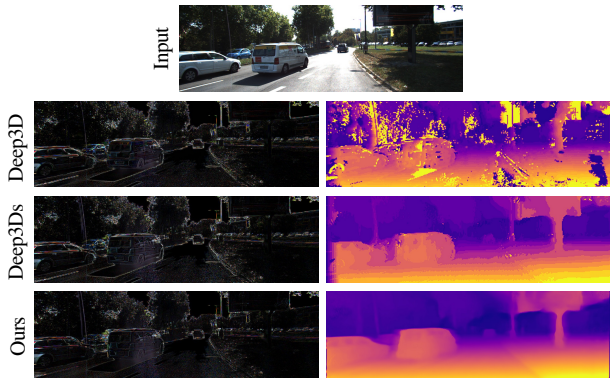


Figure 6. Image reconstruction error on KITTI. While all methods output plausible right views, the Deep3D image formation model without smoothness constraints does not produce valid disparities.

of the Velodyne, the motion of the vehicle and surrounding objects, and also incorrect depth readings due to occlusion at object boundaries. To be fair to all methods, we use the same crop as [10] and evaluate at the input image resolution. With the exception of Garg et al.'s [16] results, the results of the baseline methods are recomputed by us given the authors's original predictions to ensure that all the scores are directly comparable. This produces slightly different numbers than the previously published ones, *e.g.* in the case of [10], their predictions were evaluated on much smaller depth images ($1/4$ the original size). For all baseline methods we use bilinear interpolation to resize the predictions to the correct input image size.

Table 2 shows quantitative results with some example outputs shown in Fig. 4. We see that our algorithm outperforms all other existing methods, including those that are trained with ground truth depth data. We again see that pre-training on the Cityscapes dataset improves the results over using KITTI alone.

4.3. Stereo

We also implemented a stereo version of our model, see Fig. 8, where the network's input is the concatenation of both left and right views. Perhaps unsurprisingly, the stereo models outperforms our monocular network on every single metric, especially on the *D1-all* disparity measure, as can be seen in Table 1. This model was only trained for 12 epochs as it becomes unstable if trained for longer.

4.4. Make3D

To illustrate that our method can generalize to other datasets, here we compare to several fully supervised methods on the Make3D test set of [45]. Make3D consists of only RGB/Depth pairs and no stereo images, thus our method cannot train on this data. We use our network trained only on the Cityscapes dataset and despite the dissimilarities in the datasets, both in content and camera parameters, we still achieve reasonable results, even beating [28] on one metric and [37] on three. Due to the different aspect ratio of the Make3D dataset we evaluate on a central crop of the images. In Table 3, we compare our output to the similarly cropped results of the other methods. As in the case of the KITTI dataset, these results would likely be improved with more relevant training data. A qualitative comparison to some of the related methods is shown in Fig. 7. While our numerical results are not as good as the baselines, qualitatively, we compare favorably to the supervised competition.

4.5. Generalizing to Other Datasets

Finally, we illustrate some further examples of our model generalizing to other datasets in Figure 9. Using the model only trained on Cityscapes [8], we tested on the CamVid driving dataset [4]. In the accompanying video and the supplementary material we can see that despite the differences in location, image characteristics, and camera calibration, our model still

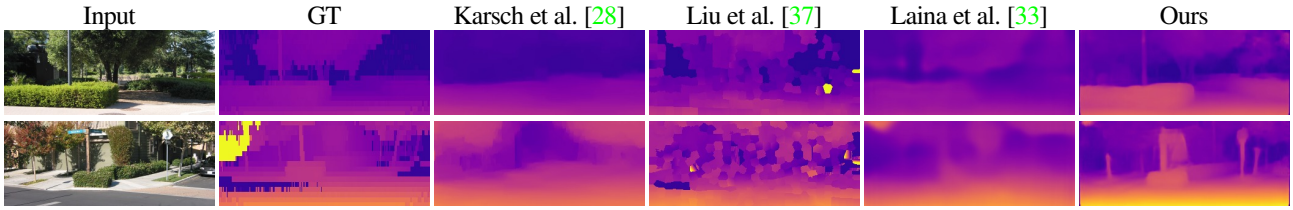


Figure 7. Our method achieves superior qualitative results on Make3D despite being trained on a different dataset (Cityscapes).

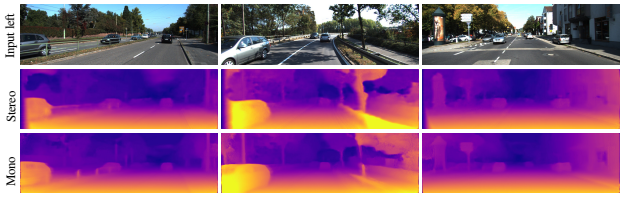


Figure 8. Our stereo results. While the stereo disparity maps contains more detail, our monocular results are comparable.

Method	Sq Rel	Abs Rel	RMSE	$\log_{10}$
Train set mean*	15.517	0.893	11.542	0.223
Karsch et al. [28]*	4.894	0.417	8.172	0.144
Liu et al. [37]*	6.625	0.462	9.972	0.161
Laina et al. [33] berHu*	**1.665**	**0.198**	**5.461**	**0.082**
Ours with Deep3D [53]	17.18	1.000	19.11	2.527
Ours	11.990	0.535	11.513	0.156
Ours pp	7.112	0.443	8.860	0.142

Table 3. Results on the Make3D dataset [45]. All methods marked with an * are supervised and use ground truth depth data from the Make3D training set. Using the standard C1 metric, errors are only computed where depth is less than 70 meters in a central image crop.

produces visually plausible depths. We also captured a 60,000 frame dataset, at 10 frames per second, taken in an urban environment with a wide angle consumer 1080p stereo camera. Finetuning the Cityscapes pre-trained model on this dataset produces visually convincing depth images for a test set that was captured with the same camera on a different day, please see the video in the supplementary material for more results.

Figure 9. Qualitative results on Cityscapes, CamVid, and our own urban dataset captured on foot. For more results please see our video.

4.6. Limitations

Even though both our left-right consistency check and post-processing improve the quality of the results, there are still some artifacts visible at occlusion boundaries due to the pixels in the occlusion region not being visible in both images. Explicitly reasoning about occlusion during training [23, 25] could improve these issues. It is worth noting that depending how large the baseline between the camera and the depth sensor, fully supervised approaches also do not always have valid depth for all pixels.

Our method requires rectified and temporally aligned stereo pairs during training, which means that it is currently not possible to use existing single-view datasets for training purposes *e.g.* [41]. However, it is possible to fine-tune our model on application specific ground truth depth data.

Finally, our method mainly relies on the image reconstruction term, meaning that specular [18] and transparent surfaces will produce inconsistent depths. This could be improved with more sophisticated similarity measures [54].

5. Conclusion

We have presented an unsupervised deep neural network for single image depth estimation. Instead of using aligned ground truth depth data, which is both rare and costly, we exploit the ease with which binocular stereo data can be captured. Our novel loss function enforces consistency between the predicted depth maps from each camera view during training, improving predictions. Our results are superior to fully supervised baselines, which is encouraging for future research that does not require expensive to capture ground truth depth. We have also shown that our model can generalize to unseen datasets and still produce visually plausible depth maps.

In future work, we would like to extend our model to videos. While our current depth estimates are performed independently per frame, adding temporal consistency [28] would likely improve results. It would also be interesting to investigate sparse input as an alternative training signal [58, 6]. Finally, while our model estimates per pixel depth, it would be interesting to also predict the full occupancy of the scene [11].

Acknowledgments We would like to thank David Eigen, Ravi Garg, Iro Laina and Fayao Liu for providing data and code to recreate the baseline algorithms. We also thank Stephan Garbin for his lua skills and Peter Hedman for his LaTeX magic. We are grateful for EPSRC funding for the EngD Centre EP/G037159/1, and for projects EP/K015664/1 and EP/K023578/1.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 5

[2] A. Abrams, C. Hawley, and R. Pless. Heliometric stereo: Shape from sun position. In *ECCV*, 2012. 2

[3] J. T. Barron, A. Adams, Y. Shih, and C. Hernández. Fast bilateral-space stereo for synthetic defocus. *CVPR*, 2015. 1

[4] G. J. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 2009. 7

[5] Y. Cao, Z. Wu, and C. Shen. Estimating depth from monocular images as classification using deep fully convolutional residual networks. *arXiv preprint arXiv:1605.02305*, 2016. 2

[6] W. Chen, Z. Fu, D. Yang, and J. Deng. Single-image depth perception in the wild. In *NIPS*, 2016. 8

[7] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015. 5

[8] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 5, 6, 7

[9] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, 2015. 2

[10] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, 2014. 1, 2, 6, 7

[11] M. Firman, O. Mac Aodha, S. Julier, and G. J. Brostow. Structured Prediction of Unobserved Voxels From a Single Depth Image. In *CVPR*, 2016. 8

[12] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015. 2

[13] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *CVPR*, 2016. 2

[14] Y. Furukawa and C. Hernández. Multi-view stereo: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2015. 2

[15] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016. 2

[16] R. Garg, V. Kumar BG, and I. Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. In *ECCV*, 2016. 1, 3, 4, 6, 7

[17] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. 5, 6, 7

[18] C. Godard, P. Hedman, W. Li, and G. J. Brostow. Multi-view reconstruction of highly specular surfaces in uncontrolled environments. In *3DV*, 2015. 8

[19] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. 3

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[21] P. Heise, S. Klose, B. Jensen, and A. Knoll. Pm-huber: Patchmatch with huber regularization for stereo matching. In *ICCV*, 2013. 4

[22] D. Hoiem, A. A. Efros, and M. Hebert. Automatic photo pop-up. *TOG*, 2005. 2

[23] D. Hoiem, A. N. Stein, A. A. Efros, and M. Hebert. Recovering occlusion boundaries from a single image. In *ICCV*, 2007. 8

[24] I. P. Howard. *Perceiving in depth, volume 1: basic mechanisms*. Oxford University Press, 2012. 1

[25] A. Humayun, O. Mac Aodha, and G. J. Brostow. Learning to Find Occlusion Regions. In *CVPR*, 2011. 8

[26] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 5

[27] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *NIPS*, 2015. 3, 4

[28] K. Karsch, C. Liu, and S. B. Kang. Depth transfer: Depth extraction from video using non-parametric sampling. *PAMI*, 2014. 2, 7, 8

[29] K. Karsch, K. Sunkavalli, S. Hadap, N. Carr, H. Jin, R. Fonte, M. Sittig, and D. Forsyth. Automatic scene inference for 3d object compositing. *TOG*, 2014. 1

[30] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[31] L. Ladickỳ, C. Häne, and M. Pollefeys. Learning the matching function. *arXiv preprint arXiv:1502.00652*, 2015. 2

[32] L. Ladickỳ, J. Shi, and M. Pollefeys. Pulling things out of perspective. In *CVPR*, 2014. 1, 2

[33] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *3DV*, 2016. 2, 5, 8

[34] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 2015. 1

[35] B. Li, C. Shen, Y. Dai, A. van den Hengel, and M. He. Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs. In *CVPR*, 2015. 2

[36] F. Liu, C. Shen, G. Lin, and I. Reid. Learning depth from single monocular images using deep convolutional neural fields. *PAMI*, 2015. 1, 2, 6, 7

[37] M. Liu, M. Salzmann, and X. He. Discrete-continuous depth estimation from a single image. In *CVPR*, 2014. 7, 8

[38] W. Luo, A. Schwing, and R. Urtasun. Efficient deep learning for stereo matching. In *CVPR*, 2016. 2

[39] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016. 2, 3, 4, 5

[40] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 5

[41] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 5, 8

[42] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. 5

[43] V. Patraucean, A. Handa, and R. Cipolla. Spatio-temporal video autoencoder with differentiable memory. *arXiv preprint arXiv:1511.06309*, 2015. 3

[44] R. Ranftl, V. Vineet, Q. Chen, and V. Koltun. Dense monocular depth estimation in complex dynamic scenes. In *CVPR*, 2016. 2

[45] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *PAMI*, 2009. 2, 5, 7, 8

[46] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 2002. 2

[47] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *PAMI*, 2016. 2, 4

[48] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 2013. 1

[49] D. Stoyanov, M. V. Scarzanella, P. Pratt, and G.-Z. Yang. Real-time stereo reconstruction in robotically assisted minimally invasive surgery. In *MICCAI*, 2010. 1

[50] X. Wang, D. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. In *CVPR*, 2015. 2

[51] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Transactions on Image Processing*, 2004. 4

[52] R. J. Woodham. Photometric method for determining surface orientation from multiple images. *Optical engineering*, 1980. 2

[53] J. Xie, R. Girshick, and A. Farhadi. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *ECCV*, 2016. 1, 2, 3, 4, 5, 6, 8

[54] J. Žbontar and Y. LeCun. Stereo matching by training a convolutional neural network to compare image patches. *JMLR*, 2016. 2, 3, 8

[55] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. Is l2 a good loss function for neural networks for image processing? *arXiv preprint arXiv:1511.08861*, 2015. 4

[56] T. Zhou, P. Krähenbühl, M. Aubry, Q. Huang, and A. A. Efros. Learning dense correspondence via 3d-guided cycle consistency. *CVPR*, 2016. 3

[57] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros. View synthesis by appearance flow. In *ECCV*, 2016. 3

[58] D. Zoran, P. Isola, D. Krishnan, and W. T. Freeman. Learning ordinal relationships for mid-level vision. In *ICCV*, 2015. 8

Unsupervised Monocular Depth Estimation with Left-Right Consistency

Supplementary Material

1. Model architecture

"Encoder"						
layer	**k**	**s**	**chns**	**in**	**out**	**input**
conv1	7	2	3/32	1	2	left
conv1b	7	1	32/32	2	2	conv1
conv2	5	2	32/64	2	4	conv1b
conv2b	5	1	64/64	4	4	conv2
conv3	3	2	64/128	4	8	conv2b
conv3b	3	1	128/128	8	8	conv3
conv4	3	2	128/256	8	16	conv3b
conv4b	3	1	256/256	16	16	conv4
conv5	3	2	256/512	16	32	conv4b
conv5b	3	1	512/512	32	32	conv5
conv6	3	2	512/512	32	64	conv5b
conv6b	3	1	512/512	64	64	conv6
conv7	3	2	512/512	64	128	conv6b
conv7b	3	1	512/512	128	128	conv7

"Decoder"						
layer	**k**	**s**	**chns**	**in**	**out**	**input**
upconv7	3	2	512/512	128	64	conv7b
iconv7	3	1	1024/512	64	64	upconv7+conv6b
upconv6	3	2	512/512	64	32	iconv7
iconv6	3	1	1024/512	32	32	upconv6+conv5b
upconv5	3	2	512/256	32	16	iconv6
iconv5	3	1	512/256	16	16	upconv5+conv4b
upconv4	3	2	256/128	16	8	iconv5
iconv4	3	1	128/128	8	8	upconv4+conv3b
disp4	3	1	128/2	8	8	iconv4
upconv3	3	2	128/64	8	4	iconv4
iconv3	3	1	130/64	4	4	upconv3+conv2b+disp4*
disp3	3	1	64/2	4	4	iconv3
upconv2	3	2	64/32	4	2	iconv3
iconv2	3	1	66/32	2	2	upconv2+conv1b+disp3*
disp2	3	1	32/2	2	2	iconv2
upconv1	3	2	32/16	2	1	iconv2
iconv1	3	1	18/16	1	1	upconv1+disp2*
disp1	3	1	16/2	1	1	iconv1

Table 1: Our network architecture, where **k** is the kernel size, **s** the stride, **chns** the number of input and output channels for each layer, **in**put and **out**put is the downscaling factor for each layer relative to the input image, and **input** corresponds to the input of each layer where $+$ is a concatenation and $*$ is a $2\times$ upsampling of the layer.

2. Post-Processing

The post-processed disparity map corresponds to the per-pixel weighted sum of two components: d^l the disparity of the input image, d''^l the flipped disparity of the flipped input image.
We define the per-pixel weight map w^l for d^l as

$$w^l(i,j) = \begin{cases} 1 & \text{if } j \leq 0.1 \\ 0.5 & \text{if } j > 0.2 \\ 5*(0.2-i)+0.5 & \text{else,} \end{cases}$$

where i,j are normalized pixel coordinates, and the weight map w'^l for d''^l is obtained by horizontally flipping w^l.
The final disparity is calculated as,

$$d = d^l w^l + d''^l w'^l.$$

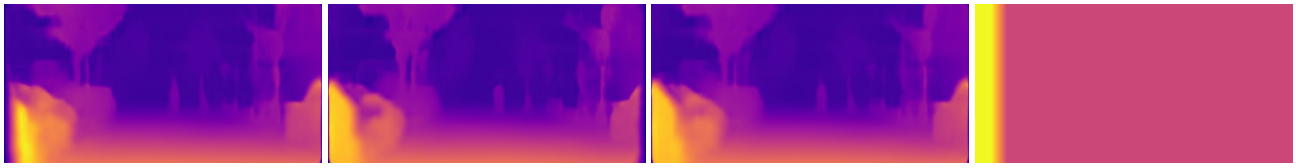


Figure 1: Example of a post-processed disparity map. From left to right: The disparities d^l, d''^l, d, and the weight map w^l.

3. Deep3D Smoothness Loss

In the main paper we also compared to our enhanced version of the Deep3D [1] image formation model that includes smoothness constraints on the disparities. Deep3D outputs an intensity image as a weighted sum of offset copies of the input image. The weights w_i, seen in Fig. 2a, can be seen as a discrete probability distribution over the disparities for each pixel, as they sum to one. Thus, smoothness constraints cannot be applied directly onto these distributions. However, we see (in Fig. 2c) that if the probability mass is concentrated into one disparity, *i.e.* $\max(w) \approx 1$, then the sum of the cumulative sum of the weights is equal to the position of the maximum. To encourage the network to concentrate probability at single disparities, we added a cost,

$$C_{max} = \frac{1}{N} \|\max(w_i) - 1\|^2, \tag{1}$$

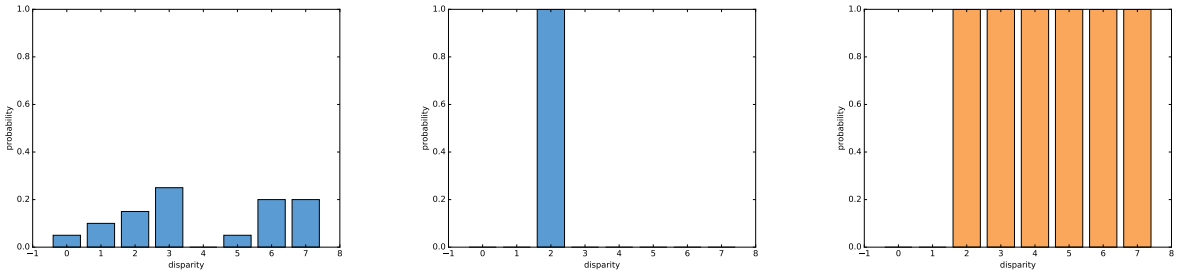
and its associated weight $\alpha_{max} = 0.02$. Assuming the maximum of each distribution is one, such as in Fig. 2b, meaning the network only picks one disparity per pixel, we can see that the (sum of the) cumulative sum $\mathrm{cs}(w_i)$ of the distribution (Fig. 2c) directly relates to the location of the maximum disparity:

$$d = \underset{i}{\mathrm{argmax}}(w_i) = n - \sum_i^n \mathrm{cs}(w_i), \tag{2}$$

where n is the maximum number of disparities.
In the example presented in Fig. 2, we can see that the maximum is located at disparity 3, and that Equation 2 gives us $d = 8 - 6 = 3$.
We use this observation to build our smoothness constraint for the Deep3D image formation model.
We can then directly apply smoothness constraints on the gradients of the cumulative sums of the weights at each pixel, so

$$C_{ds} = \frac{1}{N} \sum_{i,j} |\partial_x \mathrm{cs}(w)_{ij}| + |\partial_y \mathrm{cs}(w)_{ij}|, \tag{3}$$

and its associated weight $\alpha_{ds} = 0.1$.



(a) Probability over disparities (b) Probability over disparities (c) Cumulative sum of Fig. 2b

Figure 2: Deep3Ds per-pixel disparity probabilities.

4. More KITTI Qualitative Results

In Fig. 3 we show some additional qualitative comparisons for the KITTI dataset using the Eigen split.

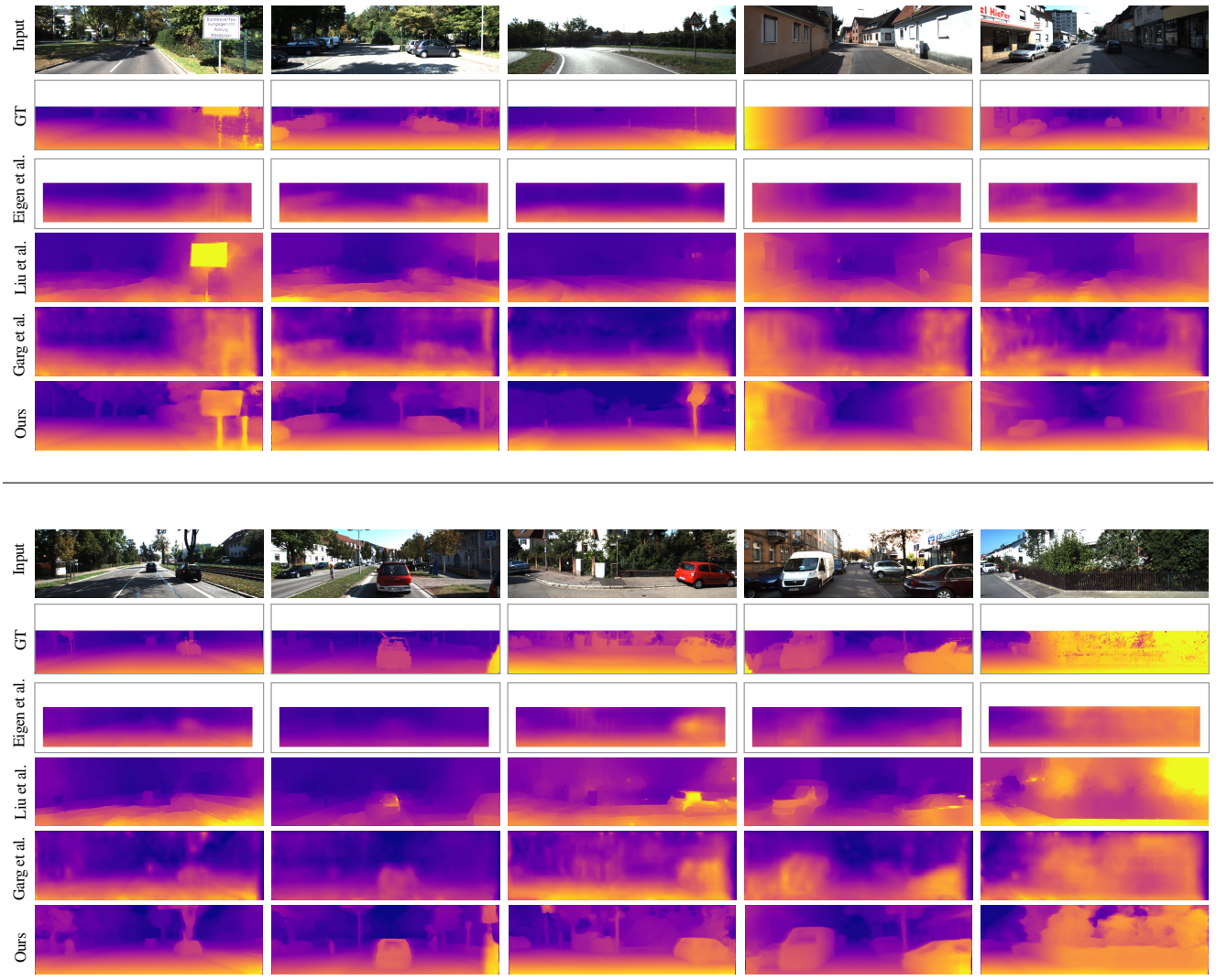


Figure 3: Additional qualitative results on the KITTI Eigen Split. As ground truth velodyne depth is very sparse, we interpolate it for visualization purposes.

5. Disparity Error Maps

As we train our model with color augmentations, we can apply the same principle at test time and analyze the distribution of the results. We applied 50 random augmentations to each test images and show the standard deviation of the disparities per pixel (see Figure 4). We can see that the network gets confused with close-by objects, texture-less regions, and occlusion boundaries. Interestingly, one test image was captured in a tunnel, resulting in a very dark image. Our network clearly shows high uncertainty for this sample as it is very different from the rest of the training set.

References

[1] J. Xie, R. Girshick, and A. Farhadi. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *ECCV*, 2016. 2

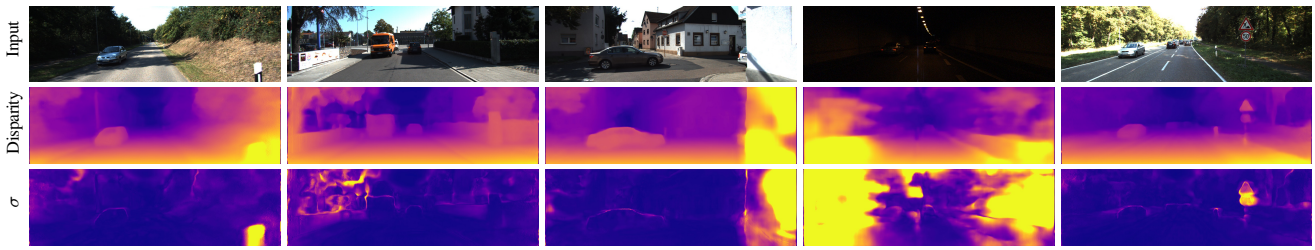


Figure 4: Uncertainty of our model on the KITTI dataset. From top to bottom: input image, predicted disparity, and standard deviation of multiple different augmentations. We can see that there is uncertainty in low texture regions and at occlusion boundaries.