Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2018

Neural networks for semantic segmentation in the food packaging industry

Mattias Carlsson

LINKÖPING
UNIVERSITY

Master of Science Thesis in Electrical Engineering

Neural networks for semantic segmentation in the food packaging industry

Mattias Carlsson

LiTH-ISY-EX--18/5113--SE

Supervisor: **Felix Järemo-Lawin**
ISY, Linköpings universitet
Erik Ringaby
SICK IVP

Examiner: **Per-Erik Forssén**
ISY, Linköpings universitet

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

Industrial applications of computer vision often utilize traditional image processing techniques whereas state-of-the-art methods in most image processing challenges are almost exclusively based on convolutional neural networks (CNNs). Thus there is a large potential for improving the performance of many machine vision applications by incorporating CNNs.

One such application is the classification of juice boxes with straws, where the baseline solution uses classical image processing techniques on depth images to reject or accept juice boxes. This thesis aim to investigate how CNNs perform on the task of semantic segmentation (pixel-wise classification) of said images and if the result can be used to increase classification performance.

A drawback of CNNs is that they usually require large amounts of labelled data for training to be able to generalize and learn anything useful. As labelled data is hard to come by, two ways to get cheap data are investigated, one being synthetic data generation and the other being automatic labelling using the baseline solution.

The implemented network performs well on semantic segmentation, even when trained on synthetic data only, though the performance increases with the ratio of real (automatically labelled) to synthetic images. The classification task is very sensitive to small errors in semantic segmentation and the results are therefore not as good as the baseline solution. It is suspected that the drop in performance between validation and test data is due to a domain shift between the data sets, e.g. variations in data collection setups and straw and box type, and fine-tuning to the target domain could definitely increase performance.

When trained on synthetic data the domain shift is even larger and the performance on classification is next to useless. It is likely that the results could be improved by using more advanced data generation, e.g. a generative adversarial network (GAN), or more rigorous modelling of the data.

Acknowledgments

First of all I would like to thank SICK IVP for giving me the opportunity to write my master thesis. I would like to thank my supervisors Erik Ringaby and Felix Järemo-Lawin for their help with technical support and with writing this thesis, and finally a special thank to Sofia Nilsson at SICK IVP for help with the data set and figuring out what is what.

Linköping, November 2017
Mattias Carlsson

Contents

1 Introduction

Semantic segmentation is the task of classifying images on a per-pixel level, leading to a higher level of scene understanding than mere classification. With semantic information an automated system can tell what and where objects are in the image, allowing e.g. automatic shelf- or bin-picking of goods and enabling the system to make decisions with increased understanding of the scene.

The semantic information can also be used for classification, where a product is classified as faulty if something is missing, in the wrong position, or found in too large or too small quantity in the scene.

Industrial applications of computer vision often have high requirements on performance, and reliable and fast detection of errors can be critical. Faulty products have the risk of eventually causing jams further down the production line, leading to costly downtime and in turn to increased waste if more products get damaged or ruined in the process.

1.1 Motivation

In the production of juice boxes with straws, one computer vision application is locating the straw on the box. If the straw is missing the product is undesirable to end-customers and will likely remain unsold in the store, leading to unnecessary waste, transportation, and storage costs. If the straw is present but misplaced it may lead to packaging problems, jams, and unnecessary downtime of the production line.

The existing baseline algorithm provided by *SICK IVP*, where this thesis was conducted, utilizes traditional image processing on depth images to locate the box and the straw to classify the product. In short the baseline algorithm fits a plane to the box in the image to compensate for box tilt, and locates the straw using Sobel-filters, the Canny edge detector and the Hough-transform. With prior knowledge of the box's and straw's expected dimensions, the baseline algorithm performs well mostly but struggles to locate the straw correctly in a few cases, e.g. when the straw is transparent or when it's not oriented as expected. An example result of the baseline solution is shown in figure 1.1. Using semantic segmentation one could possibly locate the straw more accurately while at the same time being agnostic to *expected* orientation and dimension and instead solely learn from the data itself.

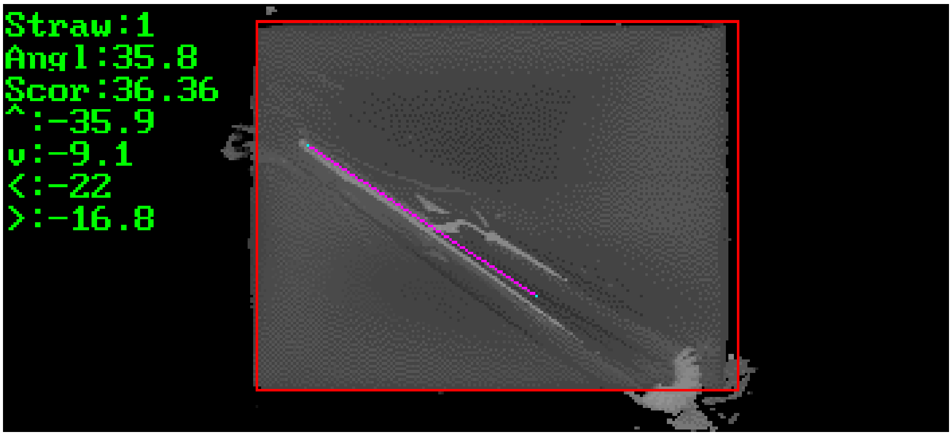


Figure 1.1: *Example depth image of juice box with straw. The red rectangle and the magenta line is where the baseline solution finds the box and straw respectively, and the green text tells the results from the baseline solution. The transparency of the straw leads to it being almost indistinguishable from the box and in this case the baseline fails to find the full length of the straw. The image is rotated 90° clockwise.*

Most state-of-the-art solutions in semantic segmentation are based on CNNs (Convolutional Neural Networks). CNNs have gained much attention in recent years following the success of AlexNet[23] on ImageNet-LSVRC[7]. ImageNet-LSVRC is an image classification challenge consisting of over a million images belonging to 1000 different classes where AlexNet outclassed previous state-of-the-art solutions based on hand-crafted features.

CNN-based solutions have ever since been the top contestants with networks becoming deeper and deeper (e.g. VGG16[31], ResNet[15]) and were even claimed to surpass human-level performance[16], making them an interesting choice for industrial applications.

The current baseline algorithm is able to run fast and reliable on the computationally limited hardware found in the integrated solution. Using a CNN on the same hardware is not feasible but the rapid development of accelerated processing in the form of embedded GPUs (Graphic Processing Units) may make CNNs a viable option in the near future.

The recent successes of CNNs is partly thanks to the increase in large annotated image sets necessary for training deep models. Large task-specific data sets for machine vision applications are however hard to come by and hand-labelling is both costly and time-consuming leading to the need of other ways to acquire data for training.

A possible solution often utilized in research is transfer-learning, where a model is pre-trained on a large hand-labelled data set (e.g. ImageNet) then fine-tuned on a smaller task-specific data set, sometimes even outperforming models trained solely on the task-specific data. However, the large discrepancy between publicly available data sets, often consisting of natural images, and the task-specific data sets found in industrial applications suggests that transfer-learning is unsuitable.

Another possibility is to generate synthetic data which can be very cost effective if it is easy to reproduce the underlying structure of real world data, but may however lack details and features found in real images which could lead to sub-optimal performance in real world applications.

1.2 Purpose

The goal of this thesis is to investigate how CNNs behave on depth images found in some machine vision applications and see if CNNs can be used in an industrial environment by studying the task of semantic segmentation of depth images.

Since hand-labelling images is time consuming and costly and large annotated data sets are hard to come by for the task specific applications often found in machine vision, this thesis will investigate how generating and training on synthetic data compares to using automatically labelled data.

1.3 Problem formulation

This thesis will try to answer the following questions:

- Can a CNN be used for semantic segmentation of depth images of juice boxes from a production environment?
- Can the resulting segmentation be used for binary classification of the juice boxes?
- How does the CNN compare to the current baseline solution on classification?
- How does the CNN perform when trained on synthetic data?

The specific task handled in this thesis is the task of semantic segmentation and classification of depth images.

The depth images contain single juice boxes, with a plastic-wrapped straw glued onto them, and the juice box is classified as faulty and rejected if the straw is missing or protruding too much from either of the edges of the juice box. This is a binary classification problem with two classes, *faulty* or *not faulty*, and is solved using traditional image processing techniques in the baseline solution.

Semantic segmentation simplifies the representation of the image and gives contextual information which can be used for classification of the juice box, and allows for changing the acceptance threshold without needing to change any model parameters.

The images are collected at different locations with varying setups using range cameras. A range camera uses laser triangulation to measure depth, which gives artifacts in regions with very reflective or transparent material and areas where the camera cannot reliably measure the depth. The artifacts, missing data, and various setups lead to domain shifts between train, validation, and test data, i.e. the underlying data distribution differs between the data sets and may be challenging for a CNN.

The evaluation of both semantic segmentation and classification is done on a hand-labelled test data set, with known straw protrusions in millimetres and hand-labelled binary masks for the segmentation.

1.3.1 Limitations

Due to the relatively short time span of this thesis and since CNN usually are very complex models consisting of millions of parameters (e.g. VGG16[31] has around 138M trainable parameters) and requiring up to billions of floating point operations during inference, it is outside the scope of this thesis to get a CNN that runs on the same hardware and that compares in evaluation time to the baseline solution.

The number of architecture configurations and hyper-parameters tested is also limited due to the long training times needed.

1.4 Related work

Semantic segmentation of images is a well studied subject where traditional approaches use hand-crafted features e.g. color, texture, context, and layout[29], or HOG (Histogram of Oriented Gradients) used in random forests[28].

More recent work are often based on CNN architectures for image classification, where AlexNet[23] and VGG16[31] are popular choices. The architectures are extended or modified in various ways to produce a class score at every pixel instead of a class-score at an image level. A common approach is to replace the last fully connected layers with convolutional layers and upsample the low resolution score map to the input size.

Plain upsampling by simply rescaling to desired size gives crude results and one way to regain some spatial resolution is to use indices based upsampling in an encoder-decoder structure as in SegNet-basic[2]. An encoder-decoder does step-wise downsampling followed by step-wise upsampling back to the input size. With indices-based upsampling the encoder-decoder can remember where the activations were the highest in previous layers and thus keep some spatial information.

SegNet[3] follows the encoder-decoder with indices based upsampling from SegNet-basic but with a major change in architecture to be based on popular

VGG16 instead leading to a large increase in performance. Bayesian-SegNet[22] further increased performance by adding Monte-Carlo sampling to its output but at the cost of increased run-time.

Another network based on VGG16 is FCN[24], which introduced skip connections. Skip connections combine the coarse predictions from deeper layers with more detailed information in shallow layers, making the results more fine-grained.

Dilated convolutions are convolutions blown up in size to increase the effective receptive field without increasing the number of parameters used. This allows small 3×3 kernels to have the receptive field of e.g. 5×5 or 7×7 but without blowing up the number of parameters. Dilated convolutions are successfully used in FCN-based DeepLab[4] and DilatedNet[32], where the latter introduces the concept of multi-scale context aggregation, which adds multiple dilated convolutions just before the classifier to vastly increase the receptive field and results in increased accuracy.

Another interesting approach is using residual networks which enables the use of very deep networks - some implementations of ResNet have consisted of up to 1000 convolutional layers. The 100-Layer Tiramisu[21] builds on the idea of [18] by concatenating the output of previous layers to the output of the current layer. This means that deeper layers have access to earlier layers output and gives more fine-grained result and makes very deep networks viable.

One of the earlier approaches to semantic segmentation was Couprie et al.[6], which proposed a multi-scale approach using three separate streams for different scales. It also utilized the additional depth information for increased accuracy and superpixels for aggregating and smoothing predictions.

The approaches above deal both with RGB and RGB-D images and generalize to any number of channels but do this in a naive way treating the additional depth information as another color channel. [14] instead uses the depth information to encode additional information in the data, including height above ground and angle with the gravity vector. This approach gives a slight increase in performance of their superpixel segmentation implementation.

[9] uses a simple depth to jet color encoding to get around the heavy computations needed by the additional data encoding of [14]. This allows a network pre-trained on color images to process the depth images. Using a two-stream network for color and depth respectively, and concatenation of the predictions, [9] get even higher accuracy on classification of RGB-D images.

Generative adversarial networks (GANs) are an interesting type of network for data generation consisting of a generator and discriminator. The generator and discriminator take part in a game where the discriminator tries to tell if an image is real or fake, and the generator tries to produce fake images that look as real as possible (see e.g. [12] for more information).

DCGAN[26] uses GAN in an unsupervised fashion to learn representations in the data, which they use as an effective pre-training step to reduce the amounts of data needed for training for the desired task.

SimGAN[30] applies a GAN as a refiner network It uses unlabelled real images to enhance and refine labelled synthetic images and achieves impressive results.

In [25] 3D CAD models are used to generate synthetic data for training. It achieves good results even when generated data lack real-world-like noise and variations.

1.5 Thesis outline

A short introduction to theory and background of CNN and semantic segmentation is given in chapter 2. Chapter 3 describes the method used and steps taken to solve the task, including the procedure for evaluation. Implementation details and results of evaluation are found in chapter 4. An analysis of the results with discussion and thoughts on the chosen method is found in chapter 5 and a conclusion with thoughts on future development is found in chapter 6.

2 Theory

This chapter introduces some background theory to CNNs and semantic segmentation.

2.1 Neural networks

According to [12] artificial neural networks are loosely based on how the brain functions and learns. The biological neuron receives input signals in the form of electric signals through its dendrites, and "fires" through its axon if it gets the correct excitation[10]. Similarly an artificial neuron, the perceptron, receives its input represented as numbers **x**, computes a weighted sum, and maps the weighted sum with an activation function[27].

A stylized visualization of an artificial neuron is shown in figure 2.1.

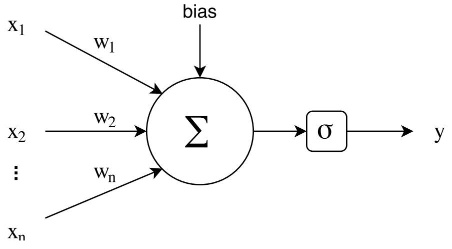


Figure 2.1: *A visualization of an artificial neuron with weights w and bias. It computes a weighted sum of its input x and maps the weighted sum through an activation function σ to produce the output y.*

Mathematically the artificial neuron can be written as

$$y = f(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}) = \sigma\left(\sum_i w_i x_i\right) \tag{2.1}$$

where $\mathbf{x}$ is the input, $\mathbf{w}$ the weights, and σ the activation function. The bias is omitted for readability.

The first artificial neuron used the step-function as activation function whereas more common activation functions today are the sigmoid the hyperbolical tangent (*tanh*), and the Rectified Linear Unit (RELU)[12].

The perceptron is a linear classifier, meaning it can separate its input into two classes with a hyper-plane (straight line in 2D). This limits the number of problems it can solve to linearly separable problems.

By combining several neurons into layers and stacking layers after each other, it is possible to learn non-linear class boundaries. This type of network is often called Multi-layer perceptron (MLP) or feedforward network and the neurons are often referred to as nodes[12]. An example MLP is shown in figure 2.2.

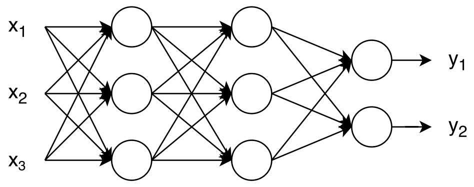


Figure 2.2: *MLP with 3 input nodes, 2 hidden layers with 3 nodes each, and 2 output nodes.*

An MLP consists of an input layer, one or more hidden layers, and an output layer. The layers are *fully connected*, meaning that the input to every node in a layer is the output of every node in the previous layer.

2.2 Convolutional neural networks

Since digital images typically are made up of up to and above millions of values, this also means that MLPs need to have up to and above millions of parameters for every single node at the input layer. This is impractical since many features found in images, e.g. edges and corners, are local in their nature and independent on the global context, making many of the weights insignificant. This also means that a large number of nodes are needed to find similar features in different parts of the image.

By instead limiting the receptive field of each neuron to a small neighbourhood and letting them sweep over the input to produce an output *feature map*, one gets a CNN.

Analogous to the perceptron, the output of a node in a CNN is the weighted sum of the input, in a local neighborhood, passed through an activation function.

The output feature map of a single node at location (m, n) can be computed using the mathematical operation convolution as

$$f(\mathbf{x};\mathbf{w})_{m,n} = (\mathbf{x} * \mathbf{w})_{m,n} = \sum_{i=-\omega}^{\omega} \sum_{j=-\omega}^{\omega} w_{i,j} x_{m-i,n-j} \tag{2.2}$$

where $\mathbf{x}$ is the input, $\mathbf{w}$ the kernel (the weights) with size $k = 2 \cdot \omega + 1$. Here $\mathbf{w}$ is assumed to be of square shape but other shapes are also possible.

A layer that performs the convolutional operation is called a convolutional layer. Another common layer is the *max-pooling* layer. Similarly to a convolutional layer it has a limited receptive field - known as pooling size - and sweeps over the input, but instead of computing a weighted sum over all the input channels it returns the maximum of its inputs channel-wise. E.g. for an RGB-color image max-pooling processes the color channels independently whereas a convolutional layer processes them all at the same time.

Max-pooling can be defined as

$$f(\mathbf{z}) = \max(z_1, \ldots, z_k) \tag{2.3}$$

where $\mathbf{z}$ here is a local image patch. Common pooling sizes are 2×2 or 3×3. Max-pooling is most often used in conjunction with downsampling using the same rate as its pooling size to reduce the spatial resolution while still keeping the highest activations.

The rate of downsampling is often referred to as strides, i.e. a stride of 2 means that the layer has a spatial step-size of 2 in its input leading to a downsampling with factor 2. A stride of 1 thus means computing the output at every input position and a stride of 1/2 corresponds to taking half steps in the input leading to upsampling with a factor 2. Downsampling and upsampling can also be achieved by letting the convolutional layer take different strides in the input, where upsampling with a convolutional layer is often called fractionally-strided convolution or "deconvolution"[8].

2.2.1 Activation functions

According to [23][12], the RELU activation function is often preferred over the sigmoid and *tanh* since it has less computational overhead and often allows faster convergence when training CNNs. [12] defines the RELU as

$$\sigma(z) = \max(0, z) \tag{2.4}$$

which exists in different variations, e.g. Leaky-RELU and parametric RELU.

In multi-class classification the *softmax* activation function is often used at the final layer and is defined as

$$y_i' = \sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.5}$$

which outputs normalized class probabilities y_i', i.e. y_i' can be seen as the predicted probability of belonging to class i[12].

2.2.2 Architecture

CNNs come in many different shapes and sizes. An example architecture with three 3×3 convolutional layers is described in table 2.1.

Table 2.1: *Example CNN architecture using* 3×3 *convolutional layers with RELU activation function (conv),* 2×2 *max-pooling (max-pool) and a final fully-connected layer (fc) with softmax activation, producing class probabilities for ten classes.*

Layer	**kernel size**	**filters**	**stride**	**input shape**	**output shape**	**parameters**
conv	3×3	32	1	$32 \times 32 \times 1$	$30 \times 30 \times 32$	320
max-pool	2×2	-	2	$30 \times 30 \times 32$	$15 \times 15 \times 32$	-
conv	3×3	64	1	$15 \times 15 \times 32$	$13 \times 13 \times 64$	18 496
max-pool	2×2	-	2	$13 \times 13 \times 64$	$6 \times 6 \times 64$	-
conv	3×3	128	1	$6 \times 6 \times 64$	$4 \times 4 \times 128$	73 856
fc	4×4	10	-	$4 \times 4 \times 128$	$1 \times 1 \times 10$	20 490
softmax	-	-	-	10	10	-

The example network takes a 32×32 image as input (e.g. a grayscale image), the first convolutional layer has 32 kernels which all sweep over the input and thus outputs 32 feature maps. These 32 feature maps are then downsampled in the following max-pooling layer resulting in 32 feature maps of size 15×15. All these feature maps are then used as input to the second convolutional layer, which has 64 kernels and thus outputs 64 feature maps. The next and final convolutional layer results in 128 feature maps of size 4×4 which are then mapped through a fully connected layer (fc) with softmax activation outputting class probabilities for ten different classes. The fully connected layer is similar to a hidden layer in an MLP, meaning its connected to every node in the previous layer. The fully connected layer is responsible for producing unnormalized class scores, and the softmax layer normalizes these class scores to a probability distribution over the classes.

The receptive field of a node in a CNN is the size of the input that affects that particular node's output. For the example CNN given in table 2.1 the first convolutional layer has a receptive field of 3, i.e. its kernel size. The following max-pooling layer has a pooling size of 2×2, meaning its output is affected by 2×2 outputs in the previous layer which translates to 4×4 in the input image. One can calculate the receptive field F_i of a node in layer i using the formula[8]

$$F_i = F_{i-1} + (k_i - 1) \prod_{j=1}^{i-1} s_j \tag{2.6}$$

where k_i is the kernel size of the nodes in layer i and s_j the stride in previous layers, $F_0 = 1$ is the receptive field at the input. Using this formula one finds that the output node of the network in table 2.1 has a receptive field of 32×32, i.e. its output is affected by the whole image.

2.3 Backpropagation

The backpropagation algorithm can be used to train neural networks and in short consists of passing input x through the network to get predictions $y' = f(x)$, comparing the predictions with the desired outcome y using a loss function $L(y, y')$, and then propagating the loss backward through the net[12].

For a particular neuron j in layer l of the network, its influence on the loss can be defined as

$$\delta_j^l = \frac{\partial L}{\partial z_j^l} = \sum_k \frac{\partial L}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{2.7}$$

where z_j^l is the weighted sum of the inputs of that neuron, i.e. the output *before* passing through an activation function, and the sum is over the k nodes in layer $l+1$. Using the chain rule one can then find the contribution of each weight as

$$\frac{\partial L}{\partial w_{jk}^l} = \sigma'(z_k^{l-1})\delta_j^k \tag{2.8}$$

where w_{jk}^l is the weight of node j in layer l for the input from node k in layer $l-1$, and σ' is the derivative of σ.

The network can then be trained by e.g. stochastic gradient descent[12]. The network is trained in steps and for every step a new sample is fed forward through the network and the weights impact on the loss is calculated. The weights are then updated according to

$$\Delta w_{jk}^l \leftarrow \eta \frac{\partial L}{\partial w_{jk}^l} \tag{2.9}$$

$$w_{jk}^l \leftarrow w_{jk}^l - \Delta w_{jk}^l \tag{2.10}$$

where η is the learning rate defining in how large steps the weights should be updated. When all samples have been fed through the network one epoch is finished. Training usually consists of many epochs.

Stochastic gradient descent is often done in so-called mini-batches, where a couple of samples are fed forward through the network and the weights are updated with the average of each samples resulting gradient, leading to smoother weight updates[12].

To further smoothen the weight updates a *momentum*[12] can be added to stochastic gradient descent which changes equation 2.9 to

$$\Delta w_{jk}^l \leftarrow \eta \frac{\partial L}{\partial w_{jk}^l} + \alpha \Delta w_{jk}^l \tag{2.11}$$

where $\alpha \in (0, 1)$ is the momentum parameter.

2.4 Loss function

In multi-class classification the true data distribution is often assumed to be a categorical distribution (multinoulli distribution). Finding the model that maximizes the likelihood of the data distribution is then equivalent to minimizing the negative log-likelihood[12]. The corresponding loss function, often called log-loss or categorical cross-entropy, is defined in [12] as

$$L_{CE}(\mathbf{y}, \mathbf{y}') = -\sum_i y_i \log y_i' \tag{2.12}$$

where $y_i \in \{0, 1\}$ is the true label and $y_i' \in (0, 1)$ is the predicted class probability, e.g. from a softmax layer. See e.g. [12] for more information.

2.5 Regularization

CNNs usually consist of up to millions of parameters leading to complex models with high representational power. If the data shown during training is unrepresentative of the true underlying structure, e.g. contains a lot of noise or only parts of the true distribution, there is a high risk of the model to *overfit* the training data and not generalize well to unseen data.

Regularization is a set of techniques to prevent overfitting by either enforcing constraints on the parameters in the network, e.g. weight normalization, or by using various techniques like *dropout*[17], *batch normalization*[20], or *data augmentation*.

2.5.1 Kernel weight normalization

In weight normalization[12], one tries to constrain the norm of the weights to small numbers by adding a regularization loss to the loss function. Having a small norm of the weights can help with training since stochastic gradient descent updates the weights by taking small steps. Common choices are L1- and L2-regularization, corresponding to using the L1- and L2-norm respectively. With L2-regularization the regularized loss function L_{reg} becomes

$$L_{reg} = L_0 + \frac{\lambda}{2} \sum_k w_k^2 \tag{2.13}$$

where L_0 is the unregularized loss function, λ a regularization parameter determining the amount of regularization, and w_k the weights of the model, excluding the biases.

2.5.2 Dropout

Dropout[17] on the other hand works by disabling the output of randomly chosen nodes during training. For every training step a different set of nodes are randomly disabled, meaning that their output is disregarded and their weights are not updated. This makes sure that the network does not rely on the output of single nodes.

With dropout the model behaves as an ensemble of several slightly different models but with shared weights. During inference (prediction) dropout is disabled and all nodes are active and the resulting output can thus be seen as a weight-averaging of the ensemble of models learned through dropout[22]. An example of one instance of dropout is shown in figure 2.3.

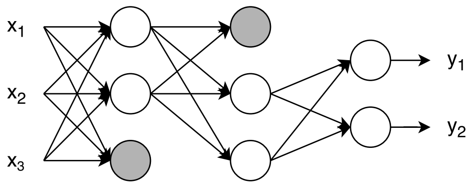


Figure 2.3: *Multi-layer perceptron with dropout. The dropped out nodes are colored gray and are kept inactive during this training step, meaning that their weights are not updated and their output is disregarded. A different set of nodes are disabled in the next training step.*

2.5.3 Batch normalization

Batch normalization effectively tries to normalize the activations of the nodes in the network to be normal distributed with zero mean and standard deviation 1. During training it calculates statistics on mini-batches used for normalization. These statistics are then used during inference and results show that it can speed up training by allowing higher learning rates and sometimes reduce the need for dropout as regularizer[20].

2.5.4 Data augmentation

Data augmentation is a method to synthetically enlarge the available data (see e.g. [23]). By adding random rotations, translations, flips, scaling and other transformations that preserve the label, one can effectively increase the number of training samples to better catch the underlying data distribution and get a model that generalizes better[12].

2.6 Data set split

To give an idea of how a network performs on real world applications on unseen data, a common approach is to split the available data set into three parts: training, validation, and test data.

The model's weights are trained on the training data while the validation data is used for tuning hyper-parameters such as learning rate, model architecture, regularization parameter etc. Validation data also gives a hint on how well the model generalizes. If the loss on validation and training data are close it generalizes well but if validation loss is significantly higher than training loss the model is overfitting the training data.

The test data is withheld during training and only used for evaluation of the network.

3 Method

This chapter describes the method used to answer the questions formulated in section 1.3. It contains a description of the data set and how it is split for training, validation and evaluation. Implementation details including network architecture and training parameters are also found in this chapter together with details on evaluation and metrics used.

3.1 Implementation

The CNN was implemented in Keras[5] using Tensorflow[1] as backend.

3.1.1 Architecture

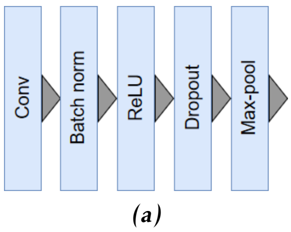
The relatively small variations and the few classes present in the data suggest that the model need not be very complex. We settled on using a network based on SegNet-basic[2] due to its simple and easily modifiable encoder-decoder structure. Our task is substantially less complex than SegNet-basics original problem domain and therefore its less than state-of-the-art performance is deemed to be more than sufficient.

SegNet-basic is an encoder-decoder with 4 blocks in the encoder and decoder respectively. It uses 7×7 convolutional layer, 2×2 max-pooling, and RELU in the encoder blocks, and 2×2 upsampling and 7×7 convolutional layer in the decoder. SegNet-basic uses a fixed feature depth of 64 throughout its layers.

The proposed architecture is similar in structure but with some ideas borrowed from its VGG16-based sibling SegNet[3]. More specifically the fixed feature depth is kept while RELU is added to the decoder blocks. Furthermore batch normalization layers are added between the convolutional and activation layers in a similar fashion to SegNet. The batch normalization layers are essential to making the network perform well on validation data, especially when training on synthetic data.

The structure of the encoder blocks are thus convolutional layer, batch normalization, RELU, dropout, and max-pooling, visualized in figure a. The structure of the decoder blocks become upsampling, convolutional layer, batch normalization, RELU, and dropout, which is visualized in figure b.

The last decoder-block is followed by a 1×1 convolution layer with softmax activation function to produce class probabilities.



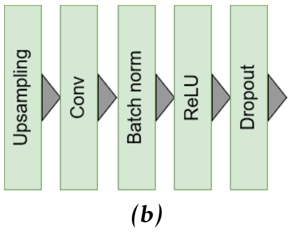


Figure 3.1: *Layer structure of block used in encoder (a) and decoder (b). (a): Encoder block consisting of convolutional layer, batch normalization, RELU, dropout, and max-pooling. (b): Decoder block consisting of upsampling, convolutional layer, batch normalization, RELU, and dropout.*

The architecture can be modified by varying the number of blocks n, the kernel size k, and the feature depth d. The number of blocks and kernel size can be directly linked to the model's receptive field, while the the number of blocks together and the feature depth control the representational power of the model.

Due to the limited time span of this thesis the model architecture space is limited to a grid search over

- $k = 3, 5, 7$
- $d = 16, 32, 64$
- $n = 2, 3, 4$

where the feature depth is fixed for all layers. The fixed feature depth may seem questionable as most recent approaches use increasing feature depth for deeper layers. Early investigation during this thesis did however not reveal any advantage of using increasing feature depth and due to the limited time span of the thesis this was not investigated further.

We found that the network with 4 blocks of 7×7 kernels with feature depth 64 performed the best closely followed by the network with 4 blocks of 7×7 kernels with feature depth 32. We settled on the latter configuration since the drop in performance is negligible but results in less wall clock time needed for training. Choosing 32 instead of 64 as feature depth also reduces the number of parameters by a factor 4 resulting in a total of approximately 350 000 parameters. The full architecture of the network is visualized in figure 3.2.

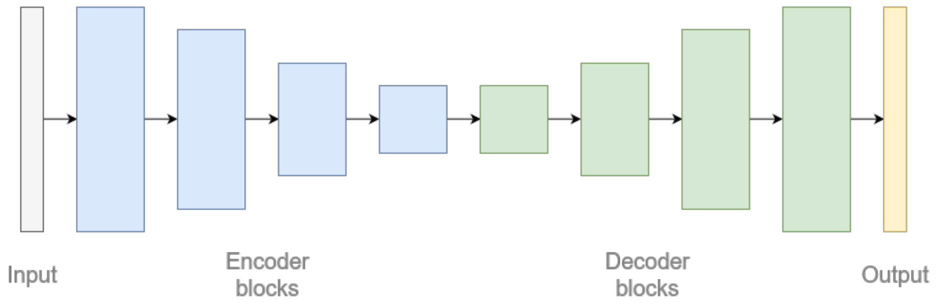


Figure 3.2: *Final network architecture consisting of 4 encoder blocks and 4 decoder blocks followed by a final* 1×1 *convolutional layer with softmax activation function.*

Using equation 2.6, one can find that the output layer of the network using 4 encoder and decoder blocks with 7×7 kernel size has a receptive field of 196×196 in the input image, meaning that the central output is affected by roughly half of the input (the input size is around 128×352). This can be put into perspective that a network with 3 blocks and the same kernel size only has a receptive field of 92×92 and removing one more block reduces that to 40×40.

To be able to use the model for classification, a few more processing steps are needed. First a post-processing step that gets rid of minor misclassifications in the segmentation map. After the post-processing the protrusions are estimated from the bounding boxes of the straw and box. The protrusion estimates are then used for classification and the box is either accepted or rejected. The data flow is visualized in figure 3.3

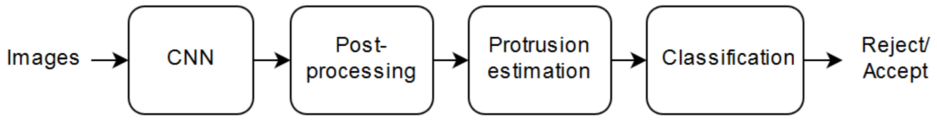


Figure 3.3: *Data flow from image through CNN, post-processing, protrusion estimation, and classification.*

3.1.2 Data augmentation

Data augmentation was implemented online to run in parallel with training, by constantly reading data from disk and generating different transforms to try and capture the true distribution of the data and to gain more robustness to unseen data. The same random seed is used for all training sessions so that the model performance is independent of randomness between training sessions.

The augmentation scheme used is inspired by [23] with random crops and horizontal flips of the data. We added random vertical flips and random planar shifts in XY-plane to simulate tilted packages.

The random crops of the data adds variations in data layout, i.e. location of box and straw in the image, which does not vary much within a data set from the same site but more so in-between sets from different sites. The random crops do however limit the receptive field of the nodes in the final layer, but by also randomizing the crop sizes the model sees some examples at full scale.

The horizontal and vertical flips together with the natural variation in rotation of the straw makes the model more resilient to various orientations of the straw. This also helps with robustness against mirrored images which can be found in the data set.

The random planar shifts consist of adding a tilted plane to the data and helps with making the model resistant to tilting of the box.

Finally random scaling and shifting of the data is added, which effectively models variations in height of the box.

Mean centering and normalization is avoided due to the large amount of missing data in the images and is also seemingly unnecessary due to batch normalization layers in the network.

3.1.3 Training

We train the network for 50 epochs using stochastic gradient descent with learning rate 0.01 and momentum 0.5. We use a mini-batch size of 4 to avoid memory-related issues while smaller mini-batches lead to longer training times without increased performance.

The learning rate and momentum seemed to work well in general. Lower learning rate and momentum slowed down learning and higher learning rate and momentum did not increase the training speed significantly.

Dropout rate was set to 10% which worked well with the small number of filters. A higher dropout rate did slow down learning and decrease performance.

The weights were initialized using Xavier initialization[11], i.e. a uniform distribution scaled with the number of nodes in the previous and following layer.

The training samples were augmented online using the data augmentation scheme described in section 3.1.2, with random crops of random size between 96×96 to 128×320 to allow variation in the layout of the training data. The random crop sizes are kept evenly divisible by 16 to be compatible with up to four 2×2 max-pooling layers.

The validation data is cropped to largest possible size divisible by 16 without any data augmentation to avoid variations in validation accuracy to depend on random factors in the data augmentation scheme.

To address the large variation in class occurrences we add class weighting in the loss function which changes equation 2.12 to

$$L_{CE,weighted}(y, y') = -\sum_i y_i w_i log(y'_i) \tag{3.1}$$

where w_i is the class weight for class i.

We calculate the class weights as

$$w_i = -log(p_i) \tag{3.2}$$

where p_i is the estimated probability of finding class i, i.e. the ratio of pixels belonging to the class to total number of pixels. The resulting class weights are found in table 3.1. Without class weights the models disregard the straw class and have a tendency to predict the box class where there is a straw.

Table 3.1: *Class weights for the classes background, box, and straw used during training of the network.*

Class	**weight**
Background	0.68
Box	0.80
Straw	3.06

3.1.4 Depth-channel encoding

Compared to RGB-images, depth gives the opportunity to encode local properties in the image like e.g. surface normals and height above ground in addition to depth. [14] uses an encoding were they add information about height above ground and angle with the gravity vector. For transfer learning [9] uses simple depth-to-color encoding with good results and [13] uses a cross modal transfer technique to pre-train for a different modality. These encodings add some computational overhead and since our data set is limited to depth and quite different from available data sets that can be used for pre-training, it is unlikely that transfer learning and encoding will increase accuracy for our specific task. Therefor we will rely on the model to learn its own representation.

3.2 Data set analysis

The real data set consists of depth images of juice boxes with straws glued onto them. The images are collected at various sites with a range camera, which uses laser triangulation to measure depth. Laser triangulation gives an accurate measure of the depth but also results in areas of the image lacking information due to occlusion, reflections, and transparency of some materials. An example image can be seen in figure 3.4.

3.2.1 Real world data sets and train-val-test split

The real data set has 34 000 pairs of 16-bit unscaled depth images and corresponding color images with graphic overlay containing the results from the baseline solution (see figure 1.1. The images also contain additional information as

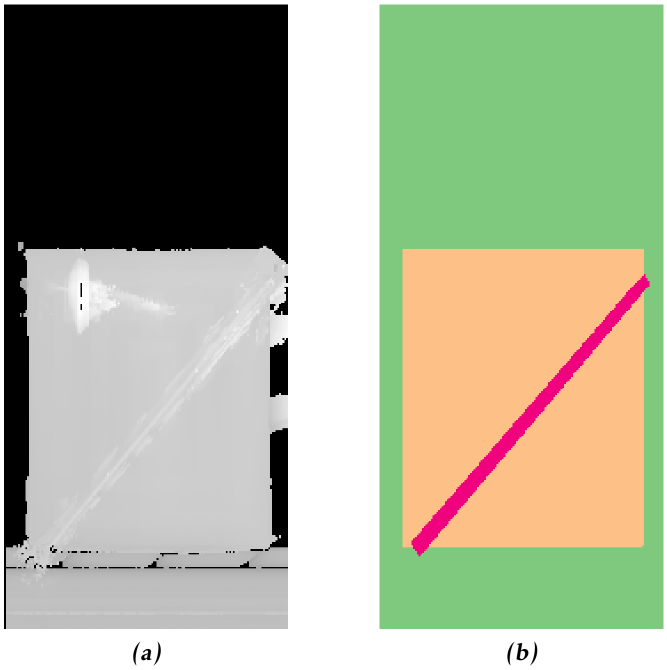


Figure 3.4: *Example test image (a) and corresponding ground truth label map (b). (a): brighter is higher (closer to the camera), black is missing data. (b): the colors fern, peach, and rose correspond to the classes background, box, and straw respectively in the label map.*

scaling factor and sampling distance and are rescaled to height in millimetres prior to training and evaluation.

Around 4 000 images have ground truth measurements of straw protrusion in millimetres and are reserved for testing. These 4 000 images are also hand-labelled using the image annotation tool Ratsnake[19] to get ground truth label maps for evaluation of semantic segmentation. An example image is shown in figure 3.4. Out of these 4 000 images, roughly 1 000 do not have any straw present.

The remaining 30 000 images are collected on various sites with variation in type of boxes, straws and setups used. 4 000 of these are exclusively used for validation and the last 26 000 are for training. The training and validation data are collected at different locations. The number of images in each set of the real data can be seen in table 3.2.

Since no ground truth label maps exist for training and validation data, the labels for training and validation are inferred from the baseline solution which can produce its results as images with overlay graphic. This gives easy access to training data at the cost of introducing some errors in the labels.

Most of the images contain large chunks of missing data, where on average 30% of the pixel values are missing, with box and conveyor pixels making up a

Table 3.2: *Data set split for real data.*

Set	Samples
Train	26 000
Validation	4 000
Test	4 000

majority of the pixels with data and only a small fraction being pixels belonging to the straw class.

The boxes are of different shapes, and the straws are of different shapes and colors, including white, black, and transparent. The transparent straws are where the baseline solution has most trouble locating the straw.

3.2.2 Automatic labelling

The 30 000 real world images reserved for training and validation without ground truth labels are labelled using the graphic overlay from the baseline solution. The results from the baseline solution are the location of the box in the image, and the straw as line segments. By filling in the box and using binary dilation on the straw we get a coarse label map. An example image with overlay graphic and corresponding extracted label map is shown in figure 3.5.

Since labels are inferred from the baseline solution this introduces some errors in the label maps used for training, limiting how accurate we can expect the CNN to get from training on the training data set.

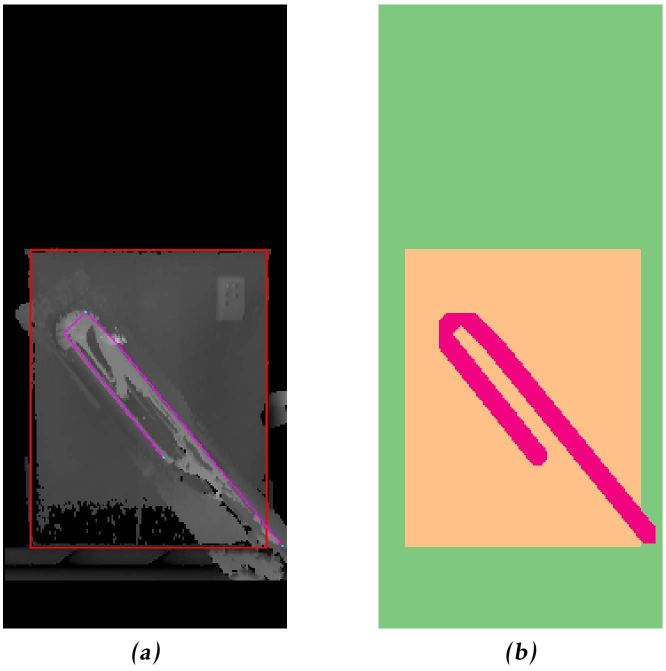
3.2.3 Synthetic data generation

The generation of synthetic data is basic with a rectangle of even height representing the box, and a straw represented as 2D height projections of cylinders. Statistics from real world data is used to estimate the distribution of box and straw dimensions, and straw type. Randomly positioning the box and the straw in the image gives a crude approximation of the true data, and by adding the proposed data augmentation scheme as for regular training data we also get some variation in height and tilt.

Although synthetic data generation can be done almost for free during training, we generate 32 000 synthetic images beforehand so that the same images are used for various experimental setup.

3.3 Evaluation and metrics

By showing the network previously unseen images we get an idea of how well the model generalizes in real world applications. The evaluation is done using the test set of 4 000 hand-labelled images described in section 3.2.1, and is performed using multiple models trained on different compositions of real and synthetic



(a) *(b)*

Figure 3.5: *Example image with baseline result overlay (a) and corresponding extracted label map (b). (a): red marks the bounding box for the box class and magenta the parametrized straw. (b): the colors fern, peach, and rose correspond to the classes background, box, and straw respectively in the label map.*

data. Pixels that have missing data are included in the statistics since often large parts of the straw and box are made up of missing data.

3.3.1 Classification

Important metrics for classification besides accuracy are precision and recall. In this context precision measures how many of the rejected samples are correct and recall measures how many of the faulty samples that are rejected. Precision is defined as

$$precision = \frac{T_p}{T_p + F_p} \tag{3.3}$$

and recall as

$$recall = \frac{T_p}{T_p + F_n} \tag{3.4}$$

where T_p are true positives (true rejects), F_p false positives (false rejects), and F_n false negatives (false accepts). A sample is defined as faulty and rejected if the straw is protruding more than 3.5 mm on any side of the box or if the straw is missing.

The protrusions are found by calculating the distance between the edges of the bounding boxes for the box and straw class found in the predicted segmentation. Scaling with the spatial sampling distance ($\sim$ 0.5mm for most images) gives a measurement is millimetres.

Morphological erosion and dilation is used to get rid of small misclassified blobs that have little effect on the metrics for semantic segmentation but larger on classification. This is achieved by removing any blob but the biggest belonging to either box or straw class, followed by erosion and dilation of the individual class label maps twice respectively, and finally removing any blob smaller than 25% of the largest blob. The post-processing is only done for evaluation of the classification and not for semantic segmentation.

3.3.2 Semantic segmentation

Comparisons between predicted results and ground truth label maps show how accurate the model is and for semantic segmentation common metrics are global pixel accuracy, per-class accuracy, and mean Intersection over Union (mean IoU). To calculate these scores the predicted results are converted to a one-hot encoding, i.e. a pixel gets assigned to the class with highest probability.

Pixel accuracy is the ratio of correctly classified pixels to total number of pixels which gives a coarse estimate of how well the model performs but can be misleading when classes are unbalanced. Per-class accuracy instead measures the pixel accuracy for each individual class. We calculate the pixel accuracy on a per-image basis and ignore the samples where a class is not present for the per-class accuracy.

IoU gives an indication of how well the predicted labels overlap with the ground truth and is defined as

$$IoU(\mathbf{y}, \mathbf{y}') = \frac{\mathbf{y} \cap \mathbf{y}'}{\mathbf{y} \cup \mathbf{y}'} \tag{3.5}$$

where a prediction having wrong size or false location lead to worse scores and a perfect overlap equates to 1. We calculate class-wise IoU on a per-image basis and set the score to 1 when a class is not present in either prediction or ground truth, and 0 when present in only prediction or ground truth.

Comparing the mean absolute error (MAE) of the predicted straw protrusions can give an indication of how well the different approaches are at locating the straw and the box. MAE is defined as

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - y_i'| \tag{3.6}$$

where y_i and y_i' are the ground truth and predicted protrusions respectively.

3.3.3 Qualitative evaluation

Qualitative evaluation is performed by inspecting predicted label maps to see how "good" they look. Due to presence of errors in the automatically labelled training data, the quantitative results cannot be more accurate than the labels themselves, meaning that the model may be making the right predictions even if the quantitative error is high. By inspecting the images where the model has the poorest performance one can get an idea of whether the error is model or data dependent.

3.3.4 Experiments

The evaluation will be performed using multiple models trained on different training data to be able to answer the questions in section 1.3 regarding how training on synthetic data compares to training on real world data. For all experimental setups the validation and evaluation is done on the validation and test data set respectively. The validation and test data sets are described in section 3.2.1.

The first setup, **AR** (All Real training data), consists of training on only real world data and will be used as a reference for comparisons with the other setups, in total 26 000 images are used for training.

The second setup, **AS** (All Synthetic training data), consists of training on synthetically generated data only. The training data is generated algorithmically as in section 3.2.3 and 32 000 images are generated beforehand.

The third setup, **ASAR** (All Synthetic, All Real), consists of combining all synthetic data with all real world data to see if adding synthetic data can increase accuracy and generalization, in total 58 000 images.

The fourth setup, **ASPR** (All synthetic, Part Real), consists of combining all the synthetic data and a subset of the real data to see if adding a small amount of real data to the synthetic data can compare to using more real data. This results in a total of 34 600 images.

The fifth and final setup, **PR** (Part Real), consists of training on the same small set of real data as in **ASPR** to see if adding synthetic data actually increases performance compared to only using a small subset of the real data. Every 10th sample of the **AR** set is used for the **PR** setup, i.e. 2 600 images.

4 Results

The results on test data for the different experiments are presented in this chapter. The first part contains the results on validation data that were used to design the model and choose training parameters. The second part contains the results from evaluation using the models from the different training setups.

4.1 Implementation

Neural networks offer an abundance of ways to be configured. The number of layers, their kernel size, and the number of filters per layer all affect the performance of the network. Changing the learning rate, using momentum, and regularization can impact how well the trained network generalizes.

4.1.1 Architecture

The network architecture is customizable where the number of encoder-decoder blocks, the kernel size, and the number of layers can be changed. We use a fixed feature depth similar to SegNet-basic as no major advantage was found in using increasing feature depth in initial studies and time for the thesis was limited. The various architectures where validated on the validation data set described in section 3.2.1.

With a fixed feature depth of 64 and a kernel size of 7×7, varying the number of blocks between 2, 3, and 4 in the encoder and decoder gives the loss on the validation data seen in figure 4.1.

By instead keeping a fixed number of blocks and kernel size, while varying the number of filters between 16, 32, and 64 the loss changes as in figure 4.2.

Finally by varying the kernel size used, while keeping the number of filters and blocks fixed, we get the validation loss seen in figure 4.3.

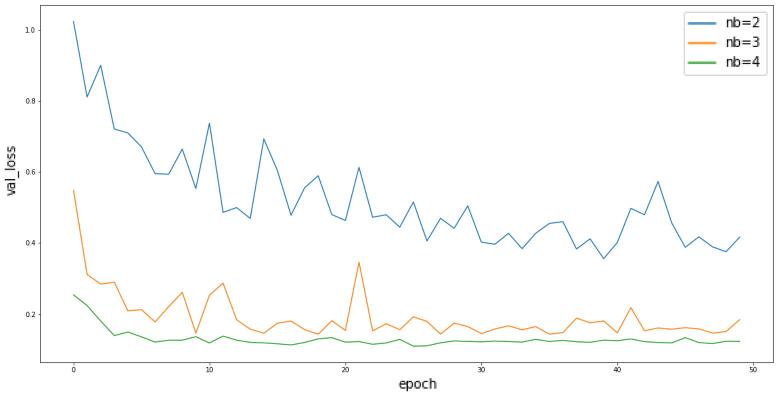


Figure 4.1: *Loss on validation data when varying the number of encoder-decoder blocks between 2, 3, 4. The kernel size and feature depth are kept constant at 7 and 64 respectively.*

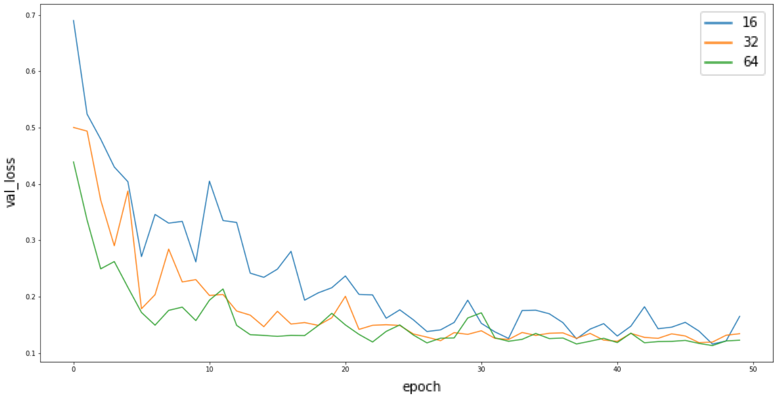


Figure 4.2: *Loss on validation data when varying the feature depth between 16, 32, 64 using a constant kernel size of 5 and 4 encoder-decoder blocks.*

As seen in figure 4.2 the models with fixed feature depth of 32 and 64 achieve very similar loss. The former is chosen since it reduces the wall clock time for training of the experimental setup models without any major drop in performance. By using a feature depth of 32 with 7×7 kernels and 4 encoder and decoder blocks we end up with the network architecture found in table 4.1. This model architecture is then trained using the different experimental setups mentioned in section 3.3.4 and used for testing.

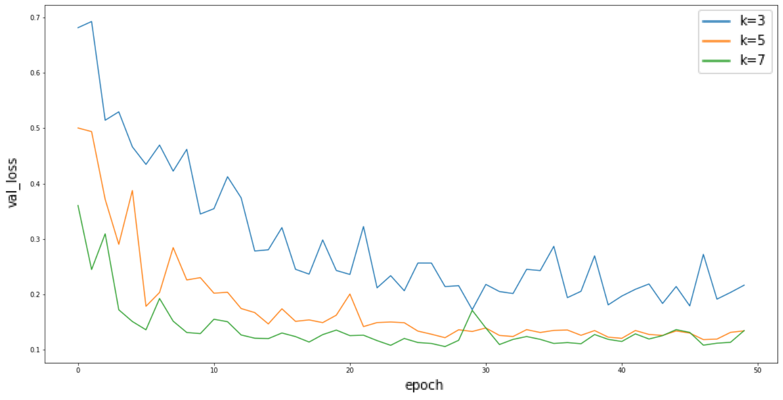


Figure 4.3: *Loss on validation data when varying the kernel size between 3, 5, 7 using a constant feature depth of 64 and 4 encoder-decoder blocks.*

Table 4.1: *Final network architecture. The layers* conv *include the convolutional layer, batch normalization,* RELU*, and dropout.*

Layer	**kernel size**	**filters**	**stride**	**receptive field**	**parameters**
conv1	7×7	32	1	7×7	1 664
max-pool1	2×2	-	2	8×8	-
conv2	7×7	32	1	20×20	50 272
max-pool2	2×2	-	2	22×22	-
conv3	7×7	32	1	46×46	50 272
max-pool3	2×2	-	2	50×50	-
conv4	7×7	32	1	98×98	50 272
max-pool4	2×2	-	2	106×106	-
upsamp1	1×1	-	1/2	106×106	-
conv5	7×7	32	1	154×154	50 272
upsamp2	1×1	-	1/2	154×154	-
conv6	7×7	32	1	178×178	50 272
upsamp3	1×1	-	1/2	178×178	-
conv7	7×7	32	1	190×190	50 272
upsamp4	1×1	-	1/2	190×190	-
conv8	7×7	32	1	196×196	50 272
fc9	1×1	3	1	196×196	99
softmax	1×1	-	1	196×196	-

4.1.2 Validation results

The results on the training and validation data of the different experimental setups are presented in table 4.2. This gives a hint on how well the various models generalize or how much they overfit the training data.

Table 4.2: *Results on train and validation (Val) data for the different experiment setups. Accuracy is the global pixel-wise accuracy and mIoU is the mean intersection over union.*

Experiment	Train accuracy	Train mIoU	Val accuracy	Val mIoU
AR	0.941	0.810	**0.948**	**0.802**
AS	**0.974**	**0.912**	0.928	0.729
ASAR	0.957	0.859	**0.948**	0.801
ASPR	0.970	0.899	0.944	0.783
PR	0.931	0.784	0.944	0.792

4.2 Evaluation

The evaluation is split into two parts, one covering the results of semantic segmentation, and the other the results of classification. The classification part also contains the results of the baseline solution.

4.2.1 Semantic segmentation

The pixel-wise accuracy achieved from the different experiments is shown in table 4.3 were the class-wise accuracy is the amount of correctly classified pixels belonging to that class and the mean accuracy is the mean of the class-wise accuracies. We see that the model **PR** achieves the highest accuracy scores on all but the straw class, followed by **AR** and **ASAR**.

Table 4.3: *Pixel-wise accuracy on* ***test data*** *for the different experiment setups over the individual classes straw, box, and BG (background). 'Mean' is the average of the class accuracies and 'Global' is the global pixel-wise accuracy.*

Experiment	BG	Box	Straw	Mean	Global
AR	0.971	0.891	0.695	0.852	0.925
AS	0.887	0.774	0.750	0.803	0.837
ASAR	0.973	0.874	0.756	0.868	0.920
ASPR	0.921	0.807	**0.795**	0.841	0.865
PR	**0.975**	**0.897**	0.735	**0.869**	**0.930**

More descriptive of the actual localization performance of the models is the results on intersection over union presented in table 4.4. Similar to pixel-wise accuracy the model PR gets the highest scores closely followed by AR and ASAR.

Table 4.4: *Intersection over Union on **test data** for the different experiment setups over the classes BG (background), box, and straw. mIoU is the mean of the class-wise IoU.*

Experiment	BG	Box	Straw	mIoU
AR	0.91	0.85	0.54	0.77
AS	0.80	0.71	0.23	0.58
ASAR	0.91	0.84	0.53	0.76
ASPR	0.84	0.74	0.46	0.68
PR	**0.92**	**0.86**	**0.60**	**0.80**

4.2.2 Classification

The scores for the classification metrics are shown in table 4.5, where **BASELINE** refers to the results achieved using the existing baseline algorithm. A sample is classified as faulty if the straw is missing or protruding more than 3.5 mm from any of the edges of the box.

Table 4.5: *Scores for classification using the different models. **BASELINE** is the result from the already existing solution. MAE is the mean absolute error of the protrusion estimates in millimetres.*

Experiment	Accuracy	Precision	Recall	MAE [mm]
BASELINE	**0.874**	**0.901**	0.796	**2.83**
AR	0.816	0.894	0.637	5.45
AS	0.588	0.507	0.883	8.16
ASAR	0.580	0.502	0.878	5.96
ASPR	0.524	0.469	**0.949**	7.22
PR	0.815	0.861	0.670	5.26

In table 4.5 we see that the baseline algorithm achieves the highest scores on all classification metrics but recall where the models with synthetic data (**AS**, **ASAR**, **ASPR**) perform better but instead has significantly worse scores on accuracy and precision. The models trained on real data (**AR**, **PR**) are fairly close in performance on accuracy and precision but has almost twice the *MAE* of the baseline and significantly lower recall score.

4.3 Qualitative results

To get a sense of why the models perform the way they do some example predictions from the different models are shown here. By showing the best and worst examples for each model one can get a sense on what data they struggle with and a hint if the error is model or data dependent.

The models consistently perform worse when there is a lot of missing data on the box and straw as seen in figure 4.4.

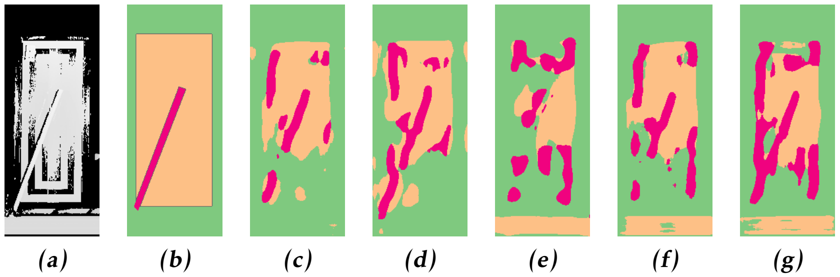


Figure 4.4: Example test image with much missing data where all models get low scores on semantic segmentation. (a) Test image; (b) ground truth label map; (c) AR, 0.46 $mIoU$; (d) PR, 0.46 $mIoU$; (e) AS, 0.32 $mIoU$; (f) ASAR, 0.43 $mIoU$; (g) ASPR, 0.35 $mIoU$.

On the other end of the spectrum the models perform well on images where there is not much missing data on the box and straw classes, which is more similar to the data found in the train and validation data set. One example is show in figure 4.5.

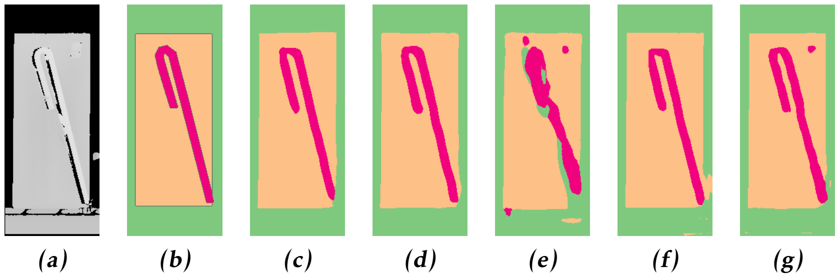
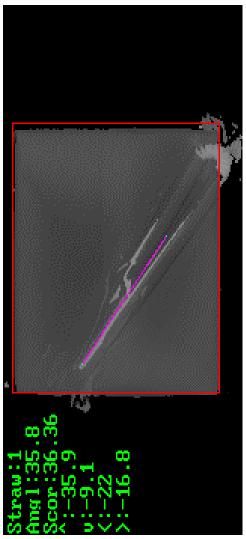
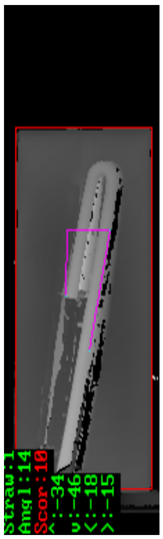


Figure 4.5: Example test image with little missing data where all models get high scores on semantic segmentation. (a) Test image; (b) ground truth label map; (c) AR, 0.83 $mIoU$; (d) PR, 0.87 $mIoU$; (e) AS, 0.78 $mIoU$; (f) ASAR, 0.87 $mIoU$; (g) ASPR, 0.86 $mIoU$.

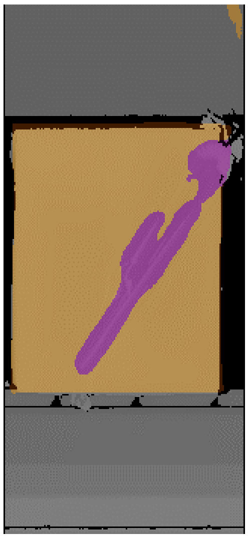
A qualitative comparison between the baseline and the AR-model is shown in figure 4.6. Both these samples are from the training set and this shows that the model has managed to learn some representation of what a straw is and not just memorized the training samples.



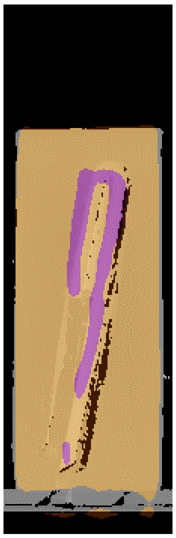
(a)



(b)



(c)



(d)

Figure 4.6: *Qualitative comparison between baseline and* ***AR*** *model. (a), (b): Two examples where the baseline solution struggles to find the true shape of the straw (magenta line). (c), (d): corresponding segmentation result overlaid on a grayscale image.*

5 Discussion

The results and chosen method are discussed in this chapter.

5.1 Results

Assessing the results we see that all models perform fairly well on the validation data set as can be seen in table 4.2. There is little to no overfit on the training data for the models trained on real world data only (**AR**, **PR**) but as expected more so on the models trained with synthetic data. Interestingly the models get almost identical scores on pixel-wise accuracy and mean intersection over union on validation data. All but **AS** reach accuracy of 94-95% and mIoU of 0.78-0.8, which is fairly good given how unforgiving intersection over union is on misclassifications.

The fact that the models trained on synthetic data achieved higher accuracy on training data than the models trained on real data may suggest that the synthetic data is more homogeneous than real data and therefore easier to fit. On the other hand, the fact that the same models still achieve good results on validation data might be an indication that there are some errors in the labelled data, as is the case in figure 4.6.

When evaluating on the test data we clearly see a tendency that the higher the ratio of real to synthetic images, the higher the scores are. **AR**, **PR**, and **ASAR** get test scores (table 4.4, 4.3) only slightly lower than validation scores (table 4.2) where interestingly the model **PR** gets better scores than the other two on test data, but worse on validation data. They all get mIoU scores of > 0.75 which is fairly good. Their low IoU scores on the straw class is however alarming since accurate localization of the straw is critical for the task of classification.

Comparing with the baseline solution clearly shows that the baseline is better suited for straight up classification as seen in table 4.5. The models **AR** and

PR are not too far behind on classification scores, both reaching accuracy of > 81% and gets precision scores only marginally lower than baseline, but *MAE* almost twice as large as the baseline result. The models **AS**, **ASAR**, **ASPR** trained on synthetic data excel on recall but on the other hand they all get precision of around 0.5 meaning that roughly half of the rejected boxes are false rejects. Worth noting is that **AS**, the model trained only on synthetic images, gets higher classification accuracy than the other two models trained on synthetic images, even though it has the worst *MAE* of all models.

As can be seen in figures 4.4–4.5 the models perform best when the data is smooth with little missing data – similar to validation and train data – and worse when patterns of missing data appear in the image as in many of the test images.

5.2 Method

Designing a CNN has endless possibilities in architecture and hyperparameters. The chosen architecture is based on SegNet-basic which is fairly basic with only convolutional, max-pooling, and upsampling layers. It is possible that more complex architectures based on ResNet or VGG16 would perform better than the chosen one, though initial studies did not show any benefit. Trying out all possible architectures is however not feasible and SegNet-basic was chosen as a base network because of its simplicity. It is also possible that errors in the auto-generated labels leads to a label error of around 5%, meaning that more powerful networks could lead to overfitting on the label errors.

The chosen network did perform well on semantic segmentation which is its intended design. The classification problem is however very sensitive to even minor errors in segmentation. A 2 pixel error may be neglible in segmentation but it corresponds to 1mm protrusion error which may be the difference between a reject and an accept.

The chosen architecture relies heavily on downsampling (16×) which may have a negative impact on the resolution of the results and it is possible that replacing the max-pooling and upsampling layers with e.g. dilated convolutions could improve resolution while retaining the receptive field.

The implemented post-processing scheme was kept simple due to time constraints and if more time was at hand, more advanced techniques could have been tested. Conditional random fields (CRF) come to mind here which could also possibly be used to improve the automatic labels for training.

The same goes for the protrusion estimates which were found using bounding boxes on the post-processed segmentation maps. Applying some form of regression could likely increase protrusion accuracy and in turn classification scores.

The synthetic data generation is very crude at best and lacks many of the details found in real world data. GANs have proven successful in other applications to improve synthetic data. They are however difficult to implement and require much time to fine-tune the parameters to converge.

Even though the synthetic data generation was simple, the models trained on synthetic data performed well on the validation set. The difference in performance on validation and test data is likely due to the domain shift between the sets, where e.g. some of the images have a very specific pattern of missing data found only in the test data set.

6 Conclusion

This chapter answers the questions from the problem formulation in section 1.3.

Can a CNN be used for semantic segmentation of depth images of juice boxes from a production environment? As the results show, a CNN can successfully be used for semantic segmentation of depth images. Even training on only synthetic data gives reasonable results, though the performance increases with the ratio of real to synthetic images. The difference in performance on test and validation data is probably mainly due to a domain shift between the sets as they are collected at different locations with different setups.

Can the resulting segmentation be used for binary classification of the juice boxes? Classification from segmentation maps is possible although sensitive to minor errors in the segmentation. More extensive post-processing and less naive protrusion estimation could possibly increase classification results.

How does the CNN compare to the current baseline solution on classification? The CNN does not perform as well as the baseline solution though comes close when trained on real images. One possible cause is that the baseline solution is fine-tuned towards the setup at each site, meaning that it's also tuned for the test data, whereas the CNN is not tuned towards data from a specific site, but instead all the data in the train and validation set. Moreover, the implemented CNN is unaware of prior knowledge about box and straw dimensions, shape and type and adding that information could also improve the segmentation results and in turn classification.

How does the CNN perform when trained on synthetic data? Training on synthetic data was not as successful due to lacking real world noise and artifacts. However, when labelled real world data is scarce it can possibly be used as a pre-training step to reduce the amount of hand-labelled data needed. If one manages to model the real world aspects in the synthetic data, e.g. with a GAN, it could possibly perform well even without fine-tuning.

6.1 Future work

For actual implementation use it would make sense to try out fine-tuning towards the target domain. In this case all future images will most likely come from the same setup. The networks still need a GPU to be able to run in real time but fine-tuning towards a single target domain could mean that shallower networks could perform just as well and the computational power needed could thus be limited. It would also be interesting to see how the network would behave if a priori knowledge of straw shape etc. was known to the network as that would be a more fair comparison with the baseline solution.

The synthetic data generation is something worth investigating further as it could lead to practically free training data. GANs and variational auto-encoders are two approaches that have shown capable in recent research and these could be used to better catch the subtle nuances found in the real data.

Finally it would be interesting to evaluate the use of a CRF for post-processing of the semantic segmentation and see how that affects the performance on classification. CRFs could possibly also improve the automatically extracted labels used for training.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Cited on page 15.

[2] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015. URL `http://arxiv.org/abs/1505.07293`. Cited on pages 4 and 15.

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. URL `http://arxiv.org/abs/1511.00561`. Cited on pages 4 and 15.

[4] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016. ISSN 0162-8828. doi: 10.1109/TPAMI.2017.2699184. URL `http://arxiv.org/abs/1606.00915`. Cited on page 5.

[5] François Chollet. keras. `https://github.com/fchollet/keras`, 2017. Cited on page 15.

[6] Camille Couprie, Clément Farabet, Laurent Najman, and Yann Lecun. Indoor semantic segmentation using depth information. In *International Conference on Learning Representations (ICLR2013), April 2013*, 2013. URL `http://arxiv.org/abs/1301.3572`. Cited on page 5.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. Cited on page 2.

[8] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, March 2016. Cited on pages 9 and 10.

[9] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin A. Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust rgb-d object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 681–687, 2015. URL `http://ais.informatik.uni-freiburg.de/publications/papers/eitel15iros.pdf`. Cited on pages 5 and 19.

[10] Carlos Gershenson. Artificial neural networks for beginners. *CoRR*, cs.NE/0308031, 2003. URL `http://arxiv.org/abs/cs.NE/0308031`. Cited on page 7.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010. Cited on page 18.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. Cited on pages 5, 7, 8, 9, 11, 12, and 13.

[13] S. Gupta, J. Hoffman, and J. Malik. Cross modal distillation for supervision transfer. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 2827–2836, June 2016. doi: 10.1109/CVPR.2016.309. URL `http://arxiv.org/abs/1507.00448`. Cited on page 19.

[14] Saurabh Gupta, Ross Girshick, Pablo Arbelaez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *Computer Vision – ECCV 2014*, pages 345–360. Springer International Publishing, 2014. ISBN 978-3-319-10584-0. URL `http://arxiv.org/abs/1407.5736`. Cited on pages 5 and 19.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 770–778, June 2016. doi: 10.1109/CVPR.2016.90. URL `doi.ieeecomputersociety.org/10.1109/CVPR.2016.90`. Cited on page 2.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV 2015)*, volume 1502, 02 2015. URL `http://arxiv.org/abs/1502.01852`. Cited on page 2.

[17] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL `http://arxiv.org/abs/1207.0580`. Cited on page 12.

[18] G. Huang, Z. Liu, L. v. Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 2261–2269, July 2017. doi: 10.1109/CVPR.2017.243. URL `doi.ieeecomputersociety.org/10.1109/CVPR.2017.243`. Cited on page 5.

[19] DK Iakovidis, T Goudas, C Smailis, and I Maglogiannis. Ratsnake: a versatile image annotation tool with application to computer-aided diagnosis. *The Scientific World Journal*, 2014, 2014. Cited on page 20.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/ioffe15.html`. Cited on pages 12 and 13.

[21] S. Jegou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, volume 00, pages 1175–1183, July 2017. doi: 10.1109/CVPRW.2017.156. URL `doi.ieeecomputersociety.org/10.1109/CVPRW.2017.156`. Cited on page 5.

[22] Alex Kendall, Vijay Badrinarayanan, , and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015. URL `http://arxiv.org/abs/1511.02680`. Cited on pages 5 and 13.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. Cited on pages 2, 4, 9, 13, and 17.

[24] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 3431–3440, June 2015. doi: 10.1109/CVPR.2015.7298965. URL `doi.ieeecomputersociety.org/10.1109/CVPR.2015.7298965`. Cited on page 5.

[25] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1278–1286, 2015. Cited on page 6.

[26] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv preprint arXiv:1511.06434*, nov 2015. URL `http://arxiv.org/abs/1511.06434`. Cited on page 5.

[27] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. Cited on page 7.

[28] Florian Schroff, Antonio Criminisi, and Andrew Zisserman. Object class segmentation using random forests. In *Proc. British Machine Vision Conference (BMVC)*, January 2008. URL `https://www.microsoft.com/en-us/research/publication/object-class-segmentation-using-random-forests/`. Cited on page 4.

[29] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1):2–23, 2009. ISSN 1573-1405. doi: 10.1007/s11263-007-0109-1. URL `http://dx.doi.org/10.1007/s11263-007-0109-1`. Cited on page 4.

[30] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 2242–2251, July 2017. doi: 10.1109/CVPR.2017.241. URL `doi.ieeecomputersociety.org/10.1109/CVPR.2017.241`. Cited on page 5.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`. Cited on pages 2 and 4.

[32] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015. URL `http://arxiv.org/abs/1511.07122`. Cited on page 5.