Computer Vision - Semantic Segmentation using PSPNet

Masters in Information Technology
Autonomous Intelligent System WS 2020/2021

Komala Balakrishna-1325111
komala.balakrishna@stud.fra-uas.de

Sahana Prasad-1324329
sahana.prasad@stud.fra-uas.de

***Abstract*— This paper explains the task of computer vision which is mainly the detection of objects and segmenting the objects which we captured using the depth camera. The detailed explanation and analysis are also included in the paper.**

The breakthrough progress is achieved by Deep Learning in sematic segmentation. Semantic Segmentation plays a vital role in understanding the machine images. For unrestricted and discrete scene, it is very challenging to do scene parsing. In our paper we will explain what global context information is capable along with pyramid scene parsing network (PSPNet). For predicting pixel level information PSPNet provide a very good framework. On various dataset state-of-the-art is performed by the proposed approach.

Keywords—. Object detection, Deep Learning, Semantic Segmentation, PSPNet.

I. INTRODUCTION

This project is a part of Autonomous Intelligent Systems module of Frankfurt University of Applied Sciences. This project is used to solve the problem of computer vision which is used to detect the object and then segmenting the object by classifying each object into different category.

The present world is the place for all the new technologies where the humans are replaced by machines to do the jobs. Any kind of objects can be detected by humans in fraction of seconds. The human skill for recognizing the object is built over the period after watching the objects many times. In the same we can train the machine by looking over the objects again and again and by trying to learn features of the objects. To obtain the most accurate results we need to build the machines and train them to be useful.

Deep learning is the function of artificial intelligence which will imitate the human brain for data processing, pattern creation which will be used in making the decision. Deep learning functions can learn without any supervision from humans. It is also able to learn the data which is not labelled and not structured. As it is a form of machine learning it can be used to identify frauds, money laundering among with many other functions. Deep learning is grown in this digital era which has led to explosion of data from all the regions of the world and in all the forms. This data is also called as big data is derived from sources like search engines of internet, e-commerce, social media, etc. This huge amount of data can be easily accessed and can also be shared through cloud computing applications. As the data is unstructured and so huge that it might take decades to comprehend by humans. Hence as a result the companies vigorously adapting Artificial Intelligence systems for the support.

Computer Vision comes the field of computer science it concentrates on creating systems which can analyze, process in the same way humans do. The common tasks achieved by computer vision are a) Object identification- The system identifies objects in video/image. b) Object classification- The system classifies to the defined category on photo/video. c) Object tracking- The system processes the video which matches the search criteria and tracks its every movement. The computer vision algorithms that we use in recent days are dependent on pattern recognition. Computer is trained on large amount of data and then images are processed by the computer, then objects are labelled, and finally patterns are found in those objects.

Semantic Segmentation is a computer vision problem which requires identification of objects that appears in the image and localizing them properly in pixel size manner. Before the revolution of deep learning in computer vision techniques such as decision trees [1] or markov random fields [2] were used for semantic segmentation. Now deep models based of Convolution Neural Networks (CNN) are dominant. Semantic segmentation architecture can be divided into encoder network which is followed by the decoder network. Semantic segmentation will not only require discrimination at pixel level it also requires a mechanism to project the features learnt at different stages of encoder onto to the pixel space. There are different approaches which employ different mechanisms as a part of decoding.

Pyramid Scene Parsing Network also called PSPNet is one of the semantic segmentation models. Scene parsing which depends on semantic segmentation is a significant topic in computer vision. The main aim is to assign a category label to every pixel in the image. The label and shape of every pixel is predicted. This project is in the extensive interest of indoor objects.

Scene parsing difficulties are associated to label variety and the scene. For segmentation of image scene PSPNet performs way better than other networks like FCN, Deeplab, U-Net. It is very well recognized algorithm for image segmentation and the paper is mentioned in the community of computer vision. Framework for scene parsing is largely dependent on fully convolutional network (FCN). There was a major issue in models based on FCN which was missing strategy to use the

global scene category hints. For understanding of complex scene, spatial image pooling was implemented to get features which is of global level. The ability is further enhanced by spatial image pooling network [3]. Pyramid Scene Parsing was proposed which was different from all the methods. The PSPNet model was required as pixel-based classifiers was unable to capture whole image context.

In this project the approach we used will achieve the state of art performance. For tasks which are of pixel level the directions are given by PSPNet. It also helps in optical flow, stereo matching, etc.

- Difficult scenery context features are embed using pyramid scene parsing network.
- Based on deeply supervised loss effective optimization strategy is developed.
- For semantic segmentation and state of art scene parsing, a practical system is built which includes all crucial details of implementation.

II. PARAMETERS OF PROJECT

A. Open Source Computer Vision Library (OpenCV)

It is an open-source software library of computer vision and machine learning. Various computer vision problems are solved using this OpenCV libraries. These libraries can also be used computer vision applications like image segmentation, image transformation, object tracking, feature detection etc.

The OpenCV library has been divided into several modules, each of which is dedicated to a specific category of computer vision problems. The word space cv contains all of the groups and functions. To get to them, we can either use the namespace cv declaration to precede the main function description, or we can use the namespace specification cv:: to prefix OpenCV class and function names. The main object belongs to the Mat class. It's basically a matrix that holds the pixel values of an image as well as a variety of attributes about the image, as implied by the class name. An picture can be formed as cv:: in the simplest case. Mat image, resulting in a 0 by 0 image. Data where image is perhaps the most significant member variable of an image object. The data member is simply a pointer to the allocated memory block containing the image data (in this example, image.data=0). Alternatively, we can define an initial size and form for each matrix unit when creating a Mat object. CV 8U specifies signed 1-byte pixel image values, CV 8UC3 specifies three channels for a color image, and CV 32F specifies 32-bit/64-bit floating point numbers. [6]

Pixels are the picture components that make up the basic image material. As a result, since images can easily have tens of thousands of pixels, efficient pixel processing is critical. Given that a matrix is a fundamental data structure in OpenCV, it seems fair to assume that each matrix element represents one pixel. However, pixel values in a gray level image are usually unsigned 8bit values, while in a color image, three such 8bit unsigned values per pixel are used to represent three primary colors, i.e., channels (Red, Green and Blue; actually, OpenCV uses BGR channel order). In a broader sense, OpenCV enables the development of matrices of different value types (as previously described), with the caveat that such operations (pixel processing) can only be applied to specific matrix types. Furthermore, choosing the right color space for some image processing might make it more successful. [6]

For many years now it has become a point of reference for experts and all passionate people of this discipline. Now this library has more than 2500 algorithms in the version 4.x. which has been optimized over the years. With image or video this algorithm allows to identify objects, detect face and it also helps in classifying actions which are performed by humans by video. The OpenCV is used for following reasons:

- Reading and writing images.
- Capturing and saving the videos.
- Processing images using filtering and transformation. It also performs the feature detection.

It has a modular structure meaning several many shared or static libraries are included in the package. The modules which are available are as follows:

- Core functionality (core) – A module which defines the basic data structure. The dense multidimensional array is also included along with basic functions of other modules.
- Object detection (objdetect) – Object detection and detecting of predefined classes like faces, eyes, mugs, people, bikes, etc.
- Image processing (imgproc) – The linear and nonlinear image filtering, color space conversion, histograms, geometrical image transformations (perspective wrapping, resize, generic table-based remapping) and so on are included in this image processing module.
- Camera Calibration and 3D Reconstruction (calib3d) – Single and stereo camera calibration, basic multiple-view geometry algorithms, stereo correspondence algorithms, and object pose estimation, elements of 3D reconstruction.
- Video analysis (video) – The object tracking algorithm, motion estimation, background subtraction algorithms are included in this module.
- 2D Features Framework (features2d) - detectors, descriptor matchers and salient feature descriptors.
- Some other modules, such as Python bindings, High-level GUI, Video I/O, and others.

In this project the implementation of all the data augmentation is done using OpenCV which does rotation.

B. TensorFlow

Among the plethora of deep learning libraries, TensorFlow, which was developed by Google researchers, is the most common. Neural networks have been a huge success in the world of deep learning, and they've gained a lot of traction in a variety of fields. Because of its versatility and scalability, this family of models has a lot of potential for promoting data analysis and modeling for different problems in educational and behavioral sciences. However, putting these models and

optimization algorithms into practice takes time and is vulnerable to errors. Fortunately, TensorFlow makes neural network analysis and deployment much easier and faster. [8]

For machine learning and deep learning, it is an open-source library. All traditional machine learning is supported by this. Without considering deep learning the TensorFlow was initially developed for a very big numerical computation. Later it was developed for deep learning as well and it proved to be very useful as a reason google open sourced the TensorFlow.

Multidimensional arrays with higher dimensions called tensors is fed as input to the TensorFlow. Multidimensional arrays handle large amount of data easily.

It is predominantly used for classification, prediction, understanding, discovering and so on. TensorFlow means data flow graphs. For parallel computing dataflow is the common model and for representation of computation data flow graphs are used by TensorFlow. The main advantages of using TensorFlow is as follows:

- It is very flexible in creating all sort of computation in python API.
- In C++ it includes ML operations which are highly efficient.
- The distributed/parallel computing is supported by TensorFlow.
- It is a very popular open source, and the community is constantly contributing to improve.
- Due to mind blowing advancement in artificial intelligence and machine learning, TensorFlow acts a powerful tool to achieve the goals.

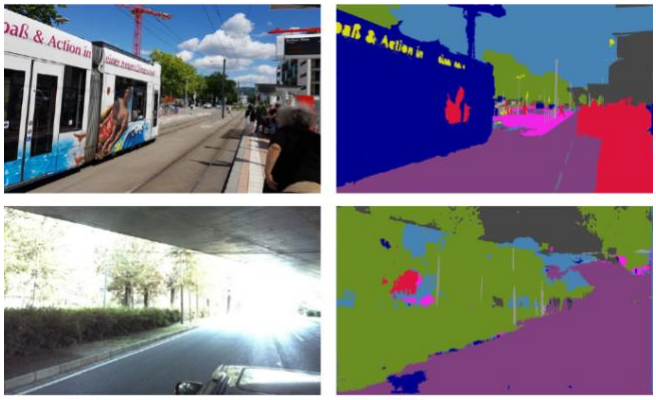
- There are many amazing projects which are done using TensorFlow for example image classification, speech recognition, jazz generation which is driven by deep learning and so on. In this project TensorFlow is used for object detection.

C. Pyramid Scene Parsing Network (PSPNet)

When FCN is applied for scene parsing, we can observe and analyze the failed cases. In complex scene parsing the performance is object and identification is improved. The common problems of complex scene parsing are summarized below:

a) Mismatched Relationship — For understanding complex scene the context relationship is very important. For example, as shown in figure 1, a person on the poster of the tram is predicted in the middle of tress with the red color based on its appearance. There is a chance of misclassification if the collection of contextual information is less.

b) Confusion Categories — There are many pairs of class labels in dataset which are very confusing in classification. For example earth and field, skyscrapper and building appears similar. Even the operson who is expert in annotation still makes an error.



(a) Input Image **(b)** Segmented Output

Figure 1: Mismatched Relationships

c) Inconspicuous Classes — There are objects of different sizes present in the scene. Many objects which are hard to find because of there small size but it wil be of high importance. For the performance to improve the different sub regions should be given great importance as it contains inconspicuous stuff.

III. NETWORK ARCHITECTURE

1. Pyramid Scene Parsing Network Framework

The architecture of Pyramid Scene Parsing Network (PSPNet) considers image context which is global for local level predictions as a result it provides better performance on datasets like cityscapes, PASCAL etc. As pixel classifier based on FCN were failing to capture the context of whole image.

The PSPNet is the foundation of our semantic segmentation. To perform semantic segmentation, a Fully Convolutional Network (FCN) was used. There were many issues with this approach, including a lack of ability to infer from context, failure of label association through the relationship between categories, and the model's ability to overlook small items when exceeding the FCN acceptance range, resulting in discontinuous predictions. In conclusion, FCN does not effectively manage the relationships between situations and global data.

PSPNet is made up of four modules, as shown in Figure 2. It extracted the feature map using a pre-trained ResNet model with the dilated network strategy, provided an input in Figure 2(a). As shown in Figure 2(b), the final feature map is 1/8 the size of the input image. The most important pyramid pooling module for PSPNet is depicted in Figure 2(c). To collect background information, a 4-level pyramid was used, with pooling kernels covering the entire image, half of the image, and small parts of the image, which were fused as the global prior. Then, in the final part of 2(c) combined the previous with the original function map. After that, a convolution layer was used to build the final prediction map in 2(d). [4]

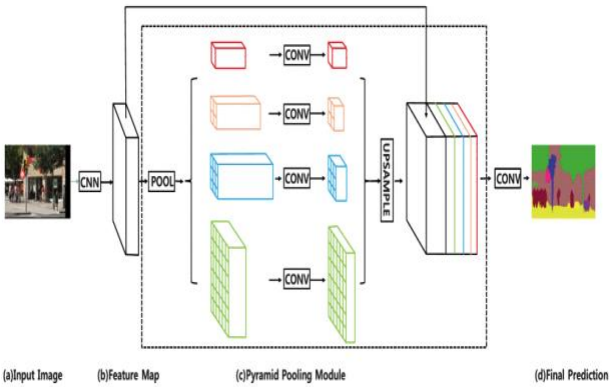


Figure 2: PSPNet Framework [4]

Method	mean IoU(%)
DPN [20]	66.8
DeepLabv2 [3]	70.4
Piecewise [14]	71.6
RefineNet [15]	73.6
DUC [30]	77.6
PSPNet [35]	78.4
PAN(ours)	78.6

Figure 3: Overview of Semantic Segmentation algorithms [10]

ResNet transmits a function map with a pyramid structure in, as shown in 2(a). In conventional networks, as the size of the network grows, it becomes more difficult to perform additional image classification optimization. The problem was solved by ResNet using skipped connections in each block. Deep ResNet's later layers also learned residues based on previous ones.

2. ResNet Model

ResNet is the backbone model for our semantic segmentation for image training, prediction, and testing. Hierarchical Learning in ResNet: The Forward Feature Learning During the training process of a residual network, the lower-level layers automatically learn an approximation of the lower-complexity features/signals in the target function. It then forwards these features to the higher-level layers in the network to further learn the higher-complexity features/signals in the target function. [13]

In contrast, in many practical tasks, neural networks give much better generalization error compared to kernels, although both methods can achieve zero training error. For example, ResNet achieves 96% test accuracy on the CIFAR-10 data set, but NTKs achieve 77% and random feature kernels achieve 85%. [14]

Method	mean IoU(%)	Pixel Acc.(%)
DFN(Res-101+RRB) [31]	76.65	–
Global Convolution Network [24]	77.50	–
ResNet101+GAU	77.84	94.96

Figure 4: Comparison with other state-of-art decoder module. [10]

2.1. ResNet-101 Model

The ResNet model was influenced by the VGG-19 model. It is one of ImageNet's most complex proposed architectures (Object detection and Image classification Challenge). Typically, in a CNN, multiple layers are linked and trained to perform different tasks. At the end of each layer, the network learns many levels of functionality. In this model, the convolutional layers are mostly 33 filters in size. The layers in ResNet have the same number of filters for the same output feature map size, and the number of filters is multiplied if the feature map size is halved to keep the time complexity of each layer constant. It performs direct down sampling by convolving layers with a two-step stride. A global average pooling layer and a SoftMax enabled fully connected layer complete this ResNet. Figure 5(a) shows the ResNet Module. Residual learning is simply the subtraction of learned input features from that layer.

ResNet does this by using shortcut connections to each pair of 33 filters, linking the input of the kth layer to the (k + x) th layer directly. The aim of bypassing layers is to avoid the problem of vanishing gradients by reusing activations from the previous layer before the layer next to it has learned its weights. Weights adjust to amplify the layer next to the current one when training the network, as well as to silence the layer before it. It has been discovered that this network is simpler to train than simple deep convolutional neural networks. It also addresses the issue of accuracy deterioration. ResNet-101 is a modified variant of the 50-layer ResNet and is a 101-layer Residual Network. [5]

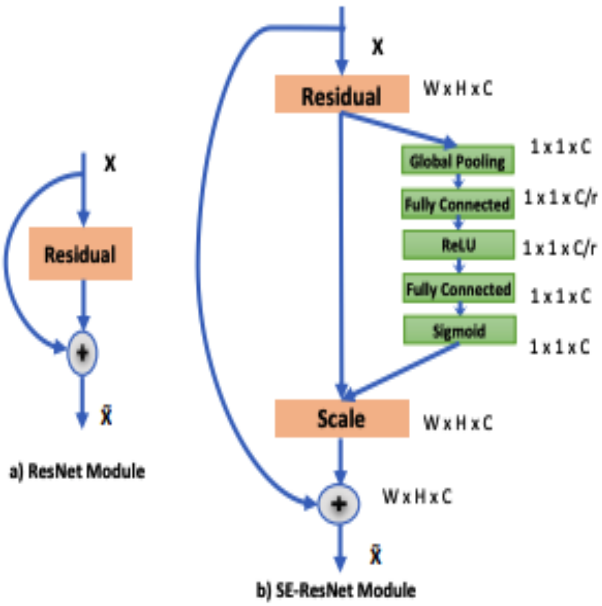


Figure 5: Block Diagram of ResNet Modules [5]

The employed structure for our classification task, ResNet, stands for Residual Networks and has an important point on computer vision problems. ResNet network uses residual connections which the gradients can flow directly through to inhibit the gradients to become zero after the applications of chain rule. ResNet-101 contains 104 convolutional layers in total. Along with, it consists of 33 blocks of layers in total and 29 of these blocks use previous block's output directly which is defined as residual connections above and these residuals are used as the first operand of summation operator used at the end of each block to get the input of the following blocks. The remaining 4 blocks take the previous block's output and use it in a convolution layer with a filter size of 1x1 and a stride of 1 followed by a batch normalization layer which performs normalization operation, and the resultant output is sent to the summation operator at the output of that block.

The important part of this model is pyramid pooling module which helps in capturing the global content of the image which in turn helps in classification of pixels based on global image available in the image. N feature maps after average pooling with n diferent sizes and by performing 1*1 covolutions the maps at all levels is reduced to N/n maps. For example if there are 1024 feature maps with size of 4 like global average pooling then at every level 1024 feature maps are reduced to 256 maps. [15]

The upsampling of N/n feature maps is done at every level to obtain input image of same dimensions. The base feature maps is concatenated with upsampled average pooled feature maps to generate the output of pyramid pooling module.

a) Input Image — The network is fed with input image which is of any shape.

b) Feature Map — Feature maps for image is constructed by taking the input image. The image is fed using scratch network or transfer leraning with convolutions which are delatd to extract the feature maps.

c) Pyramid Pooling Module — The objects which are present in different region and sizes which ranges from small area to large area is present in the image. The networks such as UNet, Fully Convolution Network (FCN) performs segmentation and upsampling at various levels for segmentation of each object in all regions to obtain feature maps. In PSPNet the feature maps are average pooled at various sizes of pools to accurately segment all size objects.

d) Final Prediction — The convolution layer is fed with 2*N feature maps and the classes obtained from final prediction is generated. Different objects with different channels or single channels are used to construct the output layer.

IV. IMPLEMENTATION

In this project we have used the Intel RealSense Depth Camera D435 to capture the images to which we performed labelling using a tool called LabelMe, which will be discussed in detail in this paper. After labelling we have used

Figure 6: RealSense Depth Camera D435 [7]

Anaconda Command Prompt in Visual Studio Code to train, predict and test the images for semantic segmentation

1. Intel RealSense Depth Camera D435

Depth is obtained primarily from solving the correspondence problem between the simultaneously recorded left and right video images, evaluating the disparity for each pixel (i.e., change between object points in left vs right images), and calculating the depth map from disparity and triangulation with Intel® RealSenseTM Depth Cameras D435 The current generation of depth algorithm in the RealSense D4 Stereo-vision ASIC is capable of picking up even the tiniest texture in a scene, particularly in bright environments, and thus performs exceptionally well outside. However, in some cases, optically superimposing texture may be beneficial, if not necessary. "Active" stereo vision relies on the addition of an optical projector that overlays the observed scene with a semi-random texture that aids in the discovery of correspondences, particularly on texture-less surfaces such as indoor dimly lit white walls. The location of these projectors on the D435 depth camera models is shown in Figure 6. The D435 is equipped with an AMS Princeton Optronics projector that has a wider emission angle but fewer spots (5000).

We used the Intel RealSense Depth Camera D435 to capture images which were fed into the semantic segmentation algorithm for training, predicting, and testing the images. The images were part of a dataset that include: Bottle, Box, Candle, Coffee Mug, Flowerpot, Foot ware, and Glass. These objects were captured individually and were captured as a set of multiple objects in one image for purposes of training and predicting the images to test whether the semantic segmentation can identify objects in an image. [7]

2. LabelMe

In the last decade, the availability of visual data has changed dramatically, especially thanks to the Internet, which has provided researchers with access to billions of images and videos. Even though vast amounts of images are available, creating a large data set of annotated images with many artifacts is still a costly and time-consuming task. Individual study organizations have traditionally created data sets that are designed to address problems. As a result, many existing

computer vision data sets only contain a limited number of object types, and only a few of them have accurate detectors

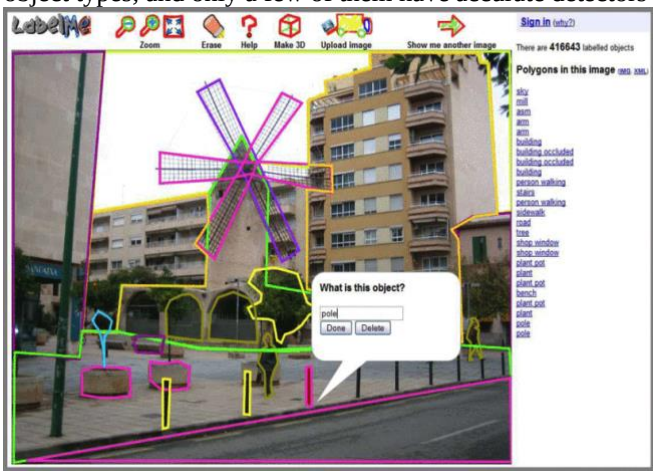


Figure 7: LabelMe for image annotation [9]

(e.g., human faces and cars). The Caltech 101 data set, which contains 101 object classes (later expanded to 256 object classes), ImageNet, the PASCAL collection, which contains 20 object classes, the CBCL-street scenes database, and the Lotus Hill Research Institute's database of scenes are notable recent exceptions. provides a description and discussion of existing data sets. LabelMe's mission is to provide a wide range of images with numerous annotated objects. In computer vision, available data sets have focused on obtaining annotated data for a collection of predefined object categories or collecting images of one or a few famous objects.

LabelMe is an online annotation platform for computer vision research that enables sharing and marking of photographs. The framework takes advantage of the web's ability to focus the efforts of a wide number of users. Since its launch in August 2005, the tool has amassed over 400 000 annotated items. Drawing polygons to outline the spatial extent of objects in images, querying for object annotations, and browsing the database are all features of the online tool (see Figure. 7). [9]

We used the LabelMe online tool to perform data annotation on the images captured by the depth camera. We drew polygons around objects in the images. Each image was marked by polygons of different colors to ensure uniqueness before this data was passed into the semantic segmentation algorithm for training, predicting, and testing the images

3. IoU (Intersection over Union)

The most widely used metric for comparing the resemblance of two arbitrary shapes is the IoU, also known as the Jaccard index. The region property of IoU encodes the shape properties of the objects under comparison, such as the widths, heights, and positions of two bounding boxes, and then calculates a normalized metric that focuses on their regions (or volumes). IoU is invariant to the size of the problem under consideration because of this property. All performance metrics used to assess segmentation, object detection, and tracking depend on this metric because of this appealing property. [12]

4. Accuracy Checking

Once we perform a semantic segmentation on the images, we compare them with the input images. There are three levels of accuracy produced where if an image is correctly segmented then it is true positive (TP), if image is not segmented but similar to input image, it is called false negative (FN) , if an image does not exist but if is segmented from the point cloud then it is called false positive (FP) . TP, FN, and FP indicate accurate segmentation, under-segmentation and over-segmentation. In order to evaluate the accuracy, we calculate recall (r), precision (p), and F1 score using the following equations:

$$r = \frac{TP}{TP + FN}$$

$$\mathrm{p} = \frac{TP}{TP+FP}$$

$$F = 2 \times \frac{r \times p}{r + p}$$

The values r, p, and F1 range from 0 to 1. If we want to obtain a higher F1 score then r and p should be very high. For example, if all the labelled images are correctly segmented then the values of r and p are one resulting in F1 equal to one. [11]

V. CODE ANALYSIS

The main coding language used to implement this project on semantic segmentation is Python. We implemented the code on Visual Studio code via anaconda prompt which allows us to compile and run the code using this command line.

This project has the following dependencies:

- Numpy version >= 1.1: *sudo pip install numpy*
- OpenCV Python: *sudo apt-get install python-opencv*
- TensorFlow version >= 3: *sudo pip install tensorflow-gpu*

The dataset for implementation will be as shown in figure 8. The folders ‘testing’, ‘training’ and ‘validation’ should have images captured from the Intel RealSense Depth Camera D435 as a .jpg format. The folders ‘testing_labels’, ‘training_labels’ and ‘validation_labels’ should have the labelled images which we have after labelling them on the online tool ‘LabelMe’ as a .png format. The image dimensions are taken to be 640x480.

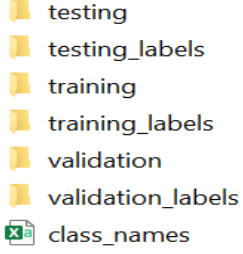


Figure 8: Input folders for Dataset

name,r,g,b
Bottle,128,0,0
Box,0,128,0
Candle,128,128,0
Coffee_Mug,0,0,128
Flowerpot,128,0,128
Footwear,0,128,128
Glass,128,128,128
Void,0,0,0

Figure 9: Class Names with RGB Values

```
import subprocess
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow.contrib.slim as slim
import argparse
import random
import os,time,cv2, sys, math
```

Figure 10: Libraries

The class_names.xlsx file holds the names of all the classes we are using in this project along with its RGB value. These RGB values are obtained after data annotation.

1. Libraries

The libraries we have used to implement our code are shown as in figure 10. We are importing the library 'matplotlib' as it is a plotting library used for python programming language and is useful in implementing numpy (Python's numerical mathematics extension)

Numpy is a python programming language library. It's used to help with the implementation of large, multi-dimensional arrays and matrices, as well as a large collection of high-level mathematical functions that work with the arrays and matrices.

2. Training and Validation

The following code as shown in Table 2, is the parameters that we have set to train the augmented images. In our project we have used the model as PSPNet and ResNet 101 as the backbone model. It is also possible to use ResNet-50 and RestNet-152 as the backbone model based on the number of images that are being trained. We are training a set of 100 images, for both training and validation.

Even before we start the training process, we need to download the backbone model ResNet-101 from utils\Download_PreTrained_Checkpoints.py as shown in Table 1. The backbone model should be saved under the folder 'models'.

TABLE 1: ResNet-101 download link

```
if args.model == "ResNet50" or args.model == "A
LL":
    subprocess.check_output(['wget','http://dow
nload.tensorflow.org/models/resnet_v2_50_2017_0
4_14.tar.gz', "-P", "models"])
    try:
        subprocess.check_output(['tar', '-
xvf', 'models/resnet_v2_50_2017_04_14.tar.gz',
"-C", "models"])
        subprocess.check_output(['rm', 'models/
resnet_v2_50_2017_04_14.tar.gz'])
```

TABLE 2: Parameters

```
parser.add_argument('--
num_epochs', type=int, default=50)
parser.add_argument('--
epoch_start_i', type=int, default=0)
parser.add_argument('--
checkpoint_step', type=int, default=5)
parser.add_argument('--
validation_step', type=int, default=1)
parser.add_argument('--
image', type=str, default=None)
parser.add_argument('--
continue_training', type=str2bool, default=False)
parser.add_argument('--
dataset', type=str, default="MyAISProjectImages")
parser.add_argument('--
crop_height', type=int, default=480)
parser.add_argument('--
crop_width', type=int, default=640)
parser.add_argument('--
batch_size', type=int, default=1)
parser.add_argument('--
num_val_images', type=int, default=15)
parser.add_argument('--
h_flip', type=str2bool, default=False)
parser.add_argument('--
v_flip', type=str2bool, default=False)
parser.add_argument('--
brightness', type=float, default=None)
parser.add_argument('--
rotation', type=float, default=None)
parser.add_argument('--
model', type=str, default="PSPNet")
parser.add_argument('--
frontend', type=str, default="ResNet101")
```

We are also applying data augmentation on our images, which is used to perform a random horizontal and vertical flip, adjusts the brightness, and rotate the images with the code as shown in Table 4. To train our images we are applying a learning rate of 0.01 with decay = 0.995. An important factor to be noted here is that the learning rate can vary for different images and it needs to be set accordingly. We are also using batch size = 1.

Even though we are training 100 images we get the validation output for around 20 images. The number of output images can be altered by varying 'num_val_images' in the parameter section shown in Table 2.

The code in Table 3 is used to convert the images into RGB values.

TABLE 3: Conversion of image to RGB values

```
def load_image(path):
    image = cv2.cvtColor(cv2.imread(path,-
1), cv2.COLOR_BGR2RGB)
    return image
```

TABLE 4: Image Augmentation

```
def data_augmentation(input_image, output_image):
    input_image, output_image = utils.random_crop(
input_image, output_image, args.crop_height, args.
crop_width)
    if args.h_flip and random.randint(0,1):
        input_image = cv2.flip(input_image, 1)
        output_image = cv2.flip(output_image, 1)
    if args.v_flip and random.randint(0,1):
        input_image = cv2.flip(input_image, 0)
        output_image = cv2.flip(output_image, 0)
    if args.brightness:
        factor = 1.0 + random.uniform(-
1.0*args.brightness, args.brightness)
        table = np.array([((i / 255.0) * factor) *
 255 for i in np.arange(0, 256)]).astype(np.uint8)
  #
        input_image = cv2.LUT(input_image, table)
    if args.rotation:
        angle = random.uniform(-
1*args.rotation, args.rotation)
    if args.rotation:
        M = cv2.getRotationMatrix2D((input_image.s
hape[1]//2, input_image.shape[0]//2), angle, 1.0)
        input_image = cv2.warpAffine(input_image,
M, (input_image.shape[1], input_image.shape[0]), f
lags=cv2.INTER_NEAREST)
        output_image = cv2.warpAffine(output_image
, M, (output_image.shape[1], output_image.shape[0]
), flags=cv2.INTER_NEAREST)
    return input_image, output_image
```

a) *Running train.py:*

TABLE 5: Command line to run train.py

```
(tf)
C:\Users\Desktop\AISProject\SegmanticSegmentati
onC1>python train.py
```

We can run train.py using the command as shown in Table 5. Once we start training the images, the checkpoints will be saved under the folder .\Checkpoints along with predicted images as shown in figure 11.

3. Prediction

Once training is completed, all the validated images are stored under the Checkpoints folder. Now we can use the newly trained model to predict single or multiple images. This step will tell us how well the model has been trained during training/validation phase. This code picks up the validated image, predicts it and then stores the newly predicted image.

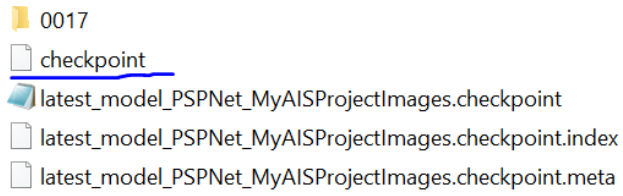


Figure 11: Checkpoint folder

a) *Running predict.py*

TABLE 6: Command line to run predict.py

```
(tf)
C:\Users\Desktop\AISProject\SegmanticSegmentation
C1>python predict.py --Image 0001.png --
checkpoint_path.\latest_model_PSPNet_MyAISProject
Images.checkpoint --model PSPNet
```

4. Testing

After we acquire our trained and validated data, we have to perform testing. In this phase, the code fetches all the validated images from the latest checkpoint and tests it.

TABLE 7: TestData Inputs

```
avg_score = np.mean(scores_list)
class_avg_scores = np.mean(class_scores_list, axi
s=0)
avg_precision = np.mean(precision_list)
avg_recall = np.mean(recall_list)
avg_f1 = np.mean(f1_list)
avg_iou = np.mean(iou_list)
avg_time = np.mean(run_times_list)
```

Upon execution of the code shown in Table 7, it will generate an excel file containing the values of accuracy, precision, mean value IoU, F1 score, etc.

a) *Running test.py*

TABLE 8: Command line to run test.py

```
(tf)
C:\Users\Desktop\AISProject\SegmanticSegmentati
onC1>python test.py --
checkpoint_path.\latest_model_PSPNet_MyAISProje
ctImages.checkpoint --model PSPNet
```

VI. RESULTS

1. Training and Validation results

We are training and validating 100 images using PSPNet model for indoor images. When we first ran train.py, we got the output window as shown in figure 12.

```
The PSPNet model has 56000328 parameters to be trained
Loading the latest fetched data!!!!!! ...

*****##Begin the training process*****
Dataset --> MyAISProjectImages
Model --> PSPNet
Crop Height --> 480
Crop Width --> 640
Number of Epochs --> 80
Batch Size --> 1
Number of Classes --> 8
Data Augmentation:
        Vertical Flip --> False
        Horizontal Flip --> False
        Brightness Alteration --> None
        Rotation --> None
```

Figure 12: Output window showing parameters

```
Average validated accuracy for epoch #0012 = 0.962434
Average class(per class) validated accuracy for epoch #0012:
Bottle = 0.884686
Box = 0.866667
Candle = 0.934081
Coffee_Mug = 0.533927
Flowerpot = 0.870881
Footwear = 0.881512
Glass = 0.795087
Void = 0.998988
Obtained Validation precision =  0.9739574115396455
Obtained Validation recall =  0.9624342447916666
Obtained Value for F1 score =  0.9674673272699913
Obtained Validation Value for IoU =  0.5916861497226441
```

Figure 13: Validated accuracy captured at epoch #0012

```
[2021-05-14 11:55:54] You are at : Epoch = 5 Count = 20 LossOfImageValue = 0.4100 Time = 137.35
[2021-05-14 11:58:07] You are at : Epoch = 5 Count = 40 LossOfImageValue = 0.0818 Time = 132.97
[2021-05-14 12:00:19] You are at : Epoch = 5 Count = 60 LossOfImageValue = 0.0760 Time = 132.35
[2021-05-14 12:02:29] You are at : Epoch = 5 Count = 80 LossOfImageValue = 0.1606 Time = 130.38
[2021-05-14 12:04:42] You are at : Epoch = 5 Count = 100 LossOfImageValue = 0.0672 Time = 132.30
Saving the latest available Checkpoint to your local directory
Saving checkpoint for this epoch
Validation is in process
```

Figure 14: Time taken to train images in epoch 5

We are training our images for 50 epochs, with learning rate of 0.01. The latest checkpoints are saved after 5 steps. In order to monitor our image training process, we get the list of validated accuracy for every class after each epoch as shown in figure 13.

Once the training is completed after every epoch, we can also monitor the time taken to train our images along with loss as shown in figure 14. Important factor to be noted here is that the loss of the image value should decrease over time.

We can see in figure 15 that, 2 of the images we had captured from the depth camera have been trained and validated according to the labelled image. In the class_names.xlsx file, we had given the RGB values 128,0,0 for class 'Bottle' and RGB values 0,128,128 for class 'Footwear'. Based on the provided RGB values the algorithm has trained and validated the respective classes to their RGB values and trained the indoor images to identify a bottle to be 'Red' (128,0,0) and footwear to be (0,128,128), etc. Each of the indoor images (classes) has been assigned its own RGB values before training.

Figure 15: Raw input image, validated image, labelled image

TestData_Name	TestData_Accuracy	Precision	Recall	F1 Score	MeanValue IoU	Bottle	Box	Candle	Coffee_Mug	Flowerpot	Footwear	Glass	Void
Image16_Color	0.84696	1	0.84696	0.917139	0.42348	1	0	1	1	1	1	1	1
Image17_Color	0.948844	0.953472	0.948844	0.951152	0.497226	1	0	1	1	1	1	1	0.99965
Image18_Color	0.850671	1	0.850671	0.919311	0.425335	1	0	1	1	1	1	1	1
Image2_Color	0.961146	0.958312	0.961146	0.959727	0.496541	0	1	1	1	1	1	1	0.995059
Image56_Color	0.952334	0.955734	0.952334	0.954031	0.496683	1	1	1	1	1	0	1	0.998451

Figure 16: TestData score

2. Testing results

When we run the testing results, we get the TestData score in the form of an excel sheet which shows the average accuracy of each class, average precision, average recall, average F1 score, and average mean IoU value.

As we can see in figure 16, after the testing process is completed, the class that has binary value of '0', is the object class that is present in the image.

3. Statistical results

During out learning process we can notice from the statistical graphs that, average IoU and average accuracy fluctuate over time, but average loss seems to fairly improve. If we train for 100 or more epochs with over 1000 images, the average loss should decrease over time.

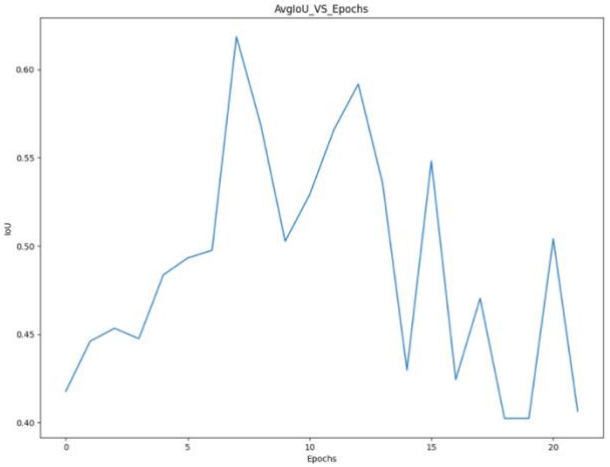


Figure 17: Average IoU vs Epochs. Results for Epoch 25

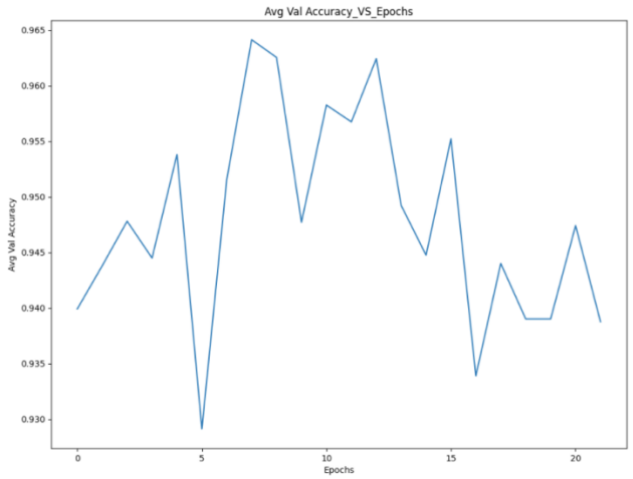


Figure 18: Average accuracy value vs Epochs. Results for Epoch 25

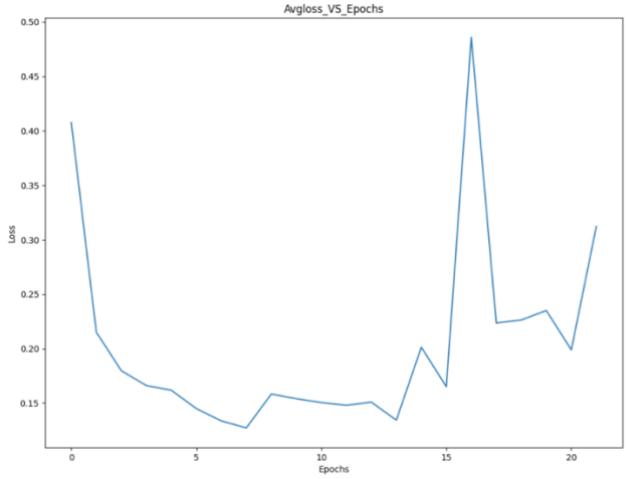


Figure 19: Average loss vs Epochs. Results for Epoch 25

VII. GITHUB LINK

Link to project code and output:
https://github.com/PrasadSahana/Computer-Vision---Segmantic-Segmentation-using-PSPNet.git

VIII. DROPBOX LINK

Link for images -
https://www.dropbox.com/sh/kugpwiiippsh19t/AAAZrDvgWJUeygsoEUB7Bjdma?dl=0

Link for Pre-Trained model (ResNet-101) -
https://www.dropbox.com/s/fvfvb3yt8k80nje/resnet_v2_101.checkpoint?dl=0

The drop box link also has checkpoint folders that can be referred to.

IX. CONCLUSION

In this project, we have successfully implemented Semantic Segmentation using PSPNet algorithm. An analysis of the different approaches for semantic segmentation was discussed. One of the approaches we found suitable for our project is to implement PSPNet using Pre-trained ResNet 101 model. UNet could also be used for semantic segmentation but as we were training indoor images and were more concerned about object differentiation, we chose this algorithm. LabelMe, Python-OpenCV, Tensorflow and numpy functions played an important role in semantic segmentation.

The raw images were captured using depth camera and were masked using online tool labelme which was then trained, validated, and tested. In the end we were able to attain accuracy of over 90% which is good but can be improved over time.

Our future aim is to expand the project by taking larger dataset for training of indoor images. We also plan to train, validate, and test our indoor images using other recommended algorithms as this could be a part of future research.

ACKNOWLEDGMENT

We would like to thank Prof. Dr. Peter Nauth and Prof. Sudeep Sharan for giving us an opportunity to closely work with them on this topic. We would also like to thank Prof. Julian Umansky and Tutor Hoang Long Nguyen for all the guidance and continuous support throughout this project.

REFERENCES

[1] J. Shotton, M. Johnson, and R. Cipolla. “Semantic texton forests for image categorization and segmentation”. In: 2008 IEEE Conference on Computer Vision and Pattern Recognition. June 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587503.

[2] J. Xiao and L. Quan. “Multiple view semantic segmentation for street view images”. In: 2009 IEEE 12th International Conference on Computer Vision. Sept. 2009, pp. 686–693. DOI: 10.1109 / ICCV. 2009.5459249.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In ECCV, 2014.

[4] X. Long, W. Zhang and B. Zhao, "PSPNet-SLAM: A Semantic SLAM Detect Dynamic Object by Pyramid Scene Parsing Network," in IEEE Access, vol. 8, pp. 214685-214695, 2020, doi: 10.1109/ACCESS.2020.3041038.

[5] P. Ghosal, L. Nandanwar, S. Kanchan, A. Bhadra, J. Chakraborty and D. Nandi, "Brain Tumor Classification Using ResNet-101 Based Squeeze and Excitation Deep Neural Network," 2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP), 2019, pp. 1-6, doi: 10.1109/ICACCP.2019.8882973.

[6] I. Culjak, D. Abram, T. Pribanic, H. Dzapo and M. Cifrek, "A brief introduction to OpenCV," 2012

Proceedings of the 35th International Convention MIPRO, 2012, pp. 1725-1730.

[7] Grunnet-Jepsen, A., Sweetser, J. N., Winer, P., Takagi, A., & Woodfill, J. (2018). Projectors for Intel® RealSense™ Depth Cameras D4xx. Intel Support, Interl Corporation: Santa Clara, CA, USA.

[8] Pang, B., Nijkamp, E., & Wu, Y. N. (2020). Deep learning with tensorflow: A review. Journal of Educational and Behavioral Statistics, 45(2), 227-248.

[9] A. Torralba, B. C. Russell and J. Yuen, "LabelMe: Online Image Annotation and Applications," in Proceedings of the IEEE, vol. 98, no. 8, pp. 1467-1484, Aug. 2010, doi: 10.1109/JPROC.2010.2050290.

[10] Li, H., Xiong, P., An, J., & Wang, L. (2018). Pyramid attention network for semantic segmentation. arXiv preprint arXiv:1805.10180.

[11] Li, W., Guo, Q., Jakubowski, M. K., & Kelly, M. (2012). A new method for segmenting individual trees from the lidar point cloud. Photogrammetric Engineering & Remote Sensing, 78(1), 75-84.

[12] Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 658-666).

[13] Allen-Zhu, Z., & Li, Y. (2019). What can resnet learn efficiently, going beyond kernels?. arXiv preprint arXiv:1905.10337.

[14] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do CIFAR-10 Classifiers Generalize to CIFAR-10? arXiv preprint arXiv:1806.00451, 2018.

[15] A. Demir, F. Yilmaz and O. Kose, "Early detection of skin cancer using deep learning architectures: resnet-101 and inception-v3," 2019 Medical Technologies Congress (TIPTEKNO), 2019, pp. 1-4, doi: 10.1109/TIPTEKNO47231.2019.8972045.

[16] https://github.com/GeorgeSeif/Semantic-Segmentation-Suite

[17] https://github.com/wkentaro/labelme