# SEGMENTATION OF IMAGES OF HANDS FOR IN-FIELD TECH SUPPORT

*Holger Krener-Iversen*

Technical University of Denmark
DTU Compute
s164143

## ABSTRACT

The use of machine learning for classification of images has been widely adopted within many different industries. However, an increasing number of projects and issues in many different industries require either object detection or semantic segmentation. 'Semantic' segmentation indicates a labelling of objects together with their outline. Semantic segmentation of images is, however, a problematic task, as it is often not an easily definable task. To get around this limitation, this paper proposes to train a particular model of a deep convolutional network to pick up on the features that make up a hand, to perform semantic segmentation of hands. The proposed model uses the U-net architecture for segmentation. The finished model was able to achieve an F1 score of 0.92.

***Index Terms—*** Semantic Segmentation, Deep Convolutional Neural Networks, U-net, Data-Augmentation, Encoder, Decoder, Pooling, Upsampling

## 1. INTRODUCTION

This work will showcase the use of the deep convolutional network architecture U-net on images of hands to provide a mask for the hand in picture. The paper will address the use of the proposed architecture, U-net, to achieve higher-level feature extractions, which is needed for a semantic image segmentation task, such as this one.
Section 2 will cover the motivation for this work, along with a brief introduction to the use-case that sparked it. Section 3 will cover the data and methods used to expand and modify it. Section 4 will explain the methods used for the project, and their following implementations. Section 5 will cover the obtained results, and sections 6 and 7 will discuss and conclude this work.

## 2. MOTIVATION

The company ShowMe Telepresence provides an app for in-field tech-service. Using the app, the user is able to call a technician from far away and receive virtual help through the phone. For this purpose, ShowMe has had a deep-learning based model for image segmentation developed, to key out the hand of the technician and/or user. This deep learning model was initially implemented in the architecture "MobileUnet" (`https://github.com/akirasosa/mobile-semantic-segmentation`), as it was imperative for the finished model to not only be able to run under the hardware constraints of an average smartphone (at the time of August 2018), but also be backwards compatible with older smartphones. "MobileUnet" is an architecture optimized for performance on smaller mobile phones. While the model is usable for ShowMe, the limited network size did come with an upper limit in performance.
This paper will look into if it is possible to develop a model that can achieve the same performance, or better, on a conventional, full-scale U-net implementation. This becomes increasingly relevant, as smartphones are becoming exponentially more powerful.

## 3. DATA

The data-set consists of 1144 pairs of pictures of size $540 \times 960$ of hands, and their corresponding masks. The data was originally gathered by a subject taking pictures of a subject's own hand using a smart-phone camera, mimicking the use-case for the end-product. The data was purposefully created in this way to achieve a higher level of authenticity. The masks for each image was created using ShowMe's in-house keying-algorithm.
Both images and masks contains the three colour channels, RGB. The mask, however, was for the sake of training the model, modified to only contain binary values (0 for background, 1 for hand). Furthermore, images and masks were both scaled down to $240 \times 135$ to speed up convergence, when training the model.

Thanks to ShowMe for providing the data-set.

## 4. METHODS

### 4.1. Data Preparation

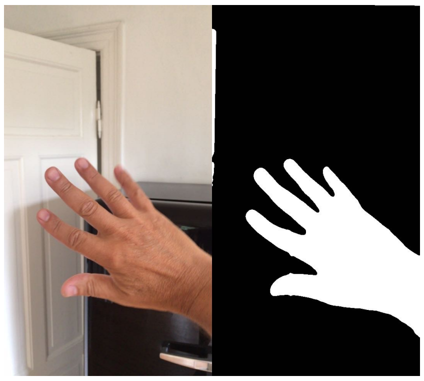
As mentioned, the data-set consists of 1144 pictures of concatenated images and masks, side by side. In the data-loading process, the pictures were thus first split up into their respective parts, and then down-sampled to the smaller resolution of $240 \times 135$. The masks were converted from classic RGB signals to binary signals by thresholding the pixelvalues to determine whether a given pixel is hand or background. Furthermore, the input images were normalized before training, so the pixel values were squished into the range of 0 to 1. This typically increases learning.
When the data-set was originally created, an effort was made to include as much variance as possible in the data, i.e different backgrounds, different lighting etc. This intrinsic variance might in itself provide some sort of regularization, but one can imagine that when creating the data, one would take a bunch of pictures in one setting, before moving on the next. The data was therefore also randomly shuffled to prevent any class-correlation.
The data-set was split into 80% training- and 20% validation-data.

#### *4.1.1. Data Augmentation*

The architecture, U-net, was originally intended for learning from quite small data-sets, e.g. below a hundred samples. In comparison, this data-set is quite large with over one thousand different training examples. Data Augmentation often serves two purposes; artificial expansion of the training set (more examples for the model to learn from), as well as the inclusion of more *variance* within the data-set, which can have a regularizing effect. For this problem, it is arguable how necessary expansion of the data-set was, but the augmentation did seem to provide a regularizing effect, which is certainly desirable.
In the data-loading process, each image was subjected to a 50% chance of being augmented with each of the following transformations;

- Random Cropping
- Horizontal Flipping
- Gaussian Noise
- Gaussian Blur
- Rotation

The augmented data-set was appended to the original data-set. An undesirable trait of this approach, however, is the increase in training time. Other (affine) transformations were experimented with, however, the selected ones had the best effect.


*Example of a picture in the ShowMe data-set*

### 4.2. Regularization

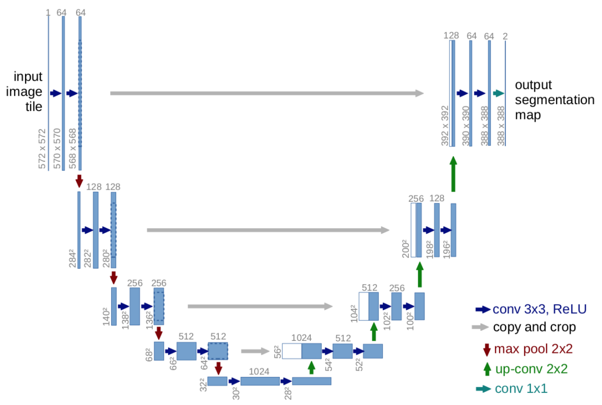
Apart from the augmented data-set, which provided a regularizing effect, other regularization techniques were implemented. The regularization techniques used are;

- Dropout
- Batch Normalization

The addition of L2 Loss Regularization was not found to provide any additional benefit.

## 5. NETWORK ARCHITECTURE

U-net is a very popular proposed model for end-to-end semantic image segmentation. The name refers to its characteristic shape, which resembles a U.



*The U-net Architecture*

The network used for this work has been adopted in the almost, same exact way, as the official U-net proposal. U-net adopts an encoder-decoder structure. The encoder side contracts the images, as they propagate through the network, while the encoder expands them again.
For every layer in the encoder, the encoder performs two 3x3

convolutions on the image in question each followed by a ReLU activation. The resulting output is then followed up with a 2x2 max-pooling operation for down-sampling the resolution of the image. At each down-sample step, the number of feature-channels in the image is doubled, by doubling the number of filters within the convolution operations.
The decoder upsamples the image to original resolution and feature-channels. For each layer in the decoder, the decoder performs two 3x3 convolution followed by ReLU (just like the encoder), before performing a transpose-convolution, also referred to as an ”up-convolution”. This can be thought of as the inverse to max-pooling, i.e. it upsamples the image.
For each upsampling step, the feature map from the corresponding side of the encoder is concatenated with the resulting map from the decoder-side. This is referred to as a ”skip-connection”. The purpose of the skip connections is to maintain the higher-level data from the encoder side in the decoder. The final output is computed with a 1x1 convolution that maps to the number of classes (here 2), along with the softmax function to provide probabilities.
In the original paper, U-net does not implement padding of the images. However, zero-padding *is* used in this work, as it is important that height and width of the masks are the same as the input images.

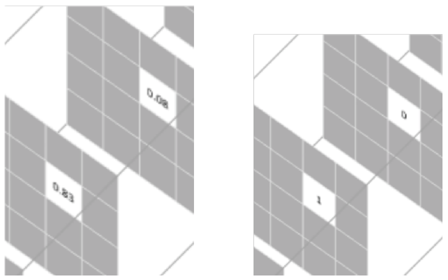
It is expected that the first couple of convolutions in the network will pick up on low-level features, such as edges and lines in the picture. As we move further down the decoder, the convolutions should be able to pick up the higher-level, more abstract features of a hand, such as fingers and the shape of the hand. The network will then in fact, during training, learn to detect these higher-level features and thus generate a segmentation map.

## 6. TRAINING

At first, a smaller version of U-net was implemented. This version only consisted of three downsamples and upsamples. Results were obtainable with this model, but once implementing the full version, results improved. This supports the theory that more downsamplings lead to better detection of higher-level features.

The network is trained on the input images and their corresponding segmentation masks, with the latter being used as labels. The network is trained on the train-set consisting of 915 images, and subsequently tested on 229.

The network outputs an image/feature map with the same number of channels, as classes in the data-set, i.e. two in this case. Image segmentation is essentially a pixel-wise classification task. The network outputs a probability per class, for each pixel, and the loss is thus calculated between the one-hot encoded labels, and the output probabilities;



*Pixel-wise output probabilities (left) are compared with one-hot encoded labels (right)*

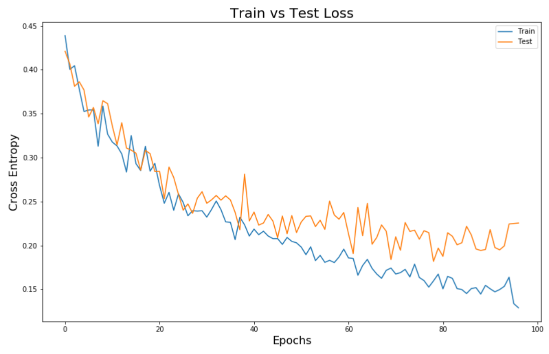
Since we are essentially dealing with a classification task, the loss function is thus chosen to be Cross Entropy, and is computed pixel-wise;

$$CE = -\sum_{x} p(x) \cdot log(q(x))$$

The loss is then backpropagated and Stochastic Gradient Descent is performed as we know it.
Although, a bunch of regularization parameters were in play, Early Stopping was also utilized to prevent overfitting and/or unnecessary, excessive training.
The model is furthermore trained with Adam Optimizer and a learning rate of 0.001. Training a network of this size, with a data-set of this size, with this many convolutions, is very computationally heavy, and reasonable training times were only achieved by utilizing a high-end GPU.
For the final model, Batch Normalization and 20% Dropout was applied alternately between convolutions in each layer of both encoder and decoder.

For inference, the output probabilities are transformed into an image by, for each pixel, assuming that the class with highest probability is the predicted one; $argmax(P(c_1), P(c_2))$

## 7. RESULTS

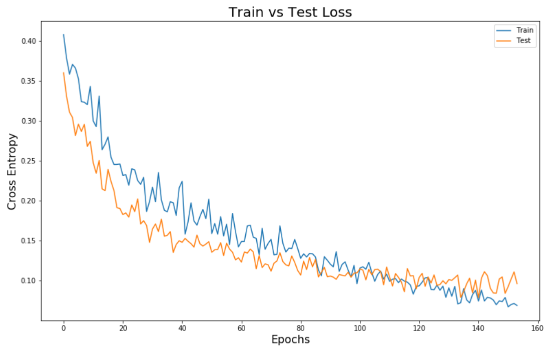


*Train- and test loss of base model*

The first experiments included an implementation of a smaller implementation of U-net. This was done to establish a starting point, or a baseline for further training and experimenting, while also saving computation power and time. This provides a sanity check, in regards to implementation. The smaller U-net consisted of only one convolution per layer, and only three downsamplings (and upsamplings), instead of 4, and as a consequence contains only 950,000 parametric weights, compared to the 7,700,000 of the full-scale U-net. The results obtained from this base model are hopeful and indicative of learning, although the results are far from ideal.

The loss of the model on the validation-set indicates learning, with the model earning a 0.81 F1-score on the validation set. In comparison, the untrained network (random guessing) achieves an F1-score of only 0.16.

Using the implementation of the full-scale U-net, results improved by a wide margin.



*Train- and test loss of full U-net*

When using the full scale of U-net, the model performs significantly better. As can be seen, slight overfitting is observed at the end of training. The Early-Stopping technique was used to automatically break the training when over-fitting seems to set in, and save the weights at the lowest test-loss, which are then loaded and used for predictions.

The regularization effect of the augmented data-set can be seen in the fact that the model consistently performs better on the validation-set, than on the training-set, until over-fitting sets in.

The final model was able to achieve an F1-score of just above 0.92 on the validation-set. This might not seem like a huge improvement from the base-model, but it does have a big effect on the output masks' resemblance to the given hand. For the output to resemble a hand well enough to fit the use case, the difference lies in the small improvements, say from an F1-score of .90 to 0.92. At first glance, one might assume Data-Augmentation is superfluous, since it only provides an

| **Model** | **Train Loss** | **Val Loss** | **Val F1 Score** |
|---|---|---|---|
| Untrained | 0.76 | 0.76 | 0.16 |
| Base Model | 0.13 | 0.25 | 0.81 |
| Dropout 10% + Batchnorm | 0.02 | 0.21 | 0.86 |
| Dropout 20% + Batchnorm | 0.05 | 0.10 | 0.90 |
| DataAug + Dropout 20% + Batchnorm | 0.05 | 0.08 | ***0.92*** |

**Table 1**: *Average Performance Metrics*

F1-score improvement of 0.02, but this is not the case. As seen in Table 1 the best model implements both high amounts of regularization and is trained on the data-set containing augmented data.

## 8. DISCUSSION

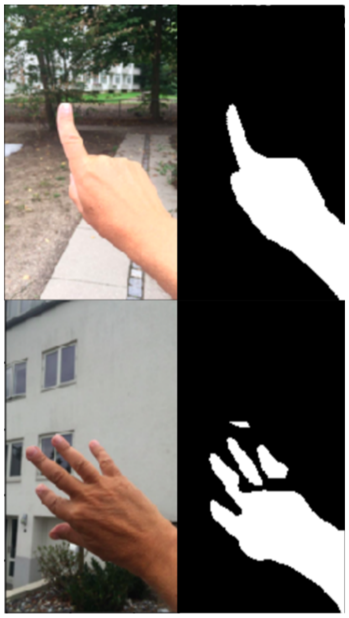
Given the results in the previous section, one can see that the model is in most cases, successful in generating a mask for a given image of a hand, that it hasn't seen before. It performs well on images with varying background, varying lighting, etc. It is clear that the model has indeed learned to recognize the features of a hand, and is not just reacting a single feature such as, say lighting difference.

The visible difference in performance between the base model, and the full U-net implementation supports the hypothesis, that this is what the model has learned to pick up on. With more convolutions and one more downsampling, the model performed remarkably better. The augmentation of the data-set surely played a role in improving the model's performance, by adding more variety to the data, and in this way, force the model to learn the underlying features.

It is uncertain by how much, but the model could probably be improved with further tuning of the hyper-parameters. Although many hyper-parameters, like learning rate, were tuned somewhat, an exhaustive search method like Grid Search would surely find a better solution for lower or earlier convergence.

Transfer learning is often used within image segmentation tasks to ease learning and speed up convergence. It is quite certain that transfer learning in this case would save training time, but it would be interesting to see if it could improve performance of the model, as well. One could imagine transferring weights from a segmentation task for other human features, for optimal performance.

Further future work could include expansion of the data-set to include different hands, of different genders and races.



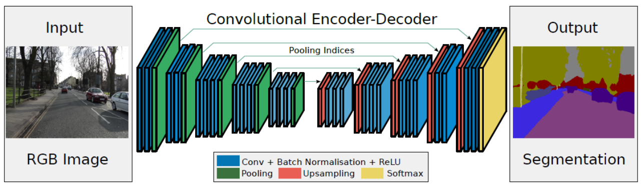
*Input & Output of the model*

This would provide the opportunity to see if that kind of variance is a learnable feature, or if the model's current learned abilities could handle such variance.

## 8.1. Experiments

It is also plausible that a different network architecture entirely might provide better results. An option, for instance, is the Segnet architecture, which has yielded promising results on multi-class semantic image segmentation. Segnet is also computationally lighter, and requires less memory, as its skip connections transfer the pooling indices, not the entire feature maps, which would leave more resources for training and tuning.

SegNet follows a similar architecture to U-net, and was (amongst other things) introduced to decrease the need for large memory and computing capacity, as is typically needed when using U-net.



*The SegNet Architecture*

Similiarly to U-net, SegNet follows an encoder-decoder structure and outputs pixel-wise classification probabilities for pixel-wise classification, and the concept is essentially the same. However SegNet implements more convolutions in both encoder and decoder, than U-net and an additional downscaling. These extra computations are made possible by freeing up computations in the skip-connections. SegNet uses a more efficient way to store the higher-level information from the decoder, by transferring the pooling indices from each max-pooling operation in the encoder, i.e. the indices for the maximum feature values. These indices are then used for the upsampling operations in the decoder, for a 'smart' upsampling, instead of a 'learned upsampling' as in U-net. When implementing U-net, a transpose-convolution was used for upsampling. SegNet thus frees up memory here, since no learning is needed for upsampling. The absence of a concatenation of the full feature maps, like in U-net, can have a negative impact on model performance - however the idea is that the freeing of resources in turn will provide a better opportunity for training and tuning of the model.

In this work SegNet was implemented using the exact architecture as in the official proposal. At first the simplest model was trained and tested, i.e without any regularization or optimization methods, and afterwards certain optimization methods was attempted, such as utilizing transfer learning by loading the weights from the VGG16 network (a different architecture intended for segmentation). However neither the simple base-model, nor the more elaborate could converge to a solution as good as was achieved with U-net. It is clear that the network was indeed learning, but got stuck in a local minimum, as it learned to decrease the loss by outputting a pure black image. Regularization was not necessary as the model is clearly underfitting. It is very plausible that it would be possible for SegNet to converge, given more hyperparameter tuning and optimization measures, however, time and resources were not available to further this experiment.

## 9. CONCLUSION

Although there is a lot of room for improvement, the results of training a deep convolutional network on this data-set for segmentation are positive. Arguably the model performs well enough on the validation data-set for ShowMe's use case, and is on-par with the performance of Akira Sosa's model "MobileUnet". It is entirely possible that the model could be improved by further tuning its hyper-parameters by performing an exhaustive search method, such as grid-search, or utilize transfer learning for optimal performance.

## 10. REFERENCES

[1] Philipp Fischer Olaf Ronneberger and Thomas Brox, ," in *U-Net: Convolutional Networks for Biomedical Image Segmentation*. University of Freiburg, Germany, 2015.

[2] Jeremy Jordan, ," in *An overview of semantic image segmentation*. 2018, `https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html`.

[3] Divam Gupta, ," in *A Beginner's guide to Deep Learning based Semantic Segmentation using Keras*. Carnegie Mellon University, 2019, `https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html`.

[4] Roberto Cipolla Vijay Badrinarayanan, Alex Kendall, ," in *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. IEEE, 2016.

[5] Akira Sosa, ," in *Real-Time Semantic Segmentation in Mobile device*, 2018.

[6] Alejandro Debus, ," in *Pytorch implementation of SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation.* 2018, `https://github.com/alejandrodebus/SegNet`.

[7] Sayan Goswami, ," in *Implementation of SegNet architecture in Pytorch.* 2019, `https://github.com/say4n/pytorch-segnet`.

[1] [2] [3] [4] [5] [6] [7]

## Appendix A: Code

A github repository containing the code can be found here: `https://github.com/FaxMan1/Deep-Learning`