Efficient Bayesian Neural Networks for Outdoor Semantic Scene Understanding Tasks in Robotics

Candidate Number: BVKP2

Internal Supervisor: Dimitrios Kanoulas

Adjunct Supervisor: Dennis Hadjivelichkov

September 2023
MSc Machine Learning
University College London

Acknowledgements

I would like to thank my supervisors Dr. Dimitrios Kanoulas and Dennis Hadjivelichkov for their help and support throughout the project. I would also like to thank Dr. Yuanchang Liu and Dr. Brooks Paige for their time on discussing ideas for the project.

Abstract

Vision-based scene understanding tasks in the robotics domain have the unique setting of needing to process data in real-time whilst achieving high reliability and safety. In this thesis, we tackle these two important challenges of improving the *efficiency* and *safety* of deep neural networks. We propose an efficient encoder-decoder architecture that utilizes light-weight building blocks, skip connections, and efficient scaling. We benchmark our results on the CamVid dataset and achieve competitive results compared to state-of-the-art, with a *lite* network scoring 66.4 mIOU on the test set with only 0.56M parameters, and a *large* scoring 73.9 mIOU with 10.52M parameters. We then configure the network into a Bayesian neural network using MC Dropout and show that the uncertainties obtained using Bayesian inference achieves almost perfect uncertainty calibration with up to 0.17 ECE. We further demonstrate a novel method of performing Bayesian inference using stochastic depth. When compared to dropout methods, stochastic depth obtains higher mIOU scores as well as lower ECE of 2.14 using deterministic test-time inference.

Table of contents

Introduction

1.1 Deep Learning for Robotics

Over the last decade, there has been a surge of interest in using neural networks in almost all aspects of robotics systems, such as robot vision, robot manipulation, and autonomous navigation. However, the high expressiveness of neural networks typically comes at the cost of increased computational requirements. The lack of efficiency of neural networks often becomes the bottleneck in the implementation process. In addition, modern neural networks present critical issues, whereby they are often "confidently wrong". The overconfidence phenomenon makes it difficult to implement neural networks in safety-critical applications, where the interpretability of the network's predictions is vital for users. In short, deep learning tasks in robotics face two open-ended critical issues: efficiency and safety.

This thesis is an attempt to address these two issues in a two-fold process. First, we extend recent advances in efficient convolutional neural networks to the encoder-decoder architecture and present an efficient and scalable architecture. Secondly, we leverage techniques in Bayesian Deep Learning (BDL) to turn our efficient network architecture into a Bayesian Neural Network (BNN). These networks are able to produce uncertainties along with the prediction, as shown in Figure 1.1, and we will show that such networks are able to achieve improved uncertainty calibration.

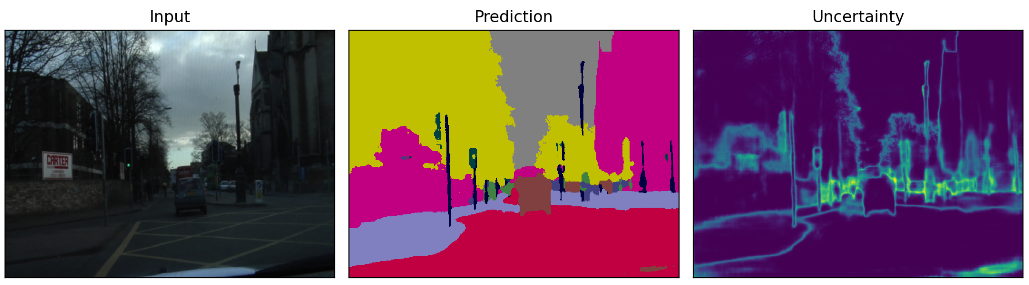


Figure 1.1 A Bayesian neural network outputs calibrated, per-pixel uncertainties along with predictions

1.1.1 Efficient Networks

Robots interact with the world in real-time, and are often required to react to or respond to rapid changes to their environment. Consider an autonomous mobile robot that relies on a deep convolutional neural network for its navigation system. A slow inference response from the neural network will mean that fast-moving obstacles will be detected with a delay, or not detected at all. Such failure mode can potentially harm the users or the system itself and incur high costs.

In addition, robotic mobile systems cannot carry expensive and powerful computational resources with them due to weight, power, and cost constraints. This setting is unlike other safety-critical tasks such as medical imaging, where the computational resources are usually stationary or remote. The constraint on resources further enforces the requirement for these systems to be *efficient*, whereby they should use a minimal amount of computation to achieve the best performance. Such limitation in resources is particularly prevalent in computer vision tasks such as scene understanding, where objects in the scene need to be segmented on a per-pixel basis so that the control system can make appropriate responses.

Modern neural networks, on the other hand, have leveraged advances in GPU hardware to achieve better performance by becoming larger and deeper. ResNet, the state-of-the-art image recognition CNN, is known for gaining increased expressiveness via depth [27]. Many image segmentation networks, such as DeepLab, use ResNet101 as the backbone encoder, which itself already uses 60M parameters [12]. The large network size and high FLOP count mean that these segmentation methods often cannot be run in real-time on most GPUs on mobile robots, let alone CPUs. Furthermore, recent advances in vision transformers require even more parameters with upwards of 100M or more [14].

This trend in using deeper and larger networks to obtain higher performance means that there is a lack of easy and accessible tools for robotics practitioners to build segmentation networks to their needs. Different robotic devices have different computational constraints depending on the hardware, and it makes sense to utilize the largest feasible network with the highest performance given the computational constraints. Thus, this thesis fills the research gap by presenting a simple and elegant solution to network scaling for encoder-decoder architectures. We will further show that the proposed network is able to retain a high performance-to-parameter ratio at varying sizes, whilst running comfortably in real-time.

1.1.2 Sources of Uncertainty

In the following, we discuss three sources of uncertainties that are present in most prediction problems: a) uncertainty due to the precision of the measuring system, b) errors in the learned model, and c) high variability of real-world scenarios.

Most real-world inputs have an irreducible degree of uncertainty due to the limited precision of the measuring device. For example, a radar sensor will have uncertainties and variability in the distance measured. A camera is limited by its sensor resolution and quality, and may not always be able to capture the full range of colours in all lighting conditions. It is also limited by its shutter speed, which can blur objects when either the camera or other objects are non-stationary. These uncertainties are often also defined as a type of *aleatoric uncertainty*, that is uncertainty regarding the data itself. This type of uncertainty is *irreducible*, regardless of the learning system and how much training data is present.

Uncertainties in the learned model can manifest in two ways: errors in the model structure itself, and uncertainties during the learning process. The model structure represents a prior belief on what the designer of the algorithm believes is a good learning model for the problem at hand. For instance, one may choose a fixed set of polynomial features for a linear regression problem, which constrains the expressiveness of the algorithm but also avoids overfitting the data too much. In the context of neural networks, one may choose to process image data with convolutional layers rather than fully connected layers, as the former retains the spatial structure of data better than the latter. However, there is always a degree of uncertainty with the structure of the model itself, and it's very unlikely for one to choose the "correct" model architecture from the start. Uncertainties can also arise during the learning process. Neural networks are trained with stochastic gradients with random initialization, which means that they usually converge to a slightly different local minimum in every experiment. Both of these uncertainties are defined as *epistemic uncertainties*, that is uncertainty regarding the model itself.

Finally, real-world data is highly dynamic and its behaviour can be hard to predict with limited data. For instance, the same road scene will look very different on a sunny day compared to a rainy and foggy day. The same tree will appear differently over different seasons, and road signs may be removed or added throughout the year. These changes represent a *domain shift* or distribution shift when the real-world data is different to the training set data. Neural networks are typically sensitive to these changes and their output qualities can drastically vary as a result. This type of uncertainty is also epistemic uncertainty, because it is uncertainty regarding the knowledge of the network, and can be reduced with more training data. See Figure 1.2 for example outputs of aleatoric and epistemic uncertainties.

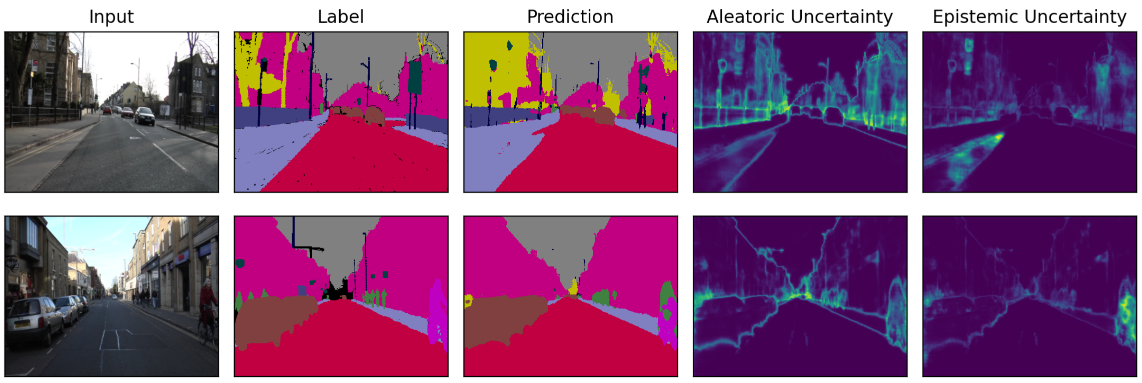


Figure 1.2 Aleatoric and epistemic uncertainties predicted by a Bayesian neural network. Note that aleatoric uncertainty tends to be higher along edges and boundaries of objects, where uncertainty arises from finite pixel resolution. On the other hand, epistemic uncertainty tends to be higher within objects, and is concerned with the classification of the object itself.

1.1.3 Uncertainty Calibration

A *reliable* algorithm should be able to take into account the aforementioned sources or uncertainty, and “knows what it knows”. In fact, the argument is often made that all intelligent agents should be Bayesian, one which is “aware” of the presence of uncertainties [4]. However, modern neural networks exhibit several critical issues that make it difficult to implement them in safety-critical applications. Firstly, they provide unreliable uncertainty estimates with frequent overconfident predictions, which means that they are unable to convey a calibrated confidence that represents the true probability of their predictions being correct [22]. Secondly, they are vulnerable to adversarial attacks, where small adversarial changes can be made to the input to result in drastically different outputs [43]. Lastly, they are unable to distinguish between in-domain and out-of-domain samples, which means that they are not aware when the input space has shifted to a domain that they have not been trained on [57].

These critical issues are present because there are always inherent uncertainties involved in the prediction process of learning algorithms, and networks that cannot predict these uncertainties correctly will inevitably face these drawbacks. On the other hand, if these uncertainties can be predicted beforehand, autonomous systems can then be programmed to perform safer actions by moving slower, or pausing and waiting for human intervention [16]. Note that, unlike supervised learning, there is no "ground truth uncertainty", instead we seek networks to have the desired property of producing *calibrated uncertainties*, where the confidence score is equal to the probability that the prediction is correct. I.e. 80% of the time when the network is 80% confident, it should produce the correct results. Mukhoti and Gal

[52] further defines two desired characteristics that follow this idea: 1) networks should be correct when it's confident, and 2) not confident when it's incorrect. We later define metrics that measure the qualities of uncertainties based on these principles in Section 3.3.

Bayesian Deep Learning (BDL) is an emerging field of research that aims to incorporate Bayesian reasoning into neural networks [19]. BDL has become a vital component in fields such as active learning [18], reinforcement learning [45], and semi-supervised learning [79]. BDL methods generally try to sample a set of neural network weights that approximate the posterior distribution, thereby making predictions that are aware of both epistemic and aleatoric uncertainty. A comprehensive review of BDL methods is covered in Section 2.2.2. This thesis utilizes relatively light-weight BDL methods [16] that have shown promising results with better calibrated uncertainties [52] and sometimes improved predictive properties [38].

1.2 Aims and Objectives

The aims and objectives of this thesis are as follows:

1. Modify the encoder-decoder architecture with efficient building blocks on both the encoder and the decoder to improve the network efficiency

2. By experimenting with network architectural designs, propose efficient methods of scaling the network to obtain a performance-to-parameter ratio that is on par with state-of-the-art methods

3. Modify the encoder-decoder networks into Bayesian neural networks using MC Dropout and show that these networks obtain desired qualities such as improved uncertainty calibration and higher predictive performance

4. Propose a novel Bayesian approximation method using stochastic depth and show with empirical evidence that such networks possess similar or improved properties as dropout-based methods

The rest of the thesis is organised as follows. In Chapter 2 we present the literature review on the two main components of this thesis: efficient convolutional neural network design and Bayesian deep learning. We place particular emphasis on methods that are closely related to techniques used in this thesis. In Chapter 3 we discuss the methodology of the thesis. In particular, we present results from preliminary experiments for designing the decoder, and we cover the relevant mathematical details for the Bayesian techniques that we use. In

Chapter 4 we show the qualitative and quantitative results from two sets of experiments. The first set of experiments attempts to remove blocks from networks in the most efficient way to preserve as much performance as possible. The second set of experiments explores Bayesian approximation using MC Dropout and stochastic depth and evaluates the corresponding uncertainty quality. Finally, in Chapter 5 we conclude the thesis and point to directions for future work.

Literature Review

This section covers the literature on the two topics that are central to this thesis: Deep Learning for Computer Vision, and Bayesian Deep Learning. For both topics, we cover a holistic view of recent literature, with a higher emphasis on works that are most relevant to this thesis.

In Section 2.1.1 we discuss the basic building blocks of modern convolutional nets and particularly focus on skip connections and attention mechanisms. We then examine recent techniques to reduce the computational costs of CNNs in Section 2.1.2, with emphasis on depthwise convolutional blocks and inverted residual blocks. We conclude the section with notable recent advances in image segmentation, such as the encoder-decoder architecture, atrous convolution based methods, and multi-branch methods in Section 2.1.3.

In the second part of this literature review, we first introduce the Bayesian learning framework in Section 2.2.1. This is followed by a review of common Bayesian deep learning methods in Section 2.2.2, including variational inference, Laplace approximation, and sampling methods. Finally, we focus in particular on Bayesian approximations techniques central to this thesis in Section 2.2.3, namely dropout-based methods.

2.1 Deep Learning for Computer Vision

This section provides a literature overview of deep computer vision, beginning with some common building blocks and techniques. In the context of scene-understanding tasks in robotics, it's important for networks to be efficient in operating in real-time. Thus, we place particular emphasis on efficient convolutional neural networks. We conclude with a summary of semantic segmentation frameworks, which can often be incorporated with efficient CNNs as encoders to produce efficient segmentation networks.

2.1.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) as deep neural networks built with convolution blocks. A basic convolution block uses a $k \times k \times c_{i-1} \times c_i$ kernel that performs the convolution operation over the input. This operation is applied over c_{i-1} input channels and outputs c_i channels. The stride of the convolution s impacts the resolution of the output feature map. For instance, $s = 1$ would keep the resolution the same, and $s = 2$ would downscale the resolution by half. The padding p of the convolution usually serves as an implementation detail to keep output dimensions consistent with input dimensions. Most CNNs adopt the strategy of decreasing the spatial dimension using convolutions or pooling modules with stride $s > 1$, whilst widening the channel depth. The intuition is that smaller feature dimensions increase the perceptive field of the networks and help with the extraction of long-distance features.

Early Architectures

The earliest CNN dates back to the 1990s when they were used to perform tasks such classifying handwritten digits. One of the earliest prominent CNN architecture is LeNet [42], built with a series of convolutional blocks and pooling layers to downsample the image, and fully connected layers as the predictor. Subsequently, AlexNet [40] was proposed as a prominent early-day CNN architecture that achieved significantly improved classification results with a deeper network architecture. GoogLeNet was proposed by Szegedy et al. [66] with a novel inception block that combines multi-scale convolutional transformations with filters of different scales.

Residual Connections

The Residual Network (ResNet) architecture [27] was designed to address the challenges of training and optimizing very deep neural networks, which often face issues such as vanishing or exploding gradients [3]. The central idea of the network is to learn residual functions using skip connections, or shortcut connections between layers, that enable the direct propagation of information through many layers. Formally, a skip connection block is described as

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x} \tag{2.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ represent the input and output vectors of the layer, such that the network only needs to learn $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$ for a given underlying target mapping $\mathcal{H}(\mathbf{x})$. ResNet alleviated the difficulties of training deep networks by allowing gradients to flow unimpeded, leading to both faster convergence and better performance.

Dense Connections

Dense Convolutional Networks (DenseNet) [33] further increase connectivity between layers by connecting every layer with every preceding layer. The features are densely connected via concatenation, further alleviating the vanishing gradient problem and encouraging feature reuse. Concretely, every ℓth layer receives feature outputs from all preceding $\ell - 1$ layers:

$$\mathbf{y}_\ell = \mathcal{F}([\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\ell-1}], \{W_\ell\}) \tag{2.2}$$

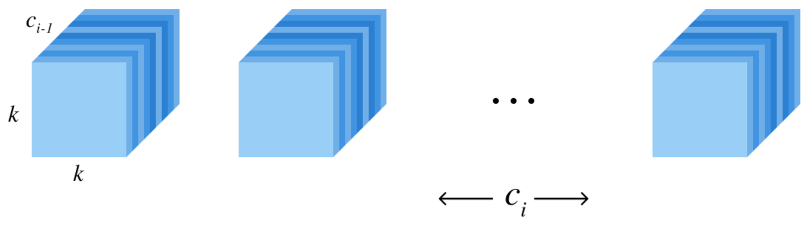
Where $[\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\ell-1}]$ refers to concatenation of the feature vectors. In addition, the authors highlight a connection between DenseNet and stochastic depth regularization [34]. Stochastic depth also allows for every layer to be connected to every proceeding layer but in a probabilistic fashion.

Squeeze and Excitation

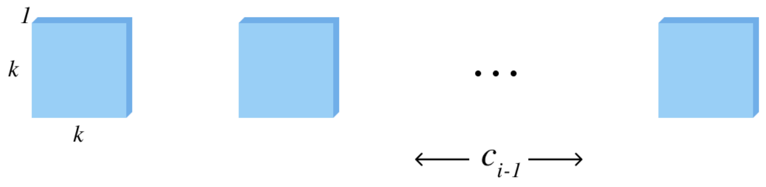
The Squeeze-and-Excitation (SE) block is a way to adaptively recalibrate channel-wise feature responses [32]. The structure of the SE block involves first a squeeze operation that aggregates feature maps across spatial dimensions. This is often performed using max-pooling or average-pooling layers. Secondly, the excitation operation is performed on this aggregated feature map, which is a self-gating mechanism with a sigmoid activation function. Formally, the gating mechanism is performed as:

$$\mathbf{s} = \sigma(\mathbf{W}_2 \times \mathrm{ReLU}(\mathbf{W}_1 \mathbf{z})) \tag{2.3}$$

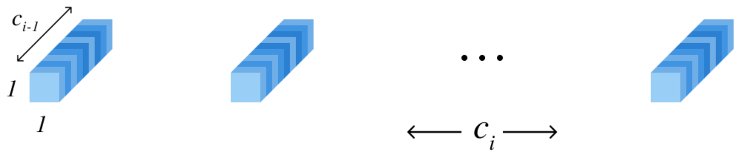
Where $\mathbf{z}$ is the output of the squeeze operation, and the dimensions of the parameters $\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{r} \times C}, \mathbf{W}_2 \in \mathbb{R}^{C \times \frac{C}{r}}$ are determined by a reduction ratio r. The recalibrated vector $\mathbf{s}$ is then multiplied channel-wise with the input feature map. The SE block can be easily inserted into any modern-day neural network with a small amount of computational overhead and has been shown to improve performance on multiple visual tasks. Additionally, similar blocks using attention mechanisms have been adopted in many segmentation frameworks for fusing features [29, 78].



(a) Standard convolution



(b) Depthwise filters



(c) Pointwise convolutions

Figure 2.1 Comparison of the parameters of standard convolution and depthwise separable convolutions. Standard convolution (a) uses $k \times k \times c_{i-1} \times c_i$ parameters. Depthwise-separable convolutions which uses $k \times k \times c_{i-1}$ parameters in the depthwise filtering stage (b) and $c_{i-1} \times c_i$ parameters in the pointwise convolution step (c).

2.1.2 Efficient CNNs

Deeper and larger convolutional neural networks have shown to be extremely good feature extractors on difficult vision datasets, but they also require enormous amounts of resources to train and are usually not practically usable in mobile applications such as robotics. In many computer vision tasks in robotics, it is often the case that inference needs to be run in real-time on a mobile device with limited computational power and low memory costs. This puts constraints on the number of parameters and inference time of deep neural networks. Therefore, the search for more efficient CNN architectures with fewer parameters to achieve similar performance has become an active field of research in the past few years [28].

Depthwise Separable Convolutions

Howard et al. [31] introduced depthwise separable convolution blocks to build an efficient convolutional neural network called MobileNet. Depthwise separable convolution is a factorised version of convolution layers that reduces the number of operations from a standard convolution. Consider a standard convolution layer with kernel size $k \times k$ requiring $k \times k \times c_{i-1} \times c_i$ parameters, where c_{i-1} and c_i are the number of output channels for the previous and current layer respectively (see Figure 2.1a). For an output dimension $h \times w$, the convolution operation has a computational cost of:

$$h \times w \times c_{i-1} \times c_i \times k \times k$$

The authors observed that such an operation involves two steps conceptually. The first step uses convolutional $k \times k$ kernels to combine filter features, and the second step combines these features into new representations with c_i channels. As such, one can split these two steps into two separate operations known as the depthwise separable convolution. In the first step, shown in Figure 2.1b, a depthwise separable convolutional filter applies a single $k \times k$ filter per input channel, with $k \times k \times c_{i-1}$ parameters and computational cost of $h \times w \times c_{i-1} \times k \times k$ [1].

The depthwise convolutional filter outputs a $h \times w \times c_{i-1}$ feature which is then passed into the pointwise convolution to create new features. The pointwise convolution operation is essentially a linear transformation with $c_{i-1} \times c_i$ parameters, with computational cost of $h \times w \times c_{i-1} \times c_i$, which finally outputs the $h \times w \times c_i$ feature with the same dimension as a standard convolution (see Figure 2.1c). The total operational cost of the depthwise convolution and pointwise convolution is:

$$h \times w \times c_{i-1} \times (c_i + k \times k)$$

Comparing this to the standard convolution, we have a reduction factor of $\frac{1}{c_i} + \frac{1}{k \times k}$. For most convolutional filters, we have $c_i > k^2$, then depthwise separable convolutions represent a reduction factor of around $1/k^2$. This would be around a 9-fold reduction for a kernel size of 3. A similar reduction factor also holds for the parameter count.

Howard et al. [31] also introduces a width multiplier to uniformly vary the output feature channels for every layer in the network, as well as a resolution multiplier for the input image. The former reduces both the memory and computational costs, whilst the latter reduces the computational costs only.

[1]We assume that the spatial dimensions of input and output features are the same to make notation simpler.

Inverted Residuals

MobileNetV2 [61] extends the mobile performance of MobileNet by introducing inverted residuals with linear bottlenecks. This module is motivated by the conjecture that manifolds of interest in neural networks can be embedded in low-dimensional subspaces. Consider a convolutional layer with an output tensor of shape $h_i \times w_i \times c_i$, it is believed that the information encoded by this tensor can be encoded into a lower dimensional manifold, which in turn can be embedded into a lower-dimensional subspace.

In general, for a layer transformation $ReLU(Bx)$ with non-zero volume S, the points mapped to *interior* S are obtained via the linear transformation B and unaffected by $ReLU$. This observation, coupled with the fact that manifolds of interest lie in lower dimensions, suggests that although *ReLU* collapses information in individual channels, information can still be completely preserved in other channels. This intuition suggests that bottleneck blocks used in common network architectures such as ResNet [27] contain all the necessary information, and thus the authors apply skip connections directly between these bottlenecks. Such inverted bottleneck design has shown to be much more memory efficient and recovers some of the regularizing power of depth-wise separable blocks.

The specific building blocks of the bottleneck residual block are as follows, first a pointwise convolution filter is applied to increase the channels from c_{i-1} to tc_{i-1}, where t is the expansion factor usually set to 6. This is then followed by a *ReLU* activation layer with a depthwise 3×3 filter followed by *ReLU*. Lastly, another pointwise convolution filter takes the channel number from tc_{i-1} to tc_i. The skip connection is made between the input and the output. The computational cost of the operation is

$$h \times w \times c_{i-1} \times t \times (c_{i-1} + k^2 + c_i)$$

Note that although this seems more expensive compared to depthwise separable convolutions, with an extra c_{i-1}, the increased expressiveness of the block means that the channel number is significantly reduced in the final network design. In addition, the authors highlight that the inverted bottleneck residual design has memory-efficient inference with the amount of memory dominated by the size of bottleneck tensors.

Network Scaling

He et al. [27] also vary the depth of the ResNet network to increase its expressiveness, and observed that deeper networks are capable of capturing more complex features. In [31, 61], the authors explored changing the input resolutions and the network width to trade-off accuracy with network size and complexity. The network width is increased by increasing the channel outputs of convolutional layers. Increasing network width tends to lead to networks that are more capable of capturing fine-grained features and that are easier to train. Similarly, increasing the input image resolution allows networks to capture more fine-grained features.

Tan and Le [68] proposed a unified framework of scaling networks by a constant compound ratio across resolution, depth, and width. The authors utilize neural architectural search to find a good baseline network architecture called EfficientNetB0, and observe better network scaling when all three dimensions are increased by a proportion. Such proportions are obtained via a small grid search. The baseline network architecture utilizes inverted residual bottleneck blocks with an additional squeeze and excitation block after the depth-wise filter, an attention mechanism that comes with slight computational overhead but performs feature recalibration. Furthermore, the authors utilize the swish activation function to increase network expressiveness, and stochastic depth in the network as a regularizer.

Neural architecture search

Another popular and more principled approach to finding efficient architectures is to utilize neural architecture search (NAS). NAS approaches commonly fix a set of building blocks with a few tunable parameters (e.g. width, depth) and seek to find a microarchitecture that optimises some objective by combining the building blocks. Tan et al. [67] utilises a neural architecture search approach with a multi-objective reward function that optimises the trade-off between network performance and inference time on CPU. A reinforcement learning agent with a recurrent neural network is used as the controller for sampling models and maximizing the expected reward using proximal policy optimization [62]. The resultant network, MNasNet, has around 2M parameters and combines inverted bottleneck blocks with kernel size 3 and 5, with squeeze and excitation blocks inserted within the block.

Howard et al. [30] uses a similar hardware-aware neural architecture search to produce MobileNetV3. The authors use a similar automated search algorithm to find an efficient neural network architecture tuned for mobile CPUs and minimize latency. Tan and Le [69] further uses NAS to find a training-aware network that minimizes the training time of the network. Interestingly, the authors observe that some of the depth-wise convolution blocks can be "fused" back into the standard convolution blocks to retrieve expressiveness. Importantly, such modifications are only made in the shallow parts of the network where the number of parameters is much lower. The authors also propose a stage-wise regularization schedule inspired by curriculum training [2], and adaptively increases the degree of regularization during training.

Other works

There have been various other methods of reducing the computational cost of CNNs over the years. Jaderberg et al. [36] proposes low-rank variations of the convolution layer by using $1 \times k$ and $k \times 1$ convolutions. Group convolution was introduced by [40] where output channels only receive information from input channels of the same group. Although this limits the field of view of the output channel to a specific subset of input channels. ShuffleNet [80] addresses this issue by utilising pointwise group convolution and channel shuffling to reduce the computation cost of the network. In SqueezeNet [35], the authors utilize the fire module which is comprised of the squeeze operation, using 1×1 filters followed by the expansion operation, using both 1×1 and 3×3 filters. The authors further prune the network using techniques such as deep compression [25] to reduce the network size for inference use.

2.1.3 Image Segmentation

The task of image segmentation was an extension of successes from early works on image classification [40, 66]. The goal of image segmentation is to produce per-pixel classifications of the input image and produce a fine-grained prediction output. In the following, we discuss prominent segmentation methods, such as FCN, Encoder-Decoder architectures, atrous convolutions, and multi-branch methods.

Fully Convolutional Network

Long et al. [44] was the first to propose the idea that fully connected layers can be completely removed from deep CNNs in the context of image segmentation. Traditional CNNs output a final vector prediction for classification via a series of downsampling or strided convolutions that continually coarsens the features in the spatial dimension. The authors showed that coarse, structural information retained by intermediate convolutional layers can be extracted and upsampled back. Such upsampling tasks can be performed using bilinear upsampling or *deconvolution layers*, retrieving per-pixel semantic classification. The resultant architecture, termed fully convolutional networks (FCN), can be trained end-to-end with relatively high performance and efficiency. Another benefit of such architecture is that pre-trained convolu-

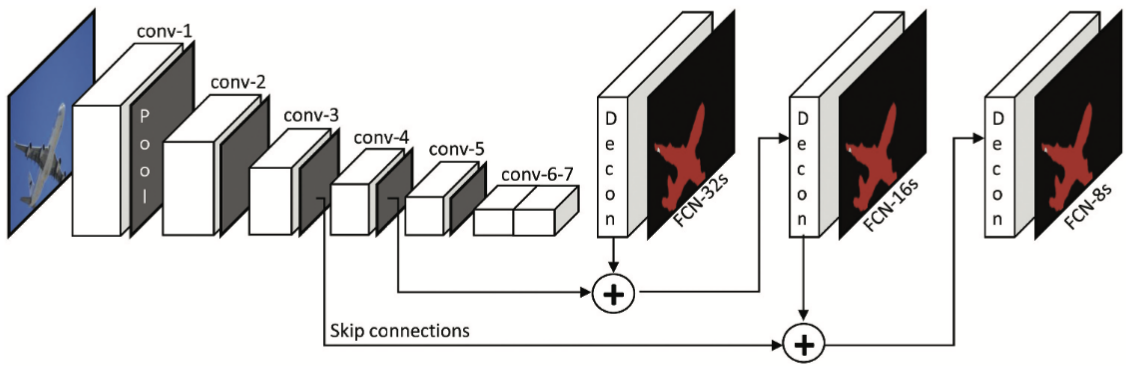


Figure 2.2 The FCN architecture upsamples intermediate features from standard convolutional neural networks using deconvolution layers and combines features via residual connections.

tional neural networks can be utilized via transfer learning, and the network only needs to learn the upsampling modules.

Encoder-Decoder Architecture

Following the work of FCN, several architectures emerged that involved a downsampling part (encoder) of the network with an upsampling part (decoder). DeconvNet [56] includes a deeper decoding network with several convolution filters at each spatial dimension and utilizes deconvolution layers to upsample the features. Bilinski and Prisacariu [5] explored dense decoder shortcuts, where in addition to the skip connections between encoder and decoder, skip connections are also added in the decoder to form a densely connected decoder that fuses features at different scales. UNet [59] was first proposed as an image segmentation network for medical image segmentation, but its simple and effective architecture has also shown promising results in a wide range of other domains [1, 77]. UNet uses a symmetrical encoder and decoder architecture, where the encoder gradually decreases spatial dimensions and increases feature dimensions, and the decoder performs the operations in reverse. Importantly, UNet utilizes skip connections between the encoder and decoder via concatenation of the features, this is shown in Figure 2.3. The idea is that such skip connections allow the transfer of denser features from the encoder to the decoder and accelerate learning. Jégou et al. [37] extends the u-shape architecture but using densely connected blocks from DenseNet [33], with similar skip connections between the encoder and the decoder.

SegNet [1] is another prominent encoder-decoder architecture that utilizes pooling indices from the encoder in the decoder. This operation is extremely lightweight and requires very little memory, but is still able to transfer some long-distance information from the encoder to the decoder. On the other hand, the authors also showed that retaining the complete feature

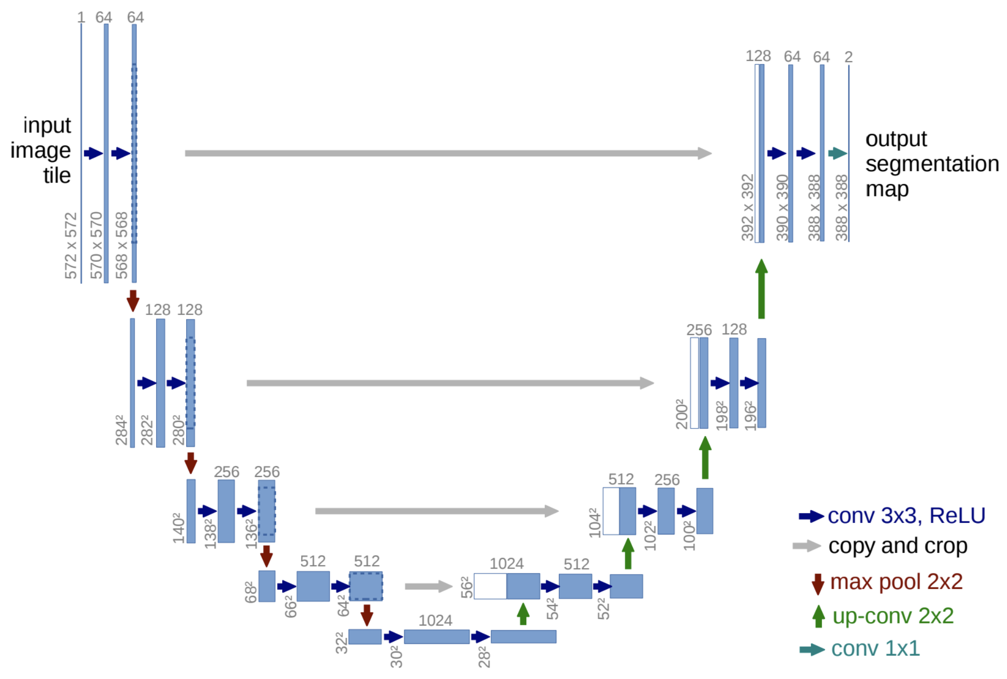


Figure 2.3 The UNet architecture uses a symmetrical encoder and decoder, with skip connections to transfer features from intermediate layers in the encoder to corresponding layers in the decoder.

from the encoder yields higher performance, but at the cost of lower inference time. Later works have also developed Bayesian variations of the network using dropout to perform Bayesian inference [38].

Atrous Convolutions

More recently, atrous convolutions (or dilated convolutions) have shown to be an effective way of performing image segmentation in a series of works on the DeepLab architectures [11–13]. Atrous convolutions have a higher field of view compared to regular convolutions but with the same parameterisation, this is achieved via "holes" in the input field. By utilizing atrous convolutions, a higher field of view can be achieved without reducing the spatial dimension of the feature space, thereby rendering the decoding operation easier. In [11], authors combined atrous convolution with atrous spatial pyramid pooling (ASPP) to combine multi-scale features. The coarse output is then bilinearly upsampled and refined using a condition random field (CRF). In [12], the authors compute atrous pooling at different scales but in parallel, with an additional global image-level feature. Such architecture was able to produce high-quality results even without CRF post-processing. Finally, in [13], the authors utilize a decoder module to refine the coarse predictions, reporting improved results from previous DeepLab versions.

Multi-branch Methods

A downside of atrous convolution based approaches is that intermediate features have to be very large to retain both spatial resolution and contextual information. In general, there always exists a tension in segmentation networks between preserving spatial information and obtaining contextual information. Yu et al. [78] proposes a bilateral segmentation network (BiSeNet) to alleviate this tension with two separate paths for extracting spatial features and contextual information separately. The authors also propose fusion blocks to combine the features using attention mechanisms. This method of performing segmentation is extremely efficient and fast, and there's an active field of research in using these multi-branch models to perform real-time segmentation [29].

Other Approaches

For completion, we also briefly mention that image transformers [14] have recently shown to be a very promising direction for image segmentation. There are works that attempt to make transformers more efficient [73], as well as ways of computing Bayesian inference with transformers [53]. However, this thesis only focuses on well-established encoder-decoder architecture that has also shown to be compatible with Bayesian approximation methods such as MC Dropout.

2.2 Bayesian Reasoning

For autonomous robotic agents, it is crucial for the algorithm be capable of providing reliable uncertainty estimates [4]. The field of Bayesian learning seeks to capture the uncertainty of learning systems. In particular, Bayesian Deep Learning (BDL) seeks to perform Bayesian learning in deep neural networks. This section begins with a brief introduction to Bayesian inference, followed by existing literature on BDL methods.

2.2.1 Bayesian Inference

Consider a dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$. In Bayesian learning, we assume a generative model of the data $p(\mathcal{D}|\theta)$, coupled with a prior belief on the model parameters $p(\theta)$, and infer a posterior distribution $p(\theta|\mathcal{D})$ from the data $\mathcal{D}$. The posterior can be computed using the Bayes Theorem [6]:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{2.4}$$

Where $p(\mathcal{D})$ is the model evidence computed as the integral $\int p(\mathcal{D}|\theta)p(\theta)d\theta$. The term $p(\mathcal{D}|\theta)$, represents the likelihood that the observed data $\mathcal{D}$ is generated by the underlying model parameterized by θ. This likelihood is typically the objective function of the learning problem, such as the cross-entropy loss, which measures the log-likelihood.

The predictive distribution on a new test point x^*, y^* is computed as

$$p(y^*|x^*, \mathcal{D}) = \int p(y^*|x^*, \theta)p(\theta|\mathcal{D})d\theta \tag{2.5}$$

However, the posterior described in Equation 2.4 is intractable for complex models such as neural networks, which effectively renders the predictive distribution $p(y^*|x^*, \mathcal{D})$ intractable in Equation 2.5. As such, BDL methods try to estimate the posterior using approximate bayesian inference techniques such as variational inference, Laplace approximation, and sampling approaches, which we will discuss next.

2.2.2 Bayesian Deep Learning

Variational Inference

Variational inference (VI) methods approximate the posterior $p(\theta|X,y)$ with a simpler and tractable parameterised distribution $q(\theta)$. The idea of VI is to find variational distribution $q(\theta)$ that minimizes the Kullback-Leibler divergence between the variational distribution and the true posterior:

$$\mathrm{KL}(qp) = \mathbb{E}_q\left[\log\frac{q(\theta)}{p(\theta|X,y)}\right]$$

However, the KL divergence cannot be optimized directly. Instead, the evidence lower bound, also known as the free energy, is optimized. One can show that maximizing the ELBO objective effectively minimizes the KL divergence.

$$\mathrm{ELBO} = \mathbb{E}_q\left[\log\frac{p(y|X,\theta)}{q(\theta)}\right] \tag{2.6}$$

ELBO can be optimized using a mini-batch of data using stochastic variational inference with Gaussian priors [20]. Blundell et al. [7] further introduced Bayes By Backprop, optimizing the free energy in Equation 2.6 directly using stochastic minibatch gradients and backpropagation. Kingma et al. [39] further extended the method using the local reparameterization trick, which has empirically shown to reduce the variance of the gradients in most cases. Lastly, Gal and Ghahramani [16] showed stochastic regularization methods to deep neural networks can be used for approximate Bayesian inference. This method is central to this thesis and is discussed in detail in the following sections.

Laplace Approximation

Laplace Approximation estimates the posterior distribution with a multivariate normal distribution. The approach is motivated by taking the second-order Taylor series expansion of the log posterior over the weights around the maximum a-posteriori (MAP) estimate, where the quadratic term appears in the same form as in the log-likelihood in a Gaussian Model [47].

$$\log p(\theta|\mathcal{D}) \approx \log p(\theta^*|\mathcal{D}) + \frac{1}{2}(\theta-\theta^*)^T\mathbf{H}(\theta-\theta^*)$$

The Laplace approximation has the nice property that the mode is centred at the same mode as the true posterior. However, by the nature of the Gaussian distribution, the Laplace approximation fails to capture multi-modal posteriors. Another bottleneck of the Laplace approximation is computing the second-order Hessian matrix, which is often very expensive to compute for modern-day neural networks. Literature has explored approximating the Hessian using off-diagonal elements [60] or low-rank approximations. In Grosse and Martens [21], the Kronecker factorization of the approximate block-diagonal Hessian is applied to obtain scalable Laplace Approximation for neural networks.

Sampling Approaches

Sampling methods approximate probability distributions non-parametrically via sampling. Popular algorithms include rejection sampling, importance sampling, and Markov Chain Monte Carlo sampling (MCMC) [6]. Methods such as rejection sampling and importance sampling suffer in high dimensions and are usually not used in practice. Notably, Hamiltonian MCMC is proposed for neural networks by Neal et al. [55]. Welling and Teh [74] proposed to combine stochastic gradients with Langevin dynamics, demonstrating that variational inference can be performed by adding noises to the stochastic gradient descent process. There are further extensions to this method using second-order information such as the Hessian [46, 64], and utilizing the wake-sleep algorithm [8].

Other Approaches

There are a few other notable approaches and directions in the Bayesian deep learning community. Several works use deterministic models and their logit outputs to quantify the uncertainty. Dirichlet prior networks output the parameters of a Dirichlet distribution, which acts to update the distribution of categorical distributions to output the predictions [48]. Evidential Neural Networks [63] interprets the logits as multinomial beliefs to output predictions for the classes as well as an additional class that captures the uncertainty. Test-time augmentation techniques have been frequently adopted in medical applications [72, 51], where image augmentation is applied at test time for stochastic forward to obtain an approximate predictive distribution.

2.2.3 BDL for Computer Vision

In the following we discuss two BDL methods that have been predominantly used in the computer vision setting, namely dropout-based methods and ensemble-based methods.

MC Dropout

Dropout was first proposed by Srivastava et al. [65] as a way to regularize neural networks. The idea of dropout is to drop a random fraction of units and their connections during training, effectively "thinner" models are being trained. At test time, the full network is recovered by activating all the units. The motivation for dropout is to prevent co-adaptation of the units and prevent the network from overfitting. When dropout layers are turned off during inference (known as weight averaging), one can also interpret the result as the average result from an ensemble of smaller networks. Since the introduction of the dropout layer, it has been widely used in almost all deep learning frameworks as a regularization technique. We discuss the Dropout layers in more detail in Section 3.2.

Gal and Ghahramani [16] showed that in addition to its regularization properties, dropout training can actually be interpreted as a Bayesian Approximation of a Gaussian process (GP). Furthermore, training with the dropout objective minimises the Kullback-Leibler divergence between an approximate distribution $q(\omega)$ and the posterior $p(\omega|\mathbf{X},\mathbf{y})$ of a GP model integrate with respect to the finite rank covariance function parameters ω. Recall that a predictive probability of a deep GP model is given as:

$$p(\mathbf{y}|\mathbf{x},\mathbf{X},\mathbf{Y}) = \int p(\mathbf{y}|\mathbf{x},\omega)p(\omega|\mathbf{X},\mathbf{y})d\omega$$

$$p(\mathbf{y}|\mathbf{x},\omega) = \mathcal{N}(\mathbf{y};\hat{\mathbf{y}}(\mathbf{x},\omega,\tau^{-1}\mathbf{I}_D)$$

The authors showed that the posterior distribution $p(\omega|\mathbf{X},\mathbf{y})$ can be approximated with a variational distribution $q(\omega)$ defined as:

$$\begin{aligned} &\mathbf{W}_i = \mathbf{M}_i \cdot \text{diag}\left([z_{i,j}]_{j=1}^{K_i}\right) \\ &z_{i,j} \sim \text{Bernoulli}(p_i) \quad \forall i = 1,...L, j = 1,...,K_{i-1} \end{aligned}$$

Furthermore, the authors showed that minimising the common cross-entropy loss objective has the effect of minimising the KL divergence between the variational distribution and the posterior:

$$\text{KL}(q(\omega)||p(\omega|\mathbf{X},\mathbf{y}))$$

This theoretical result suggests that T stochastic forward passes at test time with dropout turned on can be used to approximate the posterior prediction, whilst the variance of the passes can be interpreted as a measure of uncertainty. Kendall et al. [38] combines this technique with the SegNet architecture to produce uncertainty estimates from the network,

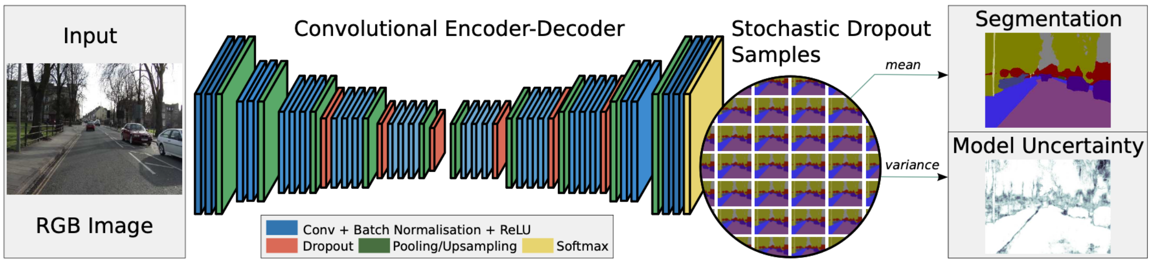

Input
RGB Image
Convolutional Encoder-Decoder
Conv + Batch Normalisation + ReLU
Dropout
Pooling/Upsampling
Softmax
Stochastic Dropout Samples
mean
variance
Segmentation
Model Uncertainty

Figure 2.4 The Bayesian Segnet uses the Segnet architecture with dropout included in the deepest encoders and decoders, T stochastic forward passes are performed at test time with dropout layers turned on, and the mean of the predictions are taken as the prediction, the variance as the uncertainty

shown in Figure 2.4. In addition, the authors reported higher network performance in accuracy when the networks were trained with dropout layers.

Ensembles

Recall that summing over stochastic forward passes has the intuitive interpretation as an ensemble of smaller networks. This leads to the natural extension of using deep ensemble methods [26] to perform Bayesian approximation. Ensemble methods seek to find a predictive distribution using a combination of individual predictors, such as averaging their predictions:

$$p(y|x) = \frac{1}{T}\sum_{i=1}^{T} p_{\theta_i}(y|x, \theta_i)$$

where we have T realizations of the network each with parameter θ_i. Recently, Lakshminarayanan et al. [41] showed that standard loss functions are proper scoring functions that reward better-calibrated predictions over worse ones. Additionally, deep ensemble models with even just $T = 5$ can be a robust framework to improve uncertainty quality for all cases such as identifying out-of-distribution samples, and calibrated uncertainty scores. The authors were able to demonstrate results on image datasets such as MNIST and ImageNet. Gustafsson et al. [24] further indicated that ensemble methods improve uncertainty quality over MC-Dropout methods.

A clear downside of ensembles, however, is that it's difficult to scale on mobile devices. As such, several recent research studies have attempted to perform more efficient variations of deep ensembles. Pruning approaches use diversity measures to reduce the complexity of ensemble models by removing some of them [23]. Distillation approaches use student-teacher frameworks to teach a single network to represent the knowledge of an ensemble [10]. Lastly, techniques such as sub-ensembles [70] and batch-ensembles [75] are proposed to reduce the computation of ensembling by sharing subsets of components among predictors.

Dropout Variants

Mobiny et al. [50] showed that similar to Dropout, Drop-connect can also approximate the variational distribution of the posterior. Rather than de-activating nodes in a network, the weights of the parameters are dropped stochastically. The authors reported the method to produce more robust uncertainty estimations. Gal et al. [17] proposed a dropout variant by replacing the discrete Bernoulli distribution with a concrete distribution relaxation. In this setting, the dropout probability is tunable using gradient methods, and the authors demonstrated improved uncertainty estimates. Some further studies have also indicated that concrete dropout improves over MC-Dropout in several metrics [38, 52].

On the other hand, more recently there has been a re-evaluation of dropout-based Bayesian inferences, where several studies observed dependence of uncertainty quality on network architectures [71] and general high variability of the uncertainty outputs [15]. This thesis further investigates the properties of Bayesian inference with MC dropout, and how architectural decisions such as skip connections, efficient convolutional blocks, and network scales can affect the quality of uncertainty approximations.

Methods

The methodology of this thesis follows a two-fold process, tackling first the *efficiency* aspect of neural networks, then the *reliability* aspect of the networks. First, we utilize efficient building blocks from state-of-the-art networks to build a scalable and efficient encoder-decoder architecture. This is discussed in detail in Section 3.1, where we perform preliminary experiments to test different upsampling methods and choose the best decoder design as our baseline network. Secondly, we incorporate the network into practical Bayesian frameworks such as MC Dropout in Section 3.2, and we introduce a novel Bayesian approximation method using stochastic depth. Lastly, in Section 3.3 we state the metrics that we will be using to both evaluate the predictive performance of the network and the quality of its uncertainty outputs.

3.1 Efficient Network Design

Many segmentation architectures have been explored in the literature recently, such as the encoder-decoder architecture [59, 1], atrous convolution-based methods [12], and more recently, bilateral segmentation networks [78] and transformer-based networks [73]. Dropout-based Bayesian inference in segmentation architectures has been introduced [38] and predominantly validated in encoder-decoder frameworks [79]. Therefore, the first part of this thesis specifically focuses on *efficient designs* of the encoder-decoder architecture. Our methodology involves designing an efficient decoding path, modifying the network with the location of skip connections, the type of skip connections (concatenation vs. addition), and the scale of the network (width and depth). We examine the effect on both the performance of the network measured in accuracy and IoU, as well as the effect on the number of parameters and runtime.

3.1.1 Network Architecture

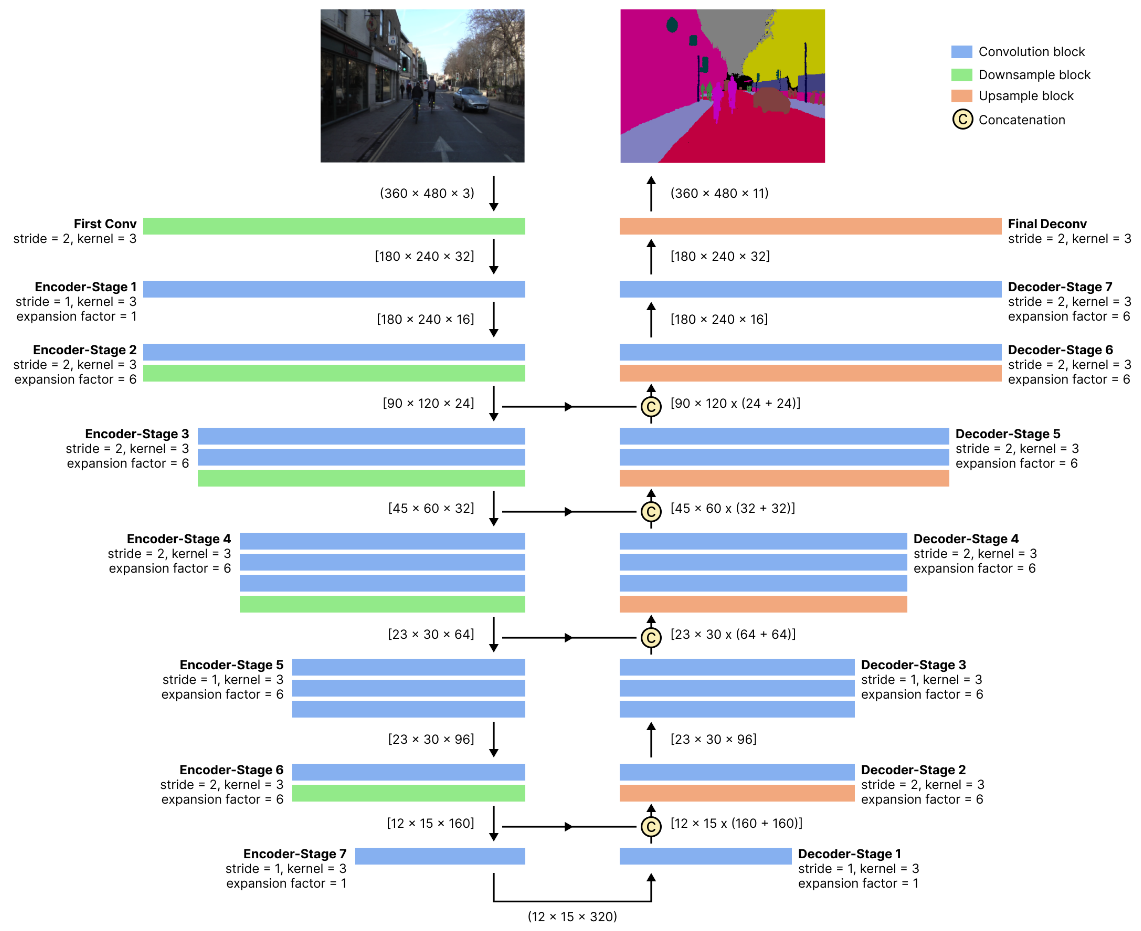


Figure 3.1 Baseline architecture of the efficient segmentation network design. MobileNetV2 is used as a backbone encoder for high efficiency, and a symmetrical decoder is built using linear bottleneck layers and depthwise separable deconvolution blocks. Skip connections are included every time the network upsamples or downsamples, to maintain information flow at every spatial dimension.

The basic design process of our network architecture is as follows: we first compare and examine state-of-the-art CNNs to select good candidates for the encoder architecture, then we utilise efficient building blocks introduced in [31, 61, 32] to build the decoder. Similar to [59], we adopt skip connections at locations where the spatial dimension changes via downsampling or upsampling to transfer the features from the encoders to decoders. This should help the network to learn both faster and produce more fine-grained features, and we further investigate the effect of skip connections in experiments described in Section 4.2.1.

Figure 3.1 shows an illustration of the *baseline* network used in this thesis. The network uses an efficient convolutional neural network as the backbone encoder, the specific choice and reasoning are discussed in Section 3.1.2. The encoder consists of several stages, where all stages perform a series of convolution blocks and some stage decreases the spatial dimension (stride $s > 1$). Each stage consists of 1 or more convolution blocks, and each block includes a batch-normalization block and a non-linear activation function such as *ReLU*. The details of the block design are discussed in Section 3.1.3.

This encoder-decoder design has a few important properties that are worth mentioning. First, the direct transfer of features from the encoder to the decoder is an easy way to recover spatial information lost from the encoder. The intermediate tensors are also relatively light-weight compared to intermediate features obtained using atrous convolutions, where high spatial resolution is maintained throughout the network [11]. The second property is that concatenating the features via skip connections similar to [33], rather than adding them like in residual blocks, results in the widening of the features but also increased parameter count. Finally, the intermediate features need to be stored and held in memory during the forward pass, and can only be released when the corresponding decoder block is reached. Therefore there's a trade-off between retaining high spatial dimension features vs. computational resources, where more fine-grained features can be retained at the cost of increased inference time and memory costs. We examine this trade-off in more detail in Section 4.2.1 by modifying skip connections.

3.1.2 Backbone Encoder

Table 3.1 SOTA CNNs' top 1 performance on ImageNet1K

Encoder	Accuracy	Params (M)	GFLOPs
ResNet18 [27]	69.8	11.7	1.81
MobileNetV1 [31]	70.6	12.7	0.59
MobileNetV2 [61]	71.9	**3.5**	**0.3**
DenseNet121 [33]	74.4	8.0	2.83
EfficientNetV1(B0) [68]	77.7	5.3	0.39
EfficientNetV1(B1) [68]	78.6	7.8	0.69
EfficientNetV1(B2) [68]	80.6	9.1	1.09
EfficientNetV2(S) [69]	**84.2**	21.5	8.37

Table 3.1 shows the top-1 accuracy of the state-of-the-art CNNs on ImageNet, ordered by the performance. An immediate observation from the table is that *not all parameters are equal*, and that the network architecture can drastically influence the parameter-to-performance ratio. For instance, whilst both ResNet18 and EfficientNetV1(B2) have around 10M parameters, the latter has a much higher performance. We highlight that MobileNetV2 is one of the lighter networks with still relatively good performance-to-parameter ratio. We

chose MobileNetV2 as the backbone encoder because it has the fewest number of parameters. In theory, our encoder can be replaced by any of the other CNNs and adopted in the encoder-decoder frameworks.

3.1.3 Decoder Design

In the encoder-decoder architecture, the purpose of the decoder is to recover the spatial dimension and to process additional contextual information from the feature. In the following, we discuss and experiment with different decoder design choices to upsample features, which are used in the decoder blocks (blue and orange) in Figure 3.1. We experiment with three variations: 1) using standard deconvolution layers, 2) using depth-wise separable (dw) deconvolution layers, and 3) using inverted residuals with dw-deconvolutional layers. We perform preliminary tests by building the networks and show that inverted residual decoders are the optimal choice that uses very little memory whilst maintaining high performance.

Standard Deconvolution

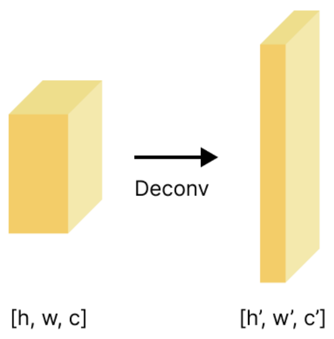


Figure 3.2 Standard deconvolution takes in feature of dimension $h \times w \times c$ and outputs $h' \times w' \times c'$

The standard deconvolution layer performs the opposite operation to a standard convolution layer, with exactly the same number of parameters, but reversing the input and output channels. The layer takes in an input feature with dimensions $h \times w \times c$, and outputs a feature with dimension $h' \times w' \times c'$. Typically we consider an upsampling ratio of 2, and usually $c' < c$ in the upsampling stages. Similar to regular convolutions, the computational cost of this operation is:

$$h \times w \times c \times c' \times k \times k$$

and requires $k \times k \times c \times c'$ memory. The visualization of the feature input and output are illustrated in Figure 3.2.

Depthwise Deconvolution

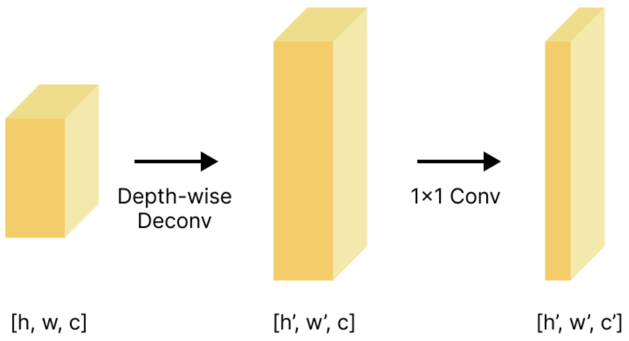


Figure 3.3 Depth-wise deconvolution takes in feature of dimension $h \times w \times c$, and first performs a filtering step to increase the spatial dimension to $h' \times w'$ without changing the number of channels, then performs a point-wise convolution to combine the representations into new representations with c' channels. Compared to standard deconvolution layers, the depthwise deconvolution uses fewer parameters.

Similar to depthwise separable convolutions, we consider depthwise separable deconvolution that works with a similar principle. The idea is to disentangle the single deconvolution step into two steps. Starting with an input tensor of size $h \times w \times c$, we first perform a filtering step that uses a depthwise deconvolution layer to expand the spatial dimension of the feature tensor without increasing the depth. Then, we apply a pointwise 1×1 convolution to recombine the features into a smaller dimension c', outputting a final feature with dimensions $h' \times w' \times c'$. The computation cost of this operation is:

$$h \times w \times c \times (k \times k + c')$$

Comparing this to the regular deconvolution operation, we achieve a reduction factor of $\frac{1}{c} + \frac{1}{k^2}$, which is typically 8 to 9 fold reduction in both parameters and computational costs.

Inverted Residual Decoder

Finally, we consider the inverted bottleneck design where features are first expanded before they are reduced back to the desired dimension. This operation is illustrated in Figure 3.4, showing the inverted bottleneck operation on a $h \times w \times c$ tensor with expansion factor t. The input feature is first expanded channel-wise to tc channels, and the feature selection step is performed on the expanded feature using depthwise deconvolution as described previously, leading to an output of $h' \times w' \times tc$. Finally, the tensor is "squeezed" back into the desired

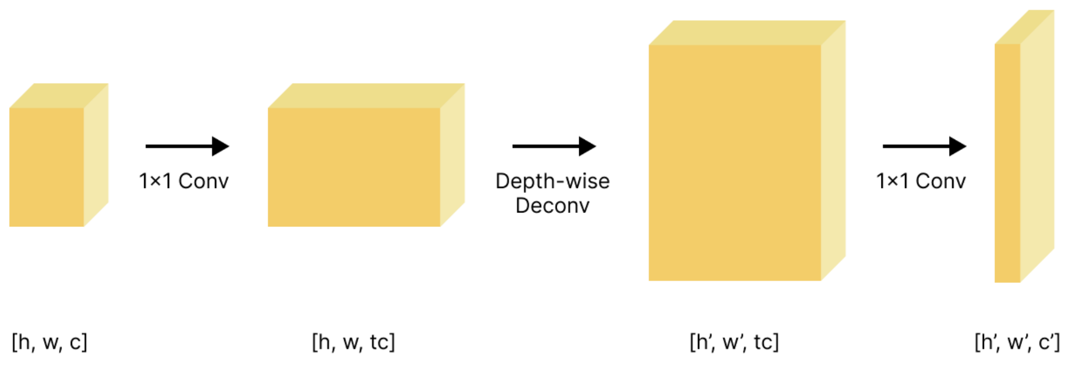


Figure 3.4 The inverted bottleneck design first expands the feature input with more channels, often with an expansion factor of $t = 4$ or $t = 6$. A depth-wise filtering step is performed on the expanded channels to increase the spatial dimensions, and point-wise convolution is applied to output the desired number of channels c'.

channel depth c', outputting $h' \times w' \times c'$. The total cost of the operation is

$$h \times w \times c \times t \times (c + k^2 + c')$$

Whilst this is more expensive than depthwise separable deconvolutions, it is empirically shown that this operation is much more expressive than the depthwise convolution. As a result, the module design requires less width. We show in the following preliminary results that with the same number of parameters, the inverted bottleneck design achieves much higher performance than the depthwise decoders.

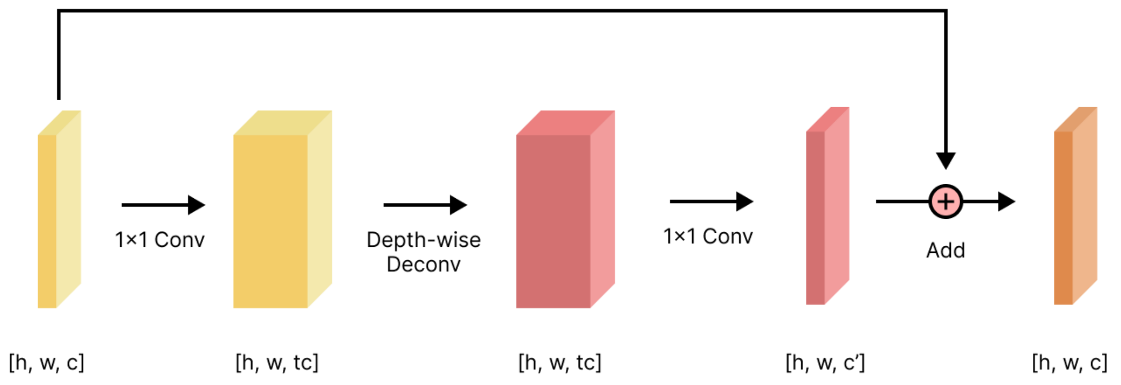


Figure 3.5 The inverted residual block design can be applied to any convolution blocks where the spatial dimension of the input and output feature is the same (i.e. stride=1), this design adds an additional residual path from the input to the output.

An additional implementation detail for the inverted bottleneck layers is inverted residuals. This is illustrated in figure 3.5, where the input is connected to the output feature via a residual connection. This is motivated by Sandler et al. [61] where the authors empirically show that such designs can produce expressive and lightweight networks. Note that the inverted residual block can be used whenever the dimension of the input and the output are equal, such as the blue blocks shown in Figure 3.1.

Preliminary Results

Table 3.2 Preliminary results of different decoder designs. Params(d): number of parameters for decoders in millions, Params(t): total number of parameters of the network in millions, Time: inference time measured in ms on GPU, Memory: memory usage measured in MB on GPU

Methods	mIOU	Params (d) ↓	Params (t) ↓	GFLOPs ↓	Time(ms) ↓	Memory
Regular	63.91	1.52	3.34	2.26	8.03	103
Depthwise	64.60	**0.19**	**2.00**	**1.38**	**7.88**	**124**
Inv-Res-0.75	65.69	1.04	**2.06**	1.56	10.54	167
Inv-Res(baseline)	**66.61**	1.86	3.67	2.24	11.47	178

We perform a preliminary experiment on the three decoder designs described above on the CamVid Dataset. We evaluate the performance of the network using mIOU, and we quantify the efficiency of the network using the number of parameters, FLOPS, runtime, and memory requirement. Training details follow the same recipe that is described in detail in Section 4.1.

Table 3.2 shows the results of the preliminary experiment. First, we see that the depthwise decoder does in fact reduce the number of parameters in the decoder by about 8-fold, and achieves the fastest inference speed out of all three experiments. Furthermore, it does not hurt performance compared to regular decoders. Our results show that the inverted residual design achieves the highest mIOU of 66.61, but uses most parameters out of all three experiments.

For fair comparison to the depthwise decoder, we reduce the number of total parameters of *inv-Res* via width and depth scaling by a factor of 0.75 (*Inv-Res-0.75*) on both the encoder and the decoder to obtain around 2M parameters. The scaled-down version of the network has an encoder that is strictly worse than the regular encoder, so one would expect performance to decrease. However, we show that even with the same number of parameters, the inverted residual design still outperforms the depthwise decoder. This suggests that the inverted residual decoder is able to recover a significant portion of the information lost in the decoder, and the performance-to-parameter trade-off is better. We therefore use *Inv-Res* as the baseline for all our future experiments. In the following chapter, we will further attempt to reduce the run time and memory requirements with experiments in Section 4.2.1.

3.2 Bayesian Inference

The central idea of Bayesian neural networks (BNN) is to sample stochastic predictions that approximate the predictive distribution obtained via the posterior. In this thesis, we focus on two main methods of achieving BNN: MC Dropout, and stochastic depth. MC Dropout is already a predominantly used method in literature [16] and serves as a good benchmark on our network. We introduce a novel method of Bayesian inference with stochastic regularization using stochastic depth. We will provide some intuition on why this method is suitable for the Bayesian framework, and demonstrate empirical evidence to show its effectiveness. The following section describes the two regularization methods in detail and outlines how network prediction and uncertainty quantification are performed at test time.

3.2.1 MC Dropout

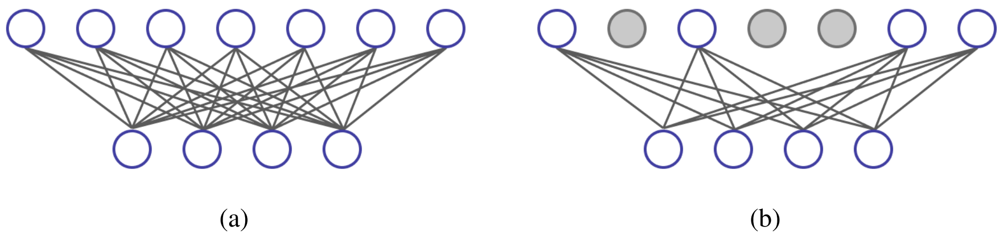


Figure 3.6 Example illustration of how dropout layers work with two layers of units. a) shows a fully connected layer with all units activated, b) shows the same layers with only 50% of the units activated.

In Section 2.2.3 we have already discussed the Bayesian interpretation of MC Dropout formalized by Gal and Ghahramani [16]. Here we state the dropout mechanism in full detail and include the implementation methods. Consider a regular neural network layer with *ReLU* activation function, we can write the forward pass as follows:

$$\mathbf{y}_{\ell+1} = \mathrm{ReLU}(\mathcal{F}(\mathbf{y}_\ell, \{\mathbf{W}_{\ell+1}, \mathbf{b}_{\ell+1}\})) = \mathrm{ReLU}(\mathbf{W}_{\ell+1}\mathbf{y}_\ell + \mathbf{b}_{\ell+1}) \tag{3.1}$$

Where $\mathbf{y}_\ell \in \mathbb{R}^{d_\ell}, \mathbf{y}_{\ell+1} \in \mathbb{R}^{d_{\ell+1}}$ are the outputs of layer ℓ and $\ell+1$ respectively, and the parameters $\mathbf{W}_{\ell+1}, \mathbf{b}_{\ell+1}$ are the associated weights and biases of layer $\ell+1$. Dropout layers modify this feedforward operation with a Bernoulli variable $\mathbf{z}_\ell \in \mathbb{R}^{d_\ell}$ as follows:

$$z_j^{(\ell)} \sim \text{Bernoulli}(p)$$
$$\mathbf{y}_{\ell+1} = \text{ReLU}(\mathbf{W}_{\ell+1}(\mathbf{z}_\ell * \mathbf{y}_\ell) + \mathbf{b}_{\ell+1})$$

Where $z_j^{(\ell)}$ represents the jth element in $\mathbf{z}_\ell$, and $(*)$ represents per-element multiplication. Effectively, dropout layers zeros, or masks p proportions of the previous layers activation. An intuitive illustration of the dropout layer is shown in Figure 3.6

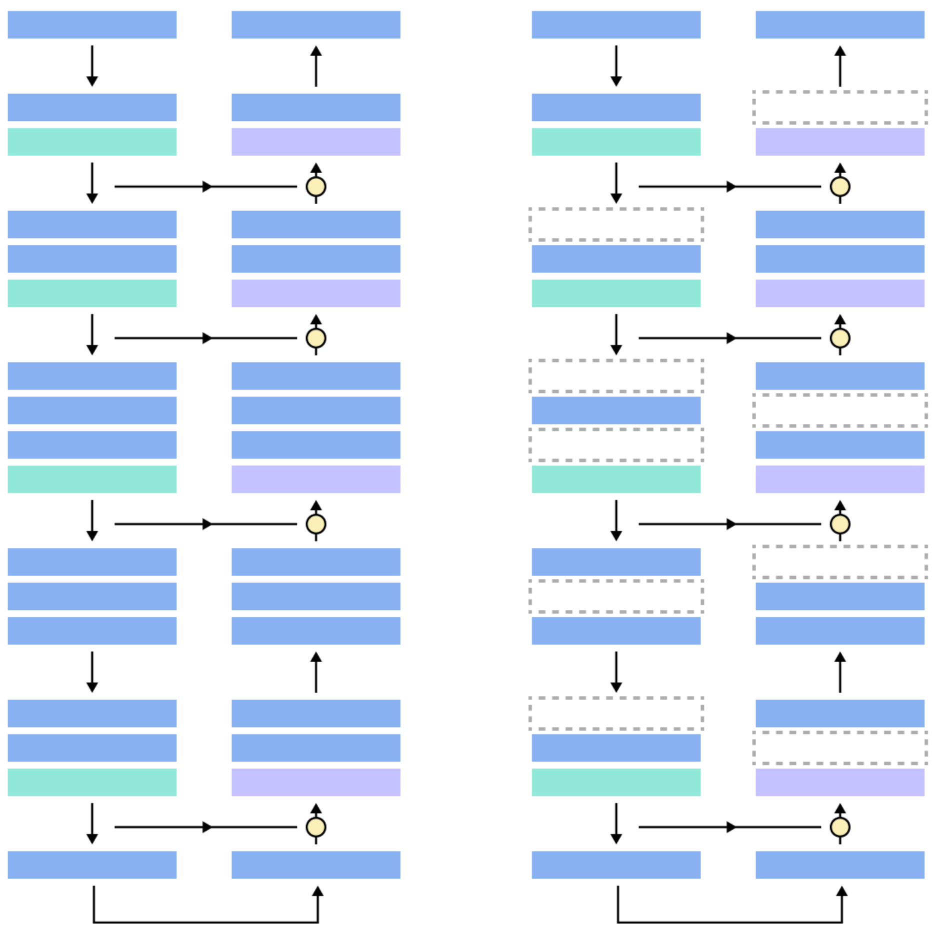
3.2.2 Stochastic Depth

Whilst residual blocks alleviate the vanishing gradient problem to a certain extent, extremely deep CNNs can still be difficult to train. Stochastic depth is a regularization technique introduced by Huang et al. [34] to help with the training process of deep networks. The idea of stochastic depth is to use shorter networks during training by stochastically skipping a proportion of layers, and the full network is retrieved at test time. An example illustration of stochastic depth applied to the baseline network is shown in Figure 3.7. This is performed with a simple Bernoulli random variable b_ℓ for each layer, with "survival" probability $p_\ell = P(b_\ell = 1)$.

$$y_\ell = \text{ReLU}(b_\ell \mathcal{F}(\mathbf{x}, \{W_\ell\}) + \mathbf{x}) \tag{3.2}$$

The authors showed that a simple linear decay rule $p_\ell = 1 - \frac{\ell}{L}(1 - p_L)$ can be used effectively to set the survival probability of each layer, with survival probability decreasing in deeper layers, where p_L is the survival probability of the last layer. We apply a similar probability assignment with the deepest part of the network having the lowest survival probability. Stochastic depth empirically increases the gradient flow during training, and yields much better results with deep models.

An interesting property is of using stochastic depth at test time can be interpreted as an *ensemble of shallower networks* with the same base building blocks but different depths. This presents a natural interpretation similar to the MC dropout, where T stochastic forward passes can be performed to produce a Bayesian approximation. The number of possible network configurations grows exponentially with the number of blocks, and we can sample networks with different depths with each stochastic pass. Therefore, we can interpret Bayesian inference with stochastic depth as a Bayesian method that captures uncertainty in the structure of the network, specifically its depth.



(a) Baseline network (b) Baseline network with stochastic depth

Figure 3.7 Example illustration of how stochastic depth could be applied to the baseline network. a) shows the network with all blocks activated, and b) shows an instance of a shallower network with stochastic depth, where blocks with dashed lines indicate blocks that are turned off (i.e. bypassed with identity functions).

3.2.3 Uncertainty Quantification

Consider that we perform T stochastic forward passes with any regularization techniques, such as MC Dropout or stochastic depth as described above. We can approximate the expected softmax outputs as

$$p(y=c|x,\mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y=c|x,w_t) \tag{3.3}$$

where w_t represents an instance of a realized weight tensor, or a sample of the weight, and $p(y=c|x,w_t)$ is the softmax probability for class c which the network predicts with weights w_t.

The uncertainty of the network is commonly computed using mutual information or predictive entropy [19]. We can measure the predictive entropy or the information given in the expected softmax probabilities as

$$\mathbb{H}(y|x,\mathcal{D}) = -\sum_c \left(\frac{1}{T} \sum_{t=1}^{T} p(y=c|x,w_t) \right) \log \left(\frac{1}{T} \sum_{t=1}^{T} p(y=c|x,w_t) \right) \tag{3.4}$$

where $\mathcal{D}$ is the training set. The mutual information measures the mutual dependence between the information given in the expected softmax output and the expected information in the softmax output

$$\mathbb{I}(y,w|x,\mathcal{D}) = \mathbb{H}(y|x,\mathcal{D}) - \mathbb{E}_{w\sim p(w|\mathcal{D})}[\mathbb{H}(y|x,\mathcal{D})] \tag{3.5}$$

$$= \mathbb{H}(y|x,\mathcal{D}) + \frac{1}{T}\sum_c \sum_{t=1}^{T} p(y=c|x,w_t) \log p(y=c|x,w_t) \tag{3.6}$$

Mutual information can be interpreted as a measure of model uncertainty, or epistemic uncertainty since mutual information is minimized when the knowledge about the model parameter does not increase the information in the final prediction. On the other hand, predictive entropy can be interpreted as the sum of epistemic and aleatoric uncertainty.

3.3 Metrics

In the following, we outline methods of evaluating the performance of the network and the quality of uncertainty estimates. Namely, we use global accuracy, average accuracy, and mean intersect-over-union (mIOU) as metrics for network performance. For uncertainty qualities, we use expected calibration errors (ECE), maximum calibration errors (MCE), and the area under the curve for precision-recall curves.

3.3.1 Network Performance

We evaluate the network's performance mainly by mIOU, but we also monitor global and average accuracy. We define these metrics below.

Accuracy (global)

Given predicted pixel $\hat{y}_i$ and corresponding ground truth label y_i, we define global accuracy as

$$\text{Accuracy(g)} = \frac{1}{N}\sum_{i}^{N}\mathbb{I}[\hat{y}_i = y_i] \tag{3.7}$$

where N defines the total number of pixels. This is usually denoted as *Acc(g)* in our tables.

Accuracy (average)

A downside of using global accuracy is that it doesn't take the frequency of each class into account, which means that the metric is biased towards classes with larger areas. We further define per-class or average accuracy, which is not influenced by the frequency of each class:

$$\text{Accuracy(c)} = \frac{1}{C}\sum_{c}\frac{1}{\sum_{i}^{N}\mathbb{I}[y_i = c]}\sum_{i}^{N}\mathbb{I}[\hat{y}_i = y_i] \tag{3.8}$$

where C is the total number of classes. This is usually denoted as *Acc(c)* in our tables.

Intersect-over-union (IOU)

Let C be the total number of classes, we further define per-class true positives N_c^{TP}, false positives N_c^{FP}, and false negatives N_c^{FN} as:

$$\begin{aligned}
N_c^{TP} &= \sum_{i,c}\mathbb{I}[\hat{y}_i = y_i | y_i = c] \\
N_c^{FP} &= \sum_{i,c}\mathbb{I}[\hat{y}_i \neq y_i | \hat{y}_i = c] \\
N_c^{FN} &= \sum_{i,c}\mathbb{I}[\hat{y}_i \neq y_i | y_i = c]
\end{aligned}$$

We define per-class IOU as the ratio between the intersect $(y \wedge \hat{y})$ and the union $(y \vee \hat{y})$, which in practice would be computed as:

$$\text{IOU}_c = \frac{N_c^{TP}}{N_c^{TP} + N_c^{FP} + N_c^{FN}} \tag{3.9}$$

and mean IOU as:

$$\mathrm{mIOU} = \frac{1}{C}\sum_{c}\frac{N_c^{TP}}{N_c^{TP}+N_c^{FP}+N_c^{FN}} \tag{3.10}$$

The mIOU metric is a suitable and commonly used metric in semantic segmentation, it measures the degree of "similarity" between the ground truth and prediction and puts the same weight on each class even if classes are imbalanced in the dataset.

Other metrics

We also sometimes monitor the speed and cost of the networks wherever we need to compare efficiency between networks. In particular, we measure the number of parameters of the network and the number of FLOPs to perform one inference. Furthermore, we track the GPU inference time and memory time averaged over 5 forward passes. These metrics help us better evaluate the efficiency of the network.

3.3.2 Uncertainty Calibration

Before we introduce the metrics for measuring the quality of uncertainty, we first need to discuss what it means for a network to produce “good” uncertainties. Recall that our learning problem has no ground truth uncertainty, rather, we require “calibrated uncertainties”. Consider a classification network with softmax outputs, where $\hat{Y}$ is the network prediction and $\hat{P}$ is the associated confidence with that prediction. We define *perfect calibration* as the confidence is equal to the true probability that $y = \hat{Y}$ given $\hat{P}$. In other words,

$$\mathbb{P}\left(\hat{Y} = Y|\hat{P} = p\right) = p \tag{3.11}$$

In practice, we discretize the space of probability into M bins each with width $1/M$. Let B_m be the set of indices of samples with confidence score $p \in \left(\frac{m-1}{M}, \frac{m}{M}\right]$. We define the accuracy of bin B_m as:

$$\mathrm{acc}(B_m) = \frac{1}{|B_m|}\sum_{i\in B_m}\mathbb{I}[\hat{y}_i = y_i] \tag{3.12}$$

Where $\mathbb{I}$ is the indicator function. We can further define the confidence of each bin as the average confidence of samples within B_m

$$\mathrm{conf}(B_m) = \frac{1}{|B_m|}\sum_{i\in B_m}\hat{p}_i \tag{3.13}$$

Calibration Error

Two natural metrics arise from the definition of calibration. The first is Expected Calibration Error (ECE) [54]:

$$\mathbb{E}\left[\left|\mathbb{P}\left(\hat{Y}=Y|\hat{P}=P\right)\right|\right]$$

It's clear that an ECE value of 0 indicates perfect calibration. Another metric commonly used is the Maximum Calibration Error (MCE) [54], defined as the maximum difference between the confidence and the true accuracy:

$$\max\left|\mathbb{P}\left(\hat{Y}=Y|\hat{P}=P\right)-p\right|$$

In practice, we can compute the ECE and MCE using discrete bins described in Equation 3.12 and Equation 3.13.

$$\mathrm{ECE}=\sum_{m=1}^{M}\frac{|B_m|}{n}|\mathrm{acc}(B_m)-\mathrm{conf}(B_m)| \tag{3.14}$$

$$\mathrm{MCE}=\max_{m}|\mathrm{acc}(B_m)-\mathrm{conf}(B_m)| \tag{3.15}$$

PAvPU

In the context of computer vision tasks such as segmentation, global statistics such as ECE and MCE are not able to capture the local quality of the segmentation. Mukhoti and Gal [52] introduced a way of evaluating uncertainty quality for computer vision and has been adopted by several other studies [50, 49]. The central idea of the metric is based on the statement that a well-calibrated model should be accurate (a) when it's confident (c), and unconfident (u) when it's inaccurate (i).

Let the normalized confidence value $I\in[0,1]$ be the threshold of confident/unconfident. We define n_{ac} as the number of pixels that are predicted correctly and confidently, n_{ic} be the number of pixels that are predicted inaccurately but confidently, n_{au} be the number of pixels that are predicted accurately but not confidently, and n_{iu} be the number of pixels that are predicted inaccurately and unconfidently. Mukhoti and Gal [52] further proposes that accuracy and certainty be performed over patches of pixels by using sliding windows. We define the accurate-certain ratio as:

$$p(\mathrm{accurate}|\mathrm{certain})=\frac{n_{ac}}{n_{ac}+n_{ic}} \tag{3.16}$$

Similarly, we define the inaccurate-uncertain ratio as:

$$p(\text{uncertain}|\text{inaccurate}) = \frac{n_{iu}}{n_{iu} + n_{ic}} \tag{3.17}$$

Lastly, we define the uncertainty accuracy (UA), to the patch accuracy vs. patch uncertainty (PAvPU) as:

$$\text{PAvPU} = \frac{n_{ac} + n_{iu}}{n_{ac} + n_{iu} + n_{au} + n_{ic}} \tag{3.18}$$

Clearly, networks with larger values of the above metrics are better. These metrics can be evaluated with a particular uncertainty threshold I [52], but can also be evaluated by sliding the uncertainty threshold from $[0,1]$ and computing the area under the curve [50, 49]. In this thesis, we compute the area under the curve as it's a more robust way that doesn't require manual tuning of the uncertainty threshold. The normalized uncertainty value is computed using $I_{\text{norm}} = \frac{I}{I_{\max} - I_{\min}}$, with the minimum and maximum values computed over a validation set. [1]

[1]In Section 4.3.4 we will discuss the validity of normalizing uncertainty this way, and why it may present issues in stochastic cases.

Experiments

4.1 Training Details

4.1.1 Dataset

The Cambridge-driving labelled Video Database (CamVid) is a roadscene dataset captured from the perspective of a driving automobile [9]. All images are semantically labelled with 11 classes such as cars, pedestrians, bicyclists, and poles. Images are captured at a resolution of 960×720, with 367, 101, and 233 images in training, validation, and testing sets respectively. Following [1], we downsize the images to 480×360 and drop ambiguous pixels labelled as "void" for training and evaluation.

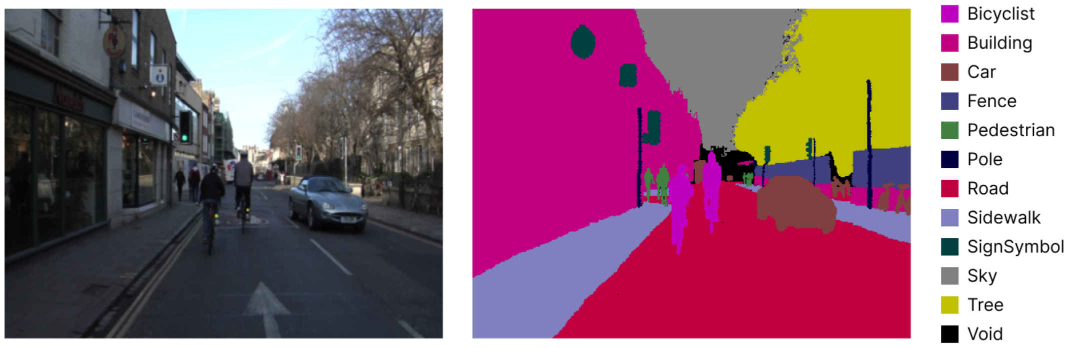


Figure 4.1 Example labelled image from the CamVid Dataset (test set)

4.1.2 Image Augementation

We follow a similar image augmentation procedure as [29, 44], with random horizontal flipping, followed by random color jittering, and random cropping from scale $[0.7, 1.3]$. Finally, the images are resized to 360×480 and normalized. Figure 4.2 demonstrates two examples of images going through the augmentation pipeline. The purpose of these image augmentation steps is to help reduce the overfitting of the network on the training set. We also note that all "void" pixels are ignored in both loss functions and metrics, therefore we use the same pixel value for filling in the blank space when images are scaled down (e.g. second row in the figure).

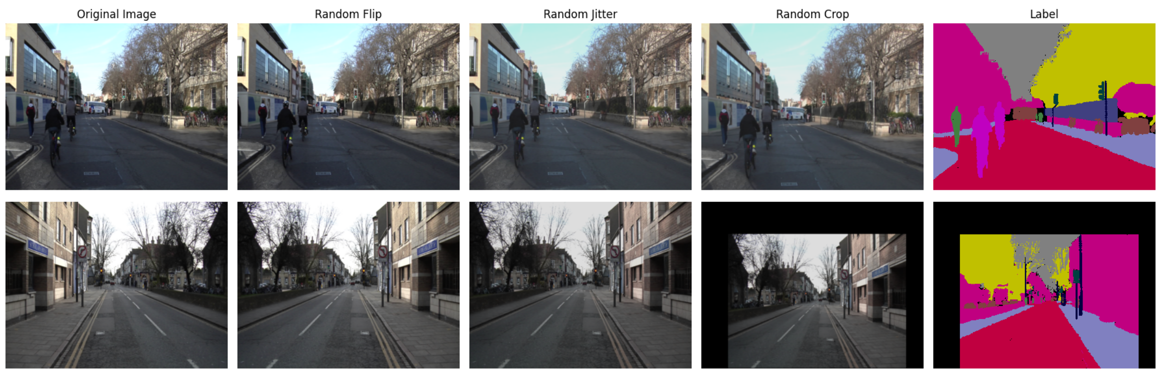


Figure 4.2 Example of two images going through the augmentation process step by step, the images are randomly flipped, jittered, and cropped. Note that the label and the image go through the same augmentation but with random parameters each time. Image augmentation is computed in parallel to the main training loop

4.1.3 Hyperparameters

All networks are trained and tested on Google Colab with an Nvidia T4 GPU with 15GB of RAM, and the networks are built and trained using PyTorch [1]. Unless explicitly stated, all experiments in this thesis utilize the same hyperparameters in the training process. This is so that we can make fair comparisons across different runs. We utilize a batch size of 10 images, which was empirically determined to have relatively stable gradients but not using too much memory during training. We train all experiments for 200 epochs, or equivalent to around 7400 steps. Similar to [11], we use RMSProp with an initial learning rate of $1e-3$ with a polynomial learning rate scheduler that updates the learning rate using $\left(1 - \frac{iter}{max\_iter}\right)^{power}$ with power 0.9. We use a weight decay of $1e-4$ to help with regularization, and unless stated otherwise, we use dropout layers after each stage, with dropout probability linearly

[1]Link to GitHub Repo containing all training and evaluation code

decaying from 0.1 to 0 from the deepest to the shallowest layers on both encoders and decoders. Finally, we use cross-entropy loss with no class balancing.

It's important to note that all networks trained with 200 epochs on a polynomial learning rate schedule are *underfitted*. An example of training history is shown in Figure 4.3, where it's clear that the network can still improve further. We make this choice simply due to limited time and computational resources, and we argue that the performance of the network at the same cut-off epoch is indicative of how well it will perform with more epochs. This also implies that the performance of the network at the end of training is typically lower than what it should be if more epochs are trained. We supply further experiments that use exponential learning rate and patience 100 in Section 4.2.2. Additionally, due to different network sizes, larger networks will be more underfitted than smaller networks.

Finally, we also point out that many papers utilize three additional "tricks" to improve results: a) pre-training on cityscape and/or ImageNet, b) fine-tuning with full-resolution images, and c) multi-scale evaluation. In this thesis, we do not perform any of these techniques to further improve results, all of our experiments are trained and tested on half-resolution images only from the CamVid dataset.

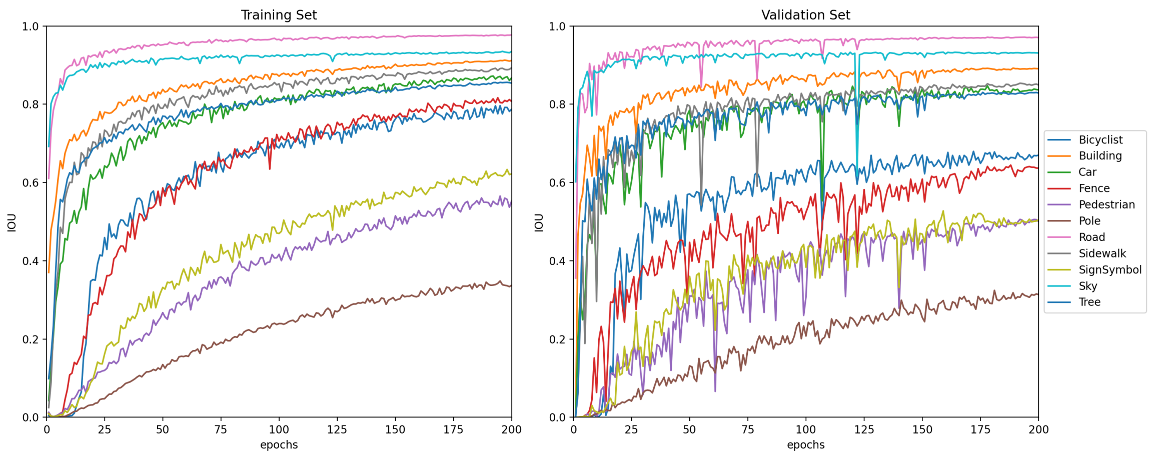


Figure 4.3 The performance history of the baseline network on the training and validation set. There tends to be a larger gap between the training and validation set on classes with high variabilities in appearances, such as cyclists, sign symbols, and fences. Note that performance on both training and validation sets has not plateaued for most classes at the end of 200 epochs.

4.2 Efficient Designs

In this section, we focus on *efficient designs* of the network, building upon the inverted residual decoder block that we tested in Section 3.1.3. Our experiments are based on several hypotheses and observations from the literature. We test a few modifications to the network architecture in Section 4.2.1 such as variations to skip connections, adding squeeze and excitation blocks, and removing decoder depths. We then test different scales of the network by changing the depths and widths in Section 4.2.2. The results from these tests then aid us in finding the most efficient way to scale the network, which we explore in Section 4.2.3 by presenting three different scales of the baseline network that achieves competitive performance against the state-of-the-art.

4.2.1 Modifications

The modifications we make to the baseline network presented in Section 3.1.3 are as follows.

No skip connections This is a simple ablation test to see how effective skip connections are. As it is commonly known, skip connections help transfer high-resolution features and usually help with thinner classes. By removing them, we expect the network to under-perform in classes with fine boundaries or small areas, such as fences and posts. In the tables below we refer to this experiment as *no-skip*, and an illustration of the experiment setup is shown in Figure 4.4a.

Dense connections With similar motivation as no skip connections, we want to see how much can extra skip connections help with performance, and how much slower inference becomes with the presence of more skip connections. Note that this experiment only differs a little bit from the baseline experiment, with additional skip connections on the first and fifth stages. We refer to this experiment as *dense-skip*, and it's illustrated in Figure 4.4b.

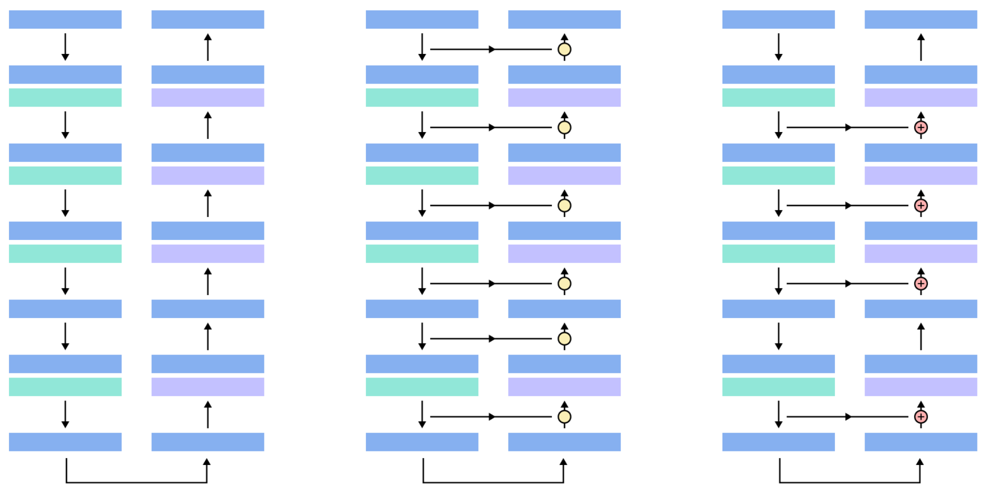
Residual connections In this variation, we add the features from the encoder to the decoder rather than concatenating it. This is motivated by the residual block [27], where skip connections in the form of addition can help with training networks across large depths. Since our network is symmetrical, it means that the output feature of each encoder block shares the exact same shape as a corresponding output feature of a decoder block, making feature addition possible. We refer to this experiment as *add-skip*, and it's illustrated in Figure 4.4c.

Squeeze and Excitation We adopt the squeeze and excitation block [32] in similar fashion to the MBConv block used in EfficientNet [68]. The squeeze and excitation block is inserted after the inverted residual expansion block. The idea of squeeze and excitation block is that features can be used more effectively through attention mechanisms, and that we don't require a significantly higher parameter count to improve performance.

Shallow decoder In this variation, we use depth = 1 in the decoder of the network. The idea of this variation is motivated by high performing segmentation architectures that use relatively light-weight decoders [11, 44], and the intuition is that the encoder architecture that reduces feature resolution whilst increases feature depth is an effective enough structure on its own. In other words, this experiment hypothesizes that decoder are closer to an implementation detail that helps to transform the encoder features into desired shapes. We abbreviate this experiment as *shallow-dec* in tables.

Table 4.1 Qualitative performance of network modifications

Method	Params $\downarrow$	GFLOPs $\downarrow$	Time(ms) $\downarrow$	Acc(g) $\uparrow$	Acc(c) $\uparrow$	mIOU $\uparrow$
baseline	3.67M	2.24	11.47	92.09	75.22	66.61
shallow-decoder	**2.59M**	**1.68**	**8.99**	92.01	74.65	66.47
no-skip	3.63M	2.21	11.38	90.28	66.96	58.51
add-skip	3.63M	2.21	11.44	91.97	74.18	65.95
dense-skip	3.77M	2.27	11.48	92.03	**75.97**	67.17
squeeze-excite	4.56M	2.27	13.97	**92.39**	75.46	**67.70**



(a) no-skip (b) dense-skip (c) add-skip

Figure 4.4 Modifications to skip connections between the encoder and corresponding decoder blocks. (a) shows the baseline network with all skip connections turned off, (b) with all skip connections turned on, and (c) uses addition rather than concatenation to transfer features

Table 4.1 shows the quantitative performance of the modifications, and table 4.2 shows the per-class breakdown of the IOU scores for each experiment. As expected, we see that *no-skip* produces the worst mIOU due to significantly lower IOU on small classes such as fence, post, bicyclist, for example, it only achieves 0.8 IOU on poles. On the other hand, we see that *dense-skip* increases the IOU scores for small classes significantly, resulting in the highest mIOU score out of all experiments, and highest scores on the five smallest classes: bicyclist, fence, pedestrian, pole, and sign symbols. Perhaps surprisingly, the increased number of skip connections (one at low depth and one at high depth, illustrated in fig 4.4) doesn't come at much a high cost in computation, with only 0.04M more parameters and 0.01ms longer inference time compared to the baseline. On the other hand, when using residual connections in *add-skip*, we see that the performance drops significantly without a noticeable change in parameter count or inference time.

From our experiments that modified the skip connections, namely *no-skip*, *dense-skip*, and *add-skip*, we show that the high-resolution features maps transferred from the encoder improve the detection of smaller classes significantly. Since transferring features via addition worsens the performance, we can conjecture that normal skip connections improve network performance by creating a wider feature map at low depth via concatenation. Note that widening the feature map via skip connection is a much cheaper operation than naively scaling the output feature width throughout the network, which would quadratically increase the number of parameters. We also show that additional connections do not significantly hinder computation in our network architecture, as the increase in inference time is marginal.

We further observe that using decoder blocks with depth 1 in *shallow-dec* is an extremely efficient way of scaling down the network, whilst *squeeze-excite* is a relatively inefficient way of scaling up the network. We see that *shallow-dec* reduces parameter count by around 30% and inference time by around 25%, but only at the cost of 0.14 mIOU. A plausible reason for this is that the inverted residual decoder block has some redundancies within it, such that it doesn't scale very well with depth. Another reason by be due to the fact that the network relies a lot on the features transferred from the encoders via skip connections, such that additional feature extraction within the decoder is not as significant. Finally, we see that although *squeeze-excite* improves mIOU, it also requires too many parameters and inference time.

Table 4.2 per-class metric breakdowns for experiments with modifications to skip connections

Method	Bicyclist	Building	Car	Fence	Pedestrian	Pole	Road	Sidewalk	SignSymbol	Sky	Tree	Acc(g)	Acc(c)	mIOU
baseline	53.7	84.8	77.6	56.8	41.8	**28.4**	95.5	81.7	40.8	93.2	78.3	92.1	75.2	66.6
shallow	55.2	84.4	78.4	55.4	41.7	26.8	95.6	**81.9**	40.9	93.1	77.7	92.0	74.7	66.5
no-skip	47.6	82.7	75.0	50.6	27.2	1.8	94.1	77.2	21.9	90.7	74.9	90.3	67.0	58.5
add-skip	54.2	84.6	78.0	55.9	41.3	24.8	95.5	81.4	38.7	93.2	77.9	92.0	74.2	66.0
dense-skip	56.2	84.9	77.6	57.0	**45.8**	28.1	95.3	80.9	41.9	93.2	77.8	92.0	**76.0**	67.2
s-e	**57.8**	**85.5**	**79.1**	**58.3**	44.4	27.9	**95.6**	81.9	**42.4**	**93.3**	**78.6**	**92.4**	75.5	**67.7**

We present qualitative results of the modifications to skip connections in Figure 4.5 on the CamVid test set. In particular, we highlight that *no-skip* produces a very smooth and coarse-grained segmentation map, reminiscent of the coarse outputs from DeepLab before CRF post-processing [11], or the upsampled outputs from low resolution features in FCN [44]. We also see that the presence of skip connections significantly improve the prediction smaller classes, notably posts in columns $1, 4, 5$.

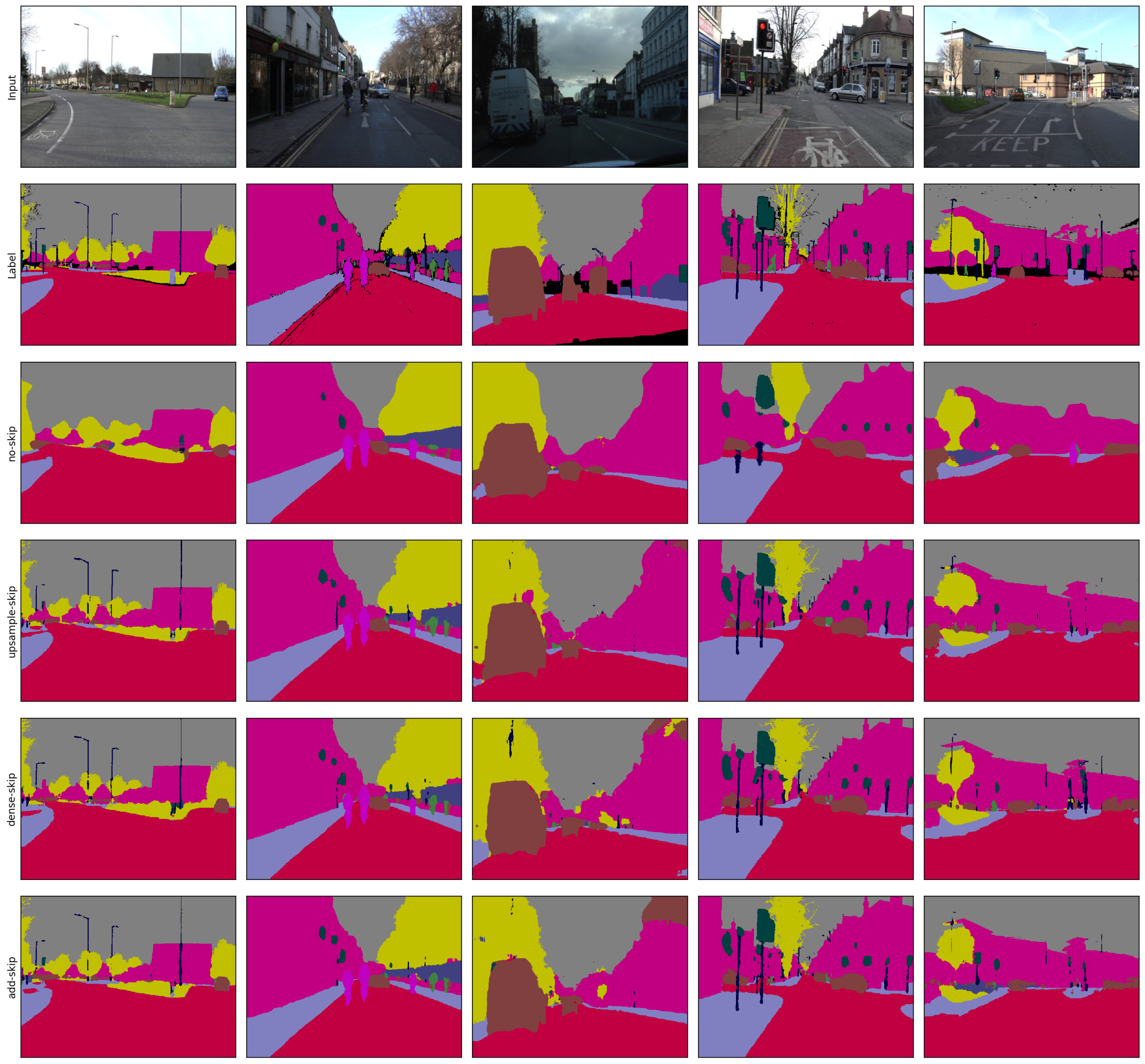


Figure 4.5 Qualitative results of modifications to skip connections. We highlight that *no-skip* produces very coarse results with smooth boundaries, as it's unable to recover the spatial resolution without the skip connections. On the opposite spectrum, *dense-skip* is able to better predict small and thin classes such as posts and signs.

4.2.2 Network Scaling

Motivated by model scaling techniques in [28, 31, 61, 27], we attempt to scale the network by changing the depth and width of the network with the same input resolution. Depth refers to the number of blocks in each stage, and width refers to the number of output channels in each stage. We do not change the input resolution for purposes of benchmarking on CamVid, where it's standard to use either full or half image resolution. Furthermore, we make modifications only from the baseline model, i.e. we do not adopt any of the modifications from the previous section for this set of experiment.

Table 4.3 Quantitative performance of network scaling

Width	Depth	Params ↓	GFLOPs ↓	Time(ms) ↓	Acc(g) ↑	Acc(c) ↑	mIOU ↑
0.5	0.5	0.67M	0.49	4.13	90.28	67.82	59.14
0.5	1.0	0.97M	0.73	6.98	90.70	69.51	60.89
1.0	0.5	2.53M	1.54	4.60	91.87	74.01	65.58
1.0	1.0	3.67M	2.24	11.47	92.09	75.22	66.61
1.0	1.5	6.98M	3.74	17.58	92.25	75.39	67.14
1.5	1.0	8.13M	4.90	14.80	92.60	78.29	69.59

Table 4.3 shows the results of varying the width scale in $[0.5, 1.0, 1.5]$ and the depth scale in $[0.5, 1.0, 1.5]$. We see that the parameter and FLOP count both scale roughly with the width and depth scale. However, we note that increasing width tends to increase parameter and floop count more than depth. This is due to the fact that depth count are rounded to the nearest integer in our implementation (e.g. the depth cannot be less than 1 for each stage, and depth value 1.4 will be rounded down to 1). We observe that our finding coincides with that or [68], where uniform scaling across more than one dimension leads to better performance. For instance, width and depth scale of 1 outperforms cases where only one dimension is scaled.

However, it's also important for us to critically evaluate the *trade-off* of increasing parameters and inference speed. Indeed, we see that increases in network width improves mIOU more, even when taking into consideration of the parameter count. By comparing the experiments with width 1.5, depth 1.0 and width 1.0, depth 1.5, we see that the wider network adds only an additional 1.2M parameters but has lower inference time and higher mIOU by 2.4. We can also intuitively conjecture that inference time should scale roughly linearly to depth and less than linearly for width due to parallel computing by GPUs.

Wider networks may be more preferable in segmentation architectures in general, since it requires more fine-grained features compared to classification networks. This is an important distinction as previous research has often placed emphasis on scaling the depth of classification networks in the context of classification tasks [27, 28]. This phenomenon, coupled with the fact that inference time scales better with width than depth, means scaling network width is preferred over scaling network depth when upscaling networks. On the other hand, when downscaling networks, we can scale both width and depth proportionally, since reducing depth decreases parameter significantly without hurting performance too much.

4.2.3 Comparison to State-of-the-arts

Table 4.4 Quantitative performance of Lite, Medium, and Large Networks, (+) indicates experiments that have been trained for more epochs

Name	Params	GFLOPs	Time(ms)	Acc(g)	Acc(c)	mIOU
Lite	0.56	0.44	3.88	90.83	69.38	61.13
Lite(+)	0.56	0.44	3.35	91.97	74.59	66.40
Medium	2.70	1.70	6.72	91.86	75.39	66.66
Medium(+)	2.70	1.70	6.72	93.16	79.78	71.63
Large	10.52	6.11	10.27	93.00	78.99	70.84
Large(+)	10.52	6.11	10.27	93.49	81.53	73.61

Utilizing the results and findings from the previous two sections, we present three versions of the baseline network, lite, medium, and large. All networks use a shallow decoder with depth 1 and dense skip connections. The smallest network, which we call *lite*, uses a width and depth scale 0.5, the *medium* network uses a width and depth scale 1.0, and the large network (*large*) uses a width and depth scale of 2.0 and 1.0 respectively. For fair comparison to state-of-the-art, we also train the network with the same recipe as one adopted by FCDenseNet [37] to see how the network performs with more epochs, we use an exponential learning rate scheduler with decay 0.995, and a batch size of 3. We monitor the mIOU and average accuracy on the validation set with a patience of 100.

Table 4.4 shows the result of these three networks trained for 200 epochs, and (+) indicates networks that are trained for more than 200 epochs following the same recipe as [37]. The results validate our discussion in Section 4.1, where networks trained with 200 epochs do underfit the data. The *lite* network uses only half a million parameters and an inference speed of 3.88 ms, equivalent to around 250 fps. On the other hand, the large network only uses around 10M parameters and still runs comfortably in real-time with an inference speed of 10.3 ms which is around 100 fps.

In Table 4.5 we list state-of-the-art methods that use the same input as us, with halved resolution. We see that the smallest version of our network, *Lite*, achieves higher mIOU than all other methodologies that are trained and tested with half-resolution. Aside from ENet, our

Table 4.5 Comparison to state-of-the-arts that are trained and tested on half resolution

Method	Resolution	Pretrain	Encoder	Params (M)	mIOU
FCN8 [44]	1/2	ImageNet	VGG	134.5	57.0
DeconvNet [56]	1/2	ImageNet	VGG	252	48.9
SegNet [1]	1/2	-	-	29.5	46.4
ENet [58]	1/2	-	-	0.37	51.3
Lite [ours]	1/2	-	MobileNetV2	0.56	66.4
Medium [ours]	1/2	-	MobileNetV2	2.70	71.6
Large [ours]	1/2	-	MobileNetV2	10.52	73.9

Table 4.6 Comparison to state-of-the-arts that are trained and tested on using full resolution. 1/2 + F: trained on half resolution followed by fine-tuning on full resolution

Method	Resolution	Pretrain	Encoder	Params (M)	mIOU
FCDenseNet56 [37]	1/2 + F	-	DenseNet	1.5	58.9
FCDenseNet67 [37]	1/2 + F	-	DenseNet	3.5	65.8
FCDenseNet103 [37]	1/2 + F	-	DenseNet	9.4	66.9
BiSeNet [78]	F	ImageNet	Xception	5.8	65.6
BiSeNet [78]	F	ImageNet	ResNet18	49.0	68.7
DDRNet23-Slim [29]	F	ImageNet	-	5.7	74.7
DDRNet23 [29]	F	ImageNet	-	20.1	76.3
RTFormer-Slim [73]	F	ImageNet	ViT	4.8	81.5
RTFormer [73]	F	ImageNet	ViT	16.8	82.5
Lite [ours]	1/2	-	MobileNetV2	0.56	66.4
Medium [ours]	1/2	-	MobileNetV2	2.70	71.6
Large [ours]	1/2	-	MobileNetV2	10.52	73.9

Lite network also uses only a fraction of the parameters used by other approaches. Note that our network uses 0.2M more parameters compared to ENet, it is justified by the significantly higher mIOU score.

In Table 4.6 we list methods that are trained and evaluated on full resolutions, most of them are pre-trained on ImageNet, and some also use multi-scale evaluation [73]. The comparison is also plotted on Figure 4.6 for visualization. We see that our network outperforms FCDenseNet and BiSeNet, where our *Medium* network outperforms both methods on mIOU with fewer parameters. On the other hand, our method still underperforms DDRNet and RTFormer. It remains future work for us to train our network with full-resolution images, with pre-training on ImageNet in order to compare with these methods fairly. Furthermore, we are unable to make fair comparison of runtime due to different hardware.

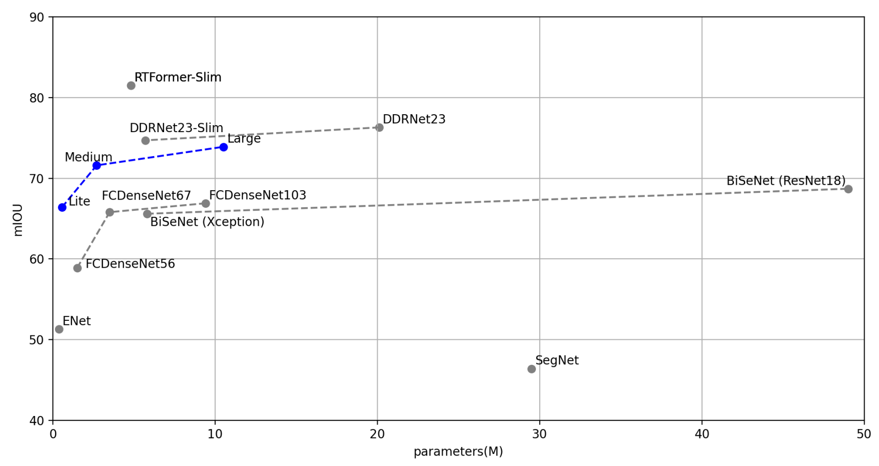


Figure 4.6 Comparison of our approach to state-of-the-art approaches, methods connected with a dashed line belong to the same paper.

4.2.4 Discussions

In the following summarise the findings from our experiments in Section 4.2.1 and 4.2.2.

Skip connections are a low-cost way to improve the detection of small classes. Our experiments with modifications to skip connections showed that concatenating features from the encoder to the decoder significantly helps the detection of small obstacles. In addition, we highlight that this operation is relatively low-cost, adding only a small fraction of parameter and inference time. Note that this may not hold in other encoder-decoder networks that do not use inverted residual blocks. The expansion steps in inverted residual blocks are the dominant computational cost of our network and hence transferring high-resolution features comes at a relatively low cost.

Scaling coefficients can be improved with a small grid search. Our network scaling experiments in Section 4.2.2 use width and multipliers in $[0.5, 1.0, 1.5]$ for convenience of quickly exploring the parameter space. A much better but perhaps more costly method would be to adopt a small grid search similar to [68], whereby we find the best scaling coefficient for the network depth and width based on a few constraints. For example, we can perform a random grid search of width and depth multipliers to determine the most efficient way to improve performance. The scaling coefficients obtained this way are more likely going to better optimize the scaling efficiency of the network.

Wider features are desired in segmentation networks. This finding is validated by our experiments involving shallow decoders and network scaling. We conjecture that network width is extremely crucial in encoder-decoder networks, allowing networks to improve their detection for small classes. We provide conjecture on why this is the case, although more rigorous experiments are required to validate this hypothesis:

The nature of network width for encoder-decoder architectures is different to classification networks. Consider a standard classification network such as MobileNet, it outputs last features with shape $(7 \times 7 \times 1024)$ to predict 1000 classes on ImageNet. This suggests that the information contained in the last feature map sits at a much lower (about 50-fold) dimensional manifold than the feature dimension itself. Now consider building an encoder-decoder architecture with MobileNet, the final feature map of the network from the decoder will have 32 channels with half the original resolution. Compared with the final prediction at full resolution ($4\times$ more pixels) with 11 channels, we actually have less information contained in the feature map than is required from the final prediction.

Therefore, the loss in spatial information from lossy downsampling and upsampling steps can be reduced by increasing the network width, which increases the sparsity at which information exists in the final feature output. Modern CNNs are typically designed and fine-tuned to the ImageNet dataset with 1000 classes, so its intermediate feature outputs are not designed for semantic segmentation. This points to a future direction of neural architecture search (NAS) for finding scalable and efficient encoder-decoder architectures, which we will further discuss in the conclusion.

4.3 Bayesian inference

This section explores three variants of Bayesian neural networks, MC dropout, stochastic depth, and combining both. Using the standard dropout-based approximation as a comparison, we show that stochastic descent is an interesting and novel way of performing Bayesian inference that exhibits several desired properties, such as improved network calibration and performance. We evaluate the results by first examining the effect of the Bayesian methods on the network performance, followed by the effect on the uncertainty calibration.

All of our experiments use the *baseline* network from our previous experiments, with additional Dropout layers and stochastic depth layers. Dropout layers are added after the activation function, and stochastic depth is applied to blocks with residual connections (i.e. input and output are the same shape, such that the block can be bypassed using identity function). We use dropout probabilities that decrease linearly from the deepest to the most shallow parts of both the encoder and the decoder. This follows the intuition from [38], where deeper layers are responsible for generating more abstract features that would influence the variance of the network further. We use a similar strategy with linearly decaying stochastic depth probabilities, which follow directly from [34] where it was shown that linearly decaying rates are preferred over constant rates. Finally, we also test a variant where we combine both dropout and stochastic depth.

The networks are trained with the stochastic regularization method turned on, and at test time we consider two different types of inference: deterministic and Bayesian (denoted as [b]). In the deterministic case, we simply turn off the dropout layers or the stochastic depth layers and allow the full network to be used for inference [2]. On the other hand when applying Bayesian inference, we keep the regularization layers on, and utilize Equation 3.3 to compute the output probabilities for inference (See Section 3.2 for details). Unless specified, we use $T = 10$ in the testing phase of the network with 10 forward passes to perform Bayesian approximation.

We test dropout probabilities and stochastic depth probabilities both in the range of $[0.1, 0.3, 0.5]$, where the value indicates the dropout probability or the bypass probability of the *deepest* block. We will denote the dropout probability as the probability that a given unit will be dropped, and the stochastic depth probability as the probability that a block will be bypassed. By increasing the probability in both sets of experiments, the networks become more underfitted as both dropout and stochastic depth are regularization methods. We also point out that for our network, stochastic depth is a slightly weaker regularizer than dropout layers, since all layers can have valid dropout layers but not all layers can be bypassed (i.e. upsample and downsample layers cannot be passed since they change the feature shape). We denote the experiments as *dropout*, *sd*, and *combined*, and indicate the probability assigned to the experiment, e.g. *dropout-0.3*. We train the network with the same protocol as before.

4.3.1 Network Performance

Table 4.7 shows the accuracy and IOU performance of our tested Bayesian networks, where [b] indicates scores obtained using Bayesian approximation, and without [b] indicates that the network is tested deterministically. Perhaps not so surprisingly, networks with higher regularization achieve lower performances. In particular, we highlight that combining both dropout and stochastic depth with high probabilities makes the network significantly underfit the dataset, scoring around 6.7 mIOU lower compared to the deterministic baseline model. We show that networks trained with stochastic depth obtained the best performance, with 0.1 probability achieving the highest global accuracy of 92.2 and 0.3 probability achieving the

[2]In practice, the layer outputs need to be scaled by a constant at test time in order to recover the full network properly

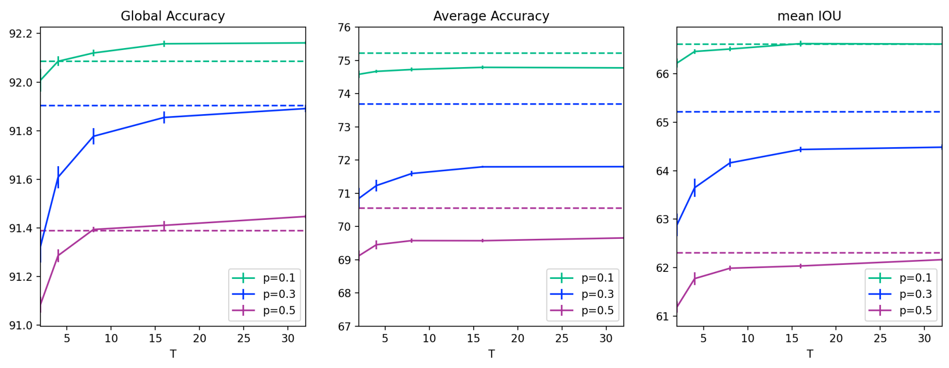
Table 4.7 Accuracy and IOU with Bayesian Approximation, [b] indicates values ran with Bayesian approximation with $T = 10$, without [b] indicates deterministic inference

Dropout	Sd	Acc(g)	Acc(g) [b]	Acc(c)	Acc(c) [b]	mIOU	mIOU [b]
-	-	92.08	-	75.42	-	66.90	-
0.1	-	92.09	92.14	75.22	74.81	66.61	66.59
0.3	-	91.90	91.87	73.68	71.74	65.21	64.34
0.5	-	91.39	91.40	70.55	69.63	62.31	62.07
-	0.1	**92.20**	**92.22**	75.15	74.92	67.01	66.97
-	0.3	92.05	92.08	**75.91**	**75.48**	**67.06**	**66.97**
-	0.5	91.80	91.83	74.82	74.37	66.09	66.00
0.1	0.1	92.08	92.10	75.05	74.40	66.70	66.50
0.3	0.3	91.78	91.73	71.97	70.91	63.92	63.30
0.5	0.5	91.28	91.28	68.38	67.13	60.62	60.14

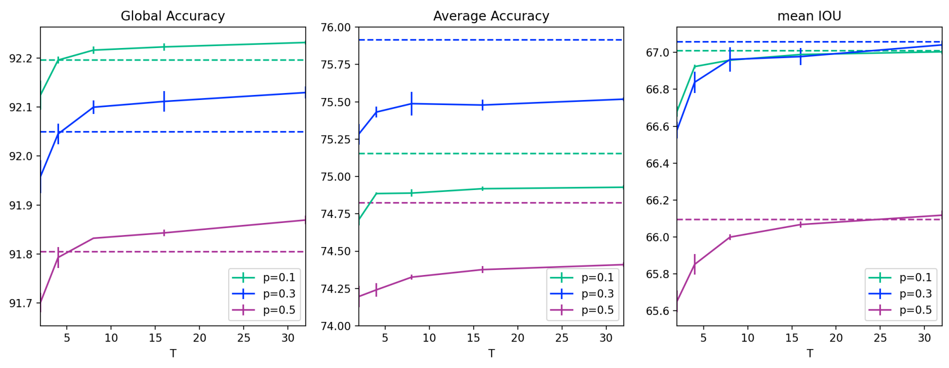
highest per-class accuracy of 75.91 and mIOU of 67.06. This is likely due to the fact that networks trained with stochastic depth are easier to optimize [34].

An interesting phenomenon that is consistent throughout most experiments is that global accuracy ran with Bayesian approximation outperforms the version of the network where stochastic regularization is turned off. We further illustrate this in Figure 4.7, where the dashed lines indicate the performance of the network without dropout, and the line shows the performance with varying T. We show that using Bayesian inference with $T > 0$ has the desired property of improving global accuracy, but the undesired property of worsening average accuracy and mIOU, which are usually the more important metrics. This finding is in agreement with [38], which showed the improvement in global accuracy with Bayesian approximation, but did not mention the decreased performance on other metrics.

Figure 4.8 shows qualitative examples of predictive entropy and mutual information of our Bayesian variants on a test set image. Qualitatively, we see that most methods are able to produce high uncertainties in regions where predictions are false. Stochastic depth-based methods tend to produce fewer variations in their outputs, which are qualitatively shown with dimmer uncertainty maps. In general, we find that qualitatively the uncertainty outputs look similar, but methods with higher regularization tend to produce higher overall uncertainties than methods with lower regularization.



(a) Performance of *dropout* variants with varying T



(b) Performance of *sd* variants with varying T

Figure 4.7 Performance of networks of Bayesian variants, dashed lines indicate performance of deterministic test-time inference, error bars show the variance of the values from 5 repeated trials. Note that global accuracy can usually be improved with Bayesian inference, but average accuracy and mIOU tend to be deteriorate.

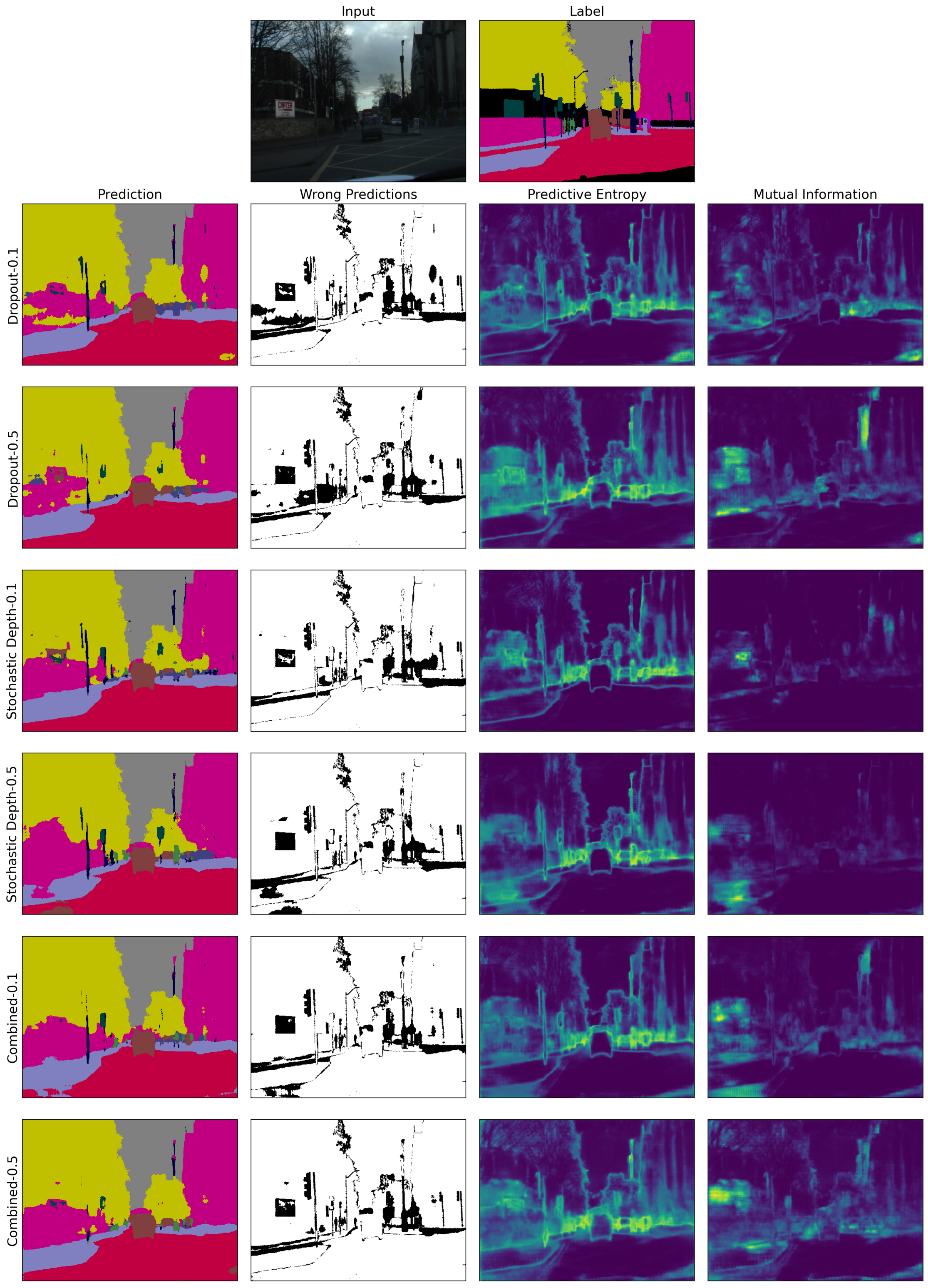


Figure 4.8 Qualitative results of Bayesian variants on a test set image, note that variants with higher probabilities (i.e. with probability 0.5) produce qualitatively more uncertainties.

4.3.2 Network Calibration

Table 4.8 MCE, ECE, and PAvPU with Bayesian Approximation, [b] indicates values ran with Bayesian approximation with $T = 10$, without [b] indicates deterministic inference

Dropout	Sd	mIOU ↑	ECE ↓	ECE [b]	MCE ↓	MCE [b]	PAvPU ↑
-	-	66.90	2.71	-	1.61	-	-
0.1	-	66.61	2.46	1.60	1.40	0.84	90.14
0.3	-	65.21	2.19	0.70	1.25	0.16	88.35
0.5	-	62.31	2.20	**0.17**	1.09	**0.05**	88.63
-	0.1	67.01	2.36	1.87	1.26	0.92	**90.33**
-	0.3	**67.06**	2.14	1.23	1.11	0.58	89.79
-	0.5	66.09	2.20	1.17	1.16	0.55	89.61
0.1	0.1	66.70	2.30	1.07	1.31	0.48	89.83
0.3	0.3	63.92	1.92	0.27	1.00	0.12	88.53
0.5	0.5	60.62	**1.72**	1.10	**0.76**	0.41	87.66

In Table 4.8 we report the three metrics for uncertainty quality: ECE, MCE, and area under PAvPU (See Section 3.3 for details). Note that we compute PAvPU with mutual information, which takes into account the model uncertainty. When compared to the network trained deterministically, we see that all Bayesian variants reduce calibration error even when tested deterministically. When Bayesian inference is used, *dropout* and *sd* reduce calibration error further. This shows that test-time Bayesian approximation, which approximates the predictive distribution by taking into account the posterior, is able to produce much more calibrated uncertainties.

The lowest calibration error obtained with Bayesian inference with achieved using *dropout-0.5*, with 0.17 ECE and 0.05 MCE. On the other hand, the lowest calibration error obtained using deterministic inference is achieved with *combined-0.5*. However, our results show deteriorated calibration errors in the *combined* experiments where both dropout and stochastic depth are added to the network with high probabilities. The best calibration errors achieved without hurting performance are obtained by *sd-0.3*, with 2.14 ECE and 1.11 MCE.

In Figure 4.9 we show the ECE and MCE of dropout, stochastic depth, and combined with probability 0.5, i.e. the highest rate for all three experiments. We see that increasing T does, in fact, decrease calibration error for both dropout and stochastic depth, but deteriorates calibration error for *combined*. We conjecture that this is likely due to the fact that the network underfits the data significantly, and that the network is underconfident in its predictions.

For the area under PAvPU, we see that the highest value is achieved with stochastic depth with 0.1 probability, and the value decreases with more regularization. On the other hand, by comparing results from Table 4.8 and Table 4.7, we observe that PAvPU tends to be somewhat directly proportional to the global accuracy of the network and therefore may not be an unbiased score that measures the quality of the uncertainties. For our experiments, the PAvPU metric is unable to distinguish between the quality of uncertainty across different runs. We will present a further critical discussion of this issue at the end of this section.

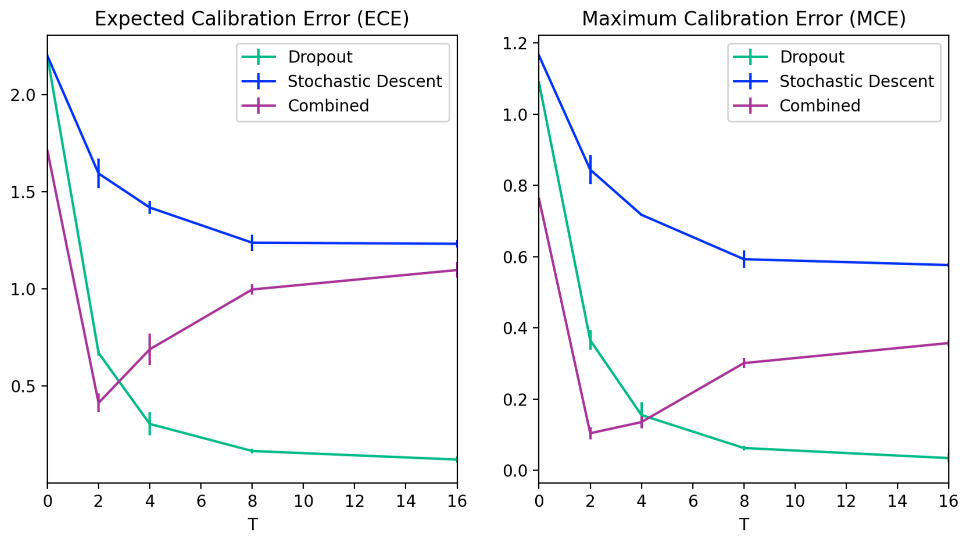


Figure 4.9 Effect of T on calibration errors for the different Bayesian methods with probability 0.5, error bars show variance obtained over 5 repeated trials over the test set.

In general, we see that using a moderate amount of stochastic regularization has the desired effect of making the network more calibrated even without using Bayesian inference. Using stochastic forward passes with as little as T values of 2 to 5, as shown in Figure 4.9, can decrease calibration error significantly further. In Figure 4.10, we show the difference in calibration error between the deterministic network trained with no dropout, and the network trained and tested with dropout 0.5 and $T = 10$. We see that the latter achieves almost perfect calibration whilst the former is overconfident. Furthermore, on the right side of the figure we plot the proportion of each probability bucket and show that the dropout variant predicts much fewer samples with high confidence. Complete calibration plots for all experiments can be found in the appendix.

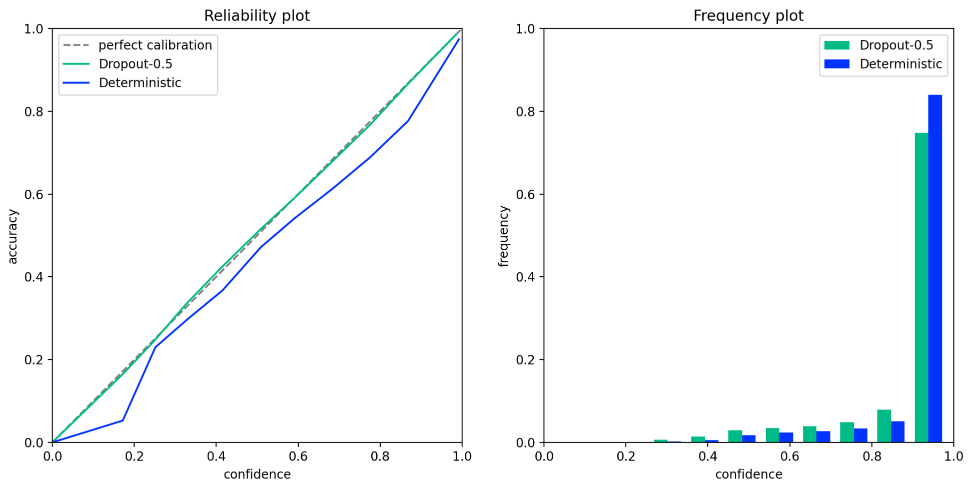


Figure 4.10 The reliability and frequency plot of stochastic vs. deterministic networks, networks with dropout show much more calibrated uncertainties whilst deterministic networks are overconfident.

4.3.3 Out-of-distribution Data

In this section we briefly experiment with testing the Bayesian networks on a dataset that comes from a different distribution as the training set, to qualitatively test its reliability on out-of-distribution data. We use a few samples of the RUGD dataset [76], which is a outdoor scene-understanding dataset for mobile robots. The majority of the images in the dataset are driving footage on forest trails, and represents a very different distribution of images to the CamVid dataset.

Figure 4.11 shows example images outputted by our Bayesian network. On terrains that are out-of-distribution such as images in first, second, and third row, we observe qualitatively extremely high uncertainties scores, particularly high epistemic uncertainties. We also see that the uncertainty is much lower on the last image on the fourth row, where the scene is much more similar to the CamVid dataset.

In Figure 4.12 we also plot the distribution of softmax probabilities outputted by the deterministic and Bayesian network respectively. We see that when comparing the softmax probabilities of the networks on the CamVid test set and the RUGD dataset, the average confidence score of on the CamVid set is much higher. In particular, we highlight that the Bayesian network reduces confidence score much more on the RUGD dataset compared to the deterministic version.

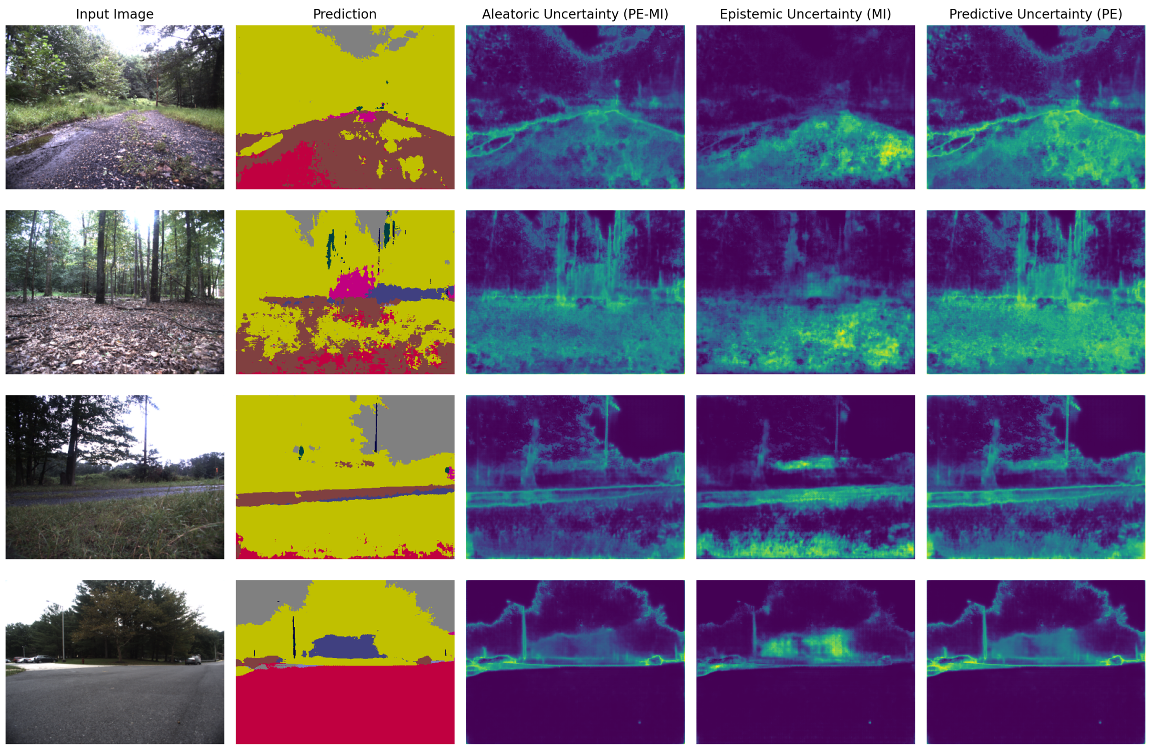

Input Image
Prediction
Aleatoric Uncertainty (PE-MI)
Epistemic Uncertainty (MI)
Predictive Uncertainty (PE)

Figure 4.11 The predictive outputs of a Bayesian network with dropout, we observe qualitatively very high epistemic uncertainty values on images that are different to our training set, e.g. the first three row, and much lower uncertainty on the last image, which is similar to the CamVid dataset.

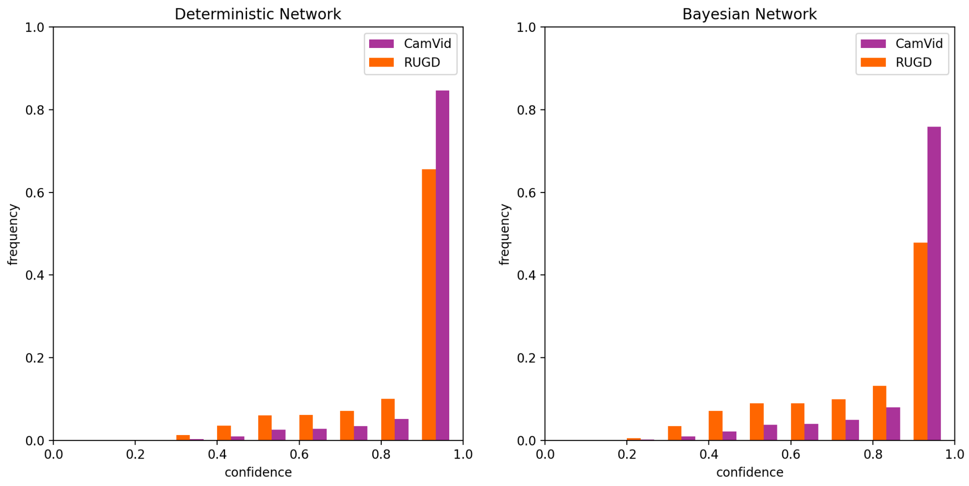


Figure 4.12 The distribution of softmax probabilities of the deterministic and Bayesian network on the CamVid dataset and the RUGD dataset. The Bayesian network produces much lower confidence on the out-of-distribution dataset compared to the deterministic network.

4.3.4 Discussions

Table 4.9 Computational requirements for different Bayesian inference, using the dropout and sd variant that obtained highest mIOU performance. Stochastic depth obtains overall lower ECE and higher mIOU compared to dropout.

Inference	Time(ms) ↓	FPS ↑	Memory ↓	mIOU ↑	ECE ↓	MCE ↓
			dropout-0.1			
Det.	11.4±0.0	88	189 Mb	66.61	2.46	1.40
T=2	21.9±0.0	46	358 Mb	65.94	1.97	1.11
T=4	36.1±4.9	28	711 Mb	66.36	1.72	0.94
T=8	61.5±11.6	16	1.39 Gb	66.50	1.66	0.87
T=16	107.4±1.9	9	2.77 Gb	66.58	1.55	0.80
T=32	215.5±5.4	5	5.54 Gb	66.65	1.52	0.80
			sd-0.3			
Det.	11.4±0.0	88	190 Mb	**67.06**	2.14	1.11
T=2	19.8±2.3	51	361 Mb	66.45	1.68	0.89
T=4	33.7±4.1	30	712 Mb	66.90	1.37	0.69
T=8	52.4±0.7	19	1.39 Gb	66.94	1.30	0.65
T=16	108.7±0.2	9	2.77 Gb	67.01	1.25	0.60
T=32	220.4±0.5	5	5.54 Gb	67.03	**1.22**	**0.58**

Trade-off between network performance and calibration error An obvious artifact of testing with Bayesian inference is that they produce lower mIOU and per-class accuracy. This presents a trade-off between improving the calibration error and achieving high performance. Our experiment show that using dropout probabilities as low as 0.1 combined with T as low as 2 to 5, is enough to reduce the calibration error by a large margin. Whilst using several stochastic forward passes for Bayesian approximation achieves significantly improved calibration errors, it also reduces the performance measured in mIOU compare to deterministic test-time inference. Furthermore, this gap is usually non-reducible even at very high T, as shown in Figure 4.7. In other words, Bayesian inference using test-time stochastic regularization trades off performance for improved calibration.

Trade-off between computation cost and calibration error Our results in Table 4.8 show that the best calibration error is obtained when Bayesian inference is used with T stochastic forward passes. However, a clear downside of this approach is that the computational costs grows with T. Table 4.9 show the computational trade-off between T, mIOU, and calibration errors. The first observation we make is that inference time does not grow linearly with T, but memory requirement roughly grows linearly with T. This is intuitive since GPU can compute inference in parallel to save compute time, but still has to produce the entire intermediate tensors. We see that with our GPU environment (Section 4.1), we lose real-time inference at around $T = 4$, where the framerate drops below 30.

Dropout versus stochastic depth Our results show that stochastic depth is a valid way of obtaining more calibrated uncertainties. We further show that Bayesian variants with stochastic depth with appropriate probabilities have the desired property of achieving higher mIOU whilst reducing ECE. We demonstrate the trade-off with *dropout-0.1* and *sd-0.3* in Table 4.9, which are the best network in terms of mIOU for the dropout and sd variants. We see that networks trained with stochastic depth obtain higher accuracy and lower calibration errors compared to dropout methods. This is true for both deterministic and non-deterministic test inferences. Therefore in practical cases, stochastic depth is a good alternative over dropout methods to achieve lower calibration errors without losing performance. On the other hand, if one is only interested in achieving the best calibration error, dropout methods with high probability (e.g. 0.5) with test-time Bayesian inference is the best choice.

PAvPU is a biased metric for measuring uncertainty Our results in Table 4.7 and Table 4.8 show that PAvPU is proportional to the global accuracy of the data. We see similar observations with papers that use this metric or its variations [52, 50, 49], where networks that perform better tend to achieve higher area under the curve. This phenomenon is due to the fact that the curve is not normalized to its performance, in the limit with maximum uncertainty threshold 1, the formula 3.18 equates to exactly the global accuracy.

We argue that PAvPU is not a good metric for measuring the quality of uncertainties due to the following reasons. 1) The quality of uncertainty should be mutually exclusive with respect to the quality of performance, in other words, the network should be allowed to predict poorly, and "knows" that it's performing poorly. 2) The PAvPU value is computed with based on a sliding window of uncertainty threshold that's derived from normalized entropy values across a validation set, which makes the metric dependent on the dataset (i.e. if it's in-domain, the entropy values will be lower, and vice versa if the validation is out-of-distribution from the training set, the entropy values will be higher), as well as noisy artifacts from the stochastic nature of Bayesian approximation. We argue that a good metric should not be dependent on the dataset itself and that such quality are defects that make the metric unsuitable for evaluating uncertainty.

Conclusion

This thesis tackled two important challenges in robotics computer vision tasks with deep learning: improving the efficiency of networks and improving the reliability of networks. We re-visited the standard encoder-decoder structure by building both the encoder and decoder using efficient convolutional blocks. The architecture was further improved using dense skip connections, shallow decoding stages, and efficient network scaling. We presented a final set of networks that achieves competitive performance with state-of-the-art approaches, with our *lite* version scoring 66.4 mIOU with only 0.56M parameters, and our *large* version scoring 73.9 mIOU wth 10.52M parameters. We show that Bayesian neural networks trained and tested with stochastic regularization exhibit much more improved calibration error than their deterministic counterparts, and further proposed a novel Bayesian approximation method based on regularization using stochastic depths. Our empirical results show that this novel approach obtains improved properties from the standard dropout-based method, obtaining improved calibration without reductions in preformance. Finally, we critically evaluated the commonly used uncertainty accuracy or PAvPU metric, and argued that it's not a suitable metric for evaluating the quality of uncertainties.

5.1 Evaluation of methodology

The following involves a critical evaluation of the methodology used in this thesis. We reflect on aspect of the methodology that could be improved in retrospect, and critically discuss the results we obtained.

Due to a high number of possible hyperparameters and design choices, we largely iterated through version of the networks by changing one modification at time from the baseline. This allowed us to understand both quantitatively and qualitatively what difference each modification makes, and combine multiple changes to develop a more efficient model. However, this method of iterating through the parameter space is somewhat ad-hoc and could have been improved by adopted techniques such as grid search and random search through the hyperparameter space, or using neural architectural search with a reinforcement learning agent to iterate over network variations. On the other hand, such approaches would be more computationally expensive, but we would more likely find a better sets of hyperparameters.

All of our networks were trained for 200 epochs by default, whilst this helped us to benchmark between different variations, it also meant that most networks were underfitted and their performance at the cut-off point may not be indicative of their final performance. For instance, smaller networks will be less underfitted than larger networks with the same number of steps. An alternative approach would have been to train all networks until the validation loss stops improving. This approach may have impact on our result for calibration errors in particular, since all of our Bayesian methods degrees of regularizing powers.

Although it's common to use half-resolution on the CamVid dataset, more recently most studies have trained and tested the networks with the full resolution images. Full resolution images are more likely to be used in practice as it contains richer information. However, a major consequence that this would have on our network is that the receptive field would much smaller relative to the input image size. Potential ways to alleviate this include adopting atrous convolutions, global pooling modules, or additional context extractors in the networks design. Using larger input image would also quadruple the number of FLOPs and likely inference time. Our networks would still be able to run in real-time, but at lower fps.

Finally with regard to metrics, we've chosen ECE, MCE, and area under the PAvPU curve. We chose these because MCE and ECE are global statistics that measure the calibration of the network, and area under the PAvPU curve is a statistics that encaptures local information. We have argued, however, that PAvPU or uncertainty accuracy is a defected metric that depends on the dataset itself and the performance of the network, and doesn't independently measure the quality of uncertainties. This meant that we lacked a suitable metric that takes the semantic nature of the task into account, such as how good “regional” or “structural” uncertainties are, which would be more useful for downstream tasks such as object avoidance.

5.2 Future Work

Neural architecture search (NAS) for semantic segmentation architectures Some state-of-the-art mobile neural networks have been developed from NAS, and we recommend a similar search for semantic segmentation structures, specifically for encoder-decoder networks. We conjectured that lightweight standard convolutional neural networks may be ill-suited for semantic segmentation tasks due to low feature width at high resolutions, which creates lossy processing of spatial information. Thus, using a more principled approach such as NAS may yield interesting encoder-decoder architectures that are more suited for preserving spatial information in semantic segmentation tasks.

Reducing the trade-off between performance and calibration error We presented Bayesian neural networks that have well-calibrated uncertainties but lower performance compared to their deterministic counterparts. We therefore concluded that there may be a trade-off between performance and calibration error. However, it raises the question of whether networks are more calibrated because they are underfitted, and whether or not the network can fit the data well and being calibrated. Our results using stochastic depth shows that it is indeed possible to retain high performance whilst reducing calibration error, but it’s unclear whether the same result can be extended to dropout-based methods. This remains an open-ended question that we leave for future research.

Effect of network architecture on Bayesian approximation Our experiments showed that using Bayesian inference that combine the results of smaller stochastic networks actually worsens performance of networks in average accuracy and mIOU. It still remains an open question on whether this is a property of MC Dropout method itself, whereby the posterior is a worse local minimum, or whether it only affects networks with certain architectures. We concur our findings with recent papers that have suggested that the uncertainty quality depends on network architectures and there are generally high variability of uncertainty qualities [71, 15]. A really interesting direction for future research is to find network architectures and hyperparameters such that consistently good posteriors can be obtained without compromising performance.

Theoretical properties of stochastic depth We've introduced the novel way of performing Bayesian approximation with stochastic depth by presenting empirical evidence that it obtains good posterior estimation with calibrated uncertainties. However, there still lacks a theoretical understanding of how the technique fits into the Bayesian approximation framework, and is an important direction for future research.

Efficient parallel processing of Bayesian approximations Although parallel computing makes Bayesian approximation at test time tractable at low T, it still increases computational costs by a non-negligible amount. Potential future direction include adding stochasticity only on the last few stages of the network, such that at test time the bulk information can be processed deterministically first before passing into stochastic parts of the network.

References

[1] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495.

[2] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

[3] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[4] Berger, J. O. (2013). *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media.

[5] Bilinski, P. and Prisacariu, V. (2018). Dense decoder shortcut connections for single-pass semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6596–6605.

[6] Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*. Springer.

[7] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR.

[8] Bornschein, J. and Bengio, Y. (2014). Reweighted wake-sleep. *arXiv preprint arXiv:1406.2751*.

[9] Brostow, G. J., Fauqueur, J., and Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97.

[10] Buciluă, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541.

[11] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2017a). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848.

[12] Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. (2017b). Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*.

[13] Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818.

[14] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

[15] Folgoc, L. L., Baltatzis, V., Desai, S., Devaraj, A., Ellis, S., Manzanera, O. E. M., Nair, A., Qiu, H., Schnabel, J., and Glocker, B. (2021). Is mc dropout bayesian? *arXiv preprint arXiv:2110.04286*.

[16] Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR.

[17] Gal, Y., Hron, J., and Kendall, A. (2017a). Concrete dropout. *Advances in neural information processing systems*, 30.

[18] Gal, Y., Islam, R., and Ghahramani, Z. (2017b). Deep bayesian active learning with image data. In *International conference on machine learning*, pages 1183–1192. PMLR.

[19] Gawlikowski, J., Tassi, C. R. N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R., et al. (2023). A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, pages 1–77.

[20] Graves, A. (2011). Practical variational inference for neural networks. *Advances in neural information processing systems*, 24.

[21] Grosse, R. and Martens, J. (2016). A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582. PMLR.

[22] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR.

[23] Guo, H., Liu, H., Li, R., Wu, C., Guo, Y., and Xu, M. (2018). Margin & diversity based ordering ensemble pruning. *Neurocomputing*, 275:237–246.

[24] Gustafsson, F. K., Danelljan, M., and Schon, T. B. (2020). Evaluating scalable bayesian deep learning methods for robust computer vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 318–319.

[25] Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

[26] Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001.

[27] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[28] Holder, C. J. and Shafique, M. (2022). On efficient real-time semantic segmentation: A survey. *arXiv preprint arXiv:2206.08605*.

[29] Hong, Y., Pan, H., Sun, W., and Jia, Y. (2021). Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes. *arXiv preprint arXiv:2101.06085*.

[30] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324.

[31] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[32] Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141.

[33] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[34] Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 646–661. Springer.

[35] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

[36] Jaderberg, M., Vedaldi, A., and Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*.

[37] Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., and Bengio, Y. (2017). The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 11–19.

[38] Kendall, A., Badrinarayanan, V., and Cipolla, R. (2015). Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*.

[39] Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28.

[40] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

[41] Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30.

[42] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[43] Lee, K., Lee, K., Lee, H., and Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31.

[44] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

[45] Lütjens, B., Everett, M., and How, J. P. (2019). Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668. IEEE.

[46] Ma, Y.-A., Chen, T., and Fox, E. (2015). A complete recipe for stochastic gradient mcmc. *Advances in neural information processing systems*, 28.

[47] MacKay, D. J. (1992). Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604.

[48] Malinin, A. and Gales, M. (2018). Predictive uncertainty estimation via prior networks. *Advances in neural information processing systems*, 31.

[49] McClure, P., Rho, N., Lee, J. A., Kaczmarzyk, J. R., Zheng, C. Y., Ghosh, S. S., Nielson, D. M., Thomas, A. G., Bandettini, P., and Pereira, F. (2019). Knowing what you know in brain segmentation using bayesian deep neural networks. *Frontiers in neuroinformatics*, 13:67.

[50] Mobiny, A., Yuan, P., Moulik, S. K., Garg, N., Wu, C. C., and Van Nguyen, H. (2021). Dropconnect is effective in modeling uncertainty of bayesian deep networks. *Scientific reports*, 11(1):5458.

[51] Moshkov, N., Mathe, B., Kertesz-Farkas, A., Hollandi, R., and Horvath, P. (2020). Test-time augmentation for deep learning-based cell segmentation on microscopy images. *Scientific reports*, 10(1):5068.

[52] Mukhoti, J. and Gal, Y. (2018). Evaluating bayesian deep learning methods for semantic segmentation. *arXiv preprint arXiv:1811.12709*.

[53] Müller, S., Hollmann, N., Arango, S. P., Grabocka, J., and Hutter, F. (2021). Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510*.

[54] Naeini, M. P., Cooper, G., and Hauskrecht, M. (2015). Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29.

[55] Neal, R. M. et al. (2011). Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2.

[56] Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528.

[57] Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32.

[58] Paszke, A., Chaurasia, A., Kim, S., and Culurciello, E. (2016). Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*.

[59] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer.

[60] Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29.

[61] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

[62] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[63] Sensoy, M., Kaplan, L., and Kandemir, M. (2018). Evidential deep learning to quantify classification uncertainty. *Advances in neural information processing systems*, 31.

[64] Simsekli, U., Badeau, R., Cemgil, T., and Richard, G. (2016). Stochastic quasi-newton langevin monte carlo. In *International Conference on Machine Learning*, pages 642–651. PMLR.

[65] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[66] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[67] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828.

[68] Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR.

[69] Tan, M. and Le, Q. (2021). Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR.

[70] Valdenegro-Toro, M. (2019). Deep sub-ensembles for fast uncertainty estimation in image classification. *arXiv preprint arXiv:1910.08168*.

[71] Verdoja, F. and Kyrki, V. (2020). Notes on the behavior of mc dropout. *arXiv preprint arXiv:2008.02627*.

[72] Wang, G., Li, W., Ourselin, S., and Vercauteren, T. (2019). Automatic brain tumor segmentation using convolutional neural networks with test-time augmentation. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 4th International Workshop, BrainLes 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Revised Selected Papers, Part II 4*, pages 61–72. Springer.

[73] Wang, J., Gou, C., Wu, Q., Feng, H., Han, J., Ding, E., and Wang, J. (2022). Rtformer: Efficient design for real-time semantic segmentation with transformer. *Advances in Neural Information Processing Systems*, 35:7423–7436.

[74] Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688.

[75] Wen, Y., Tran, D., and Ba, J. (2020). Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*.

[76] Wigness, M., Eum, S., Rogers, J. G., Han, D., and Kwon, H. (2019). A rugd dataset for autonomous navigation and visual perception in unstructured outdoor environments. In *International Conference on Intelligent Robots and Systems (IROS)*.

[77] Yao, L., Kanoulas, D., Ji, Z., and Liu, Y. (2021). Shorelinenet: An efficient deep learning approach for shoreline semantic segmentation for unmanned surface vehicles. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5403–5409. IEEE.

[78] Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., and Sang, N. (2018). Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 325–341.

[79] Yu, L., Wang, S., Li, X., Fu, C.-W., and Heng, P.-A. (2019). Uncertainty-aware self-ensembling model for semi-supervised 3d left atrium segmentation. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part II 22*, pages 605–613. Springer.

[80] Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856.

Network performance

We show the prediction outputs of the best performing *Large* Network, see Section 4.2.2 for details. The outputs are not handpicked but randomly selected from the test set.

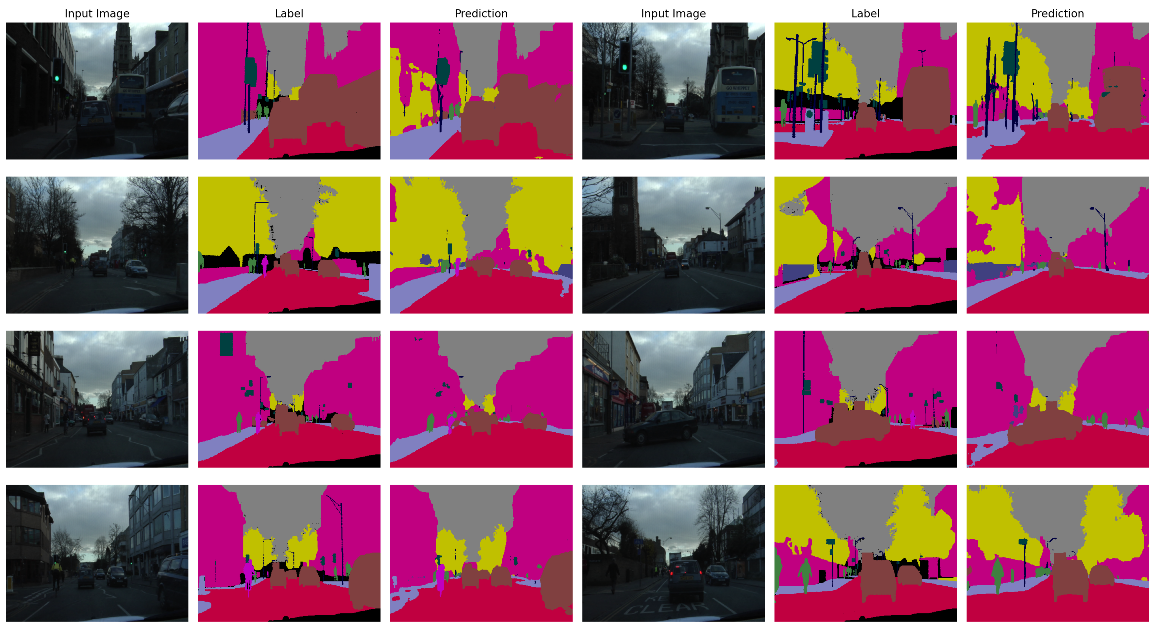


Figure A.1 Prediction and ground truth of the *large* network on the CamVid Test set. Note that black pixels in the ground truth represent "void" pixels.

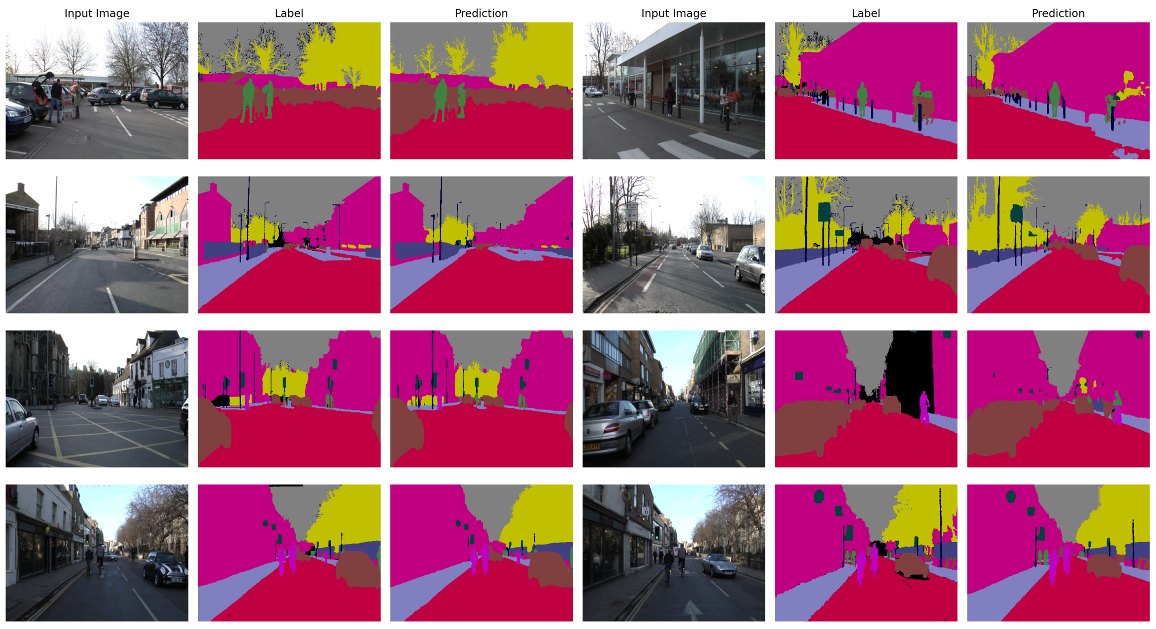


Figure A.2 Prediction and ground truth of the *large* network on the CamVid Test set. Note that black pixels in the ground truth represent "void" pixels.

Uncertainty outputs

In the following show a few more qualitative uncertainty predictions of our Bayesian variations.

Dropout

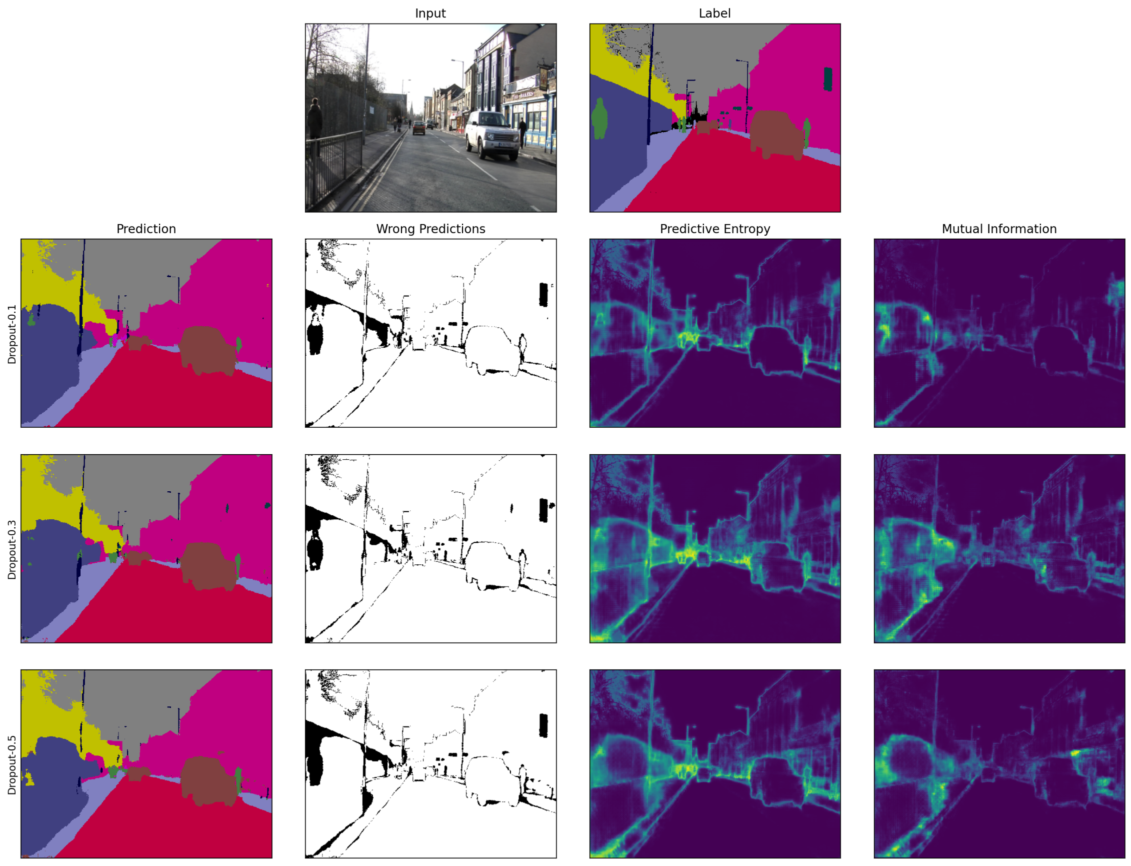


Figure B.1 Uncertainty breakdown for dropout variants

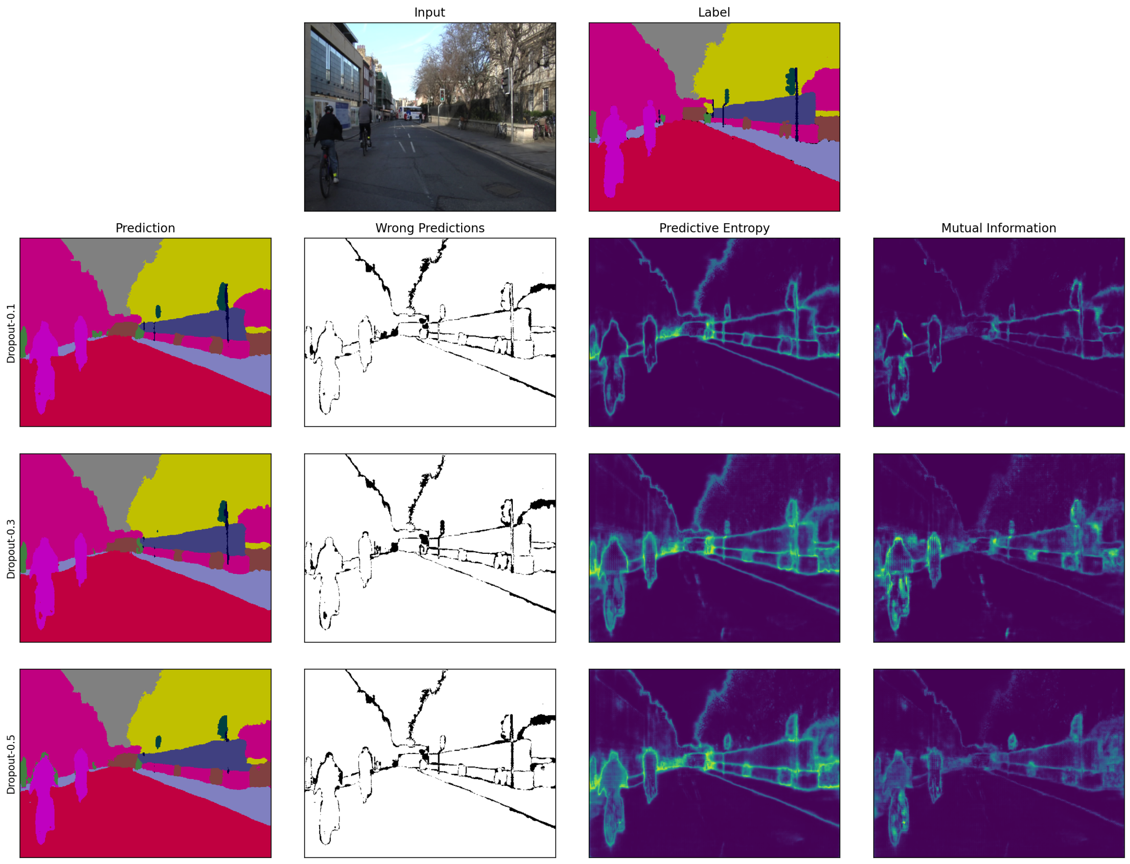


Figure B.2 Uncertainty breakdown for dropout variants

Stochastic depth

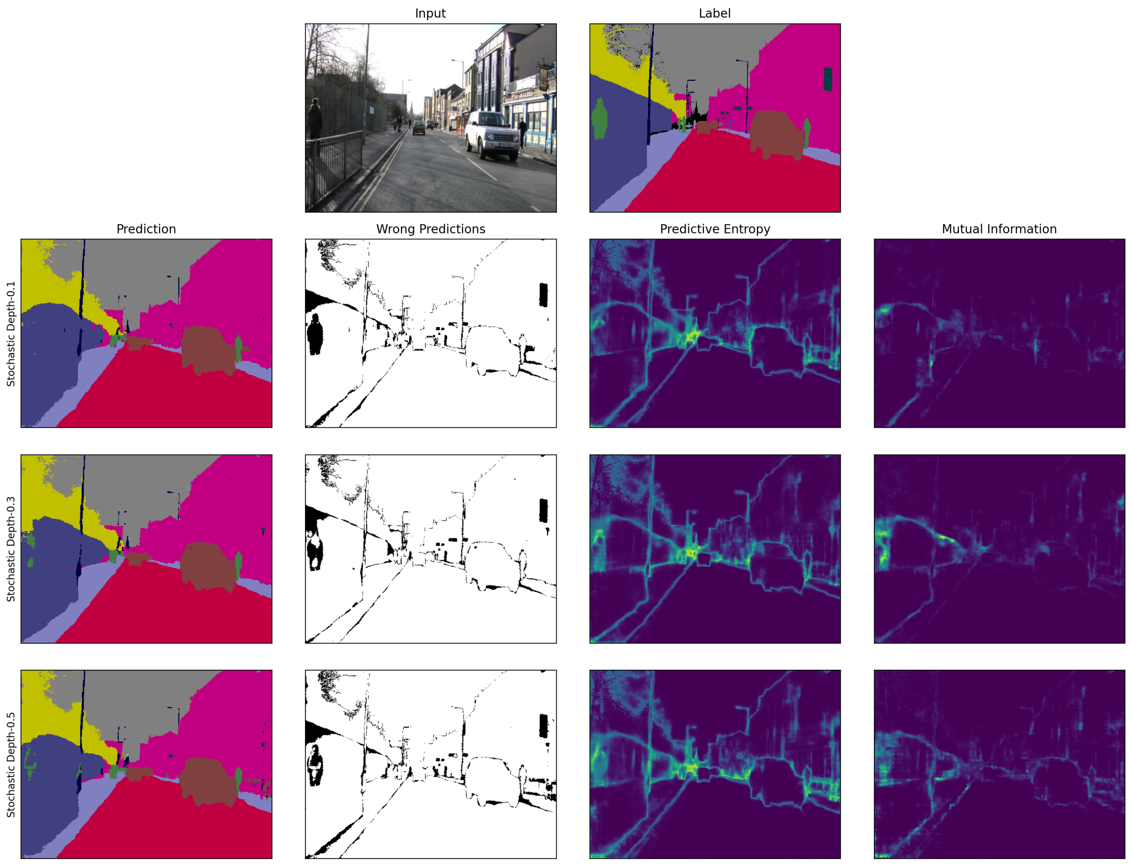


Figure B.3 Uncertainty breakdown for stochastic depth variants

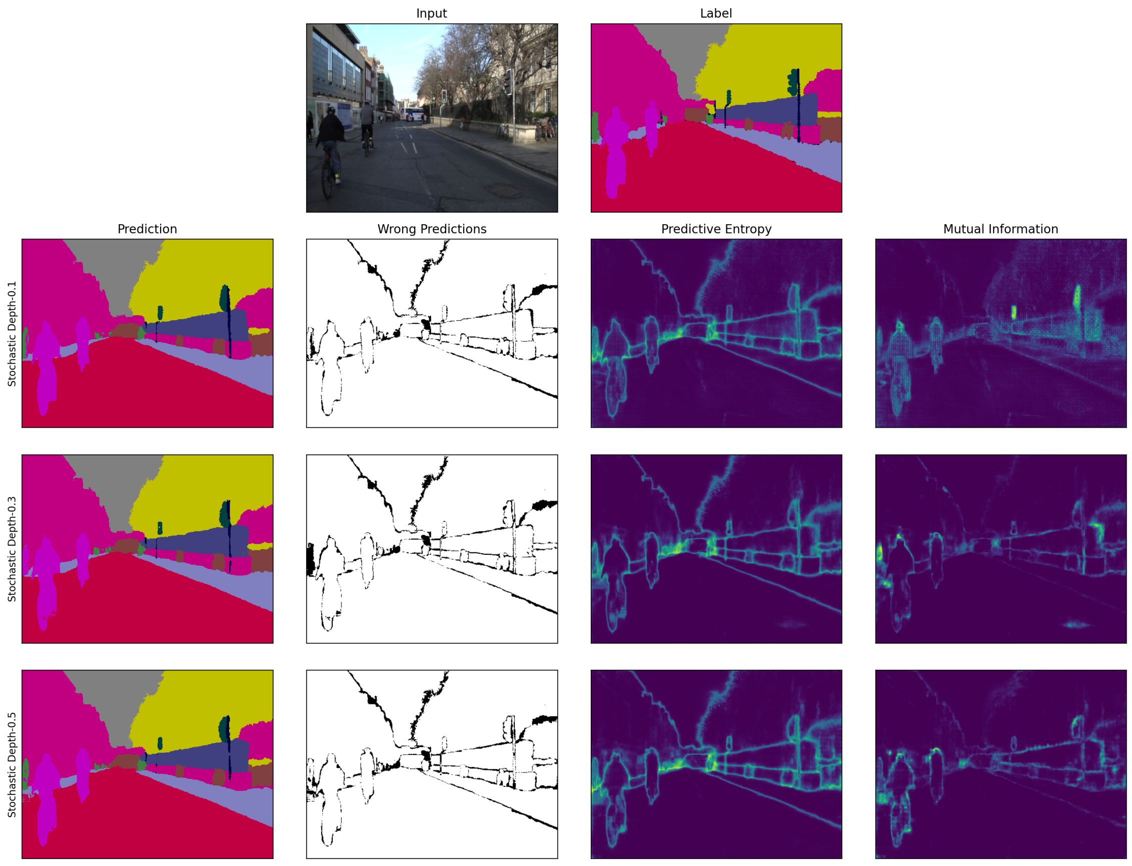


Figure B.4 Uncertainty breakdown for stochastic depth variants

Combined

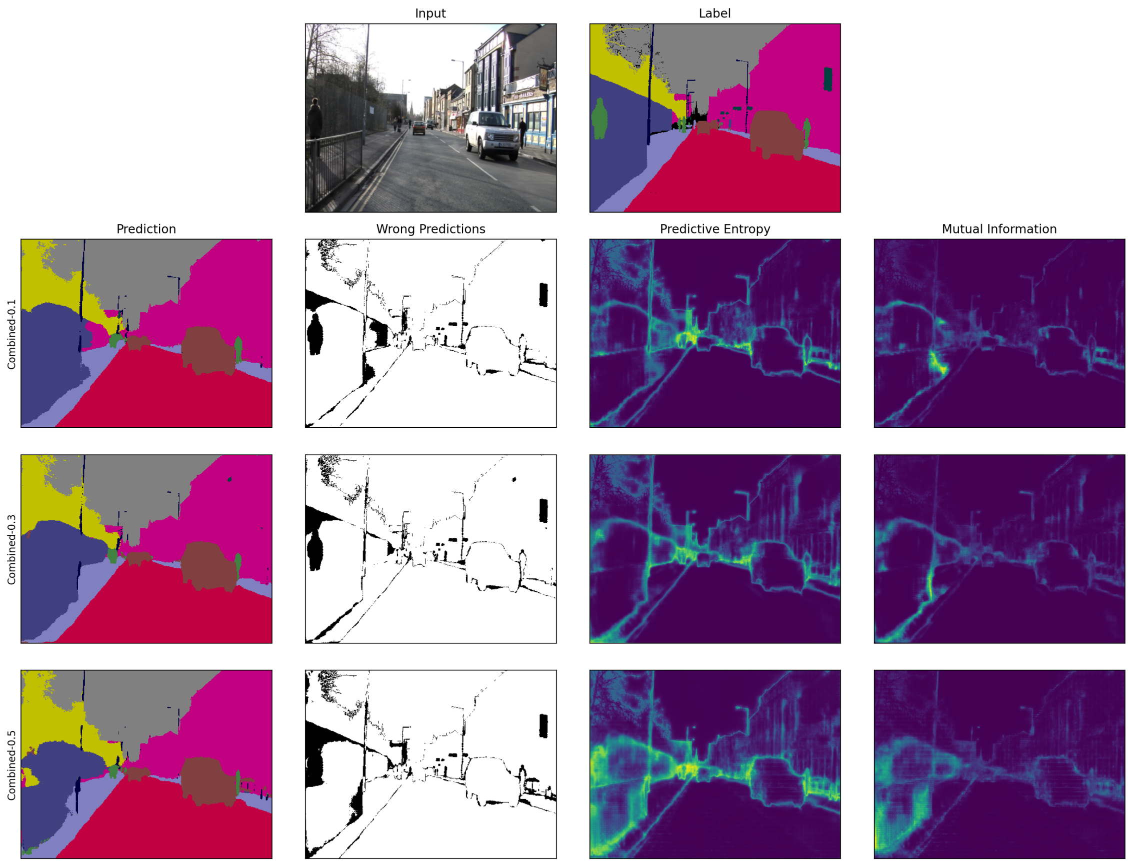


Figure B.5 Uncertainty breakdown for combined variants

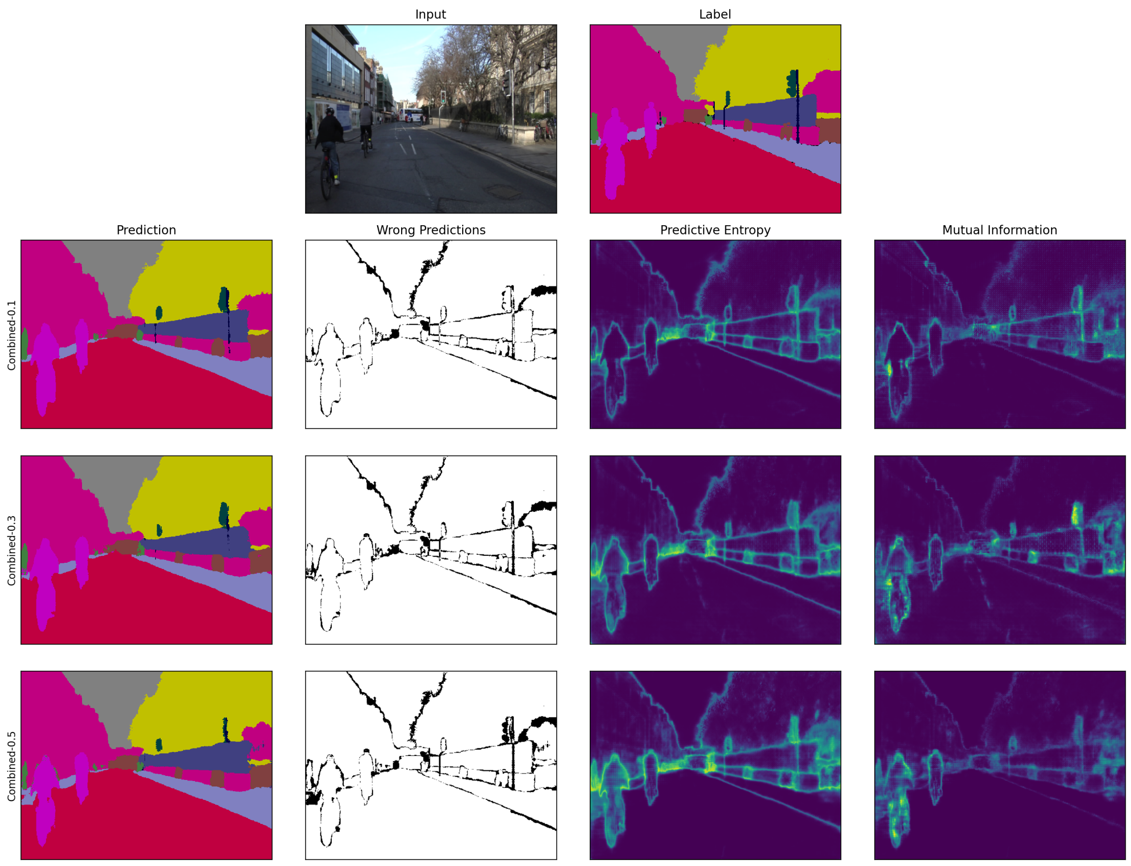


Figure B.6 Uncertainty breakdown for combined variants

Reliability Graphs

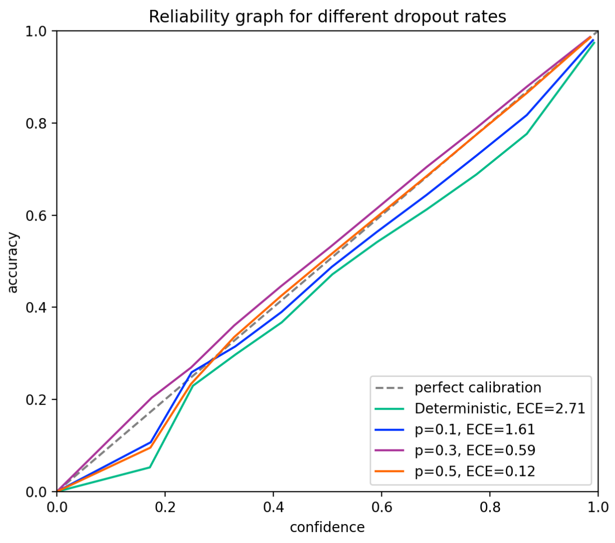


Figure C.1 Reliability graph for different dropout probability, using $T = 10$.

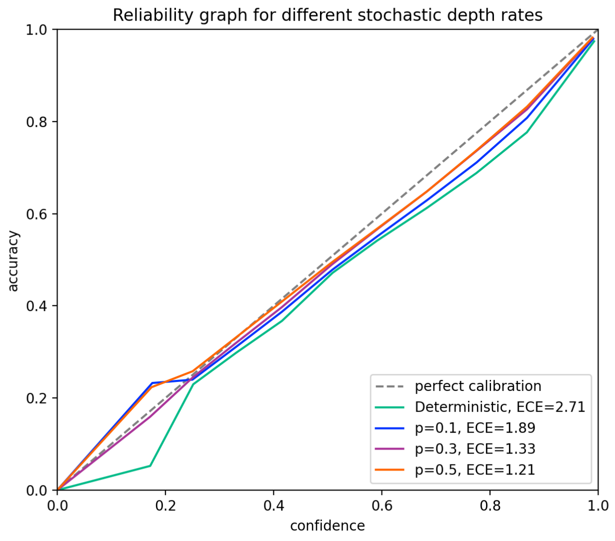


Figure C.2 Reliability graph for stochastic depth variants, using $T = 10$.

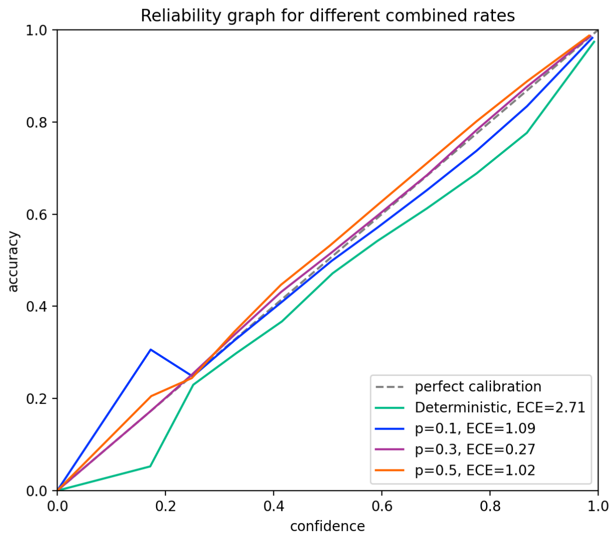


Figure C.3 Reliability graph for combined variants, using $T = 10$. Note that the network become underconfident at high probabilities