Software Design Specification for Monocular Depth Estimation Model

Prepared by: Group 11

Krishnapriya Rajeev	B39	krishofficial05@gmail.com
Namitha S Vijayan	B48	namithasvijayan@gmail.com
Saharsh Santhosh Babu	B62	saharshsbabu@gmail.com
Aravind S R	B71	200269@tkmce.ac.in

CONTENTS

REVISION HISTORY

Version	Name	Reason For Changes	Date
2.0		Architecture of model updated.	25-04-2023

INTRODUCTION

SYSTEM OVERVIEW

Monocular depth estimation is the process of estimating the distance of objects in a scene from a single image. The process of monocular depth estimation involves using deep learning techniques to learn the relationship between the image features and the corresponding depth. The goal is to train a model that can predict the depth map of a scene given an input image. The depth map is a 2D representation of the scene, where each pixel represents the distance of the corresponding point in the scene from the camera.

DOCUMENT PURPOSE

The purpose of a System Design Specification (SDS) for monocular depth estimation is to provide a detailed and comprehensive plan for designing, developing, and implementing a system that can estimate the depth of a scene from a single image. The SDS outlines the hardware and software requirements, the data collection and preprocessing steps, the model architecture, the training and testing procedures, and the deployment process. The SDS serves as a roadmap for the development team and stakeholders to understand the system's design, functionality, and performance objectives.

The SDS is essential for ensuring that the system meets the desired specifications and requirements. By providing a detailed plan, the SDS helps to identify potential issues and challenges that may arise during the development and implementation process. It also helps to establish a common understanding between the development team and stakeholders on the system's goals, functionality, and performance metrics.

PRODUCT SCOPE

The model being developed is to be implemented as an embedded software. The goal of this project is to:

a. create a highly effective, dependable, and efficient system that can precisely estimate the depth of objects in real-time while working within the limitations of the embedded system.
b. build a model should be capable of handling a variety of input situations and data types.
c. ensure that the software is optimised for performance, accuracy, and efficiency.

The final embedded software should be able to display a live feed from the vehicle's camera and notify the driver of an obstacle's proximity to their vehicle.

INTENDED AUDIENCE AND DOCUMENT OVERVIEW

The audience for this document encompasses both the group of professors as well as the team of developers and writers responsible for creating the product. The document itself provides a thorough overview of the functional and non-functional requirements of the product, as well as its functions, scope, and implementation. It outlines all the characteristics and capabilities that must be integrated into the final product. The entire document is organized into four sections:

1. Introduction
2. Overall Description
3. Requirements
4. Appendix

SYTEM ARCHITECTURE DESIGN

The system can be divided into two main components: the YOLOv5 model and the MiDaS model. Both of these are neural networks that perform one of the two required operations, i.e., objection detection and depth estimation respectively.

1. YOLOv5s

YOLOv5s is a single stage object detector, i.e., it requires only a single pass through the neural network and predicts all the bounding boxes in one go. It has 191 layers, 7,468,160 parameters and 7,468,160 gradients.

Like every other single stage detector, it has three important parts:

1. Model Backbone
2. Model Neck
3. Model Head

Model Backbone

The purpose of the backbone to extract important features from the given image. In YOLOv5s, the Cross Stage Partial Networks (CSPNet) are used as the backbone to extract information-rich features from the input image. CSPNet partitions the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy. The use of this split and merge technique allows for more gradient flow through the network. CSPNet has shown significant improvement in processing time with deeper networks.

Model Neck

The main purpose of the neck is to generate feature pyramids. Feature pyramids help model to generalise properly during object scaling. Simply put, it helps to identify the same object if it occurs in different sizes and/or scales. Feature pyramids are extremely useful and help models perform well on unseen data.

In YOLOv5s, Path Aggregation Network (PANet) is used as the model's neck to generate feature pyramids. Here, the feature hierarchy is enhanced with accurate localisation signals in lower layers by bottom-up path augmentation, which shortens the information path between lower layers and topmost feature. Additionally, adaptive feature pooling is employed, which links feature grid and all feature levels to make useful information in each feature level propagate directly to following proposal subnetworks.

Model Head

The purpose of the head is to perform object detection. It applies anchor boxes on features and generates the final output vectors with class probabilities, objectness scores, and bounding boxes. Objectness is a measure of the probability that an object exists in the proposed region of interest.

In YOLOv5s, the model head used is same as in the previous versions, i.e., YOLOv4 and even YOLOv3.

Activation Function

The activation function used in the middle and hidden layers in YOLOv5s is the Leaky ReLU activation function, while the final detection layer uses the sigmoid activation function.

Optimisation Function

There are two options for optimisation function in YOLOv5s:

1. SGD (Stochastic Gradient Descent)
2. Adam (Adaptive Moment Estimation)

While the default optimiser for training is SGD, one can change it to Adam using the --*adam* command line argument.

Loss Function

The YOLO family of neural networks use Binary Cross-Entropy with Logits Loss function from PyTorch for loss calculation. This is a compound loss calculated based on objectness score, class probability score, and bounding box regression score.

We can also calculate loss using the Focal Loss function, using the *fl_gamma* hyperparameter to train the model.

2. MiDaS

MiDaS is a machine learning model that estimates depth from an arbitrary input image. Various datasets containing depth information are not compatible in terms of scale and bias. This is due to the diversity of measuring tools such as stereo cameras, laser scanners, and light sensors.

MiDaS introduces a new loss function that absorbs these diversities, thereby eliminating compatibility issues and allowing multiple data sets to be used for training simultaneously. Hence, MiDaS is trained on 12 distinct datasets using this method of multi-objective optimization.

The architecture of MiDaS is based on the Residual Neural Network (ResNet). ResNet is a Convolutional Neural Network (CNN) architecture designed to support hundreds or thousands of convolutional layers. In order to solve the problem of the vanishing/exploding gradient, this architecture introduced two main concepts, namely:

1. Skip Connections
2. Residual Blocks

Skip Connections

ResNet stacks multiple identity mappings (convolutional layers that do nothing at first), skips those layers, and then reuses the activations of the previous layer. Skipping speeds up the initial training by compressing the network into fewer layers. Most ResNet models skip two or three layers at a time with nonlinearity and batch normalization in between.

Residual Blocks

When the network is retrained, all the layers are expanded and the remaining parts of the network – known as the residual blocks – are allowed to explore more of the feature space of the input image. Residual blocks are an important part of the ResNet architecture, as ResNets are formed by stacking these residual blocks together.

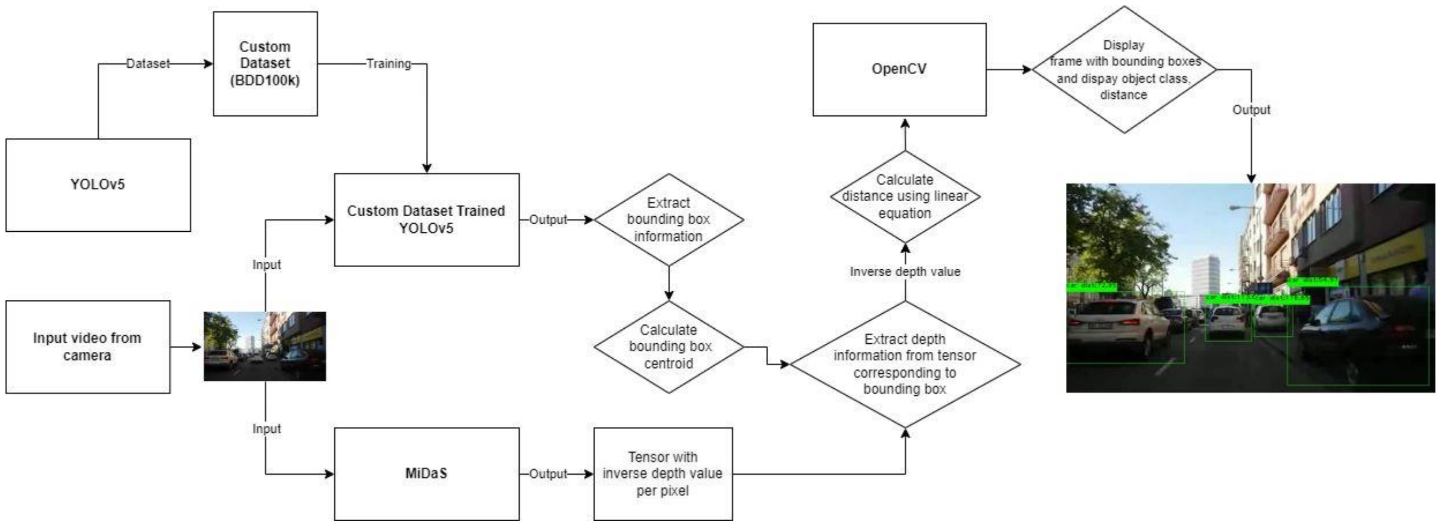
These blocks feed into the next layer, as well as into layers about 2-3 hops away. This ensures that any layers that affect the model's performance will be skipped by regularisation.

Loss Function

The loss function introduced in MiDaS is Scale- and Shift-invariant Losses. This can be defined as:

$$L_{ssi}(\hat{d}, \hat{d}*) = \frac{1}{2M}\sum_{i=1}^{M}\rho(\hat{d_i} - \hat{d_i}*)$$

where *d* and *d** are scaled and shifted versions of the predictions and ground truth, and *ρ* defines the specific type of loss function.


Dataset
Custom Dataset (BDD100k)
Training
YOLOv5
Custom Dataset Trained YOLOv5
Output
Extract bounding box information
Input
Input video from camera
Calculate bounding box centroid
Input
MiDaS
Output
Tensor with inverse depth value per pixel
Extract depth information from tensor corresponding to bounding box
Inverse depth value
Calculate distance using linear equation
OpenCV
Display frame with bounding boxes and dispay object class, distance
Output

Extracting and combining outputs:

YOLOv5 returns an output for each frame in the form of a Python dictionary. The dictionary contains a list of predictions, each of which contains a dictionary of the various details of each prediction, such as bounding box start coordinates, width, height, class of object, confidence etc. We extract the required values from this dictionary for each prediction and calculate the bounding coordinates of each object detected, as well as the centroid of the bounding box.

MiDaS returns a tensor of dimension 2 as its output, where each entry is the inverse distance corresponding to a pixel in the input frame. Using the coordinates of centroid of bounding box obtained from YOLOv5, we extract the inverse distance value of the required pixel from the tensor. This value can then be inverted and passed through a linear equation to obtain the distance value of the object from camera.

TECHNOLOGY STACK:

PYTORCH

PyTorch is a popular deep learning framework that provides a lot of functionalities for building, training, and deploying neural network models. In the case of integrating YoloV5 and MIDAS, it is used for loading and running the models, as both YoloV5 and MIDAS are implemented using PyTorch. It is used in this model to load, run, and accelerate the computation of YoloV5 and MIDAS models, making it easier to integrate and deploy these models in real-world applications.

GOOGLE COLAB

Google Colab can be used to leverage the GPU resources to accelerate the training and inference of these models. Training deep learning models can be computationally intensive and require high-end hardware, such as GPUs, which can be expensive and difficult to obtain. With Google Colab, we were able to train and run models on GPUs for free, which greatly reduce the cost and complexity of building the models.

PYTHON

Python is used as the main programming language to implement the model. Python provides many powerful libraries for scientific computing, such as NumPy, Pandas, and Matplotlib, which are essential for handling and processing large datasets, as well as visualizing the data. One of the main advantages of using Python in this project is its large and active community. Python has a massive user base that is constantly developing and improving new libraries, frameworks, and tools for data science and machine learning.

NUMPY

NumPy is a Python library for scientific computing that provides support for large, multi-dimensional arrays and matrices, as well as a large collection of mathematical functions to operate on these arrays.

1. 1. Preprocessing the input data: NumPy can be used to convert input images and videos into multi-dimensional arrays that can be fed into the YoloV5 and MIDAS models. NumPy is also used to resize, crop, and normalize the input data.
1. 2. Handling the output data: The output of the YoloV5 and MIDAS models is often in the form of multi-dimensional arrays that contain information about the detected objects or depth maps. NumPy is used to manipulate and visualize these arrays, for example, to extract the coordinates of the detected objects or to generate heatmaps from the depth maps.
1. 3. Computing mathematical operations: NumPy provides a large collection of mathematical functions to operate on multi-dimensional arrays, such as element-wise addition, subtraction, multiplication, and division, as well as matrix operations, such as dot product and transpose. These functions are used in various parts of the YoloV5 and MIDAS models, such as the loss function and the forward pass.

PANDAS

Pandas provides functions for visualizing data in various forms, such as line plots, scatter plots, and histograms. These visualizations can be used to explore and understand the datasets, as well as to evaluate the performance of the YoloV5 and MIDAS models.