UNIVERSITY OF PADOVA

Department of Information Engineering

Master of Science in ICT for Internet and Multimedia *Cybersystems*

Master Thesis

Deep Networks and Random Forests for Semantic Segmentation on 3D Data

Advisor:
Dr. Stefano Ghidoni

Co-Advisor:
Matteo Terreran, M.Sc.

Candidate:
Elia Bonetto

Academic Year 2018–2019

To those who have made all this possible.
That always supported me.
Helped me.
And more importantly: endured me.

Draco Dormiens Nunquam Titillandus
J.K.R.

Abstract

Object recognition and semantic segmentation have always been major topics in the computer vision community. The focus of object recognition is to find objects in the scene while semantic segmentation in general does not provide a simple bounding box around the objects but rather, seeks a pixel-wise accuracy.

Historically this kind of processing has been performed on 2D data: the classical 3-channels (RGB) images, reaching in some cases astonishing results. On the other hand the popularity of 3D-data elaboration has grown in the recent years thanks to the advances in the processing power and the availability of lower priced sensors.

This work will focus on semantic segmentation over 3D data, firstly by the research and the study of the state of the art, then by developing methods to transfer features between *FuseNet* [1], a convolutional auto-encoder, and *3D Entangled Forests* (3DEF) [2]. We were able to obtain meaningful insights with regards to the dataset and used learning models, pointing out a possible saturation of the performance due to their combination. Moreover, we successfully demonstrated the usefulness of transferring features and their usability in the field of semantic segmentation thanks to an performance improvement obtained with one of our proposed FuseNet networks modifications.

Contents

Abstract III

List of Figures VI

List of Tables VII

Listings IX

1 Summary 1

2 On 3D Semantic Segmentation 3
- 2.1 History . . . 3
- 2.2 Comparing 2D and 3D data . . . 5
- 2.3 3D Problems . . . 7
- 2.4 State of the Art . . . 9
  - 2.4.1 Datasets for Semantic Segmentation . . . 12
  - 2.4.2 Metrics . . . 13
- 2.5 Our Contributions . . . 14

3 FuseNet 15
- 3.1 The Original Network . . . 15
- 3.2 The COROMA RGB-D Dataset . . . 21
  - 3.2.1 Fine-Tuning . . . 22

4 3D Entangled Forests 24

5 From 3DEF to FuseNet 28
- 5.1 Input . . . 28

5.1.1 The Data . . . 28
5.1.2 The Features . . . 28
5.1.3 Generated Features Set . . . 33
5.2 Proposed Networks . . . 34
5.2.1 Sparse Fusion . . . 34
5.2.2 Only Encoding . . . 38
5.2.3 A Parallel Autoencoder . . . 42
5.2.4 Direct Fusion . . . 46
5.3 Reducing the Size of the Networks . . . 50

6 From FuseNet to 3DEF 53
6.1 Input . . . 53
6.1.1 The Data . . . 53
6.1.2 The Features . . . 53
6.2 Training and Results . . . 55
6.2.1 13 Classes . . . 55
6.2.2 40 Classes . . . 57
6.3 Shorten Trees Depth . . . 62

7 Conclusions 64

References 66

List of Figures

2.1 Classification and Detection for a group of people 4
2.2 Classification, Detection and Segmentation 4
2.3 Different representation of the same image 7
2.4 Same scene in 2D, depth and pointcloud views 9
2.5 NYUv2's depth comparison 12

3.1 SegNet network 16
3.2 Original FuseNet architecture 16
3.3 A single Fusion block 17
3.4 Sparse and Dense Fusion comparison 17

5.1 Camera Projection using the pin hole camera model 30
5.2 Examples of imported features from 3DEF to FuseNet 31
5.3 Embed 3DEF' features in FuseNet with Sparse Fusion 35
5.4 Embed 3DEF' features in FuseNet by Only Encoding them 38
5.5 Use of a Parallel Autoencoder to fuse 3DEF' features in FuseNet . . . 42
5.6 Direct Fusion of 3DEF' features in FuseNet 46

6.1 Features Importance for each depth level for the whole set of 40 trees, 13 classes trained with V4 features from FuseNet 63

List of Tables

2.1 Sample confusion matrix: actual vs predicted class. 13

3.1 FuseNet trained over NYUv2, 13-classes mapping, general results . . 19
3.2 FuseNet trained over NYUv2, 13-classes mapping, classwise accuracy 19
3.3 FuseNet trained over NYUv2, 40-classes mapping, general results . . 19
3.4 FuseNet trained over NYUv2, 40-classes mapping, classwise accuracy 20
3.5 Training results of FuseNet over reduced COROMA dataset. 1000 epochs. 21
3.6 Fine Tuning of FuseNet over reduced COROMA dataset. In the columns the reset depth, in the rows the allowed learnable parameters. Trained over elaborated depths. 22

4.1 Unary features and their dimension 26
4.2 Entangled coefficients and their description 27

5.1 Description and Dimension of our obtained 3DEF Entangled Features 33
5.2 Obtained features set with their shorter name and dimension 34
5.3 13-classes, Sparse Fusion network, general results 36
5.4 13-classes, Sparse Fusion network, classwise accuracy 36
5.5 40-classes, Sparse Fusion network, general results 36
5.6 40-classes, Sparse Fusion network, classwise accuracy 37
5.7 13-classes, Only Encoding network, general results 39
5.8 13-classes, Only Encoding network, classwise accuracy 39
5.9 40-classes, Only Encoding network, general results 40
5.10 40-classes, Only Encoding network, classwise accuracy 41
5.11 13-classes, Parallel AE network, general results 43
5.12 13-classes, Parallel AE network, classwise accuracy 43

5.13 40-classes, Parallel AE network, general results 44
5.14 40-classes, Parallel AE network, classwise accuracy 45
5.15 13-classes, Direct Fusion network, general results 47
5.16 13-classes, Direct Fusion network, classwise accuracy 48
5.17 40-classes, Direct Fusion network, general results 48
5.18 40-classes, Direct Fusion network, classwise accuracy 49
5.19 Number of parameters (in M) for original and reduced networks. Sample epoch time in seconds using a NVIDIA® Tesla T4. 51
5.20 13-class, training with reduced networks, general results 52
5.21 40-class, training with reduced networks, general results 52

6.1 13-classes, depth elaborated, 3DEF trained with and without features from FuseNet, general results 56
6.2 13-classes, depth elaborated, 3DEF trained with and without features from FuseNet, classwise accuracy 56
6.3 13-classes, depth raw, 3DEF trained with and without features from FuseNet, general results 57
6.4 13-classes, raw depth, 3DEF trained with and without features from FuseNet, classwise accuracy 58
6.5 40-classes, elaborated depth, 3DEF trained with and without features from FuseNet, general results 58
6.6 40-classes, raw depth, 3DEF trained with and without features from FuseNet, general results 59
6.7 40-classes, elaborated depth, 3DEF trained with and without features from FuseNet, classwise accuracy 60
6.8 40-classes, raw depth, 3DEF trained with and without features from FuseNet, classwise accuracy 61
6.9 Forests trained with 40 trees at depth 8, 13-classes, general results . . 62

Listings

5.1 Pointcloud Projection 29
6.1 Final Block of FuseNet 54

Chapter 1

Summary

This work regarding 3D semantic segmentation has been developed during an internship at the Intelligent and Autonomous Systems Lab (IAS-Lab) at the University of Padova under the supervision of Dr. Stefano Ghidoni, professor at the same University, and with the constant collaboration of the Ph.D. student Matteo Terreran.

The project consists in studying the state of the art in the field of semantic segmentation and to transfer feature vectors between a convolutional neural network and a random forest algorithm in order to increase their performance. More specifically, the work was to bring the deep features extracted by FuseNet, a convolutional autoencoder, to 3D Entangled Forests (3DEF), and vice-versa, adapting what have been done so far to our new requirements.

The final purpose of this work is to find out if useful information about 3D data, that may be embedded within these features, can be used from the other considered learning algorithm. This either to increase its performance or to lower the computational requirements necessary to achieve them and, even more hopefully, to achieve both of these results together.

The remainder of this work is divided as follows: in Chapter 2 a deep study of the literature is presented regarding mainly semantic segmentation, 3D-data challenges and available learning methods and datasets to perform the desired task. This is necessary to have the theoretical knowledge to perform the subsequent steps of the work.

In Chapters 3 and 4 there is a description of the two algorithms considered, respectively FuseNet and 3DEF, with details about their strengths, weaknesses and the peculiarities that will be useful to understand the work that have been performed. Moreover, in Chapter 3 regarding FuseNet, there will be a description of some experiments regarding training and finetuning made with an in-house available dataset of an industrial site that helped to better understand the model.

The development phase and the obtained results are deeply described in Chapters 5 and 6. Chapter 5 is about FuseNet and how information from 3DEF can be brought inside this network, meanwhile Chapter 6 treats about how features from FuseNet can be brought in 3DEF. All the challenges and solutions that I have found are explained and explored in these two chapters.

Finally in Chapter 7 there are some considerations about the performed work and possible future advances with regards to this problem and the obtained results.

Chapter 2

On 3D Semantic Segmentation

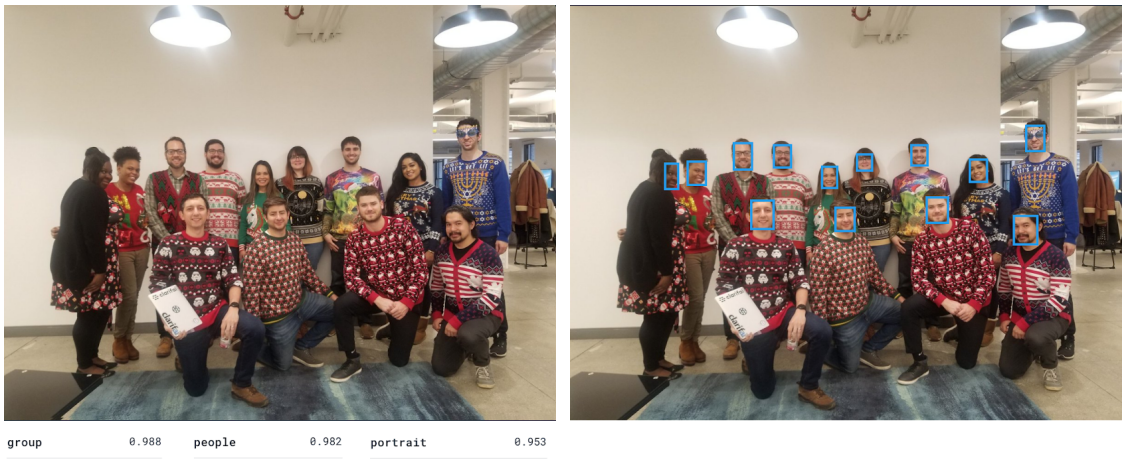
2.1 History

Vision is one of our core senses without which we could not easily interact with the world that surrounds us. We use depth estimation and experience everyday to do what seems to us basic things from driving to walk up the stairs. These may appear to be simple tasks to us but, actually, they are a very articulated merging of acts and sensing feelings that our brain manage to bring together to perform the job. Computers are still far from reaching our level of comprehension and elaboration of their surroundings especially on recognizing what there is around them. They can see but they still do not fully understand.

The first step in image understanding is classification which regards what is represented and what is inside the image as a general point of view. The core concept is to know if the image may depict a given concept that can be for example *sunset*, *beach* or if there is a *group* of people, in order to understand the scene that is represented. [3]

A more difficult task than classification is the detection of distinct objects that may be inside an image, that in the computer vision community has been a major topic since years ago. [4] Understanding the content of an image and getting an insight on where the objects are is clearly an important step toward image comprehension. Object detection consists in finding a bounding box around the target(s), so to depict a (usually) rectangular shape that will contain them at its best, and

it differs to image understanding in the way that here the scope is to know exactly which and where each separate objects are and no longer only *"concepts"*. To have a concrete example where classification may identify a *group*, detection will identify every single *face*, as in Fig. 2.1.



(a) Classification (b) Detection

Figure 2.1: Classification and Detection for a group of people [37]

More recently this has evolved into semantic segmentation. This is a development of the detection problem and is a further step from the classification one. In fact, it regards the scanning of each point of the image and set a label about which object is represented pixel by pixel so we can obtain, for example, what is depicted in Fig. 2.2. [5]

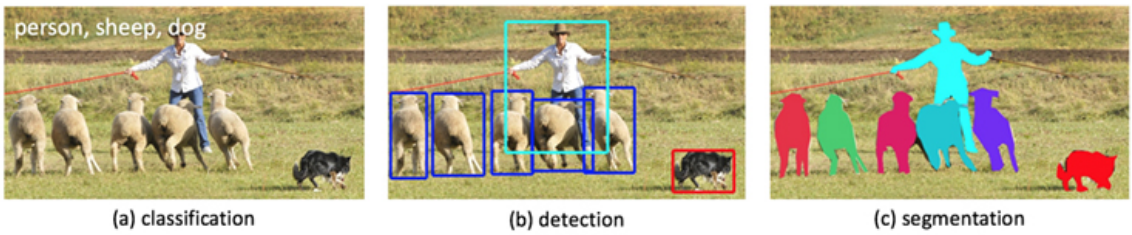


Figure 2.2: Classification, Detection and Segmentation [38]

Here there are no bounding boxes or concepts but more "simply" separated areas that will be detected in the image regarding each single target. The per-pixel basis is important since it lets us to find exactly where the object is and the shape it has

on our image even if partial occlusions happen. Occlusions and similarities between colours and shapes are the main obstacles, that coupled with the per-pixel labelling request make this problem very difficult.

All these techniques are related, since they descend one from the other, and help in image evaluation and understanding. Moreover, they are linked to what human beings can do every single moment of their lives in a blink of an eye to understand their surroundings and interact with them without even noticing the powerful process that is happening behind the scene.

Teaching to computers to do these tasks, especially semantic segmentation, can bring them to have near-human capacities in the elaboration and understanding of images: this would enable for example an automatic organization of our pictures library by recognizing people or objects to filter them out. Furthermore, these capabilities could be embedded in robots to enable interaction with the environment obtaining more powerful obstacle avoidance or task execution as picking an object from a table and placing it into a bin. Simply, we cannot pretend to have a robot that grasps a glass if it cannot identify it and where it is located.

2.2 Comparing 2D and 3D data

Historically, the majority of the works that have been developed to perform the task of semantic segmentation have been done over what we are used to see everyday: two dimensional plain images. These are essentially 2D matrices where for each point we have a single color, usually expressed by a triplet of values in the Red-Green-Blue (RGB) space.

There are many techniques that may be used to perform semantic segmentation, from clustering to edge detection or region growing. More recently machine and deep learning techniques, thanks to their performances, are increasing in popularity and have seen a rapid growth in the latest years. These have been applied to various tasks within the computer vision field including for example classifications but also in the semantic segmentation fields.

Deep learning essentially consists in a stack of layers, be them convolutional or recurrent, activation functions, regularization and pooling modules organized in

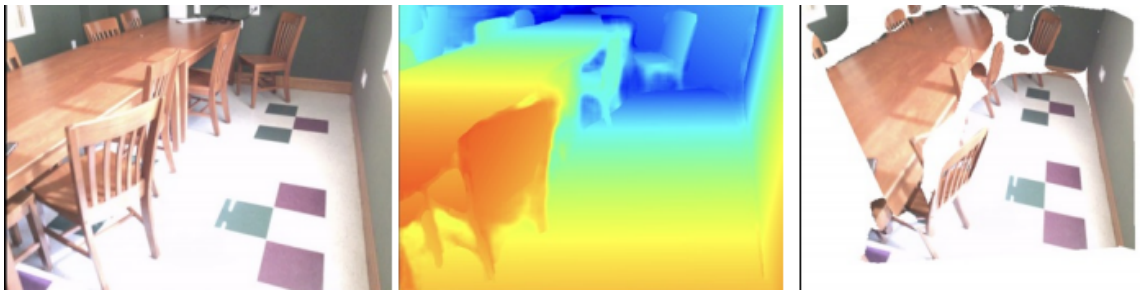
various ways that may generate different models. These models, in the latest years, have become bigger and more complex leading to an increasing of the number of parameters that have to be learned to perform the training of the networks mainly thanks to the advances in the technology of recent GPUs. The increasing of the model size corresponds to a proportional increase of the power and time consumption that are necessary to perform the learning task: this is the cost to obtain, in some cases, better overall performance like what happens with DAG-RNN that achieves 91.6% as accuracy over CamVid [5], a dataset used in autonomous driving.

More recently a challenge is brought by the wish to use images that are no longer mere 2D plain snapshot of the world but whole 3D representation that are rising in popularity in the last few years. This is the way we, as humans, see every object: we have depth acknowledgment from various sources other than stereo-vision like motion cues or relative size, almost always related to previous experience and gained knowledge. This kind of data has been available for a while now but did not have a wide spread due to its acquisitions costs. Only after the introduction of low cost devices, like the Kinect®, 3D representation reached a wide spread enabling more research possibilities especially in indoor scenarios.

3D data is mainly available as a pointcloud structure so as an unorganized and highly irregular structure consisting of triplet of coordinates, X, Y, Z, which have a color value attached to them.

Another possibility is RGB-D where each 2D image is paired with a corresponding depth map: a 2D structure in which each point contains the value of the depth of the specific point of the RGB image. RGB-D structure in the literature can be also referred as 2.5D since it is not "really" 3D: it represents the same data and it is easy to go from this representation to the relative pointcloud (as we will see in Chapter 5) but it is a couple of 2D images rather than a real 3D. As can be seen in Fig. 2.3 the three representations are surely different and related together, in fact to obtain a 2.5 dimension we couple the first two images and we can project these in the 3D space obtaining the third one.

This kind of data is of interest nowadays because it has the depth information embedded within it and, especially in robotics, that may be of use for some applications like robot grasping or environment interaction. The adoption of 3D data could



(a) 2D (b) Depth map (c) 3D version

Figure 2.3: Different representation of the same image

narrow the gap between human and machine perception. Clearly, a plain 2D world is much different from a real representation of the environment: for example, a computer cannot know by its own that the mountains in the landscape are kilometers away with respect to the car in the foreground, a thing that we grasp in less than a second by watching a given scene. We, as humans, are using the sense, experience and prediction of depth in all of our tasks; without that we would probably crash against walls, doors, other people the most of our time. It is an important part of our vision system and not a simple or obvious one: just think about babies that crash against walls or things with their head all the time because they have not experience about depth and dimensions and they still cannot elaborate correctly the information that they receive. By doing elaboration of 3D images we want to narrow this gap to enable new applications and improve overall capabilities.

2.3 3D Problems

Many problems are related to this kind of data and we have to learn how to treat this new information. For example, in the case of the pointclouds, we cannot directly use previously established methods since points are not ordered, they are sparse and may be far away one from another. In fact, unlike 2D images where usually the points are ordered, in pointclouds three points may be sorted and placed anywhere in the space. With 2D by following a row-by-column rule when we have the maximum number of columns and a coherent list of RGB values we can fully reconstruct the

image; instead with 3D the only way to understand exactly where they are is to know *all* the three components, *X*-*Y*-*Z*, of their location. With 3D we do not have ordering so we can't say for example "the fifth point on the second row" as we can do with 2D images because with 3D we do not have bounds or other ways to tell where the points will be located in the space.

This in some sense break the concept of nearness of the points since we no longer have a pre-defined regular neighbour of the point where we can extract information: in 2D images given a pixel we know for sure that it has three (if it is a corner), five (if it is along a border) or eight adjacent neighbours and this is true for every image. In 3D pointclouds not only we need to know where the point is but to find neighbours we have to define also a distance with which we have to search them and, moreover, we may or may not find these neighbours depending on the considered cloud and specific point. As can be seen clearly in Fig. 2.4, the 2D, depth and pointcloud representations are pretty different one from the other: if in the RGB image two points may be near, pixel-by-pixel, this may not be the case once we take in consideration the 3D reconstruction of the scene. Moreover, again in Fig. 2.4 we can appreciate the occlusions and holes that occur when dealing with pointclouds, their scatterings and unclear bounds along with the variable point of view from which we can see them: if we imagine to change it clearly the 2D image will keep its representation but what we see in the pointcloud change and we can loose sight of some details.

What we have just seen are only some examples of problems that are introduced by this kind of data and that inhibit us to adopt previously established methods as they are with these new structures. Most of the algorithms and networks that have been developed and used on the 2D data are based mainly on the regularity, the order and the neighbours of the points. In 2D images there are not holes, point of views or occlusions of any sort. Furthermore, with pointclouds, there are irregularities between one scene and another in the structure of the cloud: for example capturing a wall or an open-space kitchen have clearly two different 3D captures. Even more there are the noise embedded in the depth sensor and the slightly different angle with which a scene may have been recorded that may produce a tiny but significantly different cloud are obstacles to the adoption of 3D data.

Despite all these problems there are already works that points out the importance

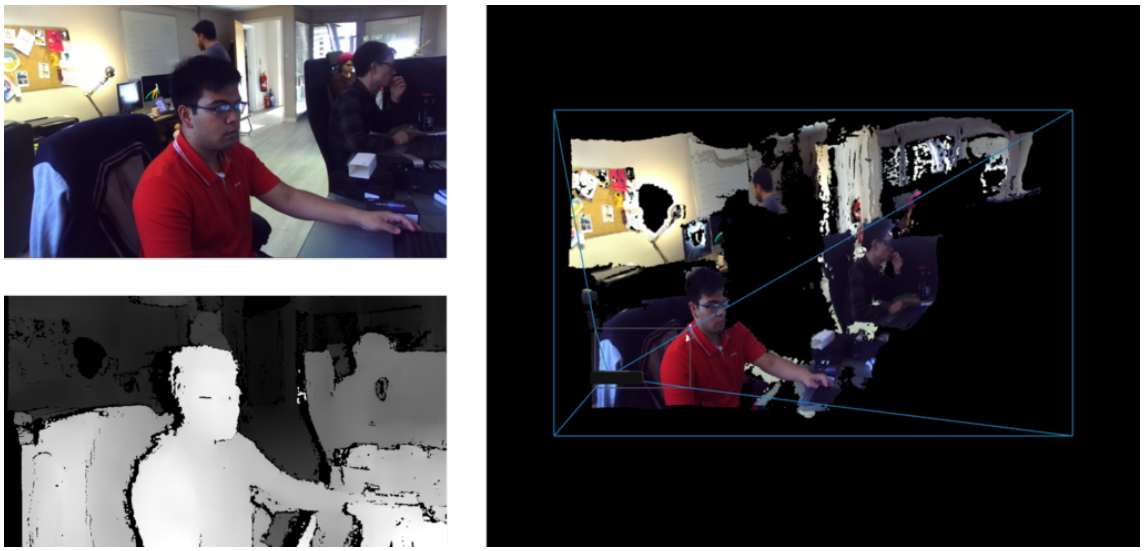


Figure 2.4: Same scene in 2D, depth and pointcloud views [39]

of this kind of data. In [6], by Tombari *et al.*, it is demonstrated how full 3D object recognition is more effective with respect to standard 2D methods. Thanks to a framework that combines Kinect®'s data, reconstruction, segmentation and recognition the authors were able to obtain stunning results. Even if the depth is not directly measured but obtained thanks to the application of a learning algorithm there are example of its usefulness: Tateno *et al.* in [7], by using a monocular camera successfully use depth prediction, 3D reconstruction and semantic segmentation to improve performances of their SLAM[2.1] procedure. So, in our opinion, it is clear that using 3D data is, and will be, an important and interesting field to study, to improve and so where to focus our work.

2.4 State of the Art

For semantic segmentation over 3D data, which is what this thesis is about, the works are mainly derived from deep convolutional networks and machine learning. So far, in the literature, the majority of the methods apply 2D techniques to 3D data by adapting the previous networks to elaborate depth maps. Another way to

[2.1] Simultaneous Localization And Mapping: the robot need to map the surrounding environment and localize itself without previous knowledge.

proceed is to regularize the pointcloud by the means of clustering or voxelization techniques: these are two methods that allows to create groupings of points in the 3D space either in a regular fashion or not. More recently, a lot of research is being done to try to use directly these data without resorting to previous techniques. The main problem of working directly with 3D data is the computational expensiveness of performing 3D convolutions and the sparsity of the points; moreover the whole preprocessing phase eventually needed to voxelize or to select regions in the 3D space is affected by the variability of the considered clouds and point of view. Most of these methods that have been proposed over the years are as one may expect supervised learning algorithms, that means that we have labeled samples over which we can train our learning models, but weakly supervised methods also exists as shown in [8].

As an example of such adaptation of 2D methods, in [8], the authors use a small variation of the widely popular VGG-16 [9] to process separately RGB and depth maps combining this with object detection and Gaussian Process Classification. Also FuseNet [1] works over RGB-D couples: it is an autoencoder-based network that tries to fuse depth and color information taking inspiration and adapt the popular SegNet [10] to achieve that.

In SegCloud [11] the authors voxelize the 3D Point cloud and process it thanks to a 3D fully convolutional neural network. This is anyway an innovative work since, differently to Dai *et al.* in ScanNet [12] that make the same prediction for all voxels in the same column, they provide fine grained prediction for each point, without enforcing assumptions.

Other works use manually crafted and extracted features from voxels, like colours averages or surface normals like in [2, 13], distance from wall like in [14] or the wide adopted ones described in [15] where the authors provide a way to encode depth information in three different channels, creating what they call *HHA* features. This was done to be able to use RGB-optimized networks directly over these three channels derived from depth: disparity, height of the pixels and the angle between normals and the gravity vector based on the estimated ground floor, respectively.

Of interests are also multi-view algorithms, like [16], enabled thanks to the fact that we are considering a 3D space, that fuse together various segmentation to refine the results. Pioneering works are also trying to to learn and take advantage of existing long-distance relations between points. An example of that are the works of Ye

et al. that subdivide the cloud in regular boxes and uses Recurrent Neural Network to process them making use of their "memory" characteristic taking inspiration from natural language processing techniques.

A notable exception that process the cloud as a whole structure is PointNet [17] where raw 3D data are processed. This network learns to extract keypoints that represent the bounds of the objects and uses fully connected layers (Multi-layer perceptrons), instead of convolutional ones, coupled with symmetric pooling to segment the results. Its evolution, PointNet++ [18], that apply the same reasoning alongside a SIFT-like procedure is a variation that try to capture local structures that were limiting the capabilities of PointNet to captures neighborhood information and features.

It can be seen from the brief report above in the latest years 3D data and semantic segmentation are two fields of wide interests characterized by a deep study. This is also because the segmentation has been used for example to improve robots navigation within SLAM systems with visual odometry[2.2] and semantic mapping [19, 5] or to allow a 3D reconstruction of the environment and bringing advancements to research as in robot navigation or skeletal tracking. [19, 20, 21] Moreover, using 3D information may advance fields like human-robot collaboration, for example in industrial environment, where shapes and sizes are of primary importance, or enabling interaction with the whole environment giving to the robot the possibility not only to understand where the door is, thanks to segmentation, but to correctly grasp its knot and open it thanks to the 3D information. Finally, thanks to 3D, we can make use of 6-Degrees of Freedom (DoF) localization that, differently from the usual 3-DoF, let us to fully localize and orient the objects in the space using three angles for orientation and three translations values with respect to the three axes X, Y and Z to characterize the object. Naturally, many more applications and methods are available and here we tried to report the most important and notable ones.

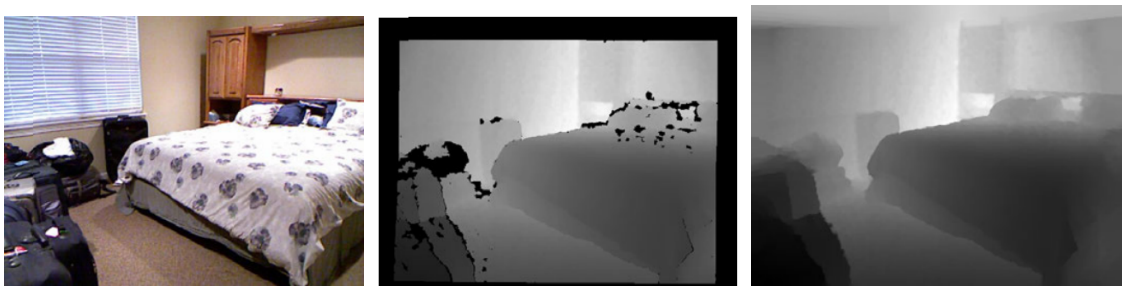
[2.2] Localization driven by visual information.

2.4.1 Datasets for Semantic Segmentation

2.4.1.1 The NYU Depth Dataset V2

There are various datasets over which we can train and test our algorithms [22]. An established testbed for the semantic segmentation on 3D data is the NYU Depth Dataset V2 (NYUv2) [23] which consists of 1449 indoor images captured with a Kinect® device and it is provided both with a 13- and 40-classes mappings [24, 25]. Both FuseNet and 3DEF have been tested against it and so we have a defined and accepted baseline over which we can compare our results and so we will use NYUv2 in our work. This is an extensively used dataset probably because it is one of the first that were available in the robotic community: it is widely adopted thanks to its indoor nature but nonetheless it is relatively small with respect to other datasets like SUNRGBD [26], that comprises RGB-D images from NYUv2, SUN3D [27] and B3DO [28], or the new Matterport3D [29]. Notably, these are all RGB-D datasets but given the camera parameters with a simple operation can be transformed in full-3D ones: 3D datasets are still costly and difficult to obtain and also the lack of learning models that can directly process them does not help their growth.

The NYUv2 dataset is available in various forms. It can be used either with its RGB-D representation and, thanks to the known parameters of the camera, with the correspond pointclouds. Moreover, one can either use as depth measurements the raw values, extracted directly from the camera, or some processed ones: there is a tool, made available by the dataset authors, that fills the holes and smooth the depth readings generating a more continuous map as can be seen in Fig. 2.5. All this data is available in a 640×480 format.



(a) RGB image (b) Raw depth (c) Preprocessed depth

Figure 2.5: NYUv2's depth comparison [23]

2.4.2 Metrics

Finally, to evaluate semantic segmentation there are well established comparison metrics. Given a binary-classification problem we can define a confusion matrix, as in Tab. 2.1, to evaluate our algorithms. In this matrix we put in the diagonal the number of correctly classified samples, while outside we have the number of wrongly classified ones. We call True Positive (TP) an outcome where the model correctly identify the positive class and True Negative (TN) an outcome where instead it correctly identify the negative class. False Positive/Negative (FP, FN) are wrongly predict outcomes where the model assign in the former case a positive class instead of the negative one and in the latter it does the opposite. Naturally the sum of all these values is the total number of samples.

Table 2.1: Sample confusion matrix: actual vs predicted class.

	Pos	Neg
Pos	TP	FN
Neg	FP	TN

Since semantic segmentation is, in general and in our case, a multi-class classification problem we can expand Tab. 2.1 for K-classes, where each sample correspond to a pixel, and is used to extract meaningful evaluation metrics. So for $i \in \{1...K\}$ we have:

- Global accuracy, or the overall accuracy of our model. This may be misleading for a dataset where the number of samples for each class is not the same, or an inbalanced dataset, since the most represented classes may bias the final results because their weight is greater with respect to the other ones.

$$Global = \frac{1}{N} \sum_i TP_i$$

- Per-class accuracy, or class precision, shows how accurate the classifier is for each one of the classes:

$$A_i = \frac{TP_i}{TP_i + FP_i}$$

- Mean accuracy, an average of per-class accuracy:

$$Mean = \frac{1}{K} \sum_i \frac{TP_i}{TP_i + FP_i}$$

- Intersection over Union, or the number of pixels common between the target and prediction masks divided by the total number of pixels present across both masks

$$IoU = \frac{1}{K} \sum_i \frac{TP_i}{TP_i + FP_i + FN_i}$$

These are widely accepted as comparison values in the literature even if there is not a common guideline. Naturally, because of the misleading information given by the global accuracy due to the inbalacing of the classes in the dataset, the other three are in general more informative. We will use these established metrics to evaluate our learning models.

2.5 Our Contributions

This work will focus over FuseNet and 3D Entangled Forests (3DEF) used over the NYUv2 dataset. The former is a convolutional autoencoder, working over 2.5D data, exposing two encoding segments, one for the depth map and one for the RGB image, and one decoder part. The latter is a learning method that, after a pre-processing of the pointclouds, uses a forest of decision trees to obtain the sought result. With FuseNet also a small study with COROMA, an industrial dataset acquired by our laboratory, have been developed. We developed various FuseNet structure variation that allow us to include information used by 3DEF in different ways, whereas with 3DEF we flawlessly incorporate features extracted in different point of the original FuseNet architecture.

Chapter 3

FuseNet

3.1 The Original Network

The authors of FuseNet started from a Couprie's work [24] where is pointed out the fact that depth information, for classes that have similar location and appearance may improve and reduce uncertainty of the segmentation for different objects that have similar appearance information. Couprie also note that for objects that have high variability in depth dimension was in any case better to use simple plain RGB: they left the fusion of depth maps and RGB images as open problem. Hazirbas *et al.* tackle exactly this problem by developing their FuseNet network.

As stated in Chapter 2, we face in this case a clear example of 2D methods adapted to RGB-D data: FuseNet, in fact, is clearly inspired by *SegNet* [10], that works with plain RGB images and has an encoder-decoder structure depicted in Fig. 3.1: this is also stated in the original FuseNet paper by Hazirbas and his colleagues.

A previous work tried to make use and integrate depth by using LSTM layers [30] but this architecture is fairly complex and hence difficult and timely to train. With FuseNet they tried to obtain a more simple and light model to obtain the same result to merge RGB and depth information. In their work, among the other things, the authors note that HHA features (briefly cited in Chapter 2) requires too much computation and are less informative with respect to what RGB channels bring to the learning procedure. Moreover, the authors demonstrate that this representation does not hold more information with respect to the plain depth.

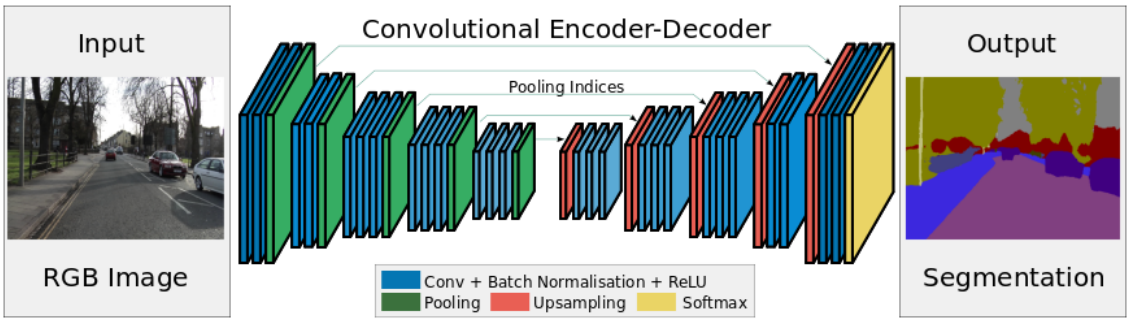


Figure 3.1: SegNet network [10]

The proposed network has an encoder-decoder structure, depicted in Fig. 3.2, with the aim of extracting and fusing features in the former side and expanding them in the latter one. As can be seen from a rapid comparison between Fig. 3.2 and Fig. 3.1, the two architectures are almost the same, differing mainly in the presence or absence of the dropout layers. This difference is resolved if we consider Bayesian SegNet [31] that maintains the same structure but add the dropout layers.

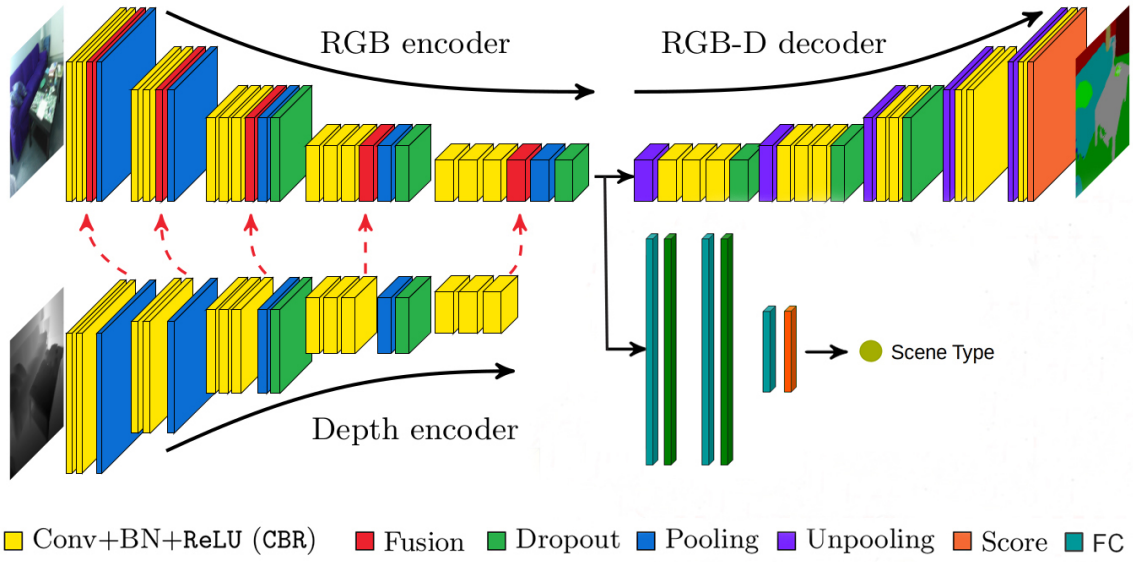


Figure 3.2: Original FuseNet architecture [40]

The encoding side of FuseNet consist in two branches, one for the RGB image and one for the corresponding depth map. The structure of each branch is directly

derived from VGG-16. In FuseNet two main fusion techniques have been applied for merging features extracted from the two different inputs: namely sparse and dense fusion. Both consist in the sum of element-wise features coming from the two branches, as in Fig. 3.3, but the former is performed before each pooling layer while the latter is performed after each one of the CBR blocks as depicted in Fig. 3.4. A CBR block is a common structure in convolutional neural networks and consists in three layers: one convolutional (C), one batch normalization (B), that helps in terms of performance and stability of the network, and one ReLU (R), which is the activation function. In this way the authors are able to transfer the encoded depth information and directly merge it with the RGB elaborated one, mixing them almost since the beginning of the network. Both inputs are normalized in the range [0,255].

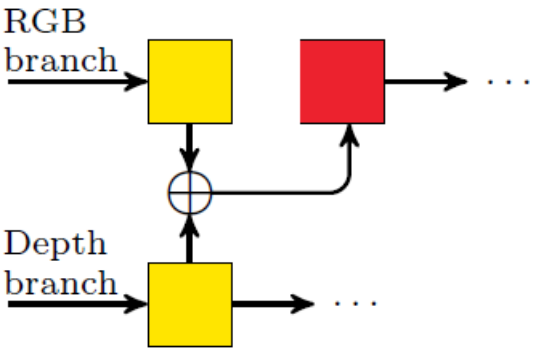
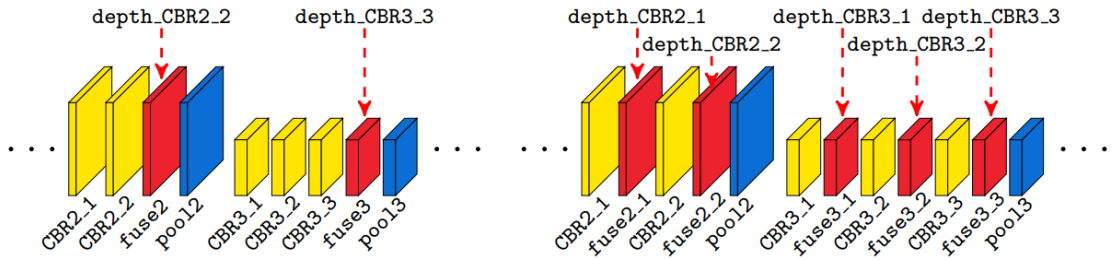


Figure 3.3: A single Fusion block [1]



(a) Sparse fusion (SF)

(b) Dense fusion (DF)

Figure 3.4: Sparse and Dense Fusion comparison [1]

After the encoding the decoder part takes care of up-sampling the encoded information to obtain the final segmentation in the original input resolution. It is made

by mirroring the same structure of the encoder, where memorized unpooling is applied as clearly depicted in Fig. 3.1. The encoded information can be used directly also with a simple classification structure made of fully connected and dropout layers to obtain the scene type as depicted in Fig. 3.2.

Hazirbas *et al.* found out that their methods successfully improves over the cited state-of-the-art methods, hence obtaining promising results by the fusion of RGB and depth in their proposed architecture. Specifically, their sparse fusion technique manage to obtain the best global and IoU scores whereas the deep fusion technique to obtain the best mean accuracy overall, making sparse fusion their favourite choice over the two possibilities. We will start from here for the experiments made in Chapter 5 but we will use also directly this network in some preliminary experiments.

Our implementation is based on a public GitHub[3.1] repository that is the official PyTorch version. Inside it we have found the trained model over NYUv2 dataset available to download: in this case the NYUv2 consists in RGB-D couples with depth elaborated (or filled), resized to 320×240, with a 40-classes mapping.

Apart from the original pretrained version we decided to train FuseNet also with the 13-classes mapping of NYUv2. This was done mainly because 3DEF, as we will see in Chapter 4, originally used this class mapping and so we were be able to extract the correct features for the right version of the data. We tried also to use raw depths and bigger images but we did not found valuable differences between these versions and the original one either in terms of training times or result of the main metrics. The results of these tests can be seen in tables 3.1 to 3.4.

Here we have a clear example of how an inbalanced dataset may affect the results: the global accuracy of FuseNet trained over the smaller (320×240, Original) and the bigger (640×480, Big) version of the dataset are near but the mean accuracy, that weights each class, has a substantial difference. This aspect is even clearer if we check classwise's accuracy: the segmentation of classes of the bigger version have usually a better accuracy with respect to the original one. Another important finding is that filled depth performs better with respect to the raw one meaning that filling depth holes for this dataset helps the classification. These considerations are true both for the training over 40-classes and 13-classes mappings: what changes between these two versions are the overall results where it is easy to notice that

[3.1] https://github.com/MehmetAygun/fusenet-pytorch

a lower number of classes leads to better scores over all the metrics. Probably, 13 classes are easier to recognize than 40 because they are less crowded and more defined overall.

	Global acc.	Mean acc.	IoU
Original	76.40	66.93	54.74
Raw depth	75.95	67.03	54.46
Big dimension	76.89	68.57	55.98

Table 3.1: FuseNet trained over NYUv2, 13-classes mapping, general results

	Original	Raw depth	Big dimension
Class0	73.57	74.25	69.62
Class1	56.37	54.99	62.98
Class2	69.16	71.47	71.56
Class3	68.54	68.50	68.02
Class4	71.50	69.43	79.33
Class5	96.92	96.21	97.58
Class6	58.28	60.40	56.51
Class7	70.88	72.36	68.59
Class8	59.19	50.30	52.64
Class9	91.87	90.59	90.88
Class10	68.75	71.06	72.80
Class11	48.55	52.13	58.14
Class12	36.52	39.68	42.74

Table 3.2: FuseNet trained over NYUv2, 13-classes mapping, classwise accuracy

	Global acc.	Mean acc.	IoU
Original	68.76	46.42	35.48
Raw depth	64.50	38.87	28.83
Big dimension	67.55	46.73	35.01

Table 3.3: FuseNet trained over NYUv2, 40-classes mapping, general results

	Original	Raw depth	Big dimension
Class0	91.08	90.68	91.47
Class1	97.04	96.73	97.09
Class2	71.53	69.14	68.40
Class3	74.77	76.32	69.25
Class4	69.17	67.75	73.56
Class5	71.46	73.05	70.80
Class6	49.51	48.62	47.95
Class7	24.17	31.94	24.61
Class8	56.82	59.99	57.84
Class9	50.09	57.04	54.64
Class10	73.33	70.85	75.73
Class11	NaN	0.00	65.24
Class12	63.51	64.18	62.45
Class13	22.75	19.18	24.31
Class14	14.67	14.26	24.13
Class15	57.85	62.17	58.19
Class16	55.02	46.16	50.28
Class17	52.88	51.79	57.76
Class18	36.20	33.02	27.00
Class19	36.14	24.59	30.45
Class20	18.70	25.64	33.43
Class21	NaN	0.00	83.33
Class22	25.31	21.24	20.62
Class23	NaN	0.00	30.52
Class24	47.57	58.24	53.18
Class25	NaN	0.00	27.43
Class26	18.99	29.70	30.99
Class27	41.41	22.06	25.98
Class28	NaN	0.00	7.33
Class29	17.37	26.38	20.24
Class30	38.47	32.63	48.76
Class31	44.46	42.77	31.74
Class32	71.57	74.62	76.08
Class33	NaN	0.0	54.8
Class34	42.89	38.43	49.13
Class35	43.45	38.92	50.61
Class36	3.04	1.17	4.71
Class37	17.89	20.31	24.90
Class38	13.72	7.84	13.26
Class39	65.62	57.44	51.10

Table 3.4: FuseNet trained over NYUv2, 40-classes mapping, classwise accuracy

3.2 The COROMA RGB-D Dataset

COROMA is a dataset available in our lab that is formed by scenes captured in an industrial environment with a Kinect®. It is a dataset born within an European project that aims to develop a cognitively enhanced robot that will execute multiple tasks within metalworking and advanced material companies. We used this dataset in its RGB-D version to initially test the capability of FuseNet to adapt to new datasets, different from the ones it is trained on. Each RGB-D couple has size 960×540 pixels. COROMA scenes consists of various possible industrial scenarios, or areas, each of them consisting of a large number of captures of the scene from a slightly different point of view and containing for example technical zones or pieces of a boat.

For this tests we worked over a subsection of the whole COROMA, consisting in one single area formed of 6549 RGB-D couples, since the scope was to get insight of the network and use it a bit before doing the work with 3DEF. The split to obtain train and test sets was made randomly with a 70%-30% ratio by taking only one over ten images to avoid overfit over a single point of view of the considered scene. Moreover, there are various possible mappings for COROMA, that originally comes with 18-classes, and in this case we use a 13 one. Finally, in this case each RGB-D couple is resized to 224×224 to speed up computation and normalized to [0,255]. We used the tool from NYUv2 to elaborate the depth maps since it is a common procedure for each one of the datasets used with FuseNet. Also in this case, we trained and compared the two version of COROMA with raw and elaborated depth as for the NYUv2 dataset.

	Global acc.	Mean acc.	IoU
Raw Depth	94.20	91.21	80.01
Elaborated Depth	93.70	89.54	78.36

Table 3.5: Training results of FuseNet over reduced COROMA dataset. 1000 epochs.

As depicted in Tab. 3.5 it is clear that, differently from the NYUv2 case, raw depths works slightly better with respect to elaborated ones meaning that it is not given that filling holes in the depth maps and elaborate them with such tool can be helpful with the task of segmentation with FuseNet network.

More complete test with COROMA and FuseNet will be done within a different work.

3.2.1 Fine-Tuning

For the first problem we slightly modified the provided network to be able to load the weights of FuseNet trained over NYUv2 with the 40-classes mapping and then use them to learn to correctly classify the COROMA dataset. By doing so we are performing fine-tuning to check whether we may be able to use a pretrained FuseNet to correctly segment our new images with a different mapping.

To do so we have different ways of acting based on how much of the original weights we keep and let to update. Indeed during the loading phase we can choose programmatically which weights we can load from the pretrained FuseNet and how much of them we can update to learn the new representation. In this sense we tried various combinations based on these two factors as shown in Tab. 3.6. For the reset of the weights we opted either for the whole last block, starting from the last unpool layer, or only the last convolutional layer. Instead, for the learnable parameters, we allowed to learn either the whole RGB-D decoder, only the last block starting before the last unpool layer or only the last convolutional layer. We preferred not to let the weights be updated in the encoder side since that one is where the network learn the representation of the data. We wanted to understand how much of the decoder side is meaningful to expand such representation to the original size. These combinations allowed us to do various tests and to obtain meaningful insights. We trained all these combinations for 150 epochs.

Reset →	Last layer			Last block		
Learn ↓	Global acc.	Mean acc.	IoU	Global acc.	Mean acc.	IoU
Last layer	8.11	10.65	1.43	—	—	—
Last block	68.02	28.91	22.16	8.51	8.68	2.02
Decoder	97.64	95.67	68.86	54.63	56.66	25.93

Table 3.6: Fine Tuning of FuseNet over reduced COROMA dataset. In the columns the reset depth, in the rows the allowed learnable parameters. Trained over elaborated depths.

As shown in Tab. 3.6 we discovered that the less of the pretrained weights we reset the better it is. The decoder needs to be free to learn and adapt its weights to achieve good performance: blocking the update procedure up to the last layer or block inhibits the possibility to obtain good results since it impede a correct expansion of the encoded information. This means that the encoding side of the network is important, since it learns a correct representation of the input data, but of equal importance are the whole decoding side that is much more related to the input dataset and the sought result. Finally, by comparing Tab. 3.5 and Tab. 3.6, one may notice that in our particular scenario the best is the fine-tuned version. Anyway all of these are only insights on how FuseNet work, we have clear in mind that these results may be biased with overfitting and other problems due mainly to the low variability in the used dataset.

Chapter 4

3D Entangled Forests

Differently from FuseNet, 3DEF do not work on RGB-D but with pointclouds. The work of Wolf *et al.* [2] acknowledges the presence in the literature of various approaches that capture contextual and geometrical information of these structures but its goal is also to obtain computation efficiency, that is still missing in most of them, and to achieve this they make use of a random forests classifier.

The base structure of random forests are decision trees. Each tree works by applying a binary test for each sample against each available feature and, after that, a split of the dataset is made by considering the most informative one. A random forests is nothing more than a collection of decision trees that operates as an ensemble: using bagging of the data, so a random sample with replacement of the dataset, and a random feature selection instead of consider every possible one ensure uncorrelation between trees. This uncorrelation helps the final classification.

The computational efficiency is of particular interest, especially in robotics applications. In the latest years, thanks to the advancements of GPUs technology, we were able to develop more accurate learning models by leveraging their computing capabilities: this allowed the creation of networks with more and more parameters but high computational power requirements and, as consequence, high energy consumption. However, this is impractical in many real-world robotics applications and scenarios where the amount of available energy is often limited: in these cases we cannot think to embed a powerful GPU directly into a robot. In fact, mobile robots

and drones may need to move for example for long periods of time without the possibility of recharging the batteries, like in a open world environment or in disaster zones where power may be missing and an internet connection to a datacenter may not be reliable or even accessible at all. Power efficiency is then clearly an important and needed to be examined field and since 3DEF are trained with CPUs are of particular interest.

The proposed method works by applying a preprocessing of the pointcloud using outlier and bilateral filters. Then the viewing angle is estimated and the cloud is rotated accordingly to have a flat floor. After that voxelization is performed with the help of Point Cloud Library[4.1] (PCL) [32], applying a region growing algorithm, and both *Unary* features and *Entangled* coefficients are calculated obtaining the final clusters (or regions). Unary features are related to each single cluster while Entangled ones describe relations that exists between all different couples of them. These features are what is effectively used by the forest learning algorithm to perform the splits and hence the segmentation task.

Each cluster's Unary feature vector is formed by 18 features that are formed by fast to compute and simple coefficients, avoiding calculation of more articulated ones like HOG[4.2] [33] or textons [34]. As can be seen in Tab. 4.1 these are used to grasp region-correlated factors: we have the mean angle and standard deviation of the cluster's surface normal and the mean and standard deviation of the color in the LAB space. After that we consider both the minimum and maximum height of the cluster with respect to the floor. Furthermore, for each cluster we have a bounding box that contains it and we describe this with its dimensions (width, height and depth), its area in the horizontal and vertical plane. Finally, for each bounding box, by taking the ratios between width, height and depth, we obtain elongation and thickness of such boxes. The binary test necessary for the decision trees learning in this case is done thanks to a threshold: if the value of the feature is above that threshold it evaluates to true and information gain is checked.

For Entangled coefficients the real problem consist in finding metrics that not

[4.1] www.pointclouds.org

[4.2] Histogram of Oriented Gradients

Feature	Dimension
Normal angle and standard deviation (std)	2
Color(LAB) mean and std.	6
Height, min and max	2
Bounding box dimensions	3
Bounding box area	2
Elongation and thickness	3
Total	18

Table 4.1: Unary features and their dimension

only can capture relations between data points but they should do that in a repeatable way and across different scenes. Also, differently to the 2D case where this is just related to pixel offsets, with pointclouds we have to remember that we are in a projective space: this simplification is no longer effective since viewing angle may change and areas that were informative in the first scene may not be so in the second one. Wolf and his colleagues proposed a scheme to select close-by segments providing contextual information directly in the 3D space. Essentially for each couple of regions they calculate these coefficients, depicted in Tab. 4.2, and craft the real Entangled Features through binary tests and constraints to be used inside the random forest algorithm for each split. These features are related to common ancestors, node descendants and other characteristic that are calculated starting from these coefficients and each decision tree structure in real time during the learning phase. This is why we brought coefficients that enabled these features and not directly the features themselves that would have been impossible. For further details on how these features are calculated we refer to [2].

The original algorithm is then evaluated over the 13-mapping version of NYUv2, using raw depths and the original size, obtaining an overall improvement, with respect to the considered comparison methods, both in class and global accuracy. Moreover, the usefulness of the Entangled Features is demonstrated by a comparison between training with only the Unary ones that can be found in the original paper [2].

Coefficient	Description
Cluster Distance	Minimum distance between regions (cluster)
Point To Plane Distance	Minimum distance between voxels of a region and centroid of a different one
Inverse Point To Plane Distance	Opposite of the previous
Horizontal and Vertical Distance	Difference between horizontal and vertical angles between regions

Table 4.2: Entangled coefficients and their description

The novelty, with respect to other approaches that use Random Forest or Conditional Random Field, is the new concept of 3D Entangled Features which can capture frequently combinations of classes, learn their 3D geometric configuration in the scene and their contextual and spatial relations. Moreover, this is learned not only for different classes but also within the same one. Finally, this method manage to obtain near real-time performances but without the need of GPUs and of high computational powers. Still, one of the drawbacks of this method, is that threshold used for test, merging of the voxels and other parameters are totally set by-hand: the forests lacks so of a general implementation. Parameters and thresholds to control for example initial voxelization of the cloud can be set at the beginning but probably needs tune if one wants to use a different dataset and there is not a given, always working, setup. For example, as we will see in sections 5.1.2 and 6.1.2, we had to disable the initial bilateral filtering and the outlier pruning in our elaborations.

Chapter 5

From 3DEF to FuseNet

5.1 Input

5.1.1 The Data

We will use FuseNet trained both over the 40- and the 13-classes mapping of NYUv2 dataset to do comparisons with both versions. As shown in tables 3.1 to 3.4, doing training over raw depths or with the bigger input dimension (i.e. 640×480) does not change the outcomes of this network in a meaningful way so we preferred to keep the smaller images.

5.1.2 The Features

As first step we need to extract the features used by the 3DEF algorithm. Here lies the first problem: the two learning methods works over two different version of the same dataset and reasons over different structures.

If FuseNet uses RGB-D couples, reasoning with convolution over the whole image, 3DEF essentially works over separated clusters trying to labelling them. Moreover, the number of clusters, their ordering and characteristics are not predictable from one cloud and another. There may be clouds with hundreds of clusters and clouds with only few dozens of them and, moreover, the *floor* may mostly belong to cluster 1 in one cloud and to cluster 10 in another one.

The problem is how to project the pointclouds to obtain the same 2D shape expected by FuseNet and create a point-by-point feature vector of fixed size.

The solution is a projection procedure. Thanks to the fact that any 3D structure captured by a camera can be projected in a 2D space if the camera intrinsic parameters are known we developed a script that given a pointcloud in input return the projected 2D RGB image in the camera plane. The geometrical reason behind that is shown in Fig. 5.1, representing the pinhole camera model, and mathematically, using homogeneous coordinates and up to a scale factor, it is

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.1}$$

$$p = A \left[R \middle| t \right] P \tag{5.2}$$

Essentially any 3D point $P = (X, Y, Z)$ can be projected in the corresponding 2D point $p = (u, v)$ given that we know the intrinsic parameters of the camera, so its focal length in the two directions (f_x and f_y) and its principal point (c_x and c_y). Rototranslation matrix $\left[R \middle| t \right]$ account for eventual needed transformations between world and camera reference frames.

So, by using the known focals, centers and the estimated rototranslations (computed by the 3DEF preprocessing algorithm), we can use Listing 5.1 that I have developed to obtain the desired result. Finally, regarding projection, the original algorithm to create the 3DEF' features uses a bilateral filter to smooth the depth information but we have to disable it since it would move the points Z values in an unpredictable way ruining essentially the whole 2D projection procedure.

```
def  project  (points, angle):
    # official camera matrix given with NYUv2 dataset
    camera = np.array([
                    [5.1885790117450188e+02, 0, 3.2558244941119034e+02],
                    [0, 5.1946961112127485e+02, 2.5373616633400465e+02],
                    [0,0,1]], np.float)

    tvec = np.array([0,0,0], np.float)
    rvec = np.array([0,0,0], np.float)

    #rotation of the camera, as applied in createeftraining_cluster
    r = Quaternion(axis=[1,0,0], angle=angle[0])
```

```
p = Quaternion(axis=[0,1,0], angle=angle[1])
y = Quaternion(axis=[0,0,1], angle=angle[2])
q = r*p*y
rvec = q.rotation_matrix

out = cv2.projectPoints(points, rvec.T, tvec, camera, None)

return out[0][:,0]
```

Listing 5.1: Pointcloud Projection

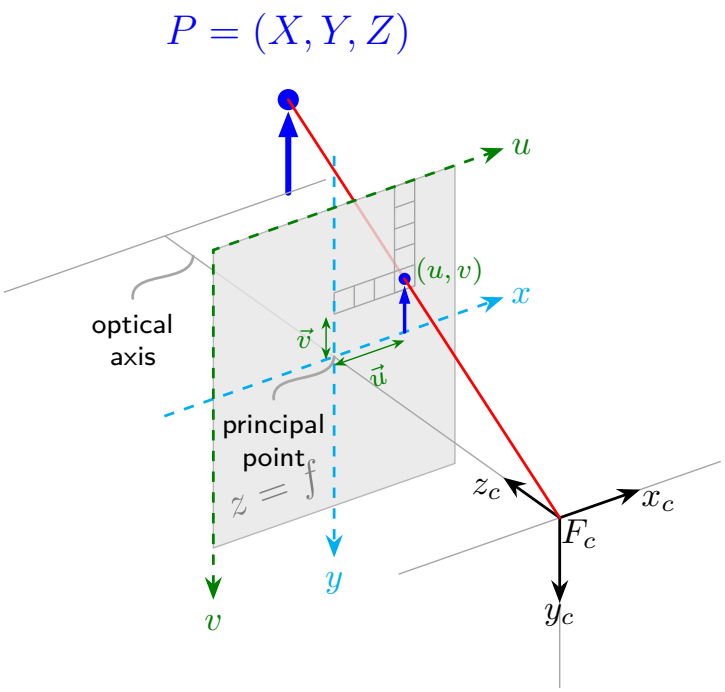


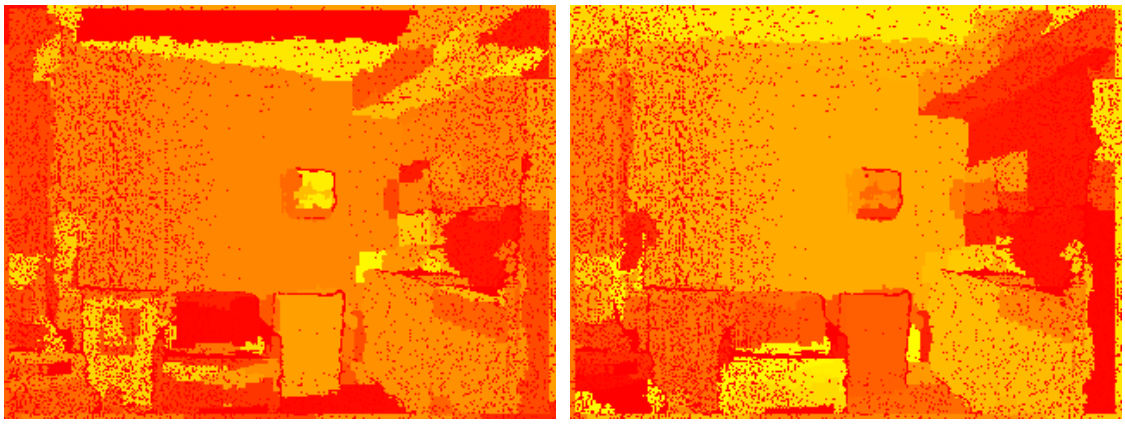
Figure 5.1: Camera Projection using the pin hole camera model

Another problem is the dataset that we have to use to train 3DEF to extract these features: the number of classes is irrelevant since the procedure to create the clusters and their feature vector is independent from them. What can influence these features are the depth (if filled or raw) and the dimension. By subsequent experiments we find out that the best way to proceed is to use the original big pointclouds to calculate the features, project them and then resize the obtained 2D matrices with a nearest neighborhood logic to avoid unwanted values changes

that would be obtained by using for example bilinear or other filters. Moreover, raw depths measurements instead of elaborated ones could not be used because too much points would have been lost: by having the same $Z = 0$ they would have the same projection as can be clearly seen in Eq. (5.2). Even with elaborated depths there are some missing points related to the use of the outlier filter but we noticed that using the bigger input, once resized, we had less holes in our final feature matrix with a reduction of about 25% of the missing points going from 4% to 3% of the whole image. This problem can be avoided by not using such filter but we tried to be as much as possible close to the initial implementation of 3DEF.

5.1.2.1 Unary Features

With Unary features the work was straightforward since for each cluster we have a single feature vector. Since we know the clustering of the cloud and the feature vector associated to each cluster we can obtain a 2D matrix where for each point we have the corresponding feature vector thanks to the projection procedure. As an example, a representation of two projected and manually selected features is represented in Fig. 5.2: each pixel represent the value of such Unary feature in the image. Regions are clearly recognizable even if they do not exactly correspond to clusters since a given feature may have the same value for a multiple number of them.



(a) Feature 12 (b) Feature 9

Figure 5.2: Examples of imported features from 3DEF to FuseNet

At this point we have a 2D matrix of the desired size where each “pixel” contains

a 18-dimension feature vector described in Tab. 4.1. Essentially we are projecting the clusters obtained with the processing of the pointclouds, with their features vectors, in the 2D dimension.

Finally, these features contains LAB values that are essentially a mapping of RGB colors we also thought about reducing the size of this feature vector from 18 to 12 by pruning these values because we already have this information in the RGB image.

5.1.2.2 Entangled Features

For the Entangled Features the thing is more difficult since these are calculated using the coefficients summarized in Tab. 4.2. These features cannot be taken as they are used in the random forests since neither using the binary tests, that are related to the forest learning structure, nor emulating them directly in the convolutional network are viable options: we have to use the "raw" coefficients. Furthermore, directly use 2D-projected clusters as for the Unary scenario where we use this procedure to obtain feature vectors for the RGB-D data can not work in this case. The power of a convolutional network is to find hidden and steady relations in the data and extract and use them to perform the given task: if fed with random, odd data the learning power will be disrupted. Just imagine that each pixel of a 2D RGB image instead of having always R as first component has a random channel: it would impossible to tell what is the channel and how to correctly visualize the image. The cardinality and ordering of clusters are not constant between pointclouds: it is straightforward to understand that this would bring variable-sized feature vectors if we take this as it is and cannot be directly used in a convolutional neural network.

Fortunately the Entangled Features are based on coefficients and these coefficients are expressed as distance values calculated between couples of clusters: given what we have just seen about cardinality and ordering we have to tackle the problem with a different point of view to be able to obtain something usable. The solution we thought about was to emulate these values by using their base factor(s). Essentially, as expressed in Tab. 5.1, for each coefficient we found its base components referring to the current cluster or voxel. By doing so we obtained a base vector of length 8. The cloud is projected in 2D as in the case of the Unary feature but for each point we have both the coordinates of the centroid of the cluster and voxel to which it

belongs, its vertical angle and the normal to it.

Finally, since create all-clusters coupling is not feasible we also thought how to partially obtain the original distances and we resort to calculate them within a 4-neighborhood widow. For a given pixel indeed we can define its 4-neighbors as its adjacent pixels along the main directions and we can take the needed distances by using the values obtained previously. In case of pixels located in the angle or in the border of the image we set these differences at 0. By doing that we were able to further increase the dimensionality of the feature vector to 24 as described thoroughly in Tab. 5.1.

	Feature	Dimension	Description
Base	Cluster (C) centroid	3	Coordinates of the cluster's centroid
	Voxel (V) centroid	3	Coordinates of the voxel's centroid
	Angles (A)	2	Vertical and horizontal angles
Advanced	CC difference	1×4	Clusters' centroids distance
	CV difference	1×4	Cluster's and voxel's centroids distance
	A differences	2×4	Angles differences
	TOTAL	24	

Table 5.1: Description and Dimension of our obtained 3DEF Entangled Features

5.1.3 Generated Features Set

Summarizing, we used elaborated depths both for FuseNet RGB-D input and 3DEF. We obtained also long and short versions of Unary and Entangled features and we will mix them together to obtain our features sets that are described in Tab. 5.2. These are the sets that we are going to use to try to improve FuseNet performances by incorporating them in the learning procedure. Each of them is obtained thanks to the procedures presented in Section 5.1.2.1 and Section 5.1.2.2. Combination of Unary and Entangled sets are made by concatenation of their basic versions, always with the Unary features placed in the first places.

These dataset will be used as input of the four variations of FuseNet that will be described in the next sections and used to train the networks over both 13- and 40-classes mappings generating a mixture of possibilities. We will train each one of the networks for 600 epochs.

Dataset	Formed by	Number of features
UN	Unary	18
UNR	Unary reduced	12
ENT	Entagled (with 4-neighborhood differences)	24
ENTR	Entangled reduced (only base values)	8
UN+ENT	Unary and Entangled	42
UNR+ENT	Unary reduced and Entangled	20

Table 5.2: Obtained features set with their shorter name and dimension

5.2 Proposed Networks

In the following sections we will present the FuseNet variations that we have thought of to embed the features used by 3DEF within this learning method. Sparse Fusion, Only Encoding and the Parallel Autoencoder variations are directly inspired by FuseNet original network, meanwhile Direct Fusion consist in the direct merging of such features.

5.2.1 Sparse Fusion

By following the same reasoning of FuseNet's sparse fusion technique, we encoded our feature vector with the same encoding structure and using the same methodology of merging them. We weight RGB, depth and our features equally. We wanted to try to test the same reasoning that FuseNet's authors made for depth with our extracted features as represented in the schematic of the proposed methodology can be seen in Fig. 5.3. The final goal is surely to incorporate what we can obtain by 3DEF preprocessing and so we thought that to start replicating the same structure could be of help.

In this case we can see that no meaningful improvements are brought by the introduction with this methodology of the features sets. In most of the cases with the 13-classes we can notice a very slight improvement in all metrics. Also for the 40-classes mapping we have a tiny improvement in mean accuracy and IoU but we can notice an overall small loss in the global accuracy. Moreover, Entangled features seems to negatively affect training with this network design since the performance with ENT and ENTR features sets are the only ones that are worse with respect to

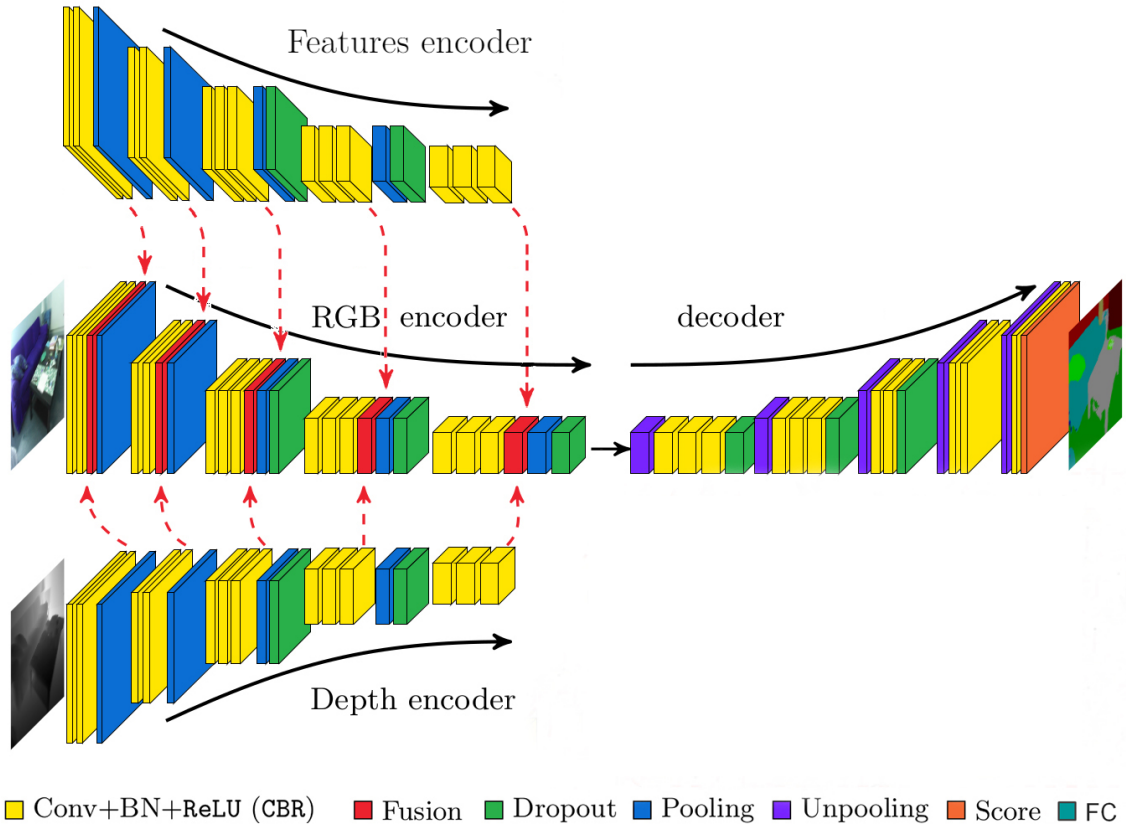


Figure 5.3: Embed 3DEF' features in FuseNet with Sparse Fusion

the original ones. In general we can say that these features in combination with this first FuseNet variation does not change things a lot in terms of performance. Finally, by doing this variation we added 33% of parameters, as can be seen in Tab. 5.19, going from 44.173M to around (depending on the considered feature set) 58.908 M, resulting of around a 40% increasing also of per-epoch training times as depicted in Tab. 5.19.

	Global acc.	Mean acc.	IoU
UN	76.44	67.52	54.93
UNR	76.51	67.29	54.70
ENT	75.19	65.28	52.83
ENTR	74.84	64.91	52.27
UN+ENT	76.86	67.63	55.29
UNR+ENT	76.54	67.30	54.96
Original	76.40	66.93	54.74

Table 5.3: 13-classes, Sparse Fusion network, general results

	UN	UNR	ENT	ENTR	UN+ENT	UNR+ENT
Class0	73.57	77.36	78.95	78.52	78.08	76.00
Class1	55.36	54.77	48.65	52.03	55.37	54.65
Class2	72.84	70.84	74.00	71.26	71.75	71.52
Class3	71.87	70.29	69.00	67.93	71.62	70.21
Class4	78.38	75.75	76.00	73.65	74.92	74.25
Class5	97.17	97.03	97.36	96.86	97.26	97.20
Class6	56.07	57.99	50.90	53.51	57.43	60.34
Class7	72.59	70.12	73.36	73.62	71.72	71.60
Class8	54.74	53.17	48.49	51.36	56.63	57.13
Class9	90.67	91.30	91.90	91.08	91.14	91.50
Class10	66.30	70.45	61.88	59.61	67.90	66.25
Class11	50.18	48.87	47.02	44.28	49.91	48.15
Class12	38.05	36.79	31.17	30.08	35.51	36.14

Table 5.4: 13-classes, Sparse Fusion network, classwise accuracy

	Global acc.	Mean acc.	IoU
UN	68.63	47.08	35.72
UNR	68.72	46.44	35.62
ENT	66.79	42.82	32.95
ENTR	67.26	43.49	33.12
UN+ENT	68.60	47.95	36.28
UNR+ENT	68.30	46.33	35.13
Original	68.76	46.42	35.48

Table 5.5: 40-classes, Sparse Fusion network, general results

	UN	UNR	ENT	ENTR	UN+ENT	UNR+ENT
Class0	91.57	92.27	91.87	91.18	91.04	91.25
Class1	96.64	96.89	96.69	96.04	96.77	96.85
Class2	68.65	70.20	69.44	70.89	69.93	70.33
Class3	76.60	76.05	74.68	78.22	77.47	76.07
Class4	69.63	73.88	75.06	72.66	72.89	74.05
Class5	70.11	72.93	71.41	69.67	70.96	76.28
Class6	49.51	52.60	49.91	48.90	52.29	48.47
Class7	23.40	23.21	14.86	20.23	25.61	29.60
Class8	55.75	57.28	45.33	48.59	58.20	48.91
Class9	53.74	53.86	45.40	43.57	54.99	46.28
Class10	71.89	69.58	66.60	68.51	71.74	69.77
Class11	NaN	NaN	NaN	NaN	NaN	NaN
Class12	62.32	61.90	60.14	60.06	57.19	59.12
Class13	22.61	24.94	17.49	17.04	18.85	31.20
Class14	12.79	12.96	15.51	19.11	15.29	14.31
Class15	57.61	55.34	51.74	53.07	64.72	56.83
Class16	58.04	57.72	34.21	39.69	52.99	55.73
Class17	56.93	49.70	48.95	53.51	52.07	49.38
Class18	37.08	38.50	40.11	42.69	37.97	34.33
Class19	36.87	31.04	30.81	30.21	37.44	31.99
Class20	19.60	21.65	17.12	16.65	24.82	23.38
Class21	NaN	NaN	NaN	NaN	NaN	NaN
Class22	20.46	13.94	12.69	18.62	23.05	23.89
Class23	NaN	NaN	NaN	NaN	NaN	NaN
Class24	44.57	48.51	33.99	39.36	44.86	53.53
Class25	NaN	NaN	NaN	NaN	NaN	NaN
Class26	28.07	24.32	27.94	22.36	32.61	25.96
Class27	39.25	32.51	32.87	36.84	50.62	35.34
Class28	NaN	NaN	NaN	NaN	NaN	NaN
Class29	29.19	26.68	17.31	7.67	30.41	28.59
Class30	40.49	34.36	29.01	36.44	42.18	31.15
Class31	40.70	42.10	42.93	36.52	53.02	41.38
Class32	72.58	74.02	68.47	66.12	70.32	76.88
Class33	NaN	NaN	NaN	NaN	NaN	NaN
Class34	43.39	45.44	43.41	43.49	44.88	47.56
Class35	46.00	45.57	39.13	40.73	43.35	37.65
Class36	4.86	2.47	1.34	2.51	1.91	0.85
Class37	22.49	22.31	15.58	14.49	18.63	14.67
Class38	14.17	13.63	13.41	10.09	10.98	12.62
Class39	63.24	60.61	60.46	63.01	60.41	61.17

Table 5.6: 40-classes, Sparse Fusion network, classwise accuracy

5.2.2 Only Encoding

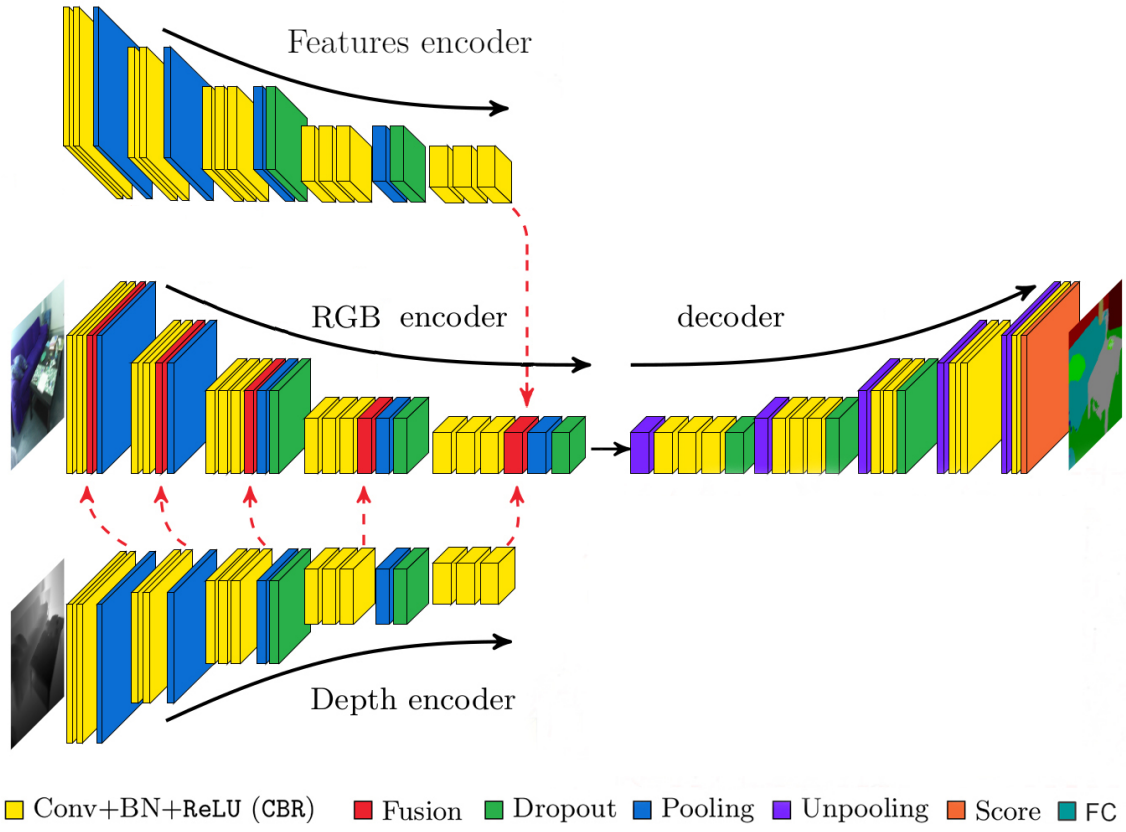


Figure 5.4: Embed 3DEF' features in FuseNet by Only Encoding them

Differently from the previous case here we adopted a slight variation where we fuse only the final encoded vector to check whether a less invasive approach could be beneficial. So we tried to encode features set by using the same structure of the FuseNet's encoding branch but instead of doing fusion of the obtained weights before each pooling layer we do it only before the last one. Schematically this is shown in Fig. 5.4, a diagram of our proposed network. As can be seen clearly also from the picture the encoder structure is necessary to merge features since all three dimensions of the merged matrices must be the same.

Opposite to the previous case, here the use of ENT and ENTR features sets does not ruin the results either with respect to the 13-classes mapping and in all cases of the 40-classes one except of the combination of it with ENT set. Apart from this

we cannot say that there are much differences between the two methods. They are near in all the considered metrics and slight variations may also be linked to the randomness of the deep learning procedure keeping in mind that with a inbalanced dataset a big variation of a single class may be also meaningless since it depends on the total number of sample of such class. Finally the number of learnable parameters is the same of Sparse Fusion version since the fusion technique does not add weights to learn but it is a simple mathematical procedure.

	Global acc.	Mean acc.	IoU
UN	76.29	66.81	54.36
UNR	76.26	66.92	54.56
ENT	76.91	67.61	55.21
ENTR	76.85	67.59	55.14
UN+ENT	76.56	67.73	55.03
UNR+ENT	76.38	66.75	54.69
Original	76.40	66.93	54.74

Table 5.7: 13-classes, Only Encoding network, general results

	UN	UNR	ENT	ENTR	UN+ENT	UNR+ENT
Class0	75.37	73.50	77.84	73.87	77.75	73.75
Class1	52.70	52.86	54.51	56.06	53.76	54.31
Class2	70.18	75.68	71.10	71.50	73.22	71.99
Class3	71.96	69.38	71.18	71.16	69.86	73.63
Class4	75.51	74.07	74.45	74.00	75.86	76.57
Class5	96.75	97.11	97.05	97.08	96.87	96.83
Class6	58.45	55.81	57.11	58.84	57.44	57.21
Class7	72.29	71.57	76.35	76.24	72.32	73.33
Class8	50.85	50.23	49.91	55.10	55.84	49.61
Class9	91.26	92.62	92.01	91.79	91.43	91.08
Class10	69.03	66.11	68.02	66.45	67.21	66.73
Class11	48.88	51.06	53.05	48.95	50.26	47.25
Class12	35.27	40.00	36.36	37.68	38.71	35.49

Table 5.8: 13-classes, Only Encoding network, classwise accuracy

	Global acc.	Mean acc.	IoU
UN	68.76	47.24	35.82
UNR	68.66	46.17	35.59
ENT	68.06	44.65	34.05
ENTR	69.12	47.38	36.20
UN+ENT	68.76	46.67	35.54
UNR+ENT	68.44	45.61	34.90
Original	68.76	46.42	35.48

Table 5.9: 40-classes, Only Encoding network, general results

	UN	UNR	ENT	ENTR	UN+ENT	UNR+ENT
Class0	91.65	92.18	92.86	91.53	91.75	91.75
Class1	96.92	96.97	96.66	96.58	96.67	96.73
Class2	70.10	70.00	67.61	71.82	71.80	72.28
Class3	77.97	73.43	76.60	80.92	76.45	74.66
Class4	69.76	68.97	72.57	72.86	69.89	73.21
Class5	73.40	73.91	75.38	72.34	74.80	71.15
Class6	52.22	51.57	58.69	50.98	52.67	48.71
Class7	27.13	29.73	23.97	26.29	25.15	28.73
Class8	58.71	55.80	56.10	57.26	53.96	56.82
Class9	51.09	57.63	45.60	59.74	56.29	57.54
Class10	71.12	70.35	69.21	69.72	70.22	70.47
Class11	NaN	NaN	NaN	NaN	NaN	NaN
Class12	61.32	63.48	56.71	61.80	64.19	57.87
Class13	20.54	20.30	20.31	18.05	20.69	26.91
Class14	16.13	16.26	21.15	14.06	16.01	7.98
Class15	56.79	58.39	56.36	60.25	55.72	57.02
Class16	57.94	58.45	37.04	46.30	56.10	58.17
Class17	51.70	53.40	55.46	54.60	48.85	49.69
Class18	37.50	32.69	51.09	41.68	33.09	29.07
Class19	33.87	24.38	24.71	39.88	25.74	27.31
Class20	23.92	17.78	18.33	23.67	19.14	21.64
Class21	NaN	NaN	NaN	NaN	NaN	NaN
Class22	22.51	13.72	17.04	21.44	21.08	18.64
Class23	NaN	NaN	NaN	NaN	NaN	NaN
Class24	48.34	44.22	49.68	55.21	48.54	51.74
Class25	NaN	NaN	NaN	NaN	NaN	NaN
Class26	25.19	27.06	24.14	22.23	28.80	25.43
Class27	37.96	41.20	21.94	32.72	37.56	26.97
Class28	NaN	NaN	NaN	NaN	NaN	NaN
Class29	32.08	20.82	15.12	26.00	25.76	27.88
Class30	38.72	37.66	37.86	44.29	40.47	33.21
Class31	42.86	40.61	46.26	45.17	45.11	41.93
Class32	73.90	75.54	63.79	71.66	71.13	68.50
Class33	NaN	NaN	NaN	NaN	NaN	NaN
Class34	42.84	46.47	42.74	47.80	45.18	45.53
Class35	43.65	43.27	39.74	38.61	47.95	42.22
Class36	2.24	0.54	1.90	1.31	1.19	0.80
Class37	18.47	15.80	13.79	21.93	18.24	15.89
Class38	17.03	14.91	7.23	10.80	14.55	12.91
Class39	60.56	62.34	60.49	61.28	62.18	61.43

Table 5.10: 40-classes, Only Encoding network, classwise accuracy

5.2.3 A Parallel Autoencoder

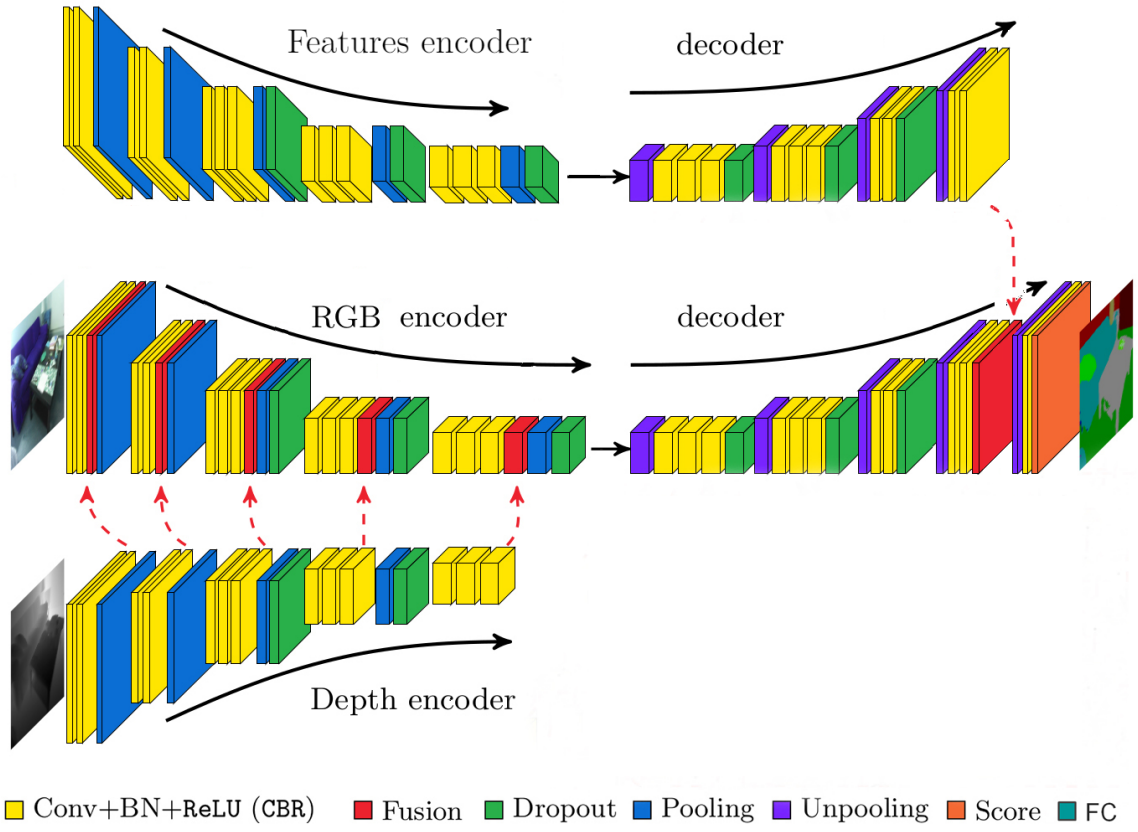


Figure 5.5: Use of a Parallel Autoencoder to fuse 3DEF' features in FuseNet

As shown in Fig. 5.5, we tried to extract some meaningful information through an encoding-decoding structure. The separation of the two branches allowed us to elaborate feature sets and RGB-D separately and it is a direct variation of the Only Encoding model: in this case in fact we have a decoder branch that further elaborate information. Also in this case we took clearly inspiration from the original structure of FuseNet by using the autoencoder method but we tried to merge the decoded information near the end of the network to separate as much as possible what is obtained from the RGB-D and from the 3DEF' feature sets. As in FuseNet, the encoder should help to compress information by keeping only the relevant one and the decoder expand it in a usable way and the sparse fusion technique is used to merge this brought information to the original FuseNet. A different processing of

the feature sets will presented in Section 5.2.4.

In this case we can see again that ENT and ENTR sets are not helping the final scores but in all other cases, for both mapping, we can notice how this procedure works in a better way in any case with respect to (almost) all the performance metrics considered. The main drawback of this crafted network is the number of parameters that is 66% more with respect to the original FuseNet making it the biggest model considered.

	Global acc.	Mean acc.	IoU
UN	76.50	67.02	54.74
UNR	76.72	67.99	55.41
ENT	76.16	67.31	54.06
ENTR	76.28	66.66	54.25
UN+ENT	76.77	67.75	55.27
UNR+ENT	76.65	67.66	55.01
Original	76.40	66.93	54.74

Table 5.11: 13-classes, Parallel AE network, general results

	UN	UNR	ENT	ENTR	UN+ENT	UNR+ENT
Class0	73.88	75.51	77.47	77.86	78.19	74.99
Class1	51.71	52.63	55.06	54.59	54.12	51.54
Class2	72.54	69.95	69.27	69.73	70.11	70.67
Class3	69.93	70.59	70.37	70.38	69.46	71.02
Class4	75.36	77.66	78.94	75.50	77.85	78.93
Class5	97.05	97.09	97.49	97.16	97.11	96.88
Class6	56.93	57.23	57.00	58.30	59.35	59.57
Class7	76.16	75.99	69.17	70.51	74.69	76.95
Class8	52.16	57.13	55.63	58.59	54.69	56.90
Class9	92.82	91.81	89.09	90.85	92.11	91.43
Class10	67.44	67.97	72.35	67.76	66.25	67.22
Class11	49.34	49.29	52.97	45.81	52.42	52.53
Class12	35.95	40.98	30.22	29.55	34.37	30.87

Table 5.12: 13-classes, Parallel AE network, classwise accuracy

	Global acc.	Mean acc.	IoU
UN	69.02	46.27	36.10
UNR	68.94	47.36	36.38
ENT	67.35	42.88	33.24
ENTR	67.40	41.84	32.43
UN+ENT	68.77	47.06	35.89
UNR+ENT	68.63	46.51	35.55
Original	68.76	46.42	35.48

Table 5.13: 40-classes, Parallel AE network, general results

	UN	UNR	ENT	ENTR	UN+ENT	UNR+ENT
Class0	92.53	92.57	90.48	92.42	91.05	92.01
Class1	96.54	96.58	96.32	96.63	96.81	96.87
Class2	69.75	70.15	71.49	71.45	71.86	71.89
Class3	76.20	77.10	73.92	77.69	78.05	75.97
Class4	72.66	73.55	71.76	70.19	73.64	71.48
Class5	76.60	77.56	71.30	77.06	75.09	71.90
Class6	50.42	52.08	51.66	56.19	55.84	49.77
Class7	30.79	28.70	30.24	23.57	30.44	24.57
Class8	57.08	55.99	53.34	49.75	54.53	50.12
Class9	48.90	47.93	45.27	35.98	50.83	52.16
Class10	70.21	69.06	68.78	65.36	69.98	71.05
Class11	NaN	NaN	NaN	NaN	NaN	NaN
Class12	62.64	60.48	57.70	56.09	61.92	64.84
Class13	22.48	28.03	19.73	19.65	20.82	27.95
Class14	15.66	14.41	12.17	15.32	14.22	20.38
Class15	58.10	61.59	61.63	54.29	62.11	59.01
Class16	54.16	55.03	40.62	35.31	53.60	54.48
Class17	46.78	54.12	45.40	45.32	53.08	54.60
Class18	30.16	34.78	32.01	35.76	31.32	38.95
Class19	26.51	30.54	38.84	24.39	31.96	28.54
Class20	19.94	23.61	14.13	19.16	23.76	16.35
Class21	NaN	NaN	NaN	NaN	NaN	NaN
Class22	13.13	20.02	11.41	19.85	17.64	16.67
Class23	NaN	NaN	NaN	NaN	NaN	NaN
Class24	42.55	52.31	44.98	29.82	53.64	47.48
Class25	NaN	NaN	NaN	NaN	NaN	NaN
Class26	32.31	21.85	31.47	14.66	22.41	30.60
Class27	30.27	40.57	11.73	27.89	36.09	40.65
Class28	NaN	NaN	NaN	NaN	NaN	NaN
Class29	30.54	29.17	26.20	16.55	32.91	31.79
Class30	43.72	33.27	27.98	25.66	38.82	33.96
Class31	35.60	41.65	36.03	37.45	41.06	36.15
Class32	69.61	72.35	52.68	62.75	74.84	70.91
Class33	NaN	NaN	NaN	NaN	NaN	NaN
Class34	46.43	50.76	39.16	39.03	45.19	40.46
Class35	51.26	49.78	34.99	35.70	41.87	44.79
Class36	1.38	2.84	1.14	0.72	2.43	2.42
Class37	20.05	18.31	19.27	18.74	18.58	16.30
Class38	14.84	14.06	11.89	10.01	13.68	15.41
Class39	63.52	59.32	62.13	62.05	59.91	61.01

Table 5.14: 40-classes, Parallel AE network, classwise accuracy

5.2.4 Direct Fusion

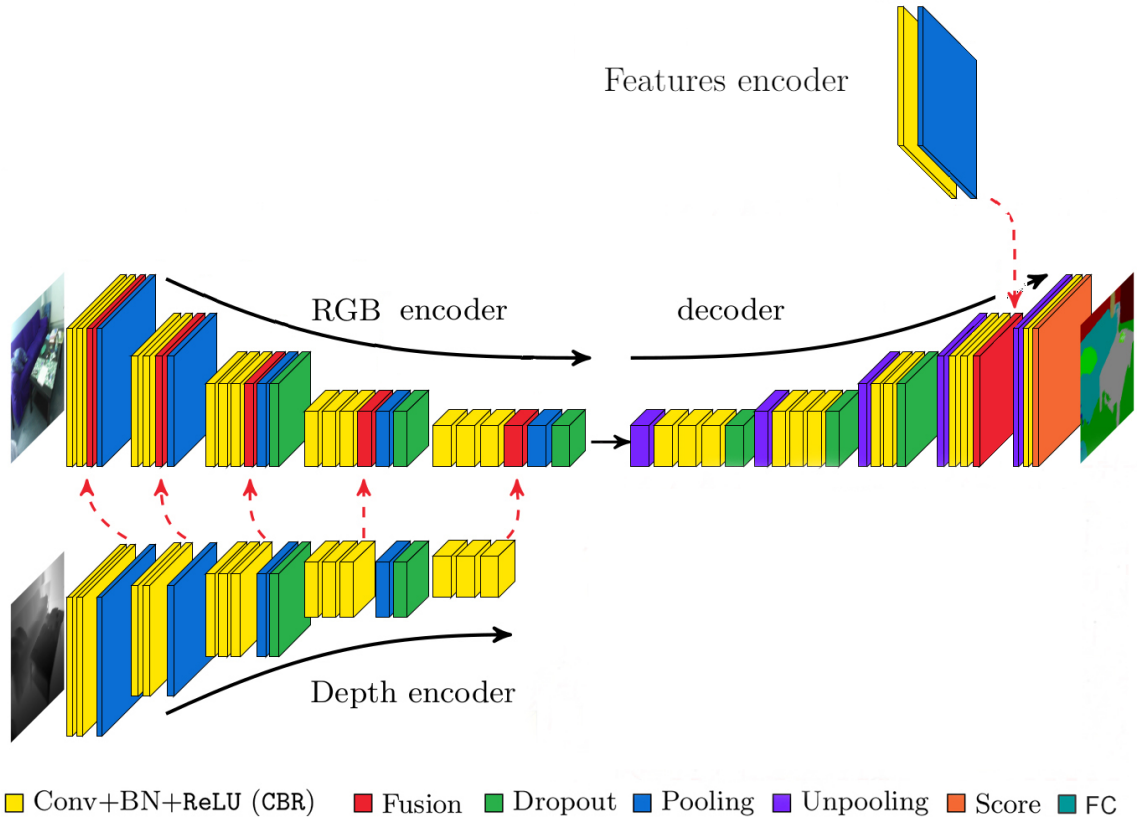


Figure 5.6: Direct Fusion of 3DEF' features in FuseNet

Lastly we tried also to directly incorporate features inside FuseNet without applying to them too much elaboration. Since we did not want to change the base FuseNet RGB-D structure for now we avoided to concatenate them with the depth maps. We instead opted to directly embed the 3DEF' feature sets in the decoder side where we could directly insert our 2D-projected vectors inside the network. We tried various variations that comprises for example concatenation with the decoded features of FuseNet or prior elaboration with various stacks of fully connected layers and then merging the result with the extracted features. Almost all of them ruined the performance of the segmentation probably because the size of the brought information was too big to be elaborated with small convolutional structures. In the end we found out that with a small processing with a CBR layer a pooling and then

a sparse fusion of the obtained features, as shown with the schematic depicted in Fig. 5.6, we managed to obtain a very small increase of the number of parameters (less than 1‰) and still a slight improvement in the performances as demonstrated in the following tables. This improvement, even if slight is comparable to what we have seen so far with the other proposed networks and, also in this case, the use of ENT and ENTR sets tend to go little worse with respect to the other sets if we consider the whole set of metrics. Notnethenless, the difference is minimal and so, as stated previously, the difference may be linked to the stochastic nature of the deep learning methods.

	Global acc.	Mean acc.	IoU
UN	76.61	67.59	54.94
UNR	76.61	67.58	54.99
ENT	76.17	67.26	54.14
ENTR	76.71	67.71	54.91
UN+ENT	76.92	68.21	55.49
UNR+ENT	76.46	67.52	54.79
Original	76.40	66.93	54.74

Table 5.15: 13-classes, Direct Fusion network, general results

	UN	UNR	ENT	ENTR	UN+ENT	UNR+ENT
Class0	76.34	78.24	85.09	78.03	76.22	72.65
Class1	54.06	51.71	49.70	53.63	55.25	56.05
Class2	70.96	69.21	68.93	68.59	72.28	72.05
Class3	69.61	69.10	71.47	72.23	70.21	69.51
Class4	75.81	75.72	74.49	78.33	74.83	74.40
Class5	97.08	97.14	96.35	97.10	97.20	96.69
Class6	59.29	55.98	60.89	55.79	60.59	59.34
Class7	72.99	75.72	71.15	74.64	74.22	71.22
Class8	52.76	51.83	53.35	55.67	53.69	53.82
Class9	91.56	92.63	90.96	90.70	90.91	90.89
Class10	69.93	68.51	64.38	71.34	70.87	70.22
Class11	52.25	53.56	47.74	46.62	53.63	55.82
Class12	36.04	39.12	39.86	37.62	36.83	35.15

Table 5.16: 13-classes, Direct Fusion network, classwise accuracy

	Global acc.	Mean acc.	IoU
UN	68.72	46.86	35.86
UNR	68.94	46.69	35.88
ENT	68.13	44.71	34.71
ENTR	68.41	45.61	35.57
UN+ENT	69.00	47.64	36.62
UNR+ENT	69.09	46.99	36.07
Original	68.76	46.42	35.48

Table 5.17: 40-classes, Direct Fusion network, general results

	UN	UNR	ENT	ENTR	UN+ENT	UNR+ENT
Class0	91.45	91.53	92.37	92.15	91.84	91.28
Class1	96.55	96.57	96.80	96.22	97.03	96.88
Class2	70.85	72.47	71.21	69.17	71.23	70.60
Class3	76.65	76.32	77.48	77.24	76.86	76.88
Class4	69.17	72.17	71.48	70.51	72.65	72.38
Class5	75.70	73.96	73.72	72.44	72.92	73.15
Class6	54.39	51.60	52.47	50.55	55.04	51.86
Class7	25.15	27.35	23.11	24.23	27.25	26.68
Class8	54.98	47.11	51.84	55.25	55.77	60.23
Class9	50.19	58.05	46.47	40.65	55.30	49.93
Class10	73.01	70.48	66.21	71.79	70.45	69.88
Class11	NaN	NaN	NaN	NaN	NaN	NaN
Class12	61.63	70.13	59.31	64.85	65.57	64.00
Class13	23.88	22.24	23.37	23.03	20.77	18.51
Class14	16.26	18.17	16.88	18.75	21.92	16.55
Class15	57.76	60.56	55.72	57.33	63.78	57.78
Class16	62.79	61.38	52.52	44.26	54.02	55.46
Class17	51.50	51.23	52.39	51.46	47.55	52.75
Class18	35.82	34.06	27.38	36.64	32.44	39.78
Class19	35.77	29.70	25.07	27.12	32.76	26.25
Class20	18.79	23.82	20.90	15.30	23.07	17.99
Class21	NaN	NaN	NaN	NaN	NaN	NaN
Class22	19.94	17.78	17.11	18.84	13.21	20.23
Class23	NaN	NaN	NaN	NaN	NaN	NaN
Class24	45.34	49.18	45.11	45.49	41.69	42.94
Class25	NaN	NaN	NaN	NaN	NaN	NaN
Class26	23.68	21.44	23.93	34.15	20.82	24.36
Class27	35.49	35.45	27.34	39.36	40.48	43.50
Class28	NaN	NaN	NaN	NaN	NaN	NaN
Class29	30.80	27.73	25.84	28.63	44.29	30.05
Class30	41.46	33.24	35.11	32.67	45.28	44.75
Class31	48.85	43.88	46.32	40.98	44.36	38.72
Class32	70.94	65.53	64.80	71.85	71.90	73.13
Class33	NaN	NaN	NaN	NaN	NaN	NaN
Class34	46.15	47.43	43.32	39.79	46.46	44.39
Class35	31.08	39.29	38.30	42.89	47.93	44.27
Class36	1.74	2.21	1.16	1.19	1.00	2.16
Class37	18.64	20.99	18.11	20.57	21.55	21.70
Class38	15.77	12.83	15.64	11.00	12.78	13.68
Class39	61.10	61.62	61.39	64.44	59.93	65.00

Table 5.18: 40-classes, Direct Fusion network, classwise accuracy

5.3 Reducing the Size of the Networks

After these attempts that did not manage to obtain big improvements with respect the original FuseNet version as last test we tried to reduce the networks size, and hence the number of parameters of the architectures. We thought about this firstly because we wanted to reduce the computational load of such a big network and more notably to check if we in some way saturated the performance over the NYUv2 dataset. In fact, we can notice that the incorporated features effectively bring some improvements: this is noticeable in most of the combinations networks-feature sets apart for few exceptions. The thing is that this improvement is very small with respect to what could be achieved since we are very far from 100% accuracy or IoU. We then thought that by reducing the size of the original FuseNet network we would ruin its performance but we may see the effectiveness of our features.

To achieve this we removed a series of layers both in encoder, and accordingly, in the decoder side. The cut has to be made in the center of the structure specularly because of the autoencoder structure. We experimented by slicing all the networks, the original and the proposed ones, in three main points:

1. after the first dropout layer
2. after the third fusion layer
3. after the second dropout layer.

We will report only the results of the cut made after the first dropout layer since the one made after the third fusion layer essentially behaves equally and the one made after the second dropout layer does not have meaningful parameters reduction and performance differences with respect to the original network.

As can be seen in Tab. 5.19, depicting the number of parameters before and after the cut for a 40-classes mapping, we can see that we have a meaningful reduction of them. As expected the accuracy obtained by the training procedure is lower than the original ones: lower number of parameters implies a lower possibility to correctly elaborate the data. What is not expected is that some combinations of features and networks effectively bring up the classification score as can be seen in tables 5.20 and 5.21. As foreseen the original version of FuseNet, reduced with this cut, performs way worse with respect to the full network in both 13- and 40-classes

mappings: this drop is around 8-9 % in both cases for global accuracy, 12% for both mean accuracy and IoU. Despite this we can see that by using our Sparse Fusion and Only Encoding version of FuseNet with ENT feature set we manage to obtain around 3% gain in all three metrics. Even if we consider single network model the two features set with ENT perform in a better way with respect to the one with only the UN. We link this behaviour not to the number of parameters but to the incorporated information: if it were a mere reason of size of the network the Parallel AE should be the best performing one.

Surely we did not achieve the performances of the original network but we managed to reduce the number of parameters by 8 times and to demonstrate that the extracted features are useful. This reduction implies a lower computational costs: the reduced models present a reduction of around one-third of the per-epoch training time as shown in Tab. 5.19. Moreover, what seemed to be the worse choice of such features, so the ENT ones, since in most of the cases they performed slightly worse, in this case it is the best one overall. In our opinion these results points out also the likely saturation of the performances for the NYUv2 dataset with FuseNet and its variation and the usefulness of the brought 3DEF' features.

Network	Original		Reduced	
	Param	Epoch time	Param	Epoch time
Sparse Fusion	58.908	125	6.968	83
Only Encoding	58.908	125	6.968	83
Direct Fusion	44.224	104	5.269	70
Parallel AE	73.591	143	8.666	94
FuseNet	44.173	90	5.218	60

Table 5.19: Number of parameters (in M) for original and reduced networks. Sample epoch time in seconds using a NVIDIA® Tesla T4.

Network	Dataset	Global acc.	Mean acc.	IoU
Sparse Fusion	UN	67.92	56.95	43.73
	ENT	**71.20**	**60.35**	**47.79**
	UN+ENT	67.83	56.66	43.59
Only Encoding	UN	67.77	56.52	43.46
	ENT	70.11	58.42	46.12
	ENT+UN	68.03	57.06	43.80
Parallel AE	UN	67.82	56.31	43.38
	ENT	68.38	55.89	43.90
	UN+ENT	68.10	57.11	43.87
Direct Fusion	UN	67.82	56.36	43.47
	ENT	68.02	54.98	43.10
	UN+ENT	67.76	56.62	43.43
FuseNet Reduced	Original	68.01	56.50	43.70

Table 5.20: 13-class, training with reduced networks, general results

Network	Dataset	Global acc.	Mean acc.	IoU
Sparse Fusion	UN	58.73	32.11	23.02
	ENT	**62.28**	**36.84**	**26.51**
	UN+ENT	59.04	33.83	24.00
Only Encoding	UN	59.03	32.77	23.34
	ENT	60.78	34.44	24.79
	UN+ENT	59.05	33.13	23.71
Parallel AE	UN	58.95	32.68	23.30
	ENT	59.68	33.59	23.82
	UN+ENT	59.12	33.14	23.53
Direct Fusion	UN	59.02	32.71	23.40
	ENT	59.52	32.91	23.44
	UN+ENT	59.31	32.69	23.50
FuseNet Reduced	Original	59.02	33.00	23.42

Table 5.21: 40-class, training with reduced networks, general results

Chapter 6

From FuseNet to 3DEF

6.1 Input

6.1.1 The Data

For 3DEF we will use again the NYUv2 dataset over 13- and 40-classes mappings, with both elaborated and raw depths. A difference from the original algorithm is that we had to disable both filters.

6.1.2 The Features

With 3DEF the reasoning is the opposite with respect to the previous case. Here we will work with pointclouds, trying to incorporate features from RGB-D data. These features comes from FuseNet trained over 13- and 40-classes NYUv2 depending on which output we want from the forests.

The projection from 2D to 3D is pretty simple: we start with the 3DEF preprocessing phase, we identify the clusters and, given the features maps from FuseNet, for each of them we average the features that falls inside the cluster's area. To maximize the number of points that we can use to do this average procedure we disabled also the outlier filter as well as the bilateral one. In this way we are able to obtain a single feature vector for each cluster. The main problem of this approach is that we cannot take features at the end of the encoding part where FuseNet managed to process and extract the useful information but we need to take them at the end of the decoder. This is because we need a 1-to-1 mapping between each 2D and

3D points to be able to obtain a way to correctly map them in the final clusters. Taking the feature vectors in the middle of FuseNet would make it impossible to correctly remap them to the corresponding 3D point. Furthermore, activation windows cannot be used: they are squared and so once mapped back to the originating pixel a given patch would overlap between clusters. A single value would correspond to multiple clusters without a way to discriminate between them. These problems are inherited from the different type of input data and learning method: by using RGB-D and pointcloud FuseNet and 3DEF reasons over different representation and learning structures to obtain the same results but in many aspects these are not well matched. The autoencoder structure works by encoding and elaborating information to obtain a meaningful representation in the middle of the network just prior to the decoder: because of that if one wants to extract features should work either at the end or after the encoder section.

In the end, because of all these reasons, we chose the final section of FuseNet to extract these features to be able to use this information inside 3DEF. We selected four places located in the final part of the network to have exactly the wanted size and be able to project them back in the 3D space. These, as depicted in Listing 6.1 are comprised of a convolutional, batch normalization, ReLU and another convolutional layers. We choose to put us after each one of these layers obtaining feature vectors of length 64 in all but the last case where the feature vector has the same length of the number of classes. The overall size of the output size is $width \times height \times num\_features$

```
self.CBR1_RGB_DEC = nn.Sequential (
                        nn.Conv2d(64, 64, kernel_size=3, padding=1),
                        nn.BatchNorm2d(64, momentum= batchNorm_momentum),
                        nn.ReLU(),
                        nn.Conv2d(64, num_labels, kernel_size=3, padding=1),
                        )
```

Listing 6.1: Final Block of FuseNet

To import these features inside the learning algorithm we saved them in a simple *csv* format and adapted the initial code to accept and work by doing tests over them. These tests were carried out simply by using the same technique with a threshold value applied with Unary features, as explained before.

6.2 Training and Results

Here we present the results of the training both for raw and elaborated depth versions of NYUv2 alongside our features from FuseNet. These, as explained before, are extracted in four different points and we number them from the leftmost one (1) to the rightmost (4): so V1 will indicate the features extracted after the first convolutional layer, V2 after the batch normalization, V3 after the ReLU layer and finally V4 if they are extracted after the last convolutional. We will point out if FuseNet is trained over raw depth with an optional `r` after the version of the extraction point. For example V1r will indicate FuseNet trained with raw depths and features extracted after the first convolutional layer.

During our tests we noticed that changing the size of the input do not change the results in a notable way: therefore, we opted to use only the smaller size to test the 3DEF. All forests have been trained with 40 trees and a max of 20-levels depth with default threshold parameters. The features for each couple depth-mapping have been extracted from FuseNet trained over the same dataset.

Firstly we compare all the extraction points in the case of 13-classes mapping and after that we will consider the 40-classes case.

6.2.1 13 Classes

6.2.1.1 Depth Elaborated

In this case the depth of the NYUv2 dataset are elaborated, so with the holes filled. It can be clearly seen in Tab. 6.1 in general adding features gives us slight worse results in all the metrics. Some exceptions are linked to particular couples of dataset/classes as can be seen in Tab. 6.2.

Probably these results are due to the fact that, as shown also as example in Fig. 6.1, Entangled features still dominates the learning procedure. Deep features are used during training but cannot push forward the accuracy of the segmentation. The insertion of such features from FuseNet reduced the number split that has to be made in 20th level, effectively speeding up computation, but with a slight cost of overall accuracy. Sure is that there is an intrinsic random factor with this kind of learning, maybe even more prominent with respect to deep learning methods, but the considerations made won't change.

Furthermore, we cannot notice any substantial difference between FuseNet trained over raw or elaborated depth or between the various extraction point.

	Global acc.	Mean acc.	IoU	Tree depth
V1	55.23	47.87	61.04	20
V1r	55.67	48.82	61.8	20
V2	55.51	48.41	61.55	20
V2r	55.23	48.09	60.91	20
V3	55.59	48.33	61.15	20
V3r	54.75	47.99	60.48	20
V4	54.79	48.26	61.1	20
V4r	55.38	48.68	61.52	20
Original	55.77	48.95	62.18	20

Table 6.1: 13-classes, depth elaborated, 3DEF trained with and without features from FuseNet, general results

	Original	V1	V1r	V2	V2r	V3	V3r	V4	V4r
Class0	65.62	67.65	66.24	66.33	65.0	65.76	66.03	65.0	61.4
Class1	9.92	10.76	9.49	9.81	10.34	10.24	10.17	10.92	10.14
Class2	54.27	54.03	54.72	54.76	53.81	54.47	53.93	54.97	55.03
Class3	32.31	30.63	31.31	30.59	30.05	30.6	29.75	29.6	31.11
Class4	71.15	69.95	70.75	69.96	68.78	71.42	69.53	69.36	71.07
Class5	96.67	96.59	96.53	96.58	96.56	96.68	96.61	96.61	96.51
Class6	25.23	24.65	24.73	24.22	24.09	25.23	24.38	25.46	25.23
Class7	60.72	60.55	65.1	63.37	63.49	61.97	61.5	62.5	63.08
Class8	26.77	22.97	25.81	25.08	21.09	23.19	23.01	24.84	26.99
Class9	74.23	74.26	74.53	74.99	74.93	75.68	72.98	72.96	74.14
Class10	41.15	41.0	41.08	39.63	41.02	39.02	41.91	39.5	41.25
Class11	49.56	44.24	45.67	45.67	44.76	44.61	46.01	45.58	46.17
Class12	28.71	25.05	28.72	28.4	31.32	29.39	28.12	30.09	30.72

Table 6.2: 13-classes, depth elaborated, 3DEF trained with and without features from FuseNet, classwise accuracy

6.2.1.2 Depth Raw

Here the forests are trained over raw depths, which is also the classical configuration presented in the paper. By comparing tables 6.1 and 6.3 it is clear how this version performs better with respect to the one with elaborated depths but the other considerations do not change. We do not have improvements brought by the introduction of new features from the FuseNet architecture.

	Global acc.	Mean acc.	IoU	Tree Depth
V1	59.96	50.75	68.99	20
V1r	59.7	50.33	68.8	20
V2	60.07	50.91	69.75	20
V2r	60.21	50.65	69.41	20
V3	59.84	50.53	69.14	20
V3r	60.26	50.79	69.75	20
V4	59.99	50.91	69.55	20
V4r	60.47	50.91	69.99	20
Original	60.32	51.28	70.06	20

Table 6.3: 13-classes, depth raw, 3DEF trained with and without features from FuseNet, general results

6.2.2 40 Classes

In these cases the considerations are the same, even if in some particular combination with 40 classes we perform worse with respect to the original version. In tables 6.5 to 6.8 there are detailed results. Overall this method, in agreement with FuseNet's results, perform worsen with the 40-classes version with respect to the 13-classes one for the same reasons of inherited precision and crowdness of the classes in the various images.

	Original	V1	V1r	V2	V2r	V3	V3r	V4	V4r
Class0	67.43	68.46	67.36	68.2	67.17	66.57	67.06	66.17	65.6
Class1	12.91	12.56	12.3	12.67	12.91	12.98	12.83	12.36	12.86
Class2	58.67	54.68	55.09	55.15	55.26	55.07	56.33	56.36	59.62
Class3	32.63	32.78	36.35	33.33	32.58	32.97	34.01	34.06	36.06
Class4	82.78	82.14	81.96	82.26	77.89	81.75	82.03	83.62	79.88
Class5	98.29	98.17	98.21	98.18	98.21	98.21	98.25	98.27	98.17
Class6	26.61	25.58	26.76	26.64	26.13	26.48	26.59	26.94	27.13
Class7	56.37	60.27	53.51	57.11	61.26	56.53	57.93	59.45	59.52
Class8	49.29	46.07	42.79	46.84	45.72	46.5	48.06	41.02	46.8
Class9	82.18	81.92	80.62	81.85	82.51	81.81	81.96	81.96	81.88
Class10	27.0	26.02	26.98	26.66	27.57	27.97	28.44	25.8	25.63
Class11	50.03	46.37	47.06	48.25	47.64	45.43	46.05	49.1	45.75
Class12	22.39	24.74	25.35	24.71	23.6	24.61	20.72	26.77	22.95

Table 6.4: 13-classes, raw depth, 3DEF trained with and without features from FuseNet, classwise accuracy

	Global acc.	Mean acc.	IoU	Tree Depth
V1	36.15	24.71	22.84	20
V1r	35.97	25.13	22.93	20
V2	35.68	25.41	23.37	20
V2r	36.71	25.51	23.48	20
V3	36.36	25.2	23.02	20
V3r	36.06	25.6	23.22	20
V4	36.86	26.47	24.61	20
V4r	36.55	26.0	23.85	20
Original	37.0	26.91	24.79	20

Table 6.5: 40-classes, elaborated depth, 3DEF trained with and without features from FuseNet, general results

	Global acc.	Mean acc.	IoU	Tree Depth
V1	40.45	27.05	26.78	20
V1r	41.15	27.74	27.35	20
V2	40.07	27.85	26.95	20
V2r	40.38	27.2	26.97	20
V3	41.67	27.79	27.49	20
V3r	42.20	28.13	28.08	20
V4	40.69	27.38	27.14	20
V4r	40.92	28.63	28.11	20
Original	42.23	29.86	30.03	20

Table 6.6: 40-classes, raw depth, 3DEF trained with and without features from FuseNet, general results

	Original	V1	V1r	V2	V2r	V3	V3r	V4	V4r
Class0	47.42	49.26	47.31	45.47	49.44	49.65	49.04	47.48	47.67
Class1	83.69	84.9	86.31	81.95	83.8	81.88	83.61	83.48	82.73
Class2	27.37	13.52	16.84	23.42	19.61	26.11	16.5	23.3	24.0
Class3	44.24	43.97	40.8	43.73	45.44	40.82	40.32	45.4	47.08
Class4	25.95	27.62	27.46	27.98	28.01	26.44	29.03	26.29	27.05
Class5	30.53	33.29	34.6	32.63	33.01	30.86	30.79	32.65	31.7
Class6	5.99	6.44	8.48	7.59	6.59	5.34	6.45	9.04	5.95
Class7	18.77	20.29	21.33	22.89	26.44	17.67	18.92	24.5	22.28
Class8	40.85	38.03	42.8	39.71	37.96	36.36	39.04	41.46	38.72
Class9	28.47	20.99	24.02	21.81	21.55	20.91	20.62	25.44	22.43
Class10	29.02	29.74	30.31	29.7	27.05	29.97	28.38	29.76	32.02
Class11	68.16	68.15	65.9	67.7	68.41	69.31	69.09	67.87	67.66
Class12	19.31	13.5	17.79	16.01	17.33	17.96	15.59	18.22	19.17
Class13	13.09	12.73	10.1	9.99	10.48	6.89	9.97	12.34	9.22
Class14	3.08	3.14	2.61	3.58	3.7	4.41	3.73	3.58	4.58
Class15	19.1	21.31	17.8	16.98	22.56	17.52	22.64	19.53	19.2
Class16	24.98	29.21	19.72	22.74	19.81	21.71	25.09	28.47	20.63
Class17	36.67	32.33	32.36	30.98	34.32	34.07	33.37	34.63	31.27
Class18	20.8	9.88	7.43	11.77	7.48	11.0	6.99	13.36	9.56
Class19	49.39	47.36	35.55	50.5	42.37	47.37	47.24	44.03	47.55
Class20	3.07	1.07	0.83	1.62	2.33	0.27	0.51	1.94	1.53
Class21	74.9	75.3	71.9	75.33	74.24	73.29	75.46	72.68	74.34
Class22	7.84	4.74	2.42	6.39	5.1	3.51	5.58	6.69	7.45
Class23	8.07	2.2	0.4	3.76	5.81	3.62	2.34	7.5	4.72
Class24	46.22	29.9	38.44	33.41	35.39	38.81	44.07	33.44	33.06
Class25	36.89	32.32	33.63	31.33	29.07	32.89	31.71	30.81	30.57
Class26	4.27	3.36	1.98	2.04	5.1	2.73	1.53	1.81	2.77
Class27	10.07	12.57	18.78	7.1	18.21	17.16	9.66	8.41	17.85
Class28	2.94	3.17	2.25	2.69	4.56	3.41	2.86	5.19	3.55
Class29	8.58	8.4	39.55	38.71	10.06	23.8	38.64	32.52	29.01
Class30	7.84	5.26	3.75	7.02	6.33	6.8	6.02	6.41	7.44
Class31	57.77	60.33	56.13	46.82	62.83	48.91	54.77	52.08	57.35
Class32	59.29	53.9	59.51	55.66	59.47	60.95	52.87	63.88	56.5
Class33	34.77	29.75	22.59	29.04	31.31	28.68	31.64	31.54	28.79
Class34	30.99	27.63	25.48	27.55	25.39	26.05	28.22	22.21	26.36
Class35	38.64	27.51	32.98	36.14	31.14	35.93	34.32	41.28	43.48
Class36	4.41	1.55	1.56	1.4	5.54	1.89	3.66	5.66	2.39
Class37	0.18	0.5	0.08	0.14	0.09	0.13	0.23	0.06	0.05
Class38	1.75	1.55	2.25	2.33	2.08	1.72	2.47	2.17	1.28
Class39	1.17	1.9	1.0	0.62	1.03	1.06	1.23	1.72	1.03

Table 6.7: 40-classes, elaborated depth, 3DEF trained with and without features from FuseNet, classwise accuracy

	Original	V1	V1r	V2	V2r	V3	V3r	V4	V4r
Class0	50.29	50.02	53.85	51.23	51.69	53.33	54.85	51.79	49.4
Class1	91.23	91.63	89.9	89.81	91.12	91.47	91.62	91.19	92.67
Class2	42.35	33.11	28.34	24.69	29.78	34.28	32.39	30.47	33.36
Class3	51.81	55.37	51.78	52.1	51.76	51.46	55.59	46.85	49.99
Class4	29.98	27.63	29.88	25.26	26.88	22.79	24.7	28.67	25.03
Class5	28.56	23.91	25.07	27.97	23.08	26.61	27.54	25.74	28.36
Class6	18.42	16.69	20.5	10.21	10.0	24.93	18.02	11.41	20.61
Class7	43.8	39.43	43.19	37.61	32.88	45.44	44.7	45.56	39.86
Class8	20.31	17.2	22.17	16.88	18.77	20.58	16.92	21.29	22.87
Class9	27.2	21.22	21.78	22.64	22.44	21.9	21.85	23.1	23.15
Class10	31.19	29.52	29.63	31.22	30.68	30.34	30.27	31.63	29.11
Class11	70.15	71.47	72.22	73.52	74.54	72.2	72.99	74.3	71.88
Class12	17.72	17.33	21.04	20.83	23.83	18.23	21.77	20.65	23.15
Class13	17.46	10.32	10.8	12.83	11.66	11.18	12.75	12.53	10.89
Class14	3.1	3.21	3.02	4.21	3.6	4.9	3.83	4.99	5.35
Class15	24.53	20.19	20.66	16.04	17.85	20.94	19.61	19.96	24.25
Class16	22.56	23.56	18.67	28.16	26.32	32.54	22.16	25.58	29.17
Class17	49.52	43.29	38.84	43.09	42.2	41.7	45.69	49.05	46.79
Class18	29.05	20.97	20.12	23.46	28.38	26.49	19.38	25.31	17.91
Class19	37.14	32.1	32.08	43.37	39.65	30.2	37.0	33.36	37.85
Class20	4.83	2.81	2.18	2.79	1.26	3.14	2.7	2.9	1.48
Class21	83.14	85.88	86.04	84.87	86.26	83.31	93.56	83.01	84.43
Class22	10.86	7.8	6.73	6.49	7.65	6.59	6.89	8.45	6.59
Class23	6.97	1.67	1.57	2.14	1.04	0.44	2.81	4.61	1.8
Class24	31.43	33.79	30.2	30.98	32.67	27.57	34.82	30.26	29.01
Class25	21.79	25.43	28.38	24.26	22.83	28.14	21.6	17.65	27.61
Class26	3.67	2.35	2.3	1.61	5.58	1.96	2.13	1.62	0.86
Class27	21.14	16.29	26.61	32.98	13.67	20.73	30.15	21.1	26.21
Class28	9.33	6.21	3.36	4.76	6.4	6.03	3.16	5.44	4.96
Class29	18.24	1.05	22.56	10.29	13.35	1.33	0.55	0.99	15.6
Class30	10.8	6.37	5.91	8.59	5.31	6.41	6.26	6.75	7.17
Class31	59.86	54.94	54.97	65.25	53.77	56.58	66.1	60.99	61.56
Class32	70.08	67.75	65.38	61.76	56.66	64.19	58.86	60.35	65.32
Class33	45.07	42.43	39.92	41.16	41.43	46.12	42.2	38.37	40.98
Class34	38.19	28.57	28.29	33.9	36.16	27.72	28.51	32.41	35.71
Class35	42.75	46.68	44.36	40.43	39.04	41.63	42.61	38.04	43.14
Class36	6.16	1.73	5.05	3.86	5.47	5.2	6.63	5.96	8.95
Class37	0.3	0.13	0.3	0.27	0.2	0.45	0.12	0.15	0.08
Class38	1.67	0.97	1.05	1.67	1.29	1.46	1.5	1.71	1.12
Class39	1.74	0.87	0.96	0.85	0.92	1.09	0.64	1.02	1.14

Table 6.8: 40-classes, raw depth, 3DEF trained with and without features from FuseNet, classwise accuracy

6.3 Shorten Trees Depth

Similar to what we have done with FuseNet we have thought about the possibility to reduce the depth of the trees, making tests every 2 levels from depth 6 to depth 20. This to check possible saturation of the performances for the method with respect to the dataset and to check the effective utilization of the features.

As said previously we can notice that the brought features are effectively utilized in the learning process as depicted in Fig. 6.1. Performances with forests trained with less trees levels exhibit the same behaviour of the previously seen results even if with expected lower scores: they are all near the same value as shown as example for 13 classes with 40 trees at depth 8 in Tab. 6.9.

	Global acc.	Mean acc.	IoU
Original	52.71	46.64	57.77
V1	51.5	45.38	56.29
V1r	53.02	46.7	57.71
V2	52.64	46.27	57.36
V2r	52.73	45.93	57.18
V3	52.07	45.48	56.2
V3r	51.19	45.92	56.0
V4	51.38	45.57	56.27
V4r	52.9	46.94	58.07

Table 6.9: Forests trained with 40 trees at depth 8, 13-classes, general results

In the end the only improvement that we noticed was a slight reduction of the number of necessary split by using FuseNet's features, not quantifiable due to stochastic nature of the learning method. This is not enough to justify the extraction and elaboration procedure necessary to embed these information inside the 3DEF.

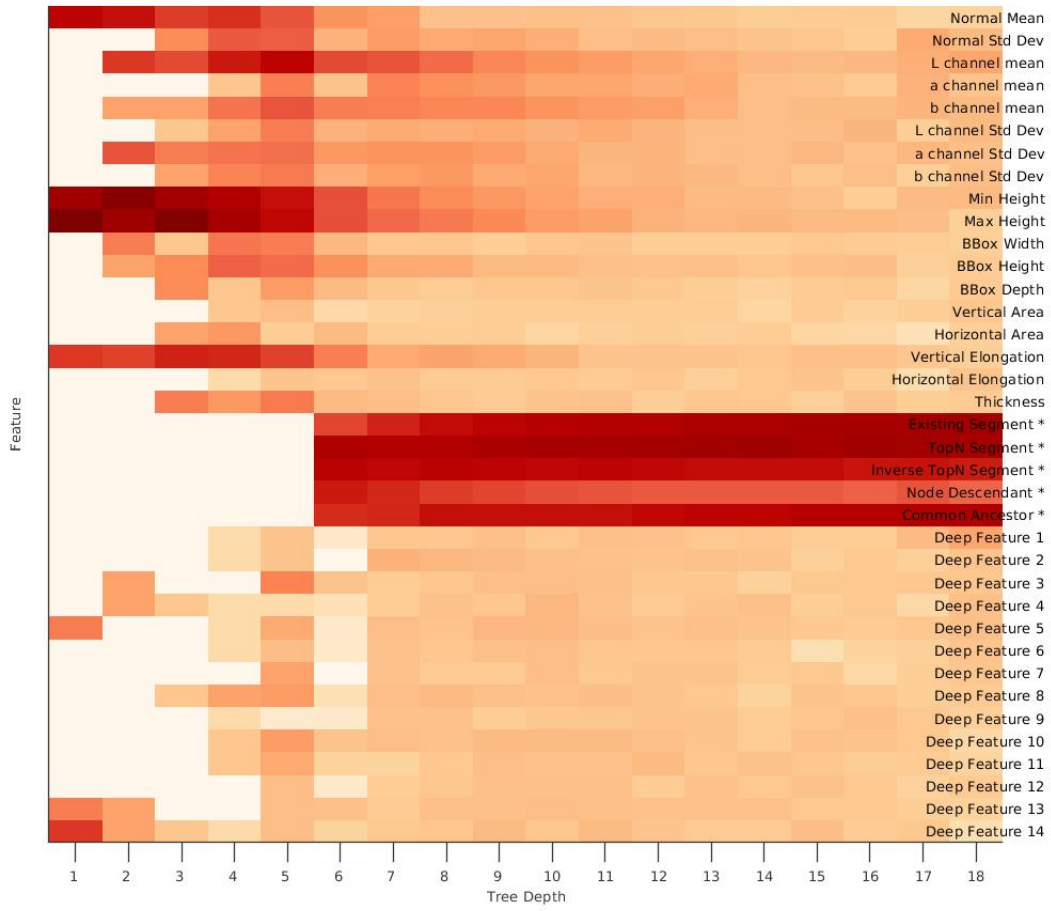


Figure 6.1: Features Importance for each depth level for the whole set of 40 trees, 13 classes trained with V4 features from FuseNet

Chapter 7

Conclusions

In this work we deeply studied FuseNet and 3D Entangled Forests.

Firstly, to better understand the FuseNet architecture, we tried to do fine-tuning and training of it with a subset of the COROMA dataset. By doing so we noticed the importance, during fine tuning, of updating weights in the whole decoding part: this suggest that encoded features absolutely need the decoding side of the network to be used correctly in a segmentation problem.

Starting from this we firstly tried to bring 3DEF features inside FuseNet. To do so we developed four different variations trying to elaborate these features and to incorporate them inside the original network in different ways. Unfortunately, this did not bring the desired results: we noticed that overall the performances were comparable in most of the cases and in a few of them we a very slight improvement was present.

After that we tried to squeeze the original network, obtaining a significant reduction of the number of trainable parameters and hence the power requirements necessary to train the models. Here, differently from the previous case, the brought information effectively helped to improve significantly the performance. This is explainable either because with the original network we reached the maximum achievable with the NYUv2 dataset or because with the NYUv2 dataset per se better results cannot be achieved. Either way we successfully demonstrated that the features used by 3DEF are useful in addition to RGB-D data at least if used within FuseNet. This work can be further developed by considering, for example, that features have been taken as-is and so a normalization procedure may be effective. Also

other combination of features, or different subsets of them may be interesting trial to further test these methodologies. Furthermore, different fusion techniques may be applied for example dense fusion as in [1] or even concatenation either of elaborated features or of "raw" ones within the depth branch. Surely, taking the features used by 3DEF within FuseNet demonstrated its possibilities within this work.

As for 3DEF the work did not bring the same results as the previous case: most of the times the forests that used deep features extracted from FuseNet performed worse than the original version. In this case we think that the main reasons could be either the quality of the information, insufficient to bring notable results, or the quantity. In our opinion bringing 64 features in a system that works with 23 can be overwhelming and, in this sense, dimensionality reduction techniques like Principal Component Analysis (PCA) may be taken into consideration. Furthermore, different features extraction techniques may be applied for example not taking the simple average over the cluster but apply some weight function or taking the average over the voxels.

In all cases the training made with NYUv2 with 13 classes performs in a better way with respect to the 40-classes mapping. As explained previously this is probably linked to the fewer requirements and crowdness of the class segmentation: a fewer number of classes is easier to learn.

To summarize, during this work, we demonstrated that using features extracted from pointclouds and taking consideration of local and spatial characteristics of the cloud and embed this information inside a deep learning method could be of use. Indeed, by bringing 3DEF' features within our models based on FuseNet we managed, in general, to obtain an improvement of the original performance. This is true especially for the combination of Entangled Features and the reduced Sparse Fusion network that shown an improvement between 3 and 4% over all the three considered metrics. To further advance this study, apart from what is introduced above, in future works we will apply the same reasoning with different datasets, like the S3DIS [35] or the SUNRGBD, and try with different deep learning methods like PointNet++.

Bibliography

[1] Caner Hazirbas et al. "FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-Based CNN Architecture". In: *ACCV*. 2016.

[2] Denis Fernando Wolf, Johann Prankl, and Markus Vincze. "Enhancing Semantic Segmentation for Robotics: The Power of 3-D Entangled Forests". In: *IEEE Robotics and Automation Letters* 1 (2016), pp. 49–56.

[3] Dengsheng Lu and Qihao Weng. "A survey of image classification methods and techniques for improving classification performance". In: *International journal of Remote sensing* 28.5 (2007), pp. 823–870.

[4] Ali Borji et al. "Salient object detection: A survey". In: *Computational Visual Media* (2014), pp. 1–34.

[5] Alberto Garcia-Garcia et al. *A Review on Deep Learning Techniques Applied to Semantic Segmentation*. 2017. arXiv: 1704.06857 [cs.CV].

[6] Keisuke Tateno, Federico Tombari, and Nassir Navab. "When 2.5D is not enough: Simultaneous reconstruction, segmentation and recognition on dense SLAM". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)* (2016), pp. 2295–2302.

[7] Keisuke Tateno et al. "CNN-SLAM: Real-Time Dense Monocular SLAM with Learned Depth Prediction". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). DOI: 10.1109/cvpr.2017.695. URL: http://dx.doi.org/10.1109/CVPR.2017.695.

[8] Li Sun, Cheng Zhao, and Rustam Stolkin. "Weakly-supervised DCNN for RGB-D object recognition in real-world applications which lack large-scale annotated training data". In: *arXiv preprint arXiv:1703.06370* (2017).

[9] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[10] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[11] Lyne Tchapmi et al. "Segcloud: Semantic segmentation of 3d point clouds". In: *2017 International Conference on 3D Vision (3DV)*. IEEE. 2017, pp. 537–547.

[12] Angela Dai et al. "Scannet: Richly-annotated 3d reconstructions of indoor scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5828–5839.

[13] Albert Montillo et al. "Entangled decision forests and their application for semantic segmentation of CT images". In: *Biennial International Conference on Information Processing in Medical Imaging*. Springer. 2011, pp. 184–196.

[14] Farzad Husain et al. "Combining semantic and geometric features for object class segmentation of indoor scenes". In: *IEEE Robotics and Automation Letters* 2.1 (2016), pp. 49–55.

[15] Saurabh Gupta et al. "Learning rich features from RGB-D images for object detection and segmentation". In: *European conference on computer vision*. Springer. 2014, pp. 345–360.

[16] Morris Antonello et al. "Multi-view 3D entangled forest for semantic segmentation and mapping". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1855–1862.

[17] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 652–660.

[18] Charles Ruizhongtai Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems*. 2017, pp. 5099–5108.

[19] Ioannis Kostavelis and Antonios Gasteratos. "Semantic mapping for mobile robotics tasks: A survey". In: *Robotics and Autonomous Systems* 66 (2015), pp. 86–103.

[20] Zhengyou Zhang. "Microsoft kinect sensor and its effect". In: *IEEE multimedia* 19.2 (2012), pp. 4–10.

[21] N Namitha et al. "Point cloud mapping measurements using kinect RGB-D sensor and kinect fusion for visual odometry". In: *Procedia Computer Science* 89 (2016), pp. 209–212.

[22] Michael Firman. "RGBD datasets: Past, present and future". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2016, pp. 19–31.

[23] Pushmeet Kohli Nathan Silberman Derek Hoiem and Rob Fergus. "Indoor Segmentation and Support Inference from RGBD Images". In: *ECCV*. 2012.

[24] Camille Couprie et al. "Indoor semantic segmentation using depth information". In: *arXiv preprint arXiv:1301.3572* (2013).

[25] Saurabh Gupta, Pablo Arbelaez, and Jitendra Malik. "Perceptual organization and recognition of indoor scenes from RGB-D images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 564–571.

[26] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. "Sun rgb-d: A rgb-d scene understanding benchmark suite". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 567–576.

[27] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. "Sun3d: A database of big spaces reconstructed using sfm and object labels". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1625–1632.

[28] Allison Janoch et al. "A category-level 3d object dataset: Putting the kinect to work". In: *Consumer depth cameras for computer vision*. Springer, 2013, pp. 141–165.

[29] Angel Chang et al. "Matterport3d: Learning from rgb-d data in indoor environments". In: *arXiv preprint arXiv:1709.06158* (2017).

[30] Wonmin Byeon et al. "Scene labeling with lstm recurrent neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3547–3555.

[31] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. "Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding". In: *Procedings of the British Machine Vision Conference 2017* (2017). DOI: 10.5244/c.31.57. URL: http://dx.doi.org/10.5244/c.31.57.

[32] Radu B Rusu and S Cousins. "Point cloud library (pcl)". In: *2011 IEEE international conference on robotics and automation*. 2011, pp. 1–4.

[33] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: 2005.

[34] Bela Julesz. "Textons, the elements of texture perception, and their interactions". In: *Nature* 290.5802 (1981), p. 91.

[35] Iro Armeni et al. "3d semantic parsing of large-scale indoor spaces". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1534–1543.

[36] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. "Learning deconvolution network for semantic segmentation". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1520–1528.

Websites

[37] Natalie Fletcher. *Classification VS Detection VS Segmentation models: the differences between them and when to use each*. 2019. URL: https://www.clarifai.com/blog/classification-vs-detection-vs-segmentation-models-the-differences-between-them-and-how-each-impact-your-results.

[38] Piotr Dollar. *Learning to Segment*. 2019. URL: https://research.fb.com/blog/2016/08/learning-to-segment/.

[39] Kenneth Dasalla. *EngD Research Project 03: Introduction of Depth Sensors*. 2019. URL: https://zubr.co/engd-research-project-03-introduction-of-depth-reality-research-project-3/.

[40] Anil Sonmez. *Implementation of FuseNet by Anil Sonmez*. 2019. URL: https://github.com/zanilzanzan/FuseNet_PyTorch.