AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

FACULTY OF ELECTRICAL ENGINEERING, AUTOMATICS, COMPUTER SCIENCE AND BIOMEDICAL ENGINEERING

DEPARTMENT OF BIOCYBERNETICS AND BIOMEDICAL ENGINEERING

Master of Science Thesis

Analiza znaczenia informacji o głębokości w rozpoznawaniu oraz segmentacji semantycznej obrazów z wykorzystaniem głębokiego uczenia się

Analysis of significance of depth in recognition and semantic segmentation of images using deep learning

Author:	*Dominika Kwaśny*
Degree programme:	*Biomedical Engineering*
Supervisor:	*Adrian Horzyk, PhD, Prof AGH*

Kraków, 2019

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Contents

1. Introduction 7

1.1. Aim 7

1.2. Scope 8

1.3. Work structure 8

2. Deep learning background 9

2.1. Artificial neural networks 9

2.2. Convolutional neural networks 10

2.2.1. Convolutional layer 11

2.2.2. Pooling layer 11

2.2.3. Dropout 11

2.2.4. Batch normalization 12

2.2.5. Fully connected layer 12

2.2.6. Loss layer 12

3. From image classification to image understanding 13

3.1. Image recognition 13

3.1.1. AlexNet 13

3.1.2. Inception (GoogLeNet) 14

3.1.3. VGG 15

3.2. Semantic segmentation 15

3.2.1. Fully Convolutional Network 16

3.2.2. U-Net 17

3.3. Evaluation metrics 20

3.3.1. Multi-label classification 20

3.3.2. Image segmentation 20

4. Depth information in computer vision systems 23

4.1. Depth acquisition 23

4.2. Depth processing 24
4.3. Current applications and future development of depth sensing 24
4.4. Datasets overview 26
4.5. Including depth in deep learning tasks 28
4.6. Fusion of modalities 28
4.7. HHA depth encoding 29
4.8. Is depth really needed 30
5. Creating experimental dataset 31
5.1. Intel Euclid Realsense camera 31
5.2. Experiment conditions 32
5.3. Image preprocessing 32
5.3.1. Image registration 32
5.3.2. Depth inpainting 36
5.3.3. Depth encoding 36
5.4. Dense semantic labelling 36
5.5. Data split and augmentation 38
6. Experiments and results 41
6.1. Implementation and training details 41
6.2. Image recognition 41
6.3. Semantic segmentation 44
6.4. Discussion of the results 46
6.4.1. Recognition 46
6.4.2. Semantic segmentation 46
7. Future work and conclusion 51
7.1. Future work 51
7.2. Conclusion 51
7.3. Acknowledgments 52
List of Figures 57
List of Tables 59
Appendix A. U-Net fusion variants 60
Appendix B. Per-class results for FCN models 62
Appendix C. Per-class results for U-Net models 64

1. Introduction

Currently, every day brings new advances in the world of computer vision. With new algorithms and more processing power than ever, we are able to analyze images faster and with more confidence. Moreover, the ideas of what we can let to be automatized seem to have no limits. Both in science and business alike, the quest to bring innovation in computer vision is recently fueled by deep learning, a subset of machine learning that uses multiple layers of neural networks arranged together to learn from provided data, achieving levels of correctness comparable with that of humans.

Such advances would not have been possible if it were not for the abundance of data at our disposal. Enormous numbers of photos are uploaded every day to the Internet and are later scraped to fuel deep learning models. At the same time, producers of smartphones advertise models with cameras supported by Artificial Intelligence to let users take better photos than before.

New possibilities also appeared with the rise of consumer-level depth sensors embedded into devices like Kinect. The resulting data are often referred to as RGBD, D standing for depth information. Incorporating this new modality can be especially beneficial to robotic computer vision systems as they can learn to operate based on their surroundings' texture or shape rather than color. Incorporating color- and light-invariant features could result in better image recognition systems, but also those of semantic segmentation where it is expected to recognize not only what is on the image, but where exactly it is located. For this type of task, each pixel in the image needs to be classified.

1.1. Aim

The purpose of this work is to analyze whether including depth information can result in better results in object recognition and semantic segmentation algorithms using deep learning and how such information should be incorporated to provide best results.

1.2. Scope

The presented work encompasses the steps mentioned below:

1. creating a dataset using Intel Euclid camera
2. data preprocessing and labeling images with masks
3. implementation of state-of-the-art architectures for image recognition
4. implementation of state-of-the-art architectures for semantic segmentation
5. investigation and experimentation for incorporating depthmaps into mentioned models
6. comparison and analysis of the results

1.3. Work structure

The work is structured as follows: first, neural networks are briefly introduced with a focus on convolutional neural networks together with various image-related tasks where they are found to be suitable. Network architectures useful for each task are also briefly mentioned. Then, some of the recent advances in image classification are described with the focus on winners of ImageNet competition. Later, two state-of-the-art architectures for semantic segmentation are given a detailed description.

In the following chapter, depth acquisition means are characterized together with their current uses and possibilities to develop applications of depth-sensing further. This chapter contains information on publicly available datasets that contain depth information. The emphasis is placed on those resources that can be of use in semantic segmentation. Additionally, methods of incorporating this modality in deep learning are given.

Subsequent chapters are related to the practical part of the thesis. Chapter 5 provides details on the Intel Euclid device used to gather the images, experiment conditions, and data preprocessing. The next chapters describe deep learning architectures used together with any modifications made along the process and achieved results. Following is a discussion of the results and conclusion.

2. Deep learning background

2.1. Artificial neural networks

The initial idea behind artificial neural networks was inspired by that of the electrochemical activity of neurons, cells that make the brain. In 1943 a neuron's mathematical model was proposed that uses a type of simple linear classifier because it can only fire if a certain threshold is exceeded by neuron's weighted inputs. This makes a neural network simply a set of units and it is those units' properties that characterize network as a whole [1].

In the following years, however, statisticians and Artificial Intelligence developers sought after other properties of neural networks, such as robustness to noise or distributed learning. At the same time, more sophisticated models of brain structures were proposed. For those reasons, it is a bit harder now to find analogies between biology and Artificial Intelligence but the basic concepts can still be explained with respect to this initial model where each unit of the network can be expressed as a weighted sum of its inputs with activation function g:

$$y = g(\sum_{i=0}^{n} w_i x_i + b) \tag{2.1}$$

The activation function is necessary for the network to learn more complex features, expressed by non-linear functions. Without it, the network would behave like a linear regression model. Some of the most popular activation functions are:

$$g(y) = \frac{1}{1+e^y} \text{ - sigmoid} \tag{2.2}$$

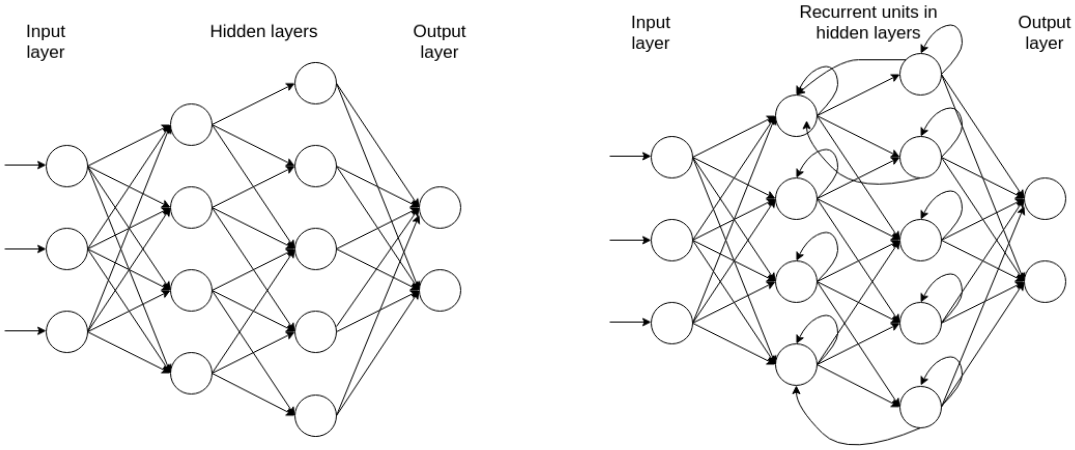
$$g(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}} \text{ - tanh} \tag{2.3}$$

$$g(y) = max(y, 0) \text{ - ReLU} \tag{2.4}$$

$$g(y) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases} \text{ - leaky ReLU} \tag{2.5}$$

The networks are also characterized by the way its units are connected:

- In a feed-forward network shown in Figure 2.1a units are put into a directed acyclic graph, that is - the connections can only have one direction. Such networks are stacked in layers in which neurons in a layer can only be given input from the layer directly before. Layers not connected to the output are referred to as hidden layers [1]. Multiple fully connected layers joined in a feed-forward manner are called multi-layer perceptrons.
- Network with nodes connected in such a way that outputs are fed again to their inputs is referred to as a recurrent network. They are similar to the brain in the sense that they have short-term memory because the response of the network to the given input is determined by its initial state. This one, however, might be determined by previous information fed to the system [1]. A multi-layer version of such networks is shown in Figure 2.1b



(a) Feed-forward network

(b) Recurrent network

Fig. 2.1. Examples of types of unit connections in different neural networks

The main field of application for perceptrons would be in organized, tabular data. Thanks to nonlinear linear activation in the neurons, they spot patterns that would not be linearly separable otherwise. Recurrent networks, in turn, work best applied to problems related to sequence prediction. Perceptrons can also achieve decent results in simple computer vision tasks, such as digits classification, yet they fail to work well on data with a more complex spatial relationship, like images where input data are related in space. As of now, convolutional neural networks deal best with image input.

2.2. Convolutional neural networks

The majority of CNNs that are currently in use are made of several "blocks" that are made of layers of operations such as convolution, activation or normalization. Some of those layers are explained briefly below, but not all blocks have to use all of them.

2.2.1. Convolutional layer

Convolutional neural networks (CNNs) are now considered a standard approach when dealing with image classification problem. Their purpose is to learn specific features using small squared segments of said images. A discrete convolution can be defined as in equation 2.6. Extending it to two dimensions yields equation 2.7, where x is an input image and K is the chosen kernel. A convolution layer is simply a convolution of an input image with a defined kernel or filter K that outputs a feature map. If convolution is done without padding image at the borders, then such a map will be of a smaller size than its input. To obtain the same dimensions, input can be padded with zeros. In ConvNets with multiple layers, the initial ones learn more low-level features as in classical image processing, for example edges, gradient or orientation and gradually learn more complex features [2]. Usually, ConvLayers are activated with ReLU.

$$(f * g) = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \tag{2.6}$$

$$y[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} K[m,n]x[i-m, j-m] \tag{2.7}$$

where

x – input image

K – kernel, for example K = $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$

2.2.2. Pooling layer

To reduce the dimensionality of the feature map and thus computing time, ConvLayer is usually followed by spatial downsampling operation also known as pooling layer. Two-dimensional pooling operation with an element of size n works on image area $n \times n$ leaving only one value from it. Usually, the maximum value of the area is used hence the name max pooling, but there is no limitation to pool minimum or average values should they provide better results. Pooling with an element which size is two reduces the feature map by a factor of two [2].

2.2.3. Dropout

To prevent the net from overfitting, dropout layer can be introduced. The dropout rate of 0.5 means that each neuron may be zeroed before passing to the next layer with 0.5 probability. Thanks to that, each time the network can sample different architecture while sharing weights

between them [3]. While useful, this technique may require more iterations before the network converges.

2.2.4. Batch normalization

Another often included layer is batch normalization, which acts as a weak regularizer on the previous output batch by subtracting the mean and dividing by the standard deviation of the batch.

2.2.5. Fully connected layer

After several blocks made of the above elements, it is necessary to flatten the last output into a feature vector and pass it to a fully connected layer, also activated with ReLU. This layer can learn a non-linear combination of features, that is, in this space it is able to learn to fit a non-linear function [2].

2.2.6. Loss layer

Finally, it is necessary to define a loss function to support learning and penalize when the predicted value has deviated from the correct one. Different loss functions should be defined depending on the task. In binary classification, the sigmoid is typically used:

$$Sigmoid : S(\mathbf{x}) = \frac{e^x}{e^x + 1} \tag{2.8}$$

In the case of non-binary classification, where the object is predicted to have one out of N mutually exclusive labels, this is done by ending the model with a layer which weights are activated with the softmax function, sometimes also a called normalized exponential function. It expresses the probability distribution over a discrete variable of N options. It can be understood as a generalization of probability distribution representation for a binary variable and is calculated as shown in equation 2.9. This way output is normalized to be between 0 and 1 while the sum of probabilities for elements in input vector $\mathbf{x}$ is 1 [4].

$$S(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \text{ for } i = 1, ..., N \tag{2.9}$$

3. From image classification to image understanding

3.1. Image recognition

Image classification or recognition is the task of extracting such features that allow assigning the image one or more labels. Input is classified as a whole, and there is no information about object location. This task appears to be the most widely used application of convolutional neural networks and indeed, obtained results can be considered state-of-the-art in their category.

In the last couple of years, majority of the breakthroughs in visual recognition comes from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual competition in computer vision that emerged on the basis of part of a public dataset called ImageNet [5].

3.1.1. AlexNet

2010 and 2012 winner was AlexNet from Alex Krizhevsky, Ilya Sutshever, and Geoffrey E. Hinton. The net comprises of five convolutional layers (some followed by max-pooling), and three fully connected layers. The network also introduces then-recent dropout as a way of curbing overfitting [3]. Figure 3.1 illustrates AlexNet adapted to share the work over two GPUs for faster training.

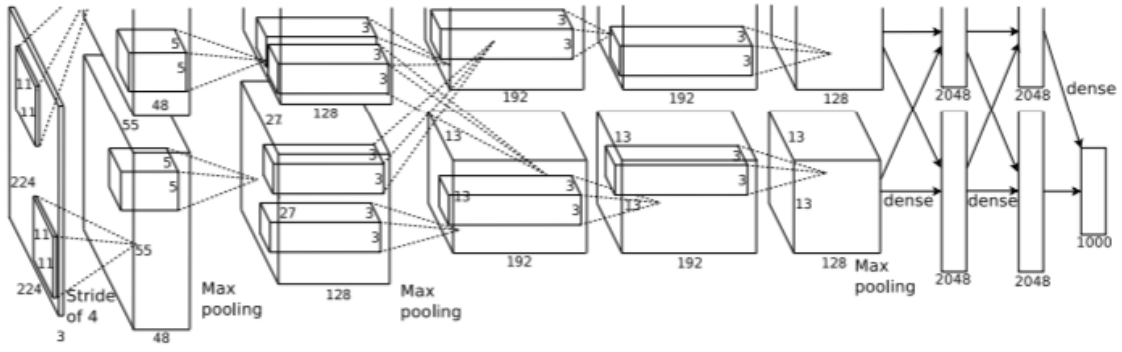


Fig. 3.1. AlexNet architecture split to work on two GPUs [3]

3.1.2. Inception (GoogLeNet)

Two years later particular version of Inception, GoogLeNet, was presented at ImageNet. This was a novel approach, breaking with the usual structure of multiple convolutional layers followed by normalization and pooling. GoogLeNet has twelve times fewer parameters than AlexNet and yet it was able to deliver better results. This is thanks to the R-CNN approach that stands for Regions with Convolutional Neural Networks [6]. Inception modules use 1×1 convolutions to drastically reduce the number of operations thus allowing to make the whole network wider and deeper. Such a module, depicted in Figure 3.2 is made of a few convolutional blocks of a various number of filters that are later concatenated into a filter bank that becomes the input of the next stage. This idea is inspired by the success of embeddings known from other deep learning tasks, such as sentiment analysis. An embedding is a form of densely compressed information about a fairly big part of an image [6].

An inception network is built by stacking such modules and 2×2 max-pooling layers to reduce the grid. GoogLeNet version is made of 22 such modules, while the number of layers in the whole model is about 100. The network also has extra classifiers on the side. Their loss is weighted and added to the total loss. This operation aims to support gradient propagation and to add extra regularization [6].

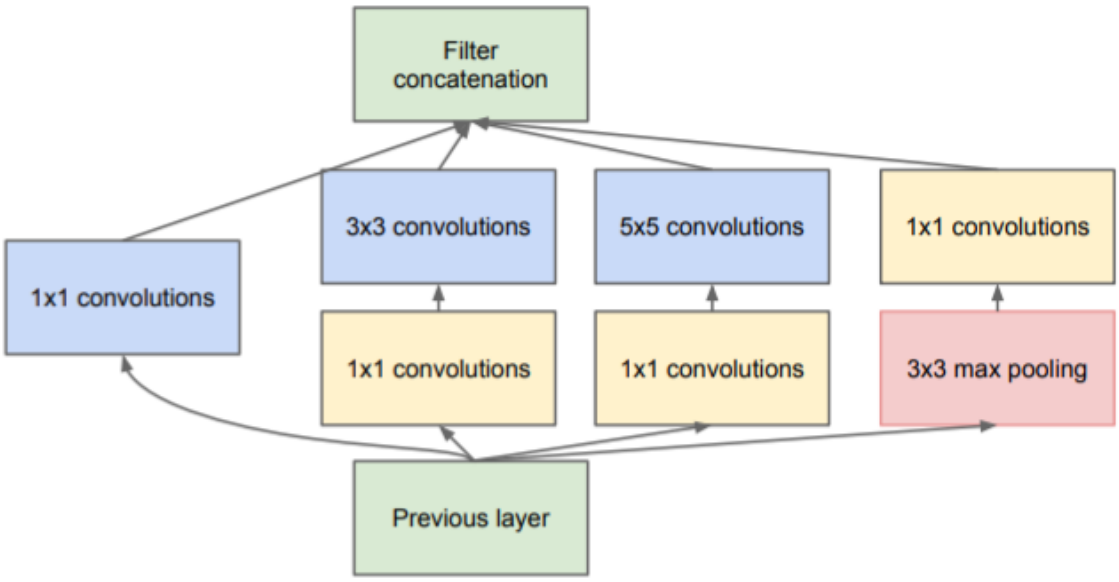


Fig. 3.2. Inception with dimension reductions [6]

3.1.3. VGG

VGG (Figure 3.3) is another 2014 ImageNet participant with a sequential structure similar to the one of AlexNet. There are, however, a few significant differences. Firstly, convolution filters are very small - 3×3 with a stride of one and the spatial resolution is preserved through padding. There are five max-pooling layers spread across the net. Two 3×3 convolution layers of stride one are equal in the size of the receptive field to one 5×5 operation, yet with two non-linear activations that result in more discriminative decision [7]. The fully connected part of the net is made of two layers of 4096 filters, 0.5 dropout and classification layer of 1000 filters.

Fig. 3.3. VGG net, adapted from [7]

3.2. Semantic segmentation

Image classification answers the question: "What's in the image?", while object detection tells "Where is it located?". To answer those two questions combined is to perform semantic segmentation. In this type of task, the model is usually given a label of the size of the image that masks specific parts of the image according to what is shown there as demonstrated by Figure 3.4. Therefore semantic segmentation attempts to classify each pixel of an image as belonging to one particular class.

Classification methods mentioned above, while achieving very good results, are unable to consider the global context of the image. There both convolution and pooling layers result in shrinking feature maps that later have to be flattened to be passed to fully connected layers. This is why some new approaches were necessary that would allow upsampling the feature maps and to replace fully connected layers with some other solution. In many of such experiments, classification models such as VGG are "reused" as feature extractors. This also has the benefit of the possibility to apply transfer learning and to reuse training weights from other tasks. A detailed description of two of such architectures is given below.

(a) Input image (b) Colored output mask

Fig. 3.4. Input-output pair used for semantic segmentation of an image. Adapted from [8]

3.2.1. Fully Convolutional Network

The network architecture built solely on convolutional layers was proposed by Evan Shelhamer, Jonathan Long, and Trevor Darrell in 2015 on Computer Vision and Pattern Recognition conference [9] and later, in the updated and extended version in 2017 in Transactions on Pattern Analysis and Machine Intelligence [10]. It makes use of the advances in image classification that are later transferred and fine-tuned to pixel-level labeling by replacing dense layers from classification tasks with crafted convolutional layers.

Architectures mentioned in 3.1 like VGG can be used as feature extractors. They take fixed-size inputs and consist of many convolutional layers that are followed by fully connected layers that ignore spatial coordinates when returning output [10]. Instead, Shelhamer, Long, and Darell dispose of two dense layers of 4096 filters and replace them with two convolutional counterparts of the same number of filters. Such a network, like its classifying original, can take input of any size, but the output remains reduced because of subsampling present in feature extracting part of the net. Further modifications are needed to obtain a map of the same resolution as fed to the network. The advantage of discarding dense layers is that FCN can work with an input of any size.

Feature map can be upsampled with various kinds of interpolation including bilinear, bicubic or the simplest, nearest neighbor methods, but in no way does this approach allow the network to learn its upsizing parameters. Ideally, one would want to "revert" convolution operation. This is possible if we remember that convolution with the kernel can also be explained through matrix multiplication with vectorized input. 4×4 array convoluted without padding with 3×3 filter yields 2×2 matrix. An equivalent version of this operation takes the input flattened to 16×1. Each step of convolution unrolled left to right is also 16 elements long [11]. Four such steps are needed to traverse our input. This operation, as a result, gives us 4×16 sparse convolution matrix **C**. Its product with 16-dimensional input is of size 4 and can be reshaped to 2×2. From here performing the backward operation is as easy as taking transposition **C.T** and multiplying it with flattened output [11]. It is worth noting in this approach that convolution filters are not fixed but rather, as with standard Conv Layers, they are learned and provided a proper activation function, obtained upsampling is not restricted to being linear [10].The transposed convolution operation is sometimes referred to as *deconvolution* or *fractionally strode convolution*, but the former term should be used with caution, as per mathematical definition deconvolution is an inverse operation of convolution and not its transposed form [11].

Going deeper with convolutions enables the network to learn more abstract features, but at the same time makes it lose the *where context* in favor of *what*, as a result, the output of FCN can be somewhat coarse [9]. To remedy this, the authors propose skips that add together results of higher layers after upsampling. With this operation, the network becomes a structure of a directed acyclic graph (DAG). Figure 3.5 provides a better understanding of how *skip connections* are handled in FCN based on VGG architecture because this feature extractor provided the best results. `Pool3`, `pool4` and `pool5` are max-pooling layers of VGG. Three versions are described in [9] and [10]: FCN-32s (32 pixel stride net, no skips), FCN-16s (16 pixel stride net, skip from `pool4` layer) and FCN-8s (8 pixel stride, skips fused from `pool3` and `pool4`. In FCN-32s, the most basic version, `pool5` is the input for former fully connected layers, now replaced with convolutions. In FCN-16s, `pool4` is upsampled with transposed convolution and later convoluted with 1×1 kernel for extra prediction. Similarly in FCN-8s, 1×1 convolution is also applied to `pool3` and fused with doubly upsampled output of `pool4` and `conv7`. One last transposed convolution with softmax activation is necessary to obtain proper output size. In presented experiments, FCN-8s proved to score highest and also to give most detailed segmentation map [9], [10].

3.2.2. U-Net

Another interesting deep learning architecture for segmentation is the one named U-Net proposed by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015 at Medical Image Computing and Computer-Assisted Intervention [12]. Similarly to FCN, it is made only of

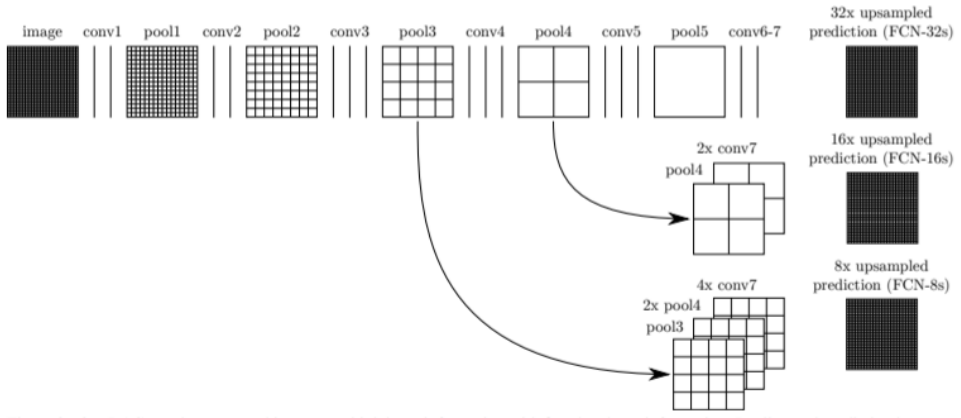


Fig. 3.5. Illustration of DAG skip connections aimed at refining coarse output of FCN and preserving high-level information [9]

convolutional layers and again, the lack of dense layers allows it to process the input of any size. It is also using skip connections to retain *where* information along with *what*.

The idea behind U-Net came from the medical field - one of those areas that would benefit significantly from AI solutions allowing for automated diagnosis. At the same time, it is a field where obtaining high quality, annotated data is expensive and often hard. This architecture attempts to address those shortcomings and perform well when trained end-to-end on limited data. As indicated in [12], one of the attempts to deal with a limited training set was patching the image in a sliding window manner and using subsequent pieces of information for training. This makes the dataset bigger and enables the network to localize. Unfortunately, at the same time, patching can mean there is less context within the images. To add to that, training must be performed on every patch separately, and overlapping them create much redundancy. Therefore new architecture was proposed to overcome those shortcomings.

Looking at Figure 3.6 one can clearly understand where the name U-net comes from. The network's two "arms" have different purposes. The left side or contracting side is not much different from the most widespread convolutional architectures. There 3×3 unpadded convolutions are activated with ReLU and max-pooled with a stride that downsamples the result by half and with every step twice as many feature channels are added. On the expansive, right side, the feature maps are upsampled twice and then convoluted with 2×2 kernel. Here, with each step, the feature channels are halved. Next follows concatenation with corresponding "level" from the left side that is cropped to accommodate for the loss of pixels in unpadded convolution from the left. Each step here finishes with 3×3 convolution activated with ReLU. Finally, 1×1 conv with 1 filter is used to output binary classification, but this layer can be adapted to work with any number of classes. The number of feature channels remains big despite the fact that it is being halved in each step in the expansive part. Thanks to them, the context information can be propagated to higher resolution channels [12]. Because convolutions in the contracting

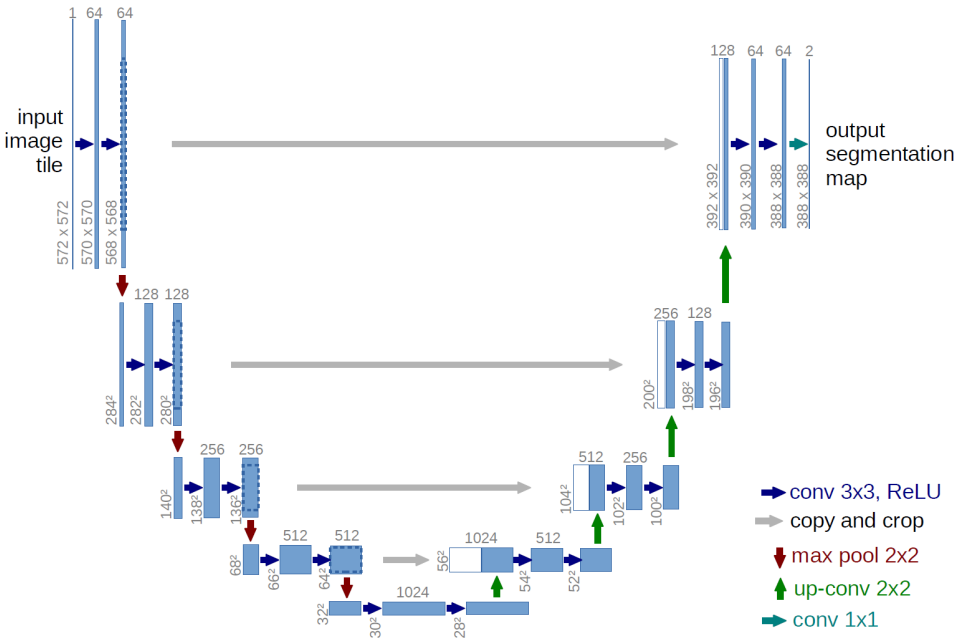


Fig. 3.6. U-net architecture [12]

part are not padded, some cropping is necessary to match the dimensions along both arms of the network. It should be noted here that for the same reason, the output image is smaller than its input by a constant number of 188 pixels in both dimensions. By design architectures that use no dense layers should work on arbitrary input, but here its size is constrained by the max-pooling operations. Each of five of them cuts the image size by half in both dimensions so input should be square and divisible by $2^5 = 32$. Later on, the same group of researchers extended U-net to non-binary classification and experimented with extra loss layers in lower resolution parts and modifying "softmax" loss with Euclidean loss to better guide the net into recognizing different categories on dental X-ray images [13]

U-net was primarily designed to segment neuronal structures on images obtained by electron microscopy and in cell tracking challenge. In such tasks, it is particularly important to make sure the network learns to separate individual cells rather than grouping them together. For this reason, the authors proposed a weight map created from each ground truth image to give more importance to separation borders and thus to enforce the net to learn them [12]. Because of limited training instances, data augmentation was proposed. Applying rotation and elastic deformation helps to train a model robust to such deformations. On top of that, deformation of biological tissues is nothing unusual and such modifications yield very realistic results.

3.3. Evaluation metrics

3.3.1. Multi-label classification

In a multi-label classification problem, the prediction is not simply correct or incorrect. To simplify the process, partially correct results could be considered incorrect. This approach, however, can be considered harsh, as it only gives information about samples that were classified as "fully" correct and it would not take into account cases where only one out of a few labels was assigned correctly. Such a metric is referred to as the Exact Match Ratio.

$$MR = \frac{1}{n_{cl}} \sum_{i=1}^{n_{cl}} I(Y_i = Z_i) \tag{3.1}$$

where

Y_i – predicted labels of class i

Z_i – true labels of class i

For accounting for partial correctness accuracy, it is proposed to define accuracy as the count of labels predicted correctly divided by the total count of labels for that class and averaged across all classes. Precision, recall and F1 score can be redefined similarly.

Another approach acknowledging partial correctness would be the Hamming loss that takes into account how many times the correct label is not predicted and how many times an incorrect label gets predicted [14].

$$HammingLoss, HL = \frac{1}{kn} \sum_{i=1}^{n} \sum_{l=1}^{k} [I(l \in Z_i \wedge l \notin Y_i) + I(l \notin Z_i \wedge l \in Y_i)] \tag{3.2}$$

where

I – an indicator function

$HL = 0$ would indicate that no labels are assigned incorrectly and no labels failed to be predicted, therefore the lower the score, the better the classification algorithm.

3.3.2. Image segmentation

Authors of Fully Convolutional Network proposed to use four metrics to assess performance semantic segmentation. They make use of the metrics known from classification problems, but also extend them to take into account that predicted masks may not overlap fully with ground truth masks. All the metrics can be derived from confusion matrix C of size $n_c \times n_c$ where n_c is the number of classes, C_{ij} are the pixels belonging to class i but predicted to belong to j. C_{ii} is then an instance of i predicted correctly, that is true a positive and t_i are all class instances.

Those metrics are:

- pixel accuracy
$$\frac{\sum_i C_{ii}}{\sum_i t_i} \tag{3.3}$$
- mean pixel accuracy
$$\frac{1}{n_{cl}} \frac{\sum_i C_{ii}}{t_i} \tag{3.4}$$
- mean intersection over union (also known as mean Dice score)
$$\frac{1}{n_{cl}} \frac{\sum_i C_{ii}}{t_i + \sum_j C_{ji} - C_{ii}} \tag{3.5}$$
- frequency weighted intersection over union
$$\frac{1}{\sum_k t_k} \frac{\sum_i C_{ii} t_{ii}}{t_i + \sum_j C_{ji} - C_{ii}} \tag{3.6}$$

4. Depth information in computer vision systems

Until very recently cameras that enable depth acquisition were mostly found in advanced robotic systems, however, in the last years, we can see depth sensors being more widely used in consumer electronics as well and thus enabling more lowcost experiments to everyone. Some of the most popular devices are several versions of Kinect for Xbox and Intel's RealSense devices.

4.1. Depth acquisition

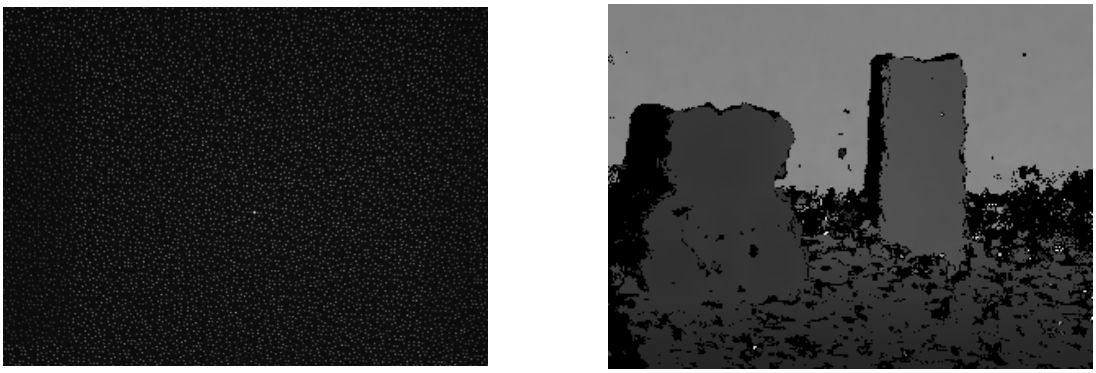
Methods retrieving distance information can be described as either passive or active ones. In the passive techniques for sensing depth, depth imaging is achieved by constructing a disparity map from stereo images. This, however, required significant computing power. Current designs of Kinects, Intel's and other devices rely on active methods out of which two main types of the sensors can be distinguished:

- Time-of-flight camera

  Those sensors, often encompassed in LIDAR systems, began to emerge at the beginning of the 21st century with Z-Cam considered to be the first depth camera available. While they do not have a very big spatial resolution, they can, in turn, provide high throughput of up to 160 images per second. The most basic TOF emit very short light pulses that are reflected by the objects on the scene and the delay resulting from the distance to the object is imaged onto the sensor [15]. Used for example in Kinect 2, ASUS Xtion 2 [16].

- Structured light

  Some other devices use an infra-red structured light projector and low-resolution infra-red camera. Light from the projector in the form of small dots, invisible to human sight, falls on surrounding objects as shown in Figure 4.1. Displacement of the dots is later registered by an infra-red sensor which is no different from a regular camera except color range. [17] Final values are distances to the laser-sensor plane, not to the sensor itself. [18]. On Figure 4.1b, one can notice some shadows surrounding the objects. This comes from the fact that they block a part of the light. If the sensor is positioned on the left of



(a) Infra-red pattern emitted by Kinect camera [18] (b) Example of the transcoded depth image

Fig. 4.1. Emitted infra-red light and image registered by the camera

the laser, the shadow appears on the left side of the objects, because the device is unable to estimate the distance that is not in the line of sight of the laser [18]. Used, for example, in Kinect 1 and Intel RealSense D435 [16].

4.2. Depth processing

One of the challenges preventing new depth datasets from becoming more available mentioned in [19] is the need for the right processing software. As of now, there are a few libraries that expose the device's features to the user like Libfreenect and some libraries allowing more advanced features - tracking, recognition, etc., but without being able to control the device itself, for example, OpenNI or NITE [20]. Sometimes also the manufacturers launch product-specific development software, like Kinect SDK or RealSense SDK from Intel that also includes GUI application for monitoring the stream.

For robotics use, it can also be convenient to use rosbags to gather necessary data from more sensors, not only images. In this option, it is the user that carries the burden of synchronizing the frames and aligning images properly. Alignment algorithms are described better in [21] and [22], as well as in Section 5.2. Resulting point clouds can be further processed with Point Cloud Library.

4.3. Current applications and future development of depth sensing

First depth sensors appeared in consumer-available electronic as an option to replace handheld controllers in game consoles. Using depth information for gesture tracking and recognition together with voice commands allowed to propose gaming experience free of cables and pads.

The idea did not gain much acclaim in the industry and was soon obsoleted. In the meantime, Kinect gained popularity at universities and in hobbyist projects as a cheap way for experimenting with new modality.

Probably the most standard and widespread use of depth is to embed it in robotics systems. It can bring a valuable contribution when creating automation or decision making algorithms as it adds unique information that cannot be assumed from RGB data only - it enables the robot to receive the information about the distance to the surrounding objects, their size or shape. Many actions such as grasping or manipulations are based on assumed geometry of objects rather than their color or texture. Color-invariant data can provide strong shape suggestion for such actions, especially if a robot is given task on previously unseen objects. Robots can also navigate better knowing how far they are from obstacles, including real-time navigation.

Primarily Kinect was associated with human interaction for gaming, and as such, it is obvious that some of the first applications included detection and localization of body parts in systems like Xbox One. Apart from gaming, depth cameras are also being used in pose estimation experiments, for example in [23], as a replacement or complement of motion capture systems where wired markers or specially designed costumes can restrict natural movements. Here, like in other applications, depth is valuable information because it is invariant to illumination or texture. Apart from that, many experiments try to employ depth maps for gestures recognition, especially sign languages, but also hand gestures used by Italians [19]. Some other "human-centered" tasks where depth importance is being examined include pose estimation, face or emotion recognition, full-body person/gender recognition and tracking interaction.

The surge in depth cameras popularity is also thought to bring advances in scene reconstruction and camera tracking topics. These are usually troublesome because to obtain ground truth poses of the camera it is necessary to employ external devices. There are attempts to bypass this limitation with the use of synthetic data, where all aspects can be easily controlled.

Similarly, deducing six degrees of freedom pose of an object can benefit from the fact that depth provides information on an absolute scale. This task requires finding such transformations that a 3D model obtained earlier can be fitted into the scene. Here, though, the use remains limited, as preparing ground-truth information is laborious task - both the 3D model and information about its pose in each image are necessary [19].

Another idea making use of the fact that depth camera readings are invariant to light changes and color-cluttered environment is to employ depth-sensing devices in systems for pedestrian detection. In [24] the authors experimented with images taken with Kinect from which features are extracted with Histograms of Oriented Gradients (HOG) method and later with AdaBoost, a classifier of ensemble type. Researchers showed that gray-scale image only can detect pedestrians fairly easily, but pairs of gray-scale and depth images allowed them to achieve reduced false positive rate and be more accurate in terms of pedestrian detection. The main drawback of this

experiment is that it was conducted in laboratory settings. Therefore it is not sure if the results will hold the same in the outside world where the amount of IR light sensed by the camera can be significantly greater.

Capturing RGB-D data started with single images allowing to represent the scenes in so-called 2.5D. Later on, videos were used to provide more information on the scene, but they still lack some information to reconstruct it fully. According to [19], obtaining the geometry of the whole surface and semantically annotating the surface itself is the next step in the reconstruction challenge. What follows could be volumetric labeling for the segmentation of whole meshes to give computers a more in-depth understanding of the scene.

Depth maps and created with them point clouds are also becoming more popular with the rise in augmented reality. Unlike virtual reality, where the whole digital world is created from scratch, augmented reality is faced with the problem of properly merging the real world with computer-generated one. When processing depth maps, occlusions are parts of the scene not visible to the camera, while in augmented reality it is necessary to occlude some objects if the virtual ones are behind the real ones or more distant to the viewer. Doing that requires reconstructing a model of a three-dimensional world like described above. Having it allows creating a mask that realistically reflects occlusion. There are a few ways to do it. In the simplest version, just a depth map can be used. In another option, a mesh is created from a point cloud representation of the scene. Later from this mesh an occlusion mask can be created. It has to be noted though, that creating a mesh is a fairly heavy operation to be performed in real-time on current devices [25].

Some very recent work is also conducted in the direction of using depth cameras for facial identification as well as emotions or expression recognition. This development, however, is expected to rise only should this type of cameras appear more broadly in laptops or mobile devices [19]. Indeed some recent smartphones feature Time of Flight sensors either on the front or on the back. Yet, for now, they appear to be mainly used for blurring the background in mobile portrait photography and to enhance camera focus in low-light scenes [26].

4.4. Datasets overview

One thing that makes use of RGB-D data research way harder than for other computer vision tasks is for sure the lack of proper data - unlike standard image analysis, it cannot be obtained by simply scraping the Internet or taking some photos with any camera that is within reach. Before the launch of Microsoft Kinect devices, this was way more expensive and required skills to build custom setups with costly 3D scanners. Such setups also had limited use, because transportation was cumbersome.

While standardized datasets can often oversimplify the task and provide too optimistic results, there is no better way to ensure transparency and reproducibility of the research than conducting tests on the same dataset. Using the same input across a variety of designed algorithms can also spark competitions as results are now easily comparable. Having a dataset ready also leaves more time for actual research and experimentation and can funnel them in previously less explored areas. In the area of deep learning, we can take into account recent advances in transfer learning, where having a dataset even barely similar to our task can help to refine results on our data.

With that in mind, during CVPR workshop in 2016, Michel Firman presented his work aimed at acknowledging worthy datasets containing depth maps [19]. Apart from suggesting resources for a variety of applications, he also evaluated them with regards to device, quantity, and quality. Type of labeling is provided as well because there can be many types of them for different tasks and even each application can have different labeling standards that can require particular processing. Annotation types for semantic segmentation and reasoning include standard dense pixel labeling with or without semantics, labels on reconstructed point clouds, 3D bounding boxes and 2D polygons. For segmentation tasks they also attempt to assess the realism of the dataset, that is, whether objects were arranged in a laboratory environment, arranged in real-world places or maybe no modifications had taken place when collecting a dataset of real-world scenes.

For semantic segmentation some of the most widely used datasets are:

- NYU Depth (versions 1 and 2) - made of video sequences showing a variety of indoor scenes recorded with Microsoft Kinect. Dense labeling is made in such a way that the data can also be used in instance segmentation tasks [27]. Some RGB, color-coded depth and semantic labels are presented in Figure 4.2

- SUN RGB-D - similar in size to PASCAL VOC, one of the biggest and most popular dataset for visual recognition. SUN RGB-D creators attempted to create a benchmark dataset that could be used in many tasks, from semantic segmentation, detection and pose estimation to total scene understanding. A few different devices were used in the creation process so extracted features can be invariant with regards to device intricacies [28]

- B3DO: Berkeley 3-D Object Dataset - crowdsourced dataset with bounding box annotations, aimed for category recognition in three dimensions [29]. It appeared few a years earlier than NYU and SUN

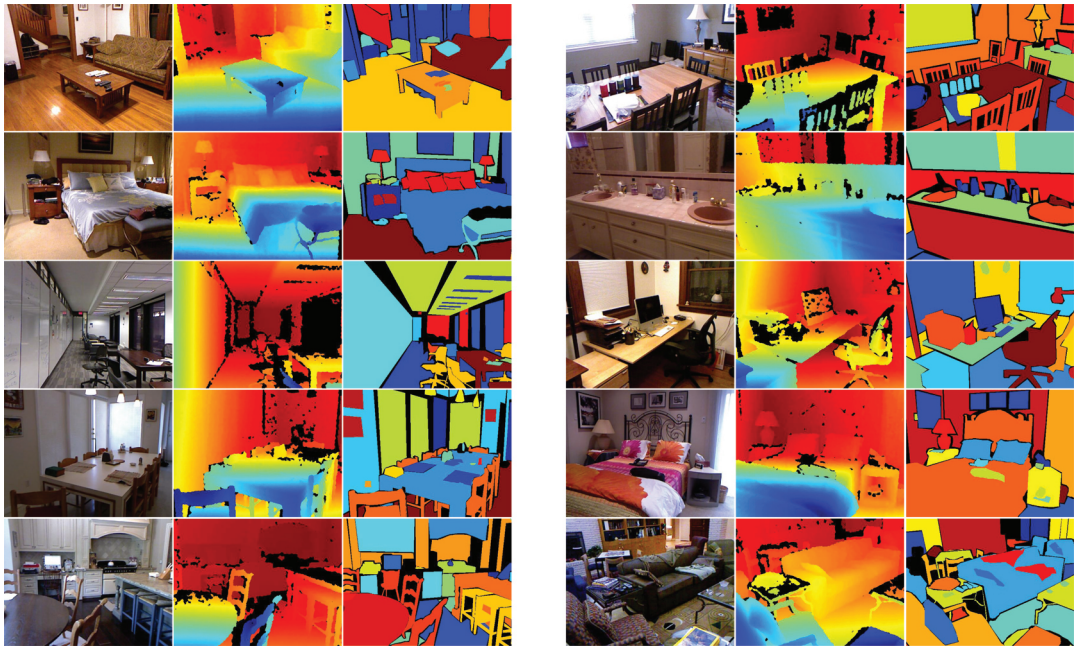


Fig. 4.2. Densely segmented samples from NYU Depth Dataset V2 [30]

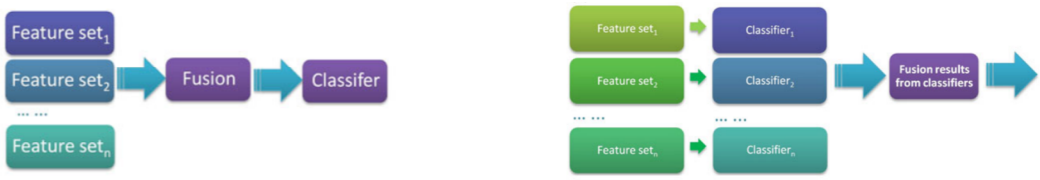
4.5. Including depth in deep learning tasks

While depth maps after some preprocessing can be treated as images, they rarely convey the same message. Objects can be distinguished easier, but texture or color information is gone. Moreover, the distance range of RGB cameras is significantly bigger than their counterparts. Including depth data in image processing pipelines in such a way that they contribute best to the final result is still a topic under research. Nonetheless, some solutions appear to be more popular than others.

The simplest approach would be to treat depth map as extra, fourth channel and simply append it to a 3-channel RGB image. It was featured in FCN evaluation on NYUD v2 dataset but yielded only moderately better results than RGB alone: for FCN-32s model trained on RGB pixel accuracy and mean intersection over union was, respectively, 60.0 and 29.2 whereas for 4-channel RGB-D model it was 61.5 and 30.5. The hypothesis provided to explain this low improvement is that the network experiences some difficulties with gradient propagation [10].

4.6. Fusion of modalities

Eventually, it appeared to be widely acknowledged that learning features for color and depth images would be more effective when done separately. This allows for extracting features that are highly specific to their modalities. Two models would be later fused together in one of the bottom layers to provide the desired output. Such an approach can also make use of transfer



(a) Early fusion

(b) Late fusion

Fig. 4.3. Early and late fusion comparison [32]

learning if it was not for the problem of finding pre-trained weights for depth segment. For this reason, it is not uncommon to see the same weights used in both parts of the model for intra-modal transfer in the RGB part and cross-modal transfer in the depth part [31].

Ionescu et al. described two ways of fusing multimodal computer vision systems. In early fusion, the classification is performed on features fused from all modalities. On the contrary, in late fusion, each feature set has its own classifier and their predictions are later summed. The difference between the two approaches is better shown in Figure 4.3 Authors also mention that other ways of combining classifiers such as product, max, mean or median were reported but using sum appears to be less sensitive to classification errors [32]. Late fusion was also used in [9] with better results than modifying the architecture to take four-channel data.

4.7. HHA depth encoding

A novel approach to depth maps was proposed by Gupta et al. in [33] and later evaluated also in [34]. In those works, a new, geocentric embedding is proposed that allows for encoding depth in three channels manner. Each pixel of the depth map is therefore expressed as:

- horizontal disparity
- height above ground
- angle between the local surface normal and inferred gravity direction

The channels are later scaled to range from 0 to 255. As it can be observed in Figure 4.4, the objects on HHA images can be easily recognized which confirms that encoding is analogous to RGB representation [31].

It might seem that such an idea runs contrary to the main principle of deep learning saying that the features should be self-learned rather than calculated apriori. The authors, however, explain that it is highly unlikely that the network could be trained to calculate such properties, especially in cases when the training dataset is limited [34]. Furthermore, as already mentioned, this new encoding appears to provide certain similarities to RGB images thus allowing HHA networks to be fine-tuned with model weights from RGB tasks.

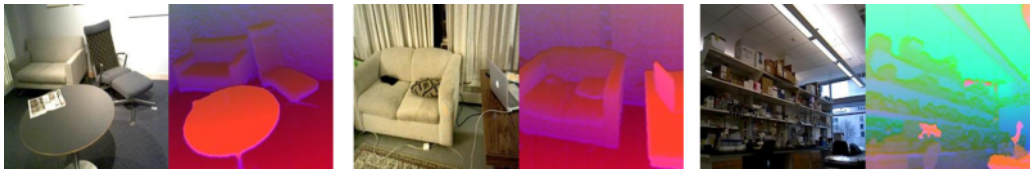


Fig. 4.4. Comparison of RGB and HHA encoded depthmaps [31]

4.8. Is depth really needed

The works quoted so far state unanimously that depth indeed provides relevant information that supports various algorithms and allows them to achieve better results. This may not be the case in all scenarios - depth sensors have their limitations but also the environment, for which the models are being developed, can be such that using another modality does not provide expected benefits. This was examined in *Depth Not Needed - An Evaluation of RGB-D Feature Encodings for Off-Road Scene Understanding by Convolutional Neural Network* [35].

Here the authors did not use time-of-flight or structured light cameras, but stereo disparities obtained by two methods: Semi Global Block Matching and Adaptive Support Weights. Five variations of depth encoding were subject to test. They were normalized to 0 - 255 range and stacked together with RGB channels. Such input was then fed to SegNet encoder-decoder architecture, modified to accommodate additional channels to provide classification at pixel-level. The dataset is comprised of images from the off-road trip in the center of England and labeled into categories such as grass, tree, sky or water so objects that may not have well-defined boundaries or a specific shape.

Classification results presented in this paper showed that depth in the form of stereo disparities provide little to no improvement in comparison with RGB only. This indicates that depth may be of higher importance in indoor scenarios, where it is easier to find well-defined objects that can be classified without doubts. Another thing to consider is that overall segmentation results were very good, so achieving even slightly better performance is quite difficult [35].

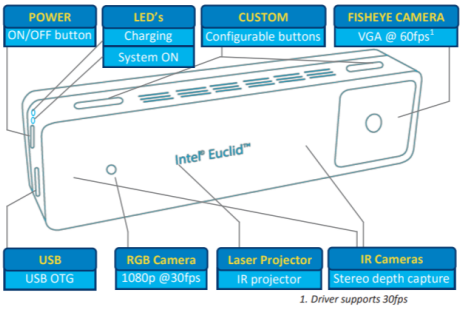
5. Creating experimental dataset

5.1. Intel Euclid Realsense camera

All RGB and depth data used for the purpose of this project were made using Intel Euclid Development Kit (Figure 5.1a). It is an all-in-one computer with Intel Atom x7-8700 Quad-Core processor, 4GB memory, 32GB storage, WiFi and Bluetooth for wireless communication and various sensors such as inertial measurement unit (IMU), proximity sensor, barometric pressure sensor, and GPS. The device has preinstalled Ubuntu and Robot Operating System (ROS) and after connecting an external screen and mouse or keyboard, it can be used out of the box. Apart from RGB and fisheye camera, it has ZR300 depth camera in RealSense technology with depth range of 0.55m to 2.8m [36]. In Figure 5.1b cameras placement can be observed.

For other cameras in the RealSense series, Intel published software development kit, unfortunately, it is not compatible with Euclid device despite using a camera from this series. In turn, Euclid can be operated using a web interface from other PC after connecting to its network or connecting the device to the same network. In the web interface, the user can choose some predefined scenarios such as cameras, turtlebot or person follower that start a set of sensors and execute predefined Euclid nodes. Nodes are in fact ROS launch files and their role is to provide an abstraction to Euclid's automation layers [36]. It is also possible to create one's own

(a) Euclid device



(b) Camera placements on the device

Fig. 5.1. Intel Euclid Development Kit [36]

scenarios and configure them to be able to run them from the browser. While the scenario is running, the user can switch to the "Monitor" tab to have a look at what the device is registering - sensors that can be monitored here are: color, depth, person tracking, fisheye, IMU (gyroscope and accelerometer), trajectory. Some parameters of the nodes can be also dynamically changed from the respective tab. The device can be also operated remotely via SSH [36].

5.2. Experiment conditions

To create a dataset for this project, a set of objects was selected. The main criteria were that a few items of the same category could be collected (i.e. different color, size, slightly altered shape, etc.) and that they are big enough to be easily distinguishable on the depth image. The sensor produces quite noisy output and for this reason objects like pens or tiny toys were excluded after some initial experiments.

To save the images, a PC was connected to the network created by Euclid device and the "Cameras" scenario was started that launches color and depth cameras. The distance measurement unit is millimeters, so the range of the image created from the sensor is too big to be displayed properly. For this reason, the depth map that can be observed on Euclid's web interface is transcoded to the range of 0 - 255. To save both raw and transcoded versions along with color images, ROS utilities present in the device were used. Rosbag is a package of tools that enables to record data from ROS topics in the form of bags that are nothing else than robot logs [37]. Five seconds long streams were saved for each situation using an appropriate command with Euclid accessed over SSH.

To extract necessary data from rosbags, Matlab R2019a was used. Extracted topics were:

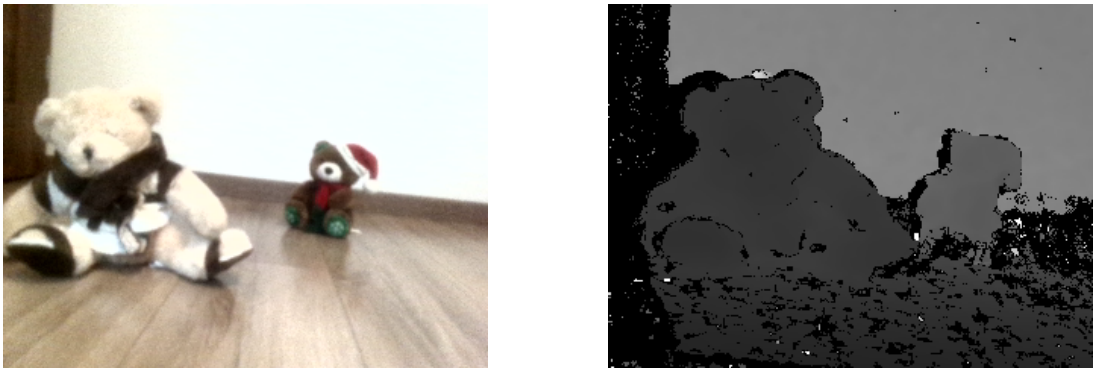
- `/camera/color/image_raw`
- `/camera/depth/image_raw`
- `/camera/depth/image_transcoded`

It is worth noting that depth images are `uint16` so map scale is 0-65535 mm or 0-65.535m. This is different from other cameras such as Kinect, that use `uint16` and need additional calculations to obtain the result in meters.

5.3. Image preprocessing

5.3.1. Image registration

Whenever two cameras are used to prepare pairs of the depth map and RGB image, it has to be taken into account that they have different parameters and they are placed in some distance



(a) Example of RGB image (b) Depth map of the same scene in transcoded units

Fig. 5.2. Example of displacement between fields of view in Intel Euclid camera

from one another. For this reason, their perspective differs a bit. The example is shown in Figure 5.2. We can see there that depth map is showing slightly different scene - teddy bear on the left appears to be a few centimeters away from the left border of the picture, while on the color image it is touching the border. The operation of aligning the two images is also called *registration*, and the process is virtually the same regardless if frames to align are two color images or color and depth map.

Some software development kits like OpenNI (for Kinect) or Intel RealSense SDK provide tools to perform the registration while reading from the stream of data. As mentioned before, RealSense could not have been used so the process was performed by the author of the project based on information in [22] and [21] and is described below.

Image registration requires us to know beforehand some information about both cameras, that is their *intrinsic* and *extrinsic*. Intrinsic describe cameras themselves:

1. their focal lengths (in pixel units): f_{xrgb}, f_{yrgb}, f_{xd}, f_{yd}
2. their optical centers: c_{xrgb}, c_{yrgb}, c_{xd}, c_{yd}

Meanwhile by extrinsic, we mean values that which describe how one camera is related to the other. Those are:

1. 3×3 rotation matrix R
2. 3×1 translation vector T

Another often used convention is to present extrinsic as a 4×4 matrix as in 5.1:

$$extr = \begin{bmatrix} R_0 & R_1 & R_2 & T_0 \\ R_3 & R_4 & R_5 & T_1 \\ R_6 & R_7 & R_8 & T_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

All parameters mentioned above can be obtained from a configuration file saved on the device's hard drive or they can be read by subscribing messages from topics `/camera/color/camera_info` and `/camera/depth/camera_info` from rosbags. In the config file, values are straightforwardly, while in the subscribed messages, the values of T and intrinsics can be read from projection matrix P (See 5.2).

$$P = \begin{bmatrix} f_x & 0 & c_x & T_0 \\ 0 & f_y & c_y & T_1 \\ 0 & 0 & 0 & T_2 \end{bmatrix} \tag{5.2}$$

It is also necessary to convert depth map values into meters - in case of Euclid that means dividing each value by 1000.

Now each (x_d, y_d) pixel from depth map can be projected to a point in three-dimensional metric space with formula 5.3:

$$\begin{aligned} x_{P3D} &= (x_d - c_{xd}) \cdot depth(x_d, y_d)/f_{xd} \\ y_{P3D} &= (y_d - c_{yd}) \cdot depth(x_d, y_d)/f_{yd} \\ z_{P3D} &= depth(x_d, y_d) \end{aligned} \tag{5.3}$$

where

$(x_{P3D}, y_{P3D}, z_{P3D})$ – P3D point coordinates in 3D metric space

Then it is necessary to apply rotation matrix R and translation vector T as in Equation 5.4 to get this point coordinates in 3D space associated with the RGB image.

$$P3D' = R \cdot P3D + T \tag{5.4}$$

Now point $P3D'$ can be projected back to (x_{rgb}, y_{rgb}) location in RGB camera using color camera intrinsics (Equation 5.5. Some kind of interpolation is necessary when doing that, as calculated values of (x_{rgb}, y_{rgb}) may not be of integer type. Here simple "nearest neighbor" approach was used by rounding the locations to nearest integer.

$$\begin{aligned} x_{P2D} &= (x_{P3D'} \cdot f_{xrgb}/z_{P3D'}) + c_{xrgb} \\ y_{P2D} &= (y_{P3D'} \cdot f_{xrgb}/z_{P3D'}) + c_{yrgb} \end{aligned} \tag{5.5}$$

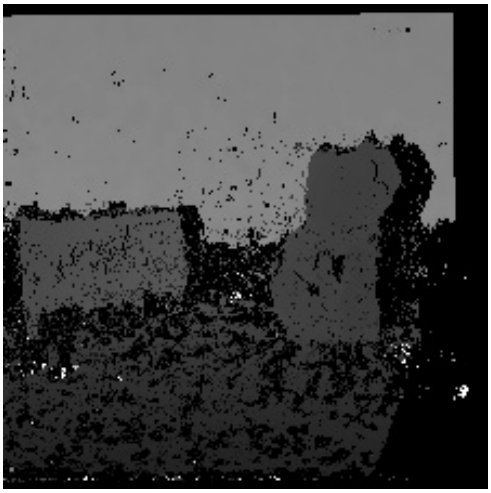
where

(x_{P2D}, y_{P2D}) – coordinates of depth map pixel mapped to RGB color space

With all that done, it is necessary to handle invalid values or occlusions that appear because not all parts of the depth map are visible to the color camera. To do that if pixel coordinates

(a) RGB image

(b) Depth map not registered to the image



(c) Depth map after registration

(d) Depth map with replaced invalid values

Fig. 5.3. Depth registration and inpainting

were negative or bigger than the width or height of the image, they were ignored. The effects of performed registration can be observed in Figure 5.3. Unfortunately as can be seen in Figures 5.3b and 5.3c the process is introducing some loss of information. This is a result of handling occlusions, and because fields of view of the two cameras cover slightly different scenes, they also have different properties, such as focal lengths.

Two optional steps can be done prior to image registration:

1. cameras can be recalibrated with use of checkerboard
2. images can require rectification

Here recalibration was not performed and per information in [38], images are rectified by Euclid's hardware component before being sent further so the only necessary step was to perform registration.

5.3.2. Depth inpainting

Later it was required to remove invalid values present on the depth maps. As said before, the maximum depth value as given by uint16 format is 65.535 meters, but the range indicated by the manufacturer is 0.55 - 3 meters. At the same time, images contain invalid values caused by the occlusion that is impossible to avoid with the method that the camera uses. It can be observed in Figure 5.3c that image quality suffers from both described issues - zero-values from occlusion and huge values of depth impossible to be registered with any available sensor.

For this reason, manual inpainting of invalid depth values was done in two steps outlined below. It is by no means the most elegant solution nor is it providing the best results. Depth inpainting is a broad subject that is outside of the scope of this work.

Depth inpainting consisted of removing:

1. Overly distant points:

   Values indicating distance bigger than 3 meters were replaced with a median value of the row that they appeared in. A masked median is chosen, in order to exclude zeros when calculating the median.

2. Zero points:

   Those values are more common than points indicating an enormous distance from the object. In those cases, a 8×8 window surrounding this point was selected, and the masked median was calculated. If it turned out to be an invalid value, then the masked median for the whole row was calculated. Finally, if that was also incorrect value, then the median value of the whole image was inserted.

5.3.3. Depth encoding

Removing incorrect values as described above is also necessary to perform HHA depth encoding mentioned earlier in Section 4.7. It was done using Python implementation created by Cheng Xiaokang and available at [39]. Some of the results are demonstrated in Figure 5.4.

5.4. Dense semantic labelling

Segmentation maps were created with Supervisely [40] web application because it has advanced labeling tools, simple interface and can be accessed remotely once the dataset is uploaded. It can also be used for collaboration when labeling needs to be shared, but in this case, images were annotated solely by the author of this work. Created segmentation maps were saved as JPEG files. Figure 5.5 shows the interface of Supervisely.

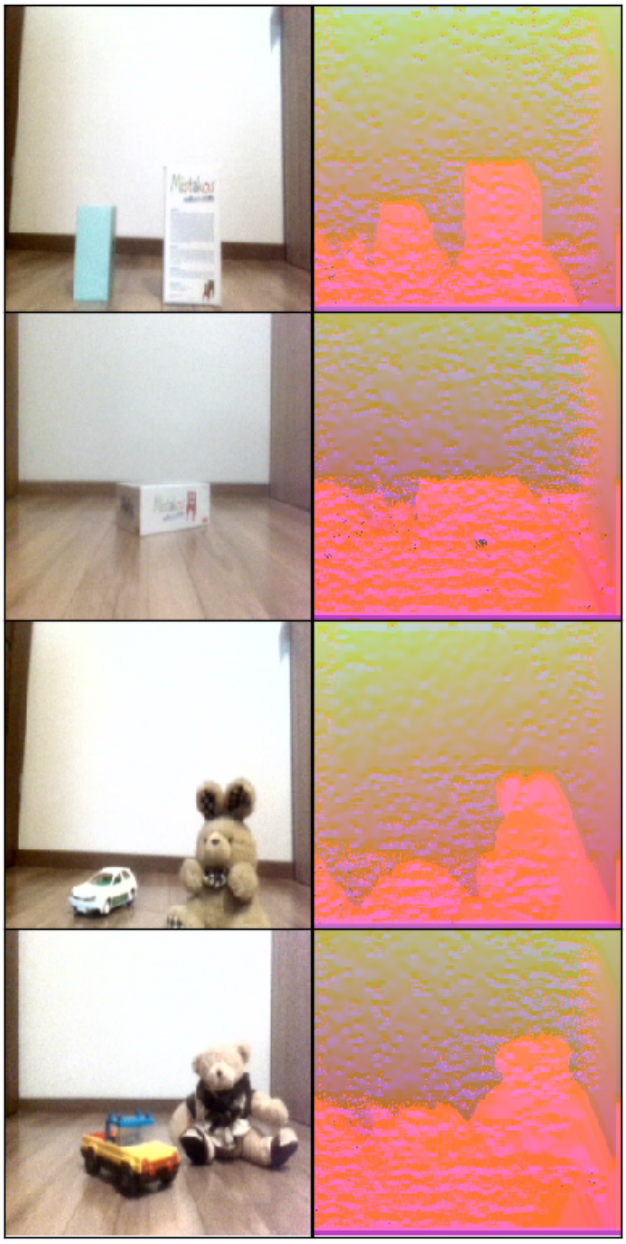


Fig. 5.4. HHA-encoded images

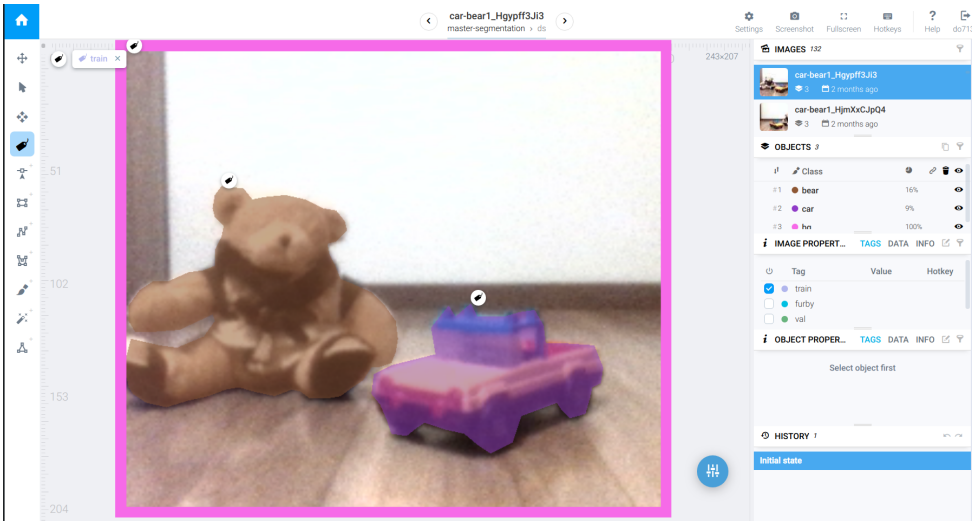


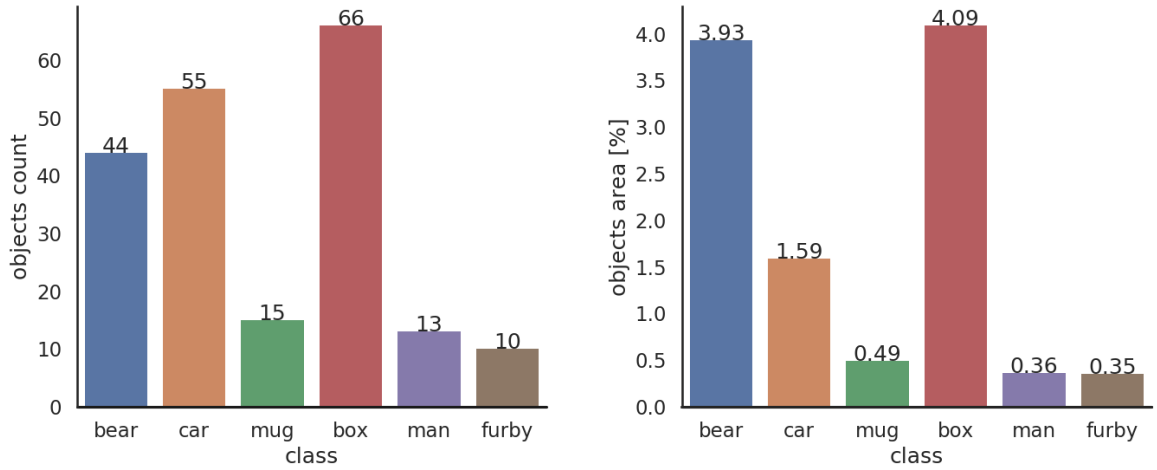
Fig. 5.5. Interface of Supervisely web application to annotate images

Pictures were taken in such a location that the background does not interfere too much with objects planned for segmentation, that is - on the floor. Each of the preselected objects was subjected to a few photos with different location and distance to the camera. There are also images with more than one objects both from the same and different classes but differentiating between instances is not in the scope of this work. There are six categories of objects in total. As it is shown in Figure 5.6, the dataset is not balanced both in terms of class counts and per percentage of pixels belonging to each class. The whole dataset is comprised of 121 images taken at two locations.

5.5. Data split and augmentation

The images are later split randomly into three subsets: training (60% - 72 images), validation (20% - 24 images) and test (remaining 25 images). The seed is used in the splitting function to ensure reproducibility. As the dataset is small compared to publicly available datasets, it was decided to employ data augmentation. The process uses various transformations to obtain new images while preserving their labels. Figure 5.7 is presenting some examples of such transformation, but they may not be exactly the same images that were used for training the models as new images are generated randomly on-the-fly and fed to the model in the training phase. Here also a seed is used to ensure that images and masks are transformed in the same way and that each model is trained on the same data. Chosen augmentation parameters were:

- rotation range: 30 degrees
- zoom range: 0.15
- width shift: 0.2



(a) Number of objects of each category

(b) Percentage of area for each class

Fig. 5.6. Dataset statistics, based on information from Supervisely

- height shift: 0.2
- shear range: 0.15
- enabled horizontal flip

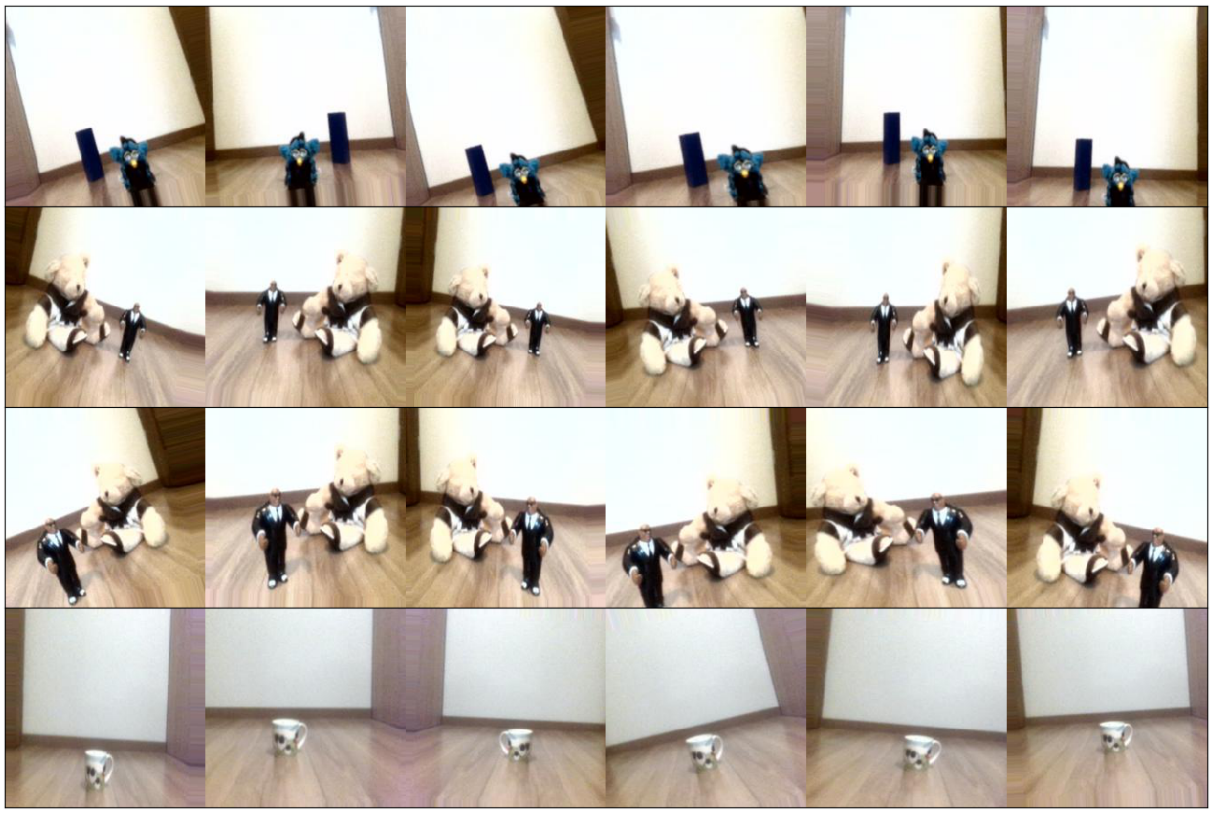


Fig. 5.7. Examples of data augmentation

6. Experiments and results

6.1. Implementation and training details

All networks described below were prepared using Tensorflow, version 1.14.0 mostly through its high-level Keras API [41]. Metrics outside of tensor scope were formulated with Numpy [42] and scikit-learn [43]. Code repository was stored on Google Drive and from there imported to Colab environment. Colab is essentially a Jupyter notebook hosted in a cloud and can be run from the browser without the need to install any extra software locally. The main advantage of using Colab over locally installed Anaconda/Jupyter notebook is that the user is given various runtimes to be used for free: CPU, GPU, and TPU. All models below are trained using Colab's NVIDIA Tesla K80 GPU with 2495 CUDA cores and 12 gigabytes VRAM.

6.2. Image recognition

As prepared images often contain more than one category of an object, the problem to solve would be that of multi-label classification. To test the impact of including distance information to the recognition problem, various approaches were examined.

Firstly, AlexNet was implemented with single input RGB as the baseline, two input versions of early and late fusion to be tested on RGB+D and RGB+HHA inputs. In early fusion, the features are concatenated after flattening but before fully connected layers. In late fusion, the outputs of the classification layer are summed. Later an attempt was given to classify images with VGG model in the same combinations.

As suggested in [44], sigmoid activation is used in the final layer of each net and binary cross-entropy is used as loss function, but other options, such as tanh activation with hinge loss are also possible. The threshold of 0.5 was set to make predictions on the test set.

Unless otherwise stated, each version is trained with a batch of eight for 20 epochs with Adam optimizer with a learning rate of 0.0001. The learning rate is halved if no improvement comes for seven epochs. Exact match ratio is monitored and the best result is saved for later inference. Hamming loss and accuracy, as defined in Section 3.3.1, are reported for each variant in Table 6.1.

Early while working on the recognition problem it became apparent that it is hard to achieve good results with the prepared dataset therefore heavy data augmentation was applied - here the training dataset is amplified eightfold. Unfortunately, that did not help with overfitting. Dropout and L2 regularization on layer activity was tested but did not give better results.

Problems also appeared when using the more complex net - VGG. In this case, the net was not learning anything new and simply predicting "box" on every image. To "unlock" learning, pre-trained weights from [45] were used on RGB input and other modalities were learned from scratch. Results of the best classification for AlexNet are shown in Figure 6.1

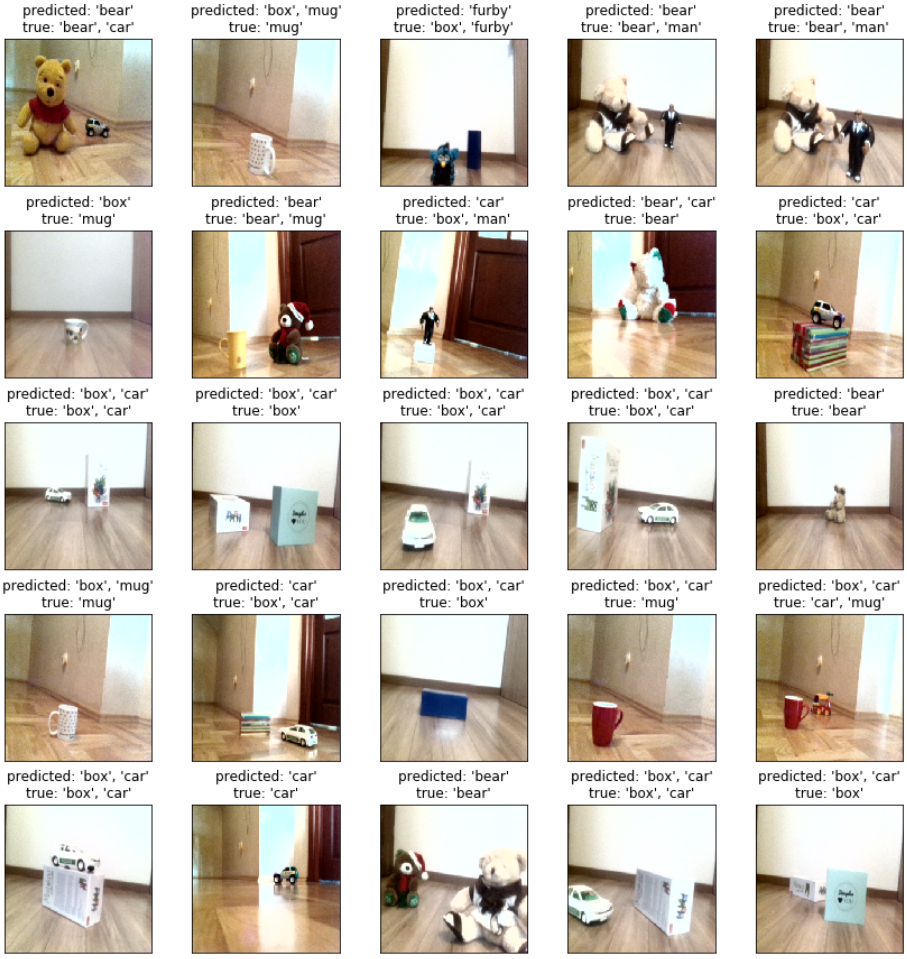


Fig. 6.1. Results of AlexNet early fusion of RGB and HHA data

Table 6.1. Image classification results

Network	Input	Accuracy score	Hamming loss
Alexnet	RGB	0.16	0.23
Alexnet early fusion	RGB, D	0.20	0.21
Alexnet early fusion	RGB, HHA	0.32	0.15
Alexnet late fusion[1]	RGB, D	0.20	0.27
Alexnet late fusion	RGB, HHA	0.16	0.21
VGG[2]	RGB	0.60	0.09
VGG[2] early fusion	RGB, D	0.52	0.11
VGG[2] early fusion	RGB, HHA	0.56	0.12
VGG[2] late fusion	RGB, D	0.40	0.13
VGG[1,2] late fusion	RGB, HHA	0.40	0.15

[1] lower learning rate: 0.00001 instead of 0.0001

[2] using pretrained weights for RGB

6.3. Semantic segmentation

For semantic segmentation, a similar approach was followed - two architectures are implemented: Fully Convolutional Networks and U-Net. Modalities fusion for FCN is implemented as discussed in [9] and [32] - in late fusion, softmax is applied to each modality separately and later summed, while in early fusion the modalities are concatenated before 4096-filters convolutions.

The baseline version of U-Net was implemented as presented in [12] and in Figure 3.6, different to the original model is the use of zero-padded convolutions to avoid shrinking output map and the change in the number of filters in the last layer for multi-class output. As initial runs showed the network was overfitting, a dropout of 0.3 was introduced after every block on the left side of the U.

Different ways of fusing depth are proposed for U-Net than for FCN. In the first approach, only left "arms" of the U are kept separate. Subsequent outputs up to the bottom dropout are concatenated in a pair-wise manner and later used in the expanding part. The intuition behind this is that different features could be extracted from two inputs, but they could be used together to upsample segmentation map. This version can appear similar to the one presented by Dolz et al. in [46], except it is not densely connected, and there are no Inception modules. In the second version, each modality is passed through its own U-shaped network. Outputs are later concatenated and passed together through two more Conv layers to obtain the final result. Figures in Appendix A show two fusions more clearly.

Here seven classes are predicted, not six like in previous experiments because the background is treated as a separate category. Initially, mean Dice loss equal to $1 - mIoU$ as defined in Eq. 3.5 was used for training and validation, but it turned out that it would quickly saturate and after a few epochs, the learning would become very slow regardless of the changes in the learning rate.

In this case batch of four images is used every time as a consensus between training time that is smaller for bigger batches and more frequent updates to the gradient coming from smaller batches. This is also due to the fact, that segmentation models have more parameters, and a bigger batch would often exceed Colab's available RAM. The training dataset is augmented in such a way that there are twice as many images used.

All models were trained with categorical cross-entropy with Adam optimizer set to the starting learning rate of 0.0001. If validation loss did not improve for five consecutive epochs then the rate was halved. Learning and validation losses can be observed in Figures 6.2 and 6.3.

For comparison purposes training is conducted for 50 epochs, but the model is kept only if results improved comparing to the previously saved state. Therefore, the inference on the test set is done using the best parameters that were achieved during the training.

Models are evaluated using four metrics described in Section 3.3.2, but due to large areas marked as background, pixel accuracy and frequency weighted intersection over union do not provide the most beneficial insights. Results for all models are presented in Table 6.2 and detailed results for each label can be found in Appendix B and C for FCN and U-Net respectively. Segmentation masks compared to ground truth for two best models are shown in Figure 6.4

Table 6.2. Results for all semantic segmentation models

Network	Input	Pixel accuracy	Mean pixel accuracy	Mean IoU	Frequency weighted IoU
FCN	RGB	0.896	0.279	0.230	0.879
FCN	HHA	0.878	0.148	0.131	0.877
FCN stack	RGB, D	0.941	0.428	0.363	0.916
FCN early fusion	RGB, D	0.951	0.489	**0.408**	0.928
FCN early fusion	RGB, HHA	0.928	0.413	0.337	0.897
FCN late fusion	RGB, D	0.943	0.429	0.375	0.919
FCN late fusion	RGB, HHA	0.892	0.208	0.190	0.877
U-Net	RGB	0.964	0.693	**0.618**	0.94
U-Net	HHA	0.915	0.273	0.246	0.894
U-Net channel stack	RGB, D	0.956	0.534	0.450	0.935
U-Net fusion 1	RGB, D	0.948	0.536	0.440	0.923
U-Net fusion 1	RGB, HHA	0.955	0.611	0.517	0.927
U-Net fusion 2	RGB, D	0.959	0.627	0.555	0.936
U-Net fusion 2	RGB, HHA	0.95	0.51	0.432	0.928

6.4. Discussion of the results

6.4.1. Recognition

Initially, the task of object recognition appeared to be an easier task of the two, as the desired output is significantly simpler than that of segmentation. The experiments proved otherwise - it was hard to obtain tolerable loss values throughout the training and in some trials, the nets would prefer to safely predict one, the most frequent category. The reason for that may be coming from the dataset, that was not robust enough to train such complex architecture. Transferring weights for VGG partially helped with that problem.

Understandably, comparing AlexNet trained from scratch to pre-trained VGG is not justified. It is, however, interesting to notice that in the case of VGG no form of including depth is beneficial, it is actually predicting mode labels incorrectly. On the other hand - when training AlexNet with distance encoded as HHA, improvement is noticeable, but the simple depth map is not helping. It is, therefore, possible that depth in the form of three-channel HHA can boost image recognition when both modalities are learned alongside each other. Although considered to require less feature extraction, the depth is not relevant when the RGB model is given the advantage of transferring the knowledge from other models.

6.4.2. Semantic segmentation

Most of the setups evaluated in semantic segmentation showed no significant overfitting. When the architecture of choice is Fully Convolutional Net, distance inclusion can indeed yield better results - 0.408 mIoU for an early fusion of RGB and depth compared to 0.279 mIoU for the baseline model. Nonetheless, the impact of depth vanishes when a more complex model is used, as baseline U-Net outperforms every other version tested, both of FCN and U-Net.

One important thing to note is that the learning rate and the number of epochs were set the same for easy comparison, but it seems that FCN would benefit from slightly faster learning and more epochs. In contrast, U-Nets reach plateau a few epochs before the end of the training. This is not surprising when comparing the number of parameters to optimize - single input FCN has about 134 million parameters while single input U-Net - only 34 million.

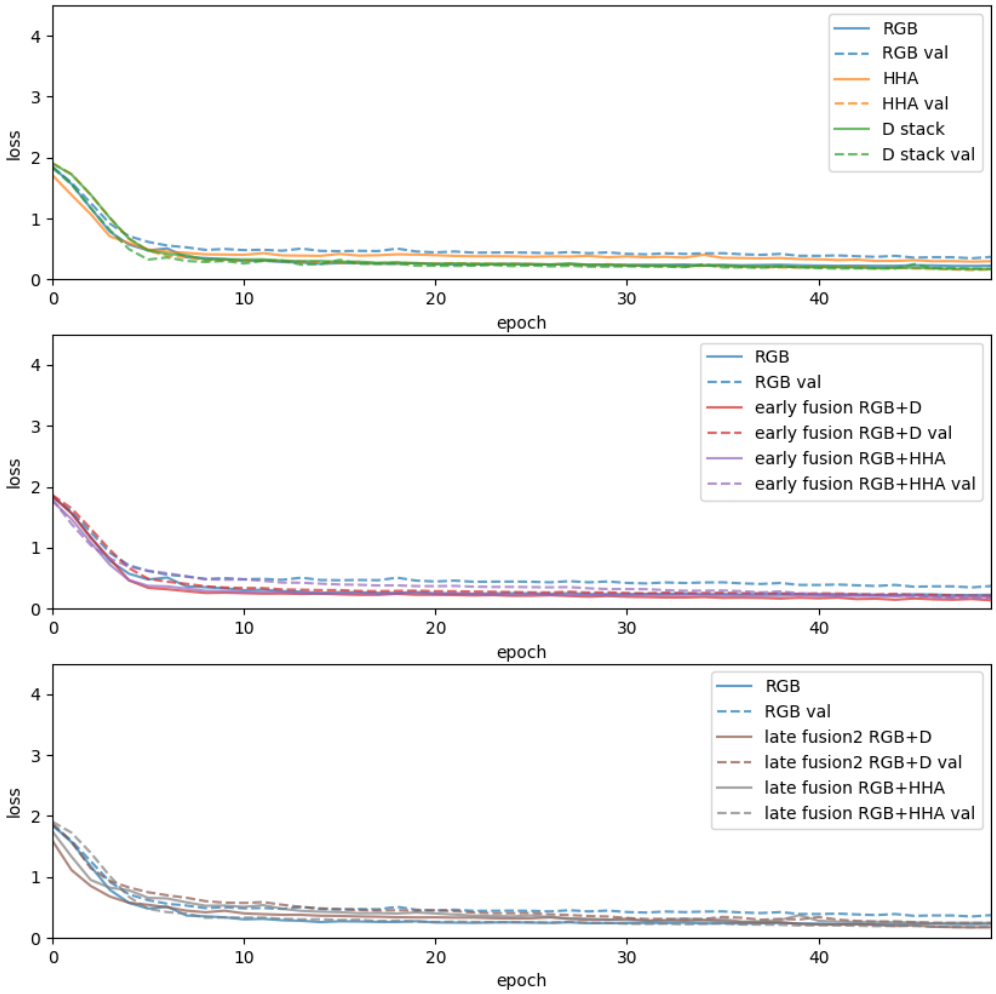


Fig. 6.2. FCN test and validation losses

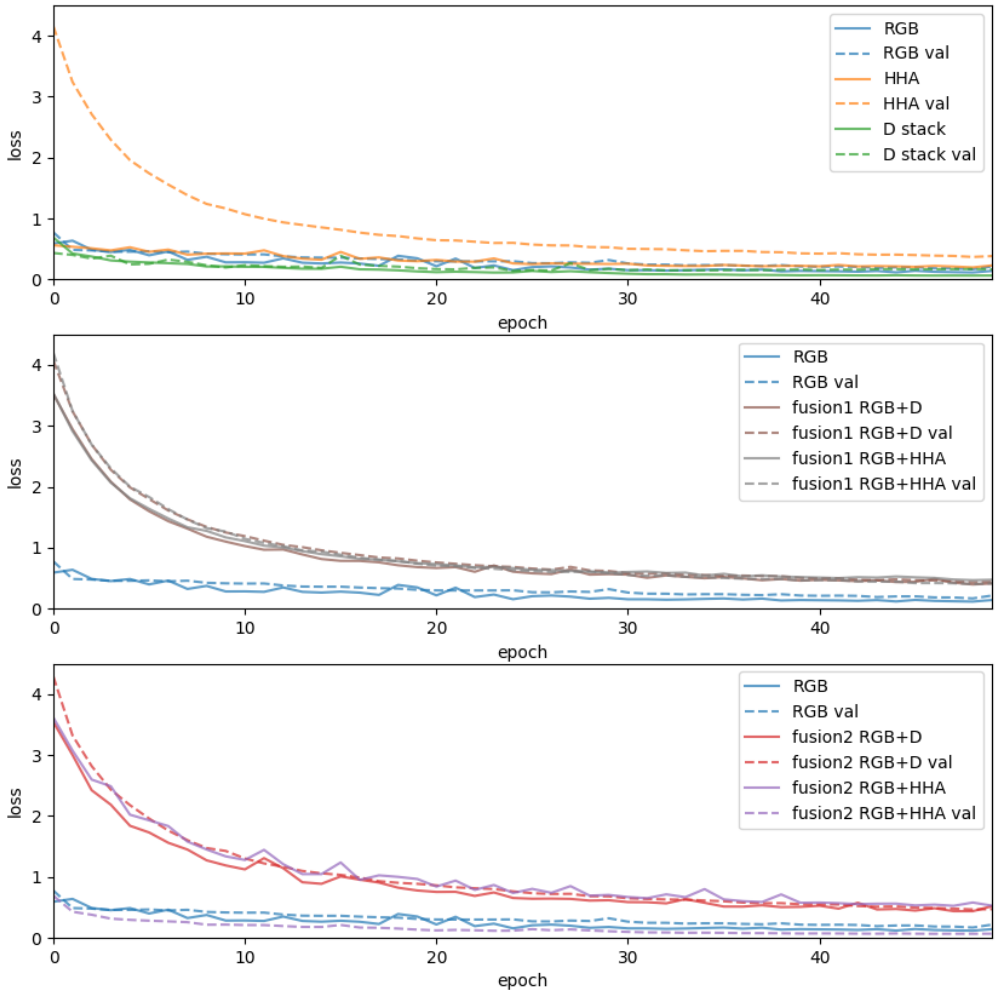


Fig. 6.3. U-Net test and validation losses

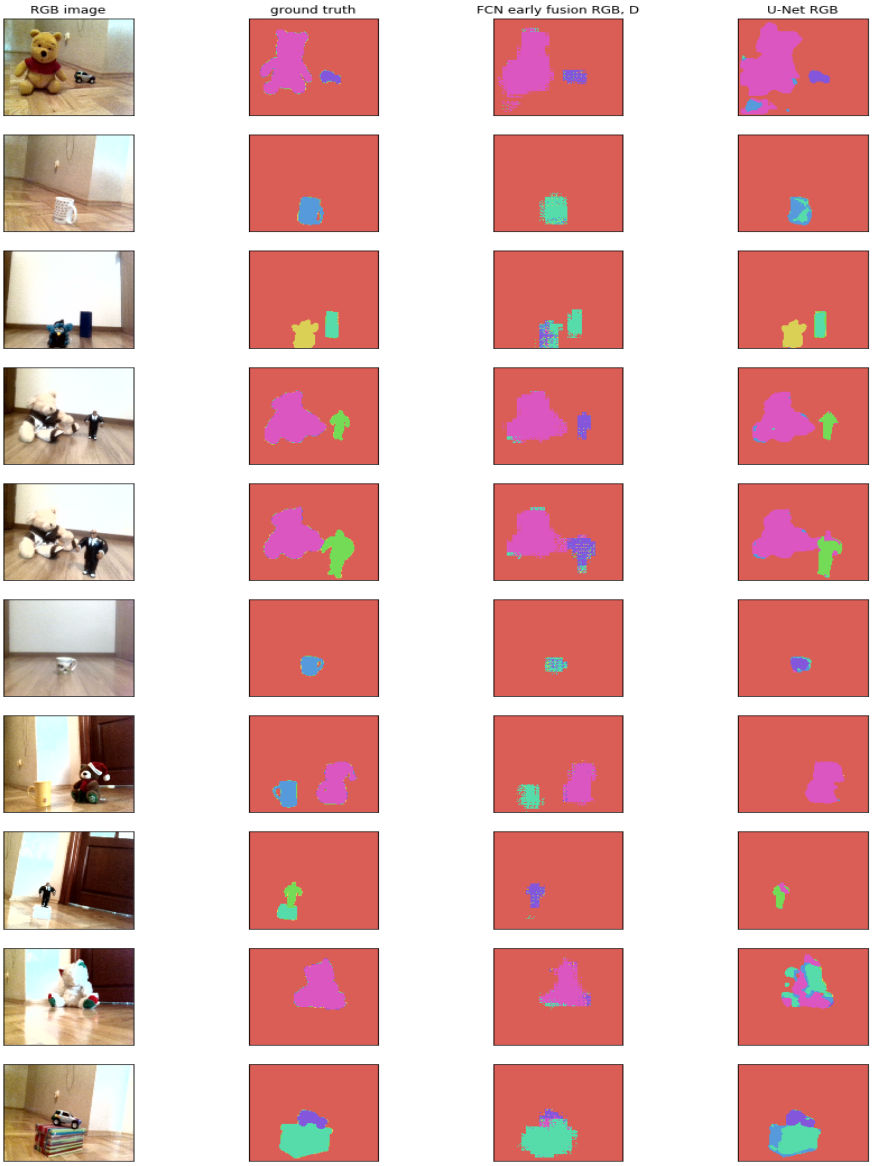


Fig. 6.4. Segmented masks for two best models

7. Future work and conclusion

7.1. Future work

By no means does this work exhaust the subject of using multimodal data in deep learning. Except for the obvious - extending the dataset, some of the possible options could include ensemble learning, where adding the results like in late fusion described above is replaced with voting. Similarly, late- and early-fusion could be put into one, large ensemble to make the best use of both approaches. Nevertheless, this would have to be preceded with a very clever design, as the model to train would grow significantly. There also remain some other fusion ideas that could be investigated when it comes to U-Net. Finally, taking inspiration from Inception nets, using embeddings for depth could bring some interesting results.

7.2. Conclusion

This thesis sought to answer the question of whether incorporating information about the distance to the object can bring significant improvements with regards to image recognition and semantic segmentation. To fulfill this task, a dataset of RGB-D images was collected using the Intel Euclid Development kit. Images were later preprocessed and labeled. Depth was also converted to HHA values that allows it to be used as a three-dimensional feature map.

In both recognition and segmentation, state-of-the-art architectures were implemented. As their usual objective is to classify RGB images, some modifications were made that allowed to incorporate depth into the architecture either as raw distance information or in the encoded form. Two-input, one-output fusion models based on them were prepared to evaluate cases in which feature extraction is done separately on modalities in late and early fusion modes.

Firstly, AlexNet and VGG networks were prepared for image recognition. The former was trained from the ground up while the latter used transferred weights. In the case of AlexNet, the small improvement was shown when incorporating depth, however, the case of VGG showed that this extra information was not strong enough to positively impact the outcome.

Later, Fully Convolutional Network and U-Net were prepared. While FCN has already been reported to be used in a multi-modal setting with depth and HHA, depth fusion has not been

found to be tested in U-Net. Therefore, two approaches to fusion in U-Net were proposed and tested. Indeed, in FCN models, using second modality improved mean intersection over union and mean pixel accuracy scores. Yet, a similar impact was not observed in U-Net. On the contrary, while baseline U-Net outperformed any FCN variant, incorporating depth only deteriorated the results.

Given the above, it is hard to answer definitively whether depth can boost the performance of recognition and semantic segmentation when deep learning techniques are used. In some settings, the impact is noticeable, however similar or better results can be achieved with a more sophisticated architecture trained only on RGB images. It is possible that some industries would still benefit greatly even from the relatively small improvement and will be willing to bear the cost of longer learning time. In most cases, though, it appears that the gain may not be proportional to the resources spent in the process.

7.3. Acknowledgments

This work was supported by the grant from the National Science Centre, Poland DEC-2016/21/B/ST7/02220.

Bibliography

[1] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[2] *A comprehensive guide to convolutional neural networks—the eli5 way*, *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53*, [Online; accessed 14-May-2019].

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, *http://www.deeplearningbook.org*.

[5] J. Brownlee, *A gentle introduction to the ImageNet challenge (ILSVRC)*, *https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/*, [Online; accessed 22-May-2019].

[6] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556*, 2014.

[8] *Review: FCN—fully convolutional network (semantic segmentation)*, *https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1*, [Online; accessed 14-May-2019].

[9] J. Long, E. Shelhamer, and T. Darrell, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[10] E. Shelhamer, J. Long, and T. Darell, "Fully convolutional networks for semantic segmentation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, 2017.

[11] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning", *arXiv preprint arXiv:1603.07285*, 2016.

[12] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, (available on arXiv:1505.04597 [cs.CV]), vol. 9351, Springer, 2015, pp. 234–241.

[13] O. Ronneberger, P. Fischer, and T. Brox, "Dental x-ray image segmentation using a u-shaped deep convolutional network", ISBI, 2015.

[14] M. S. Sorower, "A literature survey on algorithms for multi-label learning",

[15] *Time-of-flight camera*, *https://en.wikipedia.org/wiki/Time-of-flight_camera*, [Online; accessed 04-May-2019].

[16] *Depth sensors comparison*, *http://docs.ipisoft.com/Depth_Sensors_Comparison*, [Online; accessed 04-May-2019].

[17] A. McWilliams, *How a depth sensor works - in 5 minutes*, *https://jahya.net/blog/how-depth-sensor-works-in-5-minutes/*, [Online; accessed 04-April-2019], 2013.

[18] A. M.R., J. T., L. P., *et al.*, "Kinect depth sensor evaluation for computer vision applications, technical report ece-tr-6", Aarhaus University, Tech. Rep., 2012.

[19] M. Firman, "RGBD Datasets: Past, Present and Future", in *CVPR Workshop on Large Scale 3D Data: Acquisition, Modelling and Analysis*, 2016.

[20] *Libfreenect vs OpenNI*, *https://stackoverflow.com/questions/19181332/libfreenect-vs-openni*, [Online; accessed 21-May-2019].

[21] *Align depth and color frames – depth and RGB registration*, *https://www.codefull.org/2016/03/align-depth-and-color-frames-depth-and-rgb-registration/*, [Online; accessed 21-April-2019].

[22] *Kinect calibration*, *http://nicolas.burrus.name/index.php/Research/KinectCalibration\#tocLink6*, [Online; accessed 10-April-2019].

[23] S. Huh and G. Kim, "Human pose estimation from depth image using visibility estimation and key points", in *International Conference on Digital Human Modeling and Applications in Health, Safety, Ergonomics and Risk Management*, Springer, 2013, pp. 333–342.

[24] W.-C. Cheng, "Pedestrian detection using an RGB-depth camera", in *2016 International Conference on Fuzzy Theory and Its Applications (iFuzzy)*, IEEE, 2016, pp. 1–3.

[25] *Why is occlusion in augmented reality so hard?*, *https://hackernoon.com/why-is-occlusion-in-augmented-reality-so-hard-7bc8041607f9*, [Online; accessed 21-May-2019].

[26] *What is a ToF camera and which phones have it? Time-of-Flight sensor explained*, *https://www.pocket-lint.com/phones/news/147024-what-is-a-time-of-flight-camera-and-which-phones-have-i*, [Online; accessed 20-June-2019].

[27] P. K. Nathan Silberman Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from RGBD images", in *ECCV*, 2012.

[28] S. Song, S. P. Lichtenberg, and J. 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.

[29] A. Janoch, S. Karayev, Y. Jia, *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.

[30] *NYU depth dataset v2*, *https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html*, [Online; accessed 10-April-2019].

[31] L. Herranz, *Learning RGB-D features for images and videos*, *http://www.lherranz.org/2018/10/17/rgbd-features/*, [Online; accessed 15-June-2019].

[32] B. Ionescu, J. Benois-Pineau, T. Piatrik, *et al.*, *Fusion in Computer Vision: Understanding Complex Visual Content*. Springer, 2014.

[33] S. Gupta, P. Arbelaez, and J. 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.

[34] S. Gupta, R. Girshick, P. Arbeláez, *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.

[35] C. J. Holder, T. P. Breckon, and X. Wei, "Depth not needed-an evaluation of RGB-D feature encodings for off-road scene understanding by convolutional neural network", *arXiv preprint arXiv:1801.01235*, 2018.

[36] I. Corporation, *Intel euclid development kit user guide*, English, Intel Corporation, Feb. 15, 2017, 20 pp.

[37] *Bags*, *http://wiki.ros.org/Bags*, [Online; accessed 09-June-2019].

[38] *Intel's community forum*, *https://forums.intel.com/s/case/5000P00000mKrDmQAK/follow-up-on-missing-topic-when-reading-rosbag-file*, [Online (login needed); accessed 21-March-2019].

[39] C. XiaoKang, *Depth2HHA-python*, *https://github.com/charlesCXK/Depth2HHA-python*, [Online; accessed 10-April-2019].

[40] Deep Software, *Software: Supervisely*, *https://app.supervise.ly*, [Online; accessed 10-April-2019].

[41] *TensorFlow guide - high level API*, *https://www.tensorflow.org/guide\#high_level_apis*, [Online; accessed 11-April-2019].

[42] T. Oliphant, *NumPy: A guide to NumPy*, USA: Trelgol Publishing, [Online; accessed 12-April-2019], 2006–.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[44] P. Sharma, *Build your first multi-label image classification model in Python*, *https://www.analyticsvidhya.com/blog/2019/04/build-first-multi-label-image-classification-model-python/*, [Online; accessed 28-May-2019].

[45] F. Chollet, *Vgg16 weights*, *https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5*, [Online; accessed 1-May-2019].

[46] J. Dolz, I. B. Ayed, and C. Desrosiers, "Dense multi-path u-net for ischemic stroke lesion segmentation in multiple image modalities", *CoRR*, vol. abs/1810.07003, 2018. arXiv: *1810.07003*.

List of Figures

2.1 Examples of types of unit connections in different neural networks 10

3.1 AlexNet architecture split to work on two GPUs 13
3.2 Inception with dimension reductions 14
3.3 VGG net 15
3.4 Input-output pair used for semantic segmentation of an image 16
3.5 Illustration of DAG skip connections aimed at refining the coarse output of FCN and preserving high-level information 18
3.6 U-net architecture 19

4.1 Emitted infra-red light and image registered by the camera 24
4.2 Densely segmented samples from NYU Depth Dataset V2 28
4.3 Early and late fusion comparison 29
4.4 Comparison of RGB and HHA encoded depthmaps 30

5.1 Intel Euclid Development Kit 31
5.2 Example of displacement between fields of view in Intel Euclid camera 33
5.3 Depth registration and inpainting 35
5.4 HHA-encoded images 37
5.5 Interface of Supervisely web application to annotate images 38
5.6 Dataset statistics, based on information from Supervisely 39
5.7 Examples of data augmentation 40

6.1 Results of AlexNet early fusion of RGB and HHA data 42
6.2 FCN test and validation losses 47
6.3 U-Net test and validation losses 48
6.4 Segmented masks for two best models 49

A.1 U-Net modified for fusing modalities, version 1 60

A.2 U-Net modified for fusing modalities, version 2 61

List of Tables

6.1 Image classification results 43
6.2 Results for all semantic segmentation models 45

B.1 Per-class results for FCN models 62

C.1 Per-class results for U-Net models 64

A. U-Net fusion variants

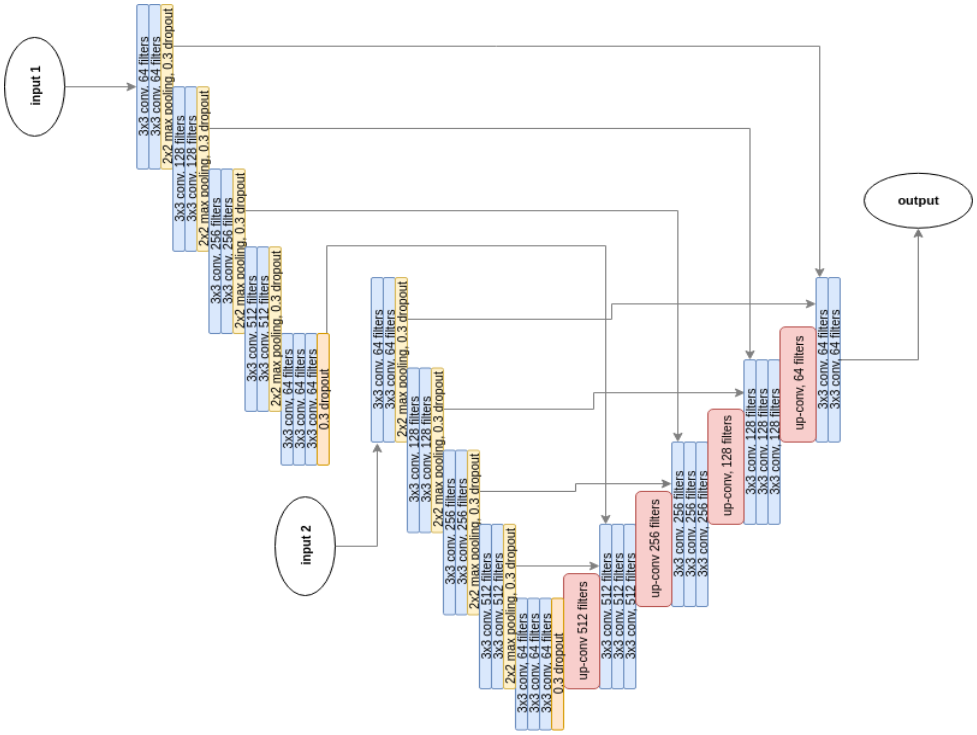


Fig. A.1. U-Net modified for fusing modalities, version 1

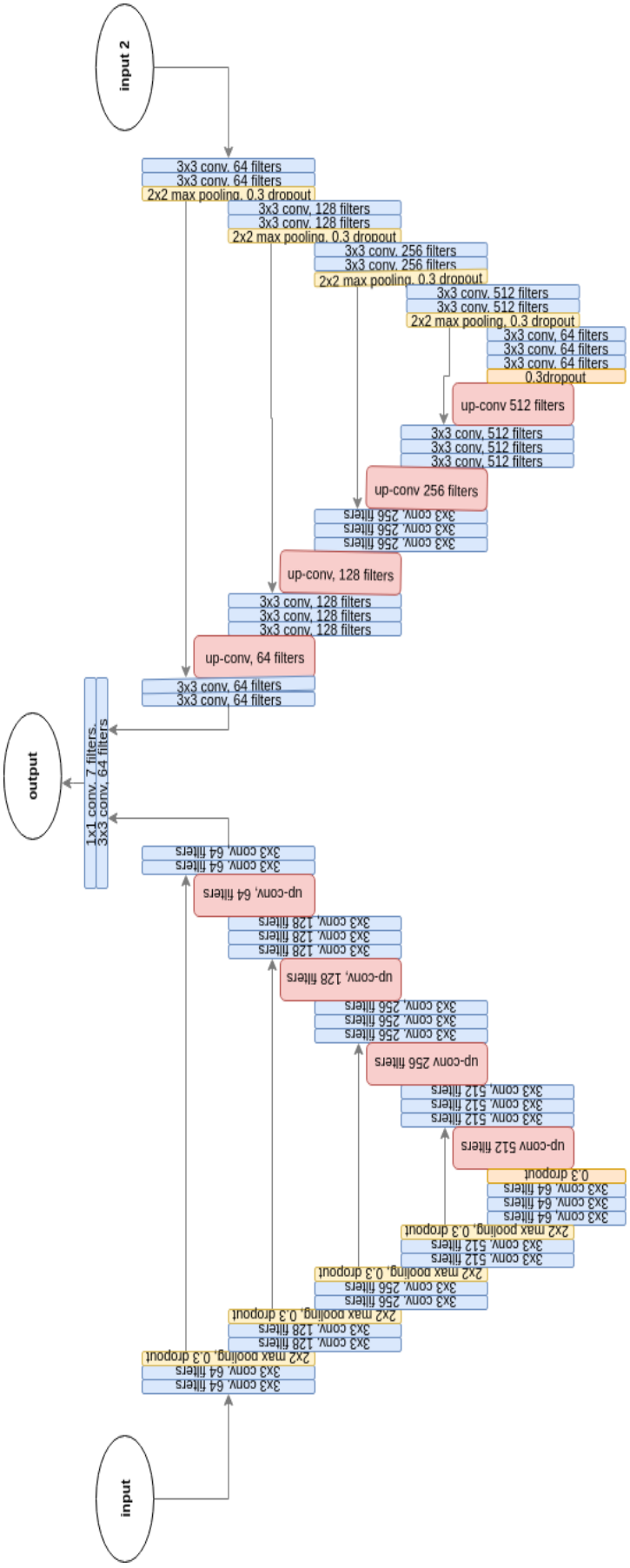


Fig. A.2. U-Net modified for fusing modalities, version 2

B. Per-class results for FCN models

Table B.1. Per-class results for FCN models

Network	Input	class	IoU	precision	recall	F1 score
FCN	RGB	background	0.922	0.926	0.996	0.96
		furby	0.049	0.092	0.096	0.094
		man	0.017	0.053	0.025	0.034
		box	0.1	0.484	0.112	0.182
		mug	0.082	0.246	0.11	0.152
		car	0.244	0.422	0.367	0.393
		bear	0.193	0.461	0.25	0.324
Unet	RGB	background	0.882	0.882	1.0	0.937
		furby	0.002	0.018	0.002	0.004
		man	0.001	0.029	0.001	0.003
		box	0.008	0.237	0.008	0.015
		mug	0.002	0.035	0.002	0.004
		car	0.002	0.057	0.003	0.005
		bear	0.022	0.564	0.022	0.043
FCN stack	RGB, D	background	0.96	0.967	0.992	0.979
		furby	0.033	0.094	0.048	0.064
		man	0.027	0.15	0.032	0.053
		box	0.66	0.783	0.807	0.795
		mug	0.054	0.204	0.069	0.103
		car	0.296	0.456	0.457	0.457
		bear	0.512	0.798	0.588	0.677

Table B.1 Per-class results for FCN models, continued

Network	Input	class	IoU	precision	recall	F1 score
FCN early fusion	RGB, D	background	0.969	0.981	0.988	0.984
		furby	0.02	0.091	0.024	0.039
		man	0.014	0.123	0.015	0.027
		box	0.634	0.731	0.827	0.776
		mug	0.099	0.443	0.113	0.18
		car	0.402	0.534	0.62	0.574
		bear	0.716	0.831	0.838	0.834
FCN early fusion	RGB, HHA	background	0.958	0.969	0.988	0.978
		furby	0.02	0.037	0.041	0.039
		man	0.041	0.081	0.076	0.078
		box	0.502	0.827	0.56	0.668
		mug	0.069	0.138	0.12	0.128
		car	0.239	0.32	0.485	0.385
		bear	0.53	0.782	0.622	0.693
FCN late fusion	RGB, D	background	0.956	0.963	0.993	0.978
		furby	0.039	0.135	0.052	0.075
		man	0.026	0.25	0.028	0.05
		box	0.577	0.8	0.675	0.732
		mug	0.043	0.442	0.046	0.083
		car	0.363	0.721	0.422	0.532
		bear	0.623	0.746	0.791	0.768
FCN late fusion	RGB, HHA	background	0.893	0.896	0.997	0.944
		furby	0.008	0.16	0.009	0.017
		man	0.0	nan	0.0	nan
		box	0.101	0.717	0.105	0.183
		mug	0.001	0.065	0.001	0.003
		car	0.079	0.596	0.083	0.146
		bear	0.246	0.806	0.262	0.395

C. Per-class results for U-Net models

Table C.1. Per-class results for U-Net models

Network	Input	class	IoU	precision	recall	F1 score
U-Net	RGB	background	0.975	0.983	0.991	0.987
		furby	0.439	0.954	0.449	0.61
		man	0.47	0.834	0.518	0.639
		box	0.726	0.873	0.812	0.842
		mug	0.306	0.452	0.487	0.469
		car	0.662	0.903	0.713	0.797
		bear	0.745	0.826	0.885	0.854
U-Net	HHA	background	0.925	0.927	0.998	0.961
		furby	0.0	nan	0.0	nan
		man	0.0	nan	0.0	nan
		box	0.358	0.645	0.447	0.528
		mug	0.0	nan	0.0	nan
		car	0.023	0.137	0.026	0.044
		bear	0.418	0.905	0.438	0.59
U-Net fusion 1	RGB, D	background	0.976	0.983	0.992	0.988
		furby	0.208	0.472	0.272	0.345
		man	0.189	0.738	0.202	0.317
		box	0.548	0.844	0.61	0.708
		mug	0.241	0.56	0.297	0.388
		car	0.309	0.436	0.514	0.472
		bear	0.607	0.668	0.869	0.755

Table C.1 Per-class results for U-Net models, continued

Network	Input	class	IoU	precision	recall	F1 score
U-Net fusion 1	RGB, HHA	background	0.972	0.986	0.986	0.986
		furby	0.336	0.678	0.4	0.503
		man	0.172	0.937	0.174	0.294
		box	0.652	0.747	0.838	0.79
		mug	0.288	0.575	0.367	0.448
		car	0.569	0.764	0.69	0.725
		bear	0.626	0.722	0.825	0.77
U-Net fusion 2	RGB, D	background	0.972	0.976	0.996	0.986
		furby	0.344	0.586	0.455	0.512
		man	0.349	0.809	0.38	0.517
		box	0.678	0.817	0.8	0.808
		mug	0.276	0.585	0.344	0.433
		car	0.594	0.824	0.679	0.745
		bear	0.67	0.882	0.736	0.803
U-Net fusion 2	RGB, HHA	background	0.972	0.978	0.994	0.986
		furby	0.341	0.574	0.457	0.508
		man	0.098	0.724	0.102	0.178
		box	0.572	0.715	0.74	0.727
		mug	0.002	0.172	0.003	0.005
		car	0.415	0.823	0.456	0.587
		bear	0.622	0.722	0.819	0.767