

**Free space detection and depth estimation using fish eye camera**

## Research Project

Faculty of Electrical Engineering and Information Technology Professorship of Digital Signal Processing and Circuit Technology

Submitted by: Adarsh Mallandur Student ID: 457084

Date: 25.03.2019

Supervising tutor: Dr.-Ing. Christian Wiede

Identifying free space or safely drivable region is a fundamental task in autonomous driving. There are numerous methods to address this problem using both computer vision and machine learning techniques. One of the most successful methods is se- mantic segmentation using supervised learning. But, most of these methods are based on narrow-angle pin-hole cameras. Autonomous vehicles need a wider field of view to understand the scene better, especially in urban surroundings. However, the main drawback of this algorithm is the expensive step of collecting manually an- notated pixel level labels. There is no open-source pixel-annotated fish-eye images of urban traffic scene available on the internet. This project aims at solving this problem by creating virtual urban traffic simulation using Unity and collecting both images and ground truth of the scene by moving around in the virtual car. These images are then trained on the segmentation network E-Net through the method of transfer learning to produce pixel level class predictions for fish-eye images. The project also generates RGBD depth images to train a machine learning model. The results capture the distortions in the fish-eye images very well and can be used as a possible solution for real time free space detection using fish-eye camera in au- tonomous driving.

#### Keywords: Fish-eye camera, semantic segmentation, transfer learning, free space detection, deep learning, urban traffic simulation

[Contents](#_bookmark0) 3

[List of Figures](#_bookmark1) 5

[List of Tables](#_bookmark2) 6

[List of Abbreviations](#_bookmark3) 7

1. [Introduction](#_bookmark4) 8
   1. [Problem statement](#_bookmark5) 8
   2. [Structure of the project](#_bookmark6) 9
2. [Related work](#_bookmark7) 10
3. [Background](#_bookmark10) 12
   1. [Machine learning](#_bookmark11) 12
      1. [Types](#_bookmark12) 12
      2. [Features](#_bookmark13) 13
   2. [The perceptron](#_bookmark14) 13
   3. [Artificial Neural Networks](#_bookmark17) 14
   4. [Training the neural network](#_bookmark19) 14
      1. [Backpropagation](#_bookmark20) 14
      2. [Gradient Descent](#_bookmark21) 15
   5. [Types of gradient descents](#_bookmark25) 16
      1. [Batch gradient descent](#_bookmark26) 16
      2. [Stochastic gradient descent](#_bookmark27) 16
      3. [Mini-batch gradient descent](#_bookmark28) 17
4. [Deep learning](#_bookmark29) 18
   1. [Articial intelligence and Deep Neural Networks](#_bookmark30) 18
   2. [Convolutional Neural Networks](#_bookmark31) 18
      1. [Convolution layers](#_bookmark33) 19
      2. [Pooling layer](#_bookmark35) 20
   3. [Semantic Segmentation](#_bookmark36) 20
   4. [Transfer learning](#_bookmark39) 22
      1. [Definition](#_bookmark40) 23
      2. [Transfer learning methods](#_bookmark41) 23

*CONTENTS*

1. [Method](#_bookmark42) 24
   1. [Dataset generation](#_bookmark44) 24
      1. [Unity Airsim setup with Windridge environment](#_bookmark46) 25
      2. [Fisheye image projection in Unity](#_bookmark48) 27
      3. [Implementation of fisheye projection with Airsim](#_bookmark50) 28
   2. [Preprocessing the data](#_bookmark52) 28
   3. [Model](#_bookmark54) 30
      1. [Network architecture](#_bookmark55) 30
      2. [Transfer learning using tensorflow](#_bookmark60) 32
      3. [Loss function and training](#_bookmark61) 33
2. [Results](#_bookmark65) 35
   1. [Training and validation performance of the model](#_bookmark66) 35
   2. [Model evaluation on simulated test data](#_bookmark68) 36
   3. [Model evaluation on real dataset](#_bookmark74) 37
3. [Concluding Remarks](#_bookmark78) 39

[Bibliography](#_bookmark79) 40

* 1. [The perceptron](#_bookmark15) 13
  2. [Some common activation functions](#_bookmark16) 14
  3. [Artificial Neural Network](#_bookmark18) 14
  4. [Finding a minimum using gradient descent](#_bookmark23) 15
  5. [Convolution operation in CNN](#_bookmark32) [[36]](#_bookmark115) 19
  6. [U-net: Encoder-decoder architecture](#_bookmark37) [[31]](#_bookmark110) 21
  7. [Dilated/atrous convolutions](#_bookmark38) [[41]](#_bookmark120) 22
  8. [Fisheye image segmentation](#_bookmark43) 24
  9. [Windridge city in Unity](#_bookmark45) 25
  10. [Windridge Scene with Airsim assets](#_bookmark47) 26
  11. [Flow diagram of the 5 pass rendering algorithm. 4 passes create the fisheye projection and an additional pass creates the geometry](#_bookmark49)

[corrected fisheye](#_bookmark49) [[5]](#_bookmark84) 27

* 1. [Fish-projection in Unity using 4 cameras and 3 render passes](#_bookmark51) 28
  2. [Visual representation of ENet modules](#_bookmark58) [[29]](#_bookmark108) 30
  3. [Training and validation losses as compared to the epochs](#_bookmark67) 35
  4. [Simulated images from unity](#_bookmark70) 36
  5. [Ground truth labels](#_bookmark71) 36
  6. [Predicted labels from our model](#_bookmark72) 37
  7. [Predicted labels from Enet](#_bookmark73) 37
  8. [LMS Fisheye groundtruth images](#_bookmark75) 38
  9. [Results of using the E-net model trained on cityscapes](#_bookmark76) 38
  10. [Results of using our model trained on simulated fisheye images](#_bookmark77) 38
  11. [Related research work on free space detection using fisheye camera](#_bookmark8) . . 10
  12. [Related research work on free space detection using fisheye camera (continued.)](#_bookmark9) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

|  |  |
| --- | --- |
| . . | 11 |
| . . | 29 |
| . . | 31 |
| . . | 34 |
| [t](#_bookmark69) . 36 | |

[5.1 Class number to pixel intensity mapping](#_bookmark53) . . . . . . . . . . . . . . [5.2 Network architecture](#_bookmark59) . . . . . . . . . . . . . . . . . . . . . . . . .

[5.3 Hyperparameters used for training](#_bookmark64) . . . . . . . . . . . . . . . . . .

[6.1 Pixel wise results comparison of E-net vs our model on test datase](#_bookmark69)

**AI** Artificial Intelligence **ANNs** Artificial Neural Networks **DNNs** Deep Neural Networks

**CNNs** Convolutional Neural Networks **GPUs** graphics processing units **RGBD** Red Blue Green Depth

**FCN** Fully Convolutional Networks

**ML** Machine Learning

**SGD** Stochastic Gradient Descent

**API** Application Programming Interface

**FOV** Field of view

In the past few years, there has been a tremendous amount of growth in research of Artificial Intelligence (AI) related fields such as computer vision, machine learning and autonomous vehicles. Despite these advances, fully autonomous car navigation in complex environments is decades away. The reasons for this can be thought of as non-availability of AI which generalises to new problems and because most computer vision algorithms produce errors at a rate which is not acceptable for fully autonomous navigation [[20].](#_bookmark99) One solution to make the algorithms less prone to errors is by enabling autonomous vehicles to perceive their environment better. In urban environments, the traffic situations can be very complex and unpredictable. To make autonomous vehicles perceive their environment better, we need to have wide-angle views. This project focusses on the task of free space detection using fisheye cameras, thus providing the autonomous vehicles with a wide-angle view.

Fisheye camera is widely used in autonomous vehicles such as for parking [[40],](#_bookmark119) vehicle surround monitoring [[26],](#_bookmark105) scene-understanding [[15],](#_bookmark94) etc. Fisheye images have strong distortions because of projecting hemisphere onto the image plane. The conventional image processing and deep learning algorithms do not take into account of these distortions and hence cannot be directly used with fisheye images. There is a lack of labelled fisheye dataset available for development of algorithms to process fisheye images. This project tries to find a solution for this problem by creating simulated fisheye images and labels.

This project presents a Convolutional Neural Network (CNN)-based semantic segmentation solution to detect free spaces in a complex urban traffic scene using a fisheye camera. To address the problem of lack of labelled fisheye dataset, the project aims at providing a solution using the Unity software framework [[37].](#_bookmark116) We demonstrate that a CNN-based semantic segmentation network can be trained on simulated fisheye images and then be used for inference on real-world fisheye images.

## Problem statement

This project aims to solve the following problems

* + 1. State-of-the-art semantic segmentation algorithms do not work on fisheye im- ages.
    2. Lack of labelled fisheye image dataset of urban traffic scenes.
    3. It is very expensive both in terms of time as well as money to generate labels man[ually[4].](#_bookmark83)

*1 Introduction*

The first problem of semantic segmentation algorithms is solved by using transfer learning, where the existing pre-trained algorithm is further trained using data of the target task. The second problem of lack of fisheye image dataset is solved by creating a simulation of urban traffic scenes using Unit[y[37].](#_bookmark116) The third problem of generating labels is solved automatically since the simulations inherently create labels to each object in the virtual environment.

## Structure of the project

The main focus of this project work can be summarised as

* + 1. To study the literature of free space detection using fisheye cameras and pro- pose a better solution.
    2. To generate labelled fisheye dataset using the Unity framework [[37].](#_bookmark116) The dataset has both semantic layers as well as depth labels.
    3. To use a pre-trained CNN to evaluate the fisheye dataset and then retrain the CNN to make better predictions.

# Related work

There are numerous research works on the problem of free space detection. The important ones are summarised in table [2.1](#_bookmark8) and table [2.2](#_bookmark9) according to chronological order. In the early methods such as [[3],](#_bookmark82) superpixels called stixels were extracted from stereo images which represented mainly vertical shapes as non-free space. Early us- age of CNNs for this task was in [[1]](#_bookmark80) where the authors used sliding window approach to train a CNN using RGB monocular images. Fisheye cameras were used for ob- stacle detection in [[16],](#_bookmark95) where the method uses wheel odometry to generate weak labels before training. Hence this can be considered as weakly supervised training. Due to the unavailability of a labelled dataset, most of the algorithms used a semi-supervised approac[h[6]](#_bookmark85) [[33][17][39].](#_bookmark118) However, this method of generating weak

labels is often expensive and is often noisy.

In [[11]](#_bookmark90) and [[32]](#_bookmark111) the authors have converted real world fisheye dataset into synthetic dataset by applying a fisheye transformation to each image. They have used labelled datasets such as cityscapes [[8]](#_bookmark87) to generate the synthetic fisheye dataset. These methods are not so efficient since synthetic images do not have the extra field of view as compared to real fisheye images.

Our method generates simulated fisheye images by stitching four images from four cameras and hence providing robust images and labels, which can be used to train neural networks. In this project, we demonstrate this by training the dataset on [E-Net[29],](#_bookmark108) a segmentation CNN trained on cityscapes [dataset[8].](#_bookmark87)

Table 2.1: Related research work on free space detection using fisheye camera

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Literature** | **Year** | **Algorithm** | **Input** | **Output** |
| The Stixel World - A Com-  pact Medium Level Repre- sentation of the 3D-World [[3]](#_bookmark82) | 2009 | Dense stereo algo-  rithm and stixel extraction | RGB  Stereo images | Stixels |
| Road scene segmentation  from a single image [[1]](#_bookmark80) | 2012 | CNN trained using  noisy labels. Slid- ing window approach is used | RGB  monocular camera images | Patch  labels |
| Obstacle detection for  self-driving cars using only monocular cameras and wheel odometry [[16]](#_bookmark95) | 2015 | Depth map compu-  tation using plane sweeping and obstacle extraction | RGB Fish-  eye images and wheel odometry | Occupancy  grids |

*2 Related work*

Table 2.2: Related research work on free space detection using fisheye camera (con- tinued.)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Literature** | **Year** | **Algorithm** | **Input** | **Output** |
| Convolutional patch net-  works with spatial prior for road detection and urban scene understanding [[6]](#_bookmark85) | 2015 | Convolutional patch  network using normal- ized position of patch as additional input | RGB  Monocular camera images | Pixel  labels |
| Free-space detection with  self-supervised and online trained fully convolutional networks [[33]](#_bookmark112) | 2017 | FCN and self-  supervised learning using weak labels from Stixel algorithm | RGB  Stereo camera images | Pixel  labels (Segmen- tation mask) |
| Free-space detection with  fish-eye cameras [[17]](#_bookmark96) | 2017 | Segmentation and  Random forest classi- fication of superpixels | RGB  monocular fisheye camera images | Labelled  superpix- els |
| CNN based semantic seg-  mentation for urban traffic scenes using fisheye camera [[12]](#_bookmark91) | 2017 | Semantic segmenta-  tion using OPP-Net | RGB  monocular fisheye images | Pixel  labels (Segmen- tation mask) |
| Minimizing Supervision for  Free-space Segmentation [[39]](#_bookmark118) | 2018 | FCN (SegNet) and  weakly supervised annotation extraction | RGB  monocular camera image | Pixel  labels (Segmen- tation mask) |
| Cnn-based fisheye image  real-time semantic segmen- tation [[32]](#_bookmark111) | 2018 | Efficient Residual  Factorized CNN (ERFNet) modified with fish eye camera images | RGB  monocular fisheye camera images | Pixel  labels (Segmen- tation mask) |
| Restricted deformable con-  volution based road scene semantic segmentation us- ing surround view cameras [[11]](#_bookmark90) | 2018 | Restricted De-  formable Convolution (RDC) and zoom aug- mentation to generate fisheye images | RGB  monocular fish eye camera images | Pixel  labels (Segmen- tation mask) |

# Background

In this chapter, we discuss the theoretical concepts of neural networks, computer vi- sion and deep learning required for understanding the implementation of the project. First, the details of artificial intelligence, machine learning and deep learning are dis- cussed. Finally, the application of deep learning to computer vision is examined in detail.

## Machine learning

Machine learning is used to solve a wide variety of problems in computer vision, speech processing, statistics etc,. It is a subset of artificial intelligence where the computer is shown data to learn patterns and solve the given problem without being explicitly programmed to do so [[2].](#_bookmark81) Computers have been successful in several tasks that were considered impossible by humans. However, they fail to do simple tasks such as recognising objects or speech. Hence, machine learning algorithms were designed to infer the result from features of the input. In most of the cases, these features needed to be hand-crafted by an expert. The algorithms work better when given better features. As the availability of data and computational power has increased in recent years, machine learning has become more and more practical and successful in various application domains.

### Types

Machine learning algorithms can be broadly classified into supervised learning, unsu- pervised learning and reinforcement learning. In *supervised learning*, the algorithm is trained using labelled training data and then shown with test data to predict the results. For example, in object detection problems a training data set with annotated images is shown to the model through which the model learns the rep- resentations and the mapping from input to output. After training the model is able to predict the labels of unseen images. Supervised learning can be further classified into classification and regression. In *classification*, the algorithm tries to predict the discrete output class the input belongs to, whereas in *regression* the output is a continuous value. *Unsupervised learning* is the process of building the model without the help of labelled data. It can also be considered as a problem of clustering a given dataset. The machine learning algorithm needs to separate the set of data points into groups in the best way possible. *Reinforcement learning* uses a system of rewards and punishments. The algorithm learns by interacting with its

environment. The algorithm receives rewards for performing correctly and penalties for performing incorrectly. The algorithm learns without the intervention of human by maximising the rewards and minimising the penalties.

### Features

Features are the attributes in the data which are most relevant to the machine learning problem. Selection of features which enhance the performance of the model is a crucial step in machine learning. Feature selection methods are used to remove redundant and irrelevant attributes from data that either do not contribute to the accuracy of the model or may even decrease the accuracy of the model.

## The perceptron

An artificial neuron or perceptron is modelled similarly to the neurons in the human brain. It takes several inputs and performs a weighted summation to produce an output. The weight of the perceptron is determined during the training stage. The training is performed using a method called *gradient descent* which will be explained in later sections.

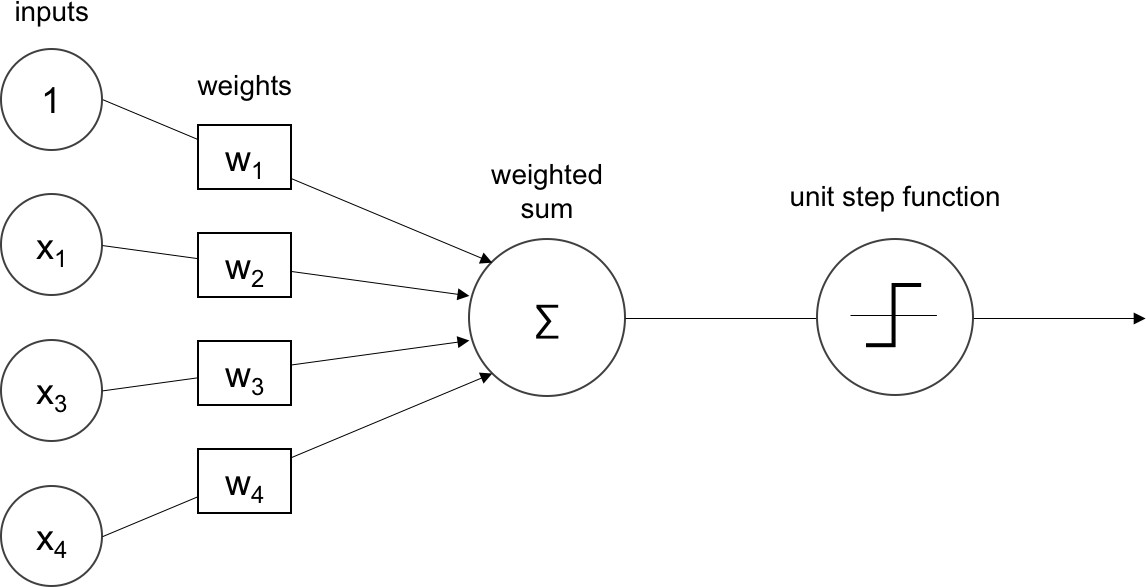
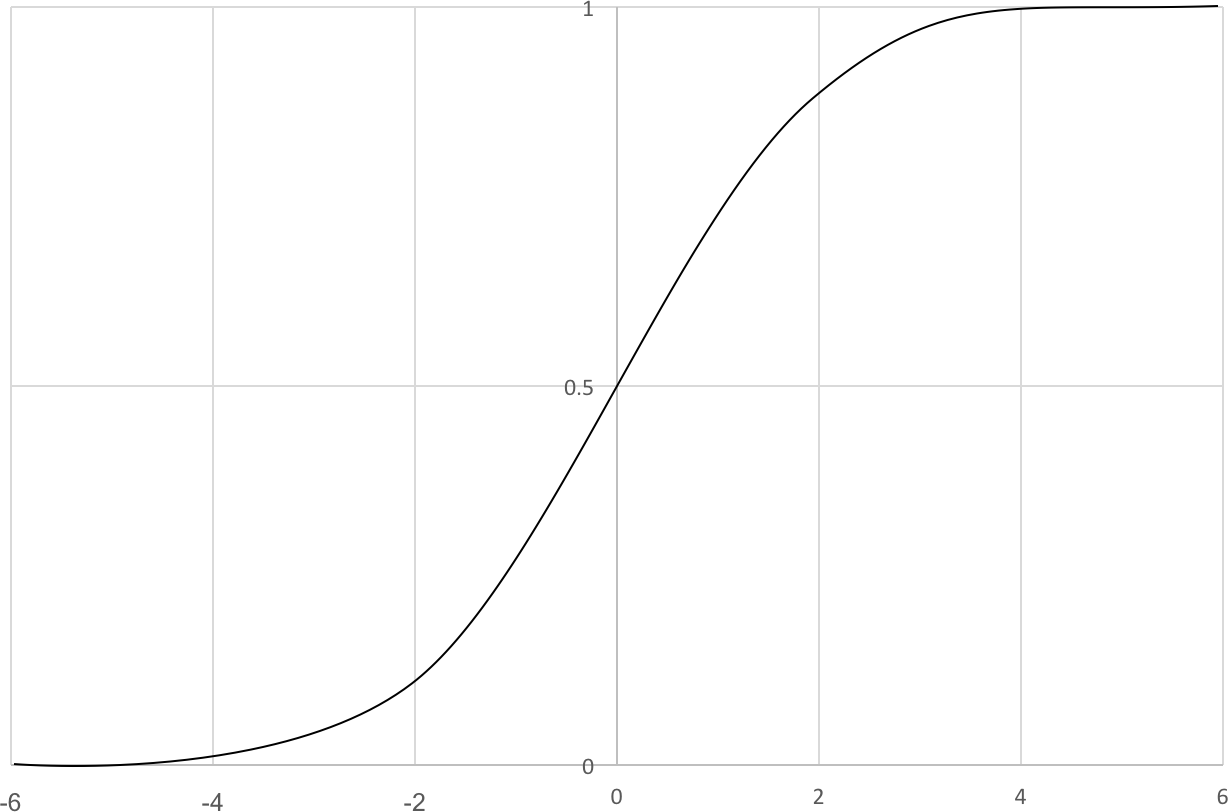
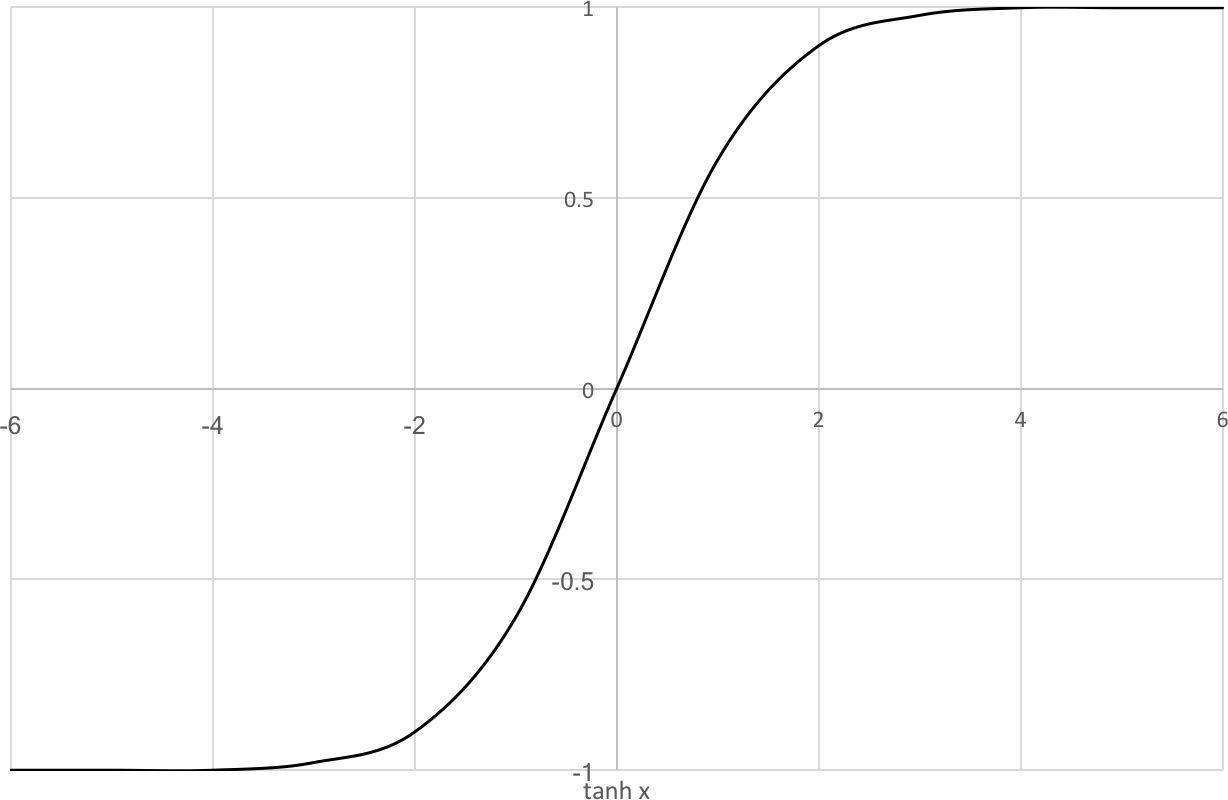
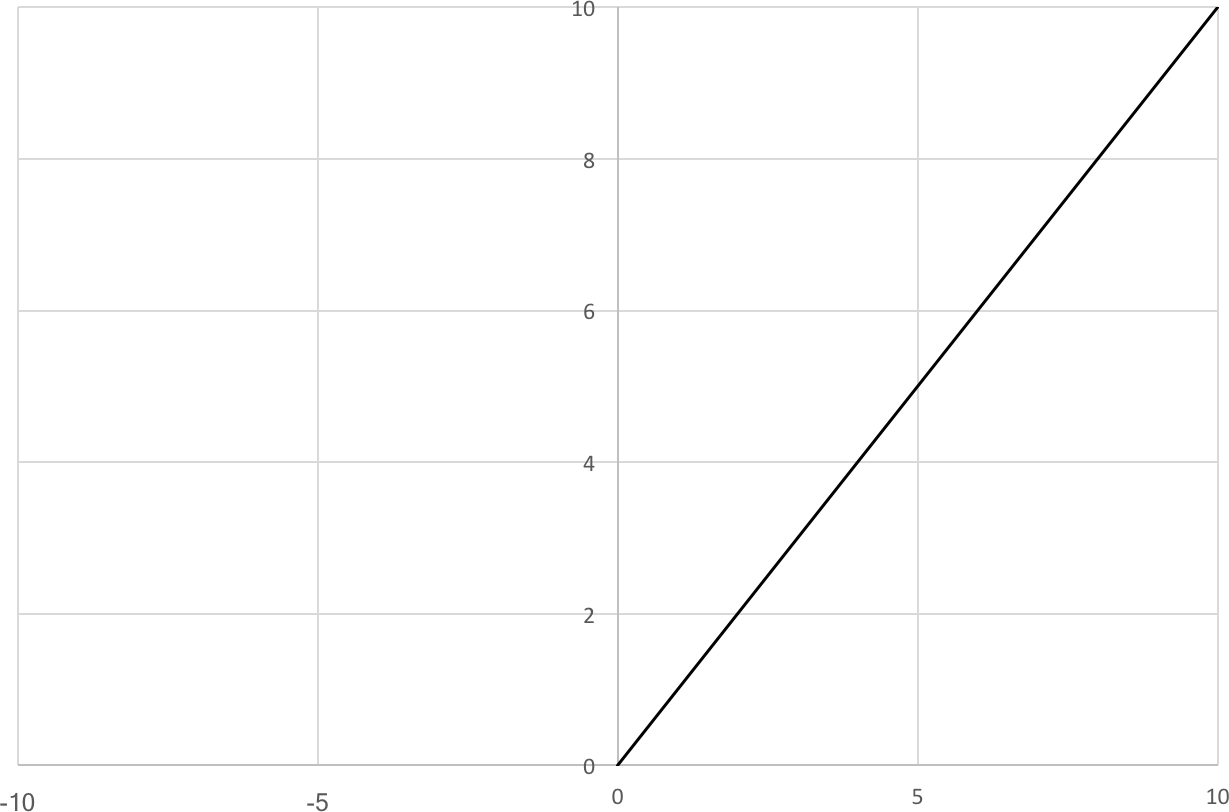


Figure 3.1: The perceptron

The weighted sum is passed through a non-linear function called *activation func- tion* to capture the non-linearities in the input. An *activation function* decides whether a neuron should fire or not. There are various activation functions as shown in [3.2.](#_bookmark16) Sigmoid is useful for converting any value to probabilities and can be used for binary classification. The tanh maps input to a value in the range of -1 to 1 and are more stable than sigmoid. The ReLU maps input x to max (0, x) and works well for a large range of problems.

* + 1. Sigmoid function (b) Tanh function (c) RELU function

Figure 3.2: Some common activation functions

## Artificial Neural Networks

The artificial neural network is a collection of perceptrons and activation functions. The perceptrons are connected to each other to form the hidden layers as shown in

[3.3.](#_bookmark18) The training process determines the values of these weights and biases. The training is started by initialising the weights and biases. The error is computed by taking the difference of the actual output to the ground truth. Based on the loss, the weights and biases are tuned in steps. The training is stopped when the error cannot be minimised anymore and during this point, the model is said to learn the features in the input.

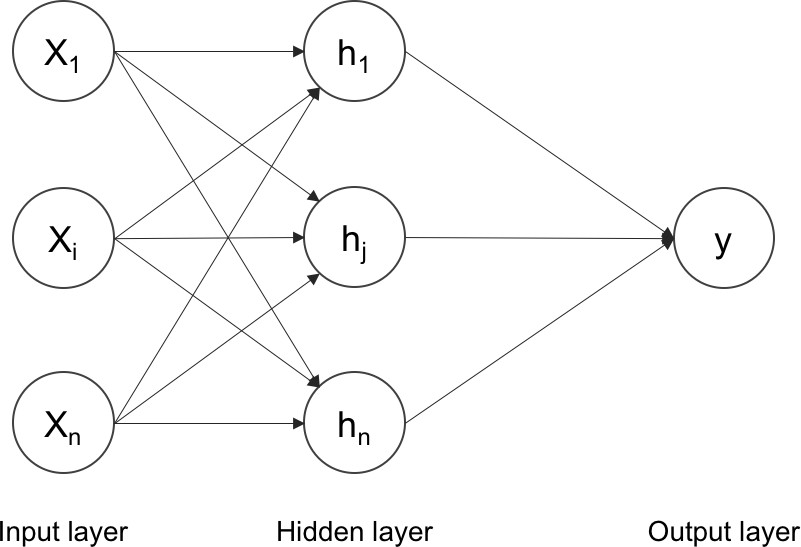


Figure 3.3: Artificial Neural Network

## Training the neural network

### Backpropagation

One of the commonly used algorithms for training the neural networks is the back- propagation algorithm. The weights are updated from the last layer based on the

error that is calculated between ground truth and actual output. The error is prop- agated through the layers until the first layer and at each step the gradients are calculated. The weights are updated using the gradient descent algorithm which is explained in the following section.

### Gradient Descent

Often in machine learning, finding the best model for a certain situation means to minimise the error of the model or maximise the likelihood of the data. Hence it can be seen as finding the solution for an optimisation problem. Gradient descent is one of the optimisation problems which is widely used in the algorithms of machine learning and deep learning.

#### Idea

Suppose we have some function **f(x)** that takes as input a vector of real numbers and outputs a single real number. One such simple function is given in equation [3.1.](#_bookmark22)

*f* (*x*) = *x*2 ∈ *R* (3.1)

The *gradient* of **f(x)** gives the input direction in which the function **f(x)** most quickly increases. One approach to maximising a function is to pick a random start- ing point, compute the gradient, take a small step in the direction of the gradient, and repeat with the new starting point. Similarly, we can try to minimise a function by taking small steps in the opposite direction, as shown in [3.4.](#_bookmark23)

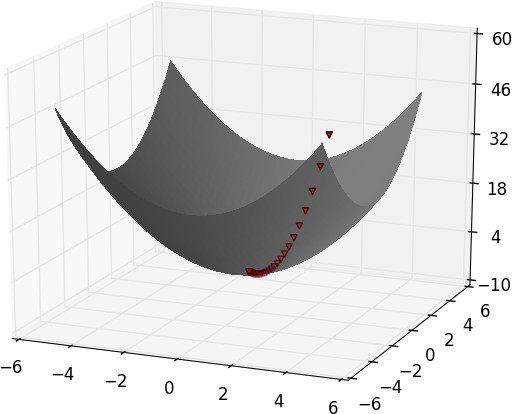


Figure 3.4: Finding a minimum using gradient descent

This procedure will work very well if the function has a unique global minimum. If a function has multiple (local) minima, this procedure might find the wrong one

of them, in which case we might re-run the procedure from a variety of starting points. If a function has no minimum, then it is possible the procedure might go on forever.

#### Estimating the gradient

If f is a function of one variable, its derivative at a point x measures how f(x) changes when we make a very small change to x. The derivative is the slope of the tangent line at (x, f(x)). The gradient can be estimated as given in [3.2](#_bookmark24) by considering h to be very small.

*Gradient* = *f* (*x* + *h*) − *f* (*x*)

*h*

(3.2)

If f is a function of multiple variables then it has multiple partial derivatives each indicating the change in f when we make small changes to one of the input variables by holding the other variables constant. The learning rate *η* determines the size of the steps we take to reach a (local) minimum. There are three variants of gradient descent as explained in the following sections.

## Types of gradient descents

### Batch gradient descent

Batch gradient descent or vanilla gradient descent calculates the gradient of the cost function w.r.t the parameters *θ* for the entire training dataset.

*θ* = *θ* − *η* · ∆*θJ*(*θ*) (3.3)

Batch gradient descent is slow since we need to calculate the gradients for the whole dataset to make just one update of the parameters. Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

### Stochastic gradient descent

Stochastic gradient descent (SGD) performs parameter update for each training sample. Batch gradient descent performs redundant computations for large gradients since it recomputes the gradients for similar examples before each update. SGD does not have this problem since it performs one update for each training data and is much faster. These frequent updates cause the objective function to fluctuate which may lead to overshooting. However, by setting the optimal learning rate, SGD can show the same behaviour as the batch gradient descent.

*θ* = *θ* − *η* · ∆*θJ*(*θ*; *x*(*i*); *y*(*i*)) (3.4)

### Mini-batch gradient descent

Mini-batch gradient descent takes the best of SGD and batch gradient descent. It updates the parameters for every mini-batch of *n* training samples. It reduces the variance of the parameter updates and hence more stable convergence. It also makes computations more efficient.

*θ* = *θ* − *η* · ∆*θJ*(*θ*; *x*(*i*:*i*+*n*); *y*(*i*:*i*+*n*)) (3.5)

#### Challenges of mini-batch gradient descent

Finding the optimum learning rate is difficult. A learning rate which is too small will take lots of time to converge and a learning rate too fast might overshoot and oscillate around the minimum.

•

Neural networks usually have highly non-convex error functions with saddle points. These are the points where one dimension slopes up and other slopes down. The SGD finds these points very hard to escape since the gradient is close to zero in all dimensions.

•

# Deep learning

Conventional machine-learning techniques were unable to process natural data in their raw form. For decades, pattern-recognition systems needed carefully engi- neered features extracted from raw data to be fed into the learning systems such as classifiers. *Representation learning* is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification [[24].](#_bookmark103) *Deep learning* methods are representation-learning methods with multiple levels of representation, obtained by composing simple non- linear modules that each transform the representation at one level into a represen- tation at a higher and more abstract level [[24].](#_bookmark103) An example of deep learning can be multiple layer feed forward neural network with non-linear activations. With enough transformations, any complex function can be learned. This is known as the *universal approximation theorem* [[9].](#_bookmark88)

## Articial intelligence and Deep Neural Networks

Deep neural networks (DNNs) form a part of broad field of Artifical intelligence (AI). DNNs are currently the foundation of many AI [applications[24].](#_bookmark103) The number of applications using deep neural networks(DNNs) have increased tremendously since they were applied successfully to speech recognition [[10],](#_bookmark89) image recognition [[23],](#_bookmark102) etc. However, there is a trade-off between the accuracy of DNNs and the cost of high computational compexity. Hence, graphics processing units (GPUs) are widely used for training and inference of deep neural networks.

## Convolutional Neural Networks

The convolutional neural networks (CNNs) are considered one of the most promis- ing solutions for various computer vision tasks such as object detection [[30],](#_bookmark109) image recognition, etc. CNNs were first introduced through the work of LeCuN in 1989 [[25]](#_bookmark104) for processing handwritten zip codes. The success of CNNs has attracted attention from companies such as Google, Microsoft, AT&T, Facebook etc., [[21].](#_bookmark100) The archi- tecture of CNN consists of multiple learning stages composed of a combination of convolutional layer, non-linear processing units and subsampling layers. Each layer generates higher level abstractions called a *feature map*. The regular neural net- works do not scale to full images since they have fully connected layers at the input. For example, an image of 240\*480\*3 would lead to neurons that have 240\*480\*3

weights. This would lead to a huge number of parameters and hence to overfitting. In a convolutional neural network, the neurons in a layer will only be connected to a small region of the layer before it, instead of a fully connected network. A convNet is made up of layers and each layer transforms an input 3D volume to an output 3D volume with some differentiable function.

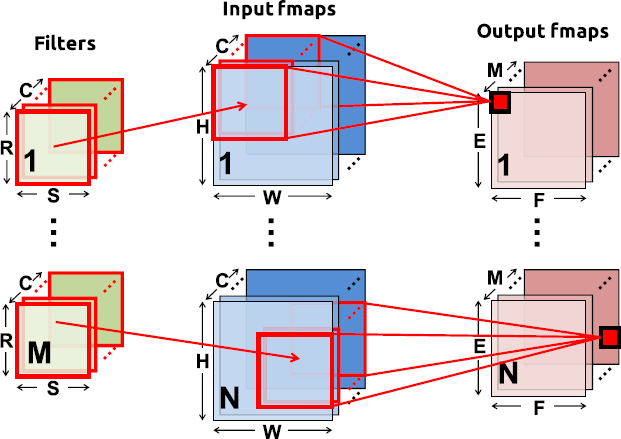


Figure 4.1: Convolution operation in CNN [[36]](#_bookmark115)

### Convolution layers

Each of the convolution layer in CNNs is composed of convolutions as shown in figure [4.1.](#_bookmark32) The weighted sum for each output activation is computed using only a small neighbourhood of input activations. This neighbourhood is known as *filter size*. The size of the ouput feature map depends on the number of filter kernels used, the padding and the stride of convolution operation. The computation of the convolution layer can be defined as below equation [4.1](#_bookmark34) [[36]](#_bookmark115)

*C−*1 *S−*1 *R−*1

Σ Σ Σ

*O*[*z*][*u*][*x*][*y*] = *B*[*u*] + *I*[*z*][*k*][*Ux* + *i*][*Uy* + *j*] ∗ *W* [*u*][*k*][*i*][*j*] (4.1)

*k*=1 *i*=1

*j*=1

0 ≤ *z < N,* 0 ≤ *u < M,* 0 ≤ *x < F,* 0 ≤ *y < E*

*E* = *H* − *R* + *U , F* = *W* − *S* + *U*

*U U*

where,

O = output feature map matrix I = input feature map matrix W = filter matrix

B = bias matrix N = batch size

M = number filters

H, W, C = input feature map dimensions R, S, C = filter dimensions

E, F, M = output feature map dimensions U = stride size

### Pooling layer

Pooling is the group of computations that reduce the dimensions of the feature maps. Pooling enables the network to be invariant to small shifts and distortions. The most common pooling operations are max pooling and average pooling where usually stride size of 2 or greater is used.

## Semantic Segmentation

Semantic segmentation is understanding an image at the pixel level. In this task, each pixel is assigned a label which tells us which object the pixel belongs to. It is a high-level task that leads us towards complete scene understanding. There are many approaches to solving the problem of semantic segmentation in computer vision literature. The popularity of deep learning and convolutional neural network has influenced the problem of semantic segmentation as well [[43].](#_bookmark122) These techniques of deep learning and CNNs surpass other approaches by a large margin in terms of accuracy and [efficiency[42].](#_bookmark121) The operation is similar to convolution but instead of multiplying with the filter kernel, sampling is done by either taking the maximum or average of the values of the feature map matrix in the neighbourhood of the filter size.

#### Different approaches to semantic segmentation

Before deep learning, there were methods like TextonForest[[35]](#_bookmark114) and Random Forest based classifiers for semantic segmentation. One of the popular initial deep learning

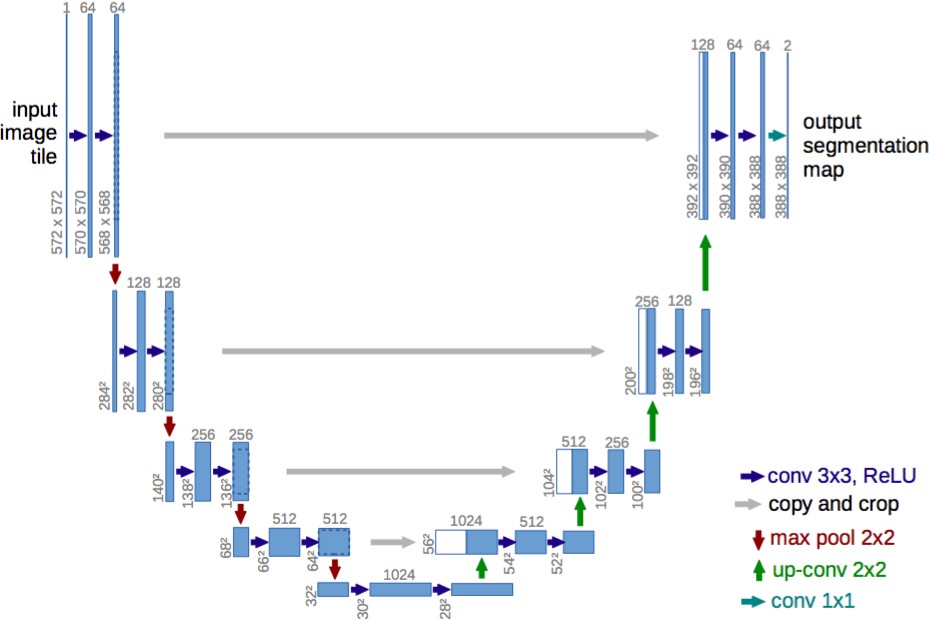


Figure 4.2: U-net: Encoder-decoder architecture [[31]](#_bookmark110)

approaches was patch classification [[7]](#_bookmark86) where a patch of the image around a pixel was used to classify each pixel into classes. In recent years, convolutional neural networks have had enormous success on segmentation problems. In 2014, a CNN method for solving segmentation called Fully Convolutional Networks [[27]](#_bookmark106) was intro- duced. It was a dense prediction method without any fully connected layers. This meant that segmentation maps of any size could be generated and was also much faster compared to the patch classification approach. All the subsequent variants of semantic segmentation are based on this approach. The other problem apart from using fully connected layers in CNNs for segmentation is pooling layers. Pooling lay- ers introduce the property of *spatial invariance* to the network whereby it discards the ’where information’. However, the semantic segmentation problem needs this ’where information’ to predict the class maps for the pixels. Two different classes of literature evolved to solve this problem.

#### Encoder - Decoder architecture

First one is the encoder-decoder architecture. The idea is that the encoder gradually reduces the spatial dimensions with pooling layers and the decoder gradually recovers the object details and spatial dimensions. There are shortcut connections from the

encoder to decoder help the decoder in recovering the object details as shown in [4.2.](#_bookmark37) U-net [[31]](#_bookmark110) belongs to this class of architecture. The key idea of fully convolutional netw[orks[27]](#_bookmark106) is that the fully connected layers in the classification networks can be viewed as convolutions with kernels that cover their entire input regions. After convolutionalizing the fully connected layers in a imagenet pretrained network like VGG, the feature maps needs to be upsampled because of the pooling operation of VGG network. This upsampling is done by *deconvolutional layers*. But, upsampling with deconvolutional layers produces coarse segmentation maps because of the loss of information during pooling. Therefore skip connections are introduced from higher resolution feature maps.

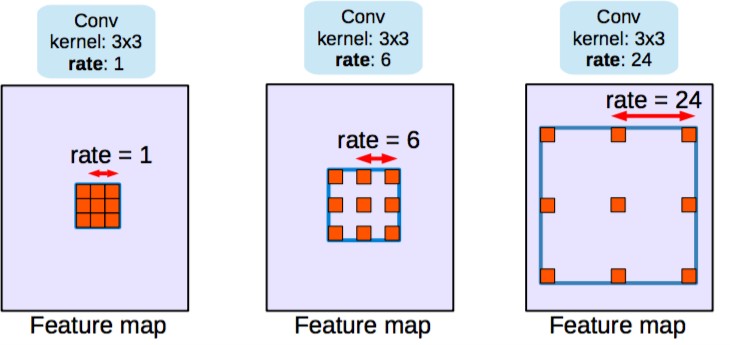


Figure 4.3: Dilated/atrous convolutions [4[1]](#_bookmark120)

#### Dilated/Atrous convolutions

The second type of architectures are called as dilated/atrous convolutions and they do not include the pooling layers. Dilated convolutional layer allows for an expo- nential increase in field of view without decrease of spatial dimensions. It can be thought of the reverse process of convolution where the spatial dimensions increase as shown in [4.3.](#_bookmark38)

## Transfer learning

Deep learning networks are *data dependent*. The scale of the model and the size of the data is linearly dependent. Many domains face the problem of insufficient training data. Often the model faces situations where it has not seen the data before and does not know how to deal with it. In all of such situations the current state-

of-the-art models either show degraded performance or even break down. Transfer learning can help us deal with such scenarios.

Transfer learning allows us to use the already existing labelled data of some related task or domain. The name transfer learning is derived from the fact that we try to transfer as much knowledge as we can from the source to our target task or domain.

### Definition

As per the definition in [[28],](#_bookmark107) transfer learning involves a domain and a task. A domain **D** consists of a feature space **X** and a marginal probability distribution **P(X)** over the feature space. Given a domain, **D = (X, P(X))**, a task **T** consists of a label space **Y** and a conditional probability distribution **P(Y**|**X)** that is learnt from the training data. Given a source domain **D***s* and a corresponding source task **T***s* as well as a target domain **D***t* and a target task **T***t*, the objective of transfer learning is to enable to learn the target probability distribution **P(Yt**|**Xt)** in **D***t* with the information gained from **D***s* and **T***s* where **D***s* ƒ= **D***t* or **T***s* ƒ= **T***t*.

Given source and target domains **D***s* and **D***t* where **D = (X, P(X))** and source and target tasks **ts** and **T***t* where **T = (Y, P(Y X))** there can be four transfer learning scenarios.

|

* + - 1. **Xs** ƒ= **Xt**. The feature space of source and target domain are different.
      2. **P(Xs)** = **P(Xt)**. Marginal probability distributions of source and target domain are different.

ƒ

* + - 1. **Ys** ƒ= **Yt**. The label spaces between the two tasks are different.
      2. **P(Ys Xs)** = **P(Yt Xt)**. The conditional probability distributions of the source and target tasks are different.

| ƒ |

### Transfer learning methods

Transfer learning has been investigated in the past [[28].](#_bookmark107) With the approach of deep learning, there are new methods of transfer learning available. The most common way of applying transfer learning for a given problem is by using pretrained features. It is also the method which has been used in this project. The lower layers of the convolutional neural networks capture the low-level image features such as edges, corners etc., while the higher convolutional layers capture more and more complex details such as body parts, faces etc. The final fully connected layers are assumed to capture information that is relevant to solving the respective task, for example, the classes in case of the classification task. Thus we can simply use the features of a state-of-the-art network such as [ENet[29]](#_bookmark108) pre-trained on Cityscapes dataset [[8]](#_bookmark87) and train a new model on these extracted features. In practice, we keep the earlier layers of the network fixed or tune them with very low learning rate so as to not unlearn these features. This simple approach has been shown to achieve great results on various computer vision tasks [[28].](#_bookmark107)

# Method

The aim of the project is to detect free space in a given image taken by a fisheye camera in an autonomous car. Free space is defined as the region in the image, where no other object is visible and the projection lies on the ground plane. The project has been divided into three sub-tasks as below

1. To create a dataset of annotated fisheye images of road scenes.
2. To use the created dataset to train a neural network.
3. To use this neural network to detect free space in test images.

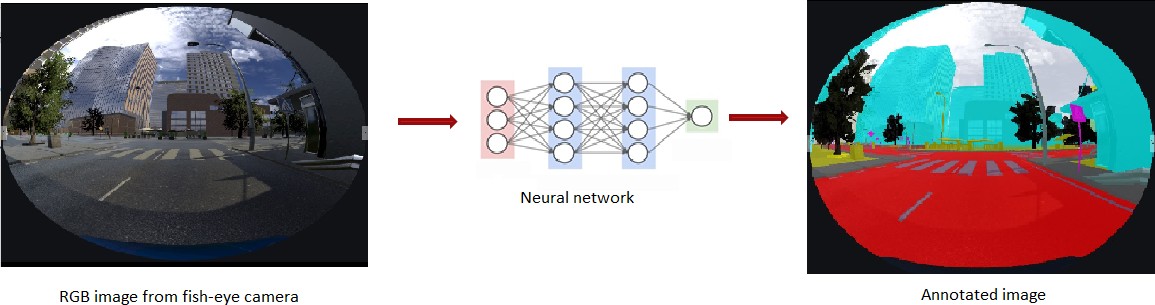


Figure 5.1: Fisheye image segmentation

## Dataset generation

There is no publicly-available annotated fisheye image dataset of road scenes. Hence in this project, simulated images and labels are generated using Unity software framework [[37].](#_bookmark116) The advantage of using simulation is that the labels are implic- itly created when an object is created inside the simulation. Unity provides free environments for development. One of the city environments which looks realistic and can be used for free is Windridge city. It has buildings, city props, road, lane markings, countryside, trees etc., (figure [5.2)](#_bookmark45) which can be used for image and label generation.



Figure 5.2: Windridge city in Unity

We now need a car which has all the physics properties such as collision body, gravity etc., to roam around the windridge city, collecting images and labels. For this purpose, there is a framework from Microsoft known as Microsoft Airsim [[34].](#_bookmark113) Airsim is a simulator for cars and drones and is built on Unreal Engine. It is open- source, cross-platform and it supports hardware-in-loop for realistic simulations. Microsoft has recently released Airsim plugin for Unity. Airsim also exposes APIs to retrieve data from the car such as steering angle, acceleration and also APIs to control the car through the code.

For the purposes of this project, the dataset generation step can be further split into the following steps.

1. To set up Unity software with Windridge environment and Airsim plugin.
2. To project the images of the simulation as fish-eye projections.
3. To store the images and label by driving around the simulation environment.

### Unity Airsim setup with Windridge environment

The installation described here is for Windows 10 x64 machine with Nvidia Quadro M2200 graphics card. For this project Unity 2018.3.2f1 version was used. For linux installations, please visit the respective websites of Unity and Airsim. Once the software is downloaded and installed, we need to install the Airsim plugin for unity.

* + - 1. Download the unity editor from the Unity w[ebsite[37].](#_bookmark116)

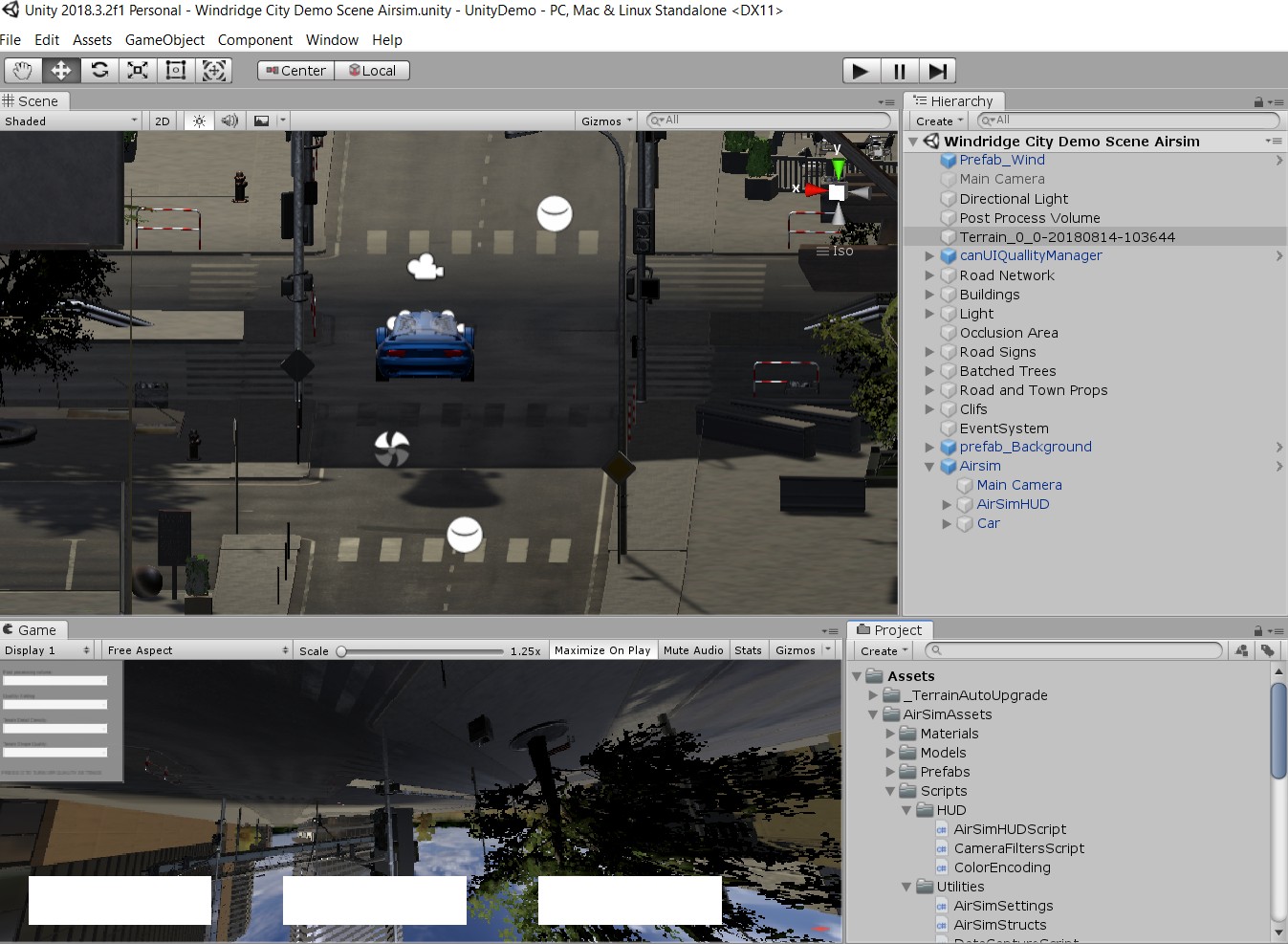


Figure 5.3: Windridge Scene with Airsim assets

* + - 1. Install Visual Studio 2017 and clone the git repo https://github.com/Microsoft/AirSim.git
      2. Run build.cmd from the command line.
      3. Now build the unity project by running build.cmd inside Unity folder of the git repo.
      4. Create a new Unity project and download Windridge city asset inside it from the unity asset store.
      5. From the UnityDemo/Assets folder of the above git repo, copy the folders AirSimAssets, Plugins and Standard Assets into the Assets folder of your new Unity project.
      6. From the UnityDemo/Assets/Scenes folder of the git repo, copy CarDemo.unity into the Assets/Scenes folder of your new Unity project.
      7. Copy ”AirsimHUD”, ”Car”, ”Main Camera” items from CarDemo scene into the Windridge City Demo Scene.
      8. Change the position of the Car and Main Camera to that of the Main Camera of the Windridge scene.
      9. Remove the old Windride camera. Make sure the new camera has Smooth flow script with Target as ”LookAt”
      10. The final scene should look as in figure [5.3](#_bookmark47)

The unity project used for this research work can be found at https://git.dst.etit.tu- chemnitz.de/admal/fisheyeroad.

### Fisheye image projection in Unity

In fisheye projections, straight lines do not appear straight. The camera in the Air- sim Car is a pin-hole camera which can give depth and segmented views. Since we need fisheye projection we need to modify the code and assets in the Unity envi- ronmnet. Unity provides a standard perspective projection with at most 120-degree field of view (FOV). Also, this limited FOV has distortion and pixel inefficiency. The solution is to render multiple perspective views and combine these together to form the required FOV [[5].](#_bookmark84)

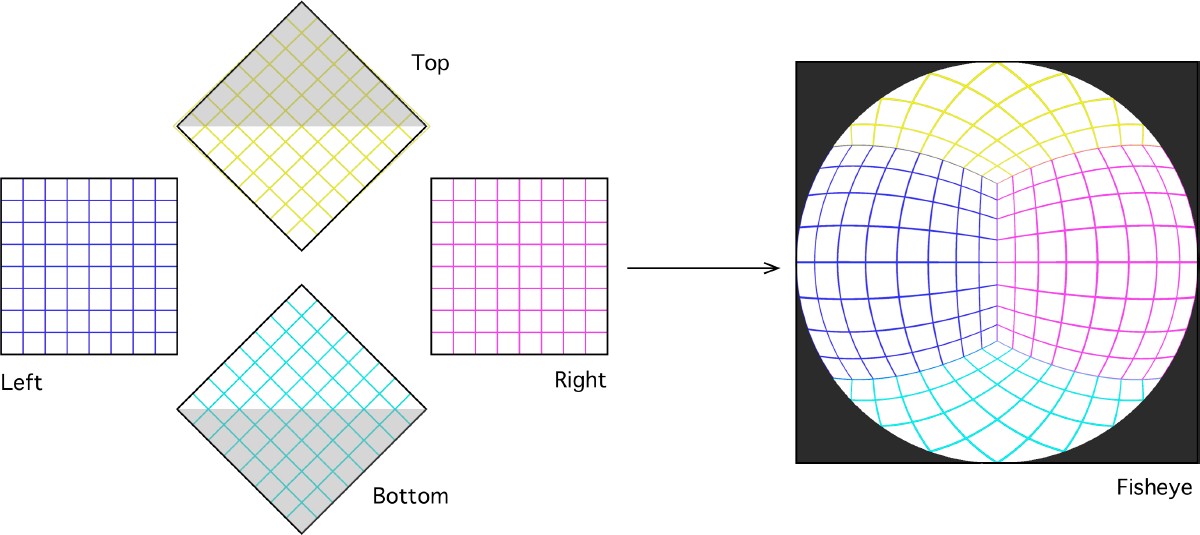


Figure 5.4: Flow diagram of the 5 pass rendering algorithm. 4 passes create the fisheye projection and an additional pass creates the geometry corrected fisheye [[5]](#_bookmark84)

According to the author in [[5],](#_bookmark84) there are two ways to do this. One is to create a vertex shader that pre-distorts the geometry of the world in such a way that the distorted geometry when viewed with an orthographic camera results in fisheye projection. But this technique is computationally expensive especially when there are multiple polygons and meshes in the scene, since a polygon may need to be drawn differently depending on both its distance and angle to the camera.

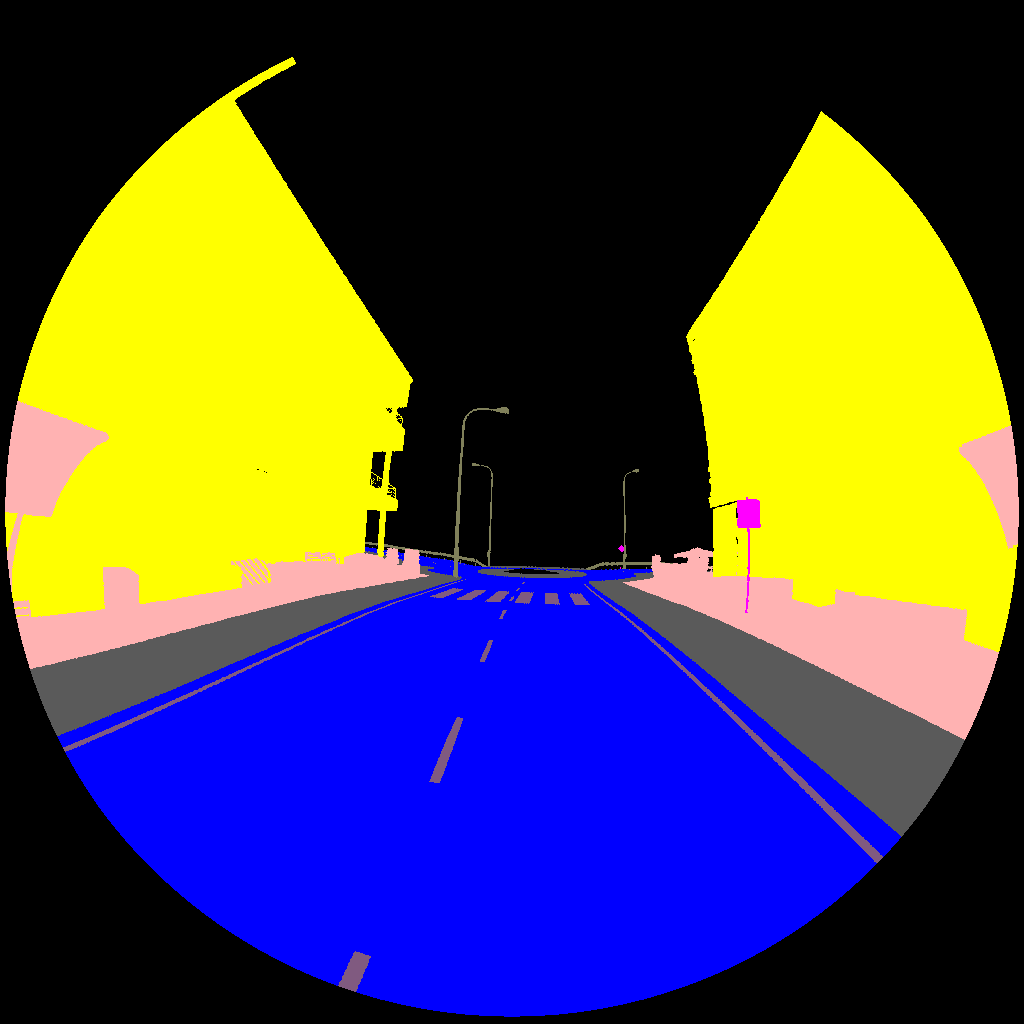
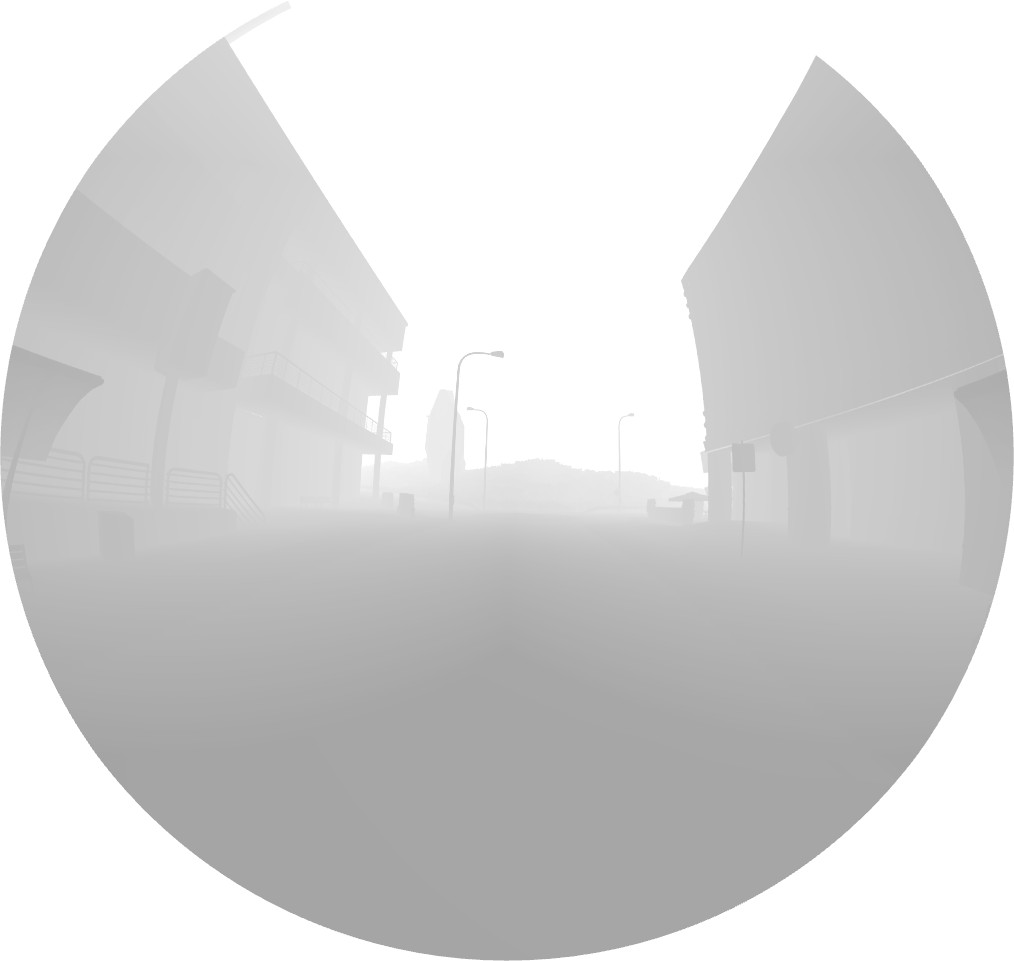
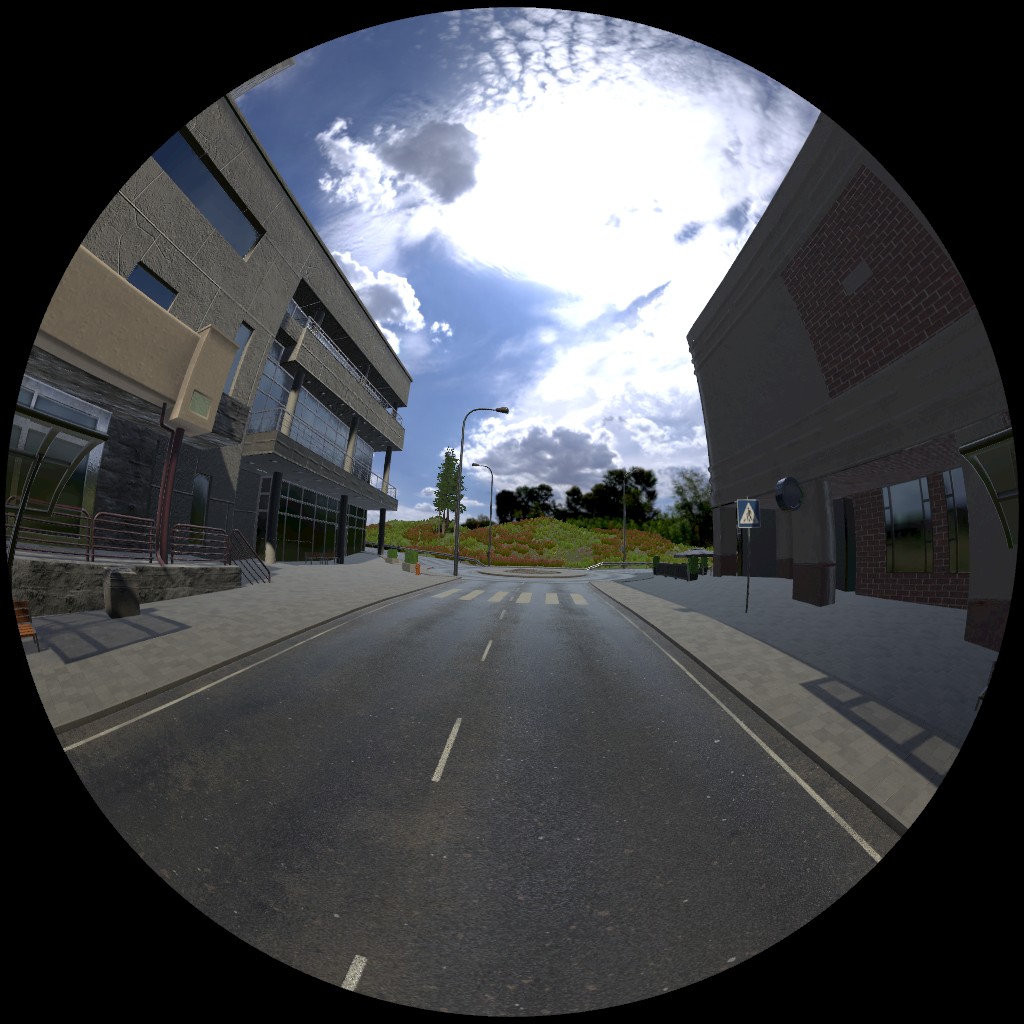
The other solution and the one used in this project is to render multiple views and to stitch those together to form a fisheye projection. Four views each of standard

perspective projection of FOV 90 degrees are taken as shown in [5.4.](#_bookmark49) These 4 renders are applied as textures to one of the 4 meshes whose texture coordinates are designed such that the final projection results in fisheye projection.

### Implementation of fisheye projection with Airsim

As mentioned in [5.1.1,](#_bookmark46) once the Windridge City environment is ready with Airsim plugin, we need to create the fisheye projection. For this we add four cameras: top, bottom, left and right, each of which renders to the four render textures which is then applied to the 4 meshes of fisheye projection. The next step is to render depth and segmentation of each of the four cameras and render to the fisheye projection. For this, four render passes are created in Unity which renders the depth and segmented view separately to the render textures of the fisheye projection as shown in figure

* 1. The car can now be driven around with the keys W,S,A,D and with the help of Unity function EncodeToPNG(), the fish-eye projection with the depth and the segmented layer data can be stored on the disk. We also extract the mapping of the semantic layer color to the class it belongs through a json file. New classes can be added by defining new layers and assigning the corresponding asset to the layer.



* + 1. Fish-eye projection (b) Fish-eye projection with (c) Fish-eye projection with

depth segmentation

Figure 5.5: Fish-projection in Unity using 4 cameras and 3 render passes

## Preprocessing the data

Using the method mentioned in the previous sections, around 2500 images are gen- erated, which are then divided into train, test and validation in the ratio of 80:10:10. The extracted images are having RGB intensities which need to be mapped to the class labels. This is done with the help of Tensorflow functions. The images are then augmented by flipping them and hence creating a dataset with around 5000 images. These images are then used to perform transfer learning on pre-trained E-Net model as explained in the next sections. The classes which are used in the project are listed

in table [5.1.](#_bookmark53) The problem is then converted to a binary classification problem by assigning Road, footpath and lane marking classes as free space and others as not free space.

Table 5.1: Class number to pixel intensity mapping

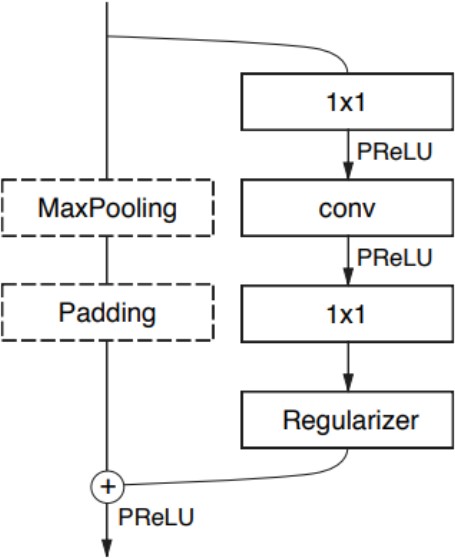
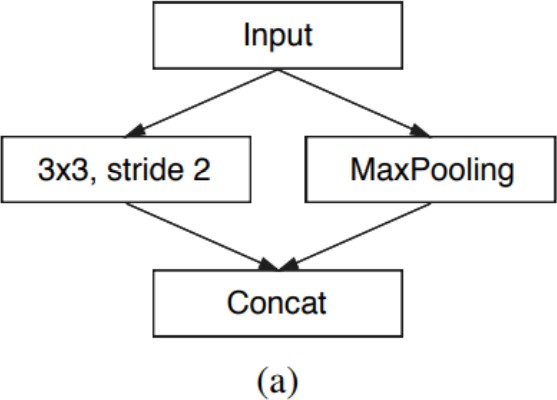
|  |  |  |  |
| --- | --- | --- | --- |
| Class number | Class name | RGB color | Free space |
| 0 | Background | [21, 18, 17] | 0 |
| 1 | Road | [0, 0, 255] | 1 |
| 2 | Buildings | [255, 255, 0] | 0 |
| 3 | Road Signs | [255, 0, 255] | 0 |
| 4 | Trees | [0, 255, 255] | 0 |
| 5 | Road and town props | [0, 178, 178] | 0 |
| 6 | Cliffs | [178, 255, 178] | 0 |
| 7 | Vehicle | [128, 128, 128] | 0 |
| 8 | Footpath | [90, 90, 90] | 1 |
| 9 | Road post | [128, 128, 90] | 0 |
| 10 | Lane Marking | [128, 90, 128] | 1 |
| 11 | Road curb | [90, 128, 128] | 0 |
| 12 | Traffic Lights | [128, 90, 0] | 0 |
| 13 | Low pixels | [0, 0, 0] | 0 |
| 14 | High pixels | [255, 255, 255] | 0 |

## Model

In autonomous vehicles, predicting free space at real time is very important. Deep neural networks for semantic segmentation have the disadvantage of requiring a large number of floating point operations and long run-times. ENet (efficient neural network) [[29]](#_bookmark108) is used for tasks requiring low latency operations. The authors claim that the network is up to 18x faster, requires 75x less FLOPs, has 79x less parameters and provides similar or better accuracy to existing mo[dels[29].](#_bookmark108) This network is used as the base model for the transfer learning in our project.

### Network architecture

The network architecture is as shown in table [5.3.](#_bookmark64) The network is divided into several stages as highlighted by the horizontal lines in the table and the first digit after each block name. Output sizes in the table are for an input image of size 512\*1024. The initial block is represented as shown in [5.6(a)](#_bookmark56) and the bottleneck is represented as shown in [5.6(b).](#_bookmark57)



* + - 1. Initial Block (b) Bottleneck block

Figure 5.6: Visual representation of ENet modules [[29]](#_bookmark108)

#### Initial block

The initial block as shown in [5.6(a)](#_bookmark56) consists of a single block. The input image of size 512 \* 1024 is fed to the input layer which is then passed through a convolution layer with 13 filters each of size 3\*3\*3. A bias vector of size 13 and convolution strides of [1,2,2,1] are used. Thus an output feature map of size 256\*512\*13 is obtained. Then, a maxpooling operation with a kernel size of [1, 2, 2, 1] and strides of [1, 2, 2, 1] is performed on the input image which gives an output feature map of size 256\*512\*3. These two feature maps are concantenated to obtain a final output feature map of size 256\*512\*16. Finally, batch normalization and PReLU activation are computed on the output feature map. The weights for convolution filter are initialized with Xavier [initializer[14]](#_bookmark93) and zero padding is used.

#### Bottleneck module

The bottleneck module is as shown in [5.6(b).The](#_bookmark57) features of the bottleneck module are summarised as below.

Each block consists of three convolution layers: 1\*1 projection that reduces the dimensionality, a main convolution layer and a 1\*1 expansion.

•

Batch Normalization [[19]](#_bookmark98) and PReLU [[18]](#_bookmark97) are added between all convolutional layers

•

If the bottleneck is downsampling, a max pooling layer is added to the main branch. Also, the first convolution is replaced with 2\*2 convolution with stride of 2 in both dimensions. Then the output is zero padded to match the size of required output feature map.

•

The main convolution (’conv’ in [5.6(b))](#_bookmark57) is either regular, dilated or full con- volution with 3\*3 filters. Sometimes, it is asymmetric convolution with a sequence of 5\*1 and 1\*5 convolutions.

•

Regularizer is Spatial Dropout [[38]](#_bookmark117) with *ρ*=0.01 before bottleneck 2.0 and

•

*ρ*=0.1 afterwards.

Bias terms are not used for 1\*1 projection and expansion to reduce the number of kernel calls.

•

In the decoder,as shown in dotted lines in [5.6(b),](#_bookmark57) max pooling is replaced with max unpooling and padding is replaced with spatial convolution without bias. Also, decoder uses ReLU instead of PReLU and no regularizer.

•

The network architecture as shown in table [5.3](#_bookmark64) is composed of an initial block followed by successive bottleneck modules. The network can be thus divided into seven layers: initial block, bottleneck 1-5 and a fully-connected layer. The layers 2-4 can be considered as encoder layers and layers 5-6 as decoder layers. The fully convolution layers at the end takes most of the processing time of the decoder.

Table 5.2: Network architecture

|  |  |  |
| --- | --- | --- |
| Name | Type | Output size |
| initial |  | 256\*512\*16 |
| bottleneck 1.0 | downsampling | 128\*256\*64 |
| bottleneck 1.1 | downsampling | 128\*256\*64 |
| bottleneck 1.2 | downsampling | 128\*256\*64 |
| bottleneck 1.3 | downsampling | 128\*256\*64 |
| bottleneck 1.4 | downsampling | 128\*256\*64 |
| bottleneck 2.0 | downsampling | 64\*128\*128 |
| bottleneck 2.1 | downsampling | 64\*128\*128 |
| bottleneck 2.2 | dilated 2 | 64\*128\*128 |
| bottleneck 2.3 | asymmetric 5 | 64\*128\*128 |
| bottleneck 2.4 | dilated 4 | 64\*128\*128 |
| bottleneck 2.5 |  | 64\*128\*128 |
| bottleneck 2.6 | dilated 8 | 64\*128\*128 |
| bottleneck 2.7 | asymmetric 5 | 64\*128\*128 |
| bottleneck 2.8 | dilated 16 | 64\*128\*128 |
| bottleneck 3.1 |  | 64\*128\*128 |
| bottleneck 3.2 | dilated 2 | 64\*128\*128 |
| bottleneck 3.3 | asymmetric 5 | 64\*128\*128 |
| bottleneck 3.4 | dilated 4 | 64\*128\*128 |
| bottleneck 3.5 |  | 64\*128\*128 |
| bottleneck 3.6 | dilated 8 | 64\*128\*128 |
| bottleneck 3.7 | asymmetric 5 | 64\*128\*128 |
| bottleneck 3.8 | dilated 16 | 64\*128\*128 |
| bottleneck 4.0 | upsampling | 128\*256\*64 |
| bottleneck 4.1 |  | 128\*256\*64 |
| bottleneck 4.2 |  | 128\*256\*64 |
| bottleneck 5.0 | upsampling | 256\*512\*16 |
| bottleneck 5.1 |  | 256\*512\*16 |
| fullconv |  | 512\*1024\*16 |

### Transfer learning using tensorflow

The dataset generated using Unity is trained on pretrained E-net model [[29].](#_bookmark108) The model is selected on the basis of the author’s claim that it performs well in real time. Pretrained E-net model which was trained on Cityscapes dataset [[8]](#_bookmark87) for 23 epochs was taken as the starting point in the model design. The fully convolution layer of the E-Net model is modified with outputs as a tensor of size 15 corresponding to the 15 classes of our dataset. The model is then trained for 100 epochs with the same hyperparameters as the original model.

1. The pre-trained model is first restored using the below code.

s a v e r = t f . t r a i n . S a v e r ( t f . t r a i n a b l e v a r i a b l e s ( ) [ : − 2 ] )

s a v e r . r e s t o r e ( s e s s , ” . / m o d e l 1 e p o c h 2 3 . c k p t ”)

1. The tensor before the fully-connected layer is retreived.

l a y e r 5 = model . graph . g e t t e n s o r b y n a m e ( ” R e l u 8 1 : 0 ” )

1. Flow of gradients to intial layers can be stopped using the below code. This is an optional step. In our project this is not done so that the entire network is trained with weights initialized from pretrined E-Net.

l a y e r 5 = t f . s t o p g r a d i e n t ( l a y e r 5 )

1. A new fully convolution layer is added with the number of neurons as the number of classes.

### Loss function and training

#### Softmax cross entropy loss

The loss function used in the project is pixel-wise cross entropy loss which examines each pixel individually, comparing class predictions to one-hot encoded target vector. Since, the cross entropy loss evaluates class predictions for each pixel vector and then averages over all pixels, it is a problem in our case since the classess have unbalanced representation in the image. The solution is to use class weights in the calculation of average loss.

#### Class weights

The authors in [[29]](#_bookmark108) have calculated the class weights for the training dataset by using a custom equation as shown in [5.1.](#_bookmark62)

where

1

*Wclass* = *ln*(*c* + *p*

*class*

(5.1)

)

c = 1.02, a hyperparameter, which restricts the class weights in the range [1, 50 ] and *pclass* is the class probability of each pixel for the whole training set.

#### Weight decay

To prevent overfitting, L2 regularization is added to the loss function. This is done by calculating the L2 norm of each of the weight and bias tensors and multiplying them with the weight decay factor. The new loss function with L2 regularization is as shown in equation [5.2.](#_bookmark63)

*Ltotal* = *L* + *λ*"*w*"2

2

(5.2)

These L2 terms are collected for all the weights and biases and are added to the weighted softmax cross entropy loss function.

#### Training

The model is trained using Adam optimizer [[22]](#_bookmark101) for a total of 100 epochs. The hyperparameters used for training are listed in table

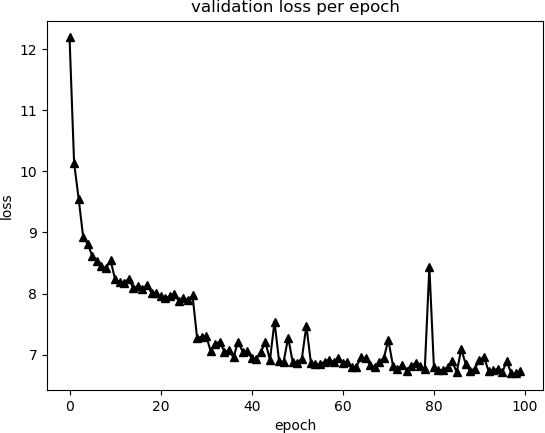
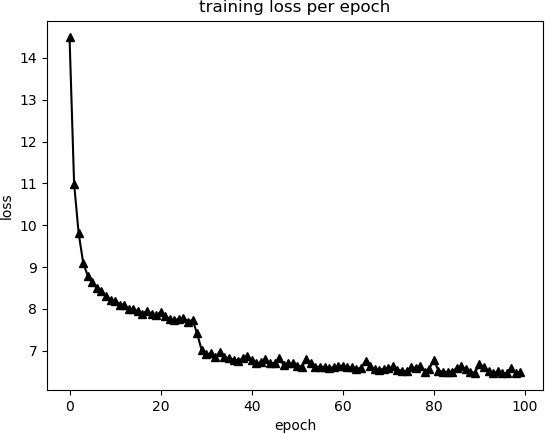
Table 5.3: Hyperparameters used for training

|  |  |
| --- | --- |
| Hyperparameter | Value |
| Learning rate | 0.0005 |
| Weight decay | 0.0002 |
| Epochs | 100 |
| Batch size | 4 |
| Keep probability *ρ* | 0.01, 0.1 |

# Results

## Training and validation performance of the model

The results comparing the loss to the number of epochs for both training and vali- dation is as shown in figure [6.1.](#_bookmark67) As seen in the figure, the training and validation loss decrease from a high value to values close to 4.0 This shows that the model is not overfitting the data and has learnt the features well. The model can be further trained by varying the hyperparameters and modifying the architecture, however, these results are enough to demonstrate that a dataset generated from the simulation can be used to train and validate a neural network.



* + 1. Training loss vs epochs (b) Validation loss vs epochs

Figure 6.1: Training and validation losses as compared to the epochs

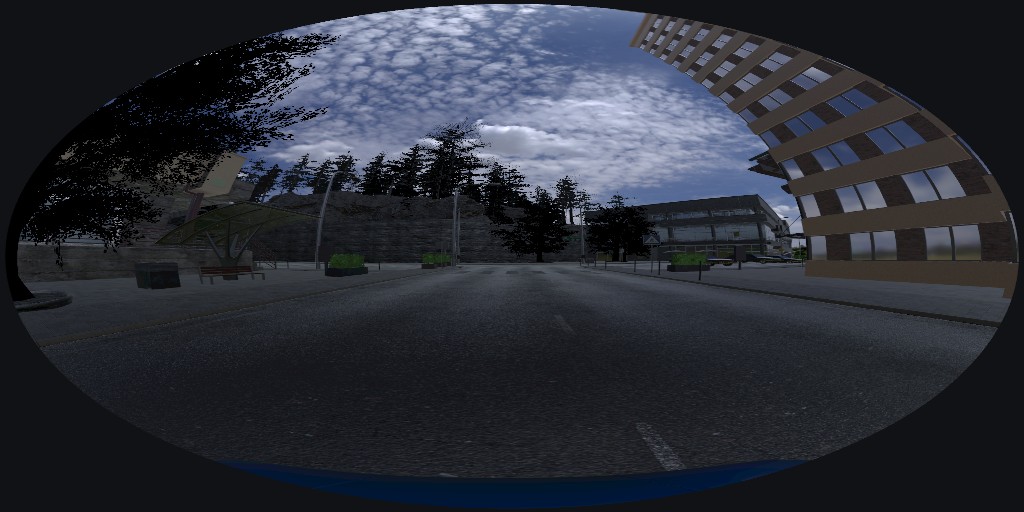
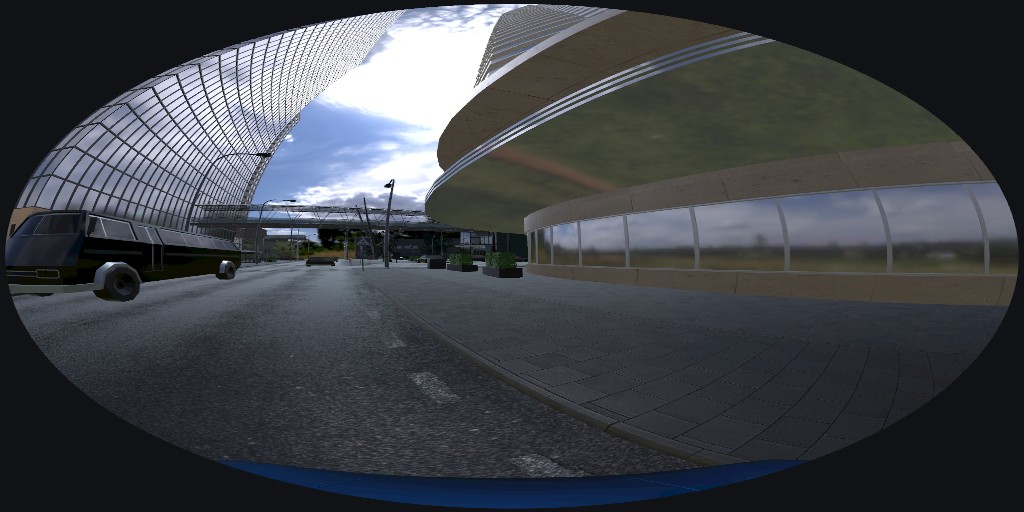
## Model evaluation on simulated test data

For evaluating the model, a test set of 141 images was created initially which is used to evaluate after the training. The model predicts 15 classes, however for our purposes, we make the problem binary classification by assigning the labels as free space or not as shown in table [5.1.](#_bookmark53) The average metrics for pixel accuracy, pixel precision and pixel recall are then calculated and are given in table [6.1](#_bookmark69)

Table 6.1: Pixel wise results comparison of E-net vs our model on test dataset

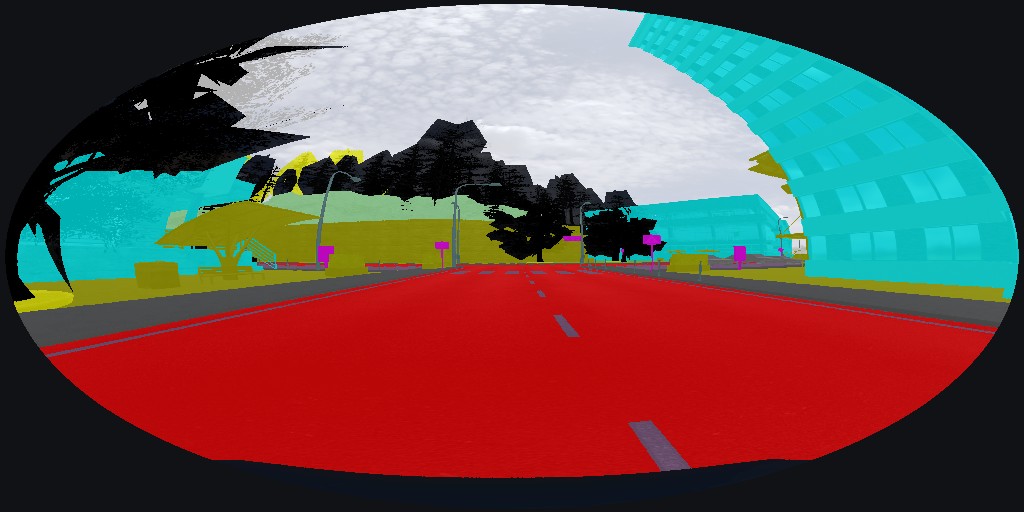
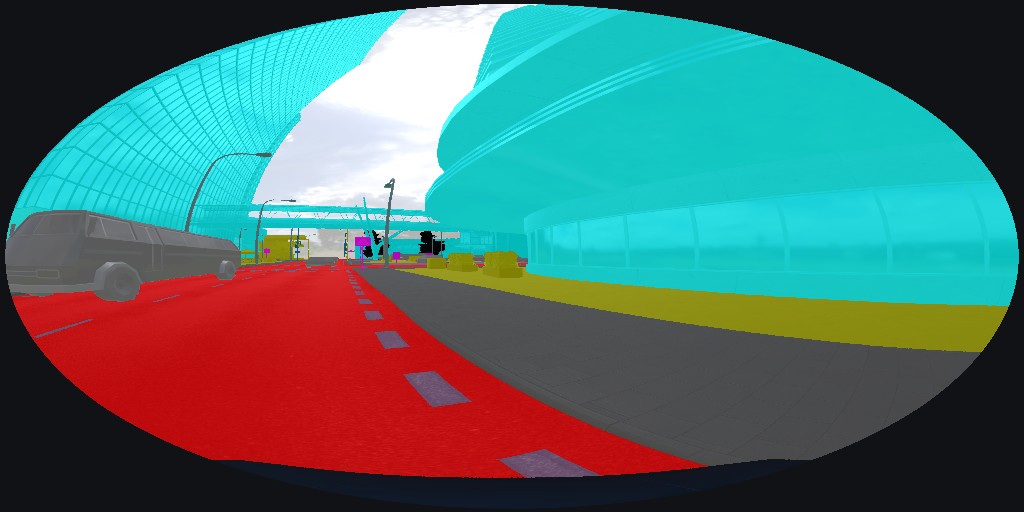
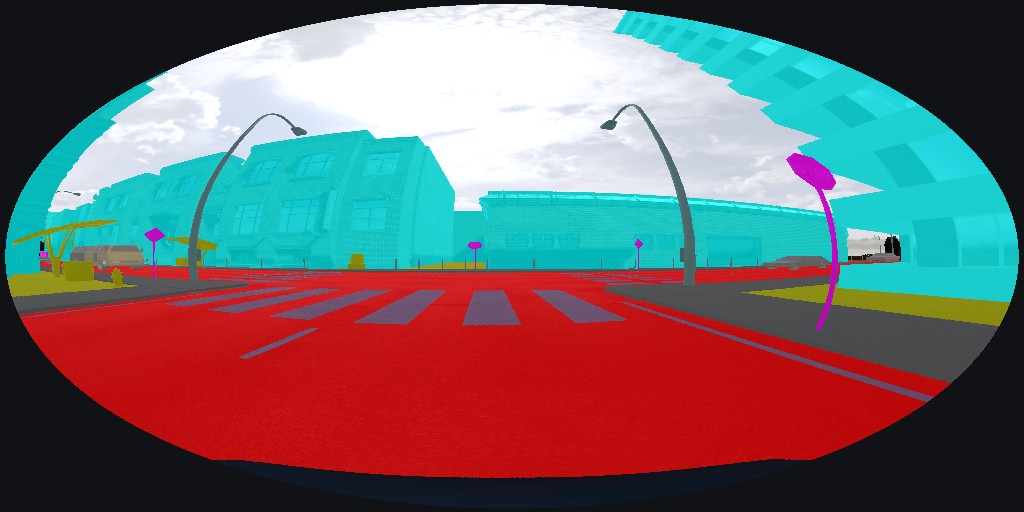
|  |  |  |  |
| --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall |
| E-Net model | 71.2% | 77.3% | 56.7% |
| Our model | 87.2% | 98.5% | 55% |

The images and ground truth are as shown in figure [6.2](#_bookmark70) and [6.3](#_bookmark71) respectively. The predictions of using our model trained for 100 epochs is as shown in figure [6.4.](#_bookmark72) The predictions of E-net model are shown in figure [6.5.](#_bookmark73) As seen from the figures and from table [6.1](#_bookmark69), our model performs better on fisheye images than the E-net model.



* + 1. (b) (c)

Figure 6.2: Simulated images from unity

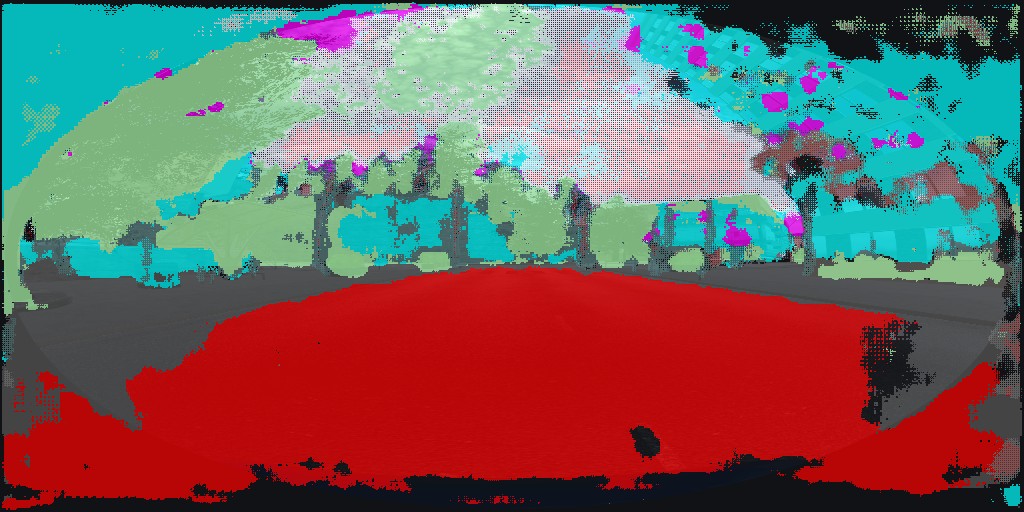
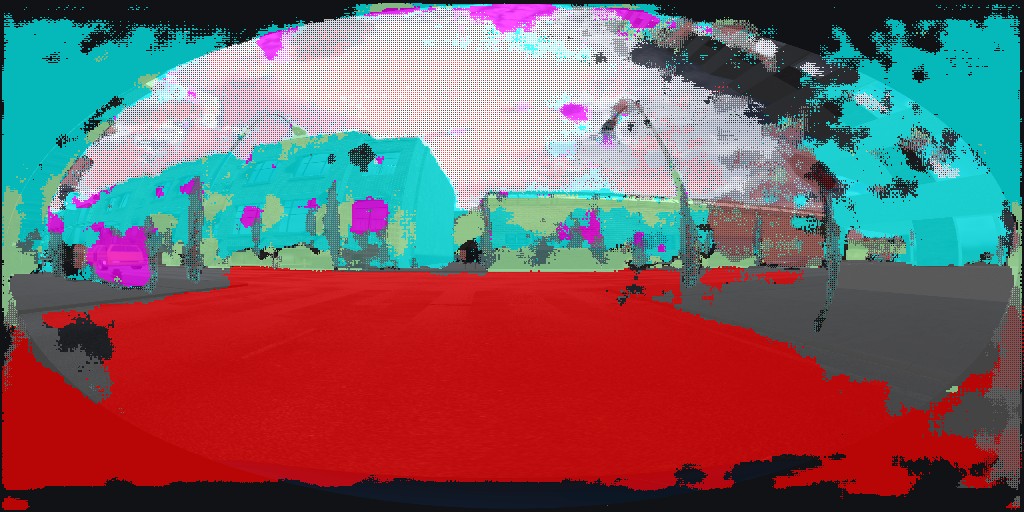


(a) (b) (c)

Figure 6.3: Ground truth labels

(a) (b) (c)

Figure 6.4: Predicted labels from our model



(a) (b) (c)

Figure 6.5: Predicted labels from Enet

## Model evaluation on real dataset

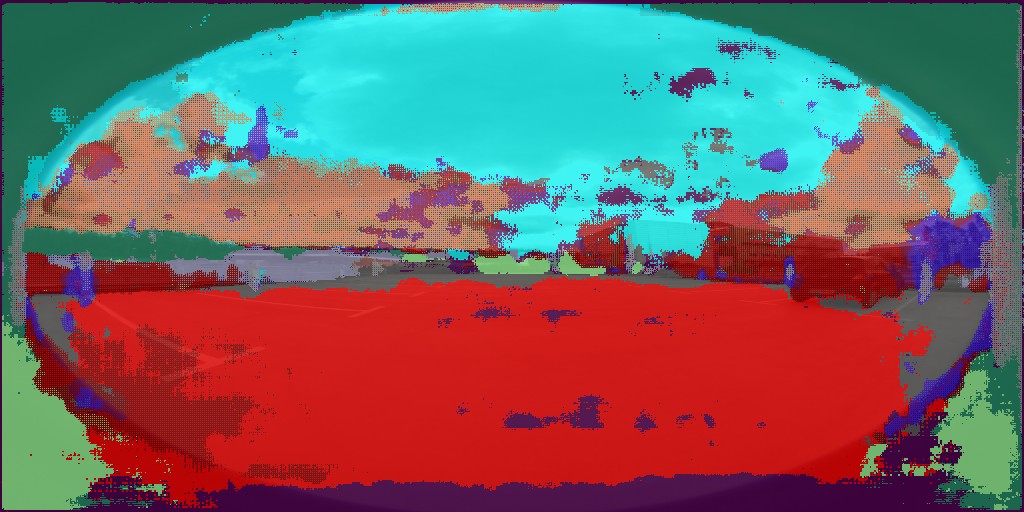
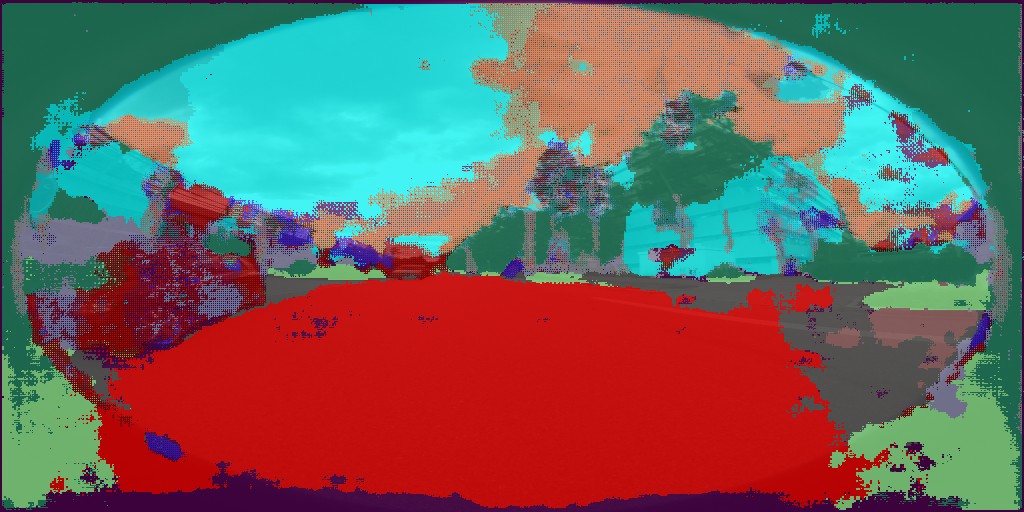
The main goal of this project is to detect free space in real images capture by fisheye lens. Hence, for the evaluation of the model, real world dataset called LMS Fisheye

1. is used. The authors have used Fujinon FE185C057HA-1 fisheye lens which can capture FOV of 185 degrees.

For comparison, the E-Net [[29]](#_bookmark108) model which was trained on pin-hole camera im- ages of cityscapes dataset [[8]](#_bookmark87) was used to evaluate LMS fisheye [[13]](#_bookmark92) images of road scenes. Some example ground truth images are shown in figure [6.6.](#_bookmark75) The results of using E-Net model trained on cityscape images are in figure [6.7](#_bookmark76) and the results of using our model trained on simulated images from Unity are as shown in . The quantitative analysis is not performed due to unavailability of ground truth labels. However, looking at the results we can see that our model is performing better than the E-net model.

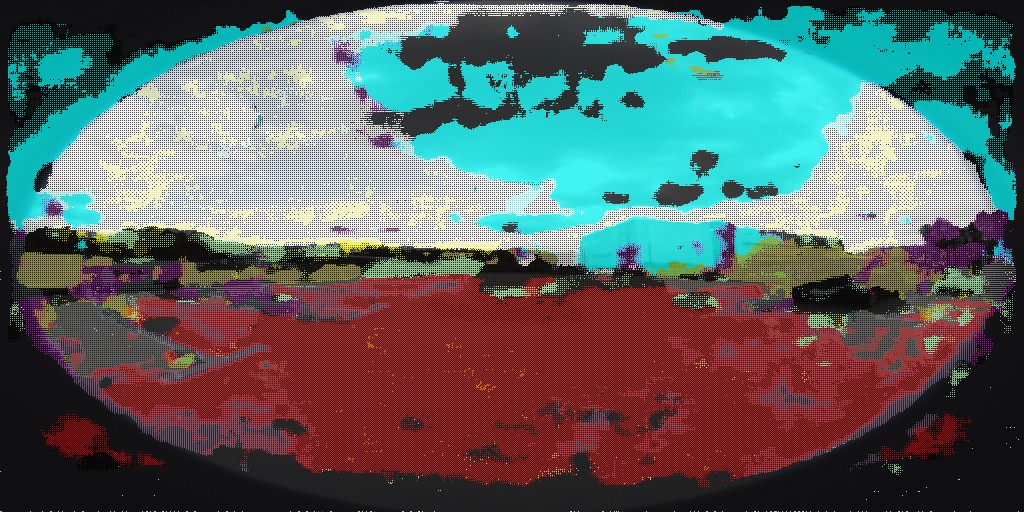
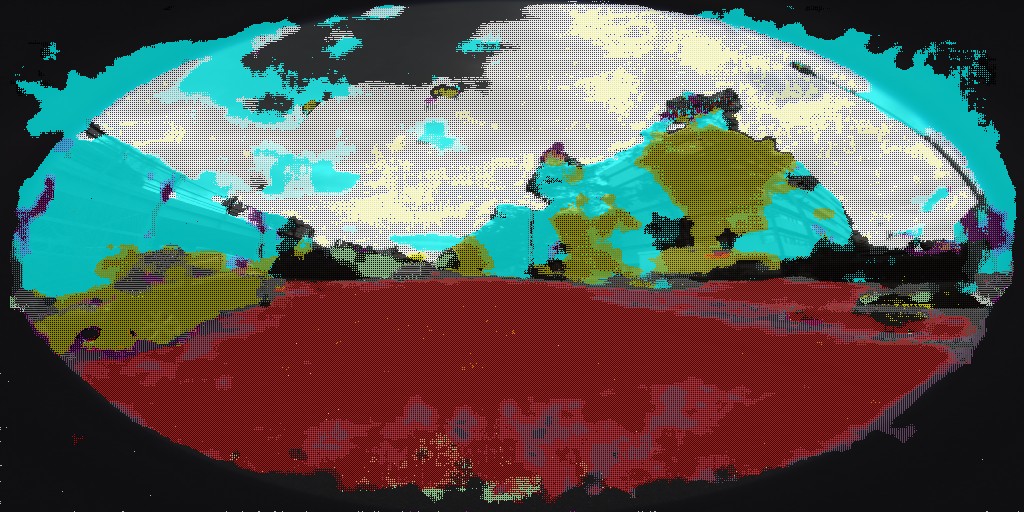
* 1. (b) (c)

Figure 6.6: LMS Fisheye groundtruth images



(a) (b) (c)

Figure 6.7: Results of using the E-net model trained on cityscapes



(a) (b) (c)

Figure 6.8: Results of using our model trained on simulated fisheye images

As seen in the results, our model clearly outperforms E-net on fisheye images. How- ever, the model can be further improved by training on a larger dataset and by using better hyperparameters. Through this project, we demonstrated that simula- tion can be used to create images as well as labels when there is no labelled dataset available. Using this simulated dataset, we can train a neural network, either from scratch or through transfer learning to obtain good results. The trained neural net- work can then be used for inference in real world datasets thus solving the problem of unavailability of a labelled dataset. Also, through this project we demonstrated the importance of using fisheye cameras to better understand the surroundings in a autonomous car. We also showcased the potential of using Unity simulation to create any dataset for neural network training. Further work can be done on au- tomating the dataset creation in Unity and also on using the neural network to drive the car inside Unity environment.

1. Alvarez, J.M., Gevers, T., LeCun, Y., Lopez, A.M.: Road scene segmentation from a single image. In: European Conference on Computer Vision. pp. 376–

389. Springer (2012)

1. Awad, M., Khanna, R.: Efficient learning machines: theories, concepts, and applications for engineers and system designers. Apress (2015)
2. Badino, H., Franke, U., Pfeiffer, D.: The stixel world-a compact medium level representation of the 3d-world. In: Joint Pattern Recognition Symposium. pp. 51–60. Springer (2009)
3. Bearman, A., Russakovsky, O., Ferrari, V., Fei-Fei, L.: What’s the point: Se- mantic segmentation with point supervision. In: European conference on com- puter vision. pp. 549–565. Springer (2016)
4. Bourke, P.: idome: Immersive gaming with the unity3d game engine. CGAT09 Computer Games, Multimedia and Allied Technology 9, 265–272 (2009)
5. Brust, C.A., Sickert, S., Simon, M., Rodner, E., Denzler, J.: Convolutional patch networks with spatial prior for road detection and urban scene under- standing. arXiv preprint arXiv:1502.06344 (2015)
6. Ciresan, D., Giusti, A., Gambardella, L.M., Schmidhuber, J.: Deep neural networks segment neuronal membranes in electron microscopy images. In: Ad- vances in neural information processing systems. pp. 2843–2851 (2012)
7. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
8. Cs´aji, B.C.: Approximation with artificial neural networks
9. Deng, L., Li, J., Huang, J.T., Yao, K., Yu, D., Seide, F., Seltzer, M.L., Zweig, G., He, X., Williams, J.D., Gong, Y., Acero, A.: Recent advances in deep learning for speech research at microsoft. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing pp. 8604–8608 (2013)
10. Deng, L., Yang, M., Li, H., Li, T., Hu, B., Wang, C.: Restricted deformable convolution based road scene semantic segmentation using surround view cam- eras. arXiv preprint arXiv:1801.00708 (2018)
11. Deng, L., Yang, M., Qian, Y., Wang, C., Wang, B.: Cnn based semantic seg- mentation for urban traffic scenes using fisheye camera. In: 2017 IEEE Intelli- gent Vehicles Symposium (IV). pp. 231–236. IEEE (2017)
12. Eichenseer, A., Kaup, A.: A data set providing synthetic and real-world fisheye video sequences. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1541–1545 (March 2016)
13. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedfor- ward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256 (2010)
14. Haltakov, V., Belzner, H., Ilic, S.: Scene understanding from a moving cam- era for object detection and free space estimation. In: 2012 IEEE Intelligent Vehicles Symposium. pp. 105–110. IEEE (2012)
15. H¨ane, C., Sattler, T., Pollefeys, M.: Obstacle detection for self-driving cars using only monocular cameras and wheel odometry. In: 2015 IEEE/RSJ Inter- national Conference on Intelligent Robots and Systems (IROS). pp. 5101–5108. IEEE (2015)
16. H¨anisch, S., Evangelio, R.H., Tadjine, H.H., P¨atzold, M.: Free-space detection with fish-eye cameras. In: 2017 IEEE Intelligent Vehicles Symposium (IV). pp. 135–140. IEEE (2017)
17. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
18. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
19. Janai, J., Gu¨ney, F., Behl, A., Geiger, A.: Computer vision for au- tonomous vehicles: Problems, datasets and state-of-the-art. arXiv preprint arXiv:1704.05519 (2017)
20. Khan, A., Sohail, A., Zahoora, U., Qureshi, A.S.: A survey of the recent archi- tectures of deep convolutional neural networks. arXiv preprint arXiv:1901.06032 (2019)
21. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
22. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM 60, 84–90 (2012)
23. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature 521(7553), 436 (2015)
24. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation 1(4), 541–551 (1989)
25. Liu, Y.C., Lin, K.Y., Chen, Y.S.: Bird’s-eye view vision system for vehicle surrounding monitoring. In: International Workshop on Robot Vision. pp. 207–

218. Springer (2008)

1. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)
2. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering 22(10), 1345–1359 (2010)
3. Paszke, A., Chaurasia, A., Kim, S., Culurciello, E.: Enet: A deep neural net- work architecture for real-time semantic segmentation. CoRR abs/1606.02147 (2017)
4. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
5. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015)
6. S´aez, A., Bergasa, L.M., Romeral, E., L´opez, E., Barea, R., Sanz, R.: Cnn- based fisheye image real-time semantic segmentation. In: 2018 IEEE Intelligent Vehicles Symposium (IV). pp. 1039–1044. IEEE (2018)
7. Sanberg, W.P., Dubbleman, G., et al.: Free-space detection with self-supervised and online trained fully convolutional networks. Electronic Imaging 2017(19), 54–61 (2017)
8. Shah, S., Dey, D., Lovett, C., Kapoor, A.: Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: Field and Service Robotics (2017), <https://arxiv.org/abs/1705.05065>
9. Shotton, J., Johnson, M., Cipolla, R.: Semantic texton forests for image cat- egorization and segmentation. In: 2008 IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–8. IEEE (2008)
10. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE 105(12), 2295–2329 (2017)
11. Technologies, U.: Unity editor (2019), <https://unity3d.com/>
12. Tompson, J., Goroshin, R., Jain, A., LeCun, Y., Bregler, C.: Efficient object localization using convolutional networks. In: Proceedings of the IEEE Confer- ence on Computer Vision and Pattern Recognition. pp. 648–656 (2015)
13. Tsutsui, S., Kerola, T., Saito, S., Crandall, D.J.: Minimizing supervision for free-space segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 988–997 (2018)
14. Wang, C., Zhang, H., Yang, M., Wang, X., Ye, L., Guo, C.: Automatic parking based on a bird’s eye view vision system. Advances in Mechanical Engineering 6, 847406 (2014)
15. Yu, F., Koltun, V.: Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122 (2015)
16. Zhao, B., Feng, J., Wu, X., Yan, S.: A survey on deep learning-based fine- grained object classification and semantic segmentation. International Journal of Automation and Computing 14(2), 119–135 (2017)
17. Zhu, H., Meng, F., Cai, J., Lu, S.: Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation. Journal of Visual Communication and Image Representation 34, 12–27 (2016)