UNIVERSITY OF GOTHENBURG

Deep Neural Network Compression for Object Detection and Uncertainty Quantification

Master's thesis in Computer science and engineering

MOHAMMAD AHRAZ ASIF
GEORGIOS TZELEPIS

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Deep Neural Network Compression for Object Detection and Uncertainty Quantification

MOHAMMAD AHRAZ ASIF

GEORGIOS TZELEPIS

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Deep Neural Network Compression for Object Detection and Uncertainty Quantification

MOHAMMAD AHRAZ ASIF
GEORGIOS TZELEPIS

Supervisor: Dr Eren Aksoy, Halmstad University
Advisors: Saimir Baci, Selcuk Cavdar, Volvo Group
Examiner: Graham Kemp, Chalmers

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2019

Deep Neural Network Compression for Object Detection and Uncertainty Quantification
MOHAMMAD AHRAZ ASIF
GEORGIOS TZELEPIS
Department of Computer Science and Engineering
University of Gothenburg and Chalmers University of Technology

Abstract

Neural networks have been notorious for being computational expensive. Their demand for hardware resources prohibits their extensive use in embedded devices and puts restrictions on tasks like real time tracking. On top of that, neural networks are usually deterministic and provide no uncertainty which is crucial on safety decision tasks and physical sciences.
In this work techniques were developed to reduce the computational cost of neural networks, such as Model Compression infused with a novel dynamical clustering and Knowledge Distillation while estimating the impact of such techniques on the uncertainty of the model by using Bayesian neural networks. A brief introduction is made on deep learning and the tools used. Furthermore the ideas of Model Compression, Knowledge Distillation and Bayesian Deep Learning were extended analytically. All three approaches were tied together followed by the final discussion.

Keywords: Deep Learning, Model Compression, Knowledge Distillation, Bayesian Deep Learning, Object Detection.

Contents

1 Introduction

1.1 Motivation

Deep learning, a subset of techniques within machine learning artificial neural networks, is becoming evermore present across a wide variety of industries. One such example is the automotive industry, where deep learning techniques are not only considered "mature and viable technology" [1] but are also applied in multiple, and often vastly different domains within business. From predicting fuel consumption of diesel engines [2] to the cybersecurity of in-vehicle networks [3], deep learning is becoming an increasingly viable approach to a variety of different problems.
One area of application of deep learning is within the field of autonomous driving, where neural networks are used in scene segmentation, object detection [4] and route planning. While considered mature and reliable, neural networks have the distinct disadvantage of being computationally expensive to train and use. In computationally resource constrained environments such as vehicles, it becomes paramount to optimize the networks with respect to both memory and inference time. Thus, in domains where achieving high levels of accuracy is vital (e.g. due to safety reasons) and where resources are limited, methodologies to make neural networks more efficient are essential. Furthermore, due to the prevalence of deep learning in a variety of industries and processes, efficiency gains will likely benefit a wide variety of projects, individuals and speed up research.
The models that are used for these tasks will return a value which represents the confidence of the predictions. For example if a model has been trained for cat-dog classification tasks it will learn to distinguish rather well the difference between those two classes. But what if the model is presented with a sample outside the distribution of the dataset that was used for training?
Such scenarios are prevalent in more serious settings, such as medical examinations with structures that the system has never observed before or scenes for autonomous vehicles that the steering system was not trained for. Ideal behavior from the network would be to return a prediction accompanied with a level of uncertainty of this prediction, enabling the identification of such uncertain predictions by people or systems relying on high confidence predictions. Bayesian Deep Learning can provide a solution in this problem, but it has an impact in the computational cost.

1.2 Thesis Structure

The report was divided into four sections, each with a respective investigative summary and the results of those investigations presented. The initial sections[2, 3, 4] of Model Compression, Knowledge Distillation and Bayesian Deep Learning detail three separate tracks of the investigation and the findings associated. Each of these sections is self-contained and does not refer to each other. Finally, the combined use of Model Compression, Knowledge Distillation and Bayesian Deep Learning are presented in the last section [5].
The target audience was intended to be those with a background in the basic concepts of Machine Learning and Deep Learning. The first two sections [2, 3], Model Compression and Knowledge Distillation can be followed by one who is familiar with the concepts of neural networks, and is knowledgeable of convolutional networks problems, such as object detection and classification. The third section of the thesis [4]], Bayesian Deep Learning, requires one to be familiar with bayesian statistics and probability theory on top with deep learning for classification problems. Finally the fourth section [5], is where all the different combinations of all three previous methods take place and can be split into four smaller sections so that the reader can focused into the pair of the method

that is interested in. Section 5.1 can be read by anyone who went through Model Compression and Knowledge Distillation, sections 5.2 and 5.3 investigate the impact of Model Compression and Knowledge Distillation respectively to the model's uncertainty while section 5.4 is a combination of all the 3 methods.

1.3 Background

A neural network is an interconnected graph representation of multiple processing layers, consisting of nodes called "neurons", each of which use an "activation function" to conduct a (typically) non-linear transformation on input data. Neural networks consist of an initial "input layer", which transforms the initial input data into a numeric representation which can be processed within a neural network. Output layers are used to transform the abstract representations of the input data formed by previous layers into classifications or predictions, which are then used in analysis or fed into other systems for further processing [5] [1].
The input and output layers of neural networks are connected by "hidden layers". Like the input and output layers, the hidden layers are also interconnected neurons which conduct the activation function upon the input data. They conduct the process of transforming the input data into different representations to extract latent information from the input signals. This enables a neural network to conduct higher order operations, such as separating data that was previously not linearly separable, typically by increasing the dimensionality of the input into a plane where it is separable.
Between the processing layers, the nodes are connected to each other by a weighted "edge", through which the output of the previous node is fed into the current one. When the assigned "learning algorithm" is executed during the training phase of a neural network, it is these weights that are modified towards producing the correct output. There are several different types of learning approaches; supervised learning where a model is trained on a labeled (typically manually so) training data, unsupervised learning in which a model conducts statistical observations about the given data and uses those metrics for evaluation or semi-supervised which typically uses a combination of both.
One example of a popular and effective learning algorithm is *backpropagation*. During one *iteration* of the learning algorithm, initially a *forward pass* is conducted where the output of the activation functions of all of the neurons from the input to the output layer are calculated. Subsequently, a *backwards pass* is conducted: starting from the output layer the neurons are traversed once again however this time deriving the error of the output of each neuron by using gradient descent. Subsequently, the weights are modified such that the output of each layer is changed to produce the correct overall output.
Once an iteration is conducted on each instance of the training data, an *epoch* is said to have been completed. Typically, during the training phase of a neural network, several epochs are conducted. Additionally, there are several parameters that may be manually tuned to achieve optimum results and performance, which range from but are not limited to: the number of hidden layers and units, the number of neurons a given neuron is connected to, the learning rate of the gradient descent function, the number of epochs to conduct, etc.
The training and processing time of a neural network largely depends on the size of the input data, the number of hidden layers, the number of neurons, the number of connections between the neurons and finally the hardware profile on which the neural network is being executed. Neural network architectures used in complex domains involving large amounts of data (e.g. real-time

object detection for video streams) involve many layers with many neurons.. These "deeper" neural networks (i.e. those with more layers and neurons) tend produce more accurate results, at the cost of performance. Consequently, this leads to the performance of such neural network being very slow, with large processing and training times (i.e. days-weeks).
State-of-the-art neural network algorithms for object detection such as FasterRCNN [6], Mask-RCNN [7] or YOLOv3 [8] consist of several hundred layers each having between hundreds to thousands of neurons with weighted connections. Furthermore, they leverage the use of *convolutional layers* which are used specifically for image analysis. They are capable of decomposing an image into abstract representations which enable the neural network to more easily identify patterns within the image. By using convolutional layers, neural networks are able to break down an image into the representations that are most relevant towards the task the neural network is being trained for.
In order to leverage the use of such state-of-the-art network architectures and computationally expensive techniques such as residual blocks or convolutional layers within resource constrained environments such as embedded systems, it is important that investigations are conducted into how network architectures can be made more efficient through the reduction of parameters that have negative or no impact towards useful predictions. Furthermore, increasing inference efficiency results in a more energy-efficient neural network (as fewer computations have to be done to achieve state-of-the-art results) and a more memory-efficient neural network reducing requirements on storage.

1.4 Architectures

Techniques to optimize the inference time and memory footprint of neural networks were applied to several neural network architectures within the domains of classification and object detection problems, with the aim to improve state-of-the-art networks to perform more efficiently. Large-scale, public and open datasets were chosen for evaluation using open-source implementations of popular neural network architectures.

1.4.1 Classifiers

MNIST classifier The simplest architecture used for evaluation was a classifier for the MNIST dataset [9]. It consisted of a convolutional layer with one input channel, 20 output channels and a 5x5 kernel. The next layer was another convolutional layer with 20 input channels, 50 output channels and a 5x5 kernel. Subsequently there were two fully connected layers with 800 input channels and 500 output channels, and finally with 500 input channels and 10 output channels. All of the layers used ReLU activation function and max pooling was applied after each layer. After the final layer a softmax function was applied to obtain the class probabilities.

Cifar shallow classifier Another simple architecture, trained and evaluated against the Cifar-100 [10] small images dataset. It consisted of a convolutional layer with three input channels, six output channels and a kernel size of 5x5 followed by a MaxPool layer. Next was another convolutional layer with 6 input channels, 16 output channels and a 5x5 kernel, and finally three fully connected layers with 800 input channels and 120 output channels, 120 input channels and 84 output channels and finally 84 input channels and ten output channels respectively. Max pooling

was applied after both convolutions and the ReLU activation function was used between the first two fully connected layers. For this model, training and evaluation was done on the CIFAR dataset.

Dogbreeds Classifier Residual Networks[11] were implemented for the Stanford Dogbreeds dataset, which is a subset of Imagenet [12] data focusing on dogs, to classify 120 classes of dogs. Residual Networks were made consisting of blocks(Figure [1]) that use skip connections to transfer the input information after to the last layer of the block. Also batch normalization[13] was applied after each convolution in order to help converging to the optimal solution by solving the internal covariate shift. More specifically the networks to be used were ResNet18, ResNet34 and ResNet50. The two former are consisted of basic block while the later with bottlenecks. Only one fully connected layer was attached as a feature extractor. The feature layers are pre-trained on Imagenet there only the extractor was fine-tuned.

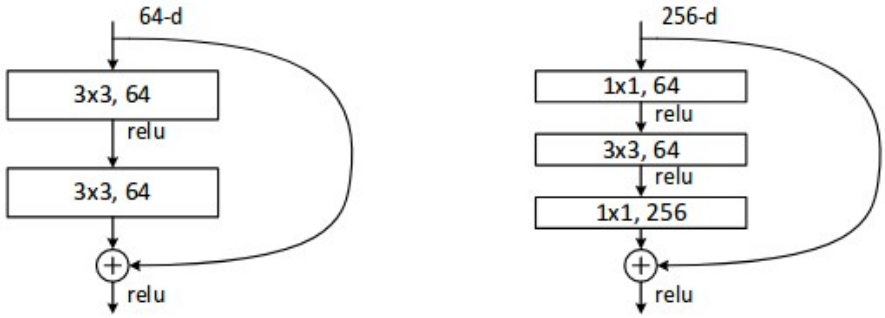

Figure 1: Basic and Bottleneck residual blocks

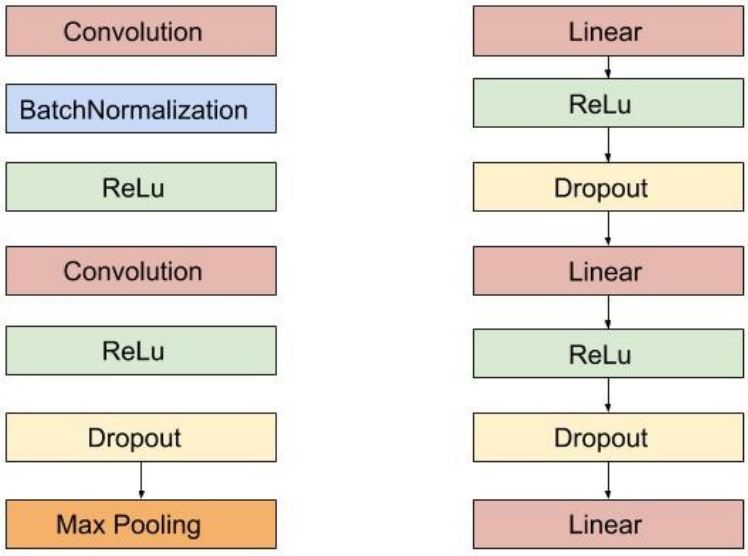

Figure 2: Convolutional block (left) and Feature Extractor (right) blocks

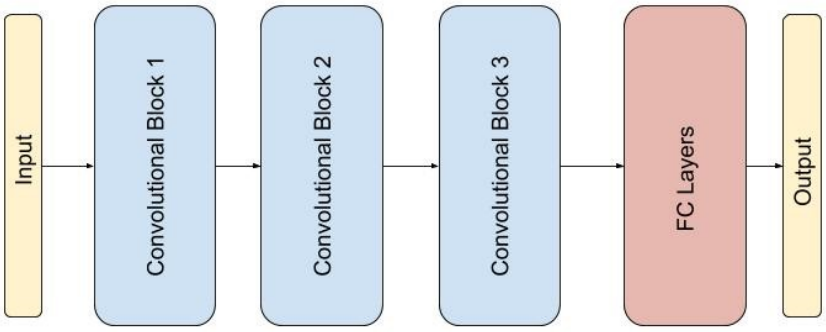

Figure 3: Overall CIFAR10 Model Classifier

CIFAR10 classifier For the CIFAR10 dataset in order to produce superior results than the simple architecture previously mentioned, two new architectures were constructed. One was a plain neural network that consisted of three convolutional blocks (Figure:[2]) which included batch normalization and dropout before downsampling with max pooling (Figure:[3]) and a residual network which is consisted of basic blocks. It was half the size of ResNet18. It is worth noting that dropout should be in place after batch normalization in each block since according to Li et al. [14], if the dropout is used before the batch normalization it can cause a shift in the weight distribution and thus minimizing the effect of batch normalization during training.

1.4.2 Object Detectors

YOLOv3 A state-of-the-art object detection network architecture, YOLOv3 [8] has been shown to have astonishing performance in object detection problems regarding the inference time. It is a one-stage detector where the fully convolutional network predicts the bounding boxes and class probability of those boxes in one pass.

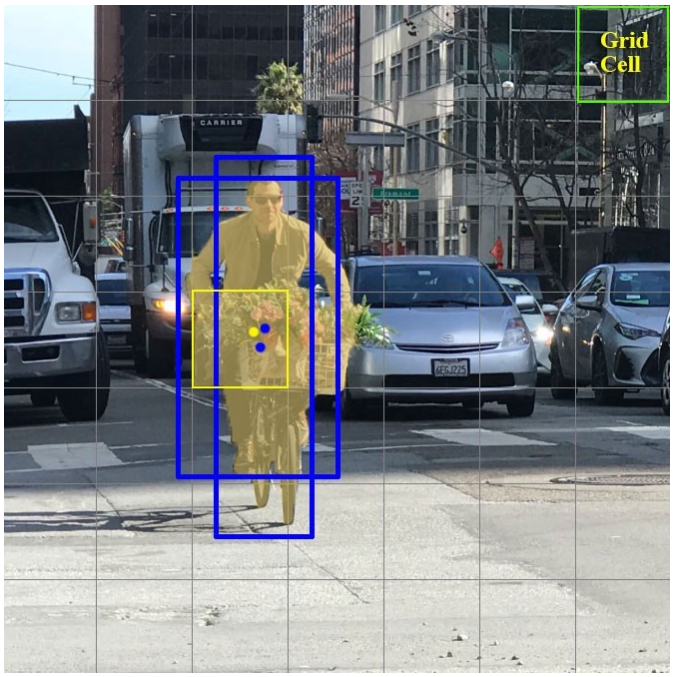

Figure 4: Each grid cell is responsible for classifying one object only. Then a center of this object is estimated and several boundary boxes are generated. Image Source [15]

More precisely after YOLOv3 is fed an input, it predicts tensors that correspond to 3 different scales. The scales that are used in this thesis are $13 \times 13, 26 \times 26$ and 52×52. Those scales can be represented as a grid over the input image and are used to enable the network to detect objects of multiple sizes. The dimensions of the tensor one can see in 5, comes from the formula

$$(N \times N) \times (3 \times (4 + 1 + NumClasses))$$

If the center of the object's ground truth falls into a specific cell, then this cell is responsible for classifying and detecting the object. The corresponding objectiveness score for that specific cell is "1" and "0" for the others. For each grid cell, 3 different boundary boxes are assigned to them which are initialized by using K-means clustering and during training the network learns to compute their precise coordinates. After the boundary boxes are generated from the grid cell, the one which overlaps the most with the ground truth via an Intersection over Union rule will be used to detect the object.

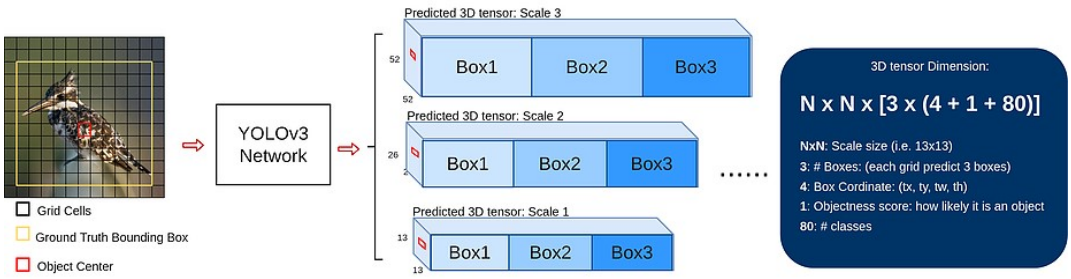

Figure 5: YOLOv3 detector used for the MS-COCO[16] dataset which contains 80 classes. Source:[17]

The network is made up of 75 convolutional layers and no fully connected layers. In addition the architecture is inspired by Residual Networks(ResNet) and from Encoder-Decoder models

such as Feature Pyramid Networks (FPN)[18](Figure:[6]). The encoder, which is responsible for down-sampling, has a stride of two and is used instead of max-pooling layers. Subsequently an FPN is used to maintain the details that are lost during the encoding part of the network by applying element-wise summation to the corresponding layers of the decoder, while ResNet adds skip connections within the residual blocks that allow the network to be influenced from previous layers in order to generate better features.

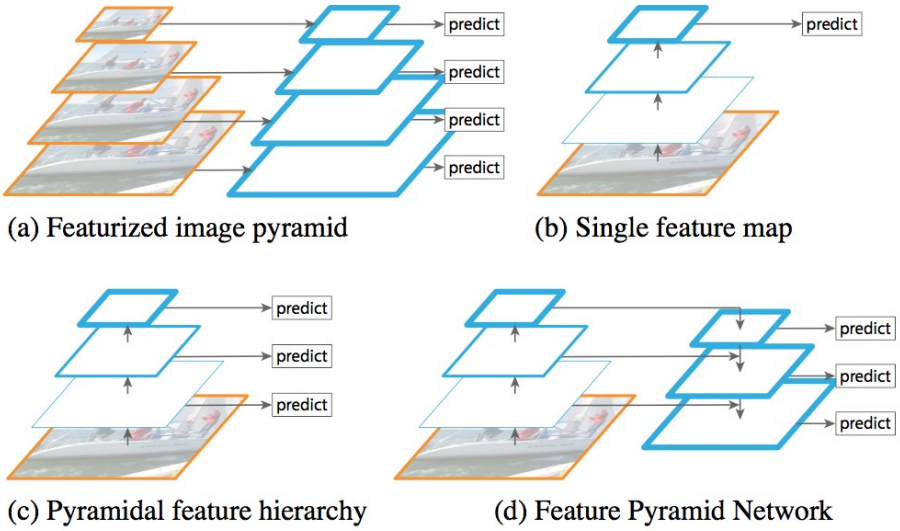

Figure 6: Feature Pyramid Networks

One of the main problems in detection is capturing the small sized objects. This is due to a phenomenon that occurs as the network gets deeper, there is further and further distortion and therefore reduced information from the downsampling FPN's. Therefore it is better to extract the small objects from the feature map as early as possible. However since the features tend to be different in the deeper layers YOLOv3 uses an FPN-like technique and concatenates later convolutional layers with earlier features. A detailed network architecture can be seen in Figure[7].

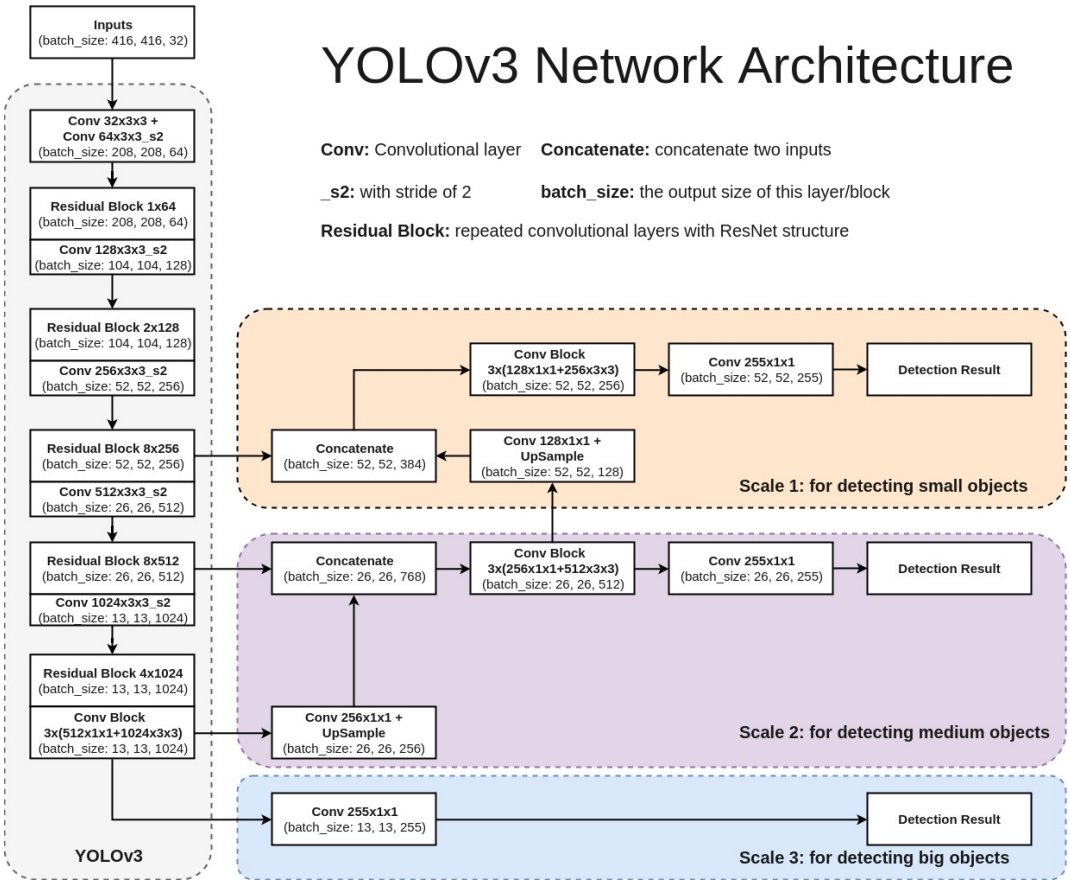

Figure 7: Overall model for YOLOv3 Source:[17]

Faster-RCNN Another state-of-the-art object detection algorithm, Faster-RCNN [6](Figure:[8]) is a two-stage detector. The network takes the last feature map of a convolutional layer and by using an operation called Regional Proposal Networks, it separates what is likely background and what likely an object. The areas that are candidates containing an object are called Regions of Interest (RoI) and are the ones who are used to detect the objects through further processing by the detection network.

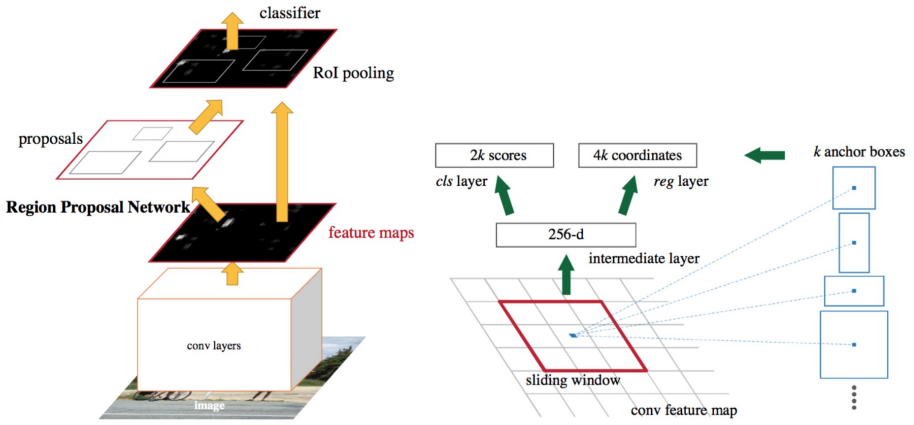

Figure 8: Faster-RCNN pipeline. Image Source:[19]

Faster-RCNN is composed of three different neural networks; (1) The Feature Network which is responsible for constructing the features and is usually a ResNet,(2) The Regional Proposal Network (RPN) that is usually a plain three layered convolutional network which detects the regions on the last feature map with a high probability of containing an object and generate a number of bounding boxes for them called Regions of Interest (RoI) and (3) the Detection Network, that is usually made up of four Fully Connected Layers which take those RoIs produced by the RPN and generate the final class and bounding box.

For training the RPN, a number of bounding boxes are generated though a procedure called anchor boxes. Every pixel of the feature image is considered an anchor. Those anchors are positioned uniformly across both dimension of the image. Usually nine boxes of different shapes and sizes are generated for each anchor (Figure[9]).

As a result, tens of thousands of boxes are generated. To reduce their numbers, the first step is to call the Non-Maximum Suppression(NMS) which removes boxes that overlap with each other. Boxes that have higher scores are candidates for the detection network. These RoIs are used by the RPN to classify objects as foreground and the rest as background through an Intersection over Union (IOU) process with the ground truth. The RPN is also responsible for tightening the center of the anchor boxes around the target. This is called bounding box regression and measure the distance from the center of the ground truth box to the anchor box through a smooth L_1 loss. That loss is used in the backpropagation to train the RPN.

In the Detection Network, IOUs of all the RoIs and the ground truth boxes are calculated. Depending on IOU thresholds (e.g. foreground above 0.5, and background between 0.5 and 0.1), labels are generated for a subset of RoIs. The difference with the RPN is that here there are more classes. Also a similar procedure with the bounding box regression takes place as well. The loss calculation is similar to that of the RPN.

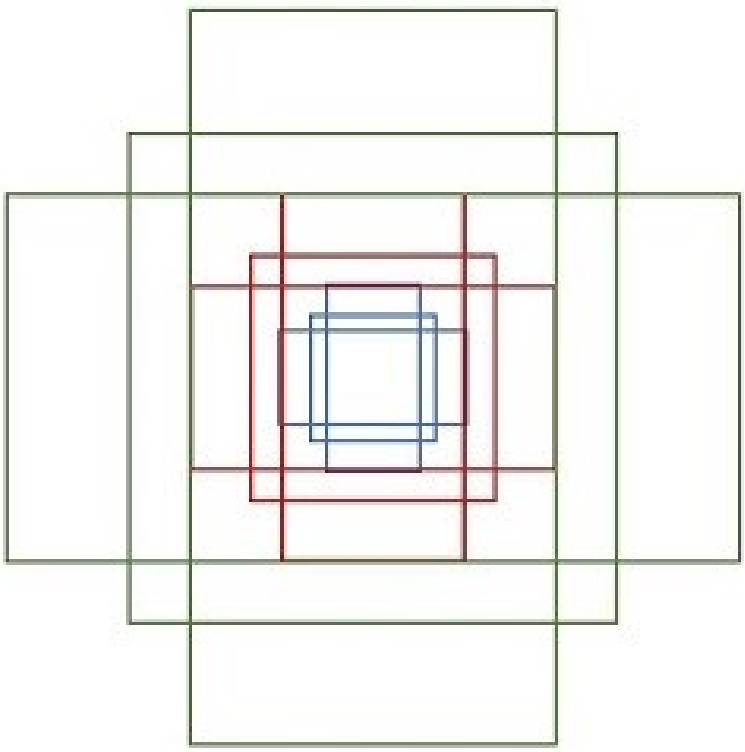

Figure 9: Anchor boxes of different size and shape. 9 in total for one anchor. Image Source:[19]

1.5 Goals

The intent of the investigation was to find a set of techniques that were network agnostic (i.e. were not dependant on a specific neural network architecture) and had minimal impacts on accuracy, while:

1. Reducing the inference time of a network

   Reduction in inference time can enable usage of network architectures in "real-time". This is vital for domains such as computer vision for object tracking where scenes can be as fast as 30 frames per second. Any reduction in inference time can enable a network architecture to get closer to real-time.

2. Reducing the memory footprint of a network

   Reduction in memory footprint can enable the usage of network architectures in resource-constrained environments where disk space or RAM is limited, such as embedded systems.

In order to achieve both goals, investigations into directly reducing the number of computations for a given network architecture were conducted. Reducing the number of computations, either by simplifying network parameters into forms that enable more computationally efficient arithmetic (i.e. integer multiplication vs float multiplication) or through the removal of low-importance parameters. These are possible ways to reduce the number of required computations while also reducing the memory footprint.

1.6 Contributions

The contributions made in this thesis were as follows:

1. **Application of class-blind pruning in object detection neural network architectures (specifically YOLOv3 and FasterRCNN)**

   To our knowledge, pruning methodologies have not been applied to object detection neural network architectures. In this thesis, it is shown that class-blind pruning is capable of removing low-importance parameters for object detection while maintaining reasonable performance [2.5.3, 2.5.4].

2. **A dynamic method of determining the number of clusters to choose when applying K-Means quantization**

   An extension to the model compression methodology described by Han et. al. in [20] (further explained in section 2.4.1) is created to enable better clustering of network weights. It is shown that using a dynamic method of clustering which determines the number of clusters required per layer in a deterministic fashion is capable of reducing the accuracy that is lost by K-Means quantization.

3. **Uncertainty quantification of network architectures before and after Knowledge Distillation**

   To our knowledge, uncertainty quantification has not been used to reason about the impacts of Knowledge Distillation on the uncertainty of predictions made by neural networks. In this thesis, the impacts of Knowledge Distillation on uncertainty in classification tasks were shown [5.3.1].

4. **Uncertainty quantification of network architectures before and after Model Compression**

   Related work has been done in the past regarding uncertainty quantification and layer-wise pruning but to our knowledge that hasn't been done so far on quantization. In this thesis, the impacts of class-blind pruning and quantization on classification tasks have been studied [5.2].

2 Model Compression

2.1 Introduction

There have been various experiments to reduce the size of a model throughout the years. Model Compression refers to the use of such techniques to reduce the inference time and the memory footprint of a network. One such example of a Model Compression pipeline would be the one in [20], which leverages the use of the techniques of network pruning and quantization. A high level representation of the compression procedure can be seen in Figure 10.

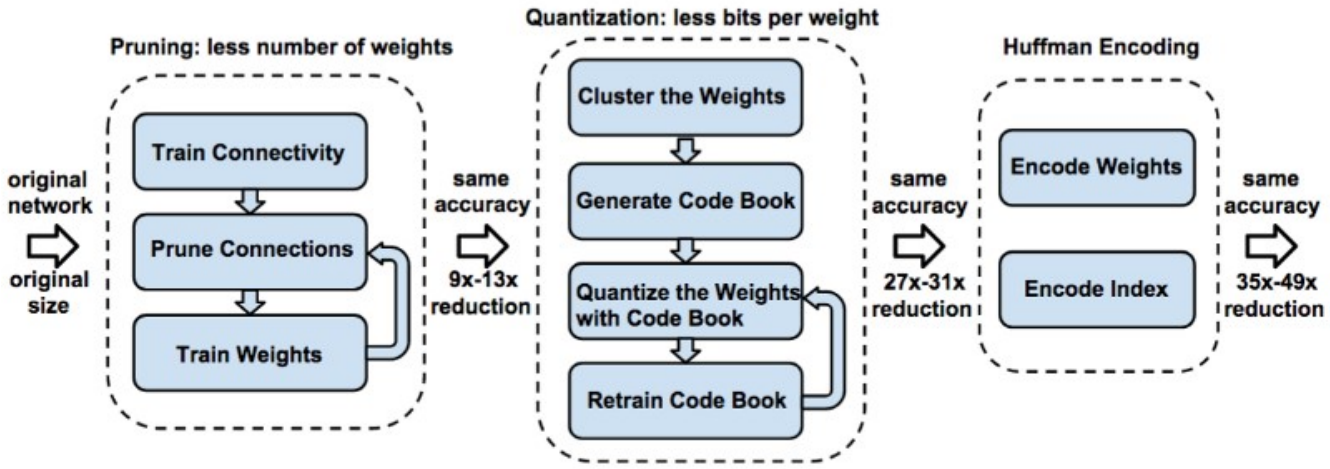

Figure 10: Deep Compression - Han et. al [20]

2.2 Pruning

Network pruning involves reducing the number of parameters of a network (e.g. weights) through their removal based on a criterion or threshold [21]. There exist several proven methods of ranking network parameters based on a variety of different criterion, from generating thresholds based on the distribution of the weights as used by Han et. al. in their Deep Compression methodology [20] (see figure 10) to more sophisticated parameter ranking techniques such as using Taylor Expansion to approximate the importance of a parameter [21].

2.2.1 Class-blind, class-uniform & class-distribution pruning

In [22], See et. al. investigated the use of three different model pruning techniques, with the goal of reducing the number of weights of neural machine translation (NMT) architectures: class-distribution, class-blind and class-uniform. They define the concept of a "weight class", subsets of all of the weights of the model (as shown in Figure 11). Essentially, these weight classes can be considered as the weights per layer (i.e. the input weights of a given layer would all be the same weight class).
Class-distribution pruning, as employed by Han. et. al. [20] defines a threshold value θ based on the standard deviation σ of the given weight class, controlled by a scaling hyperparameter λ. All weights with a magnitude lower than the threshold value are removed from the network, reducing

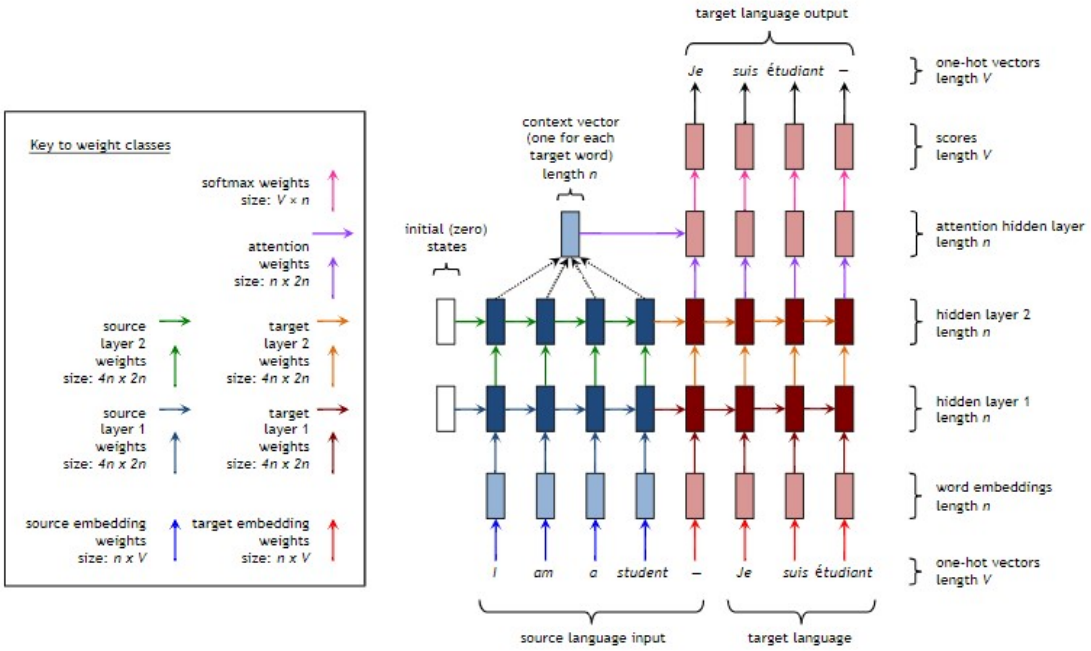

Figure 11: NMT Weight Classes - See et. al [22]

memory and inference time. Furthermore, the class-uniform method of pruning removes the same percentage of weights from each of the weight classes, such that the total number of parameters pruned is equal to the number of parameters pruned per weight class. Finally, class-blind pruning does not consider weight class at all; instead, all of the weights across the network are ordered by magnitude and the bottom percentage, as controlled by a hyperparameter, are pruned (e.g. the bottom 20% of the weights for a network).

$$\theta = \lambda\sigma$$

See et. al. [22] found the class-blind method to be the most effective in regards to the percentage pruned versus the accuracy of the pruned model. Class-blind pruning drops off the least in regards to accuracy, with class-distribution and class-uniform pruning both dropping at a sharper rate after around 30% of parameters pruned. The intuition here being that as the class-blind method compares the weights globally, it is more biased towards removing low magnitude weights than class-distribution or class-unfirom weights, which will prune a significant proportion of all weight classes regardless of their overall magnitude.

2.2.2 Post-pruning strategy

After pruning, an often conducted practise is to fine-tune the remaining weights by retraining the network to improve accuracy. In some instances, pruning has acted as a form of regularization by actually increasing the accuracy of the model [22]. Han et. al. consecutively retrain and prune the network after the pruning the model until the accuracy begins to decrease. This enables the network to retain as few weights as possible. The majority of existing pruning techniques inherit the weights from the original network after the pruning is completed and use those as the initial weights for fine-tuning.

Recent work has found the inheritance of those weights might not be completely necessary. Liu et. al. [23] investigated the use of evaluating six different pruning methodologies with both retraining from inherited weights against initializing the weights using the Parametric Rectified Linear Unit (PReLU) initialization strategy from [11]. The results showed that the pruned models retrained from the inherited weights often had lower accuracy scores than those that were trained from re-initialized weights. In many cases, starting from re-initialized weights in the pruned network actually improved the accuracy beyond that of the original network.
The results shown by [23] motivate that the value of pruning might not necessarily be in the ability to remove redundant weights; instead, it can be seen as a method of architecture search, i.e. attempting to find the best network architecture.

2.3 Quantization

Quantization refers to the act of reducing the memory required to represent a network parameter [24]. There are many quantization techniques that exist, that fall into two large categories: deterministic quantization and stochastic quantization. Deterministic quantization involves mapping a network parameter to a value such that the parameters occupy less space. Stochastic quantization involves modeling network parameters as discrete distributions and sampling from them.

2.3.1 Rounding

Rounding is an example of a deterministic quantization technique. A parameter is mapped directly to a quantized parameter that occupies less space, typically at a lower precision. For example, rounding a weight down from float64 to float32 can be seen as a form of deterministic quantization. There is a loss in accuracy of the model depending on how severe the rounding operation is. Furthermore, rounding can also destroy the training by reducing the precision of the gradients (i.e. flattening them). Therefore, many deterministic quantization algorithms keep track of the original weight value during training, and subsequently re-quantizing.

2.3.2 Vector Quantization

Vector quantization is another method of deterministic quantization. It involves clustering weights using traditional clustering techniques (e.g. KMeans) and using the centroids as the weight values. By mapping all the weights to a respective centroid, the total amount of weight values to be stored is reduced, as a low-bit index to the centroid can be used instead. The use of vector quantization techniques allows for training as well, as employed the Deep Compression method proposed by Han et. al. [20] They leveraged the use of K-Means clustering on a per-layer basis to generate centroids for the weights. During training, gradients were aggregated per centroid and simply summed and added to the centroid. This enabled the centroid to benefit from training without destroying the accuracy for the entire model. Han et. al.[20] refer to this as *trained quantization*, and fine-tune the network after quantization by employing the use of trained quantization iteratively until converging to an accuracy. This procedure is visualized in Figure 12.

2.3.3 Stochastic Quantization

In stochastic quantization, network parameters such as weights or gradients are modeled as discrete distributions, and instead sampled from a distribution. One such example of this is when the

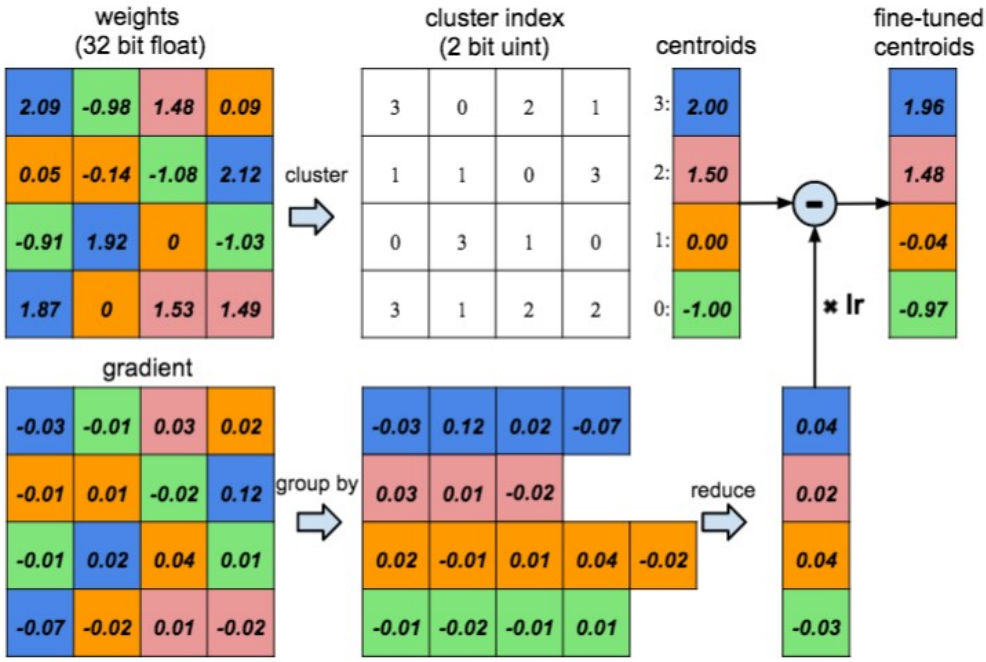

Figure 12: Vector Quantization - Han et. al [20]

parameters of these distributions are typically inferred through the use of learning algorithms such as the Expectation Back-propagation algorithm [25]. Furthermore, the use of stochastic quantization also adds a regularization effect to the model. Finally, only the distribution and its parameters have to be stored thus reducing the memory occupied by representing the parameters.

2.4 Methodology

Specifically class-blind pruning and K-Means quantization of network parameters were chosen as the techniques of Model Compression to be investigated when applied to neural network architectures within the domains of object detection and classification. Where possible, open-source implementations of popular neural network architectures (specifically FasterRCNN and YOLOv3) were forked and minimal modifications were made to the implementations to enable compatibility with the techniques developed, while maintaining the original model architecture. For all state-of-the-art architectures assessed, pre-published pre-trained weights for all of the models were used as a baseline. All implementations were done in PyTorch and CUDA.

2.4.1 Pipeline

The Model Compression pipeline consisted of firstly applying iterative class-blind pruning and then subsequently applying k-means quantization to the network.

Pruning Initially, a percentage of parameters to be pruned is assigned. This value was then used to derive a prune threshold which specifies the boundary value for parameters to be pruned. Practically, any network weight with a magnitude below this threshold was pruned. Next, the model was pruned using the threshold value, re-trained using a small number of epochs (between three and five) and set of weights producing the highest accuracy between the epochs were maintained.

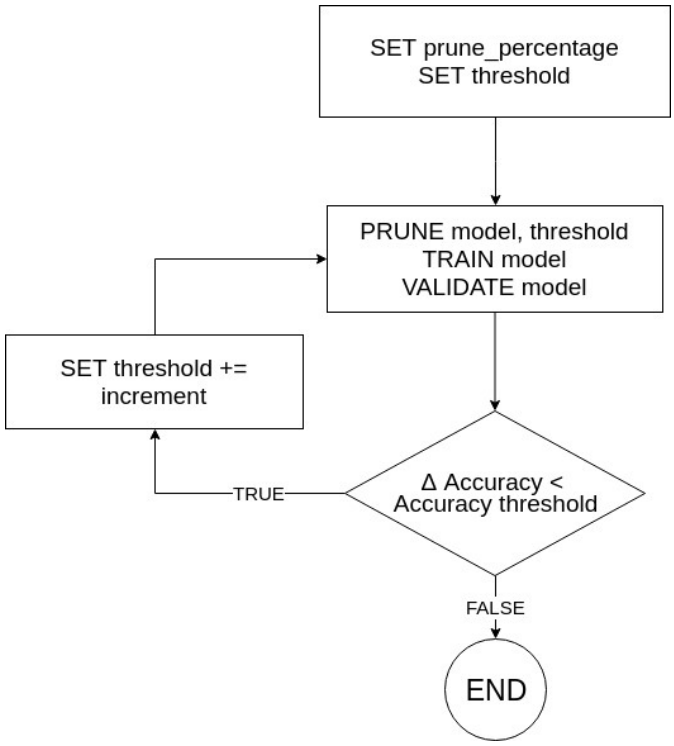

Figure 13: Iterative class-blind pruning procedure

Initial	Quantized (32 clusters)	Quantized (dynamic clustering)
0.5890	0.4400	0.5010

Table 1: mAP of YOLOv3 on COCO2014 with and without k-means quantization

Finally, the network was tested using a test data set. If the accuracy was seen to have dropped beyond that of the specified accuracy threshold, the pruning was ceased and the model was saved. However, if the accuracy threshold had not been reached, the prune percentage was incremented and more network weights were removed.

Quantization After pruning, the network was quantized using K-Means quantization. A modification made to the original K-Means quantization proposed in [20] was the number of clusters supplied to the K-Means algorithm when sorting the weights of a given layer. When the number of clusters were too low, especially for networks with a large number of weights such as YOLOv3, there was a significant decrease in the accuracy of the network. However when the number of clusters were too high, the quantization algorithm was almost unusable due to the amount of memory needed. Therefore a deterministic equation was created to dynamically scale the number of clusters based on the number of parameters within a given layer, as shown in equation 1.
Let $\mathcal{C}$ and N denote the number of clusters and parameters per set of clusters respectively. Then for each layer l with P number of parameters, the number of clusters c is given by

$$c_l = \left\lceil \frac{P_l}{N_l} \right\rceil * \mathcal{C}_l \tag{1}$$

The impacts of the use of this dynamic method of clustering can be seen in Table 1. By using a greater number of clusters for larger layers, the decrease in mAP on YOLOv3 on the COCO2014 dataset was minimized.
Furthermore, instead of utilizing a code-book in order to map the weights of clusters to a low-bit identifier later used for a lookup to the quantized weight value, the implementation presented in this thesis instead simply mapped the centroid back over the clustered weights. The reason for this was due to the decrease in speed caused by the code-book, as due to limitations in time and experience the team was unable to implement the code-book in CUDA and therefore the lookup was CPU-bound.

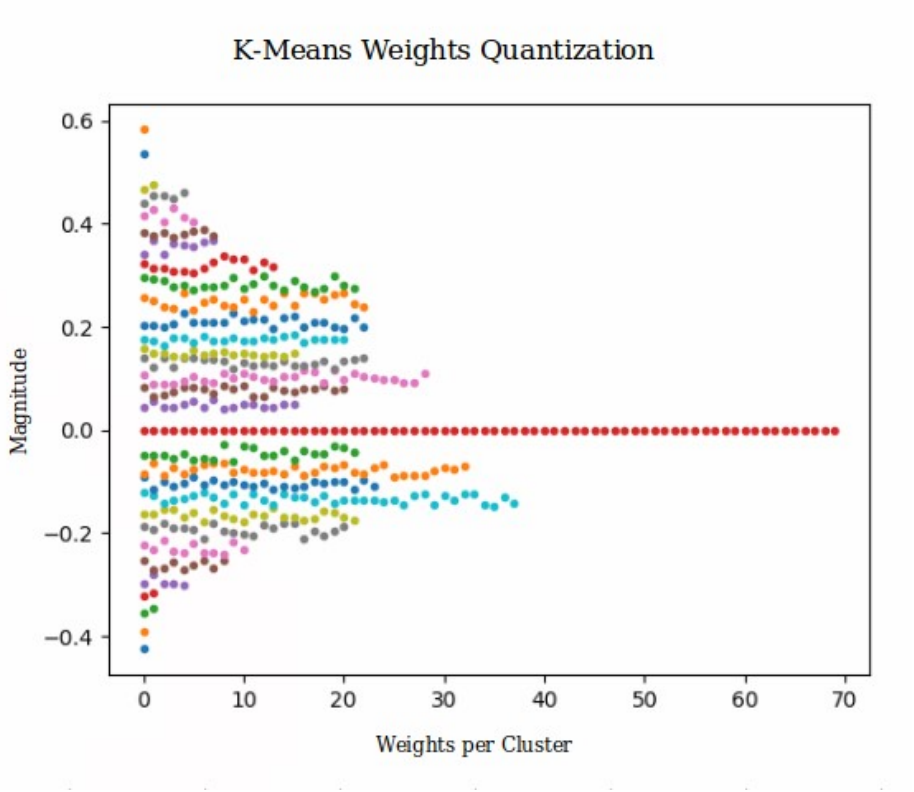

Figure 14: K-Means Quantization on MNIST Classifier

The effects of K-Means quantization *after pruning* on an early layer of the MNIST Classifier architecture is visualized in Figure [14]. The data points represent individual weight values grouped by cluster and each data point was colored with its respective cluster. As expected, there was large cluster of values with a magnitude of zero, due to the sparsification caused by pruning. The clusters were spaced equidistantly between the minimum and maximum magnitude of the available, as specified by Han et. al. in [20]. This enabled weights with extreme magnitudes to not be pulled into the nearest cluster, which could potentially cause a large change in the value. The centroid for each weight cluster was mapped back over the weight to simulate the effects of K-Means quantization.

Architecture (metric)	**Percentage Pruned**	**Initial**	**Pruned**	**Pruned Quantized**
MNist Classifier (accuracy)	95.00	0.990	0.993	0.993
CIFAR10 Shallow Classifier (accuracy)	84.68	0.587	0.623	0.621
YOLOv3 (mAP)	59.70	0.589	0.537	0.530
FasterRCNN (mAP)	26.27	0.677	0.662	0.612

Table 2: Model Compression Accuracy Results

	Memory required for network parameters (bytes)		
Architecture	**Initial**	**Pruned**	**Pruned Quantized**
MNIST Classifier	1724320	88420	9478
CIFAR10 Shallow	248024	38008	3936
YOLOv3	247586164	99036300	2239175
FasterRCNN	190270652	140390612	6472874

Table 3: Model Compression Predicted Memory Reduction Results

2.5 Results

Presented in Table 2 are the performances of four different model architectures, two classifiers and two object detectors, at different stages of the pruning pipeline as shown in Figure 13. The *Initial* column presents the accuracy or mAP of the architecture using pre-trained weights before the model compression pipeline is applied. The *Pruned* column presents the accuracy or mAP after the pruning step of the pipeline. The *Pruned Quantized* column presents the final accuracy or mAP after the entire model compression procedure has been applied.
Furthermore, presented in Table 3 are the predicted memory requirements for the model at different stages. All network parameters are assumed to be float32 tensors occupying four bytes. For the results presented at the *Pruned* step, all the weights that have been removed from the model are excluded. For the *Pruned Quantized* step, the compression ratio for the parameters remaining is calculated using the following formula, as provided by Han et. al. in [20]. Per layer, given k clusters, $log_2(k)$ bits are required to encode the indices for the references to the cluster centroids. Given n weights, with each connection being represented by b bits and given that only k full precision weight values will be stored, the compression rate r is given by:

$$r = \frac{nb}{nlog_2(k) + kb}$$

Using this compression ratio, the number of bytes occupied by the quantized model was estimated. Both the classifier and object detection architectures were pruned using the aforementioned prune pipeline (see Figure 13). Presented in the following subsections are the percentage of parameters (weights) pruned and the accuracy after the retraining step had completed.

2.5.1 MNist Classifier

The dataset used for training and evaluation was the MNIST dataset, a popular hand-writing classification dataset [9]. As shown in Figure 15 the MNIST classifier architecture saw very little drop in accuracy through pruning. Counter-intuitively, the MNIST classifier actually began to

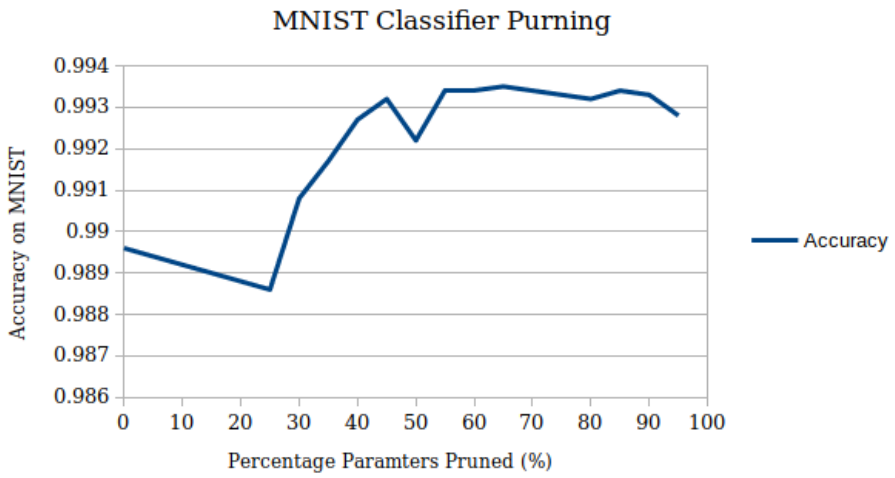

Figure 15: MNIST Pruning Results

increase in accuracy, all the way up to 95% of parameters pruned. The final difference between the initial accuracy of the trained-from-scratch model and the pruned model at 95% pruning was +0.003, indicating a highly over-parameterized model. Furthermore, the increase in accuracy could be attributed to the additional training conducted on the sparse model, or could be a symptom of over-fitting to the data. At a 100% prune rate the accuracy drops to 0.1, as expected as there are ten classes to predict. This indicates that the model was not making any meaningful predictions at a 100% prune rate. Therefore the result at 100% prune percentage was omitted from the figure. Furthermore, after quantizing even at 95% pruning there was essentially no loss in accuracy, as seen in Table 1.

2.5.2 CIFAR Classifier

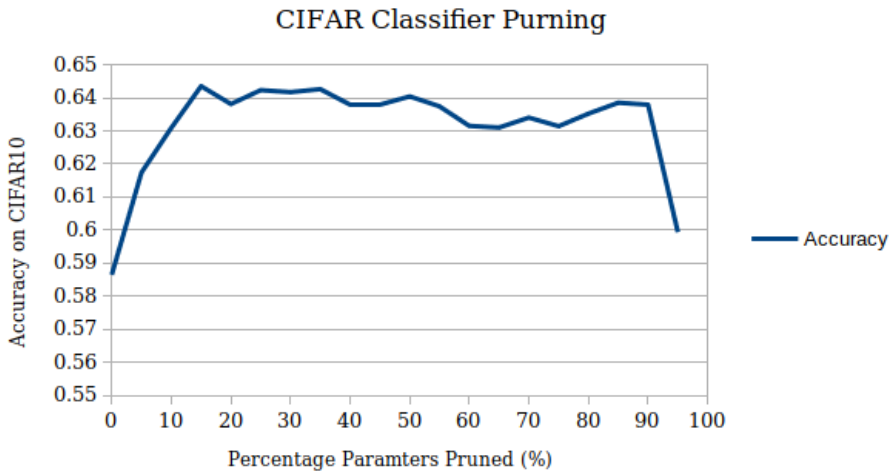

Figure 16: CIFAR Pruning Results

The dataset used for training and evaluation was the CIFAR10 dataset consisting of 10 classes

[26]. Similar to the MNIST classifier architecture, the CIFAR classifier also experiences essentially no drop in performance when being pruned as shown in Figure 16. There is also a slight increase initial increase in accuracy which remains until approximately 40% of the parameters have been pruned. At its peak, the CIFAR classifier managed an accuracy increase of approximately 0.056 at a pruning percentage of 25%. Once again, this indicates a highly over-parameterized model of which the ideal network architecture can be much sparser. At 100% pruning the accuracy falls to 0.1, which indicates that the model is not making any meaningful predictions. Therefore the result at 100% pruning was omitted from the figure.
When the full Model Compression pipeline was applied, the percentage of parameters pruned were 84.68% with an accuracy increase of 0.036. The subsequent quantization step inferred a small loss in that accuracy of 0.01, however the accuracy was still improved by 0.034 from the original model.

2.5.3 YOLOv3

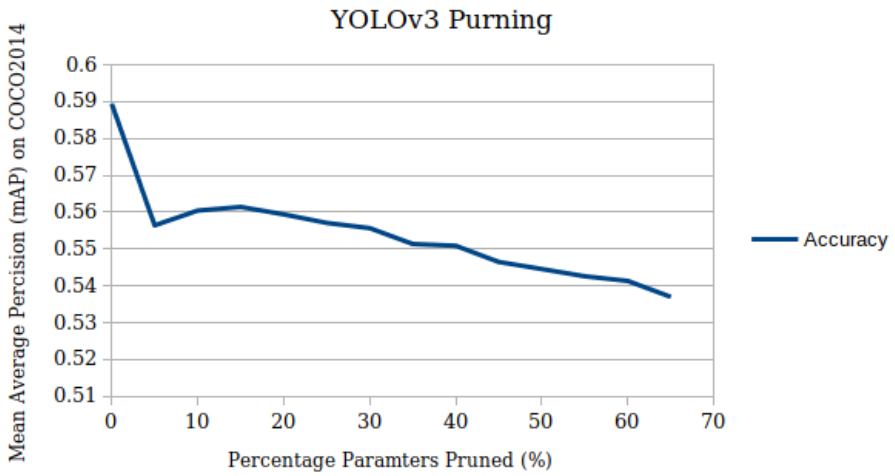

Figure 17: YOLOv3 Pruning Results

The dataset used for training and evaluation was the COCO2014 dataset, a popular dataset for object detection [16]. Having applied the prune pipeline to YOLOv3, as shown in Figure 17, with a prune threshold of 0.05, the model was pruned to 65% sparsity before the threshold was reached. The initial 5% pruned caused the steepest decline in the mAP, indicating the potential loss of important weights with low magnitudes. However, subsequent pruning iterations saw small increases in accuracy before a slow decline until the 65% mark. The large number of parameters pruned with a relatively small impact on the mAP indicates that YOLOv3 is overparameterised and sparsification can lead to a faster, smaller model. Furthermore, it is worth noting that after every pruning iteration the model was retrained for three epochs. It is possible that significantly increasing the number of retraining epochs might minimize the drop in mAP.
When undergoing the full Model Compression pipeline, a prune percentage of 59.70% was applied as it had the best weights for the given compression run. Subsequently, the quantization of the network resulted in a minimal decrease in the mAP of the model. The overall loss in mAP through the entire Model Compression pipeline was 0.059.

2.5.4 FasterRCNN

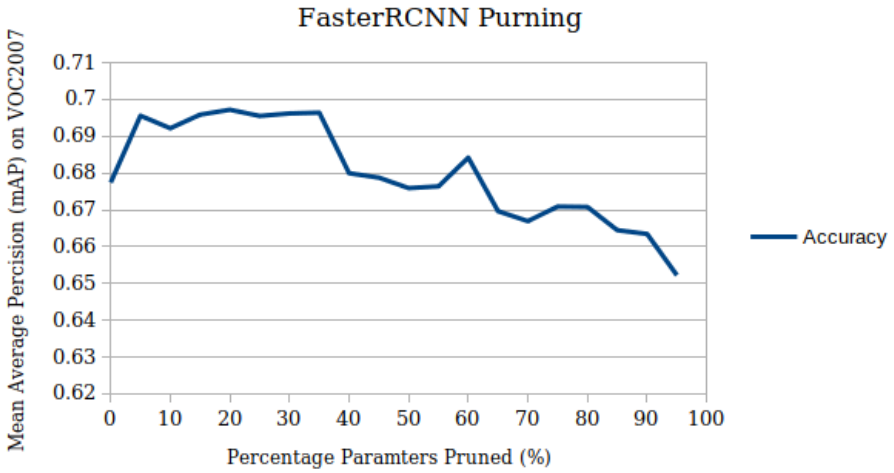

Figure 18: FasterRCNN Pruning Results

The dataset used for training and evaluation was the VOC2007 dataset [27], a popular dataset for object detection. Within the FasterRCNN architecture only the base (detector) of the network was pruned, which represents approximately 56% of the entire network weights. Figure 18 indicates the percentage of parameters *of only the base of the network*. Unlike YOLOv3, when pruning FasterRCNN there was very little drop in accuracy. In fact, the iterative pruning methodology pruned the detector to 100% parameters pruned before the initial prune thresholds specified (0.05) were reached. This indicates that the detector is heavily over-parameterised, with the network having a drop in mAP of 0.025 at 95% parameters pruned. The peak mAP reached was at 20% parameters pruned with an increase in mAP of approximately 0.02. As the prune percentage was incremented units of 5%, it is entirely possible that a finer interval could result in an even higher accuracy. At a pruning percentage of 100% the mAP dropped to 0.1, which indicates that the model is not making any meaningful predictions. Therefore the result at 100% pruning was omitted from the figure.
Under the full Model Compression pipeline a prune percentage of 26.27% (approximately 46.70% of the weights of the base) was applied and the subsequent quantization resulted in an overall decrease in mAP of 0.065. The prune percentage was applied as it resulted in the highest mAP during the compression run, and the quantization step caused the largest decrease in mAP.

2.6 Discussion

The successful application of the Model Compression pipeline towards neural network architectures used for at object detection and classification indicates the potential of such techniques in reducing inference time and improving the energy efficiency of neural networks.
The use of quantization and pruning can be seen as managing a tradeoff between performance and accuracy; the greedier the pruning (i.e. high prune percentages) and quantization strategies applied (i.e. quantization into fewer clusters or lower bits) the greater the loss in performance at the expense of reduced computation. The application of such methodologies should attempt to

address the usage context of the generated network before settling on how much the models should be compressed.

Pruning often caused a performance increase at different percentages for different architectures. This is likely due to the removal of low-importance weights that simply interfere with network predictions, however often the percentage of parameters pruned were very high when the performance increase manifested. The regularization theory as stated by Han et. al. [20] appears to hold true even for high-complexity models such as FasterRCNN.

Model Compression in the two object detection architectures chosen proved to be very promising, with the detector of the FasterRCNN being almost entirely prunable with minimal impact to accuracy. Furthermore, YOLOv3 was pruned to quite a significant extent with very little drop in mAP, with up to 50% of its parameters pruned while maintaining not breaking the threshold of 0.05 mAP. The use of Model Compression in advanced architectures such as these can enable execution on resource constrained environments or real-time scenarios.

Han et. al. demonstrated in [20] that the sparsification (i.e. pruning) of weight matrices leads directly to lower inference times. This is because fewer floating point operations have to be conducted and fewer gradients having to be computed due to due to fewer weights being involved with inference. However, due to limitations with PyTorch and CUDA regarding the use of sparse matrices, we were unable to directly collect the inference time improvements as a result of pruning. However it still holds true that, for example, pruning 50% of network parameters will lead to a proportionate 50% drop in inference time for forward and backward propagation, as shown in [20]. Further investigations implementing the investigation in an alternative framework such as TensorFlow where sparse matrices are available can be valuable in further solidifying this claim.

3 Knowledge Distillation

3.1 Introduction

Knowledge Distillation (Figure:[19]) is a Model Compression technique that is used to improve the computational efficiency of deep learning models on devices with limited hardware resources by having a limited impact on accuracy. Initially, a deep and complex network (teacher) is trained which can extract important features from given data and can produce good predictions. Then, a smaller network (student) is trained with a combination of the original training set and the distilled knowledge (or dark knowledge) from a teacher network. Thus, the student is able to produce improved results, better than just training the network on the dataset. In this section, Knowledge Distillation will be explored for image classification on CIFAR-10[26] and MNIST[9] datasets and for object detection on PASCAL-VOC2007[27].

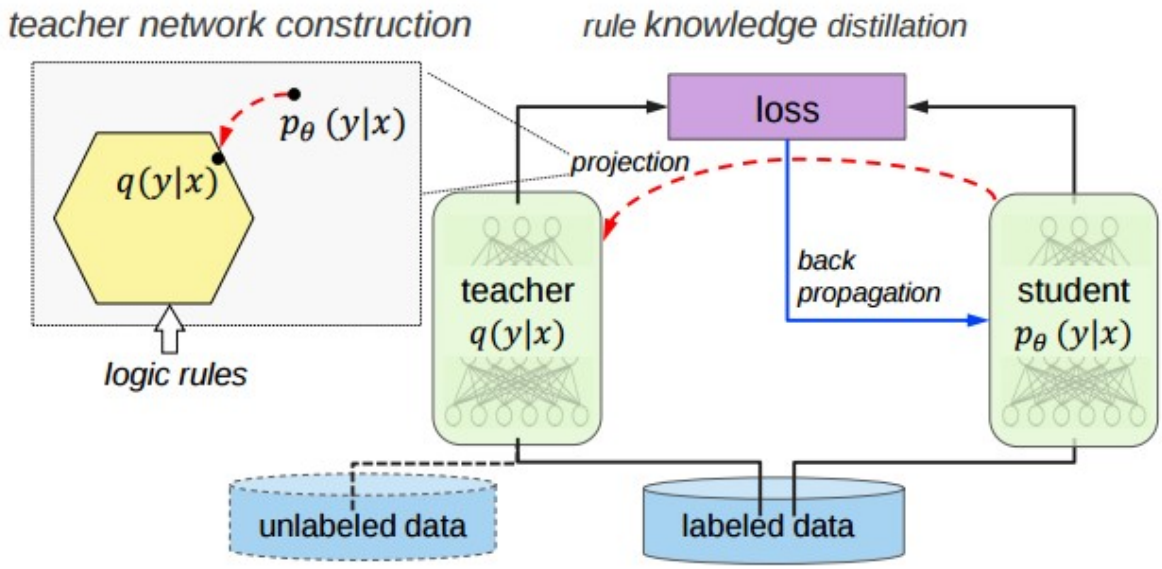

Figure 19: Knowledge Distillation - Zhiting et. al. [28]

3.2 Classification

Transferring the knowledge that is already acquired from the teacher network to a student network can be done by the use of class probabilities produced by the teacher model as soft targets for training the small model. For this knowledge transfer stage, the same training set used for the teacher model is employed for the student model (or at least a subset). When the soft targets have high entropy, they provide much more information per training case than that of hard targets and much less variance in the gradient between training cases. Thus, the student can often be trained on much less data than the teacher while using a much higher learning rate.
To distill the knowledge that the student network gains from the teacher network, logits are used (the inputs to activation function of the output layer). This is done by minimizing the squared difference between the logits produced by the teacher model and the logits produced by the student

model. The aforementioned logits are fed into a modified activation function within the student network and training is conducted using the produced probabilities. In the case of a classification problem with the softmax activation function, the following modification is used:

$$P_t(\alpha) = \frac{\exp(q_t(\alpha)/T)}{\sum_{i=1}^{n} \exp(q_t(i)/T)} \tag{2}$$

At $T = 1$ the activation function behaves as a regular softmax, resulting in the outputs being almost identical to those of the teacher model. For high values of T, all outputs have nearly the same probability. Alternatively, lower values when $T \approx 0$ cause the more expected rewards affect the probability (i.e. the probability of the output with the highest expected value tends to 1).
Therefore, the value of T of the final softmax is raised until the teacher network produces a suitably soft set of targets. Then the same T is used while training the student model to match these soft targets and thus, with higher values of T, it is possible to produce a "softer" probability distribution over the classes. Afterwards, once the student model is trained, the regular softmax ($T = 1$) is used in the student model for real predictions.
A new loss function that will be minimized in order to transfer the knowledge from the teacher to the student model is necessary as well:

$$\mathcal{L}_{KD}(\mathbf{W_{student}}) = \lambda T^2 KL(y_s^{kd}, y_t^{kd}) + (1 - \lambda)\, CrossEntropy(y_s, y_{target})$$

where y_s^{kd} and y_t^{kd} are the softened outputs from the student and the teacher respectively using the same value for T and KL is the KullbackLeibler divergence. The λ hyperparameter is responsible to tune the influence of the teacher to the student. The first part of the function measures to what extent the student will lean towards the softened softmax distributions whereas the second forces the optimization towards the targeted labels. For example with $\lambda = 0$ the loss function will take the form the original *CrossEntropy* and for $\lambda = 1$ corresponds using only the dark knowledge from the teacher without being influenced at all from the labeled data.

3.2.1 Advantages

There are several advantages of applying the Knowledge Distillation:

- Less compute requirements and superior performance under production constraints
- Better accuracy than a stand-alone small model
- Does not require a large dataset available for the student network as long as the teacher is well trained

3.2.2 Drawbacks in Object Detection

Knowledge Distillation as a technique is applied globally to the predictions of an object detector and can, in some instances, boost the background errors and produce worse results than the model were trained from scratch and fine tuned without the use of KD.

3.2.3 Methodology

In a recent paper by Mirzadeh et al.[29] (Figure:[20]) it has been suggested that Knowledge Distillation works better when the gap between the teacher and the student model size is not too large. Following their approach, three models were implemented. A shallow model as a student, deep model that acts as the teacher and can achieve considerably higher accuracy than the student and a model of intermediate size and accuracy that is referred to as the teacher assistant (TA). Thus, in order to maximize the effects of Knowledge Distillation:

1. Train all three models on the dataset
2. Apply Knowledge Distillation on the teacher and the TA.
3. Apply Knowledge Distillation on the TA and the student (the TA acts as the teacher here)

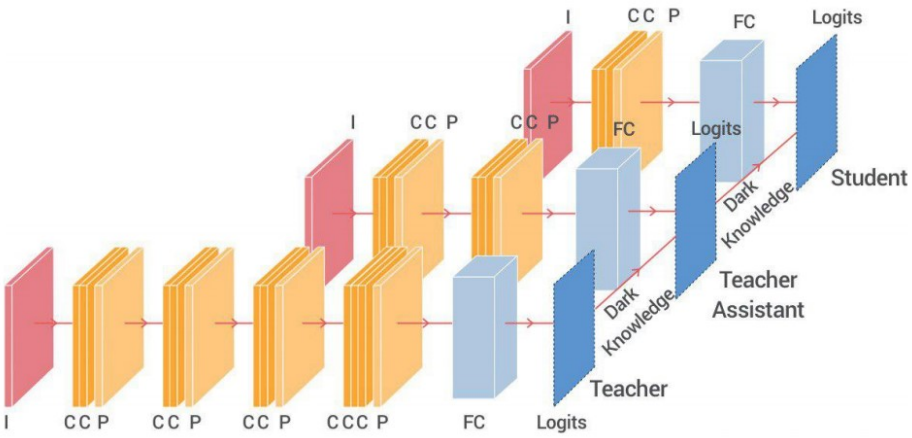

Figure 20: Knowledge Distillation graph with Teacher Assistant, where I is the input, C stands for Convolutional Layer, P for Pooling and FC for Fully Connected layers. Source:[[29]]

3.2.4 Results

Knowledge Distillation was applied on two different datasets. One is the dogbreeds dataset which has 120 different classes with ResNet18 as the student, ResNet34 as the TA and finally ResNet50 as the teacher. The pretrained ResNet models hosted by PyTorch were used and fine-tuning was applied on the dog-breeds dataset. The other dataset was CIFAR10 where the plain CIFAR10 classifier was used as the student, ResNet8 as the TA and ResNet18 as the student. All those models are trained from scratch, using learning rate decay per 100 epochs starting from 0.1 and ending with 0.001. In both cases stochastic gradient descent was used with momentum 0.9 and weight decay equal to $1e-4$. The hyper parameters T and λ that are responsible for the target softening and the amount of dark knowledge that will pass from the student to the teacher were hand tuned to $T = 1.5$ and $\lambda = 0.5$. This is because the amount of data that were available in both datasets were enough to give confident predictions in the teacher and thus the temperature T was that low in comparison with the experiments in the original paper.

Accuracy	Teacher	TA	Student
Before KD	0.9415	0.9283	0.9205
After KD	0.9415	0.9293	0.9275

Table 4: Knowledge Distillation CIFAR Results

Accuracy	Teacher	TA	Student
Before KD	0.8347	0.8014	0.7513
After KD	0.8347	0.8136	0.7770

Table 5: Knowledge Distillation Dog Breeds Results

The training using Knowledge Distillation from the teacher to the assistant and from the assistant to the student, was done in 50 epochs since the models were already pretrained individually in both datasets. From the results it can be seen that Knowledge Distillation improved both the accuracy of the student and the accuracy of the teacher assistant by $\sim$ 2% in the dogbreeds dataset (Table:[5], Figure: [21] [22]). While the increase is not that high in the CIFAR10 dataset, which is expected since all three models' predictions are higher and closer to each other than their dogbreeds counterparts, it can still be observed that there is an increase in the accuracy of about $\sim$ 1% (Figure:[4]).

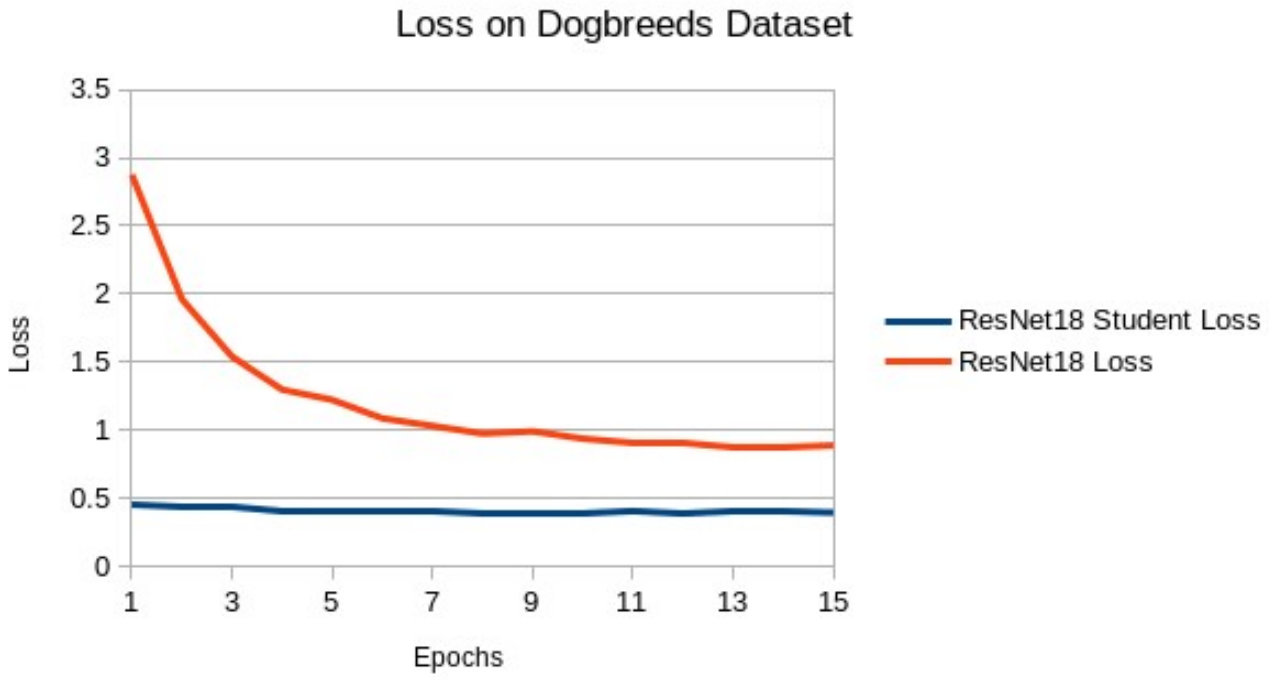

Figure 21: ResNet18 student provided lower loss in comparison to a fine-tuned ResNet18 from Imagenet

3.3 Discussion

Knowledge Distillation is a technique that is used to improve the predictions with the help of another model that is more accurate. It is not meant to be used to mimic the predictions of a deep

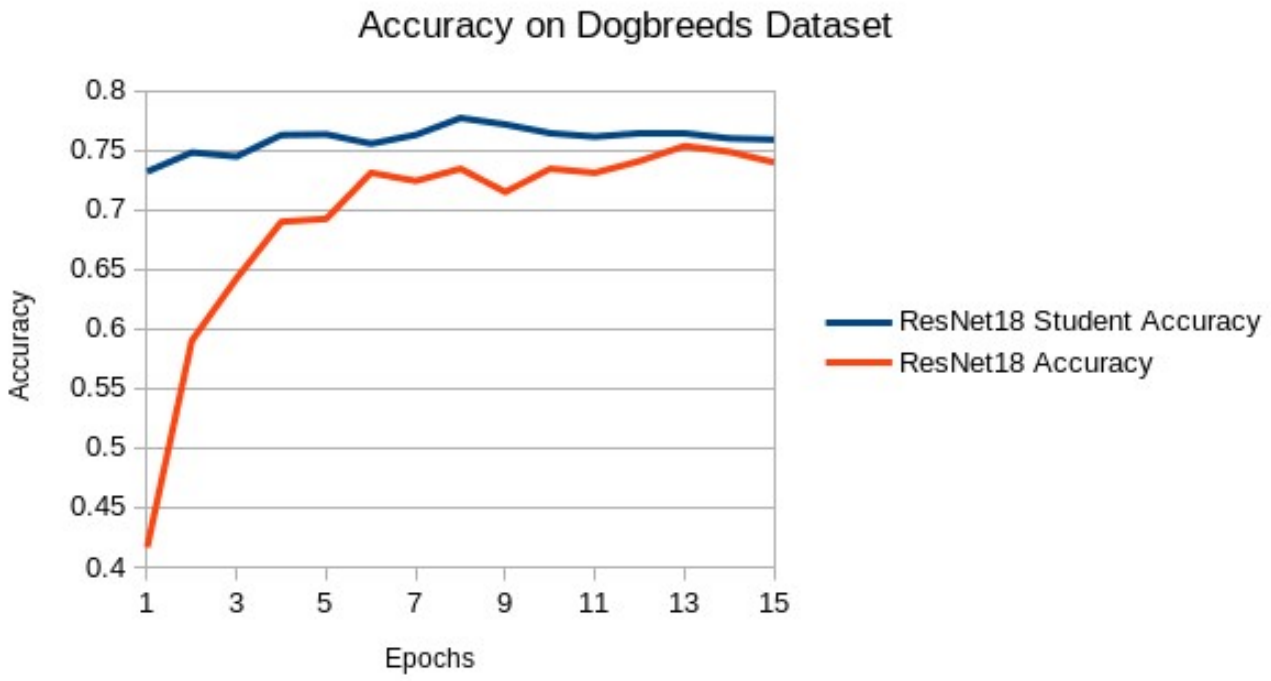

Figure 22: ResNet18 student predictions were more accurate in comparison to a fine-tuned ResNet18 from Imagenet

and cumbersome model but rather be influenced by it. Also, the outcome is consistent since it has improved the student's accuracy in every classification task that was used. On the downside, Knowledge Distillation cannot be used to Object Detection, since its conception in the first place was dependent to the modified softmax (equation 2).
From a technical point of view, Knowledge Distillation is very simple to use since the only difference from a standard training is the introduction of a new Loss function while its very flexible since there was no difference in the results using it before or after training a model, thus making it possible to be applied into well trained models.

4 Bayesian Deep Learning

4.1 Introduction

Convolutional neural networks for image processing and recurrent neural networks for sequence processing models are both extensively used in a variety of domains. These approaches use regularization tools such as dropout to improve their performance. However, fields such as physics, biology, and manufacturing are ones in which representing model uncertainty is of crucial importance. With the recent shift in many of these fields towards the use of Bayesian uncertainty new needs arise from deep learning.
To the best of our knowledge, there are two main approaches regarding Bayesian Deep Learning. One is by Yarin Gal[30] where dropout is formulated as a Bernoulli distribution and is used as a means of variational inference to infer the posterior weight distribution of the network. The other approach is Bayes by backprop[31] which was introduced by Blundell where the weights of the model are computed as functions of a mean value and a variance (similar to stochastic quantization). The focus within the investigation was the variational dropout method due to its popularity and impressive results in various neural network architectures.

4.2 Variational inference

In classical machine learning methods, given that parameters are random variables, for given input data $X = \{X_i\}_{i=1}^N$ and labels $Y = \{Y_i\}_{i=1}^N$ there exist a:

- prior $\omega : P(\omega)$
- likelihood $P(Y|\omega, X)$
- posterior $P(\omega|X, Y) = \frac{P(Y|\omega,X)P(\omega)}{P(Y|\omega)}$
- predictive distribution given new input x^*

$$P(y^*|x^*, X, Y) = \int P(y^*|x^*, \omega)P(\omega|X, Y)d\omega$$

As in real world applications, the posterior distribution is intractable due to the very large number of parameters. A solution to this problem is to approximate the posterior distribution with a simple known distribution $q_\theta(\omega)$. This is possible by using the Kullback-Leibler (KL) (Figure:[23]) divergence to minimize the distance from the posterior:

$$KL(q||p^*) = \sum_x q(x) \log \frac{q(x)}{p^*(x)}$$

$$\begin{aligned}
\theta^* &= argmin_\theta \left\{ KL(q_\theta(\omega)||p(\omega|X, Y))\right\} = argmin_\theta \{ \int q_\theta(\omega) \log \frac{q_\theta(\omega)p(X, Y)}{p(X, Y|\omega)p(\omega)} d\omega \Big\} = \\
&= argmin_\theta \left\{ \int q_\theta(\omega) log \frac{q_\theta(\omega)}{p(\omega)} d\omega - \int q_\theta(\omega) \log(p(X, Y|\omega))d\omega \right\} = \\
&= argmin_\theta \left\{ KL(q_\theta(\omega)||p(\omega)) - \mathbb{E}_{q_\theta(\omega)}[\log(p(X, Y|\omega))] \right\}
\end{aligned}$$

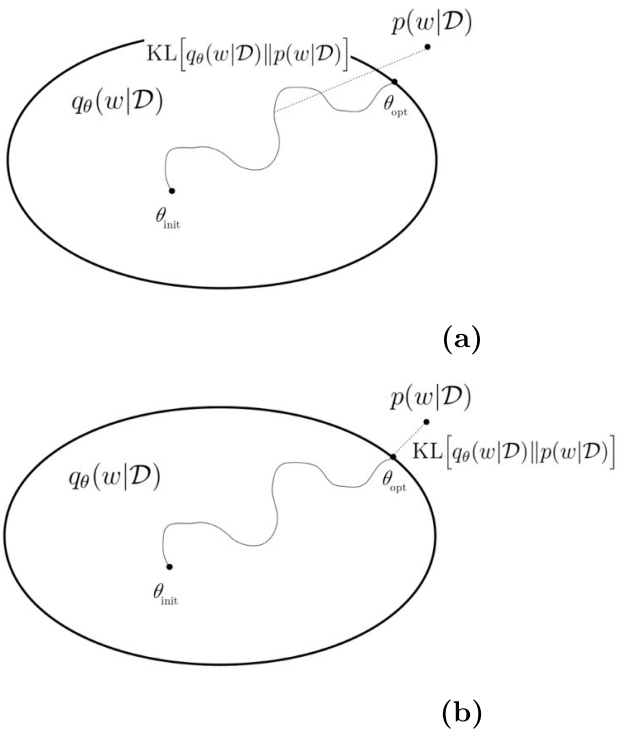

Figure 23: A graphical intuition of KL-Divergence with unoptimized and optimized θ on top and bottom respectively, where $\mathcal{D} = (X, Y)$ [32]

Where the expected log-likelihood measures how well the samples from the approximate posterior explain the data and the KL-divergence ensures that this explanation does not deviate too far from the prior of our beliefs. Finally the predictive distribution is approximated with:

$$q_\theta(y^*, x^*) = \int P(y^*|x^*, \omega) q_\theta(\omega) d\omega$$

4.3 Dropout as Variational Inference

The approximate inference in Bayesian Neural Networks, follows the same steps as it is shown in the previous section. $q_\theta(\omega)$ is defined to approximate a posterior $p(\omega|X, Y)$. Then the KL-divergence is used to minimize:

$$KL(q_\theta(\omega)||p(\omega|X, Y)) \propto -\int q(\omega) \log(p(X, Y|\omega)) d\omega + KL(q_\theta(\omega)||p(\omega)) := \mathcal{L}(\theta)$$

The log-likelihood integral is very hard to compute, so it is approximated with Monte-Carlo integration $\hat{\omega} = q_\theta(\omega) : \hat{\mathcal{L}}(\theta) = -\log(p(Y|X, \hat{\omega}) + KL(q_\theta(\omega)||p(\omega))$.
This is an unbiased estimator $\mathbb{E}_{\hat{\omega} \sim q_\theta(\omega)} = \mathcal{L}(\theta)$ which converges in the same optima as $\mathcal{L}(\theta)$. To infer repeat:

- Sample $\hat{\omega} \sim q_\theta(\omega)$
- Minimize(one step):

$$\hat{\mathcal{L}}(\theta) = -\log(p(Y|X, \hat{\omega}) + KL(q_\theta(\omega)||p(\omega))$$

with respect to θ.
The $q(\cdot)$ given $\mathbf{z}_{i,j}$ Bernoulli random variables and variational parameters $\theta = \{\mathbf{M}_i\}_{i=1}^{L}$ (set of matrices) is specified:

- $\mathbf{z}_{i,j} \sim \text{Bernoulli}(p_i)$ for $i = 1, \ldots, L$, $j = 1, \ldots, K_{i-1}$
- $\mathbf{W}_i = \mathbf{M}_i \cdot \text{diag}([\mathbf{z}_{i,j}]_{j=1}^{K_i})$
- $q_\theta = \prod q_{\mathbf{M}_i}(\mathbf{W}_i)$

In summary to minimize the divergence between $q_\theta(\omega)$ and $p(\omega|X, Y)$:

- Sample $\mathbf{z}_{i,j} \sim \text{Bernoulli}(p_i)$ and set:

$$\hat{\mathbf{W}}_i = \mathbf{M}_i \cdot \text{diag}([\hat{\mathbf{z}}_{i,j}]_{j=1}^{K_i})$$

$$\hat{\omega} = \{\hat{\mathbf{W}}_i\}_{i=1}^{L}$$

- Minimize(one step)

$$\hat{\mathcal{L}}(\theta) = -\log(p(Y|X, \hat{\omega}) + KL(q_\theta(\omega)||p(\omega))$$

with respect to $\theta = \{\mathbf{M}_i\}_{i=1}^{L}$(set of matrices)
This is equivalent to:

- Randomly set columns of $\mathbf{M}_i$ or units of the network to zero
- Minimize(one step):

$$\hat{\mathcal{L}}(\theta) = -\log(p(Y|X, \hat{\omega}) + KL(q_\theta(\omega)||p(\omega))$$

with respect to $\theta = \{\mathbf{M}_i\}_{i=1}^{L}$(set of matrices). Thus it is shown that dropout can be modeled as a variational inference to the posterior distribution.
For the convolutional layers the procedure is very similar. A prior distribution is placed over each kernel and used to approximately integrate each kernels-patch pair with Bernoulli variational distributions. Then Bernoulli random variables $\mathbf{z}_{i,j,h}$ are sampled and for patch n they are multiplied by the weight matrix $\mathbf{W}_i \cdot \text{diag}([\mathbf{z}_{i,j,n}]_{j=1}^{K_i})$. This is equivalent to approximating a distribution modelling each kernel-patch pair with a distinct random variable, tying the means of the random variables over the patches. This distribution randomly sets kernels to zero for different patches. This is also equivalent to applying dropout for each element in the tensor $\mathbf{y}$ before pooling. Implementing our Bayesian CNN is therefore just using dropout after every convolution layer before or after pooling, depending whether the model is using an Encoder-Decoder architecture or not. An example of an Encoder-Decoder model, where dropout is used after pooling and close to the bottleneck, and produces better results is Bayesian Segnet[33] (Figure:[24]).

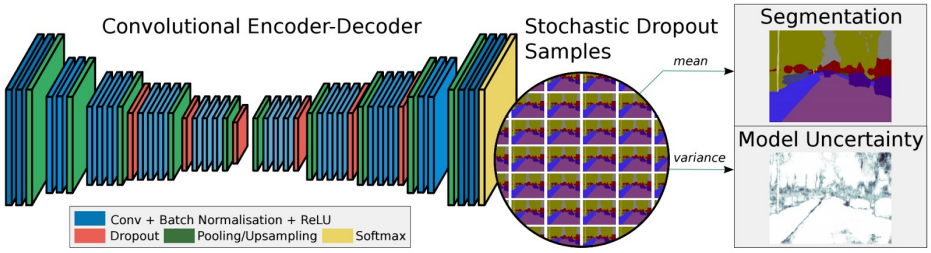

Figure 24: Bayesian Segnet[33] architecture using variational dropout and estimating the uncertainty of an image

4.4 Concrete Dropout

In summary, the variational interpretation of dropout is seen as approximating a distribution $q_\theta(\omega)$ to the posterior in a Bayesian neural network with a set of random weight matrices $\omega = \{\hat{\mathbf{W}}_i\}_{i=1}^L$ with L layers and θ the set of variational parameters. The optimization function is represented as:

$$\hat{\mathcal{L}}_{MC}(\theta) = -\frac{1}{M}\sum_{i\in S}\log(p(y_i|f^\omega(x_i))) + \frac{1}{N}KL(q_\theta(\omega)||p(\omega)) \tag{3}$$

with θ parameters to optimise, N the number of data points, S a random set of M data points, $f^\omega(x_i)$ the neural network's output on input x_i when evaluated with weight matrices realisation ω and $p(y_i|f^\omega(x_i))$ the model's likelihood, e.g. a Gaussian with mean $f^\omega(x_i)$. The KL-divergence is a regularization term which ensures that the approximate posterior distribution $q_\theta(\omega)$ does not deviate too far from the prior distribution $p_\theta(\omega)$. Assuming that the set of variational parameters for the dropout distribution satisfies $\theta = \{\mathbf{M}_i\}_{i=1}^L$ set of mean weight matrices and dropout out probabilities such that $q_\theta(\omega) = \prod q_{\mathbf{M}_i}(\mathbf{W}_i)$ and $q_{M_i}(\mathbf{W}_i) = M_i \cdot \text{diag}([\mathbf{z}_{i,j}]_{j=1}^{K_i})$ where $\mathbf{z}_{i,j} \sim Bernoulli(p_i)$ for a single random weight matrix $\mathbf{W}_i$ of dimension $K_{i+1} \times K_i$. The KL-divergence can be approximated by:

$$KL(q_\theta(\omega)||p(\omega)) = \sum_{i=1}^{L} KL(q_{M_i}(\mathbf{W}_i)||p(\mathbf{W}_i)) \tag{4}$$

$$KL(q_M(\mathbf{W})||p(\mathbf{W})) \propto \frac{i^2(1-p}{2}||\mathbf{M}||^2 - K\mathcal{H}(p) \tag{5}$$

where

$$\mathcal{H}(p) := -p\log(p) - (1-p)\log(1-p) \tag{6}$$

is the entropy of a Bernoulli random variable with probability p which can be seen as a dropout regularisation term.

One of the difficulties with the approach above is that grid-searching over the dropout probability can be expensive and time consuming, especially when done with large models. This is especially difficult in when one is operating in a continuous learning setting such as reinforcement learning. So it is of great need that the droupout probability p is not handtuned but learnable through optimization.

Minimising the KL divergence between $q_M(\mathbf{W})$ and the prior is equivalent to maximising the $\mathcal{H}(p)$ in equation [6]. The scaling regularisation term means that the large models will push the dropout probability towards 0.5, which is the highest it can attain, more often than the smaller models but as the amount of data N is increased the dropout probability will be pushed towards 0 because of the first term in equation [3].

Trying to optimize equation [3] w.r.t. the parameter p using the re-parametrization trick, leads to an increase of the variance of the predictions which is not manageable. A solution to this problem was suggested by Kingma et.al. [34] using Gaussian dropout. However unlike the Gaussian dropout, the intent is to optimize the parameter of a Bernoulli distribution. The pathwise derivative estimator assumes that the distribution at hand can be re-parametrised in the form $g(\theta, \epsilon)$

with θ the distribution parameters, and ϵ a random variable which does not depend on θ. This cannot be done with the Bernoulli distribution.
To deal with the this problem, Gal[35] suggested to replace the Bernoulli by using the Concrete distribution relaxation which is a continuous distribution and is used to approximate discreet random variables. So instead of sampling from the discrete Bernoulli distribution, the samples are realised from the Concrete distribution with some temperature t which results in values in the interval $[0, 1]$. Thus the for the Concrete relaxation of the Bernoulli variable $\mathbf{z}$, $\hat{\mathbf{z}} = g(\theta, \epsilon)$, where ϵ does not depend on θ, and can be reduced to a simple sigmoid distribution:

$$\hat{\mathbf{z}} = \text{sigmoid}\left(\frac{1}{t}\left(\log(p) - \log(1-p) + \log(u) - \log(1-u)\right)\right) \tag{7}$$

with $u \sim Uniform(0, 1)$. With the Concrete relaxation of the dropout masks, it is now possible to optimise the dropout probability using the re-parametrization trick.

4.5 Methodology

In practice, in order to make use of the variational properties of dropout, first the model must be trained by using dropout. Then during inference, instead of turning the dropout off, it was kept activated (unlike the typical case where dropout is not used in normal inference) and several Monte Carlo predictions made for each minibatch. The mean for each training sample was the final prediction from which the uncertainty can be computed. The dropout used for the convolutional layers was responsible for dropping kernels while the one used in the fully connected layers was dropping neural connections. An overview of the describe method can be seen in Algorithm [1]

input : data x, labels y, base model Conv$(\cdot)$, feature extractor FC$(\cdot)$
output : prediction $\hat{y}_{mc}$, model uncertainty difference from the ground truth η
parameter: initial dropout probability p, Monte Carlo samples *Reps*

for $i = 1$ **to** *Reps* **do**
  $e_i \leftarrow ConcreteDropout(Conv(x), p)$
  $o_i \leftarrow$ Concatenate(e_i, extFeatures)
  $\hat{y}_i \leftarrow ConcreteDropout(FC(o_i), p)$
end
$\hat{y}_{mc} \leftarrow \frac{1}{Reps}\sum_{i=1}^{Reps}\hat{y}_i$
$\eta \leftarrow \frac{1}{Reps}\sum_{i=1}^{Reps}(\hat{y}_i - y)^2$

Algorithm 1: Concrete-Variational Dropout Algorithm

4.6 Results

For the experiments the MNIST classifier described previously was used, but after each layer dropout was applied. The model was pretrained on MNIST initially for 50 epochs and for 30 epochs extra after the addition of dropout. The number of Monte Carlo forward passes used per minibatch was 5. Concrete dropout achieves MNIST accuracy of 99.13%, achieving higher accuracy than the hand-tuned dropout from the keras-team.
Further investigation on Bayesian deep learning has been conducted on the CIFAR10 dataset. Following the same procedure as with the MNIST classifier, a Bayesian deep neural network

is created by replacing normal dropout in the deep CIFAR10 classifier (see [3]) with Concrete dropout.
The model was trained for 200 epochs, using stochastic gradient descend, with learning rate decay from 0.1 down to 0.001, momentum 0.9 and weight decay $5e-4$.The number of Monte Carlo forward passes used per minibatch was 10. Its estimated accuracy was 0.9118 which is approximately the same with its deterministic counterpart.
The model's uncertainty and its dependency on the number of training samples, is demonstrated in Table:[6], where the uncertainty decrease can be observed while the number of samples is increased(Figure:[25). Also, it is worth noting that the accuracy of the model and its standard deviation of its predictions are increased along with the number of training samples, which is probably an indication of the model's confidence on the predictions(Figure:[26],[27]). For each subset of CIFAR10 the model was training from scratch as it is described in the previous paragraph using the same hyper-parameters.

CIFAR10 percentage used for training	Accuracy	Uncertainty	Prediction Standard deviation
5%	0.6107	1.1536	0.0033
20%	0.8633	0.4475	0.0048
50%	0.9042	0.3154	0.0059
100%	0.9118	0.3151	0.0070

Table 6: Demonstration of the Uncertainty dependence on training dataset size

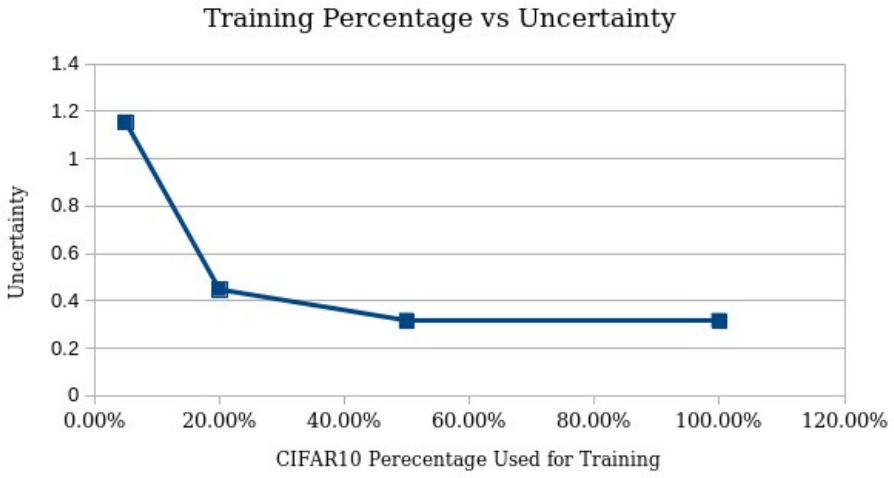

Figure 25: Bayesian Neural Network Uncertainty vs the number of the training samples

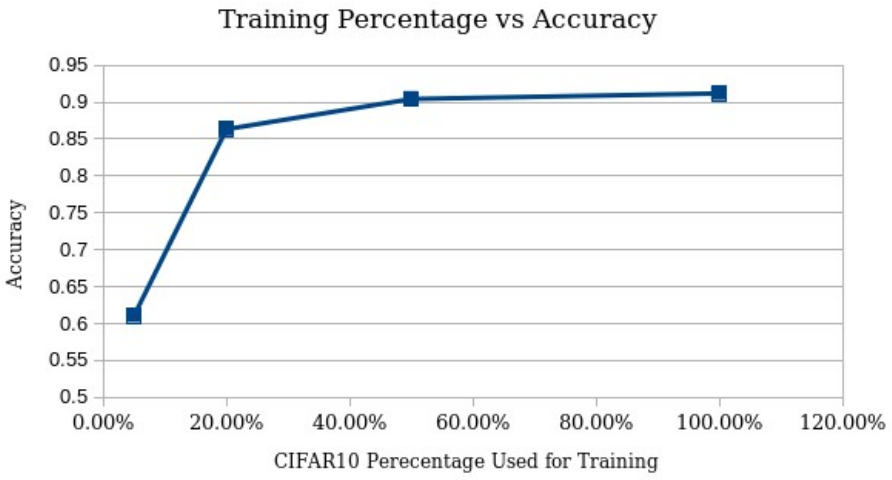

Figure 26: Bayesian deep learning accuracy vs the number of the training samples

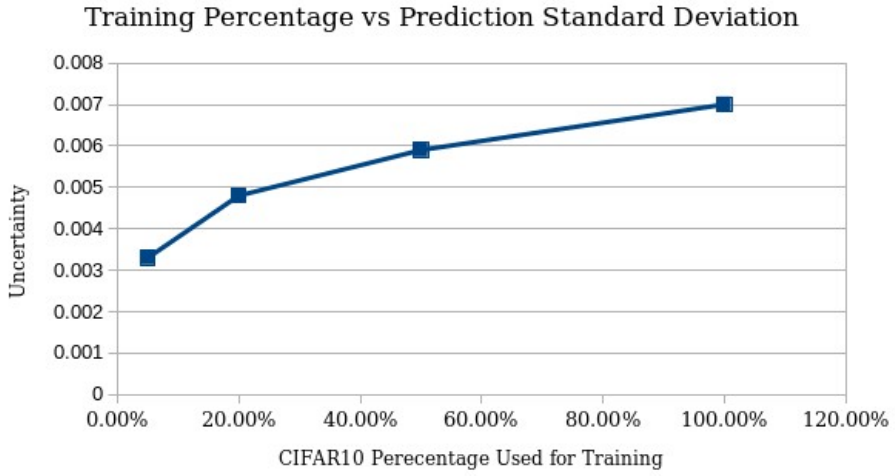

Figure 27: The standard deviation of the model's predictions vs the number of the training samples

4.7 Discussion

In this section, it has been shown that dropout has a regularization effect on the network's prediction because it variates with the posterior distribution of the weight. Also, concrete dropout has been shown that it can compute the optimized dropout probability p and thus it has been used as a substitute to Bernoulli dropout. By having dropout activated during inference, it is possible to estimate the bayesian network's predictions. Several Monte Carlo forward passes are performed and their mean is the network's predictions while the average loss acts as the model's uncertainty estimation. On the downside, one major drawback is the performance. While the results of Variational dropout were accurate, the need for several Monte Carlo forward passes during inference results to a big performance decrease.

5 Combining the methods

5.1 Model Compression and Knowledge Distillation

Combining Model Compression and Knowledge Distillation resulted in several different pipelines. The accuracy increase that occurred in the early stages of pruning was leveraged in order to achieve the highest possible accuracy by combining the two techniques. Knowledge Distillation was experimented with and without a teacher assistant and it was found the use of teacher assistant achieved better accuracy. Thus the subsequent focus was on the cases that included a teacher assistant as part of the pipeline. The initial experiments applied Knowledge Distillation followed by Model Compression on the student network.

1. Train the teacher, the student and teacher assistant on the dataset
2. Use Knowledge Distillation between the teacher and the teacher assistant
3. Use Knowledge Distillation between the teacher assistant and the student
4. Apply pruning to obtain the highest possible accuracy of the student followed by quantization

In the second approach pruning was applied to the teacher followed by Knowledge Distillation into a student model and finally quantization on the student. The student model achieved higher accuracy than just using Knowledge Distillation or Model Compression.

1. Train the teacher, the student and teacher assistant on the dataset
2. Prune all three models until they achieve the highest possible accuracy
3. Use Knowledge Distillation between the pruned teacher and the pruned teacher assistant.
4. Use Knowledge Distillation between the pruned teacher assistant and the pruned student.
5. Apply pruning to achieve the highest possible accuracy of the student followed by quantization

The results in this instance were worse than in the previous approach. That was likely due to pruning, making the model less flexible to adjust to the dark knowledge that is transferred from the teacher to the student. But since the dark knowledge was dependant only on the outputs of the teacher, the final approach chosen was to use Knowledge Distillation and pruning step by step, enabling the transfer of the best possible dark knowledge from the teacher to the student. A flowchart of this pipeline can be seen in Figure:[28].

1. Train the teacher, the student and teacher assistant on the dataset
2. Prune the teacher to figure out the highest possible accuracy
3. Use Knowledge Distillation between the pruned teacher and the teacher assistant.
4. Prune the teacher assistant to figure out the highest possible accuracy
5. Use Knowledge Distillation between the pruned teacher assistant and the student.

6. Apply pruning to figure the highest possible accuracy of the student followed by quantization

The accuracy achieved for the student model, using this pipeline was the highest of all three approaches. That was likely due to the combination of the teacher being more accurate due to pruning and the flexibility of the student to adapt to the teacher's dark knowledge by having all its parameters available to re-calibrate during Knowledge Distillation.

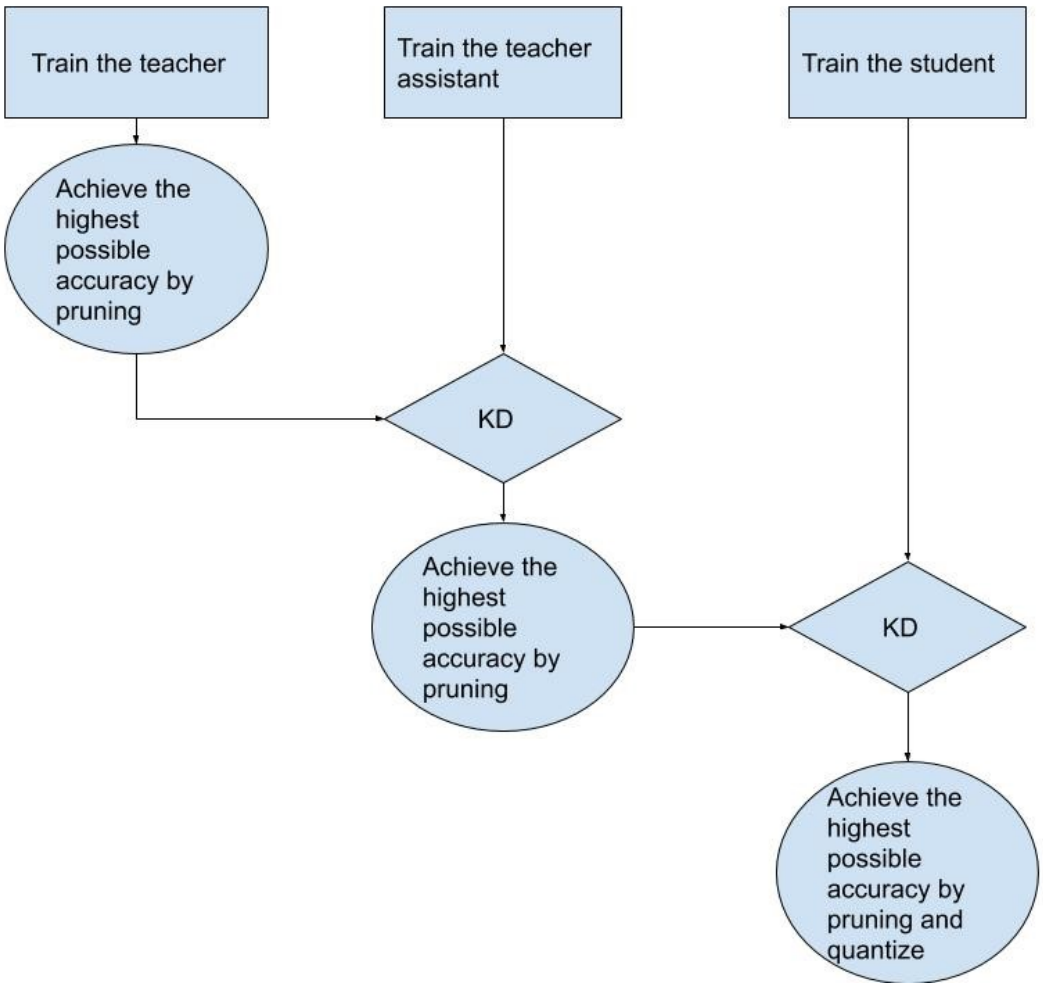

Figure 28: Optimal pipeline combining Model Compression and Knowledge Distillation

5.1.1 Results Model Compression and Knowledge Distillation

One set of experiments using the chosen Model Compression and Knowledge Distillation pipeline as shown in (Figure [28]) were conducted using the neural network architectures: ResNet50, ResNet34 and ResNet18 as teacher, teacher assistant and student networks respectively. Knowledge was distilled from a well-trained ResNet50 down into a much smaller ResNet18 architecture. The ResNets were trained and knowledge distilled using the training procedure described in Section:[3.2.4]. Subsequently, the model was pruned and quantized in order to attain a high accuracy. Shown in Table:[7] are accuracies from the different instances of the model at different stages of the pruning pipeline for all of the models. Furthermore, the teacher model does not have a Knowledge Distillation accuracy as it was the source model for the Knowledge Distillation process. Finally,

Accuracy	**Teacher**	**TA**	**Student**
Initial	0.8348	0.8014	0.7423
KD	—	0.8205	0.7735
Pruning + KD	0.8381	0.8220	0.7775
KD + Pruning + Quantization	—	—	0.7736

Table 7: Results for the compression & Knowledge Distillation pipeline

only the student model was quantized therefore there are no quantization accuracies for both the teacher and TA model.
The Knowledge Distillation process for the dogbreeds dataset before the pruning pipeline caused an approximate 3% increase in accuracy for the resulting pruned and quantized model. For classification tasks it is apparent that through the use of a teachers assistant and the transfer of dark knowledge with additional compression, the model architecture of a network can be greatly sparsified.

5.2 Model Compression and Bayesian Deep Learning

Investigations were conducted into applying Model Compression on a Bayesian Neural network that was implemented with the use of dropout. A CIFAR10 classifier (Figure:[3]) was used with Concrete dropout substituting normal dropout . In the one instance, only the dropout in the convolutional and the fully connected layers were replaced by Concrete dropout while in the another the batch normalization was replaced as well. Even though the results were similar, the use of batch normalization allowed the training of the network to be done with a higher learning rate and thus the convergence to the optimal accuracy occurred faster than without its use.
Since the dropout simulated a Bayesian distribution over the weights, the forward pass used a distribution of random variables and not deterministic parameters. Thus feeding a sample several times to the network resulted in different numerical results. Using these results it was possible to determine the certainty of network predictions through the similarity of the output; if the multiple outputs were vastly different, the network was highly uncertain of its results. Alternatively if the results were very similar, the network had high certainty. Therefore after the network was trained several forward passes were made during inference and the mean value of those predictions were used as the final prediction .
In order to utilize both Model Compression and Bayesian Deep learning the pipeline described in Figure[29] was used. More specifically:

1. Train the Bayesian networks with dropout as usual.
2. Apply the pruning procedure as seen in Figure[13] but the evaluation of the network's accuracy after each pruning step is done by making several forward passes and computing their mean value
3. Use quantization once the pruning loop is over
4. Compare the predictions of the network before and after Model Compression is applied on it

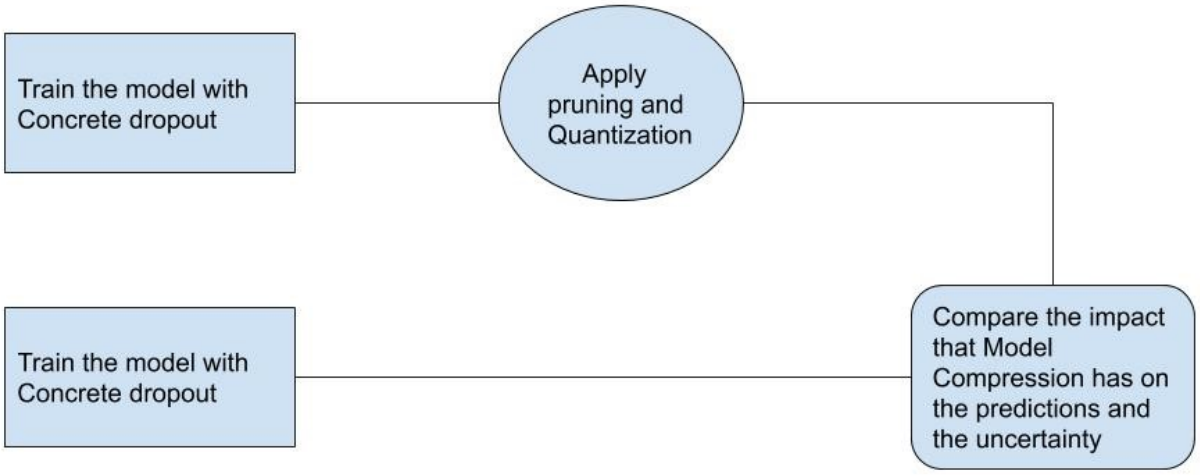

Figure 29: Model Compression and Bayesian Deep Learning workflow

5.2.1 Results Model Compression and Bayesian Deep Learning

Using the CIFAR classifier architecture variant using dropout as described in section 1.4.1, a neural network was trained on the CIFAR10 dataset. The data was split into a training set and a test set, as provided by PyTorch. The classifier was trained from scratch, using an adaptive learning rate which changed every 100 epochs, starting from 0.1 and stopping at 0.001. Stochastic gradient descent was used with a momentum of 0.9 and a weight decay equal to $5e-4$. Dropout was enabled both during training and testing. In order to obtain a mean for the predictions, several inferences run per image were collected and averaged.
Subsequently the Model Compression pipeline was applied to the resultant classifier network. At each step of the Model Compression pipeline, several forward passes were collected in order to reason about the uncertainty of the model. Inference was repeated ten times for calculating a mean prediction per image. It can be observed that while pruning has a positive impact on the model's accuracy, its uncertainty is higher. Quantization had a negative effect in both accuracy and uncertainty as it can be seen in Table:[8].

	Accuracy	**Uncertainty**
Pre Prune	0.9112	0.3497
Post Prune (49.71%)	0.9134	0.3768
Quantization	0.9095	0.3894
Post Prune & Quantization	0.9066	0.3943

Table 8: Model Compression and Uncertainty Quantification for the Bayesian Classifier

5.3 Knowledge Distillation and Bayesian Deep Learning

Knowledge Distillation has proven to be a powerful tool in increasing the accuracy of a model through the influence of a teachers dark knowledge. Thus it is possible that a Bayesian Network which has undergone Knowledge Distillation might have superior predictions. To verify this hypothesis Knowledge Distillation was applied from a deterministic teacher into a Bayesian student model.

1. Train the deterministic models (teacher and assistant) and the Bayesian student
2. Use Knowledge Distillation between the teacher and the assistant
3. Use Knowledge Distillation between the assistant and the Bayesian student
4. Compare the predictions of the network before and after Knowledge Distillation is applied on it

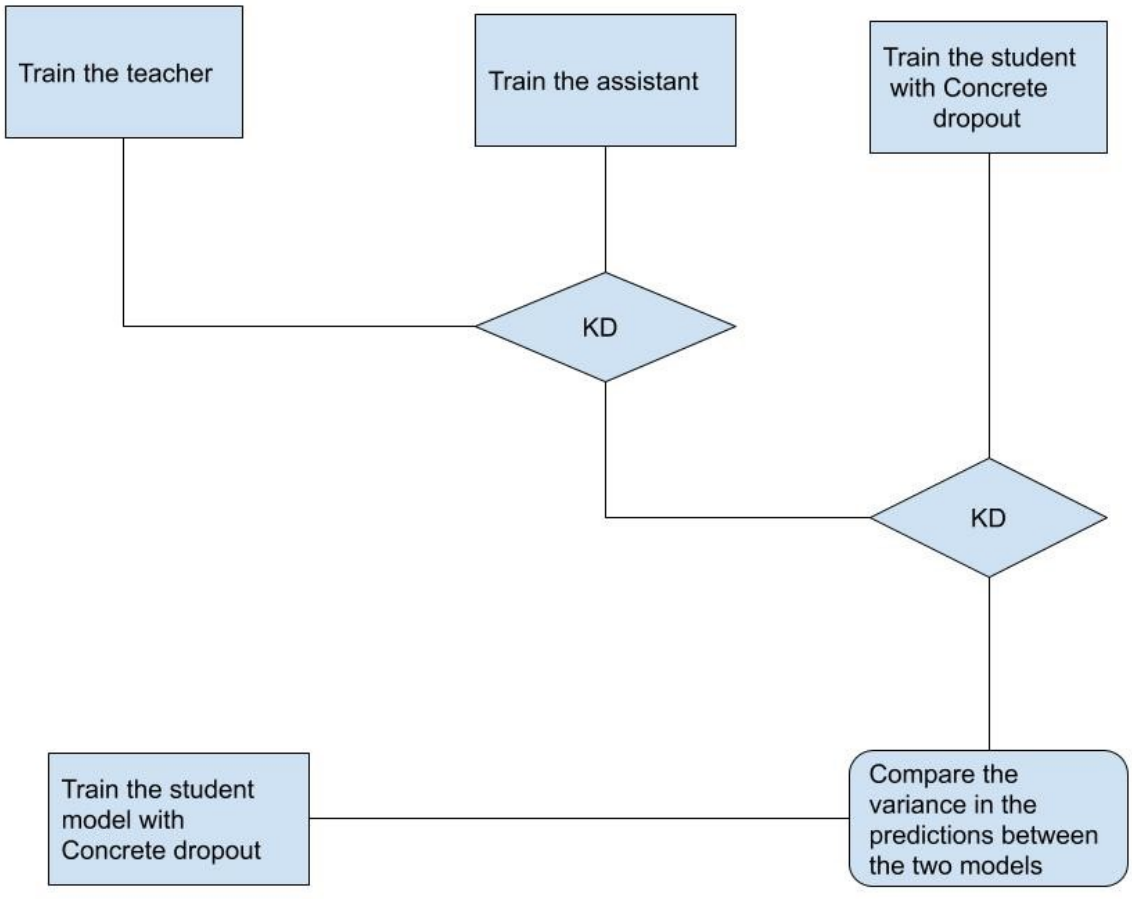

Figure 30: Knowledge Distillation and Bayesian Deep Learning pipeline

	Teacher	**TA**	**Student Accuracy**	**Student Uncertainty**
Initial	0.9415	0.9283	0.9112	0.3497
Knowledge Distilled	—	0.9293	0.9113	0.3103

Table 9: Knowledge Distillation and Uncertainty Quantification for the Bayesian Classifier

5.3.1 Results Knowledge Distillation and Bayesian Deep Learning

The normal variant (i.e. non-dropout) and the dropout variant of the CIFAR10 classifier neural network architecture described in Section 1.4.1 were used to investigate the impacts of Knowledge Distillation on model uncertainty. Using the pipeline described in Figure 30, investigations were conducted using the neural network architectures: ResNet18, ResNet8 and the CIFAR10 drop-out classifier as teacher, teacher assistant and student networks respectively. Knowledge was distilled from a well-trained ResNet18 down into the aforementioned CIFAR10 classifier architectures.
The ResNets were trained and knowledge distilled using the training procedure described in Section 3.2.4. Dropout in the CIFAR10 classifier was enabled both during training and testing. In order to obtain uncertainties for the predictions, several inferences were run per image were collected. The non-dropout variant of the CIFAR10 classifier was trained from scratch and used the same training procedure as described in Section 5.2.1.
Knowledge distilling into the CIFAR10 classifier architecture gave very little boost to the accuracy of the student model. Furthermore, it was shown that the uncertainty of the knowledge distilled model was lower than that of the trained-from-scratch model as it can be seen in Table:[9]. Perhaps applying a similar methodology to a more completed domain such as an object detection would yield greater insight as the performance impacts would be more apparent.

5.4 Model Compression, Knowledge Distillation and Bayesian Deep Learning

Finally, all three techniques that were investigated were combined with the goal of evaluate the impacts of Model Compression and Knowledge Distillation on model uncertainty. The optimal pipeline that was described in Figure[28 was applied on the CIFAR10 classifier using the CIFAR10 dataset, however instead of a deterministic student the dropout variant of the CIFAR10 classifier architecture was used.

1. Train the deterministic models(teacher and assistant) and the Bayesian student
2. Prune the teacher to achieve the highest possible accuracy
3. Use Knowledge Distillation between the pruned teacher and the teacher assistant.
4. Prune the teacher assistant to achieve the highest possible accuracy
5. Use Knowledge Distillation between the assistant and the Bayesian student
6. Apply the pruning procedure as seen in Figure[13] where the evaluation of the network's accuracy after each pruning step is done by making several forward passes and computing their mean value
7. Compare the predictions of the network before and after Model Compression and Knowledge Distillation are applied

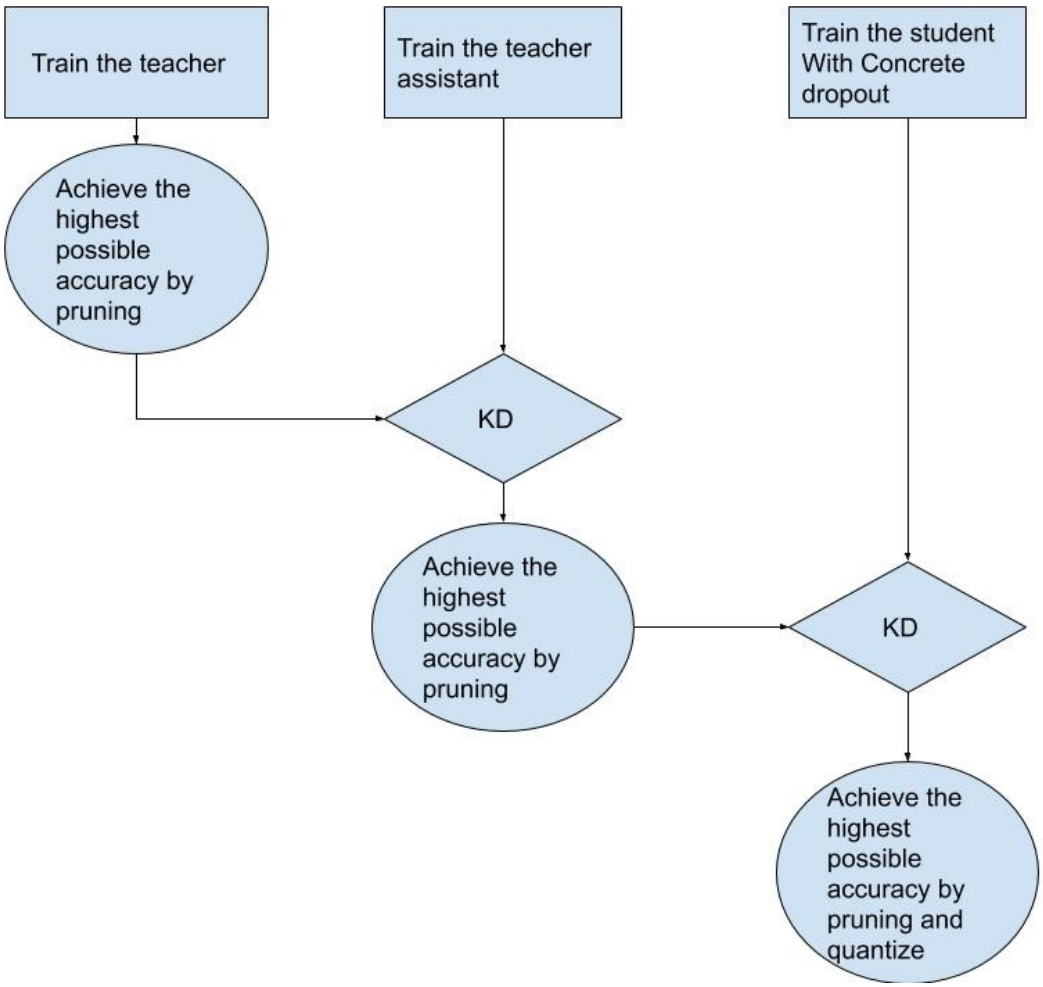

Figure 31: Combining Model Compression, Knowledge Distillation and Bayesian Deep Learning

5.4.1 Results Model Compression, Knowledge Distillation and Bayesian Deep Learning

Knowledge Distillation and the pruning pipeline were applied to the aforementioned CIFAR10 dropout classifier architecture to enable Uncertainty Quantification of a knowledge distilled, pruned and quantized model. Using the pipeline described in Figure 31, investigations were conducted using the neural network architectures: ResNet18, ResNet8 and the CIFAR10 drop-out classifier as teacher, teacher assistant and student networks respectively. Knowledge was distilled from a well-trained ResNet18 down into the aforementioned CIFAR10 classifier architectures.
The ResNets were trained and knowledge distilled using the training procedure described in Section 3.2.4. Dropout in the CIFAR10 classifier was enabled both during training and testing. In order to obtain uncertainties for the predictions, several inferences were run per image were collected. The non-dropout variant of the CIFAR10 classifier was trained from scratch and used the same training procedure as described in Section 5.2.1.
Shown in Table 10 are the accuracies the teacher, teacher assistant and student models. Additionally the uncertainty of the student model is also shown. The teacher does not have an accuracy for the Knowledge Distillation step as the teacher is the source for the distillation. Impressively the

	Teacher Accuracy	**TA Accuracy**	**Student Accuracy**	**Student Uncertainty**
Initial	0.9415	0.9283	0.9112	0.3497
KD	—	0.9293	0.9123	0.3103
Pruning + KD	0.9434	0.9289	0.9128	0.3741
KD + Pruning + Quantization	—	—	0.9045	0.3956

Table 10: Model Compression and Uncertainty Quantification and Bayesian Deep Learning for the Bayesian Classifier

network was capable of being knowledge distilled, pruned and quantized while maintaining only an approximate drop in accuracy of approximately 0.0067. Knowledge Distillation and pruning actually improved the accuracy by 0.0016 from the original accuracy awhile the quantization resulted in the majority of the loss in accuracy. Again, as in the previous sections, it is observed that Knowledge Distillation reduces the model's uncertainty while pruning and quantization increase it.

These results were consistent with those exhibited in the previous runs, without indicating that reducing model size can have a negative impact on model certainty.

6 Discussion

It has been shown in Section:[2] how class blind pruning and quantization can reduce the size of the model. The main idea was to find the parameters over the whole model that have minimal impact and set them to zero, thus reducing the computational cost. Class blind pruning that is applied on the whole model was chosen instead of layer-wise pruning which was used by Han et al. [20] due to the fact that network tends to over parameterize in the deep layers. Thus by using class-blind pruning, it is possible to maintain the early parameters which are important for the construction of the high level features while removing the ones who are not contributing much into the final predictions. The percentage of the weights that were pruned were depending largely on the model, the dataset and the task that network was performing. Object Detectors despite the fact that they have vastly more parameters than classifiers, arrived at a lower pruning percentage. Even for Faster-RCNN which was using ResNet101 as a backbone that contains 44.5 million parameters and was trained on a realatively easy dataset as VOC2007, the percentage of pruned weights was lower than all the classifiers. It is worth noting that when a model is vastly overparametarized on a dataset, class-blind pruning produces a regularization effect. A similar effect has been shown that occurs with targeted dropout[36] where dropout is applied on the lower absolute lower values of the layers. A hypothesis why this occurs is that a smaller model generalizes better than an overparametarized deep model. That effect though is less visible when the accuracy gets closer to the Bayes error in classifiers. Increase in the prediction accuracy has been observed in Faster-RCNN. However that is not the case with YOLOv3 where an initial drop occurs. It was assumed that the main reason why this accuracy increase does not occur in YOLOv3 is the fact that it was trained on MS-COCO which is a rather difficult dataset with more classes than VOC2007 and more objects close to each other.
For the Quantization choosing K-means as a method of clustering was implemented to run on the GPU since, applying K-means over millions of parameters was not efficient. The identity 1 was utilized in order to create a dynamic clustering that was dependant on the number of parameters for each layer. While the weights were clustered significantly there was not a significant drop in the accuracy. However at least a small drop in the accuracy is to be expected by using quantization which is more to the fact that it requires clustering, than lowering the precision of the weights.
Knowledge Distillation is a handy tool to increase the accuracy of a shallow model given that there is a deep network that outperforms its shallow counterpart. Using KL-Divergence between the student and the teacher on a modified loss function in order to transfer the dark knowledge between the two models helped the student to increase its accuracy by $\sim 2\%$. Additionally using a teacher assistant when the difference between the student and the teacher is big proved useful. That impact of Knowledge Distillation however was minimized when the student and the teacher outputs were close to each other. Still, even then, Knowledge Distillation was improving the accuracy of the student. A major drawback though, is that it can only be applied on classification tasks since the main idea is to soften the logits of a softmax function. Furthermore the hyper parameters need to be hand-tuned which can be time consuming.
In the Bayesian Deep Learning section a brief introduction into Variational Inference was made, which is a technique to approximate the posterior distribution which is intractable with a simple known distribution. It was then shown that dropout creates a regularization effect since it variates with the posterior distribution of the weights. Finally the final extension of variational dropout was introduced; concrete dropout, which makes possible to optimize the dropout probability, thus eliminating the tedious task of hand-tuning it. Bayesian models are competitive in their accuracy

to that of their deterministic counterparts and proving an uncertainty estimation as well. However since the weights are random variables, and the additional need to make Monte Carlo forward passes to get the predictions, there was an increase inference time. Another issue is that the variational use of dropout causes incompatibility with batch normalization since the impact of batch normalization is reduced if it follows dropout.

For the final section [5], all three methods were combined using different pipelines. Three different pipelines were experimented with Model Compression and Knowledge Distillation. It seems that applying Knowledge Distillation on a student model before Model Compression worked the best. The reason for this could be that fine-tuning from the teacher's dark knowledge works better when the model has more trainable parameters. Thus, it was chosen to improve the teacher's accuracy as much as possible followed by distilling its knowledge to the student. This approach improved the performance of the student model in every dataset, in comparison using only one of those two methods.

In order to understand the impact of Model Compression on the models ability to generalize, both pruning and quantization were applied on a model that is stochastic. Despite the fact that the random variables that were used as an input in the Concrete distribution were discrete after quantization, after variating with the posterior distribution of the weights the accuracy loss was low.

Knowledge Distillation helps the student model to produce better results by trying to minimize the student's predictions distance from the predictive distribution of the teacher as well the ground truth. That also has a slight impact to the uncertainty of the student but not as much as pruning. Softening the teacher's logits even further by increasing the temperature T or increasing the Dark Knowledge that is transferred to the student while maintained similar levels of accuracy it resulted into an even further increase of the uncertainty.

For the final pipeline, where Model Compression, Knowledge Distillation and Bayesian Deep learning were combined, the Model Compression and Knowledge Distillation pipeline was used, however instead of having a deterministic student, a stochastic student was used in its place. Applying all these techniques together on CIFAR10 a stochastic student was created which was capable of achieving competitive results in comparison to one were Model Compression and Knowledge Distillation were not applied.

Overall, the techniques of pruning, quantization and Knowledge Distillation were implemented in order to reduce the computational cost of deep learning models while evaluating the impacts in the models uncertainty using Bayesian Deep Learning. It appears that Model Compression is more flexible as it can be applied to all models and tasks, while Knowledge Distillation requires a teacher that outperforms a student and can be used only for classification problems due to its dependency in the softmax activation.

7 Future Work

One possible extension of the investigation conducted would be to apply the full Knowledge Distillation, model compression and Bayesian inference pipeline in Object Detection and Semantic Segmentation networks. While Model Compression has been applied to detectors as YOLOv3 and Faster-RCNN in the investigation conducted, there was a lack of application of Knowledge Distillation and Bayesian Uncertainty Quantification in detection.

7.1 Two Stage Detector Knowledge Distillation

Knowledge Distillation as a technique applied globally on the predictions of an object detector can boost the background errors and produce worse results than if the model were trained from scratch and fine tuned it without the use of Knowledge Distillation. In other words, it is not be possible to improve the results as Knowledge Distillation will have negative effect on the model as suggested by Li et al. [37].
However, Li proposes an alternative approach for two stage detectors. The use of a different loss function can be used to train the student model, which enables the Knowledge Distillation to achieve increased accuracy for object detection:

$$\mathcal{L}(\mathbf{W}) = \lambda_1 \mathcal{L}_m(\mathbf{W}) + \lambda_2 \mathcal{L}_{gt}(\mathbf{W}) \tag{8}$$

$$\mathcal{L}_m(\mathbf{W}) = \frac{1}{2N} \sum_k \frac{1}{m_i} \|u^{(i)} - r(v^{(i)})\|_2^2 \tag{9}$$

$$\mathcal{L}_{gt}(\mathbf{W}) = \mathcal{L}_{cls}(\mathbf{W}) + \lambda_2 \mathcal{L}_{reg}(\mathbf{W}) \tag{10}$$

in which $\mathcal{L}_m$ is the L_2 loss of feature mimic were m_i is the dimension of the features extracted by the i^{th}region proposal, L_{gt} is the classification and regression loss function of region proposal network from Fast-RCNN, λ_1 and λ_2 are the loss weight balance parameters. N is the total number of the proposals sampled, $u^{(i)}$ is the feature sampled by spatial pyramid pooling from the feature map of the large model based on the i^{th}proposal. $v^{(i)}$ is the sampled output feature of the small network and r is the regression function to transform $v^{(i)}$ to the same dimension as $u^{(i)}$. By optimizing this loss function, the small network can be trained under both the ground-truths and the additional supervision from the large models.

7.1.1 Implementation Details and Results

The framework of mimic training is illustrated in Figure [32]. A large Faster-RCNN or R-FCN network is trained in typical fashion to optimize softmax loss and smooth L_1 loss by the supervision of the ground-truths on the training data. The training is done in two stages.
During the first stage, the feature mimic architecture contains two networks. The large network is initialized by the weights of the well trained detection network and the layers are fixed during the training process. The small network is randomly initialized. A Region Proposal Network is added to the end of the small network to generate object proposals. Initially, an entire image is forwarded through both of the networks to produce two different feature maps. Given the proposal regions generated by the RPN on small network, the sampler will extract the features at the proposal regions on the feature map of both the large network and the small network.

In the second stage a Faster R-CNN or a R-FCN detection network is fine-tuned using the region proposal network on the training dataset. Further more, the newly added layers are randomly initialized. Fine-tuning using the ground-truth supervision only might degrade the features learned by mimicking the first stage. The prediction of the detector in Faster-RCNN or R-FCN detector can be regarded as a classification task. Therefore the logits matching supervision can be added to the second stage of detection pipeline to further transfer the knowledge of the large detection models to the small models. Moreover, using the two-stage mimic, not only the proposal related information is transferred but also the category classification information learned by the large model can be passed to the small network. Richer information from the large model can further help the small network mimic the large model. The inference process will remain the same as the original Faster-RCNN or R-FCN without any increased parameters.
On top of that Bayesian Deep Learning can produce a distribution of RoI for the detector. While the inference will be slow, a distribution over the Regions of Interest might result in better predictions of the network by using the mean.

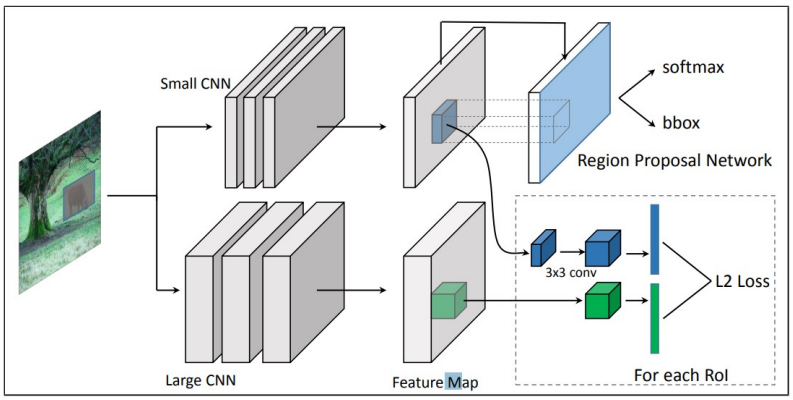

Figure 32: Overall architecture of feature mimic by proposal sampling. A Region Proposal Network generates candidate RoIs, which then are used to extract features from the feature maps.[[37]]

As shown in the conducted investigations, pruning shallow networks that consist of a large number of parameters can be done without minimal loss in the accuracy, and in some instances by improving it. This can be useful with FasterRCNN when using Wide Residual Networks[38] as the base of the model, which are shallow with a large number of parameters and take full advantage of the CUDA parallel computations in order to reduce the inference time. Further pruning and quantizing such as Wide Residual Networks may further improve inference time and the memory footprint of architectures such as FasterRCNN.

8 Conclusion

Tying the results to the initial goals presented in Section 1.5, it can be seen that the findings presented did indeed contribute towards the reduction of inference time (goal one) and reduction of memory footprint (goal two).
Goal one was supported by the findings presented in Section 2.5, where model compression with the use of class-blind pruning and dynamic K-Means clustering was successfully applied to the object detection neural network architectures YOLOv3 and FasterRCNN and two other small classifier neural network architectures. The model compression pipeline presented by Han et. al. in [20] was extended as described in Section 1.6 and in Section 2.4, resulting in a reduction in the accuracy drop caused by the pruning and quantization steps of the model compression pipeline. While, due to framework limitations the investigation presented in this thesis was unable to obtain concrete inference time measurements, the sparsification of weights matrices can speed up the inference time as it was presented by Han.
Goal two was supported by the findings presented in Section 2.5, showing the reduction in the memory footprint of the neural network architectures after the model compression pipeline was applied. By leveraging the dynamic K-Means clustering approach as described in Section 2.4, the model compression pipeline was modified to reduce the accuracy loss caused by quantizing layers with a large numbers of parameters into too few clusters as described in Section 1.6.
Finally, the use of uncertainty quantification enabled insights into the impacts of pruning, quantization and Knowledge Distillation on model uncertainty, as presented in Section 5.

References

[1] F. Falcini, G. Lami, and A. M. Costanza. Deep learning in automotive software. *IEEE Software*, 34(3):56–63, May-Jun. 2017.

[2] Adnan Parlak, Yasar Islamoglu, Halit Yasar, and Aysun Egrisogut. Application of artificial neural network to predict specific fuel consumption and exhaust temperature for a diesel engine. *Applied Thermal Engineering*, 26(8-9):824–828, 2006.

[3] Min-Joo Kang and Je-Won Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*, 11(6):e0155781, 2016.

[4] Raia Hadsell, Ayse Erkan, Pierre Sermanet, Marco Scoffier, Urs Muller, and Yann LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 628–633. IEEE, 2008.

[5] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[8] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[9] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. *http://yann.lecun.com/exdb/mnist/*, 2010.

[10] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research). *http://www.cs.toronto.edu/ kriz/cifar.html*, 2010.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

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

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[14] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *arXiv preprint arXiv:1801.05134*, 2018.

[15] Jonathan Hui. Real-time object detection with yolo, yolov2 and now yolov3. *https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088*, 2018.

[16] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[17] CyberAILab. A closer look at yolov3. *https://www.cyberailab.com/home/a-closer-look-at-yolov3*, 2018.

[18] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[19] Wang Yong. Faster-rcnn. *https://www.cnblogs.com/wangyong/p/8513563.html*, 2018.

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

[21] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440, 2016.

[22] Abigail See, Minh-Thang Luong, and Christopher D. Manning. Compression of neural machine translation models via pruning. *CoRR*, abs/1606.09274, 2016.

[23] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *CoRR*, abs/1810.05270, 2018.

[24] Yunhui Guo. A survey on methods and theories of quantized neural networks. *CoRR*, abs/1808.04752, 2018.

[25] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pages 963–971, 2014.

[26] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). *http://www.cs.toronto.edu/ kriz/cifar.html*, 2010.

[27] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[28] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing deep neural networks with logic rules. *CoRR*, abs/1603.06318, 2016.

[29] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *arXiv preprint arXiv:1902.03393*, 2019.

[30] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[31] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[32] Felix Laumann. Medium blog. *https://medium.com/@laumannfelix*, 2018.

[33] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

[34] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2575–2583. Curran Associates, Inc., 2015.

[35] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017.

[36] Aidan N Gomez, Ivan Zhang, Kevin Swersky, Yarin Gal, and Geoffrey E Hinton. Targeted dropout. 2018.

[37] Quanquan Li, Shengying Jin, and Junjie Yan. Mimicking very efficient network for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6356–6364, 2017.

[38] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Glossary

Bayesian Network A network architecture that consists of parameters that are represented by random variables which follow distributions.

Bounding Box A region in a given image representing the location of an object.

Classification A machine learning task that involves predicting classes for given data.

Convolutional Layer A layer in a neural network used to extract abstract features from a specific area of the output of the previous layer. Used primarily with image data.

Deterministic Network A network architecture that consists of parameters that represent a specific numeric value.

Fully Connected Layer A layer in a neural network that connects every neuron from the previous layer to every neuron of the subsequent layer.

Logit The logarithm of an odd or probability.

Monte Carlo A broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.

Object Detection A machine learning task that involves recognizing objects in a given scene through their detection and subsequent classification.

Regularization Adding information or noise to a network training procedure in order to reduce overfitting.

Semantic Segmentation Pixel-wise classification on a given image where each pixel belongs to a class.