The Pennsylvania State University

The Graduate School

College of Engineering

ADAPTIVE NEURAL NETWORK ARCHITECTURES FOR POWER AWARE INFERENCE

A Thesis in

Computer Science and Engineering

by

Skyler Anderson

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2019

The thesis of Skyler Anderson was reviewed and approved* by the following:

Vijaykrishnan Narayanan
Distinguished Professor of Computer Science and Engineering
Thesis Advisor

John Sampson
Assistant Professor of Computer Science and Engineering

Chitaranjan Das
Distinguished Professor of Computer Science and Engineering
Department Head of Computer Science and Engineering

*Signatures are on file in the Graduate School

ABSTRACT

As Neural Networks are deployed in edge devices, design choices are often dictated by power and compute limitations despite variability in these factors. In this work we propose a novel method for adaptive neural network architectures to boost performance as power becomes available without necessitating a complete reconfiguration of model parameters. We show that due to the stochastic nature of neural networks, models can be constructed with enough independence to be effectively ensembled while sharing a common convolutional base. This allows for a trade-off between power consumption and model accuracy without redeploying an entirely new model. Our method is agnostic to the base network and can be added to existing networks with minimal retraining. We find this approach particularly well suited for IoT devices and distributed autonomous applications where available power and compute resources are time varying.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The success of this thesis is largely thanks to my adviser and committee member: Vijay Narayanan and Jack Sampson. Without their guidance and valuable feedback this work would not have found its direction. I would also like to thank my lab mates, Jinhang Choi and Peter Zientara, for helpful discussions. Finally, thanks to my supportive parents, Carrie and Jim Anderson, and lastly, Mo. This work was supported in part by the Semiconductor Research Corporation (SRC) Center for Brain-inspired Computing Enabling Autonomous Intelligence (C-BRIC). The views and findings expressed herein are solely those of the author and do not reflect the opinion or position of the sponsor.

Chapter 1

Introduction

As computers become more powerful, human-like capabilities have begun to emerge. In particular, deep learning has risen as a powerful method of solving both supervised and unsupervised learning problems and has set state of the art performance in several key areas **[1, 2, 3, 4].** In some cases, the computer performance enabled by deep learning exceeds that of their human counterpart **[5]**. Thus, we have reached an inflection point where computers can no longer be left out of important decisions, such as those involving life or death situations like medical image processing or autonomous driving, without potentially sacrificing on the quality of these important decisions. Like the human brain, however, neural networks exhibit many 'black box' properties such as lack of explainability and a theoretical understanding that leaves much to be desired. Therefore, the optimization of computational learning systems is itself a complex undertaking.

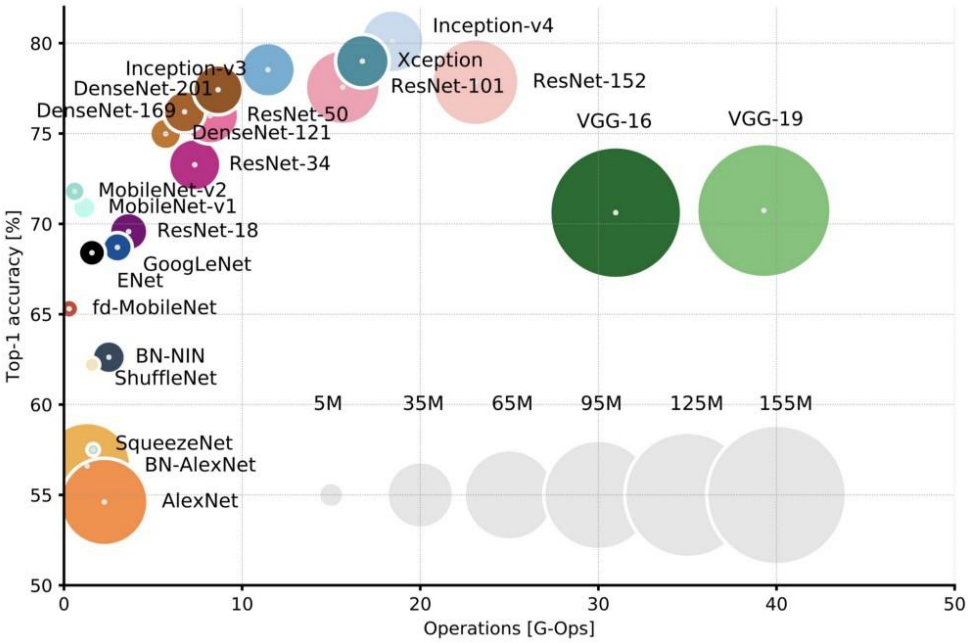

Figure 1. G-Ops vs accuracy for top classification networks. Figure taken from [6].

Among the problems that deep learning approaches have been applied, computer vision is one of the most heavily studied, optimized, and strongest success cases. Figure 1 shows the progression of classification accuracy as a function of operations. We see that there is not a linear relationship between accuracy and operations or model size. Additionally, because every model has a unique structure, it is difficult to directly compare networks or identify marginal complexity and accuracy tradeoffs. The highest performing models are highly performant but come at the cost of large computational requirements. On the other hand, more lean models that are well suited to IoT deployment and edge devices are not always capable of desirable accuracy. There is an infinite design space when it comes to network architectures. Each network is optimized, to the best of our abilities, to maximize performance within some given constraints. Those constraints could revolve around low power requirements, minimizing model footprint, or in other cases the goal may be maximizing accuracy at all costs. Nevertheless, it is not clear how to navigate this design space to optimize for accuracy with respect to these constraints due to the black box properties of neural networks.

Figure 1 is composed of classification networks but as many other vision tasks are an extension of classification, they use the same networks for the convolutional base. The model's classification performance is often used as a proxy for model capacity and translates closely to related tasks.

Embedded inference engines are gaining popularity as the networks they power surpass acceptability thresholds. As artificial intelligence proliferates in IoT devices and other power-varying systems the need for run-time-adaptive architectures is increasingly important. Modern applications of deep neural networks exist across multiple domains such as autonomous navigation, robotics, and autonomous drone swarms [7, 8]. Conversely, one could fit a network to the power budget for a given deployment scenario. These naive approaches to model choice are born out of necessity rather than optimality. In many scenarios the target accuracy is time varying. Despite this fact, a single model is used for inference everywhere.

Dynamically Constrained Inference Deployments

The standard method for employing these networks is to choose the smallest or least compute intensive network that achieves the maximum required accuracy for a given task. While there has been substantial prior work targeting specific design points in terms of accuracy and power constraints for inference, many real-world deployment scenarios are more fundamentally dynamic.

Unmanned aerial vehicles (UAV) are already being used to autonomously navigate uncertain environments [9] as well as for search-and-rescue operations in dangerous or broad search spaces. Performing these actions requires advanced computer vision and machine learning algorithms that must strike a balance between computational demand, latency, and accuracy. The hardware fitted to a network node is limited by thermal, power, and form constraints [8] which may make the use of a single node unsatisfactory. If it is known that more powerful inference

engines may be required an offload mechanism will be present. Typically, this includes transmitting the data to a cloud for server-side processing which incurs transmission costs and increased latency. Instead, as these drones are often deployed in fleets, we take advantage of the fact that the total local compute power varies with time. Designing a static algorithm that works as desired under all circumstances is impractical. Additionally, we show that it is unnecessary. As swarms of drones are deployed simultaneously they can perform collaborative inference [10] when every drone has a complete processing pipeline. Our work is more general and considers secondary drones as auxiliary compute units that need not carry complete hardware. We also note that drones are not the only application scenario. Energy harvesting IoT devices, solar powered devices, and more can similarly benefit from our proposed adaptive network.

Adaptive Neural Network Architectures for Power Aware Inference

Models can be compressed in size by reducing bit width [11] or their knowledge distilled into a smaller model [12]. Both techniques are applied during or after training. [13] dynamically terminates inference based on entropy at intermediate points in the network but is still a statically defined network. On the other hand, when there is no power constraint, models aimed at achieving state of the art performance can have hundreds of millions of parameters and require trillions of multiply-add (MAdd) operations [14, 15]. Our proposed network exists as an adaptive compromise between large, overprovisioned, networks and those that have been squeezed to their limits.

In this work we propose a novel methodology for improving arbitrary network accuracy without modifying the core network architecture or weights. As an independent component it can be instantiated on demand to effectively utilize dynamically available power. Our proposed method requires minimal retraining and is network agnostic. Our focus is on semantic image segmentation, but the technique can be trivially extended to related domains.

Chapter 2

Background and Related Work

Deep neural networks (DNNs) have become ubiquitous in a variety of machine learning tasks. DNNs have set state of the art performance on key benchmarks in several domains such as computer vision, natural language processing, and speech processing. Despite impressive performance they come at the cost of high computation and latency [16]. GPUs offer massive speedups by exploiting data level parallelism in network operations but are inflexible to run time modifications. There has been recent interest in high speed inference at the edge using FPGA neural accelerators that allow for partial reconfiguration with minimal impact on latency [17]. In this work we consider how this new-found capability can be used for power aware inference to improve model performance as more power or compute resources come online without modifying the convolutional base network.

Learning Systems and Computer Vision

Efficient Network Architectures

Networks used for semantic image segmentation are typically fully-convolutional, meaning they consist entirely of convolutional layers, optionally with pooling, and do not employ fully-connected layers. As with most image classification type tasks, these networks are often fine-tuned after being trained on a classification dataset where at least part of the objective (read: loss function) is shared with the final task. In the case of image segmentation, networks are commonly pretrained on ImageNet [18], a large-scale classification dataset consisting of over one million training images and one thousand object categories.

DeeplabV3+ currently holds the record on several leaderboards, including PASCAL VOC [19], for semantic image segmentation [14]. Like many segmentation networks it takes on the encoder-decoder type structure where a convolutional base network is used as the feature extractor, or encoder, and the features are then turned into output predictions by a decoder. Targeting power efficient network architectures, we begin with the variant of DeeplabV3+ [14] that uses MobileNetV2 as the convolutional base network.

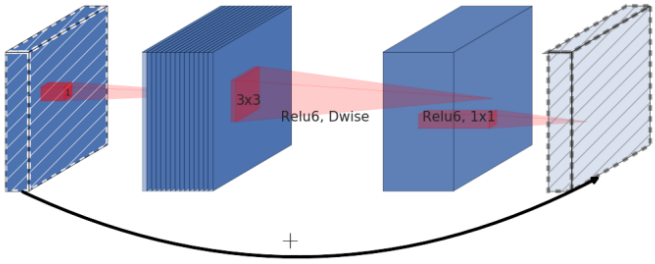

Figure 2. Inverted Residual Block. Figure taken from [20].

MobileNetV2 is built on the inverted residual block shown in Figure 2 borrowed from [20]. These residual blocks are an effective for performing high-dimensional filtering without incurring much of the expenses usually incurred by large convolutional operations. The inverted residual block begins with a 1x1 convolution to project incoming tensors to a higher dimensionality where spatial filtering is performed with depth wise convolution and then the tensor is projected back to a lower dimensionality with a pointwise convolution, completing the depth wise separable convolution [21]. There are several key benefits to these building blocks. Projecting to a high dimension before filtering increases network capacity by temporarily overprovisioning the transformation space while propagating tensors at compressed dimensionality reduces memory and computation footprint without decimating the activation manifold [3, 20].

Like MobileNet [20] is SqueezeNet [22]. SqueezeNet preserves accuracy while using fewer parameters by retaining spatial resolution in feature maps. All else equal, by not downsampling until late in the network, retaining large activation maps leads to higher accuracy [22]. Additionally, [22, 21, 23] have found success with liberal use of 1x1 convolutions for two reasons: it has 9x fewer parameters than a 3x3 convolution, and it can be an effective method for dimensionality reduction [22].

Depthwise Separable Convolution

Convolution is the workhorse of deep learning. It has proven to have a powerful ability to extract abstract features from high dimensional data and its linearity gives it nice optimization properties [2]. Depthwise separable convolution is a factorization of standard convolution into a depthwise convolution and a 1x1 pointwise convolution. Whereas a normal convolution takes computation cost of [20]:

$$D_k \bullet D_k \bullet M \bullet N \bullet D_F \bullet D_F$$

For an input tensor of shape $D_F \; x \; D_F \; x \; M$ with kernel of shape $D_k \; x \; D_k \; x \; M \; x \; N$, depthwise convolution has a computation cost of:

$$D_k \bullet D_k \bullet M \bullet D_F \bullet D_F$$

Which leads to a reduction in computation of

$$\frac{1}{N} + \frac{1}{D_k^2}$$

Depthwise separable convolution is not to be confused with spatially separable convolution commonly used in the image processing community.

Fully Convolutional Networks

Fully connected layers are a powerful tool for producing model output. Fully connected layers are equivalent to a convolution with kernel size equal to input tensor size. This gives each

neuron a global view of the incoming tensor as every element is considered, however it comes at a very high computation cost. In some networks [5, 15] the fully connected layers can account for most network weights despite being a small fraction of the total layers. Recently there has been a trend towards fully convolutional networks. Replacing the fully connected layers with a small kernel, HxWxDxD', where D' is the total number of classes, followed by a global average pooling approximates the effect of a classifier and produces remarkably similar results with a fraction of the computation [23, 3].

Static Optimization

Model Compression

Due to the size and memory requirements of neural networks several works have emerged on methods for reducing model footprint. [11] exploits linear structures that exist in 4 dimensional convolutional kernels by applying singular value decomposition (SVD) to the weights. [24] prunes connections whose strength doesn't exceed a threshold and fine tunes the remaining network. This method, when applied cautiously, can reduce model size by an order of magnitude without any loss in accuracy.

Knowledge distillation [12] has proven to be a powerful scheme for condensing the information encoded in a large, cumbersome, model into a smaller and more compact model. By training a smaller network on the smoothed logits of a cumbersome model the network is trained to learn complex interactions between classes and their relations that don't exist in one-hot labels. This way the knowledge extracted can be shifted from a large, teacher, model to a smaller, student, model.

Model Quantization

Binary weight networks and XOR networks have taken aim at eliminating expensive 32-bit convolutions that incur billions of multiply-accumulate operations with binary values computed using only XNOR and bitcount [25]. Similarly, because networks are often run at full 32-bit floating-point precision there has been research into the necessity of full precision and its effect on accuracy. There has been some success in moving to fixed point and mixed precision during inference without significant loss in accuracy [26].

Dynamic Optimization

Instead of optimizing prior to deployment, [13] has aimed at run-time improvements to model efficiency. Due to the inherent inequalities between images and even object categories, some inputs can be identified even before reaching the end of a network. By adding intermediate classifiers throughout a network and measuring the entropy, it is possible to terminate inference early and return a prediction with comparable accuracy to the full forward pass. Similarly, [27] dynamically routes features through a network to most effectively classify objects based on the level of ambiguity that exists.

These works are orthogonal to ours. Our model is fully convolutional and makes use of depthwise separable convolution, to emulate one that would be used on a low power device, but to make clear the source of our improvements and contributions we do not attempt to compress or quantize it. In short, our model aims to show a power proportional solution for edge devices that optimizes for performance given a time varying resource budget with an adaptive neural network architecture.

Chapter 3

Methods

Generally, neural networks are often considered to exist in two pieces: an encoder, and a decoder. When a more powerful network is desired, practitioners often reexamine the encoder as it makes up the vast majority of network layers. However, in a dynamic deployment scenario it is not possible to modify the encoder without infusing changes that will propagate throughout the network. Since our goal is to improve performance without disturbing the base network configuration, we consider opportunities to improve the decoder. It is the job of the decoder to consolidate information extracted in prior layers and produce an accurate prediction. As [28, 21, 3, 29] have found, performing parallel operations on the same feature map and combining the outputs is an effective method for improving network performance. We consider how this same idea can be applied to the decoder to perform independent, parallel, operations on the feature map to produce outputs that can be combined with greater accuracy.

In this section we discuss our proposed feature decoders, and how they use atrous convolution for dense feature extraction. We also discuss the benefit of a secondary decoder and our proposed method of combining multiple network predictions as an ensemble.

Auxiliary Decoders

The baseline decoder is pictured in Figure 3 and is from the lightweight variant of [14, 30]. Aiming for a light-weight decoder that still has the benefits of atrous spatial pyramid pooling (ASPP) [28, 14, 31] it only has two branches. The two branches consist of a 1x1 convolution and a global average pooling. This way they perform feature extraction and consolidation at the narrowest and widest possible, fields of view, respectively. [32, 33, 21] have shown the importance of context aggregation in neural networks for accurate classification which is most

effectively achieved by the global average pooling that produces a channel wise likelihood for features. Oppositely, the 1x1 convolution processes features in a narrow field of view which is effective for aggregating local and narrowly concentrated information.

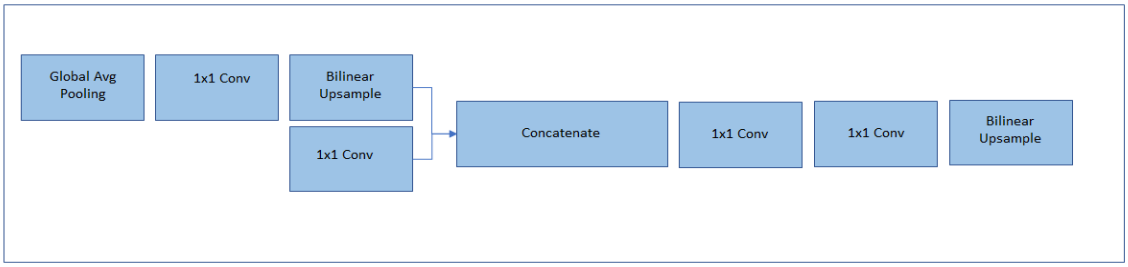

Figure 3: Baseline decoder structure.

Our first auxiliary decoder is shown in Figure 4. It is an extension of the baseline decoder to include three more 3x3 convolution branches with varying dilation rates. It is based on the decoder from [30, 14] but omits the skip connection and a few convolutional layers. Having three more convolutional layers with varying dilation rates allows for the decoder to extract features at arbitrary resolution and aggregate them for a more powerful prediction [28].

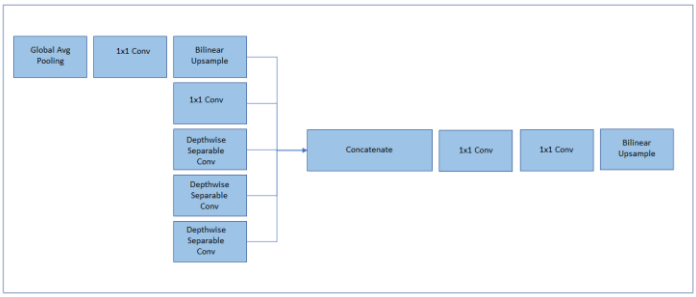

Figure 4. Modified Xception decoder.

The feature maps operated on by the baseline and first auxiliary decoders are at output stride of 8. Many prior works [33, 32, 30] have found image level features to be important to improving segmentation boundaries. To recover finer grained structural information, we recreate the decoder from [14] by adding a skip connection from early layers in the network. The

concatenated features are thus upsampled to an output stride of 4 and concatenated with the image level features. Then a few more convolutional layers combine the rich semantic information from the ASPP module with structural information from the entry flow for more accurate border refinement. The second decoder, is pictured in Figure 5.

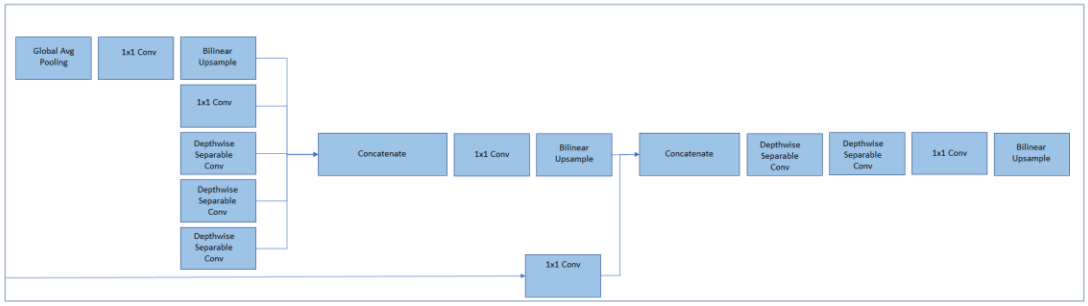

Figure 5. Modified Xception decoder with skip.

Atrous Convolution

Fully convolutional networks have proven to be an effective method of semantic segmentation [30, 14, 23]. They extend classification networks to make a prediction at every pixel instead of a single prediction to represent the entire image. Image classification networks have effectively achieved translation invariance and implicit contextual encoding with strided convolution or pooling operations [12, 34]. However, for segmentation networks, this repeated decimation of tensor spatial resolution poses a problem because it must eventually be recovered. Deconvolution layers have been used to work as a learned up-sampling of the data [34]. A more effective approach is to use ‘atrous convolution’ also known as dilated convolution [31, 35]. Dilated convolution expands the local receptive field of a filter without increasing the number of weights by expanding the filter along its spatial dimensions. By increasing the dilation rate of a filter instead of employing pooling operations or other methods of down sampling the spatial

resolution is maintained and less interpolation is required to convert a feature map to a prediction at the input resolution.

Ensemble Learning

Ensemble learning is a proven method of increasing model performance in machine learning [36]. The goal is to combine several models that are unlikely to fail identically to produce a net output that is more robust than any model individually. Broadly, there are two categories of ensembles: averaging and boosting methods. The former averages independent model outputs to produce a single prediction and the latter constructs a collection of sequential weak classifiers that can correct for errors produced by prior models to create a more powerful ensemble. In this work we focus averaging methods for improving model performance. Motivated by power efficient applications of neural networks it is impractical to construct an ensemble of independent models to average. However, due to the stochastic nature of neural networks we consider the possibility of constructing independent classifiers that reuse features computed by a single base network. In this way we do not need independent feature extractors. Enough variation in feature decoders exists to produce outputs dissimilar enough to have constructive interaction when combined.

Entropy Based Voting

Typically, independent model predictions are simply averaged to produce a single prediction. However, in the case of semantic segmentation, especially when ensembling models build on a shared set of features, we find that the greatest opportunity for refining improvement is along areas of uncertainty. Areas of uncertainty exist most prominently at the border of objects or classes, see Figure 6.

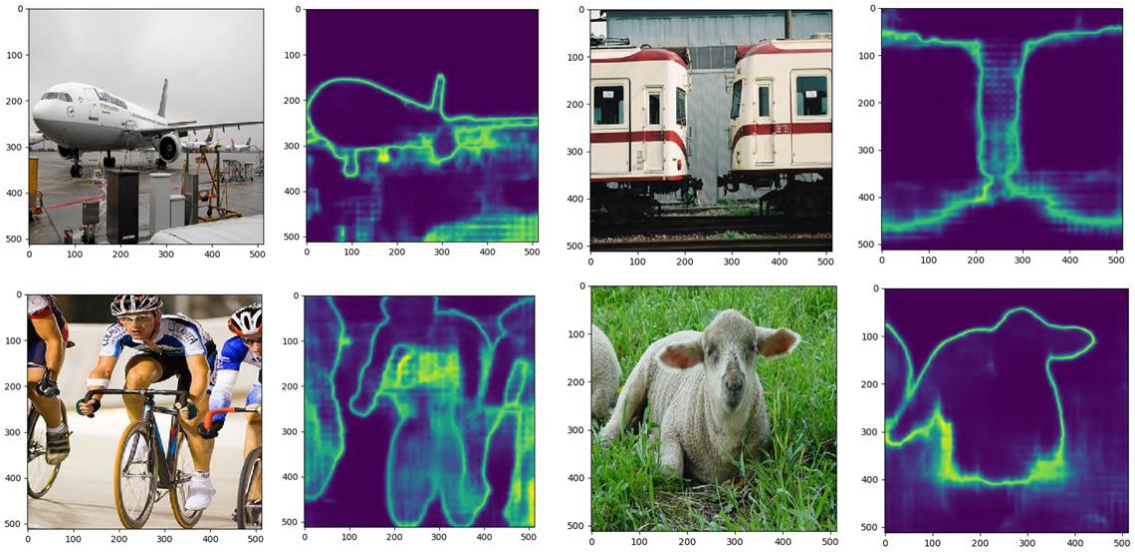

Figure 6. Entropy map of model predictions.

In these regions model predictions tend to have irregular shapes as features blur together and the network struggles to identify a precise boundary. Despite originating from shared features, the two network outputs produce slightly different predictions. Where the predictions are most likely to benefit from one another is in these regions of uncertainty. We use entropy as a measure of uncertainty and calculate the entropy at every pixel as equation.

$$f\left(\tilde{y}_{1(i,j)}, \ldots, \tilde{y}_{N(i,j)}\right) = argmax \sum w_n \frac{1}{H\left(\tilde{y}_{n(i,j)}\right)} \sigma\left(\tilde{y}_{n(i,j)}\right)$$

$$\sigma(\mathrm{y}) = \frac{\mathrm{e}^{y_\mathrm{i}}}{\sum_j \mathrm{e}^{y_j}} \qquad H(y) = -\sum y_i * \log(y_i)$$

Predictions are made according to the maximum value predicted in the classification tensor at a given pixel but there are no restrictions on scale. Therefore in order to combine outputs with equal scale we first transform each to a probability distribution with the softmax function computed as equation. The final output is then a sum between the two probability distributions weighted by a scalar factor and inverse entropy.

Entropy based voting can have the distinct advantage over simple averaging for combining predictions that the prediction that is more confident in a pixel will have a stronger effect on the output. This allows each classifier to powerfully influence the result when it is likely to be correct and minimize its contribution when it is uncertain.

Entropy values can be very small when a prediction is highly confident. The entropy of a corresponding pixel may be very small for multiple classifiers and for all intents and purposes they exhibit equal confidences. Despite this, one may have an order of magnitude smaller value and will dominate the output prediction. To mitigate this effect, we also consider a log-adjusted entropy defined as:

$$f\left(\tilde{y}_{1(i,j)}, \dots, \tilde{y}_{N(i,j)}\right) = argmax \sum w_n \max(-\log\left(H\left(\tilde{y}_{n(i,j)}\right)\right), eps)\sigma\left(\tilde{y}_{n(i,j)}\right)$$

Chapter 4

Experimental Evaluation

Our experimental evaluation considers two decoders, each more powerful than the baseline. We present our training procedure and results and finish with a table summarizing the different configurations and their corresponding model size and accuracy. Our base network has 2.1M parameters. Different configurations that we consider with their respective parameters, and performances are summarized in Table 1.

PASCAL VOC

We perform our evaluation on the PASCAL VOC [19] semantic segmentation data set consisting of 20 foreground classes and one background class. The dataset contains 1,464 train images, 1,449 validation images, and 1,465 test images.

Training Protocol

The base network is pretrained on COCO [37] and fine-tuned on the VOC train data as well as [38] extra annotations for a total of 10,582 training images(*trainaug*). We fine tune the extra decoder with the base network frozen on the trainaug set. The network is trained for 100K steps with input size of 512x512 and uses an adaptive moment optimizer [39] with base learning rate of 0.001 dropped by a factor of ten after 50 and 100 epochs with decay 0.000005. Following [30] we use a batch size of 16 for stability in batch normalization layers and duplicate images containing hard classes (bicycle, chair, potted plant, and sofa). Our model is evaluated according to the mean intersection over union defined below:

$$mIOU = \frac{TP}{TP + FP + FN}$$

Where TP, FP, and FN are the total true positives, false positives, and false negatives, respectively.

Loss

Our loss is the sum of the categorical cross entropy computed between the ground truth distribution, *p*, and the model prediction, *q*, after a softmax activation is applied along the last dimension of the output tensor. The loss is defined below:

$$L(p, q) = \sum_{m \in M} \sum_{n \in N} - \sum_{x \in X} w_{x(m.n)} p(x_{m.n}) \log(q(x_{m.n}))$$

We find it helpful to add a weight term, *w*, to decrease the contribution of the background class to account for class imbalance in the dataset. We set the weight of background to 0.85 and everything else to 1. We also use label smoothing [23] with an on-value of 0.9 to improve generalization. Our evaluation is performed at single scale with output stride (OS) of 8. We expect that our performance would improve if we evaluate at multiple scales and horizontal flip [14, 40] but the focus of our work is real-time applications, so we omit these exercises because they drastically increase computation cost.

Results

We add the first auxiliary decoder and train on the *trainaug* set. Evaluation results on the validation data is shown in Figure 7.

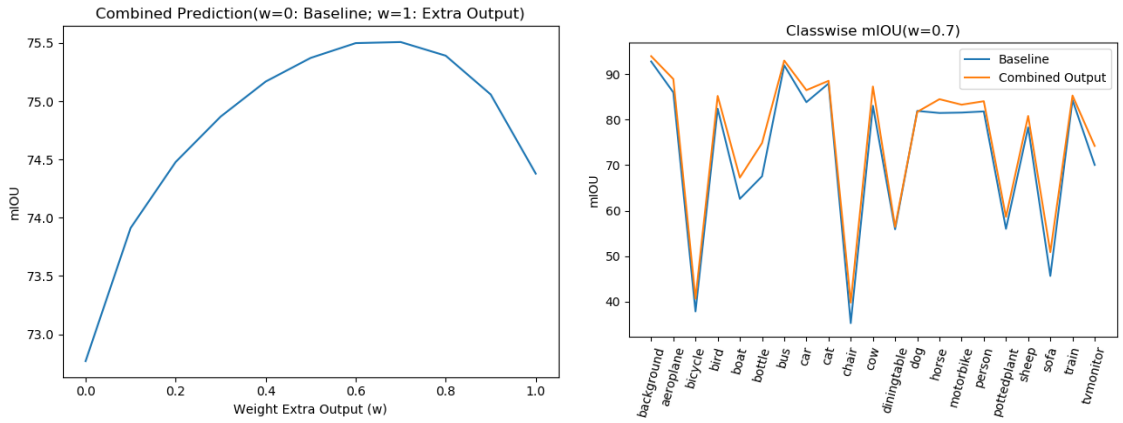

Figure 7. Total and classwise mIOU of combined model.

When adding a second decoder the accuracy improves. An interesting interaction appears as the model performs best when the output is a a weighted sum between the predictions made by the two decoders. Despite the auxiliary decoder having more parameters and branches, it benefits from infusion with the prediction from the baseline decoder. The overall accuracy improves by 2.7 mIOU, a considerable margin. For reference, [14] , which currently holds the state-of-the-art performance on PASCAL VOC, is aimed at server side deployment and has 25 times the number of parameters as our base network and achieves 84 mIOU. Figure 7 also shows the classwise distribution of mIOU scores. The improvement is universal, but unequal.

Next, we train the second auxiliary decoder and show the results in Figure 8 of the entire model, including the baseline decoder and both auxiliary decoders, with every combination of decoder weight summing up to 1 in increments of 0.1. We see that the best performing model contains near-equal contribution from each decoder, agreeing with our hypothesis that ensembling multiple, independent, classifiers is an effective strategy for improving model performance. Despite the effects of dropout or parallel convolutions there is still considerable independence to be gained by completely unique decoders. These results show that enough independence between classifiers exists that the sum is less likely to fail, especially identically, than any single one.

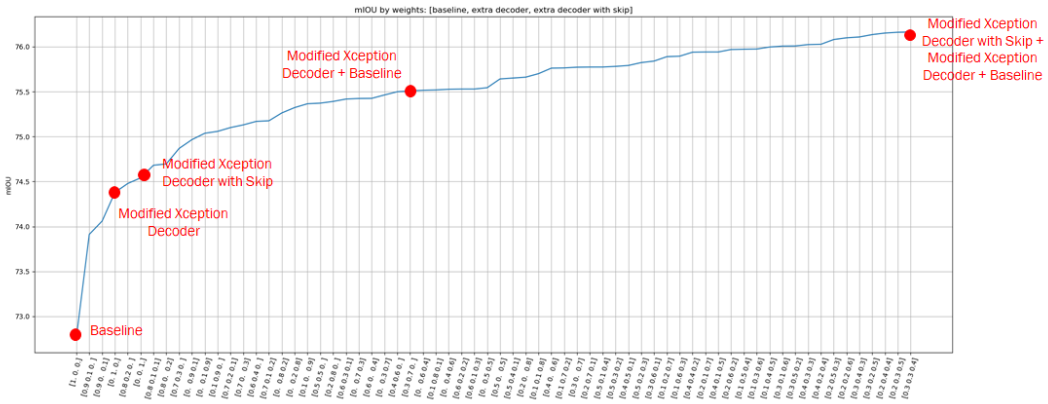

Figure 8: Comparison of all combinations of decoders using entropy-based averaging.

Figure 9: mIOU vs weights shown for entropy-based averaging, log-adjusted averaging, and simple weighted averaging.

For a comparison of all combination techniques see Figure 9. Interestingly, there exists large separation between techniques when the baseline decoder is weighted more heavily. Overall

our best performing model uses a simple weighted mean. Figure 9 shows the model prediction combining strategies at every weight combination. Complete results are summarized in Table 1.

Model Variant	**mIOU**	**Model Size (in parameters)**
Baseline	72.8	2.1M
Xception Decoder	74.4	2.7M
Baseline + Xception Decoder	75.5	3.0M
Baseline + Xception Decoder with Skip	75.6	3.2M
Xception Decoder + Xception Decoder with Skip	75.8	3.7M
Baseline + Xception Decoder + Xception Decoder with Skip	**76.2**	4.0M

Table 1. Model variants, mIOU scores, and total model weights.

The results in Figures 8 and 9 reflect these increases in accuracy. See Appendix A for a visualization of results. By adding more powerful decoders we see that there are fewer checkerboard artifacts which are a product of the lack of local contextual information. Additionally, borders are better defined, and objects are segmented more smoothly. Figures 8 and 9 show the effectiveness of ensembling the decoders as the joint output is more accurate than either is independently.

We show that there is considerable accuracy that can be extracted from a model by employing auxiliary decoders, and they can be deployed in a dynamic fashion. Each decoder can

be invoked without dependence on one another allowing for marginal tradeoffs to be made. This allows for the ability to navigate the design space with respect to marginal accuracy and computational complexity from a base network.

Chapter 5

Conclusion

State of the art algorithms for segmenting images and producing dense predictions rely on convolutional neural networks. Applications that use these networks are often limited to a single choice due to the high overhead of changing networks as well as the non-linear relationship between model size and performance. This leads to inefficient use of dynamically available resources. In this work we have developed a novel method for improving the accuracy of a network without modifying the core architecture or weights. This opens opportunities for power aware network reconfiguration such that available power and compute can be more effectively utilized in real time applications. Additionally, we show that the stochastic nature of neural networks allows for ensembling of semi-independent classifiers built from a common convolutional base. This lays the foundation for improving the performance of arbitrary models with minimal retraining and run-time adaptive architectures to maximize available resource usage and performance.

Appendix A

Visualization results on held out validation set

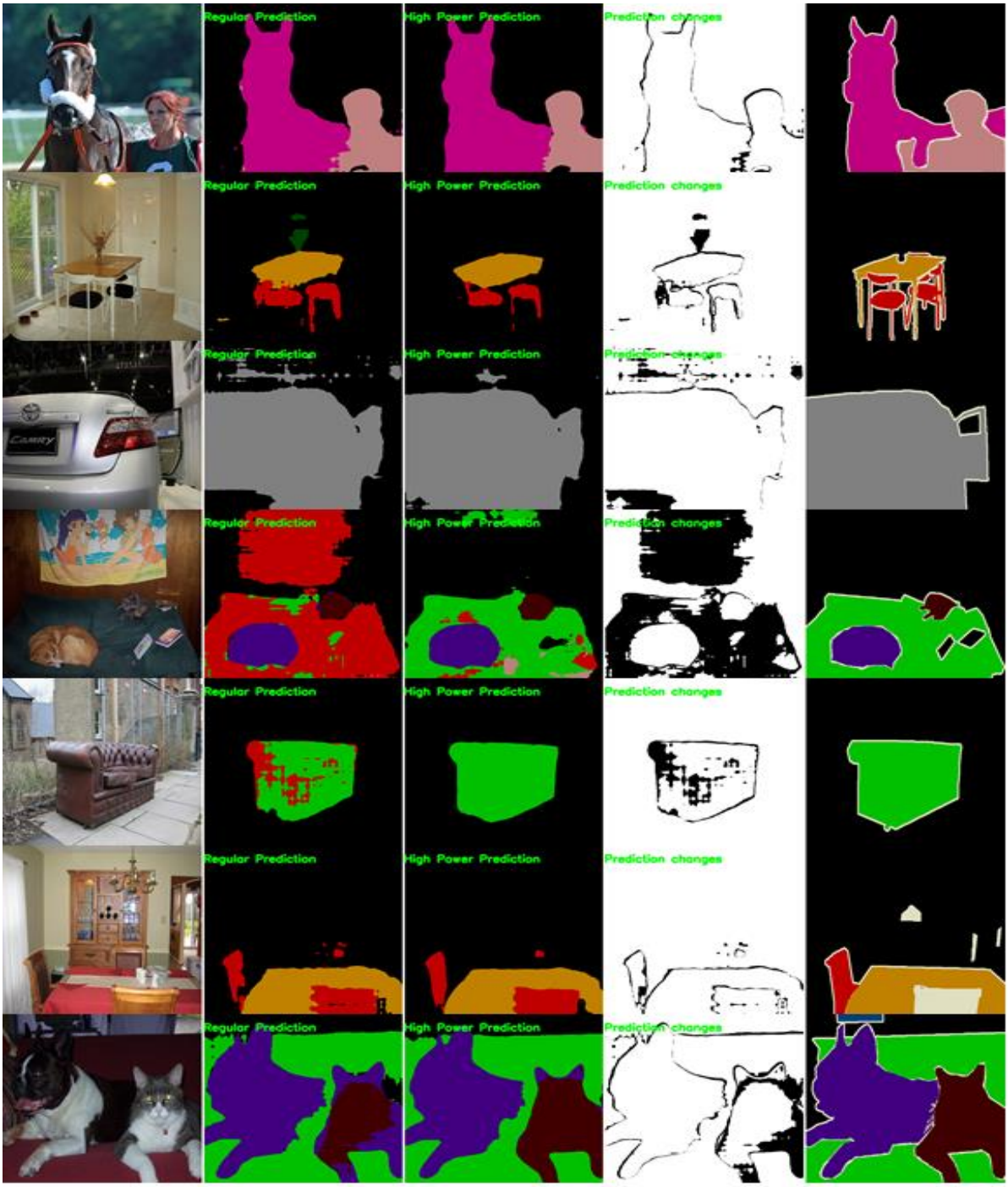

Figure 13. Visualization results on validation set. From left to right: input image, Baseline prediction, model with all three decoders, differences between predictions, ground truth labels.

Appendix B

For decades, computer vision researchers have struggled to emulate the human visual system for several reasons including differing sensing modalities, processing patterns, and a lack of deep neurological understanding of the brain. But perhaps the most important is that humans themselves cannot formalize a strict definition for their visual information processing system. Take for example a cat. Putting together a set of cohesive rules that exactly define a cat is impossible. Is it the presence of fur, the shape of the ears, a tail? No, because a dog can have all of the same features. Let us pretend for a moment that we did have a strict definition of a cat. Now consider how that definition changes under the presence of occlusions, deformities, or other confusing scenarios. Finally, even if an exact definition of a cat existed we must then encode it in terms of image pixel values. We quickly see that this is not feasible.

Images are among the highest dimensional data that is commonly processed. A standard 300 by 300 color image has 270,000 pixels, each of which are represented by an 8-bit value in the range [0,255]. Therefore, there are $256^{270,000}$ possible images that could exist at that resolution. Despite the high dimensionality of images, they are still one of the most effective means of transmitting information. It is for this reason that we turn to machine learning, and most recently, deep learning.

Neural Networks

Traditional computer vision practitioners have struggled to build algorithms that are able to perform with accuracy sufficient for industrial applications. It is only in recent years that the possibility of human level performance was within reason and that is largely due to the rise in neural networks. At the core of deep learning is the multilayer perceptron (MLP). An MLP is a series of neurons arranged in layers. The quintessential MLP has an input layer, a variable number of hidden layers (layers between the input and output), and an output layer. Each neuron performs a dot product between vectors of incoming values and weights, $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{w})$, typically followed by a non-linear activation function $\emptyset$. The mapping between input and output of a neuron is described in eq (X).

$$f(\boldsymbol{x}, \boldsymbol{w}) = \emptyset(\boldsymbol{x} \cdot \boldsymbol{w}) = \emptyset\left(\sum_{i=0}^{n} x_i \cdot w_i\right)$$

Convolutional Neural Networks

Whereas the MLP takes input in the form of a 1D vector, CNNs are specialized for data that exhibits a grid-like shape such as an image. The convolutional operator is the most fundamental building block of the modern CNN. Convolution is a type of linear operator that effectively ‘slides’ one function over another:

$$s(t) = \int x(a)w(t-a)da$$

Or defined as a descrete convolution:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Activation functions

Because convolution is a linear operation it is commonly followed by a non-linear function known as an activation function. Introducing these non-linearities allows the network to model more complex functions [2]. Without these non-linearities a network would not be able to approximate high-dimensional non-linear functions such as those that exist in image recognition tasks. Common activation functions are as follows:

Sigmoid:

$$\emptyset(x) = \frac{e^x}{e^x + 1}$$

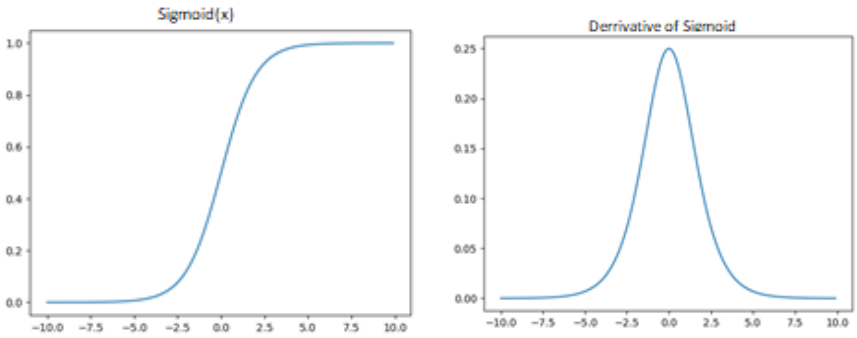

Figure B-1. Sigmoid activation and derivative.

The sigmoid function resembles the binary nature of neurons found in the brain. Figure B-1 show the response and derivative, respectively, of the sigmoid function. Due to its resemblance of the binary nature of neurons found in the brain the sigmoid function was a commonly used activation in early neural networks. It also gave credence to the parallels between artificial neural networks and those found in neuro-biological architectures. However, despite the rise in neuromorphic computing the sigmoid activation has fallen out of favor due to the narrow

band of non-saturation region as well as having a maximum first order derivative of 0.25 meaning that the magnitude of gradients during training will be reduced by at least a factor of four at every layer that sigmoid is used [2].

ReLU:

$$\emptyset(x) = \max(x, 0)$$

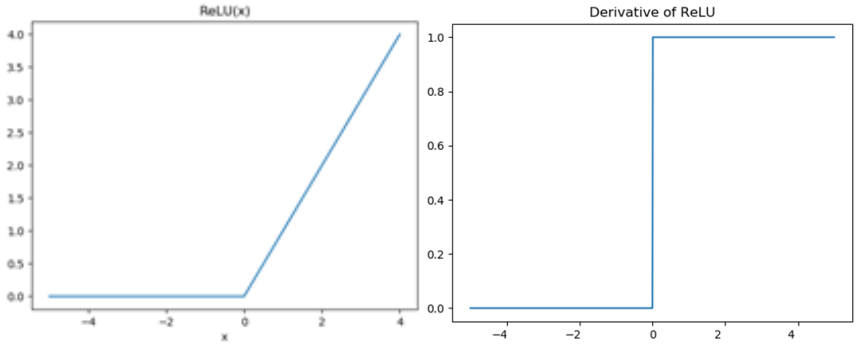

Figure B-2. ReLU activation and derivative.

Instead of sigmoid, the most popular activation function is the Rectified Linear Unit [41] (ReLU). ReLU is a relatively simple activation function that can be quickly computed and allows for better flow of gradients. Whereas networks using sigmoid tend to suffer from slow training due to vanishing gradients the same is not true of ReLU. Similarly, ReLU shows better network convergence properties [5].

Gradient-based optimization:

Initially a neural network is filled with random weight values. There exist methods to improve the initialization of weights [42, 43, 44] but they are unimportant to developing an understanding of optimization. Training of a network then begins by feeding inputs in and producing an output. The output is evaluated against a loss function. A loss function is a

differentiable function defined to quantify the error between the network prediction and the ground truth label. The network weights are then changed such that these changes will minimize the loss. This process repeats until the network converges and the loss no longer improves. There are two key components to determining how to update network weights after each training step: backpropogation and stocastic gradient descent(SGD).

Back Propagation

Back propgation is an algorithm developed in the 1960's [43] to compute the derrivatives of every weight in a network with respect to the loss function. It is possible to find the minimum of a differentiable function, however as modern neural networks contain millions of parameters it is computationally intractible to do so. Instead, the training procedure consists of computing the gradients of network parameters using the backpropogation algorithm and updating the weights in the opposite direction. This process of taking steps in the opposite direction of the gradient is known as stochastic gradient descent. "Stochastic" because training examples are drawn randomly and "gradient descent" because the network 'descends' the loss surface.

Stochastic Gradient Descent:

Backpropagation give the value of the gradient and SGD updates weights accordingly to improve network performance. The goal of network training is to find the minimum of the loss surface. By finding the gradients of each parameter with respect to the loss function it is known which direction to move the weights to minimize the loss value. SGD is an iterative process updating weights according to the opposite direction of the gradient surface.

References

[1] C. Sun, A. Shrivastava, S. Singh and A. Gupta, "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era," ICCV, 2017.

[2] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in *CoRR*, 2015.

[4] A. Karpathy, "What I learned from Competing against a ConvNet on ImageNet," 2 September 2014.

[5] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[6] A. Cabzuabu, E. Culurciello and A. Paszke, "An Analysis of Deep Neural Network Models for Practical Applications," in *ICLR*, 2017.

[7] A. Saeed, A. Abdelkader, M. Khan, A. Neishaboori, K. Harras and A. Mohamed, "On Realistic Target Coverage by Autonomous Drones," in *ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2018.

[8] P. A. .. Zientara, J. Choi, J. Sampson and V. Narayanan, "Drones as collaborative sensors for image recognition," in *IEEE International Conference on Consumer Electronics (ICCE)*, 2018.

[9] Barry and T. R. Andrew J., "Pushbroom Stereo for High-Speed Navigation in Cluttered Environments," in *IEEE International Conference on Robotics and Automation*, 2014.

[10] H. Su, S. Maji, E. Kalogerakis and E. Learned-Miller, "Multi-view Convolutional Neural Networks for 3D Shape Recognition," in *ICCV*, 2015.

[11] D. Emily, Z. Wojciech, B. Joan, L. Yann and F. Rob, "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation," in *NIPS*, 2014.

[12] G. Hinton, O. Vinyals and J. Dean, "Distilling the Knowledge in a Neural Network," NIPS, 2014.

[13] S. Terrapittayanon, B. McDanel and H. Kung, "BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks," in *ICPR*, 2016.

[14] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," in *ECCV*, 2018.

[15] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *ICLR*, 2015.

[16] M. Abadi, A. Agarwal and e. al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," arxiv prepring, 2016.

[17] Microsoft, "Project Catapault," Microsoft, 2018.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and F.-F. Li, "Imagenet large

scale visual recognition challenge," *International Journal on Computer Vision,* pp. 211-252, 2015.

[19] M. Everingham, S. M. A. Eslami, L. Van, C. K. I. W. Gool, J. Winn and A. Zisserma, "The pascal visual object classes challenge a retrospective," *IJCV,* 2014.

[20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *CVPR*, 2018.

[21] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017.

[22] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," in *ICLR*, 2016.

[23] C. Szegedy, S. Ioffe and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *CoRR*, 2016.

[24] S. Han, J. Pool, J. Tran and W. Dally, "Learning both Weights and Connections for Efficient Neural Networks," in *NIPS*, 2015.

[25] M. Rastegari, V. Ordonez, J. Redmon and A. Farhadi, "XNOR-NET: ImageNet Classification Using Binary Convolutional Neural Networks," arXiv preprint arXiv:1603.05279, 2016.

[26] C. Sakr, Y. Kim and N. Shanbhag, "Analytical Guarantees on Numerical Precision of Deep Neural Networks," in *ICML*, 2017.

[27] M. McGill and P. Perona, "Deciding How to Decide: Dynamic Routing in Artificial Neural Networks," in *ICML*, 2017.

[28] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," in *TPAMI*, 2017.

[29] O. Ronneberger, P. Fischer and T. Brox, "U-NET: Convolutional Networks for Biomedical Image Segmentation," in *MICCAI*, 2015.

[30] L.-C. Chen, G. Papandreou, F. Schroff and H. Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation," in *CoRR*, 2017.

[31] K. He, X. Zhang, S. Ren and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Neural Networks for Visual Recognition," in *TPAMI*, 2015.

[32] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun and A. Yuille, "The Role of Context for Object Detection and Semantic Segmentation in the Wild," in *CVPR*, 2014.

[33] J. Redmon and A. Farhadi, "Yolo9000: Better, Faster, Stronger," in *arXiv preprint*, 2016.

[34] H. Noh, S. Hong and B. Han, "Learning Deconvolution Network for Semantic Segmentation," in *ICCV*, 2015.

[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single Shot Multibox Detector," in *ECCV*, 2016.

[36] D. Optiz and R. Maclin, "Popular Ensemble Methods: An Empirical Study," *Journal of Artificial Intelligence Research*, p. 169=198, 1999.

[37] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *ECCV*, 2014.

[38] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji and J. Malik, "Semantic contours from inverse detectors," in *ICCV*, 2011.

[39] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2014.

[40] S. Ren, K. He, R. Girshick and J. Sun, "Faster RCNN: Towards real time object detection with region proposal networks," in *NIPS*, 2015.

[41] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *ICML*, 2010.

[42] A. M. Saxe, J. L. McClelland and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *ICLR*, 2014.

[43] L. B. G. O. K.-R. M. Yann LeCun, "Efficient BackProp," 1998.

[44] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010.

[45] V. Badrinarayanan, A. Kendall and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," in *PAMI*, 2017.

[46] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," in *Proc of the IEEE*, 1998.