Todo list

Real-time segmentation on smartphone

AXEL DEMBORG

Master in Computer Science
Date: May 2, 2018
Supervisor: Hossein Azizpour
Examiner: Danica Kragic
Swedish title: Realtids segmentering på smartphone
School of Electrical Engineering and Computer Science

Abstract

Sammanfattning

Acknowledgments

Contents

Chapter 1

Introduction

For 3D scanning of human bodies specialized hardware has traditionally been used. However with the recent developments in convolutional neural networks (CNN) where high quality object segmentation [37] and pose estimation [23] have been performed from RGB images it should be possible to do online segmentation of human bodies with commodity smartphone cameras. An issue for mobile deployments of these networks however is their shear size meaning that they can't fit in the on-chip SRAM and instead have to reside in the power hungry off-chip DRAM making applications up to 100 times more power consuming [21]. Another issue concerns the computational load of the models and means that the networks can't run in real-time on the relatively slow processing power of a smartphone.

Further issues with these neural networks are that the require huge amounts of labeled data to train and that generating high quality ground truth data is expensive. This is especially true for tasks like semantic segmentation where pixel level annotations have to be made for each image and labeling a single image is a tedious task, not to mention the thousands required for training.

1.1 Research Questions

To try and address these issues this thesis will focus on two research questions:

Make this into actual text

1. To what extent can modern neural networks be slimmed down and optimized for real-time execution on smartphones?

2. How well does performance from networks trained on synthetic datasets transfer to real world data?

Chapter 2

Background

2.1 Related work

Convolutional Neural Networks (CNN) where first introduced in 1998 [34] and since then larger and larger CNNs have slowly become the state of the art method for most areas of computer vision. Notably *AlexNet* [33] in 2012 proved that that deep CNNs could be used for high resolution image classification by beating the previous state of the art [43] on the *ImageNet* classification challenge [13]. To do this *AlexNet* used 60 million parameters and 650,000 neurons. Training of the network was only made feasible by the use of multiple graphical processing units (GPUs) [33].

In the areas of object detection and semantic segmentation it was *Regions with CNN features* (*R-CNN*) [16] that in 2014 first showed that CNNs could be successfully be applied to these fields by significantly improving over the previous state of the art in object detection [41] and after minor modifications matching the performance of the state of the art in semantic segmentation [7] with a system not specifically built for the task. For object detection *R-CNN* works as a hybrid system with *selective search* [48] producing proposals for object regions and a CNN, pre-trained on *ImageNet* [13] and fine-tuned for region classification, generating fixed length features for each region, finally classifying each region by running class specific *support vector machines* [5] on these features. Some issues with *R-CNN* are that it requires multistage training, training the CNN to give good features and training the SVMs for classification, and that it is slow, for training but most notably at inference where one image is processed in 47s. These problems are addressed with further work resulting in *Fast R-CNN* [15] where the CNN isn't run once per proposed region but instead once for the entire network generating a convolutional feature map that is then pooled with a region of interest (RoI) pooling layer to produce a feature vector for each region. These feature vectors are then feed into a fully connected neural network with two sibling output layers that perform both classification and bounding box refinement in parallel. With these improvements *Fast R-CNN* achieves faster inference and higher accuracy than its predecessors and does so with a arguably much more elegant design.

Even though *Fast R-CNN* improved speed significantly it was nowhere near real-time performance. Further performance improvements were introduced with *Faster R-CNN* [40] where *selective search* for region proposals is replaced with Region Proposal Networks, fully convolutional neural networks that take as input the convolutional feature maps as described from *Fast R-CNN* and outputs region proposals. Since this approach for region proposals shares most of its computation with the classification network the region proposals are practically free and frame rates of 5fps are achievable. The region proposal networks not only speed up computation but also prove to give better accuracy region proposals and thus raise over all accuracy in the system as well [40]. Even further improvements to this framework was achieved with the introduction of *Mask R-CNN* [23] which expands upon *Faster R-CNN* by adding a third branch for a segmentation mask besides the branches for bounding box refinement and classification making the system able to predict not only the general bounding box of items in the image but also which exact pixels belong to the object. Since segmentation is a pixel-by-pixel prediction problem *Mask R-CNN* replaces the spatially quantizing RoIPool operation from *Fast R-CNN* with a quantization-free layer called RoIAllign.

Some parallel work on semantic segmentation resulted in *SegNet* [3], a fully convolutional encoder-decoder network. Here the encoder network encodes the input image down into a lower dimensional feature space while storing the indices of the max pooling operations. The low dimensional representations are then run through a decoder network which is architecturally a mirror image of the encoder network but where the max pooling operations have been replaced with upsampling layers that use the stored indices from the corresponding pooling layers to maintain the granularity of the images. The final layer in the network is a softmax and hence the outputs are the probabilities of each pixel belong to each class. Continued work on segmentation utilizes blocks of so called *DenseNets* [26], CNNs where every layer is connected to every layer after it enabling the training of exceedingly deep network architectures by alleviating the vanishing gradient problem and promoting feature reuse between the layers. By using these *DensNets* in a very deep encoder-decoder structure where skip connections restore image granularity during upsampling the state of the art in image segmentation has been pushed even further [30] while still reducing the amount of parameters required for the models by a factor 10 as compared to the previous state of art.

Despite their impressive performance on a wide range of problems neural networks are still prohibited from running locally on mobile devices with slow processors, limited power envelopes or limited memory due to their large size and big computational load. For example modern neural networks can't fit on the on-chip SRAM cache of mobile processors and instead have to reside in the much more power hungry off-chip DRAM memory making applications up to 100 times more power consuming [21]. Regarding inference speed the most modern networks for object segmentation [23] run at 5fps but that is on high performance GPUs meaning that mobile performance is far from realtime. Due to these limitations applications of neural networks for mobile use cases are either to forced give up on state of the art performance or to be run on off-site servers which requires steady network connections and incurs delays, both of which may be intolerable for real-time mobile applications, self driving cars and robotics [31]. However work on understanding the structure of the learned weights in neural networks [14] has showed that there is significant redundancy in the parameterization of several deep learning models and that up to 95% of weights in networks can be predicted from the remaining 5% without any drop in accuracy. This indicates that models could be made much smaller while still maintaining performance and several such approaches for squeezing high performance networks into small memory footprints and computational loads have been proposed. The most prominent approaches will be presented below.

2.1.1 Quantization of weights

Modern neural networks are usually based on 32-bit floating point representations of parameters. It has been shown however that networks are quite resilient to noise and even that some noise can improve training [36]. Since reduced precision variables can be modeled as noise this means that networks can be compressed by changing to a less accurate format without any loss in performance. This can be done either by reducing the bit accuracy after training [51] or by doing the entire training in reduced accuracy [27] [18]. The benefits of using a reduced format like this for representation is not only that the models take less space but also that the individual multiplications become cheaper and hence the networks run faster.

2.1.2 Weight sharing

One of the most direct approaches for removing the redundancy in parametrization from neural networks is by forcing the networks to share weights between different connections. This is precisely what *HashedNets* [10] does by fixing the amount of weights K^l that are to be used in each layer making the weights $\vec{w^l} \in \mathbb{R}^{K^l}$ and using hashing functions to map each element in the virtual weight matrices V^l_{ij} to one of these weights $V_{ij} = w_{h(i,j)}$ with $h()$ being a hashing function. With the weight matrices defined in this fashion *HashedNets* can be trained like normal networks with the gradients with respect to the weights calculated from the gradients with respect to the virtual matrices as

$$\frac{\partial \mathcal{L}}{\partial w^l_k} = \sum_{ij} \frac{\partial \mathcal{L}}{\partial V^l_{ij}} \frac{\partial V^l_{ij}}{\partial w^l_k}$$

This method gave a compression of about 20 times before any notable loss in accuracy was introduced during tests on variations of the MNIST dataset which seems to agree very well with the results from [14].

Other notable work focuses on the use of k-means clustering to cluster the weights in networks after training [17], this proves to work very well and manages to compress the models with a factor 16 with no more than a 0.5% drop in classification accuracy on the ImageNet dataset. Further work in this area explores the effects of pruning away low-weight connections and iterative retraining of the pruned networks

[21]. This lets the authors compress models with a factor 9 - 13 without any loss in performance while getting sparser weight matrices that could potentially speed up calculations. These two lines of research, clustering and pruning, where merged into a single framework called *deep compression* [19] where a three stage approach is taken to model compression. First low-weight connections are pruned away and the network is retrained to compensate for this, in the second stage k-means clustering is performed on the weights and again the network is retrained to make the clusters take the most useful values, finally Huffman coding [50] is used to reduce the storage required for the weights. This process allows *deep compression* to compress networks with a factor 35 without any loss in accuracy. Despite these very impressive results however *deep compression* comes with a major drawback, it can't be run efficiently in its compressed form and the full weight matrices have to be rebuilt at inference time to use the models on commodity hardware. To alleviate these problems hardware has been designed that could perform prediction directly from the compressed models. This so called *efficient inference engine* [20] would enable inference 13 times faster than GPU while being 3400 times more energy efficient.

2.1.3 Student-teacher learning

Student-teacher learning is a type of model compression where a smaller and/or faster to compute *student* network is trained by making it learn the representations learned by a larger *teacher* network. This idea was first introduced for compressing ensemble models produced by *Ensemble Selection* [8] which consist of hundreds of models of many different kinds, support vector machines, neural networks, memory based models, and decision trees into a single neural network [6]. This work leverages the neural networks property of being universal approximators [12], meaning that given sufficiently much training data and a big enough hidden layer a neural network can learn to approximate any function with arbitrary precision. This is done by not directly training the student network on the relatively limited labeled training data available but instead on large amounts of pseudo random data that has been given labels by first being passed through the large teacher ensemble. This compression technique yielded student networks up to 1000 times smaller and 1000 times faster to compute than their ensemble teachers with a negligible drop in accuracy on some test problems.

Further work on student-teacher learning experiments with why deep neural networks usually perform better than shallow ones, even when they have the same amount of parameters. This was done by training shallow student models to mimic deep teachers [2]. The work introduces two major modifications that make training of these student models feasible, firstly the student model isn't tasked with just recreating the same label as the teacher but also the same distribution which is achieved by regressing the student to the logits, log probability, values of the teacher as they were before softmax. Getting predictions from the student is then achieved by adding a softmax layer to the end of it after training. Secondly a bottleneck linear layer is added to the network to speed up training. With these modifications they are able to train flat neural networks for both the TMIT and CIFAR-10 datasets with performance closely matching that of single deep networks. Continued analysis of flat networks however shows that depth and convolutions are critical for getting good performance on image classification datasets [49]. Empirically this claim is supported by training state of the art, deep, convolutional models for classification on the CIFAR-10 dataset and then building an ensemble of such models using that as a teacher for shallow students. The student models were then compared to deep convolutional benchmarks that were not trained in a student-teacher fashion. To make sure that the networks were all performing to the best of their abilities and thus making the comparison fair Bayesian hyperparameter optimization [46] was used. Through this thorough analysis it was shown that shallow networks are unable to mimic the performance of deep networks if the number of parameters is held constant between them, these findings are also in agreement with the theoretical results that the representational efficiency of neural networks grows exponentially with the number of layers [35].

Improvements to the student-teacher learning method have been proposed where the student is tasked with minimizing the weighted average of the cross-entropy between its own output and the teacher output when the last layer is softmax with increased temperature, yielding softer labels, and the cross-entropy between the student output and the correct labels when they are available. This framework is called *Distillation* [24] and proves to work very well for transferring of information from teacher to student. The framework is demonstrated by training a student model with only 13.2% test error on the MNIST dataset despite only having seen 7s and 8s during its own training. These results mean that distillation manages to transfer knowledge about how a 6 looks from the teacher to the student by only telling it to what degree different 7s and 8s don't look like 6s.

Continued work lead to the creation of *FitNets* [42] which goes in the opposite direction to previous attempts at student architectures and instead proposes very deep but thin students. To enable learning in these deep student networks a stage-wise training procedure is used. In the first stage intermediate layers in the teacher and student networks are selected, these are called *hint* and *guided* layers respectively. The guided layer in the student is then tasked to mimic the hint layer in the teacher through a convolutional regressor that compensates for the difference in number of outputs between the networks, this procedure gives a good initialization for the first layers in the student and allows for it to learn the internal representations of the data from the teacher. The second stage of training is then distillation as described above but with the small addition that the weight of the loss against the teacher is slowly annealed during training. This annealing allows for the student to lean heavily on the teacher for support in early stages of training and learn samples which the even the teacher struggles with towards the end of its training. Using this approach the *FitNets* manage to produce predictions at the same level or in some cases even better than models with 10 times more parameters.

Some more recent work [44] builds upon the ideas from *FitNets* with not only letting the students mimic the output of teachers but also some intermediary representations. Unlike the way it is done *FitNets* however the student is not tasked with reconstructing the exact activations of the teacher in the intermediate layers but instead the attention maps, regions in the image that the teacher uses to make its predictions, and thus teaches the student where to look. A few different methods for calculating these attention maps are proposed in the paper but notable is that they are all non parametric meaning that no extra layers of convolution have to be learned to make the student attention maps comparable to the ones from the teacher. This attention transferring approach proves to give good results on a number of difficult datasets including *ImageNet* and is also shown to work well together with distillation.

2.1.4 Architectural optimizations

Another orthogonal approach for compression is to optimize the convolutional layers them selves making them require less parameters or less computation to perform their tasks but still keep as much as possible of their representational power. One of the simplest things that can be done here is to replace single layers of $N \times N$ convolutional filters with two layers with $N \times 1$ and $1 \times N$ filters respectively, this reduces the amount of parameters that have to be stored per channel from N^2 to $2N$ and the amount of multiplications that have to be made scale in the same way. These so called asymmetrical convolutions have seen successful use in inception models [47]. Other variations on the convolutional operator that help compress the networks are dilated convolutions [52] where an exponentially expanding receptive field is achieved without the need for any extra parameters. There have also been some promising results from *depthwise separable convolutions* where the convolution is factored into a depthwise convolution followed by a pointwise 1×1 convolution reducing the computational load with a factor 8 to 9 for 3×3 convolutional kernels [25]. This scheme was introduced in [45] and has since seen been successfully used in *Inception* models [29].

SqueezeNet [28] presents a different take on how to get smaller models in that it rather optimizes the architecture of the network than any of the constituent parts, this approach gives a network with *AlexNet* performance but with 50 times fewer parameters than *AlexNet*. This is done by focusing on the usage of 1×1 convolutional filters, reducing the amount of channels that go in to the larger filters and by holding out on downsampling so that feature maps are kept large through the network. It was also proven that these results were orthogonal from compression by running the SqueezeNet through the *deep compression* framework [19] and getting further 10 times compression with out accuracy loss.

MobileNets [25] combine these two approaches, utilizing both *depthwise separable convolutions* and a heavily optimized architecture to build networks specially suited for mobile vision applications. In doing so *MobileNets* also introduce two hyper-parameters, *width-multiplier* and *resolution-multiplier* that help design models with a optimal trade off between latency and precision given the limitations of the available hardware.

Another network specially designed for real-time segmentation on mobile devices is *ENet* [38]. Here dilated convolutions are used together with asymmetrical convolutions to give a large receptive field without introducing that many parameters. The network is built as an encoder-decoder network but with a much smaller decoder, the argument behind this being that decoder should simply upsample the output while fine-tuning the details which should be a simpler task than the information processing and extraction that the encoder is performing. Attention has also been payed to quickly downsampling the feature maps which saves on computation but then not downsampling so aggressively after that, keeping much of the spatial information in the images. Together these improvements give a network that performs on par with *SegNet* but that requires 79 times fewer parameters and is 18 times faster at inference.

An other modern network architecture that has been specifically designed for fast and efficient segmentation is *LinkNet* [9]. Here residual blocks [22] are used to produce a state of the art network which can process high resolution frames at almost 10 fps.

Chapter 3

Experiments

3.1 Data

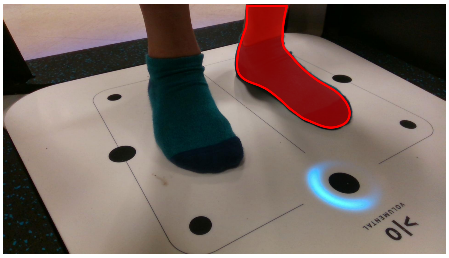
Two datasets of segmented feet were used for the experiments.

Firstly a dataset where images and 3D-models where extracted from *Volumental's* 3D-scanner and composited with floor images scraped from the internet to create synthetic images of feet on normal floors. This process is illustrated in fig. 3.1. This dataset is called *synthetic* and is divided into training, validation and test sets with 43896, 12960 and 14168 images respectively. The different sets have been constructed to ensure that there is no overlap of floors or feet between them. Example images from this dataset can be seen in fig. 3.2.

Secondly a dataset of 111 images taken of employees at *Volumental* that has been segmented by hand. This dataset is called *real* and is primarily used for testing how the methods generalize from the synthetic data to real images. Example images can be seen in fig. 3.3.

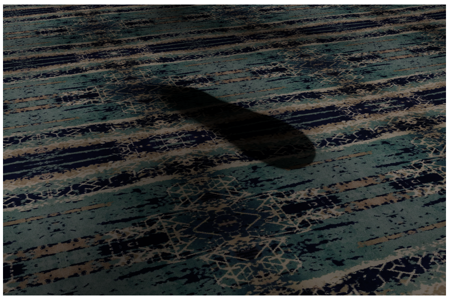
3.1.1 Data augmentation

One restriction with the *synthetic* dataset is that, due to the placement of the cameras, the images from the scanner only come from four different angles, rather fixed in regards to the foot and that they are all taken from approximately the same distance. To enable the algorithms to learn more general foot features despite these restrictions the original images are augmented by dividing each image into a 3×3 grid, selecting a random point in each corner cell, and performing an affine transformation to make these selected points the corners of a new

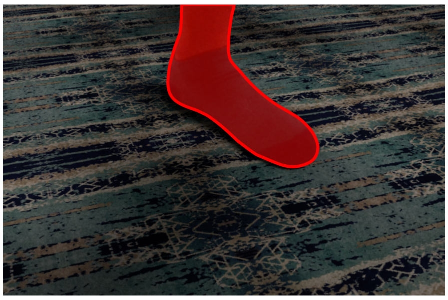


(a) The foot on the scanner

(b) Floor texture



(c) Texture in scanner plane with added shadows



(d) The final image with segmentation

Figure 3.1: Steps for creating the synthetic data

Figure 3.2: Examples of images and segmentations from the synthetic dataset

$256 \times 256px$ image.

This augmentation introduces some random zoom, rotation, shear and translation to the images and should hence help the algorithms learn generalizable features. Some example of this can be seen in fig. 3.4

3.2 Network architectures

For the experiments four different neural networks were evaluated against each other. Details on each of these follow below.

Figure 3.3: Examples of images and segmentations from the real dataset

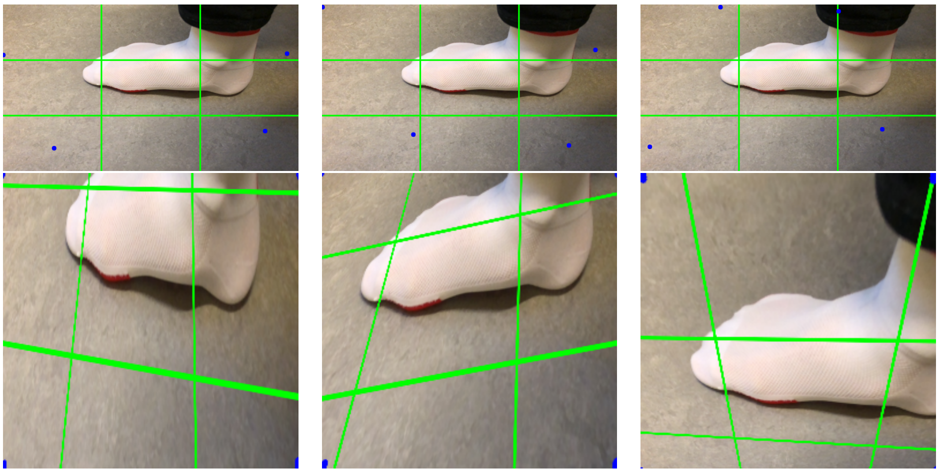


Figure 3.4: Some random augmentations on the same image. Top row is the original image with the sampled corners are indicated in blue and the sample boundaries are indicated in green. Bottom row is the resulting image.

3.2.1 ENet

hang on, why is my enet 10x bigger and 3 times slower than the original?

This is a re-implementaiton of the *ENet* architecture [38] where the unpooling operations from the upsampking blocks has been replaced with addition of the corresponding feature map from the downsampling path due to restrictions in *Keras*

3.2.2 LinkNet

This is a straight re-implementation of the LinkNet architecture [9].

3.2.3 MobileSeg

This is a network for semantic segmentation that was built using the body of the *MobileNet* architecture [25] as the encoder of the network and then adds the upsampling path from *LinkNet* to this body and skipp connections are introduced where the dimensions correspond to those in *LinkNet*

3.2.4 FastLinkNet

This network is designed as a hybrid between *LinkNet* and *MobileNet* where the the layout of *LinkNet* has been copied but inspired by *MobileNet* all the convolutions in the encoder blocks have been replaced with depthwise separable convolutions, thus reducing model size with approximately a factor 4. Further inspiration was taken from the resolution multiplier in the *MobileNets* which is used to reduce the resolution of the incoming image data to reduce the computation needed to process it by downscaling the images by a factor 2 before segmentation and then bilineraly upscaling the image to the full input size at the end.

3.3 Loss functions

To train the networks for semantic segmentation three different loss functions were evaluated.

3.3.1 Cross entropy

The classical loss function for classification tasks is *Cross Entropy*.

$$L_{CE} = (L)(Y, \hat{Y}) = -\sum_i Y_i \log \hat{Y}_i$$

Where Y is ground truth and $\hat{Y}$ is the prediction and the index i goes over all the pixels in the images. Since semantic segmentation is the task of classifying each pixel in the image this is a reasonable loss function.

3.3.2 IoU loss

In semantic segmentation a common performance measure is the intersect over union *IoU* metric between the predicted segmentation and the ground truth. *IoU* is defined as follows.

$$IoU = \frac{I}{U} = \frac{TP}{FP + TP + FN}$$

Where I is the intersect between the prediction and ground truth, U the union between them and FP, FN, and TP indicate the false positive, false negative and true positive respectively. These different quantities are illustrated in fig. 3.5.

This metric is used since it gives small objects as much weight as bigger ones. This stands in comparison to using the proportion of correctly classified pixels where classifying a image with a lot of background and a tiny foot as all background would give good results.

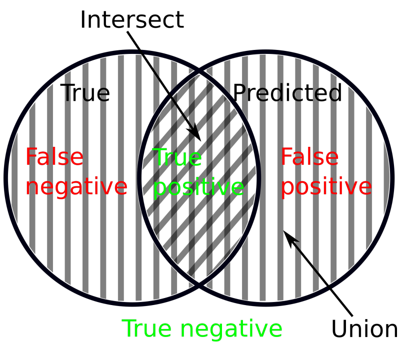


Figure 3.5: Notation for *IoU*

To directly optimize for *IoU* Rahman and Wang [39] introduced a *IoU Loss*.

$$L_{IoU} = 1 - IoU$$

If we now let $V = 1, 2, \ldots, N$ be the set of pixels in the image, $\hat{Y}$ the softmax outputs from the network at each pixel and Y the ground truth segmentation. The intersection $I(\hat{Y})$ and union $U(\hat{Y})$ can be approximated with the following:

$$I(\hat{Y}) = \sum_{v \in V} Y_v \odot \hat{Y}_v$$

$$U(\hat{Y}) = \sum_{v \in V} \left(Y_v + \hat{Y}_v - Y_v \odot \hat{Y}_v \right)$$

Where $\odot$ represents the element-wise Hadamard product.

The *IoU* loss hence becomes

$$L_{IoU} = 1 - \frac{I(\hat{Y})}{U(\hat{Y})}$$

3.3.3 Distillation

When training using distillation loss [24] a large previously trained network is used to help guide the student model during training. The loss function here is defined as:

$$L_{distillation} = \mathcal{L}(Y, \hat{Y}) + \lambda T^2 \mathcal{L}(\hat{Y}^*, \hat{Y}^*_{teacher})$$

Where $\mathcal{L}$ is the cross entropy loss from section 3.3.1 and $\hat{Y}^*$ and $\hat{Y}^*_{teacher}$ are the softened, high temperature softmax outputs, see fig. 3.6 from the student and teacher networks respectively. T is the temperature with which the softmax were softened and it is used to compensate for the fact that the gradient with respect to the soft loss decreases with a factor $\frac{1}{T^2}$ as explained in the initial paper [24]. The high temperature softmax is defined as follows:

$$\hat{Y}^*_i = \frac{e^{a_i/T}}{\sum_j e^{a_j/T}}$$

Here a are the activations from the preceding layer and T is the temperature. If $T = 1$ we get the normal softmax values and if $T > 0$ we get a softer output distribution making it easier for the student to learn from the low probability classes of the teacher.

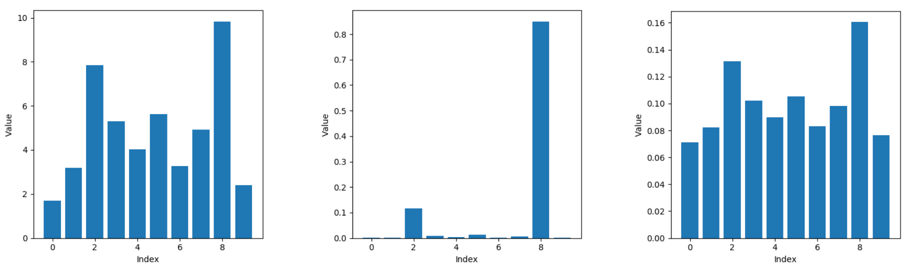


Figure 3.6: An illustaration of softmax using different temperatures, to the left is the input, in the middle output from $T = 1$ softmax and to the right $T = 10$

3.4 Training

The networks were defined and trained in *Keras* [11] using the *tensorflow* [1] backend, enabling easy experimentation and fast execution on the GPUs available. Hyperparameter optimization was handled by the *hyperopt* [4] package which worked neatly along with the other frameworks.

Training was run for 50 epochs of 50 steps each with a batch size of 32. $ADAM$ [32] was used as the optimizer and hyperparameter optimization was run for 100 iterations of random search in the search space $0.000001 < \eta < 0.01$, $0 < p_{dropout} < 1$, $f_{loss} \in \{L_{CE}, L_{IoU}, L_{distillation}\}$. And $1 < T < 100$, $0 < \lambda < 10$ when distillation loss is used.

3.5 Benchmarking

To evaluate the degree to which the models can be run on a smartphone they were tested on a *ZTE Axon 7* android phone using *tensorflows* benchmarking tools. From this inference time, and number of Floating point operations (*FLOPs*) calculated for inference was extracted. The size required to store the models was also measured.

Chapter 4

Results

The resulting segmentations from the best trained models of each architecture can be seen for the real dataset in fig. 4.1 and for the synthetic data set in fig. 4.2. The mean pixel accuracy and IoU metrics achieved by the models on the test set of the synthetic dataset and the real dataset can be seen in table 4.2 and the results from the benchmarks on the models are listed in table 4.1.

Table 4.1: Comparison of size and inference speed between the different models

Model	FLOPs [B]	Inference time [ms]	Size [Mb]
EmilSeg	55.37	4439	85
ENet	10.38	1045	15
LinkNet	1.75	234	45
MobileSeg	2.68	555	18
FastLinkNet	0.287	73	7.2

Table 4.2: Comparison of the test performance between the different models

Model	Test accuracy	Test IoU	Real accuracy	Real IoU
EmilSeg	0.9912	0.9860	0.9740	0.9814
ENet	0.9774	0.9116	0.9738	0.9117
LinkNet	0.9797	0.9141	0.9724	0.9145
MobileSeg	0.9834	0.9146	0.9744	0.9144
FastLinkNet	0.9797	0.9153	0.9707	0.9148

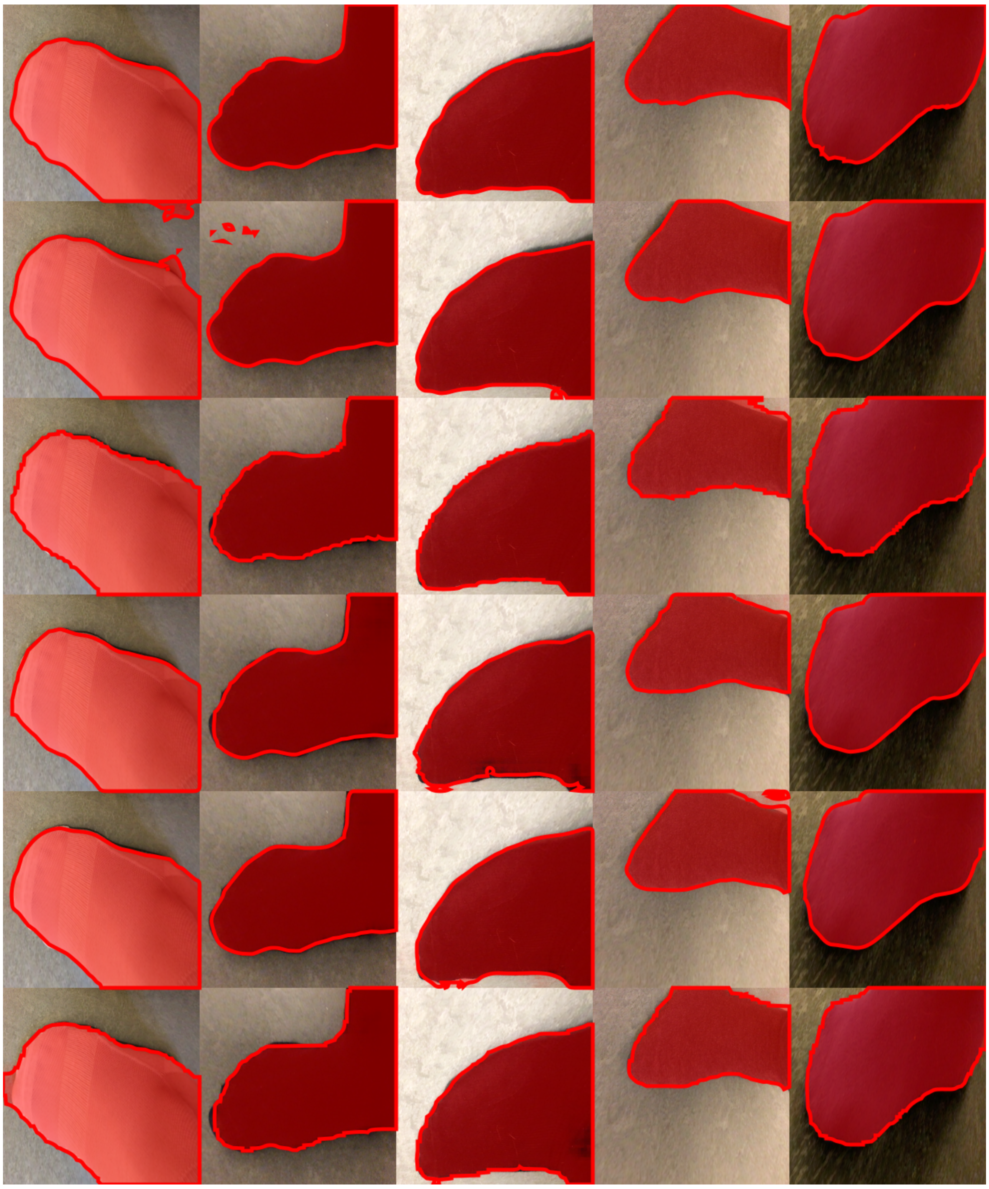


Figure 4.1: Resulting segmentations on the real dataset from the different models. First row is the ground truth, below that EmilSeg, ENet, LinkNet, MobileSeg, and FastLinkNet in that order.

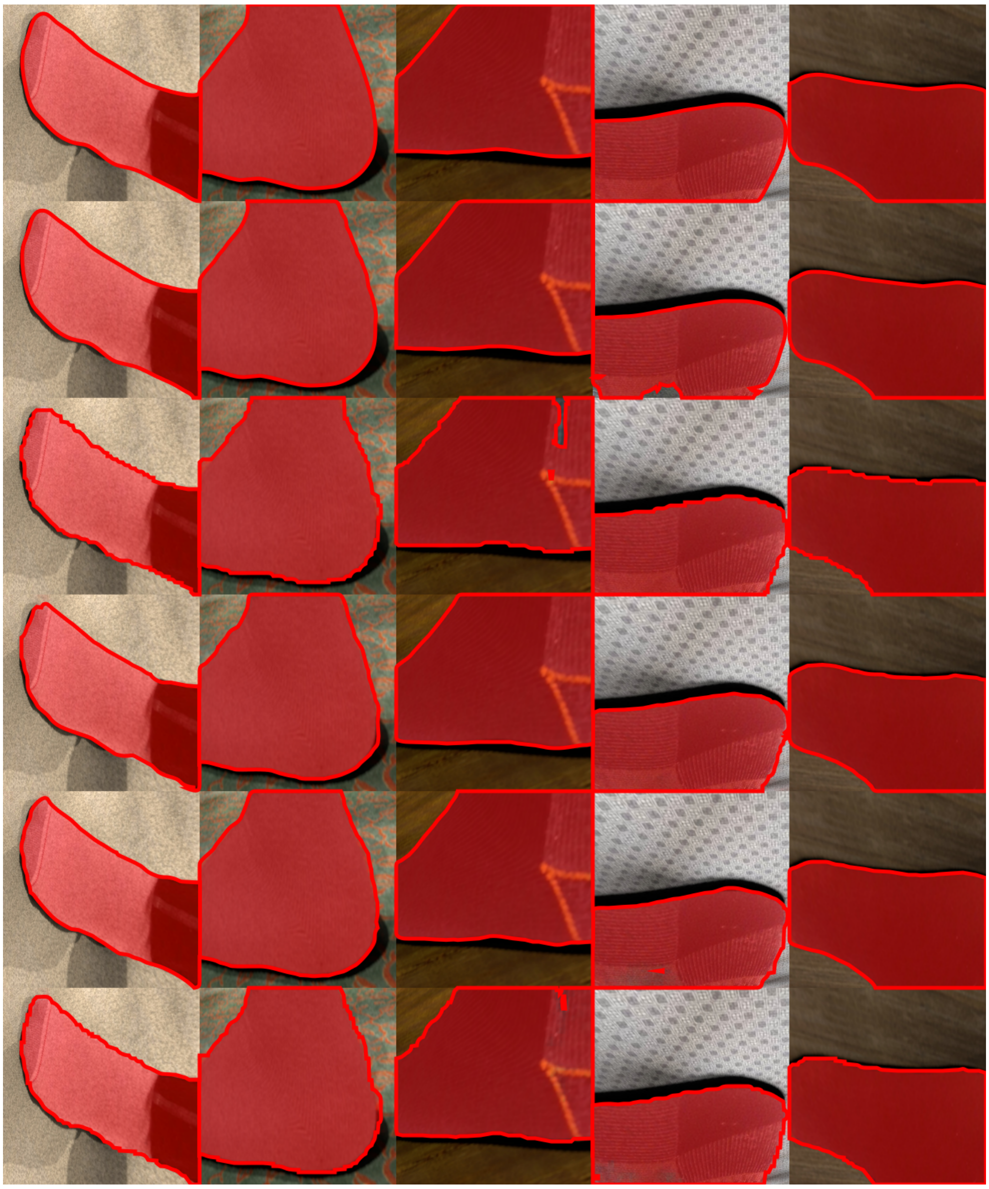


Figure 4.2: Resulting segmentations on the test set of the synthetic dataset from the different models. First row is the ground truth, below that EmilSeg, ENet, LinkNet, MobileSeg, and FastLinkNet in that order.

Chapter 5

Discussion

From table 4.2 it is interesting to note that all of the trained networks perform almost the same and very close to the big baseline model *EmilSeg*. This might be due to the fact that the datasets contain too little complexity and hence present a rather easy problem for the models. Some tests from running the models on a smartphone and testing it out in real world applications seem to support this since the model really struggles to handle complex scenes, multiple feet in the image, differing image scales, bare feet etc. Some tests of this can be seen in appendix A.

5.1 Further work

+ What other things can be done to increase performance? + Use temporal aspect of real data + Add some momentum to pixels, kind of like persistence of vision in humans + Add LSTM at bottleneck (too slow) + Feed last prediction back as additional channels (tested a bit, didn't get it to work) + Do pretraining on other data + Train encoder on say ImageNet to learn visual features? + Train for segmentation on something more general (DAVIS?) and finetune for feet? + Something about how the synthetic dataset is built, may be, using actual 3d-models or GANS? would give better training data?

This is just some notes for now

Due to time constraints the teacher model *EmilSeg* in distillation was not subjected to extensive hyperparameter optimization and was not necessarily trained to full convergence meaning that distillation could get better results if more time was spent on tweeking not only the students but also the teacher.

Bibliography

[1] Martin Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Jimmy Ba and Rich Caruana. "Do deep nets really need to be deep?" In: *Advances in neural information processing systems*. 2014, pp. 2654–2662.

[3] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling". In: *arXiv preprint arXiv:1505.07293* (2015).

[4] James Bergstra, Daniel Yamins, and David Daniel Cox. "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures". In: (2013).

[5] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM. 1992, pp. 144–152.

[6] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. "Model compression". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 535–541.

[7] Joao Carreira et al. "Semantic segmentation with second-order pooling". In: *European Conference on Computer Vision*. Springer. 2012, pp. 430–443.

[8] Rich Caruana et al. "Ensemble selection from libraries of models". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 18.

[9] Abhishek Chaurasia and Eugenio Culurciello. “LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation”. In: *arXiv preprint arXiv:1707.03718* (2017).

[10] Wenlin Chen et al. “Compressing neural networks with the hashing trick”. In: *International Conference on Machine Learning*. 2015, pp. 2285–2294.

[11] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[12] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[13] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[14] Misha Denil et al. “Predicting parameters in deep learning”. In: *Advances in neural information processing systems*. 2013, pp. 2148–2156.

[15] Ross Girshick. “Fast r-cnn”. In: *arXiv preprint arXiv:1504.08083* (2015).

[16] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[17] Yunchao Gong et al. “Compressing deep convolutional networks using vector quantization”. In: *arXiv preprint arXiv:1412.6115* (2014).

[18] Suyog Gupta et al. “Deep learning with limited numerical precision”. In: *International Conference on Machine Learning*. 2015, pp. 1737–1746.

[19] Song Han, Huizi Mao, and William J Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”. In: *arXiv preprint arXiv:1510.00149* (2015).

[20] Song Han et al. “EIE: efficient inference engine on compressed deep neural network”. In: *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE. 2016, pp. 243–254.

[21] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[22] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[23] Kaiming He et al. "Mask r-cnn". In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE. 2017, pp. 2980–2988.

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[25] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[26] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Vol. 1. 2. 2017, p. 3.

[27] Itay Hubara et al. "Quantized neural networks: Training neural networks with low precision weights and activations". In: *arXiv preprint arXiv:1609.07061* (2016).

[28] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[29] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. 2015, pp. 448–456.

[30] Simon Jégou et al. "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE. 2017, pp. 1175–1183.

[31] Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. "Flattened convolutional neural networks for feedforward acceleration". In: *arXiv preprint arXiv:1412.5474* (2014).

[32] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: `1412.6980`. URL: `http://arxiv.org/abs/1412.6980`.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[34] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[35] Shiyu Liang and R Srikant. "Why deep neural networks for function approximation?" In: *arXiv preprint arXiv:1610.04161* (2016).

[36] Alan F Murray and Peter J Edwards. "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training". In: *IEEE Transactions on neural networks* 5.5 (1994), pp. 792–802.

[37] Dhruv Parthasarathy. *A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN*. `https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4`. Accessed: 2018-02-06.

[38] Adam Paszke et al. "Enet: A deep neural network architecture for real-time semantic segmentation". In: *arXiv preprint arXiv:1606.02147* (2016).

[39] Md Atiqur Rahman and Yang Wang. "Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation". In: *International Symposium on Visual Computing*. Springer. 2016, pp. 234–244.

[40] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[41] Xiaofeng Ren and Deva Ramanan. "Histograms of sparse codes for object detection". In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pp. 3246–3253.

[42] Adriana Romero et al. "Fitnets: Hints for thin deep nets". In: *arXiv preprint arXiv:1412.6550* (2014).

[43] Jorge Sánchez and Florent Perronnin. "High-dimensional signature compression for large-scale image classification". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pp. 1665–1672.

[44] Nikos Komodakis Sergey Zagoruyko. "Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer". In: *International Conference on Learning Representations* (2017). URL: `https://openreview.net/forum?id=Sks9_ajex`.

[45] Laurent Sifre and PS Mallat. "Rigid-motion scattering for image classification". PhD thesis. Citeseer, 2014.

[46] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[47] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[48] Jasper RR Uijlings et al. "Selective search for object recognition". In: *International journal of computer vision* 104.2 (2013), pp. 154–171.

[49] Gregor Urban et al. "Do deep convolutional nets really need to be deep and convolutional?" In: *arXiv preprint arXiv:1603.05691* (2016).

[50] Jan Van Leeuwen. "On the Construction of Huffman Trees." In: *ICALP*. 1976, pp. 382–410.

[51] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. "Improving the speed of neural networks on CPUs". In: *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*. Vol. 1. Citeseer. 2011, p. 4.

[52] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions". In: *arXiv preprint arXiv:1511.07122* (2015).

Appendix A

Real world tests

add a capture image button to the app and go out in the world and take some cool images