Real-time segmentation of feet on smartphone

AXEL DEMBORG

Master in Computer Science
Date: May 29, 2018
Supervisor: Hossein Azizpour
Examiner: Danica Kragic
Swedish title: Realtids segmentering av fötter på smartphone
School of Electrical Engineering and Computer Science

Abstract

Image segmentation, the problem of dividing an image into meaningful parts is a very interesting problem in computer vision. Segmentation is relevant for everything from self-driving cars to 3D-modeling and synthetic camera effects. Even though there currently exist models for producing high-quality segmentations, most of these are too resource intensive to run in real-time on mobile devices, something that is more and more requested for user experience in computer vision apps and safety in self-driving cars.

To build models capable of real-time segmentation on smartphones this work focuses on building streamlined neural networks and compression of these using effective approximations of the convolutional operator and student-teacher training techniques. Specifically, this is done for segmentation of feet from the background which is a part of a larger project for 3D-scanning of feet on smartphones conducted at Volumental.

The fastest neural network produced in this work manages to run at over $10fps$ on an Android smartphone from 2016 without the use of hardware acceleration. Despite this speed, the network manages to achieve a mean intersect over union ($mIoU$) score of over 93% on a held out test set.

Sammanfattning

Bildsegmentering, problemet att dela in en bild i meningsfulla bitar, är mycket intressant för datorseende. Segmentering är relevant för allt från självkörande bilar till 3D-modelering och syntetiska kamera effekter. Även om det finns modeller som producerar goda segmenteringar är många av dessa allt för resurs krävande för att kunna köras i realtid på mobilhårdvara, något som blir allt mer efterfrågat för användarupplevelsen i datorseende appar och säkerheten i självkörande bilar.

För att bygga modeller kapabla att göra realtids segmentering på smartphones fokuserar detta arbete på att bygga ströpressentationmlinjeformade neuronnät och att komprimera dessa med hjälp av effektiva approximationer av faltningsoperatorn och student-lärar träning. Specifikt görs detta för segementering av fötter från bakgrund vilket är en del av ett större projekt för mobil 3D-scanning som på företaget Volumental.

Det snabbaste neuronnätet som byggts i arbetet klarar av att köra i $10fps$ på en Android telefon från 2016, utan att använda hårdvaruacceleration. Trotts denna hastighet lyckas nätverket uppnå ett *mean intersect over uninon* ($mIoU$) på över 93% på testdata.

Acknowledgments

Firstly I would like to thank my supervisors, Alper Aydemir and Hossein Azizpour for the help and support they have given me in compleeting this thesis. This would have been much harder and less fun without you.

I would also like to thank Emil Ernfeldt for his fantastic help with creating and managing the datasets used in this work as well as for building the monstrosity of a network that was used as a teacher in some of the experiments.

Further I would like to extend a huge thanks to the entire staff at Volumental whom have made it a joy to go to work each day, even when my code has just been crashing. It has been a true pleasure to share an office with all of you!

Contents

Chapter 1

Introduction

Segmentation is the problem of partitioning an image into multiple, meaningful regions, allowing for an easier understanding of images. Semantic segmentation is when the class of the region is also predicted, this problem can also be formulated as predicting the class of each pixel in an image. These are both very important problems in computer vision and are relevant for everything from scene understanding and autonomous driving to synthetic camera effects. Early approaches utilizing low-level image features to perform segmentation have recently been outmatched by machine learning models processing the entire image at once. Deep convolutional neural networks (CNN:s) have been especially prominent for these tasks.

With these rapid developments, modern smartphones with their high-quality cameras, quite able processors and immense availability have become a sought-after target to run CNN:s on. Deep neural networks are however normally very large models and tend to require a lot of computation to evaluate. These requirements are problematic for mobile deployments. Firstly, the large models can't fit in the on-chip cache and instead have to reside in the power-hungry off-chip DRAM making applications up to 100 times more power consuming [21]. Secondly, the mobile processors can't keep up with the computational load of the models and hence take seconds to run. This gives performance far from real-time which is required for smooth operation in many cases. To address these issues methods for compressing models and designing streamlined models that can be run in real-time on mobile hardware has been investigated.

In the intersection of these two lines of research, segmentation and

streamlining of neural networks lies this thesis. The project was performed at the company *Volumental*, a company specialized in 3D scanning of feet for custom shoe recommendations. The companies current product is a designated 3D scanner but for the online shopping experience, a mobile app with similar capabilities is required. The first step towards such an application is to create a model that can reliably perform segmentation of feet in real-time on a smartphone.

A problem for supervised machine learning methods and deep neural networks in particular is that they require huge amounts of annotated data for training. Producing these labeled datasets can be very time-consuming and expensive especially for image segmentation where producing a single high quality ground truth frame is a tedious task for a human. Having access to data from Volumental's dedicated 3D scanner, however, means that high quality synthetic datasets can be produced and learned from, reducing the need for manual annotation.

Problem formulation: *To build a system for real-time segmentation of feet that can reliably be run on a smartphone and be trained on synthetic data.*

1.1 Research Questions

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

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) were first introduced in 1998 [34] and since then deeper and deeper 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 can be used for high-resolution image classification by beating the previous state of the art [44] on the *ImageNet* classification challenge [13]. To do this *AlexNet*, visualized in fig. 2.5, used 60 million parameters and 650,000 neurons. Training of the network was only made feasible by the use of multiple Graphics Processing Units (GPUs) [33].

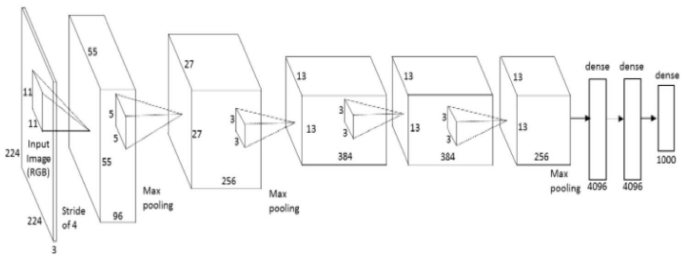


Figure 2.1: The architecture of *AlexNet* with its 60 million parameters distributed in eight layers, five convolutional and three fully connected. Figure from Krizhevsky, Sutskever, and Hinton [33].

The following sections will start with a summary of the work that has been done on extending CNNs for semantic segmentation and related problems like object detection. This will be followed by a dis-

cussion about how these networks are not suitable for mobile applications. Finally, approaches for compressing and building networks that can be run on mobile devices will be presented.

2.1.1 Semantic segmentation

In the areas of object detection and semantic segmentation *Regions with CNN features* (*R-CNN*) [16] first showed in that CNNs can be successfully be applied to these fields by significantly improving over the previous state of the art in object detection [41] 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* [49] 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, this pipeline is illustrated in fig. 2.2.

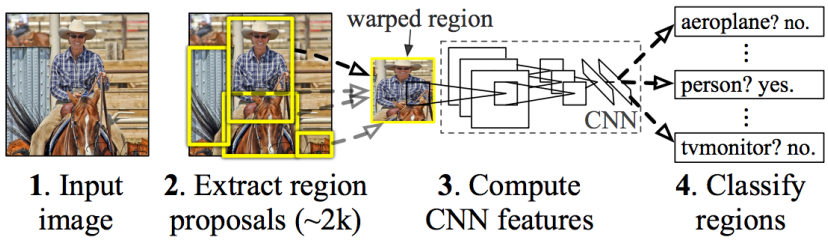


Figure 2.2: The object detection pipeline of *R-CNN* which illustrates the region proposals from selective search, the computation of CNN features from these regions and the final SVM classification. Figure from Girshick et al. [16].

Some issues with *R-CNN* are that it requires multistage training, training the CNN to give good features and the SVMs for classification, which is slow. Not only 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]. In this work, the network isn't run once per proposed region but instead once for the entire image 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 fed into a fully connected

neural network with two sibling output layers that perform both classification and bounding box refinement in parallel, this architecture is illustrated in fig. 2.3. With these improvements *Fast R-CNN* achieves faster inference and higher accuracy than its predecessors and does so with an arguably much more elegant design.

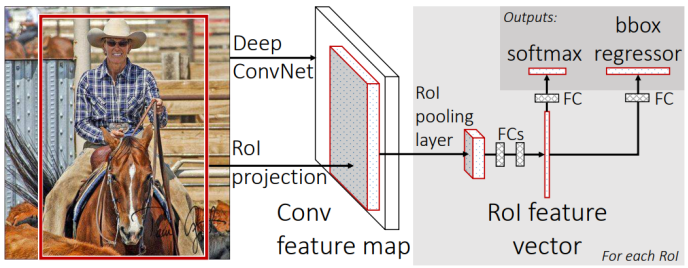


Figure 2.3: The architecture of *Fast R-CNN* illustrates how the convolutional feature map is only calculated once. Figure from Girshick [15].

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. These are fully convolutional neural networks that take as input the convolutional feature maps introduced in *Fast R-CNN*[15] and outputs region proposals. Since this approach for region proposals shares most of its computation with the classification network, as illustrated in fig. 2.4, the region proposals are practically free and frame rates of 5fps are achievable on a K40 GPU. The region proposal networks not only speed up computation but also prove to give better accuracy region proposals and thus raise overall accuracy in the system as well [40].

Even further improvements to this framework were 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.

Parallel work on semantic segmentation for medical imaging resulted in *U-Net* [43], a fully convolutional encoder-decoder network.

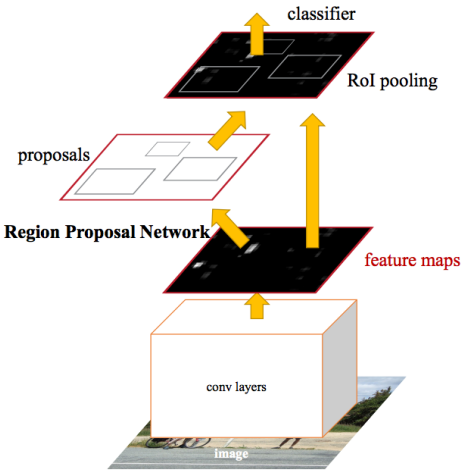


Figure 2.4: The architecture of *Faster R-CNN* shows how the feature maps are also used for region proposals. Figure from Ren et al. [40].

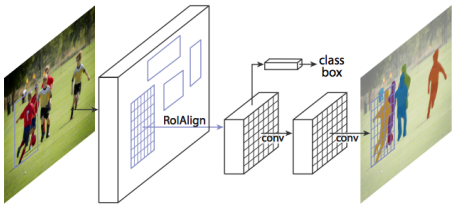


Figure 2.5: *Mask R-CNN* adds a branch for segmentation to the body of *Faster R-CNN*. Figure from He et al. [23].

Here the encoder projects the image into a lower dimensional feature space. 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 transpose convolutional layers. Further, the corresponding feature map of the downsampling path is concatenated to the upsampled features, thus preserving granularity of the images, see fig. 2.6. The final layer in the network is a softmax and hence the outputs are the probabilities of each pixel belonging to each class.

Further work with encoder-decoder structures for segmentation resulted in *SegNet* [3]. This network is architecturally very similar to *U-Net* with the major differences being how image granularity is preserved. Instead of concatenating the feature maps like they looked before max-pooling in the encoding branch to the upsampled feature

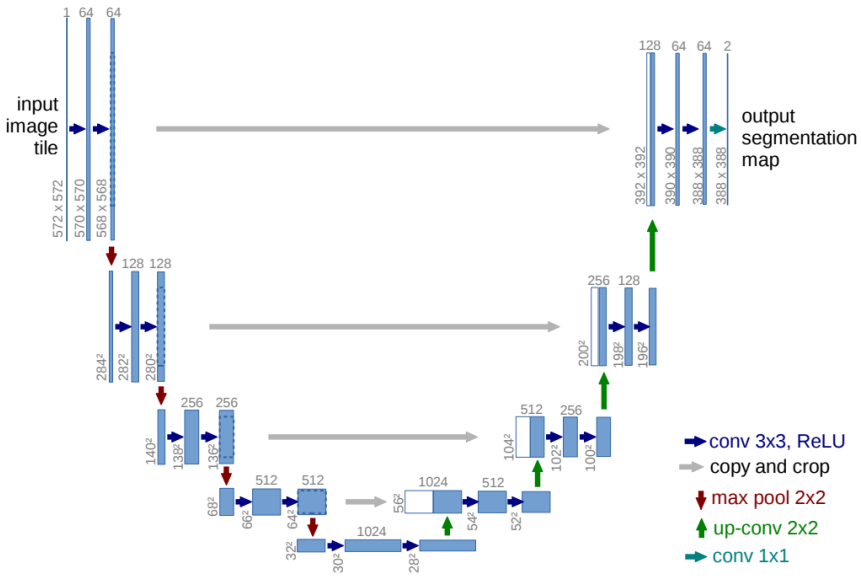


Figure 2.6: The U-shaped architecture of *U-Net* with skip connections for the feature maps in the middle. Figure from Ronneberger, Fischer, and Brox [43].

maps just the indices of which value was kept in max-pooling is stored in *SegNet*. This means that one integer, the index of the kept value, is stored instead of four floats, the values themselves have to be stored for each cell that max-pool is run over. This reduces the memory footprint of the network significantly.

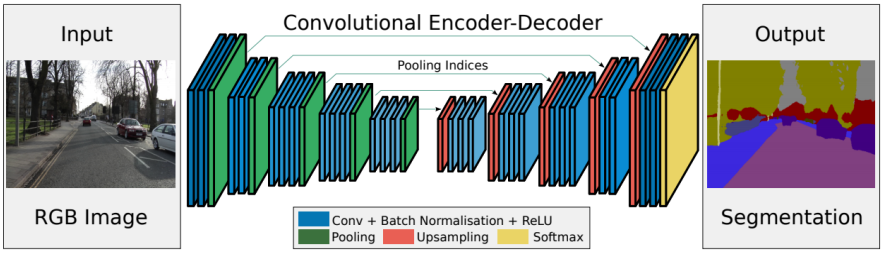


Figure 2.7: The architecture of *SegNet* illustrating the symmetrical encoder-decoder structure. Figure from Badrinarayanan, Handa, and Cipolla [3].

Continued work on segmentation utilizes blocks of so-called *DenseNets* [26], CNNs where every layer is connected to every layer after it, see fig. 2.8, enabling the training of exceedingly deep network architec-

tures by alleviating the vanishing gradient problem and promoting feature reuse between the layers.

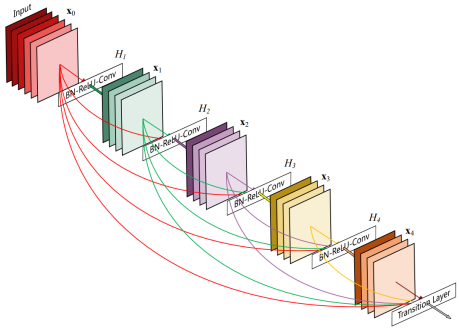


Figure 2.8: An example of a *DenseNet* which illustrates how all the layers are connected. Figure from Huang et al. [26].

Jégou et al. [30] use these *DenseNets* in a very deep encoder-decoder structure where skip connections, as seen in fig. 2.9, restore image granularity during upsampling the state of the art in image segmentation has been pushed even further while still reducing the number of parameters required for the models by a factor 10 as compared to the previous state of the art.

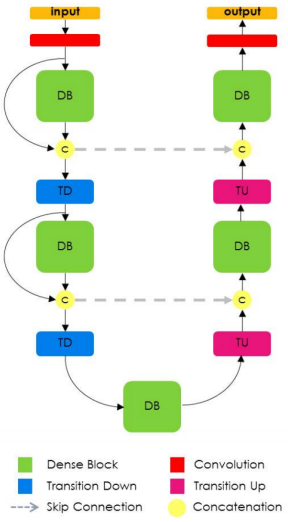


Figure 2.9: The architecture of *One Hundred Layers Tiramisu* where DenseNets are used for segmentation. Figure from Jégou et al. [30].

2.1.2 Network compression

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 modern networks for object segmentation [23] run at 5fps. That, however, is on high performance GPUs meaning that real-time performance on mobile devices is still largely intractable. 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 latency, 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 can 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 in the following sections.

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 [37]. 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 [52] 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.

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_{ij}^l 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

$$\frac{\partial \mathcal{L}}{\partial w_k^l} = \sum_{ij} \frac{\partial \mathcal{L}}{\partial V_{ij}^l} \frac{\partial V_{ij}^l}{\partial w_k^l}$$

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 [51] is used to reduce the storage required for the weights. This process is illustrated in fig. 2.10 and it 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 can 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.

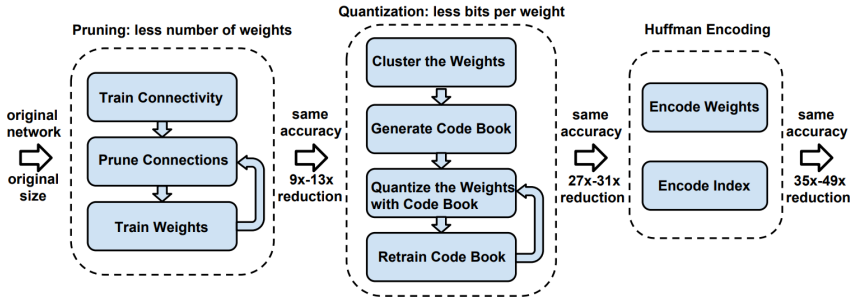


Figure 2.10: The compression pipeline of *deep compression*. Figure from Han, Mao, and Dally [19].

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 network's 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

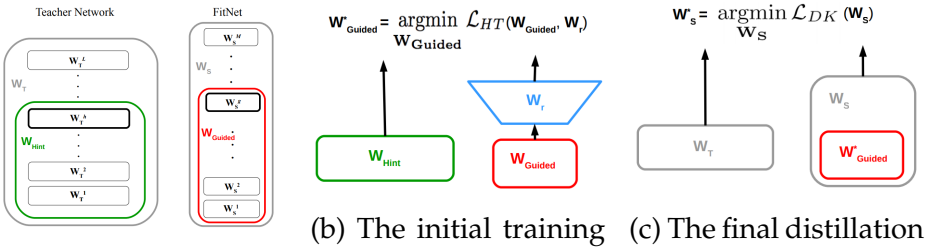
their ensemble teachers with a negligible drop in accuracy on some test problems [6].

Further work on student-teacher learning investigates why slow to compute, deep neural networks perform better than their shallower and faster siblings, even when the amount of parameters is the same between them. 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. Second, a bottleneck linear layer is added to the network to speed up training. With these modifications, the authors were 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 [50]. 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 [47] 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].

Distillation 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 transfer-

ring 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.

FitNets Continued work led 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. The training procedure is illustrated in fig. 2.11.



(a) The student and teacher networks

(b) The initial training of the guided layers in the student.

(c) The final distillation training of the entire student.

Figure 2.11: The training procedure for *FitNets*. Figures from Romero et al. [42].

Attention transfer Some more recent work [45] 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 in *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.

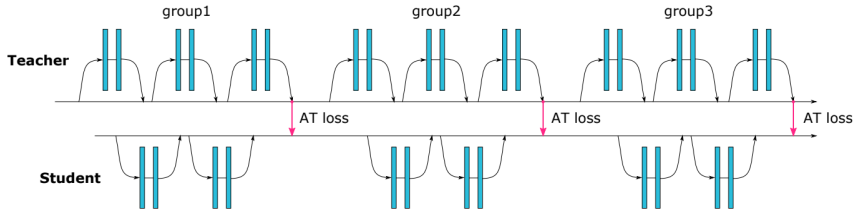
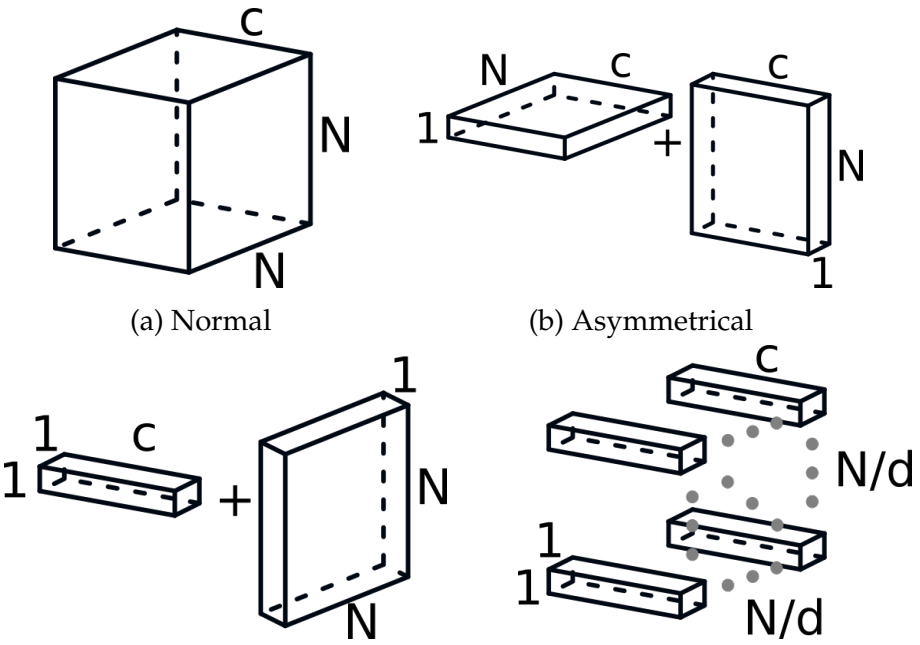


Figure 2.12: Illustration of *Attention transfer* showing the tight bond between teacher and student during training. Figure from Sergey Zagoruyko [45].

Optimized convolutions

Another orthogonal approach for compression is to optimize the convolutional layers themselves making them require fewer 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 [48]. Other variations on the convolutional operator that help compress the networks are dilated convolutions [53] 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 [46] and has since seen been successfully used in *Inception* models [29].



(a) Normal (b) Asymmetrical

(c) Depthwise separable (d) Dilated

Figure 2.13: Visualization of the shapes of the weight matrices to produce one feature map for each of the types of convolution. All in the case of a $N \times N$ filter with c channels. The dilation factor d corresponds to how sparsely the dilated convolutions are applied.

Streamlined architectures

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 number of channels that go into 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 especially suited for mobile vision applications. In doing so *MobileNets* also introduce two hyper-parameters, *width-multiplier* and *resolution-multiplier* that help design models with an optimal tradeoff 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 paid 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.

Another 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. To maintain image granularity skip connections are used, these can be seen in fig. 2.14.

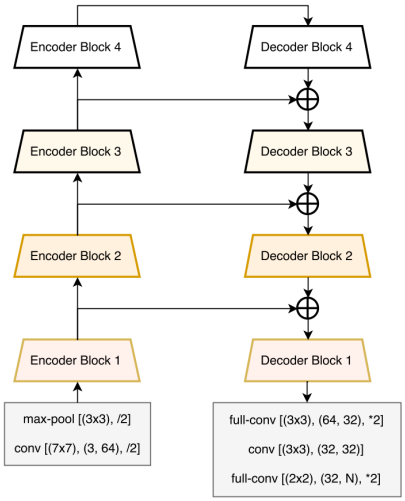


Figure 2.14: Architecture of *LinkNet*. Figure from Chaurasia and Culurciello [9].

Chapter 3

Method

In this project, we will try and build a model for real-time segmentation of feet on a smartphone. Since a system that is deployable right now is required, compression approaches like deep compression that need custom hardware to be fully utilized are out of the question. Instead, focus will be placed on how the highly streamlined architectures like MobileNets and LinkNet can be used and modified to produce give the required performance. The effectiveness of student-teacher learning in these models will also be evaluated by training the networks using knowledge distillation from a big segmentation network previously designed at Volumental. Here distillation is preferred over more recent approaches like FitNets or attention transfer since the available teacher was not specifically designed to a teacher and finding layers suitable for hinting or transferring attention maps from is hence a problem in itself and outside the scope of this work.

3.1 Pipeline

To train models for segmentation datasets with known segmentations are used. The images are augmented to artificially expand the available training data and the augmented image is feed into the model. The resulting predicted segmentation is compared to the ground truth and from this, a loss is calculated using which the parameters in the models can be updated. This pipeline is presented in fig. 3.1. The datasets used in this project along with the data augmentation used and the different loss functions and CNN models tested will be presented in the following sections.

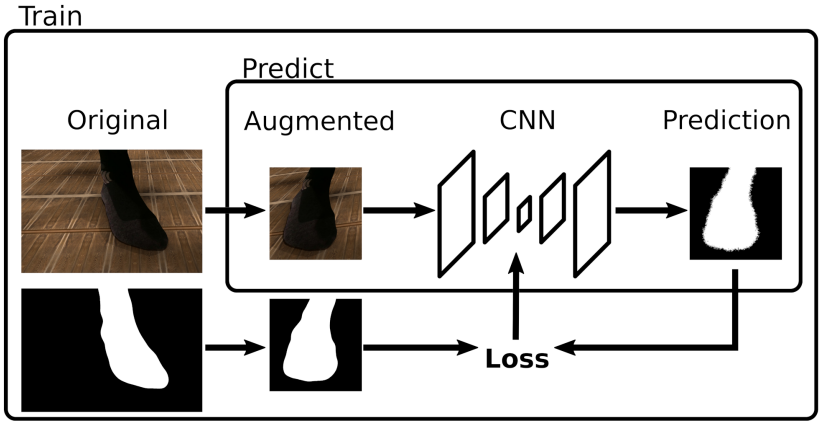


Figure 3.1: The data pipeline for training and predicting with the model.

3.2 Data

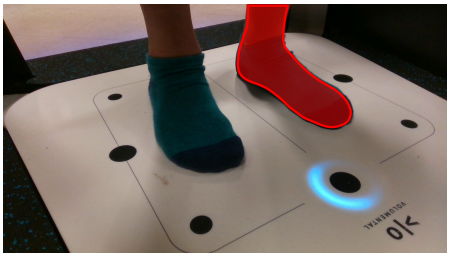
Two datasets of segmented feet were used for the experiments.

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. 3.2. 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. 3.3. Here the training set is directly used for training, the validation set is used for hyperparameter optimizations and to track overfitting during training, and finally, the held out test set is used for evaluating the performance of the models after training.

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. 3.4.

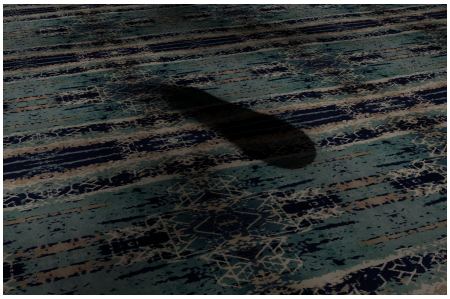
3.2.1 Data augmentation

One restriction with the *synthetic* dataset is that, due to the placement of the cameras, the images from the scanner only come from four dif-

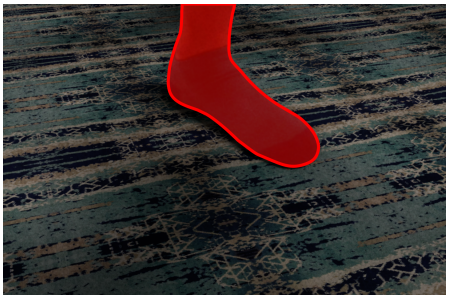


(a) The foot on the scanner

(b) Floor texture



(c) Texture in scanner plane with added shadows



(d) The final image with segmentation

Figure 3.2: Steps for creating the synthetic data

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

ferent angles. Fixed with regards to the foot and all taken from approximately the same distance. To enable the models 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 $256 \times 256px$ 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. 3.5

Figure 3.4: Examples of images and segmentations from the real dataset

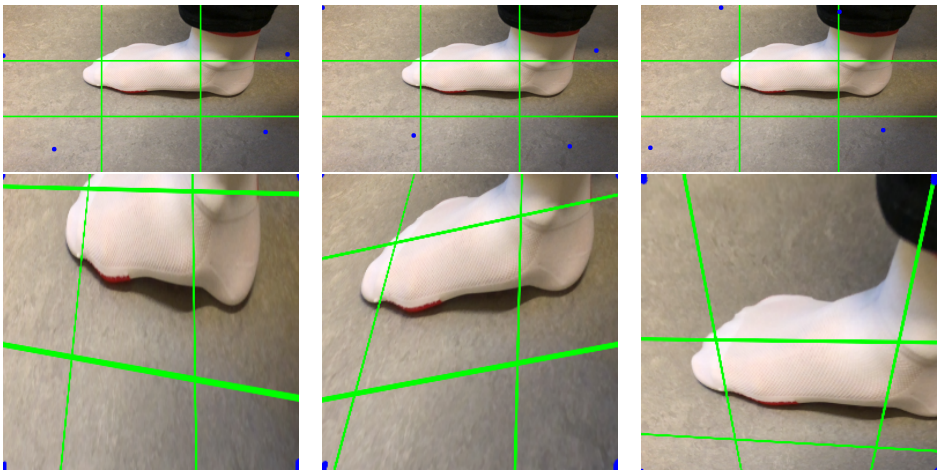


Figure 3.5: Some random augmentations on the same image. Top row is the original image with the sampled corners are indicated in blue and the sample boundaries are indicated in green. Bottom row is the resulting image.

3.3 Network architectures

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

ENet

This is a re-implementation of the *ENet* architecture [38] where the unpooling operations from the upsampling blocks have been replaced with addition of the corresponding feature map from the downsampling path due to restrictions in *Keras*. An illustration of what this means can be seen in fig. 3.6.

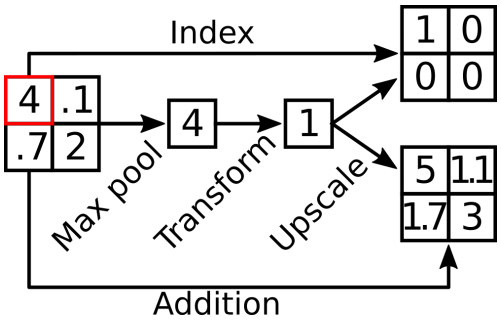


Figure 3.6: A simple illustration of the difference between using the indices from max pooling (upper branch) and to add the feature map as it looked before max pooling (lower branch) to maintain image granularity during upsamlpling.

LinkNet

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

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 skip-connections are introduced where the dimensions correspond to those in *LinkNet*

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 bilinearly upscaling the image to the full input size at the end.

3.4 Loss functions

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

3.4.1 Cross entropy

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

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

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.

3.4.2 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. 3.7.

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 an image with a lot of background and a tiny foot as all background would give good results.

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

$$\mathcal{L}_{IoU} = 1 - IoU$$

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:

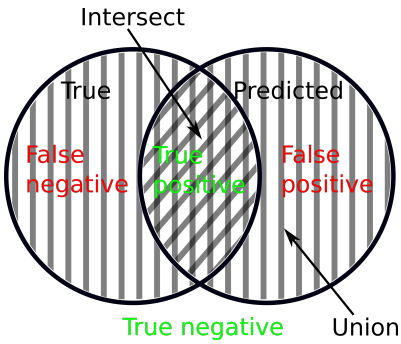


Figure 3.7: Notation for the relations between prediction and ground truth.

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

Where $\odot$ represents the element-wise Hadamard product.
The IoU loss hence becomes

$$\mathcal{L}_{IoU} = 1 - \frac{I(\hat{Y})}{U(\hat{Y})}$$

3.4.3 Distillation

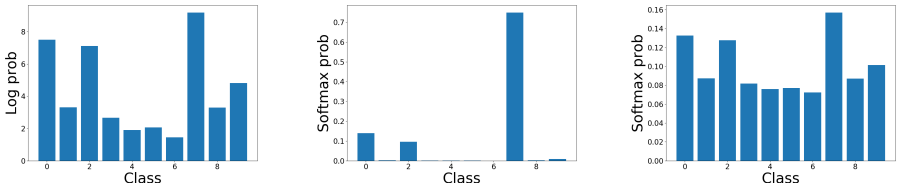
When training using distillation loss [24] a large previously trained network is used to help guide the student model during training. The loss function here is defined as:

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

Where $\mathcal{L}_{CE}$ is the cross-entropy loss from section 3.4.1 and $\hat{Y}^*$ and $\hat{Y}^*_{teacher}$ are the softened, high temperature softmax outputs, see fig. 3.8 from the student and teacher networks respectively. λ is the mixing factor controlling how much influence should come from the hard ground truth and how much should come from the soft student-teacher training. 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.



(a) Log-likelihood (b) Normal softmax (c) Softmax, $T = 10$

Figure 3.8: 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 $T = 10$

Chapter 4

Experiment

To test the suitability of the different models and loss functions presented in chapter 3 they were trained and evaluated. The training procedure and the different performance metrics used will be presented in the following sections.

4.1 Training

The networks were defined and trained in *Keras* [11] using the *tensorflow* [1] backend. This framework enabled easy experimentation and fast execution on the GPUs available, a *GeForce GTX 1080 Ti* and a *GeForce GTX TITAN*. For training the networks the *ADAM* optimizer[32], an adaptive extension to stochastic gradient descent that has become very popular in the deep learning community, was used. Training was run for 50 epochs each consisting of 50 steps with a batch size of 32. The training partition of the synthetic dataset is used for training and the loss on the validation set is monitored and the learning rate reduced by a factor 10 if validation doesn't decrease for 3 epochs.

In this training, a bunch of hyperparameters have to be chosen beforehand. To determine good values for some of the most important of these, namely learning rate λ, the dropout probability $p_{dropout}$, the loss function $\mathcal{L}$, and the parameters of distillation loss temperature T and mixing factor λ hyperparameter optimization is used. This means that the entire training procedure was run 100 times, each time with the hyperparameters randomly sampled from search space in table 4.1 and the performance on the validation set is monitored for these runs. This means that we can find the settings that give the optimal performance.

Table 4.1: The search space used in hyperparameter optimization. The temperature T and the mixing factor λ are only relevant when the loss function $\mathcal{L} = \mathcal{L}_{distiallation}$.

Hyper parameter	Search space
Learning rate η	$0.000001 < \eta < 0.01$
Dropout $p_{dropout}$	$0 < p_{dropout} < 1$
Loss function $\mathcal{L}$	$\mathcal{L} \in \{L_{CE}, L_{IoU}, L_{distillation}\}$
Temperature T	$1 < T < 100$
Mixing factor λ	$0 < \lambda < 10$

Hyperparameter optimization was facilitated by the python package *hyperopt* [4].

4.2 Evaluation

When evaluating the models two major things are of interest for this work. Firstly, how accurate the models are, how well they perform the prediction task. Secondly, How efficient the models are, how well they can be run on mobile hardware. A good balance between these two is necessary, the models are useless if they give perfect segmentations but takes seconds to do so for a single frame or can't fit on a smartphone. Likewise, models that can be reliably run at 20 fps but that give results comparable to random are also useless. The metrics used to quantify how well the models perform both of these tasks will be presented in the following sections.

4.2.1 Accuracy measures

To compare the predicted segmentations from the models to the ground truth data and quantify how well they correspond a lot of different accuracy measures can be used. A few of these different measures are presented in the following sections. For the experiments, these different measures have been calculated on both the held out test set of the synthetic dataset and the dataset of real images.

The accuracy metrics are defined for a test set of M images, each with C classes. The notation of true positive *TP*, true negative *TN*, false positive *FP*, and false negative *FN* is the same one used in fig. 3.7.

To index the sets the upper index is used for the class and the lower one for the image, hence TP_i^j is the true positive in the j:th class of the i:th image. It is noteworthy that only two classes are used in this project, *foot* and *not foot*.

Mean pixel accuracy

A simple metric for computing the similarity between a predicted segmentation and the ground truth is to just take the average accuracy over all the pixels in all the images. Mathematically this becomes:

$$mA = \frac{1}{M}\sum_{i}^{M}\sum_{j}^{C} TP_i^j$$

Mean IoU

As discussed in section 3.4.2 just looking at correctly classified pixels is problematic in that large classes can overpower the results and make a rather poor prediction look good on paper. To get around this we can look at the intersect over union metric

$$IoU_i^j = \frac{TP_i^j}{FN_i^j + TP_i^j + FP_i^j}$$

which if we average over all classes and images looks like this:

$$mIoU = \frac{1}{MC}\sum_{i}^{M}\sum_{j}^{C} IoU_i^j = \frac{1}{MC}\sum_{i}^{M}\sum_{j}^{C} \frac{TP_i^j}{FN_i^j + TP_i^j + FP_i^j}$$

Mean Dice

A very similar metric to IoU is *Dice* or as it is also known the $F1$ metric. It is calculated as follows.

$$mDice = \frac{1}{MC}\sum_{i}^{M}\sum_{j}^{C} \frac{2TP_i^j}{FN_i^j + 2TP_i^j + FP_i^j}$$

Mean average precision

If we set a threshold x and say that predictions with a $IoU \geq x$ are correctly classified. We can then look at the proportion of correct clas-

sifications for different such thresholds.

$$mAP_x = \frac{1}{MC} \sum_{i}^{M} \sum_{j}^{C} \begin{cases} 1 : IoU_i^j \geq x \\ 0 : IoU_i^j < x \end{cases}$$

4.2.2 Efficiency measures

To quantify how efficient the models are to store and compute the following metrics were used.

Storage space

The size of the model, in full float 32 representation was measured in megabytes (MB). This is then used as a measure of how well the models could fit into the cache of mobile processors and for how expensive the models are to store and transfer.

FLOPs

The amount of floating point operations (FLOPs) required for inferring a single frame of $256 \times 256px$ using the models were calculated using the built-in benchmarking tools in *tensorflow*. A FLOP corresponds to a single multiplication, addition or similar between two floating point numbers. This means that the amount of FLOPs is not dependent on the hardware the model is run on but just on the model itself. However, due to optimizations, both at the framework and processor levels two models with the same amount of FLOPs may take different times to compute.

Inference time

To get an accurate measure of how the long the models actually took to compute on mobile hardware the time required to infer a single $256 \times 256px$ frame was measured on a phone. The phone used was a ZTE Axon 7 android phone and timing was facilitated by the benchmarking tools in *tensorflow*.

Chapter 5

Results

The efficiency metrics evaluated on the different models can be seen in table 5.1. The resulting hyperparameters from hyperparameter optimization can be seen in table 5.2. Interesting to note here is that two smaller models, ENet and FastLinkNet, require far less regularization trough dropout, this is expected since it is harder to overfit with fewer parameters. Plots of the training and validation loss during training of the final models can be seen in fig. 5.1. The fact that training and validation loss stabilize to almost the same value for all of the models indicates that almost no overfitting has occurred.

The accuracy metrics introduced in section 4.2.1 were calculated for both the held out test set from the synthetic dataset and the real dataset, results from this can be seen in table 5.3 and table 5.4. Plots of the mean average precision metric (mAP_x) evaluated at a range of different threshold values x can also be seen in fig. 5.2

Table 5.1: Comparison of size, FLOPs and time required for inference of the different models. Timing was done on a *ZTE Axon 7* android smartphone.

Model	GFLOPs	Inference time [ms]	Size [MB]
EmilSeg	55.37	4439	85
ENet	1.11	366	**1.8**
LinkNet	1.75	234	45
MobileSeg	2.68	555	18
FastLinkNet	**0.287**	**73**	7.2

Table 5.2: The resulting hyperparameters from hyperparameter optimization. *N/A* means that those hyperparameters don't exist for the selected loss function.

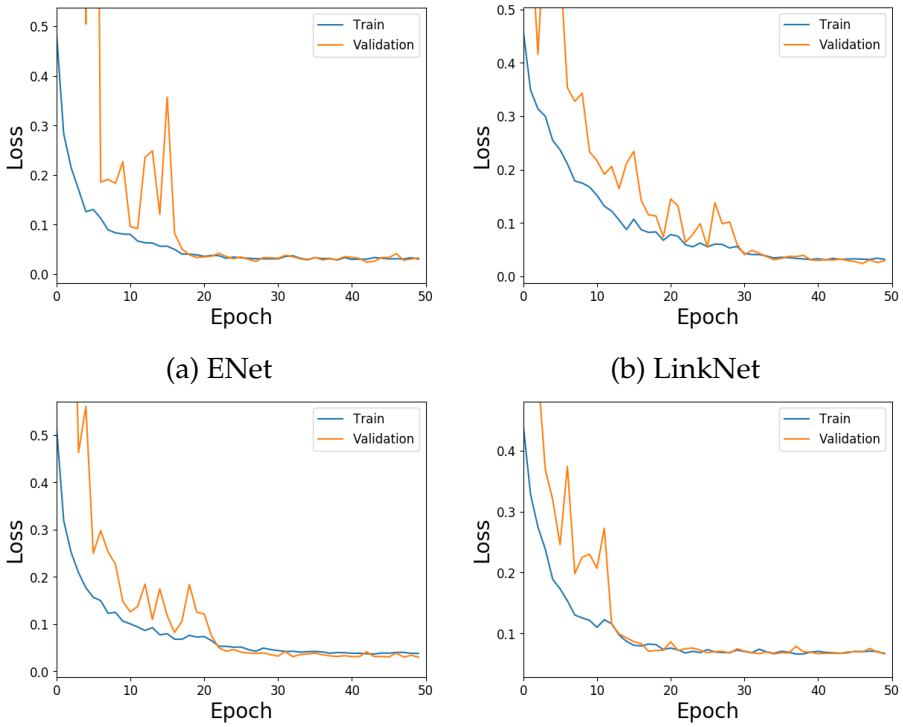
Model	η	$p_{dropout}$	$\mathcal{L}$	λ	T
ENet	0.006164	0.08435	$\mathcal{L}_{CE}$	*N/A*	*N/A*
LinkNet	0.00837	0.321	$\mathcal{L}_{IoU}$	*N/A*	*N/A*
MobileSeg	0.002642	0.2852	$\mathcal{L}_{CE}$	*N/A*	*N/A*
FastLinkNet	0.005706	0.04245	$\mathcal{L}_{IoU}$	*N/A*	*N/A*

Table 5.3: The different accuracy metrics evaluated on the models using the held out test set of the synthetic data.

Model	*mA*	$mIoU$	*mDice*
EmilSeg	**0.9925**	**0.9842**	**0.9914**
ENet	0.9877	0.9738	0.9858
LinkNet	0.9847	0.9677	0.9821
MobileSeg	0.9883	0.9751	0.9866
FastLinkNet	0.9689	0.9357	0.9641

Table 5.4: The different accuracy metrics evaluated on the models using the real dataset.

Model	*mA*	$mIoU$	*mDice*
EmilSeg	0.9742	0.9486	0.9722
ENet	0.9609	0.9250	0.9587
LinkNet	0.9764	0.9535	0.9750
MobileSeg	**0.9778**	**0.9552**	**0.9764**
FastLinkNet	0.9670	0.9348	0.9652



(a) ENet (b) LinkNet

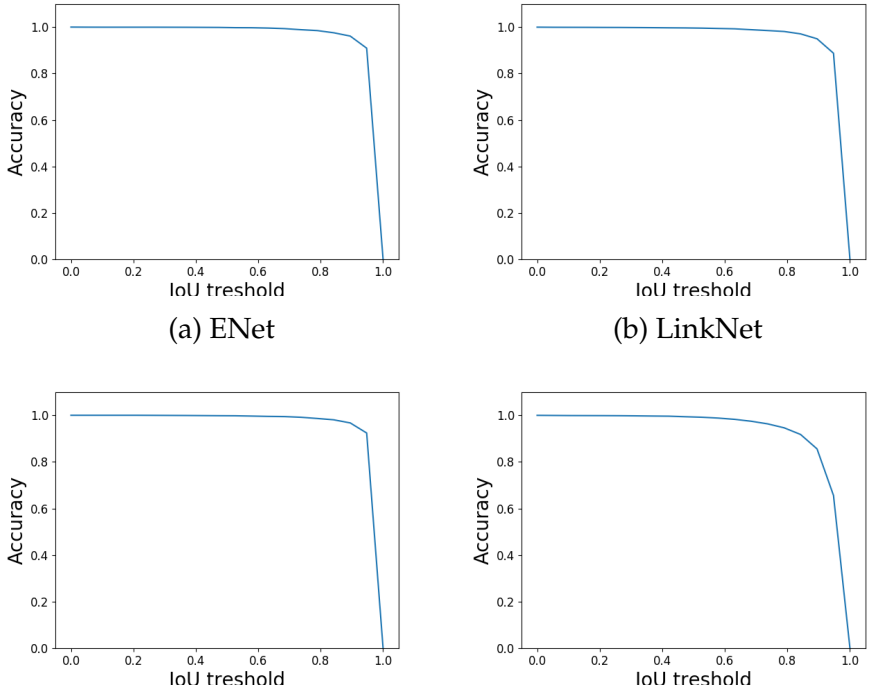
(c) MobileSeg (d) FastLinkNet

Figure 5.1: Loss in train and validation sets during training of the final models.

Images of the resulting segmentations for all the models on the synthetic and the real datasets can be seen in fig. 5.3 and fig. 5.4 respectively.

To illustrate the tradeoff between inference speed and accuracy that lies in the choice of model the accuracy is plotted vs inference speed for the different models in fig. 5.5.

To compare the different loss functions used for training, histograms over the achieved validation accuracies during hyperparameter optimization of FastLinkNet is displayed in fig. 5.6. There is one his histogram per loss function.



(a) ENet (b) LinkNet

(c) MobileSeg (d) FastLinkNet

Figure 5.2: The mean average precision mAP_x evaluated at a range of threshold values x for each of the trained models.

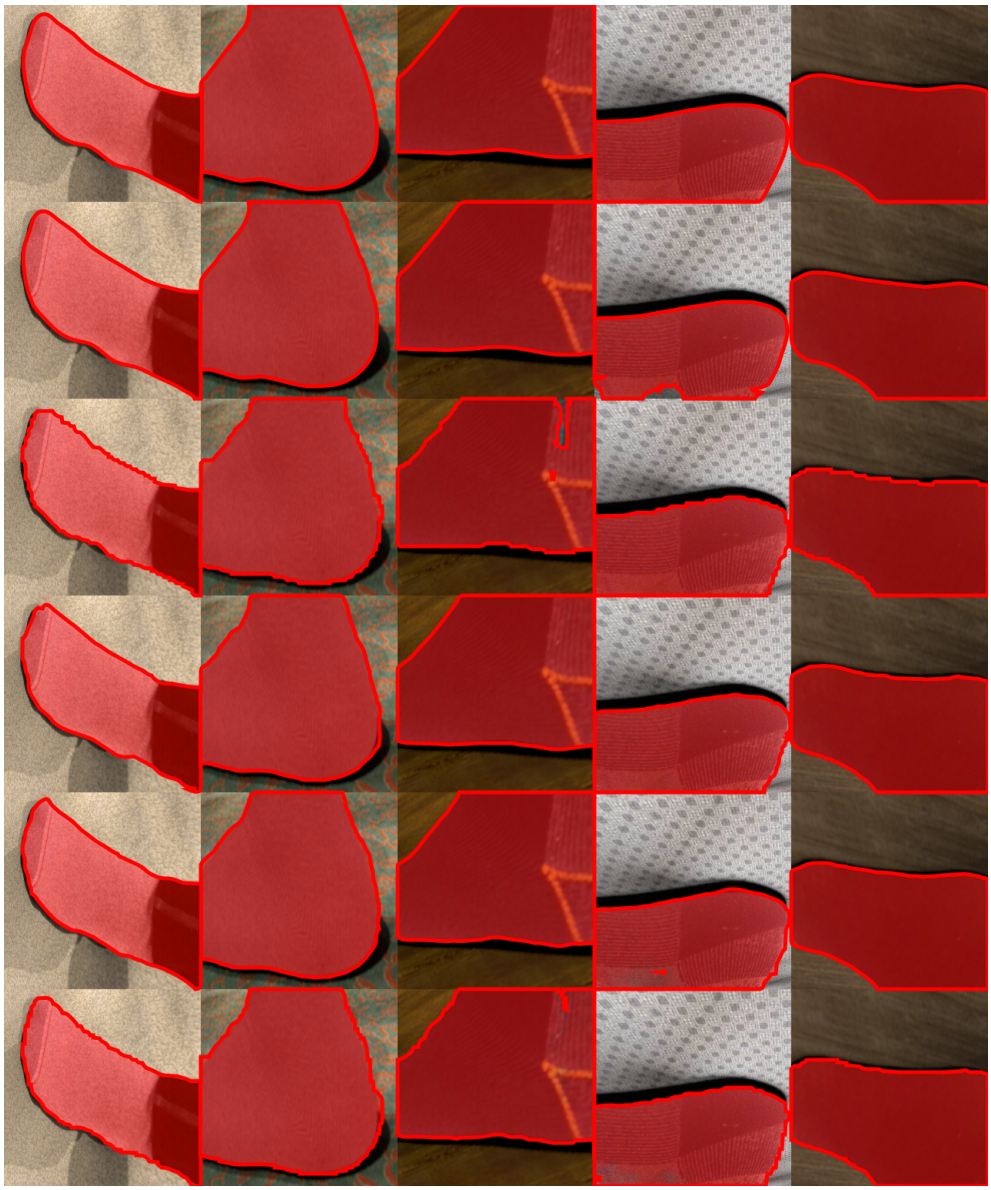


Figure 5.3: Resulting segmentations on the test set of the synthetic dataset from the different models. First row is the ground truth, below that EmilSeg, ENet, LinkNet, MobileSeg, and FastLinkNet.

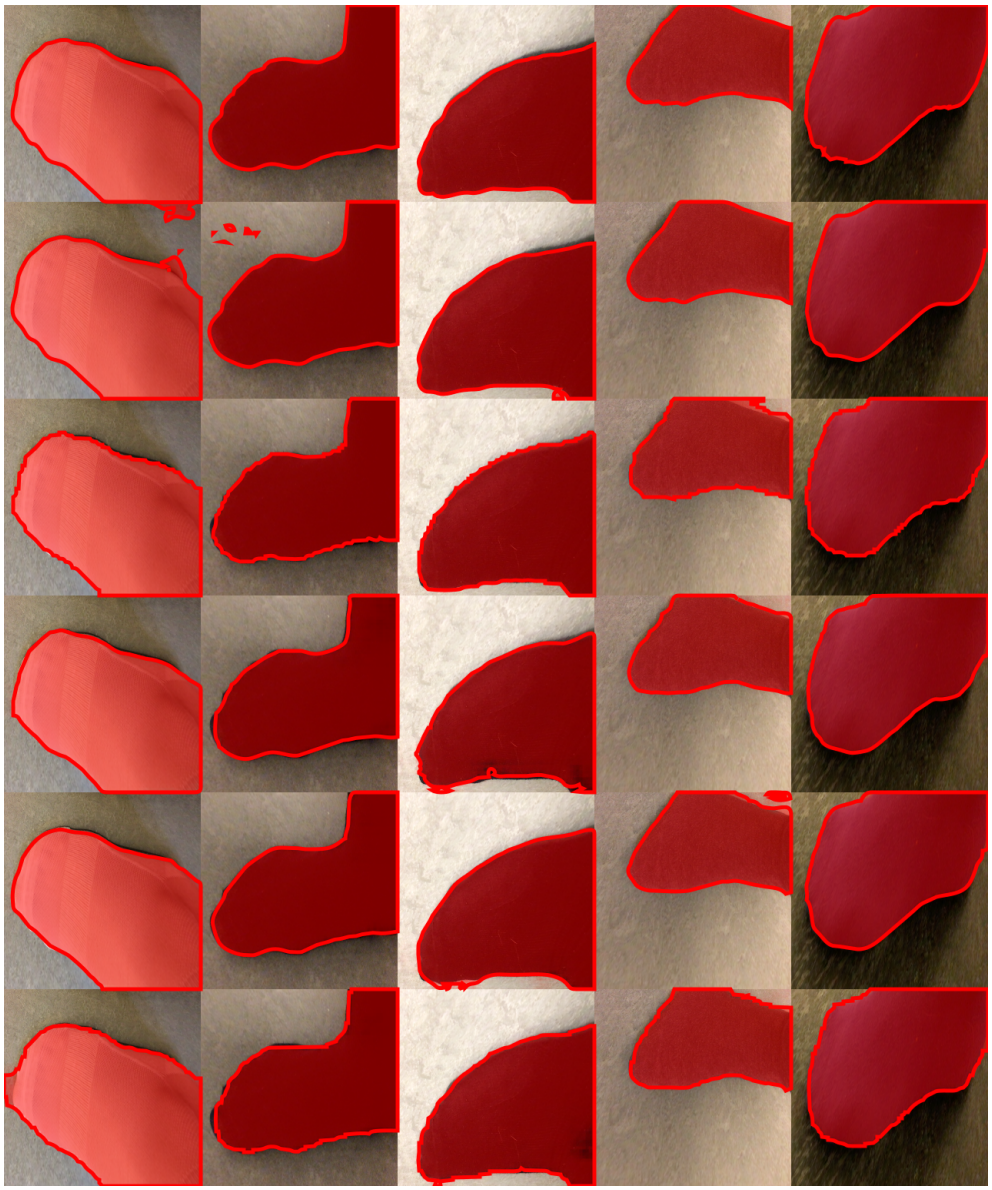


Figure 5.4: Resulting segmentations on the real dataset from the different models. First row is the ground truth, below that EmilSeg, ENet, LinkNet, MobileSeg, and FastLinkNet.

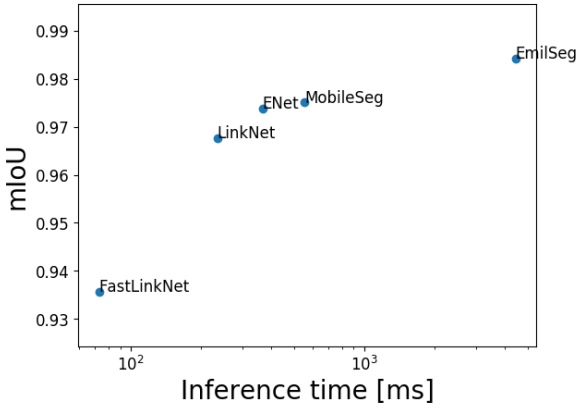


Figure 5.5: Mean IoU on test set vs inference speed for the different models. Note that the time axis is logarithmic.

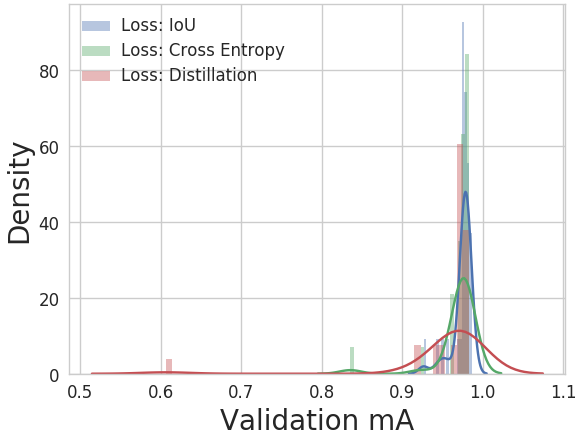


Figure 5.6: Histograms over the mean accuracy on the validation set for all the hyper optimization rounds on FastLinkNet. To give a sense for the performance of the different loss functions there is one histogram per loss function.

Chapter 6

Discussion

If we look at fig. 5.5 it is apparent that quicker inference comes at the cost of reduced prediction quality on the test set. However this loss of 5% (table 5.3) is rather small compared to the speed gain of over a factor 60 (table 5.1) when comparing EmilSeg to FastLinkNet. These results indicate that most of the performance in large neural networks like EmilSeg can successfully be represented in way smaller and quicker networks.

A closer look at the fastest network, FastLinkNet, gives that it with its $73ms$ inference time can run at over $10fps$ even with some overhead for loading images and other processing. This is even more noteworthy if we consider that these benchmarks were run on a smartphone from 2016 and that current mobile machine learning frameworks can't leverage the graphical processing units (GPUs) and digital signal processors (DSPs) in the devices. Getting these specialized processors working are expected features as the mobile machine learning frameworks mature which means that further performance boosts may come soon. As it stands now $10fps$ can almost be considered realtime for many applications and the networks meager size of $7.2MB$ makes it easy to package into apps and to cache.

A comparison between the performance on the test set of the synthetic data, table 5.3, and the data set of real images, table 5.4, shows that even though there is a degradation in performance when testing on real images it is quite small. This small difference means that generalization from the synthetic to the real data works well. It is interesting to note however that the difference between the models shrinks significantly when we look at the real data. This might imply that some of

the edge the bigger models have over the smaller ones is overfitting to the synthetic data.

In fig. 5.6 a comparison between the different loss functions used in the project is made by looking at the performance on the validation set of all the versions of FastLinkNet trained during hyperparameter optimization and sorting them by loss function. It can be seen that cross entropy and IoU loss seems to perform rather similarly with IoU being slightly ahead. Surprisingly though distillation performs the worst which might be due to the teacher model not being powerful enough or the search space, table 4.1, being bigger for this loss. Both of these possible explanations highlight an interesting drawback with student-teacher training however, there is a lot of added complexity and that makes for more places where it can go wrong.

Another thing to note are the remarkably high scores, both on the test set and the real dataset as seen in table 5.3 and table 5.4 and backed up by the qualitatively very good segmentations in fig. 5.4 and fig. 5.3. This performance might not transfer all that well to real-world applications however since the images in the dataset are all taken from similar angles and distances, in good lighting on uncluttered floors, and always contain a single foot. These restrictions in the dataset may make it difficult for the models to cope with more complicated real-world scenarios.

6.1 Further work

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 tweaking not only the students but also the teacher.

Further performance could also be gained from not just using the current frame for segmentation but the entire stream of information. This could, for example, be done by adding a long short-term memory (LSTM) cell to the bottlenecks of the segmentation networks or by feeding the predicted segmentation at each frame back as a fourth channel in the input for the next frame. These approaches, however, would require segmented video data for training or heavy augmentation of the existing images to produce plausible fake video from still

images. Another issue with these approaches is also that they can add a lot of computational overhead to the models and hence slow down execution to an unacceptable degree.

An approach to getting better and more reliable results from the networks that would not slow down execution would be to rethink the training process. For example pre-training the encoder on *ImageNet* or the entire network on another segmentation dataset like *CamVid* could improve performance. To get more plausible segmentations from the networks they could also be trained as a generative adversarial network (GAN) as proposed by Luc et al. [36].

The quality of the synthetic dataset could also be improved, either by training a *GAN* to produce the training images or by using the full 3D-models of the feet from the scanner, mapping the texture from the images back to this model and then placing the foot at an arbitrary angle and distance from the virtual camera. Giving a greater variance in the data.

6.2 Impact

When handling data like images of feet in this fashion the question of privacy is always relevant and being careful about anonymizing data is important. Further, the aim of this work is to automate away sales clerks and instead give that experience online. This overall trend of automating away work is bound to make some people unemployed and a serious discussion about how this situation about how this unemployability is to be handled in society is well overdue. However, an online shopping experience where a good fit is acquired at first try is going to reduce returns and in turn shipping and the carbon footprint of online retail overall.

On a more technical level, the same methods that help reduce the power required and computational resources needed for running neural networks on smartphones are being applied to larger systems as well. This reduces the energy consumption of machine learning models that are becoming more and more prominent in our lives and helps reduce the power requirements of the industry overall.

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] Pauline Luc et al. "Semantic Segmentation using Adversarial Networks". In: *CoRR* abs/1611.08408 (2016). arXiv: 1611.08408. URL: http://arxiv.org/abs/1611.08408.

[37] 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.

[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] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[44] 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.

[45] 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.

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

[47] 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.

[48] 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.

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

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

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

[52] 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.

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