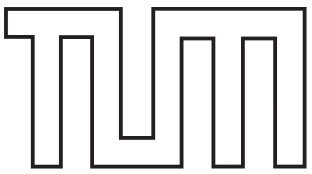


Technische Universität München
Chair of Media Technology
Prof. Dr.-Ing. Eckehard Steinbach

Master Thesis

Uncertainty-Based Improvement of a Visual Classification System

Author:	Jianxiang Feng
Matriculation Number:	03679926
Address:	Connollystr. 03/W06 80809 Munich
Advisor:	Prof. Dr. -Ing. Eckehard Steinbach
Supervisor:	Dr. Zoltán-Csaba Márton(DLR), Maximilian Durner(DLR)
Begin:	11.10.2018
End:	25.04.2019

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.

München, April 21, 2019 _____

Place, Date Signature

München, April 21, 2019 _____

Place, Date Signature

Abstract

Classifiers based on deep neural networks have achieved tremendous success on different kinds of task. However, at the same time, this kind of classifier could not provide reliable uncertainty estimation about their predictions, which can easily lead to seriously overconfident predictions and sub-optimal decision in the down-stream tasks. From a systematic perspective, this is undesirable and even hazardous in wide range of safety-critical applications such as deployment of robots, medical diagnosis and so on. On the other hand, in object recognition task, some predictions are uncertain because they have similar appearances and thus ambiguous, although their relationship with contextual objects are distinguishable. For example a marker has similar appearance with a toothbrush, when a calculator is recognized nearby, a prediction of marker should have higher probability. This kind of uncertainty from data ambiguity can be considered in order to obtain a more robust classifier.

In this work, regarding the first issue, we investigate applying dropout variational inference and Laplace approximation in Bayesian neural network to obtain reliable model uncertainty estimation on object recognition task. Predictions with low uncertainty can be used to label observed data automatically, which can decrease manual effort on training a more accurate classifier when robot is confronted with objects in real world environment, which has a slight domain gap with objects in training set. When it comes to the second issue, object co-occurrence statistics is introduced as scene contextual information via conditional random field to handle the data ambiguity and improve the exiting results. We have conducted experiments to show that reliable uncertainty estimates can be obtained via dropout inference and even improved with proposed variants of dropout inference and its ensemble. Additionally, those improved uncertainty estimation can be used with semantic statistic to train a more accurate domain specific classifier.

Contents

Chapter 1

Introduction

1.1 What is uncertainty?

This question seems a little philosophical and difficult to answer. However, in the context of modeling in machine learning, I find one answer reasonable and illustrative, that is "Uncertainty arises because of limitations in our ability to observe the world, limitations in our ability to model it, and possibly even because of innate nondeterminism."[KFB09]. In spite of innate nondeterminism, uncertainty can be further categorized into two main types :**epistemic uncertainty** and **aleatoric uncertainty** [DKD09][SBD$^+$14] [KG17].

The former one, epistemic or model uncertainty illustrates the uncertainty in modeling, which is associated with **imperfect models** of the real world because of insufficient or imperfect knowledge of reality, which corresponds to our ability of modeling, and thus refers to uncertainty caused by a lack of knowledge, i.e., it refers to the epistemic state of the decision maker. Therefore it may be reduced through collecting more knowledge. Examples for this kind of uncertainty can be uncertainty of model parameter, or uncertainty of model structure depending on which kind of model and so on. One simple example to illustrate this kind of uncertainty would be linear regression with limited observed points, where we can use different curves to explain those points and then predict unobserved points with those curves. If we have observed enough points in some intervals, the models or curves that are able to explain the data are more restricted and if there are not enough points, there would be more curves that can explain these points, which indicates high uncertainty. In reality, it's quite normal to have inadequate data, and thus difficult to model the underlying distribution quite well and precisely. Therefore we could explain the data with several or many different models, where the model uncertainty comes in.

The latter one, aleatoric or data uncertainty, accounts for the **inherent randomness** of underlying phenomenon which is expressed as variability in observed data and meanwhile associated with the limitation of our observing ability. It's treated as non-reducible within our ability to observe the data. Examples for this kind of uncertainty could be noise induced

by limited resolution of sensor or ambiguity from content of data which are unrelated to the model we use to represent these data. One example can be classification of images captured by cameras with different resolutions, images with lower resolutions would have higher uncertainty because they are more ambiguous and less illustrative compared with those with higher resolutions.

In other words, epistemic uncertainty refers to the reducible part of the (total) uncertainty, whereas aleatoric uncertainty refers to the non-reducible part. Model uncertainty and aleatoric uncertainty are combined and expressed in the final prediction, which is so called predictive uncertainty.

1.2 Why do we need uncertainty?

Aforementioned model uncertainty represents the belief of model about its predictions and thus bring us more useful information about predictions related with model and observed data instead of just a crispy prediction, whose predictor, is always overconfident when advanced neural network architectures are employed[GPSW17]. As far as I am concerned, this kind of uncertainty plays an important role in two following scenarios.

The first one are those requiring **high safety guarantee** or **optimal decision-making**, in which false predictions can cause hazardous consequences, which is so called AI safety[AOS+16]. With reliable model uncertainty estimation, we are able to know when the model is uncertain about its prediction and then people or operator are aware of that and can take corresponding counter measures in order to avoid unnecessary accidents. With more and more wide-spread deep learning algorithms in real life applications, concrete examples for this case include steering control in self-driving car[MGK+17], disease diagnosis in medical domain[LAA+17], or even tasks in aerospace or other domains requiring higher precision and stronger robustness and so on. Although someone would argue that if the trained model can perform perfectly, that means that it is able to achieve 100% accuracy, then we do not need this kind of uncertainty information. However, as mentioned in [DL91], it's hard or even impossible to have enough training data to define a precise model, which is in coincidence with reality because it's hard to define a task very precisely and get access to enough samples drawn from its real distribution. Uncertainty information would be more valuable and important when algorithms are confronted with multiple tasks[KGC]. Other relevant tasks such as misclassified and out-of-distribution data detection[HG16] and adversarial attacks [KGB16][FCSG17] have also attracted more and more research interests in uncertainty estimation in order to address these challenges.

The second one are scenarios requiring **interaction between algorithms and people or environments**. Model uncertainty can build a bridge between machine learning algorithms and human beings which can construct a more robust and more data-efficient machine learning system. This scenario involves different kinds of tasks such as active learning[GIG17], reinforcement learning[BCKW15][OBPVR16][GMR], automatically la-

beling, industrial components inspection and so on. The idea behind that is, with reliable model uncertainty estimation we know which data sample or prediction the model is unfamiliar with. In active learning or reinforcement learning, those unfamiliar data samples can be used to train our model more quickly and efficiently. In tasks involving human robot interaction, we can know when the model requires help from human experts based on uncertainty estimation. For example, in industrial component inspection task, we firstly train a model which can perform quite well on some evaluation datasets and then deploy them in real applications. At this moment, we assume that all predictions made by the model are true all the time. However, there is no guarantee that this model has learned to represent the real distribution of data perfectly and is robust against slight or even large domain transfer. If the distributions of the real world data vary because of unexpected factors such equipments aging or changes of weather condition. The model would fail silently and lead to unexpected accidents. With reliable model uncertainty estimation, this kind of weakness can be mitigated or even eliminated.

On the other hand, aleatoric uncertainty mainly assist in **improving model performance** by quantifying and considering this kind of information by representing noisy level of predictions and making good use of data points with less noise. By taking this information into account, the model can be trained more optimally and yield better performance, some examples of image data and Lidar data can be referred to [KC16][FRD18]. Instead of representing noise, data uncertainty also includes uncertainty from semantic view point, which we can also incorporate into training and yield more robust model. This kind of uncertainty is not quantified explicitly but can be incorporated conceptually with help of specifically designed model and high level statistics cues or features to **disambiguate prediction** directly. One example of this can be modeling contextual information among pixels with conditional random field in semantic segmentation task[KK11][SM+12][LSVDHR16]. The idea behind that is simple and conditional random field provides a principle way to achieve this idea. Since uncertainty caused by appearance ambiguity represents strong correlation between different categories. Conditional random field can model correlation or dependency between random variables by taking into account high order potentials. In semantic segmentation task, because of the reason of tractable computation, at most second order potentials are taken into account and can yield better performance compared results with just first order potentials.

1.3 How can we obtain and handle uncertainty in deep learning?

As we know, deep learning is a powerful black box model while it has achieved tremendous success in different tasks. Therefore to obtain and handle uncertainty in deep learning model and data can help us to get more understanding about this black box model and make deep learning based application more robust and safer.

Regarding model uncertainty or quality measure that acts similar to model uncertainty, there are many different ways to obtain this kind of quality measure of prediction. On the one hand, because of intractability of real model uncertainty in deep neural networks, there are some **ad-hoc approaches** that try to obtain such a quality measure on specific tasks which are solved usually with model uncertainty such as misclassified and out-of-distribution(OOD) detection task. [LLS17] combines temperature scaling and input preprocessing which is called adversarial training together and yields a simple but effective out-of-distribution sample detector, but this approach requires additional out-of-distribution dataset for training which is hard to collect in reality. [DT18] modifies the loss function to train a classifier against OOD data, however this approach is hard to generalize in case of slight domain transfer. [LLLS17] treats uniform distribution as ground truth distribution of OOD data and show that OOD data can be generated via generative adversarial network(GAN). This approach seems practical, but training GAN effectively does not always work for images of large size because of high dimension of data distribution.

On the other hand, there are some more theoretically sound approaches which employ **Bayesian neural network**[Mac92][Nea12], **bootstrap**[OBPVR16] or **ensemble** method[LPB17] or even ensemble of Bayesian neural network[SG18]. Although Bayesian neural network provides a principal way to obtain model uncertainty by putting randomness on model parameters, it's difficult to infer the exact posterior distribution over model parameters of deep neural network because of its complex architecture and large scale dataset. Therefore approximate inference plays an important role in addressing this issue. The golden standard for approximate inference is Hamiltonian Monte Carlo[Nea12], which requires dealing with whole batch training data and storing the samples of posterior distribution. While the former issue is addressed partially by stochastic gradient Langevin Dynamics[WT11], the latter one attracts many research interests like [BRMW15][WVL$^+$18]. However, since it's a Markov Chain Monte Carlo(MCMC) method, to assess the convergence is still non-trivial.

There are some alternatives to MCMC, one popular one is variational inference(VI)[HVC93]. In VI, this inference problem is cast into optimization problem by minimizing the Kullbach-Leibler divergence between the approximate posterior distribution and real posterior distribution. Therefore approximate distribution family needs to be chosen and thus has a large impact on the result, which may impede the performance if the chosen distribution family is not flexible enough for the problem. [Gra11] firstly employed fully factorized Gaussian with a biased gradient estimator on a practical problem. Later [BCKW15] improved the result with the same approximate distribution but different estimator with help of "reparameterization trick"[KW13]. HLA15 also used the same approximate distribution but with expectation propagation[Min01] instead of VI. Since the fully factorization assumption may impose an restriction on the flexibility of the approximate distribution between the correlations between weights could not be learned and expressed. By taking this into account, there are many attempts trying to use more expressive approximate distribution such as Bernoulli (treated as mixture of two Gaussian with small variance)[GG16] and Gaussian [KSW15] which capture variances

of rows in weight matrix. Another more expressive one is matrix variate Gaussian distribution[LW16][SCC17][ZSDG17], which capture both rows and columns correlation in weight matrix. Additionally inspired by the idea of normalizing flow in latent variable models [LW17]applied normalizing flows to auxiliary latent variables to produce more flexible approximate posteriors.

Another alternative is Laplace approximation[Mac92], which put a Gaussian centered at MAP estimate of model parameters with a covariance determined by the inverse of Hessian of the log-likelihood. The main issue of this approach for the large neural network is the large size of Hessian which is quadratic of the number of parameters and inversion operation on it whose complexity is cubic over number of elements of Hessian matrix. Recently [RBB18] working out this problem by approximating the Hessian or the Fisher matrix of log-likelihood with kronecked-factorization which reduces the storage size and complexity of inversion operation. The resulting posterior can also be treated as matrix variate Gaussian.

However, most of those ideas are verified with a relatively simple architecture and on toy or small-scale datasets and need to be modified a lot in order to serve the existing advanced architectures, which is undesirable in our work and many practical applications. Therefore dropout inference[GG16] looks promising considering this restriction and thus is adopted to obtain model uncertainty in our work. On the other hand, scalable Laplace approximation [RBB18] has the potential resolving this issue, because it requires only one MAP point estimate of model parameters, so there is no need to modify the training phase which could save a lot of computation effort and unnecessary modifications for many existing architectures.

When it comes to aleatoric uncertainty, as noise of data itself, it can be modeled by adding another head on top of the network [KG17]. However, the uncertainty caused by appearance ambiguity is not trivial to include in deep neural network training. Therefore another model, **conditional random field**[LMP01], is adopted based on the predictions of discriminative neural network classifier. As is mentioned in previous section, choice of features of second order potentials is key factor in this problem. In this work, because we work on scene object recognition task, therefore the correlation between objects in specific scene are of importance. The correlation can be represented by co-occurrence matrix co-occurrence[LRKT10][RV09][GRB08][RVG+07].

1.4 Contributions and Structure

This work can be divided into two main parts, both focusing on uncertainty-based improvement for a deep learning based classifier on object recognition task.

The first part is to employ dropout inference to obtain reliable and calibrated uncertainty from classifier with ResNet50[HZRS16] as backbone. We evaluate the prediction and calibration performance of this probabilistic classifier and then based on uncertainty

estimation, we select predictions with low uncertainty/high confidence as automatically labeled data which is used for training a more accurate classifier while the performance of original one declines a lot in case of slight domain transfer which is caused by possibly by light condition changes, appearance changes and even recording equipment changes. This is proof-of-concept idea which tries to prove that improved uncertainty estimation can assist in automatic labeling and reducing manual effort in fine-tuning a new classifier in case of slight domain transfer during deployment of robot. In this part, we consider advanced version of dropout inference and its variants for improving the results both in terms of accuracy and calibration.

The second part is to handle the uncertainty caused by appearance ambiguity. To note that again, this kind of uncertainty is not modeled directly, but the concept is taken into account in choosing model and features. To achieve this goal, we choose conditional random field to model the correlation between objects with similar appearance but different semantics. As key factor of the model, we employ binary co-occurrence matrix of categories as pairwise features.

There are several main contributions of this work:

- to show that dropout inference and Laplace approximation can obtain better uncertainty information on multi-class classification problem with 51 classes, which can be further improved by learning dropout rates during training and its ensemble.
- to show that manual effort in data collection in slight domain transfer learning task can be reduced by making use of reliable uncertainty estimation.
- to show that uncertainty caused by appearance ambiguity can be resolved by integrating contextual information via condition random field.

The structure of this work is as follows, in order to give reader a clear theoretical background of techniques in this work, there is a brief review of theory of Bayesian neural network(BNN) including dropout inference and Laplace approximation as well as conditional random field(CRF) in chapter 2. In chapter 3, details about technical approaches including modifications of ResNet50, variants of dropout and combination of BNN and conditional random field are explained. In chapter 4, different experiments are performed and the results are showed and discussed. There are mainly three kinds of experiments in this chapter which are uncertainty estimation, automatically labeled data for fine-tuning and performance when combining BNN and CRF. Then summary, conclusion as well as meaningful future work come in chapter 5.

Chapter 2

Theoretical Background

2.1 Bayesian neural network

In this chapter, a brief review on theory of Bayesian neural network is given. Because calculating the exact posterior distribution over the parameters of large and complex neural network is intractable, proper techniques to perform approximate inference are required, which can be categorized into two mainstreams, variational inference(VI) and Markov Chain Monte Carlo(MCMC). However, since larger dataset and more complex neural network are popular and working pretty well nowadays, traditional approximate inference techniques are difficult to scale to large dataset and complex advanced architectures. Therefore modern approximate inference needs to be adopted. In this section, we introduce and evaluate two modern approximate inference methods for neural network, which are dropout variational inference[GG16] and scalable Laplace approximation[RBB18]. Additionally, we introduce concrete dropout [GHK17] to improve uncertainty estimation by learning dropout rate from data instead of tuning them by hand before training.

2.1.1 Introduction

Let $\mathcal{D}$ denote an observable dataset consisting of a set of input and output pairs, that is $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\} = \{(x_i, y_i)_{i=1}^N\}$, where $x_i \in \mathbb{R}^D$ and $y_i \in \mathbb{Y}$, $\mathbb{Y}$ is set of labels, and $f^{\boldsymbol{\omega}}$ denote one parametric model which is in our case neural network with $\boldsymbol{\omega}$ its parameters which are weights $W_{1:L}$ and biases $b_{1:L}$ for L layers.

In supervised learning, our goal is to learn a distribution of output conditioned on input, $p(\mathbf{y}|\mathbf{x})$ that can explain the underlying data distribution based on the observables $\mathcal{D}$ which are samples drawn from the underlying data distribution. Then this learned model can be used to make predictions on unobservable data samples under the same data distribution. In classification where output is label represented by discrete integer, the likelihood function

is defined based on parametric model, which is softmax score:

$$p(y = d|\mathbf{x}, \boldsymbol{\omega}) = \frac{exp(f_d^{\boldsymbol{\omega}})}{\sum_{d'} exp(f_{d'}^{\boldsymbol{\omega}})} \tag{2.1}$$

where d is possible class of random variable y, and f_d^w represents the probability that class d is assigned to y.

In regression case, the likelihood is Gaussian:

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega}) = \mathcal{N}(\mathbf{x}; f^{\boldsymbol{\omega}}(\mathbf{x}), \tau^{-1}\mathbf{I}) \tag{2.2}$$

with model precision τ, which represents the inverse of noise level of the outputs.

As mentioned above, learning aims to find model(s) which is parameterized by a set of parameters that can explain the data well, which means the likelihood should be maximized w.r.t. model parameters over the observables. On the other hand, some prior constraints are imposed on the model via prior distribution of the model parameters $p(\boldsymbol{\omega})$. The probability distribution of model parameters is updated from prior distribution into posterior distribution after observing training dataset $\mathcal{D}$ via Bayes' theorem:

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\mathbf{Y}|\mathbf{X})} \tag{2.3}$$

where $p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})d\boldsymbol{\omega}$ is so called model evidence or marginal likelihood, whose integration is always intractable.

After obtaining posterior distribution over model parameters, we can make predictions by marginalizing the likelihood of unseen input points such as $x^\star$ over model parameters, which leads to predictive distribution over output:

$$p(y^\star|x^\star, \mathcal{D}) = \int p(y^\star|x^\star, \boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D})d\boldsymbol{\omega} \tag{2.4}$$

Pointing out the difference between deterministic neural network and Bayesian neural network can help understanding the mechanism of Bayesian neural network better. In figure 2.1, we use graphical model to express these two kinds of neural network, in which solid point denotes deterministic variable, and circle denotes random variable, while the shaded circle denotes observed random variable. Plate notation denotes N observed data pairs $(\boldsymbol{x}, \boldsymbol{y})$. Figure 2.1 demonstrates that parameters $\boldsymbol{\omega}$ in deterministic is normal variable which is a point estimate done by Maximum a posterior method:

$$\boldsymbol{\omega}^\star = \underset{\boldsymbol{\omega}}{argmax}\{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})\} \tag{2.5}$$

On the other hand, in Bayesian neural network, the model parameter $\boldsymbol{\omega}$ is random variable with distribution parameterized by θ. To note that for explanation of concept here, we do not make any assumption on the function parameterized by θ, which means it can be arbitrary function. This distribution over model parameters can be inferred based on Bayes' rule in Eq.2.3. However, for model with large number of parameters, it's hard to compute the model evidence (denominator of r.h.s of Eq.2.3) with integration tractably. Approximation needs to be employed to solve this problem. It's worth noting that if we choose the approximate distribution to be delta function, $q_\theta(\boldsymbol{\omega}) = \delta(\boldsymbol{\omega} - \theta)$, Bayesian neural network is recovered into deterministic neural network. Because the computation of model evidence $p(\mathbf{Y}|\mathbf{X})$ is always intractable in complex model and large dataset, we need to resort to approximate inference, which will briefly be introduced in next section.

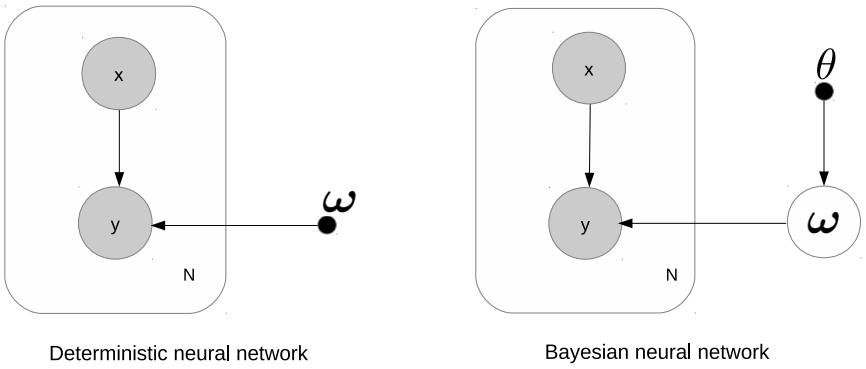


Figure 2.1: Difference between parameter estimation of deterministic neural network and Bayesian neural network.

2.1.2 Dropout variational inference

Introduction

Variational inference cast inference into optimization by minimizing the Kullbach-Leibler divergence between approximate posterior distribution and the real posterior distribution. However, there is no analytical definition of this KL divergence because the real posterior distribution is unknown. We can derive a lower bound which is also called evidence lower bound($ELBO$) which bounds the log marginal likelihood with Jensen's inequality. And

from that we know that marginal likelihood is the sum of $ELBO$ and KL divergence between approximate posterior and real posterior. The derivation is given in the following:

$$\begin{aligned}
\log(p(\boldsymbol{Y}|\boldsymbol{X})) &= \log(\int p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})d\boldsymbol{\omega}) \\
&= \log(\int q_\theta(\boldsymbol{\omega})\frac{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{q_\theta(\boldsymbol{\omega})}d\boldsymbol{\omega}) \\
&\geq \int q_\theta(\boldsymbol{\omega})\log(\frac{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{q_\theta(\boldsymbol{\omega})})d\boldsymbol{\omega} \\
&= \mathbb{E}_{q_\theta(\boldsymbol{\omega})}[\log(p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega}))] - KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))) \\
&= ELBO
\end{aligned} \tag{2.6}$$

where $p(\boldsymbol{Y}|\boldsymbol{X})$ is the likelihood, $p(\boldsymbol{\omega})$ is the prior distribution over model parameters, $q_\theta(\boldsymbol{\omega})$ is the approximate posterior distribution over parameters which is parameterized by θ.

We can get $\log(p(\boldsymbol{Y})|\boldsymbol{X})$ by calculating the sum of $ELBO$ and KL divergence between approximate posterior $q_\theta(\boldsymbol{\omega})$ and real posterior $p(\boldsymbol{\omega}|\boldsymbol{X},\boldsymbol{Y})$:

$$\begin{aligned}
&ELBO + KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\boldsymbol{X},\boldsymbol{Y})) \\
&= \int q_\theta(\boldsymbol{\omega})\log(\frac{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{q_\theta(\boldsymbol{\omega})})d\boldsymbol{\omega} + \int q_\theta(\boldsymbol{\omega})\log(\frac{q_\theta(\boldsymbol{\omega})}{p(\boldsymbol{\omega}|\boldsymbol{X},\boldsymbol{Y})})d\boldsymbol{\omega} \\
&= \int q_\theta(\boldsymbol{\omega})\log(p(\boldsymbol{Y}|\boldsymbol{X}))d\boldsymbol{\omega} \\
&= log(p(\boldsymbol{Y}|\boldsymbol{X}))
\end{aligned} \tag{2.7}$$

When we maximize the $ELBO$ w.r.t the parameter of approximate posterior θ, which is also called **variational parameter**. We use these two terms interchangeably in the following. It's equivalent to minimizing the KL divergence because the log marginal likelihood is not a function of θ. Then we have a well-defined objective which is the $ELBO$, in which the first term is called expected log likelihood which ensures the model can explain the data well and the second term is called regularization term which ensures the approximate posterior does not deviate too far from the prior distribution. Now the inference problem has been cast into an optimization problem:

$$\theta^\star = \underset{\theta}{argmin}\{KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\boldsymbol{X},\boldsymbol{Y}))\} \tag{2.8}$$

which is equivalent to

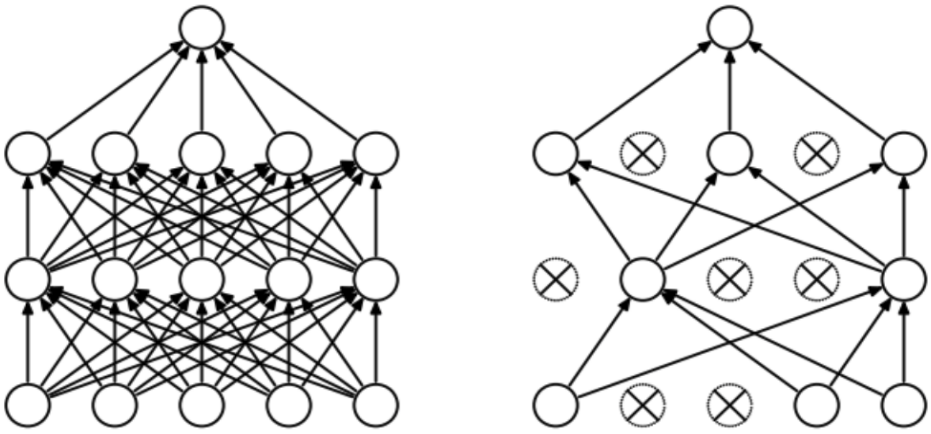
$$\theta^\star = \underset{\theta}{argmax}\{ELBO\} \tag{2.9}$$

However, there are still some difficulties if we want to solve this optimization with this objective. The first one is to deal with large size dataset, which induces large computation in the expected log likelihood term. [Gra11] shows that this can solved by data subsampling which is also called stochastic optimization. Another one is that we need to obtain the derivatives of $ELBO$ w.r.t. approximate parameter θ. Since the model parameters are samples from approximate distribution, good estimator for the derivatives is required which will be introduced in next subsection.

Dropout variational inference

Dropout[SHK$^+$14] is originally introduced as regularization approach in training deep neural network which can improve the generalization performance. Although the author said this idea is inspired from human beings sexual reproduction, there are different interpretations trying to explain why it can work such as ensemble perspective and Bayesian perspective. In this subsection, the Bayesian interpretation of dropout is introduced and used for improving the uncertainty estimation of neural network.

The mechanism of dropout is straightforward, each units of specific layer is multiplied by a random variable under Bernoulli distribution with $1 - p$ as its parameter, where p is dropout rate. In each iteration of training, dropout is turned on, which means each unit is multiplied by the sample drawn from Bernoulli distribution in forward propagation which is kept in derivatives back propagation during current iteration. In testing, the sampling phase is turned off, only one forward propagation is needed to obtain predictions. Normally, in order to avoid rescaling weights in testing, which is used to keep the output magnitudes in the same scale when dropout is off, rescaling of the output of dropout is always done in training. In figure 2.2, there are two figures showing how dropout is turned on and off.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.2: How dropout works[SHK$^+$14].

Bayesian interpretation of dropout: In VI, the goal is to minimize the KL divergence between approximate distribution and the real posterior distribution over the model parameters. This objective is equivalent to maximization of evidence lower bound($ELBO$). When dropout is interpreted in Bayesian way, the distribution over hidden units is reformulated as distribution over weight matrices. The training objectives of neural network with dropout is proved to be similar as the $ELBO$ of Bayesian neural network with Bernoulli distribution factorized over the input dimension of weight matrix. In the following, we will explain this interpretation including following key factors: **approximate distribution**, **training objective**, **marginalization in testing** by using one simple example in classification case in figure 2.3, in which we define the hidden layer as the first layer and output layer as the second layer and assume that we use L2 regularization and put a prior distribution of fully factorized Gaussian over weights initially, which can be generalized to multi-layer neural network easily.

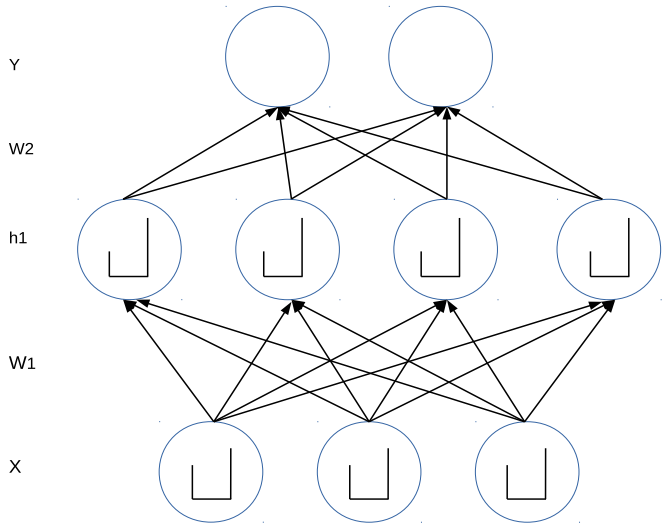


Figure 2.3: An example of two layer neural network with dropout, a Bernoulli random variable is imposed on each unit of input layer and hidden layer.

Approximate distribution: Let's denote $\boldsymbol{y} \in \mathbb{R}^{m \times D_2}$ as output, $\boldsymbol{x} \in \mathbb{R}^{m \times D_0}$ as input, $\boldsymbol{h}_1 \in \mathbb{R}^{D_1}$ as the response of hidden layer, where m represents number of data instances, D_i is dimensionality of i-th layer, where 0-th layer represents input layer and $i \in \{1, .., L\}$, $L = 2$ in this example. Further we define $\boldsymbol{\omega} = \{(\boldsymbol{W}_i)_{i=1}^{L}\}$ as model parameters, and $\boldsymbol{\epsilon}_i \in \mathbb{R}^{D_{i-1}}$ as the Bernoulli distributed random vector parameterized by $\boldsymbol{p}_i \in \mathbb{R}^{D_{i-1}}$, for i-th layer. In normal dropout, elements in vector $\boldsymbol{p}_i$ have same values p_i, which means that there is one keep rate or (1−dropout rate) for each layer. In the following, we will use $\boldsymbol{p}_i$ and p_i interchangeably depending on the context if there is no special specification. To note that since weight matrix is treated as random variable, we use $\boldsymbol{M}_i \in \mathbb{R}^{D_{i-1} \times D_i}$ to

denote position of non-zero element in Bernoulli distribution for $\boldsymbol{W}_i$. To note that bias $\boldsymbol{b}_i \in \mathbb{R}^{D_i}$ is absorbed into W_i by appending a new row at the end of weight matrix and 1 at the end of each data input vector, which is also called homogeneous coordinate. We also assume that approximate weight distribution is factorized over layer, which yields

$$q_\theta(\boldsymbol{\omega}) = \prod_{i=1}^{L} q_\theta(\boldsymbol{W}_i).$$

To start with the formulation of dropout, we model likelihood of output conditioned on input with softmax scores of neural network in classification case:

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\omega}) &= \sigma((\boldsymbol{h}_1 \odot \boldsymbol{\epsilon}_2)\boldsymbol{M}_2) \\ &= \sigma(\boldsymbol{h}_1(diag(\boldsymbol{\epsilon}_2)\boldsymbol{M}_2)) \\ &= \sigma(\boldsymbol{h}_1\boldsymbol{W}_2) \\ &= \sigma(\phi(\boldsymbol{x}(diag(\boldsymbol{\epsilon}_1)\boldsymbol{M}_1) + \boldsymbol{b}_1)\boldsymbol{W}_2) \\ &= \sigma(\phi(\boldsymbol{x}\boldsymbol{W}_1)\boldsymbol{W}_2) \end{aligned} \tag{2.10}$$

where $\odot$ is Hadamard product(element-wise product), $\sigma(a_j) = \frac{exp(a_j)}{\sum_k exp(a^k)}$ is softmax function, $\phi(\cdot)$ is element-wise non-linear activation function such as rectified unit function.

From the equation above, we have

$$\begin{aligned} &\boldsymbol{W}_i = g(\boldsymbol{M}_i, \boldsymbol{\epsilon}_i) = diag(\boldsymbol{\epsilon}_i)\boldsymbol{M}_i \\ &\text{with } \boldsymbol{\epsilon}_i \sim p(\boldsymbol{\epsilon}_i) = Bernoulli(\boldsymbol{p}_i) \end{aligned}$$

which means weight matrix $\boldsymbol{W}_i$ is a random variable whose probability density function is parameterized by $\boldsymbol{p}_i$ and $\boldsymbol{M}_i$, which are denoted by $\theta = \{(\boldsymbol{M}_i, \boldsymbol{p}_i)_{i=1}^L\}$ in Eq.2.8, where $i = 1, .., L$ denotes i-th layer of the network, and $L = 2$ in this example. The expression of approximate posterior distribution is not obvious, but we can define the its form as

$$\begin{aligned} q_\theta(\boldsymbol{\omega}) &= \prod_{i=1}^{L} q_\theta(\boldsymbol{W}_i) \\ &= \prod_{i=1}^{L} \int q_\theta(\boldsymbol{W}_i|\boldsymbol{\epsilon}_i)p(\boldsymbol{\epsilon}_i)d\boldsymbol{\epsilon}_i \end{aligned} \tag{2.11}$$

with

$$\begin{aligned} q_\theta(\boldsymbol{W}_i|\boldsymbol{\epsilon}) &= \delta(\boldsymbol{W}_i - g(\boldsymbol{M}_i, \boldsymbol{\epsilon}_i)) \\ &= \delta(\boldsymbol{W}_i - diag(\boldsymbol{\epsilon}_i)\boldsymbol{M}_i) \end{aligned} \tag{2.12}$$

As we can see, the approximate posterior distribution over the parameter matrix puts a same Bernoulli distribution over the input dimension of parameter matrix, which is the row dimension in this example. Meanwhile each element of the same row is multiplied with same realization of the random variable but with different non-zero position, which is corresponding to different expectation of each row element and thus induces correlations between row elements. To make this definition more clear, based on the computation of expectation and variance of Bernoulli distribution, we can write down the first and second moment of the approximate distributed random variables in the following:

$$\mathbb{E}_{q_\theta}(\boldsymbol{W}_i) = \boldsymbol{M}_i \odot \boldsymbol{P}_i \tag{2.13}$$

where $\boldsymbol{P}_i = [\boldsymbol{p}_i, ..., \boldsymbol{p}_i] \in \mathbb{R}^{D_{i-1} \times D_i}$.

Covariance matrix of parameter matrix is:

$$\big[Cov_{q_\theta}(vec(\boldsymbol{W}_i))\big]_{jk} = \mathbb{1}\big[l = m\big] m^i_{lq} * m^i_{mn} * p_i * (1 - p_i) \tag{2.14}$$

where

$$j = l * D_{i-1} + q$$

,

$$k = m * D_{i-1} + n$$

, which is the linear mapping between element index before and after matrix vectorization. m^i_{lq} is the element of l-th row and q-th column in matrix $\boldsymbol{M}_i$, which applies to m^i_{mn} as well. $\mathbb{1}\big[i = j\big]$ denotes indicator function, which is equal to 1 only when i is equal to j and otherwise 0. Because $vec(\cdot)$ operation converts matrix $\boldsymbol{W}_i \in \mathbb{R}^{D_{i-1} \times D_i}$ into column vector $vec(\boldsymbol{W}_i) \in \mathbb{R}^{D_{i-1}D_i \times 1}$ by stacking the columns of matrix on top of one another. Then it's easy to see that covariance matrix $Cov_{q_\theta}(vec(\boldsymbol{W}_i)) \in \mathbb{R}^{D_{i-1}D_i \times D_{i-1}D_i}$.

From Eq.2.14, we can see that the entire covariance matrix is consisting of $D_i * D_i$ diagonal sub-matrices whose dimensionality are $D_{i-1} \times D_{i-1}$. When observing covariance matrix of parameter matrix $\boldsymbol{W}_i$ w.r.t. approximate posterior distribution $q_\theta(\boldsymbol{W}_i)$, we know that samples of row vectors in weight matrix are drawn independently and thus covariance between rows are zero. On the other hand, samples for different weights in the same row are drawn at the same time because they are multiplied by the same realization of ϵ_i, from which covariances between weights within the same row are induced. Therefore, by fitting this approximate distribution to the real posterior distribution weights within the same row can be learned. This means that the approximate distribution family have more flexibility to approximate the real posterior distribution when compared with other common approximate posterior distribution family that assumes distribution for each parameter is independent such as fully factorized Gaussian.

Training objective: Up to now, we have only analyzed the approximate distribution of dropout inference. As is mentioned in introduction section of this chapter, we know that we perform optimization of $ELBO$ w.r.t. the approximate distribution parameters, in order to obtain a good approximation to the true posterior distribution over parameters, which we can use in testing to obtain more reliable uncertainty estimation. In the following, we will show that training a neural network with dropout is equivalent to optimizing the $ELBO$ w.r.t. variational parameters.

At first, let's define the training objective of neural network with dropout, which is cross entropy between predictive distribution and target distribution plus L2 regularization:

$$L_{dropout} = \sum_{i=1}^{N} \big[- log(p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\omega})) \big] + \lambda(\sum_{i=1}^{L} ||\boldsymbol{W}_i||^2) \tag{2.15}$$

where N represents the size of entire dataset, λ is L2 regularization coefficient. We want to maximize the likelihood over the entire dataset w.r.t. the model parameter $\boldsymbol{\omega}$, which is known as maximum likelihood estimation. Equipped with L2 regularization and Gaussian prior over parameters, we can obtain a max-a-posterior estimation by minimizing Eq.2.15.

Normally we use gradient descent to tune our parameters in training, which requires first derivative of objective w.r.t. model parameters $\boldsymbol{\omega}$. Since nowadays large size of dataset is ubiquitous, which means N in Eq.2.15 is too large, we could not obtain exact gradient of entire batch with efficient computation. Therefore we use data of mini-batch to estimate gradient, which is so called stochastic gradient descent(SGD). Apart from making computation tractable, noise of gradient estimation in each mini-batch is helpful for optimization procedure to escape the poor local minimum. The expression of gradients required in each iteration of SGD is given in the following:

$$\frac{\partial L_{dropout}}{\partial \boldsymbol{\omega}} = \frac{N}{K} \sum_{i \in S} \big[- \frac{\partial log(p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\omega}))}{\partial \boldsymbol{\omega}} \big] + \lambda \frac{\partial (\sum_{i=1}^{L} ||\boldsymbol{W}_i||^2)}{\partial \boldsymbol{\omega}} \tag{2.16}$$

where S is one random subset of entire dataset, and $|S| = K$.

On the other hand, let's have a look at $ELBO$ in Eq.2.6, if we want to maximize $ELBO$, which is equivalent to minimize negative $ELBO$ w.r.t. the variational parameter θ, with SGD. The gradients are computed with the following expression:

$$\frac{\partial(-ELBO)}{\partial \theta} = \frac{N}{K} \sum_{i \in S} \big[- \frac{\partial \mathbb{E}_{q_\theta(\boldsymbol{\omega})} \big[log(p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\omega})) \big]}{\partial \theta} \big] + \frac{\partial KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))}{\partial \theta} \tag{2.17}$$

In order to calculate two terms in Eq.2.17, we need to introduce one technique called reparameterization trick[KW13] for the first term and one condition called KL condition[Gal]

for the second term in the following.

Re-parameterization trick: when we take a close look at the first term in Eq.2.17, we know that we need to compute the gradients of expectation w.r.t. the parameters of distribution to which this expectation is subject. That means, we need to estimate the gradients of those parameters because our objective is generated from these parameters randomly. Fortunately, there are different approaches to estimate the gradients of this kind of parameters such as score function or REINFORCE estimator[Wil92], re-parameterization trick [KW13] and so on. As is stated in [KW13], re-parameterization trick has lower variance than score function estimator and is also unbiased. Let's have a quick recap of this technique and see it's already a built-in part in neural network with dropout.

To identify the problem, we can write a more general form of calculus we want to solve in the following:

$$I(\theta) = \frac{\partial}{\partial\theta}\mathbb{E}_{p_\theta(x)}\big[f(x)\big] = \frac{\partial}{\partial\theta}\int f(x)p_\theta(x)dx \tag{2.18}$$

Assume that $p_\theta(x)$ can be re-parameterized as $p(\epsilon)$ which is a parameter-free distribution such that random variable x can be generated from a deterministic differentiable function with θ and ϵ as arguments, that is

$$x = g(\theta, \epsilon) \text{ with } \epsilon \sim p(\epsilon)$$

Then we can derive the estimator of the gradients w.r.t. distribution parameters θ with $p(x|\epsilon) = \delta(x - g(\theta, \epsilon))$:

$$\begin{aligned}
I'(\theta) &= \frac{\partial}{\partial\theta}\int f(x)p_\theta(x)dx \\
&= \frac{\partial}{\partial\theta}\int f(x)(\int p_\theta(x|\epsilon)p(\epsilon)d\epsilon)dx \\
&= \frac{\partial}{\partial\theta}\int (\int f(x)p_\theta(x|\epsilon)dx)p(\epsilon)d\epsilon \\
&= \frac{\partial}{\partial\theta}\int (\int f(x)\delta(x - g(\theta,\epsilon))dx)p(\epsilon)d\epsilon \\
&= \frac{\partial}{\partial\theta}\int f(g(\epsilon,\theta))p(\epsilon)d\epsilon \\
&= \int \frac{\partial}{\partial\theta} f(g(\epsilon,\theta))p(\epsilon)d\epsilon \\
&= \int f'(g(\epsilon,\theta))\frac{\partial}{\partial\theta}g(\theta,\epsilon)p(\epsilon)d\epsilon \\
&= \mathbb{E}_{p(\epsilon)}\big[f'(g(\epsilon,\theta))\frac{\partial}{\partial\theta}g(\theta,\epsilon)\big]
\end{aligned} \tag{2.19}$$

From practical point of view, the expectation of last line in expression Eq.2.19 can be approximated with Monte Carlo integration. In dropout inference, we know that if we fix the dropout rate to $1 - p_i$ in Eq.2.12 during training. Then $\boldsymbol{\epsilon}_i$ in Eq.2.12 is a parameter free random variable and $g(\cdot)$ is a differentiable function w.r.t. its input argument $\boldsymbol{M}_i$. Now the variational parameter θ only contains $\{(\boldsymbol{M}_i)_{i=1}^L\}$, which are exactly the weights to be learned in training neural network with dropout. As a consequence, if we estimate the gradient of $ELBO$ w.r.t. $\{(\boldsymbol{M}_i)_{i=1}^L\}$, which is the first term in Eq.2.17, it's equivalent to calculate the gradient of dropout loss w.r.t. model parameters $\boldsymbol{\omega}$ which is the first term in Eq.2.16.

KL condition: The second term in Eq.2.17 is proved to be equivalent to the second term in Eq.2.16 for a large enough number of hidden units when we specify the model prior to be a product of independent Gaussian distributions over each weight with prior length scale l, that is:

$$p(\boldsymbol{\omega}) = \prod_{i=1}^{L}(p(\boldsymbol{W}_i))$$

with

$$p(vec(\boldsymbol{W}_i)) = \mathcal{N}(0, l^{-2}\boldsymbol{I}_{D_{i-1}D_i})$$

To establish this condition, we need to make a approximation of the approximate posterior distribution $q_\theta(\boldsymbol{\omega})$ in Eq.2.11, where we approximate $q_\theta(\boldsymbol{W}_i|\boldsymbol{\epsilon}_i)$ in Eq.2.12 as a narrow Gaussian with a very small standard deviation. As we know, $q_\theta(\boldsymbol{W}_i)$ factorizes over each row of the weight matrix. Then that means $q_\theta(\boldsymbol{\omega})$ is a mixture of two Gaussians with small standard deviations, and one component fixed at zero:

$$\begin{aligned} q_\theta(\boldsymbol{\omega}) &= \prod_{i=1}^{L} q_\theta(\boldsymbol{W}_i) \\ &= \prod_{i=1}^{L}\prod_{j=1}^{D_{i-1}} q_\theta(\mathbf{w}_{i,j}) \\ &= \prod_{i=1}^{L}\prod_{j=1}^{D_{i-1}} \left[p_i\mathcal{N}(\mathbf{m}_{i,j}, \sigma^2\boldsymbol{I}_{D_i}) + (1-p_i)\mathcal{N}(\mathbf{0}, \sigma^2\boldsymbol{I}_{D_i})\right] \end{aligned} \tag{2.20}$$

where $\mathbf{w}_{i,j} \in \mathbb{R}^{D_i}$ is the j-th row of weight matrix $\boldsymbol{W}_i$ and p_i is the parameter of Bernoulli distributed random variable of i-th layer. With this, the KL divergence between approximate posterior and prior is KL divergence between mixture of Gaussian and a single Gaussian. Reader who has interest can refer to derivation of KL divergence between mixture of

Gaussian and single Gaussian in [Gal]. Then we have KL condition in the following:

$$KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \approx \sum_{i=1}^{L}\sum_{j=1}^{D_{i-1}}\big[\frac{p_i}{2}(l^2\boldsymbol{m}_{i,j}^T\boldsymbol{m}_{i,j} + D_i(\sigma^2 - \log(\sigma^2) - 2\log l - 1) - \mathcal{H}(\boldsymbol{p}_i)\big] \tag{2.21}$$

with

$$\mathcal{H}(\boldsymbol{p}_i) = D_{i-1}(-p_i\log p_i - (1-p_i)\log(1-p_i))$$

for large enough D_i. If we **fix dropout rate** in training and compute the gradients of KL divergence w.r.t. variational parameters $\theta = \{(\boldsymbol{M}_i)_{i=1}^L\}$, which is now exactly the same as the model parameters $\boldsymbol{\omega} = \{(\boldsymbol{W}_i)_{i=1}^L\}$. Then the term entropy of dropout rate can be ignored. We can see that, if we choose $\lambda = \frac{l^2 p_i}{2}$, then it's equivalent to the gradients of regularization term in dropout loss function(cf. Eq.2.16):

$$\frac{\partial KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))}{\partial\theta} \approx \frac{\partial\sum_{i=1}^{L}\lambda||\boldsymbol{M}_i||^2}{\partial\theta} \tag{2.22}$$

With aforementioned re-parameterization trick and KL condition, we know that the computation of gradients of objective function w.r.t. model parameters in Eq.2.16 is equivalent to that of negative $ELBO$ w.r.t. variational parameter in Eq.2.17. That means, if we train a neural network with dropout, it's equivalent to train a Bayesian neural network with approximate posterior distribution over model parameters defined in Eq.2.11.

Marginalization in testing: After we have obtain the parameters θ of approximate posterior distribution over model parameters $q_\theta(\boldsymbol{\omega})$, we can marginalize all possible parameters over approximate posterior distribution to get final predictive distribution of output, similar to Eq.2.4 but now with approximate posterior distribution over model parameter, which has an expression in Eq.2.23. Because the integration is hard to evaluate analytically. In practice, we always use Monte Carlo integration:

$$\begin{aligned} p(\boldsymbol{y}^\star|\boldsymbol{x}^\star,\mathcal{D}) &= \int p(\boldsymbol{y}^\star|\boldsymbol{x}^\star,\boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D})d\boldsymbol{\omega} \\ &\approx \int p(\boldsymbol{y}^\star|\boldsymbol{x}^\star,\boldsymbol{\omega})q_\theta(\boldsymbol{\omega})d\boldsymbol{\omega} \\ &\approx \frac{1}{K}\sum_{i=1}^{K} p(\boldsymbol{y}^\star|\boldsymbol{x}^\star,\hat{\boldsymbol{\omega}}_{\boldsymbol{i}}) \end{aligned} \tag{2.23}$$

where $\boldsymbol{\omega} \sim q_\theta(\boldsymbol{\omega})$ and $\hat{\boldsymbol{\omega}}_{\boldsymbol{i}}$ is one of K realizations of $\boldsymbol{\omega}$, which is equivalent to turning on dropout in testing time.

Concrete dropout

Based on the aforementioned description of dropout inference, we can see that if we fix dropout rate in training, then parameters of approximate distribution θ only contains $\{(\boldsymbol{M}_i)_{i=1}^L\}$, which is equivalent to $\boldsymbol{\omega} = \{(\boldsymbol{M}_i)_{i=1}^L\}$ in neural network with dropout. Therefore optimizing any neural network with dropout is equivalent to a form of approximate inference in a probabilistic interpretation of the model. This means that the optimal weights found through the optimization of a neural network with dropout are the same as the optimal variational parameters in a Bayesian neural network with the same structure.

However, fixing dropout rate in train is not a trivial task for several reasons. Firstly, as is shown in [SHK+14], different dropout rates have different impacts on model capacity and thus model performance. To choose an optimal dropout rate manually requires repeating tedious experiments and thus higher time consumptions and computation effort. Secondly, if we want our model not only to achieve satisfied performance but also to possess reliable uncertainty estimation. The dropout rate should matter a lot because it can influence the flexibility of approximate distribution family from the perspective of Bayesian interpretation.

Accordingly, one direct counter measure is to learn dropout rate from the data [GHK17]. One major difficulty to learn dropout rate from the data in gradient-based optimization is that it's not trivial to estimate gradients of expectation w.r.t. parameters of a discrete distribution. Before when the approximate distribution is continuous, we estimate this gradient with help of re-parameterization trick. As introduced in the last section, re-parameterization trick requires re-parameterizing samples of the distribution into a differential function with variational parameters and a parameter-free random variable as input arguments. For continuous distribution, this function can be found easily if they have tractable inverse cumulative density function or functional form like Gaussian[KW13]. For most of discrete distributions such as Bernoulli distribution or categorical one, they lack useful re-parameterizations due to the discontinuous nature of discrete states [MMT16].

Confronted with this issue, [JGP16] and [MMT16] come up with one approach where "max" operation in Gumbel-Max trick is replaced by "softmax" function, which can yields a practical re-parameterization for discrete random variable. With this, gradients w.r.t. parameters of discrete approximate distribution can be computed and used in gradient-based optimization. In this subsection, a quick recap of this approach is given in the following.

Re-parameterization for Bernoulli distribution: Firstly, Gumbel-Max trick[MTM14] is introduced in figure 2.1.2, which is used for drawing samples from a discrete distribution which is parameterized by set of unnormalized probability $\{\alpha_i\}_{i=1}^n$ via inverse cumulative distribution function of Gumbel distribution, where $\alpha_i \in \mathbb{R}_{>0}$ and n denotes the number of class. Assuming that we use one-hot encoding vector for the class representation, that is $\boldsymbol{d} \in \{0,1\}^n$ and $\sum_{i=1}^n d_i = 1$. The Gumbel-Max trick proceeds

as follows(cf. figure 2.1.2):

- sample $G_i \sim \text{Gumbel}(0,1) = -\log(-\log(\text{Uniform}(0,1)))$, for $i = 1,..,n$
- compute $x_i = \log(\alpha_i) + G_i$, for $i = 1,..,n$
- set $d_k = 1$, where $k = argmax_i\{x_i\}_{i=1,..n}$, and $d_i = 0$ for $i \neq k$

Then we obtain one sample from this discrete distribution and the probability for specific class.

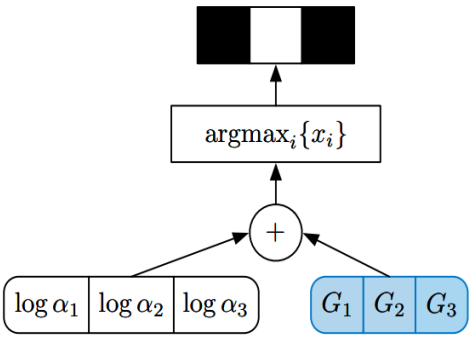


Figure 2.4: Example of Gumbel-max trick for drawing samples from a discrete distribution whose random variable has 3 states and $\{\alpha_i\}_{i=1,2,3}$ as class parameters representing the possibility of occurrence of that class. $\{G_i\}_{i=1,2,3}$ are i.i.d Gumbel$(0,1)$ [MMT16].

As observed in Gumbel-Max trick, the sampling step is re-paramterized by one function with distribution parameters and parameter-free random variable as arguments. This function is component-wise addition of two input arguments followed by $argmax$ operation. However, $argmax$ operation is not differential w.r.t. distribution parameter α_i.

With a little abuse of notation, we denote temperature by λ here. Fortunately, the $argmax$ operation can be approximated by a continuous function, $softmax$ function with λ as temperature parameter, which is differential w.r.t. distribution parameters. With this replacement, we can obtain a continuous approximation to one-hot encoding vector $\boldsymbol{d}$ on the simplex $\triangle^{n-1} = \{\boldsymbol{x} \in \mathbb{R}^n | x_k \in [0,1], \sum_{i=1}^n x_i = 1\}$:

$$x_k = \frac{\exp((\log\alpha_k + G_k)/\lambda)}{\sum_{i=1}^n \exp((\log\alpha_i + G_k)/\lambda)} \tag{2.24}$$

The sampling steps are similar to Gumbel-Max trick, but with smoothed $softmax$ operation instead of $argmax$(cf. figure 2.5). When $\lambda \to 0$, the $softmax$ function is recovered to $argmax$ operation, when $\lambda \to \infty$, this operation generates uniform vector instead of one-hot encoding vector. In practice, we set temperature as 0.1.

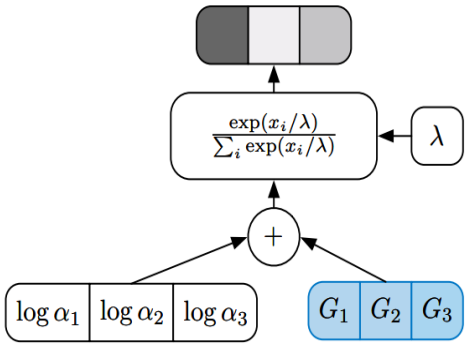


Figure 2.5: Example of continuous approximation to Gumbel-max trick for drawing samples from a discrete distribution whose random variable has 3 states and $\{\alpha_i\}_{i=1,2,3}$ as class parameters representing the possibility of occurrence of that class. $\{G_i\}_{i=1,2,3}$ are i.i.d Gumbel$(0, 1)$[MMT16].

When it comes to Bernoulli discrete distribution in our case, it becomes simpler because there are only two states in this distribution and samples live in two dimensional simplex. On the other hand, the difference of two Gumbels distributed random variable is similar to a logistic distributed random variable. With the probability of state 1 which can be expressed by:

$$\begin{aligned}\mathbb{P}(d_1 = 1) &= \mathbb{P}(\log\alpha_1 + G_1 > \log\alpha_2 + G_2) \\ &= \mathbb{P}(\log\alpha_1 - \log\alpha_2 + G_1 - G_2 > 0)\end{aligned}$$

where $G_1 - G_2 = \log(\text{Uniform}(0, 1)) - \log(1 - \text{Uniform}(0, 1))$, which is the inverse cumulative density function of Logistic(0,1). Then we can infer the re-parameterization of continuous approximation of sample $x \in [0, 1]$ from Bernoulli distributed random variable with p as parameter as follows:

$$x = \text{sigmoid}\big(\frac{1}{\lambda}(\log p - \log(1 - p) + \text{u} - \log(1 - u))\big) \tag{2.25}$$

where $u \sim \text{Uniform}(0, 1)$.

In order to make explanation more clear and validate the continuous approximate re-parameterization, there is one plot (figure 2.6) showing the relationship between average values of 100 samples drawn from this continuous approximation of Bernoulli distribution with different probability parameter from 0 to 1 with step of 0.1. We can see that the empirical average of samples drawn from Eq.2.25 is close to expectation of Bernoulli distribution of corresponding parameter. Additionally, we show one single sample in the figure. As we can see, most of samples are lying on the boundary of range $[0, 1]$ and only few of them lie in the interior.

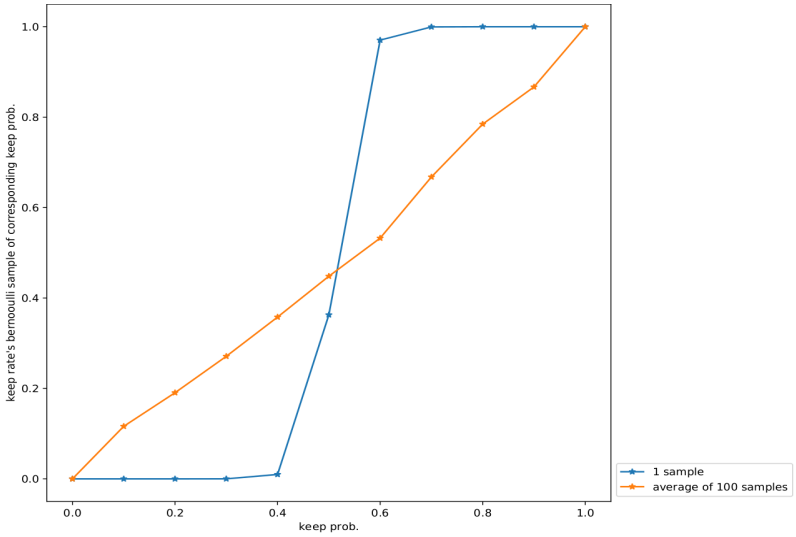


Figure 2.6: One sample and average value of 100 samples drawn from continuous approximation of Bernoulli distribution with parameter $p = [0.1, 0.2, ..., 1.0]$ and temperature $\lambda = 0.1$.

Dropout regularization: Because keep rate of dropout is being optimized, the parameters of approximate distribution θ contains both $\{(\boldsymbol{M}_i)_{i=1}^L\}$ and $\{(\boldsymbol{p}_i)_{i=1}^L\}$. In order to compute gradients of ELBO in Eq.2.17, we employ categorical re-parameterization to estimate the gradients w.r.t. Bernoulli distribution parameter $\boldsymbol{p}_i$ in the first term, for the second term, we could not ignore the term with $\boldsymbol{p}_i$ in KL condition Eq.2.21, which is the entropy term. Consequently, unlike Eq.2.22, the KL divergence term should be:

$$\begin{aligned}\frac{\partial KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))}{\partial\theta} &\approx \frac{\partial \sum_{i=1}^L \lambda||\boldsymbol{M}_i||^2 - \beta\mathcal{H}(\boldsymbol{p}_i)}{\partial\theta} \\ &= \frac{\partial}{\partial\theta}\Big(\sum_{i=1}^L \lambda||\boldsymbol{M}_i||^2 - \beta D_{i-1}(-p_i \log p_i - (1-p_i)\log(1-p_i))\Big)\end{aligned} \tag{2.26}$$

where λ and β are coefficients for L2 regularization and dropout regularization, respectively. They are hyper-parameters which can be tuned based on performance on validation set. From the equation above, we can see that this term maximize the entropy of Bernoulli distribution, which means this term pushes $\boldsymbol{p}_i$ to 0.5. The coefficient of the dropout regularisation term means that large models will push the dropout probability towards 0.5 much more than smaller models, but as the amount of data N increases the dropout probability will be pushed towards 1 because of the expected log likelihood in the first term. One of reasons behind could be pushing $\boldsymbol{p}_i$ to 0.5 would decrease the capacity of the model which will decrease the expected log-likelihood.

2.1.3 Laplace approximation

Introduction

In this section, another method to approximate the real posterior distribution over weights is introduced, which is called Laplace approximation[Bis06]. It requires only point estimation of model parameters, thus there is no need to re-train existing model which has already obtained an point estimate. This characteristic makes this approach more appealing because it's not required to modify the model and thus can be applied to any existing models. The idea behind that is to put a Gaussian approximation on the maximum a posterior point estimate of the model parameters, which is one mode of posterior distribution. The reasons why we consider this method are as following:

- it has easy compatibility to existing network. To perform Laplace approximation, we only need one point estimation of model parameters which is already available for most of architectures. That also means, we do not need modify the training phase and we can have approximation of the true posterior for all trained models.
- it can capture relationship between model parameters. As mentioned in the first chapter, most of approximation approaches make assumption that parameters are independent to each other for simplicity, scalability as well as computation effort. That could be a quite strong restrictions on the approximation leading to bad performance.

In the following, we introduce the basic idea of Laplace approximation and further introduce its scalable version for deep neural network based on Kronecker factor approximation.

Assume we have a point estimate of model parameters via maximum a posterior estimation:

$$\begin{aligned} \boldsymbol{\omega}^{\star} &= \underset{\boldsymbol{\omega}}{argmax}\{p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})\} \\ &= \underset{\boldsymbol{\omega}}{argmax}\{\frac{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\boldsymbol{Y}|\boldsymbol{X})}\} \\ &= \underset{\boldsymbol{\omega}}{argmax}\{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})p(\boldsymbol{\omega})\} \end{aligned} \tag{2.27}$$

After taking a second order Taylor expansion of the logarithm of of posterior distribution $p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})$ around its mode $\boldsymbol{\omega}^{\star}$, assuming that the prior of weights is uniform, then we

have:

$$\begin{aligned} \log p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X}) \approx \log p(\boldsymbol{\omega}^\star|\boldsymbol{Y},\boldsymbol{X}) + \\ (\boldsymbol{\omega}-\boldsymbol{\omega}^\star)^T \frac{\partial \log p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})}{\partial \boldsymbol{\omega}} + \\ \frac{1}{2}(\boldsymbol{\omega}-\boldsymbol{\omega}^\star)^T \frac{\partial^2 \log p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})}{\partial \boldsymbol{\omega}^2}(\boldsymbol{\omega}-\boldsymbol{\omega}^\star) \\ = \log p(\boldsymbol{\omega}^\star|\boldsymbol{Y},\boldsymbol{X}) - \frac{1}{2}(\boldsymbol{\omega}-\boldsymbol{\omega}^\star)^T \boldsymbol{H}(\boldsymbol{\omega}-\boldsymbol{\omega}^\star) \end{aligned} \tag{2.28}$$

where

$$\begin{aligned} \boldsymbol{H} &= -\frac{\partial^2 \log p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})}{\partial \boldsymbol{\omega}^2} \\ &= -\frac{\partial^2 \log\big(p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})\big)}{\partial \boldsymbol{\omega}^2} - \frac{\partial^2 p(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}^2} \end{aligned}$$

which is negative Hessian of the log posterior. The first order term in Eq.2.28 vanishes because we expand the function around a local maximum $\boldsymbol{\omega}^\star$, where the first derivative is zero. If we exponentiate this equation, we can get the following Gaussian-like functional form:

$$p(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X}) \propto p(\boldsymbol{\omega}^\star|\boldsymbol{Y},\boldsymbol{X}) \exp\{-\frac{1}{2}(\boldsymbol{\omega}-\boldsymbol{\omega}^\star)^T \boldsymbol{H}(\boldsymbol{\omega}-\boldsymbol{\omega}^\star)\} \tag{2.29}$$

which means $\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\omega}^\star, \boldsymbol{H}^{-1})$. Therefore, we can obtain a Gaussian approximate distribution with local maximum $\boldsymbol{\omega}^\star$ as mean and inverse of negative Hessian as covariance matrix.

If we use Gaussian prior for weights. Then it's easy to know that the second order derivative of prior distribution term $\frac{\partial^2 p(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}^2}$ is a identity matrix multiplied by regularization coefficient. And the non-trivial part is the first term, second derivatives of log likelihood. To make the explanation uncluttered, we define the negative Hessian of log likelihood with $\hat{\boldsymbol{H}}$ instead:

$$\hat{\boldsymbol{H}} = -\frac{\partial^2 \log\big(p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})\big)}{\partial \boldsymbol{\omega}^2}$$

However, we should note that for a large training set, it's always infeasible to analyze the gradients or Hessian exactly. To resolve this, normally we estimate the expectation of gradients or Hessian in each mini-batch. That means we need to estimate Hessian with empirical average Hessian computed in mini-batches:

$$\hat{\boldsymbol{H}} \approx N \mathbb{E}_{p(\boldsymbol{Y},\boldsymbol{X})}[\hat{\boldsymbol{H}}]$$

where N is size of training samples, and

$$\mathbb{E}_{p(\boldsymbol{Y},\boldsymbol{X})}[\hat{\boldsymbol{H}}] \approx -\frac{1}{K}\sum_{k}\big[\frac{1}{M}\sum_{i}\frac{\partial^2 \log\big(p(y_{ik}|\boldsymbol{x}_{ik},\boldsymbol{\omega})\big)}{\partial \boldsymbol{\omega}^2}\big] \tag{2.30}$$

where K is total number of mini-batch, and $(y_{ik}, \boldsymbol{x}_{ik})$ is the i-th training data sample in k-th mini-batch.

Fisher information matrix is an approximation to the expected negative Hessian of exponential family log probability:

$$\hat{\boldsymbol{F}} = \mathbb{E}_{p(\boldsymbol{X},\boldsymbol{Y})}[\hat{\boldsymbol{H}}]$$

Therefore we can use Fisher information matrix as replacement of expected Hessian for the log likelihood term. The derivation of equivalence between expected Hessian and Fisher matrix is straightforward. We define Fisher matrix $\boldsymbol{F}$ in the following:

$$\begin{aligned}\hat{\boldsymbol{F}} &= \mathbb{E}_{p(\boldsymbol{Y},\boldsymbol{X})}\big[\frac{\partial}{\partial \boldsymbol{\omega}}\log p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})\frac{\partial}{\partial \boldsymbol{\omega}}\log p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\omega})^T\big] \\ &\approx \frac{1}{K}\sum_{k}\big[\frac{1}{M}\sum_{i}\big(\frac{\partial}{\partial \boldsymbol{\omega}}\log p(y_{ik}|\boldsymbol{x}_{ik},\boldsymbol{\omega})\frac{\partial}{\partial \boldsymbol{\omega}}\log p(y_{ik}|\boldsymbol{x}_{ik},\boldsymbol{\omega})^T\big)\big]\end{aligned} \tag{2.31}$$

We include the derivation of equivalence between negative expected Hessian and Fisher matrix for case that parameter is scalar in appendix A.1, which can be extended to case for vector easily. Therefore we can obtain

$$\hat{\boldsymbol{H}} \approx N\hat{\boldsymbol{F}} \tag{2.32}$$

and

$$\boldsymbol{H} \approx N\hat{\boldsymbol{F}} + \tau\boldsymbol{I} \tag{2.33}$$

Actually, computing outer product of gradients requires less computation. But to compute the empirical average of outer product we need to save values with amount quadratic to the number of model parameters inducing large storage overhead. More than that, the computation complexity of inversing matrix is cubic of dimensionality of this matrix which is the number of model parameters. This is also infeasible because the number of parameters in deep neural nets can exceed million easily. Therefore an efficient approximation for Fisher matrix is required.

Scalable Laplace approximation for neural network

In order to mitigate the computational burden in Laplace approximation above, Kronecker-factored approximation curve(KFAC) in [MG15] can be used to approximate the Fisher information matrix $\boldsymbol{F}$, with which Laplace approximation can be performed for neural network[RBB18]. This approximation is derived by approximating the large blocks matrix of Fisher(corresponding to each layer) by Kronecker product of two much smaller matrices. While several times more expensive to compute this, the issues of storage and inversion in Laplace approximation can be mitigated. Additionally, because we need this Fisher information matrix after training, this little expensive computation is required to perform once.

With the notations used in previous subsection, where dropout variational inference is introduced, we will derive the approximation in the following. For clarity, we repeat the definitions of existing notations and new defined ones here. A neural network transforms its input $a_0 = \boldsymbol{x}$ to an output $a_L = p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\omega}) = \sigma(\boldsymbol{x}, \boldsymbol{\omega})$ through a series of L layers. The units each receive as input a weighted sum of outputs of units from the previous layer and compute their output via a non-linear activation function $\phi(\cdot)$. We denote by $\boldsymbol{s}_i$ the vector of these weighted sums for i-th layer, and by $\boldsymbol{a}_i$ the vector output of non-linear activation function. Computation performed for each layer $i \in 1, ..., L$ is given as follows:

$$\boldsymbol{s}_i = \boldsymbol{a}_i \boldsymbol{W}_i \text{ and } \boldsymbol{a}_i = \phi(\boldsymbol{s}_i) \tag{2.34}$$

We define $\tilde{\boldsymbol{w}}$ to be the vector consisting of all of the network's parameters concatenated together, i.e. $\tilde{\boldsymbol{w}} = \left[vec(\boldsymbol{W}_1)^T \, vec(\boldsymbol{W}_2)^T \, ... \, vec(\boldsymbol{W}_L)^T\right]^T$, where vec is the operator which vectorizes matrices by stacking their columns together. For simplicity we first define the following notations:

$$\partial v = -\frac{\partial \log p(\boldsymbol{y}, |\boldsymbol{x}\boldsymbol{\omega})}{\partial v} \text{ and } \boldsymbol{g}_i = \partial \boldsymbol{s}_i$$

The first derivatives of objective w.r.t. $\tilde{\boldsymbol{w}}$ is defined as follows:

$$\partial \tilde{\boldsymbol{w}} = \left[vec(\partial \boldsymbol{W}_1)^T \, vec(\partial \boldsymbol{W}_2)^T \, ... \, vec(\partial \boldsymbol{W}_L)^T\right]^T \tag{2.35}$$

From Eq.2.31, we know $\hat{\boldsymbol{F}} = \mathbb{E}[\partial \hat{\boldsymbol{w}} \partial \hat{\boldsymbol{w}}^T]$ which can viewed as L by L block matrix, with the (i, j)-th block $\hat{\boldsymbol{F}}_{i,j}$ given by $\hat{\boldsymbol{F}}_{i,j} = \mathbb{E}[vec(\partial \boldsymbol{W}_i) vec(\partial \boldsymbol{W}_j)^T]$.

Noting that $\partial \boldsymbol{W}_i = \boldsymbol{a}_{i-1} \boldsymbol{g}_i^T$ and $vec(\boldsymbol{u}\boldsymbol{v}^T) = \boldsymbol{v} \otimes \boldsymbol{u}$, then we have

$$vec(\partial \boldsymbol{W}_i) = vec(\boldsymbol{a}_{i-1} \boldsymbol{g}_i^T) = \boldsymbol{g}_i \otimes \boldsymbol{a}_{i-1}$$

Then we can rewrite

$$\begin{aligned}
\hat{\boldsymbol{F}}_{i,j} &= \mathbb{E}[vec(\partial \boldsymbol{W}_i) vec(\partial \boldsymbol{W}_j)^T] \\
&= \mathbb{E}[(\boldsymbol{g}_i \otimes \boldsymbol{a}_{i-1})(\boldsymbol{g}_j \otimes \boldsymbol{a}_{j-1})^T] \\
&= \mathbb{E}[(\boldsymbol{g}_i \otimes \boldsymbol{a}_{i-1})(\boldsymbol{g}_j^T \otimes \boldsymbol{a}_{j-1}^T)] \\
&= \mathbb{E}[(\boldsymbol{g}_i \boldsymbol{g}_j^T) \otimes (\boldsymbol{a}_{i-1} \boldsymbol{a}_{j-1}^T)]
\end{aligned}$$

Here we need to define $\tilde{\boldsymbol{F}}_{i,j}$ as approximation to $\hat{\boldsymbol{F}}_{i,j}$ in the following:

$$\begin{aligned}\hat{\boldsymbol{F}}_{i,j} &\approx \tilde{\boldsymbol{F}}_{i,j} \\ &= \mathbb{E}[\boldsymbol{g}_i \boldsymbol{g}_j^T] \otimes \mathbb{E}[\boldsymbol{a}_{i-1} \boldsymbol{a}_{j-1}^T] \\ &= \mathbb{E}[\boldsymbol{G}_{i,j}] \otimes \mathbb{E}[\boldsymbol{A}_{i,j}]\end{aligned} \tag{2.36}$$

where $\boldsymbol{G}_{i,j} = \boldsymbol{g}_i \boldsymbol{g}_j^T$ and $\boldsymbol{A}_{i,j} = \boldsymbol{a}_{i-1} \boldsymbol{a}_{j-1}^T$. The expectation of a Kronecker product is, in general, not equal to the Kronecker product of expectations, and so this is indeed a major approximation to make, and one which likely won't become exact under any realistic set of assumptions, or as a limiting case in some kind of asymptotic analysis. Nevertheless, it seems to be fairly accurate in practice, and is able to successfully capture the "coarse structure" of the Fisher information Matrix. In this work, we only consider the case that layers are independent to others. Therefore the Fisher information matrix of the entire network is a diagonal block matrix, which means $\tilde{\boldsymbol{F}}_{i,j}$ is non-zero only for $i = j$. For i-th layer, we can compute the Fisher information matrix with Eq.2.36. The two factors in Kronecker are outer product of gradients of pre-activation $\boldsymbol{G}_i = \boldsymbol{g}_i \boldsymbol{g}_j^T$ and outer product of activation from previous layer $\boldsymbol{A}_i = \boldsymbol{a}_{i-1} \boldsymbol{a}_{i-1}^T$, respectively:

$$\tilde{\boldsymbol{F}}_i = \boldsymbol{G}_i \otimes \boldsymbol{A}_i$$

If we treat them as covariances of resulting Gaussian. That means each Gaussian has a Kronecker factored covariance, corresponding to a matrix normal distribution[GN99], which considers the two Kronecker factors of the covariance to be the covariances of the rows and columns of a matrix. The two factors are much smaller than the full covariance and allow for significantly more efficient inversion and sampling. The final posterior of Laplace approximation for i-th layer is:

$$\boldsymbol{W}_i \sim \mathcal{MN}(\boldsymbol{W}_i^\star, \mathbb{E}[\boldsymbol{A}_i]^{-1}, \mathbb{E}[\boldsymbol{G}_i]^{-1}) \tag{2.37}$$

If we consider Gaussian prior and scale of Fisher information matrix(cf. Eq.2.33), then the resulting posterior can be written in the following form:

$$\boldsymbol{W}_i \sim \mathcal{MN}(\boldsymbol{W}_i^\star, (\sqrt{N}\mathbb{E}[\boldsymbol{A}_i] + \sqrt{\tau}\boldsymbol{I})^{-1}, (\sqrt{N}\mathbb{E}[\boldsymbol{G}_i] + \sqrt{\tau}\boldsymbol{I})^{-1}) \tag{2.38}$$

where N is the size of training set and τ is the standard deviation of Gaussian prior. However, we find that in practice, to treat them as hyper-parameters and tune them w.r.t. the predictive performance on a validation set can yield better performance. The possible reasons for this are two fold. Firstly, nowadays the scale of dataset is really large which can exceed tens of thousand easily. The large size of data could be probably redundant and if we take the redundant data into account and thus increase N. This will lead to unreasonably high precision for Gaussian and underestimate the uncertainty significantly. Second, Laplace approximation requires positive definite Hessian and τ always serves as damp factors to fulfill this condition, therefore it needs to be chosen and tuned carefully.

2.2 Conditional random field

Graphical models allow us to encode relationships between multiple variables using a concise, well-defined language. Because it's hard to solve the problems deterministically in real world. *Probabilistic graphical model*(PGM)[KFB09] can assist in this case by encoding a joint or conditional probability distribution with graph, which means that PGM can encode the relationships between multiple random variables.

The various types of PGM differ by the graph structure and the conditional independence assumptions. There are mainly two types, directed and undirected graph. Directed graphical model which is also called Bayesian network, specifies the family of distributions by means of directed acyclic graph and its joint distribution is factorized over each node which is a random variable with probability distribution conditioned on its parents. On the other hand, undirected graphical model, also known as Markov random field has undirected acyclic or cyclic graph. Its joint distribution is factorized over cliques, which is a fully-connected subgraph within the graph. This means, the marginal probability distribution in undirected graphical model is not normalized over nodes, but over cliques.

In a object recognition task, the input is set of images and output is set of predicted labels. Because the dimensionality of input is always large so that it's difficult to model it directly. Nevertheless, we want to exploit the contextual relationship between different objects. To this end, it's better to just model the relationship between output random variables, which obey the conditional distribution of output given input. *Conditional random field*(CRF), first proposed by [LMP01], is a suitable model for this purpose. In this chapter, a concise introduction of CRF is given, including definition, learning and inference in the following subsection.

2.2.1 Definition

CRF can model the conditional distribution of a set of output random variables $\boldsymbol{y}$ given a set of inputs $\boldsymbol{x}$, that is $p(\boldsymbol{y}|\boldsymbol{x})$. In the scope of scene object recognition problem, assume we have multiple objects in one scene denoted by $\boldsymbol{x} = [x_1, ..., x_n]$ and their labels $\boldsymbol{y} = [y_1, ..., y_n]$. We want to learn $p(\boldsymbol{y}|\boldsymbol{x})$ with CRF and make predictions for unseen data.

To formulate the problem clearly, we denote the set of output random variables by $\boldsymbol{y}$ and overall output domain by $\mathcal{Y}$. CRF for one scene $\boldsymbol{x}$ is defined by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where a set of nodes $\mathcal{V}$ representing the random variables in $\boldsymbol{y}$, and a number of undirected edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ representing the dependencies between random variables. The output domain of CRF is the product of individual variable domains $\mathcal{Y}_i$ so that we have $\mathcal{Y} = \underset{i \in V}{\times} \mathcal{Y}_i$, where $\mathcal{Y}_i = \mathcal{L}$, which is a set of all possible labels. With these notations, we can define the probability distribution of $\boldsymbol{y}$ conditioned $\boldsymbol{x}$ with model parameter $\boldsymbol{\theta}$ as

$$p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{x}, \boldsymbol{\theta})} \prod_{c \in \mathcal{C}} \psi_c(y_c, x_c, \boldsymbol{\theta}) \tag{2.39}$$

where $\mathcal{C}$ is the set of cliques of graph $\mathcal{G}$, and $Z(\cdot)$ is normalizer to make sure $\sum_{\boldsymbol{y}\in\mathcal{Y}} p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) = 1$, which is also called partition function. $\boldsymbol{\theta}$ stands for the parameters of CRF to be tuned during training phase. $\psi_c(\cdot)$ is called potential function or factor which encodes the contribution of clique c to the total density. The order of number of random variables in clique c can be arbitrary number. However, for computational reason, we only consider factor with number of random variables up to two in normal case. The first order potential is called *unary* potential which means that the clique of this kind of potential contains only one random variable, that is $\psi_i(y_i, x_i, \boldsymbol{\theta})$. Unary potential can be treated as the likelihood that a label is assigned to this random variable independently to others. The second order potential is called pairwise potential $\psi_{i,j}(y_i, y_j, x_i, x_j, \boldsymbol{\theta})$ characterized by edges in the graph. This kind of potential states the compatibility of two random variables (y_i, y_j) being assigned to a certain pair of categories.

To make the explanation more clear, we illustrate a simple example in figure 2.7, we ignore $\boldsymbol{\theta}$ here and explain it later and use $\psi_{i,j}$ for $\psi_{i,j}(y_i, y_j, x_i, x_j)$ to make expression of conditional probability uncluttered:

$$\begin{aligned} p(y_1, y_2, y_3, y_4) &= \frac{1}{Z(x_1, x_2, x_3, x_4)} \psi_1\psi_2\psi_3\psi_4\psi_{1,4}\psi_{1,2}\psi_{2,3}\psi_{2,3}\psi_{1,2,4} \\ &= \frac{1}{Z(x_1, x_2, x_3, x_4)} \psi_1\psi_2\psi_3\psi_4\psi_{1,4}\psi_{1,2}\psi_{2,3}\psi_{2,3} \end{aligned}$$

where the transformation from first line to second line comes from the ignorance of potentials with order higher than two.

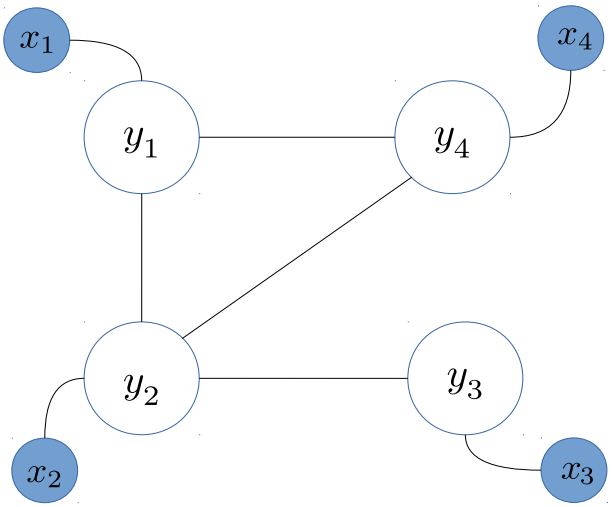


Figure 2.7: Simple example of conditional random field.

According to Hammersley-Clifford theorem [HC68], the the factors must be always positive, we can employ log-linear models for the potential functions, which results in:

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) &= \frac{1}{Z(\boldsymbol{x},\boldsymbol{\theta})}\prod_{c\in\mathcal{C}}\exp(\langle\phi(x_c,y_c),\boldsymbol{\theta}\rangle) \\ &= \frac{1}{Z(\boldsymbol{x},\boldsymbol{\theta})}\exp(\sum_{i\in V}\langle\phi_u(x_i,y_i),\boldsymbol{\theta}_u\rangle + \sum_{(i,j)\in\mathcal{E}}\langle\phi_p(x_i,x_j,y_i,y_j),\boldsymbol{\theta}_p\rangle) \end{aligned} \tag{2.40}$$

being $\langle\cdot,\cdot\rangle$ the inner product, and $\phi(x_c,y_c)$ the sufficient statistic of the factor over clique c. Because we consider the potentials up to second order, only feature function for unary potential $\phi_u(\cdot)$ and feature function for pairwise potential $\phi_p(\cdot)$ are considered here. This function can also be treated as feature function which extracts feature of the data over clique.

Now we have made assumptions of graph structure and conditional dependency for the graph. This can be interpreted as a filter of a probability distribution family. Only the class of probability distribution that satisfies these assumptions such as up to second order potentials can be expressed and learned with this graph. To further identify different probability distribution in this class, we need to parameterize it and identify it with specific parameters which can learned during training.

Parametrization of the probability distribution is expressed explicitly in Eq.2.40. This means that the dimension of the parameter in each kind of potential should be the same as the dimension of feature function. In this work, the feature function should be a function of input and class label, while the feature function can also be the function of only input and weight should be defined as matrix whose output dimension is the number of class serving as a linear mapping. In this work, we only consider the former case.

Regarding **unary feature**, we utilize the probability vector $p(y_i|x_i)$ which is predictive likelihood from a discriminative classifier which is a deep neural network in this work.

Pairwise feature in this work is quite simple context cue in the scene which is defined as binary co-occurrence matrix $\boldsymbol{H}(y_i,y_j)$. If one category occurs with another category in the same scene, then the corresponding entry in the matrix is filled with 1, otherwise 0.

Although obtaining this pairwise feature requires knowing the ground truth labels of objects in the scene, it still makes sense in the application of this work. Because we use T-LESS dataset which will be introduced in the experiment part, each of scene images in this dataset contains multiple industrial-relevant texture-less objects, some of them are combination of objects of other categories. We want to simulate the situation that, we treat each scene as one product which is composed of multiple sub-parts. Because we know the product, and thus we also know the real labels of components of this product which is corresponding to the co-occurrence of objects in this scene and our goal is to classify the components with help of this contextual information.

Based on these two kinds of features, we have the formulation of likelihood function of CRF model for one scene as follows:

$$p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{x},\boldsymbol{\theta})}\exp(\theta_u \sum_{i\in V} p(y_i|x_i) + \theta_p \sum_{(i,j)\in\mathcal{E}} \boldsymbol{H}(y_i, y_j)) \tag{2.41}$$

where $\boldsymbol{\theta} = \{\theta_u, \theta_p\} \in \mathbb{R}^2$.

2.2.2 Learning

The most popular approach is estimate $\boldsymbol{\theta}$ is maximum likelihood estimate(MLE). The likelihood for one scene is defined in Eq.2.41. Normally we have large size of scenes in training set where each of them contains multiple objects. We denote training set by $\mathcal{D} = [d_1, ..., d_m]$, where each scene is represented by $d_i = (\boldsymbol{x}^i, \boldsymbol{y}^i)$. For each scene, we create a CRF model for it. Therefore we can denote the whole set of graph by $\boldsymbol{\mathcal{G}} = [\mathcal{G}_1, ..., \mathcal{G}_m]$, where $\mathcal{G}_i = (V_i, \mathcal{E}_i)$. This is consistent to previous notations when we make little modifications for $\boldsymbol{x}$ and $\boldsymbol{y}$: $\boldsymbol{x}^i = [x_1^i, ..., x_n^i,]$ and $\boldsymbol{y}^i = [y_1^i, ..., y_n^i]$. For computational convenience and stability, negative log likelihood as objective function is always employed. The objective in MLE on training set is defined as fallows:

$$\mathcal{NLL}(\boldsymbol{\theta};\mathcal{D}) = -\sum_{i=1}^{m}\big(\theta_u \sum_{j\in V_i} p(y_j^i|x_j^i) + \theta_p \sum_{(j,k)\in\mathcal{E}_i} \boldsymbol{H}(y_j^i, y_k^i) - Z_l(\boldsymbol{x}^i;\boldsymbol{\theta})\big) \tag{2.42}$$

where

$$Z_l = \log\big(\sum_{\boldsymbol{y}^i\in\mathcal{Y}}\big(\theta_u \sum_{j\in V_i} p(y_j^i|x_j^i) + \theta_p \sum_{(j,k)\in\mathcal{E}_i} \boldsymbol{H}(y_j^i, y_k^i)\big)\big) \tag{2.43}$$

With this expression, our problem is defined in the following form with MLE:

$$\boldsymbol{\theta}^\star = \underset{\boldsymbol{\theta}}{argmin}\big[\mathcal{NLL}(\boldsymbol{\theta};\mathcal{D})\big] \tag{2.44}$$

The log likelihood function of CRF is proved to be concave [KFB09], which means the local optimizer is also the global optimizer although it's not unique. To optimize this objective w.r.t. the model parameters on dataset with large size, we can employ stochastic gradient descent (SGD), which requires to calculate the gradient of object w.r.t. the model parameters $\boldsymbol{\theta}$. The gradients can be computed as follows, where we use $\boldsymbol{\theta}$ to represent parameters in order to make the expression clear:

$$\frac{\partial}{\partial\boldsymbol{\theta}}\mathcal{NLL}(\boldsymbol{\theta};\mathcal{D}) = -\sum_{i=1}^{m}\big(\phi(\boldsymbol{x}^i, \boldsymbol{y}^i) - \mathbb{E}_{p(\boldsymbol{y}|\boldsymbol{x}^i;\boldsymbol{\theta})}\big[\phi(\boldsymbol{x}^i, \boldsymbol{y})\big]\big) \tag{2.45}$$

where the expectation $\mathbb{E}_{p(\boldsymbol{y}|\boldsymbol{x}^i;\boldsymbol{\theta})}\big[\phi(\boldsymbol{x}^i, \boldsymbol{y})\big]$ stands for the partial derivative of the log-partition function, which is retrieved by:

$$\mathbb{E}_{p(\boldsymbol{y}|\boldsymbol{x}^i;\boldsymbol{\theta})}\big[\phi(\boldsymbol{x}^i, \boldsymbol{y})\big] = \sum_{\boldsymbol{y}\in\mathcal{Y}} p(\boldsymbol{y}|\boldsymbol{x}^i;\boldsymbol{\theta})\phi(\boldsymbol{y}, \boldsymbol{x}^i) \tag{2.46}$$

The gradients of parameters corresponds to the difference between the empirical expectation of its associated sufficient statistics (the sufficient statistics of ground truth) and its expectation over the estimated probability distribution $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$. The goal of the optimization is to decrease the gradients until they reach zero, which means that we match of first moment of sufficient statistics over the output distribution. Therefore this way is also called moment matching.

When taking a closer look at Eq.2.45 and Eq.2.46, to calculate the gradients in each step of SGD, we need to compute $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$, which requires to compute the partition function $Z(\cdot)$. Because the number of possible assignments grows exponentially in the number of classes, it's infeasible to get a exact solution in practice. This problem also appears when we want to obtain a probability distribution(marginal inference) instead of a single prediction(MAP inference) in testing time. To compute the probability distribution in probabilistic modeling always refers to as inference. There are a bunch of research works focusing on inference. Because the CRF model in this work is fully connected, we adopt a popular and effective approach which is called loopy belief propagation and introduced in next subsection. This approach can provide a approximation to the log-partition function $Z_l(\cdot)$, which is required in Eq.2.46 and to the marginal probability distribution of random variables in $\boldsymbol{y}$ to be plugged in Eq.2.45 in the graph with cycles and exact solution in graph without cycle.

2.2.3 Inference

Loopy belief propagation introduced by [Pea14], also known as sum-product algorithm, is a popular approach for probabilistic inference in graphical models. Briefly, this approach is based on the exchange of statistical information among the nodes in the graph according to their relationships. The core idea behind that is for each node in the graph to maintain its belief, which is a k-dimensional vector $\boldsymbol{b}(y_i)$ if the node is discrete random variable where k is the number of class. The i-th entry of this vector indicates the probability that i-th class is assigned to this node. The belief of a node evolves as it receives *messages* from it neighbors. A message $\boldsymbol{m}_{ij}(y_j)$ sent from node y_i to node y_j encodes belief of y_i about what class node y_j should belong to. This message, in turn, based on the messages it received from all the other neighbors except for the recipient in a previous time-step.

The algorithm keeps sending messages until the graph is calibrated, which means that the messages of two consecutive iterations are less than one threshold. However, in practice, the algorithm can not calibrate or converge. A maximum number of iteration can be set to obtain satisfactory results. The messages computation is as follows:

$$\boldsymbol{m}_{ij}(y_j) = \sum_{y_i \in \mathcal{Y}_i} \psi_{ij}(y_i, y_j, x_i, x_j, \theta_p)\psi_i(y_i, x_i, \theta_u) \prod_{y_k \in \mathcal{N}(y_i)\backslash y_j} \boldsymbol{m}_{ki}(y_i) \tag{2.47}$$

where $\mathcal{N}(y_i)\backslash y_j$ is the set of node neighbors of y_i except for y_j. Once the iteration reaches maximum number or the graph is calibrated, we can compute the belief of each node as

follows:

$$\boldsymbol{b}(y_i) = \frac{1}{Z_i}\psi_i(y_i, x_i, \theta_u) \prod_{y_j \in \mathcal{N}(y_i)} \boldsymbol{m}_{ji}(y_i) \tag{2.48}$$

where Z_i is the normalizer of unnormalized belief of node y_i to ensure the sum of marginal probability of y_i is 1.

Chapter 3

Technical Approach

In this chapter, we will explain the approaches employed in this work. Firstly, confronting with the limitations of concrete dropout, we propose another version of concrete dropout, which is called multiple dropout(multi-drop). Secondly, we introduce the modified architecture of ResNet50 in order to incorporate different inference techniques in BNN into this powerful feature extraction network. Thirdly, combination of BNN and CRF in this work is introduced, where CRF works as a post-processor to capture the relationship between random variables. Last but not least, we explain the approach combining aforementioned techniques for continuous learning in the last section.

3.1 Multiple dropout

From figure 2.2 and expression of approximate distribution in equation 2.11, we choose only one probability of Bernoulli distributed random variable for each layer, therefore random vector $\boldsymbol{p}_i = [p_i]^{D_{i-1}}$ for i-th layer, which stacks same value into a vector. While the dropout regularization term pushes the probability of Bernoulli to 0.5 to maximize its entropy, the expected likelihood term tries to increase the probability because decreasing probability will lead to a different model with lower capacity and thus low performance. An equilibrium state between them should be achieved in training. With concrete dropout introduced above, we could extend dropout for each hidden units instead of each layer(cf. figure 3.3), which means random vector $\boldsymbol{p}_i = [p_i^k]_{k=1}^{D_{i-1}}$. The difference between multiple dropout and concrete dropout can be seen easily by comparing 3.3 and 2.3. While the first term in gradients computation stays the same, the second term should be modified to:

$$\begin{aligned}\frac{\partial KL(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))}{\partial\theta} &\approx \frac{\partial \sum_{i=1}^{L} \lambda||\boldsymbol{M}_i||^2 - \beta\mathcal{H}(\boldsymbol{p}_i)}{\partial\theta} \\ &= \frac{\partial}{\partial\theta}\big(\sum_{i=1}^{L}\lambda||\boldsymbol{M}_i||^2 - \beta\sum_{k=1}^{D_{i-1}}(-p_i^k \log p_i^k - (1-p_i^k)\log(1-p_i^k))\big)\end{aligned} \tag{3.1}$$

The reasons behind multi-drop are as following:

- to increase flexibility in tuning variational parameters. The tunability of parameter of Bernoulli random variable in the likelihood term is low because there is only single parameter controlling the entire layer. As is observed in the experiments(cf.figure 3.1), these parameters are always increased for each layer. The reason for this is probably that reducing it would lead to low capacity and thus low likelihood.
- the solution space of concrete dropout should be a subset of the solution space of multi-drop if all of them are reachable. Because if it's optimal to assign same probability for each hidden units, this can be recovered in training with multi-drop. Otherwise, other optimal solutions of assigning different probabilities to different hidden units could be considered.
- last but not least, multi-drop can extend the flexibility and diversity of the dropout approximate distribution family by adding more parameters. Hence the truth posterior can be approximated by the approximate posterior better.

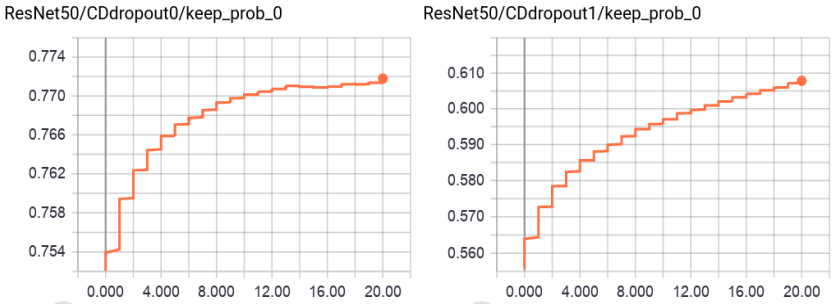


Figure 3.1: Changes of keep probability of the first two concrete dropout layers during training.

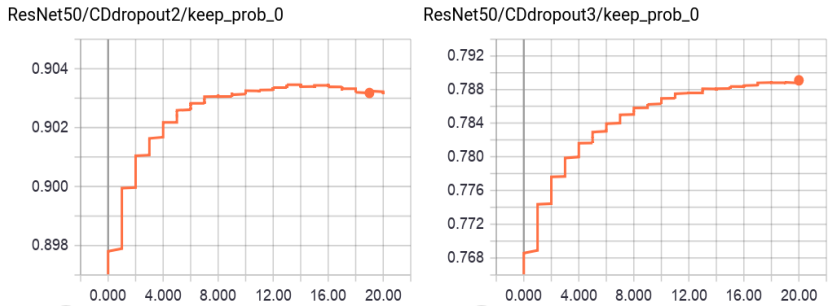


Figure 3.2: Changes of keep probability of the last two concrete dropout layers during training.

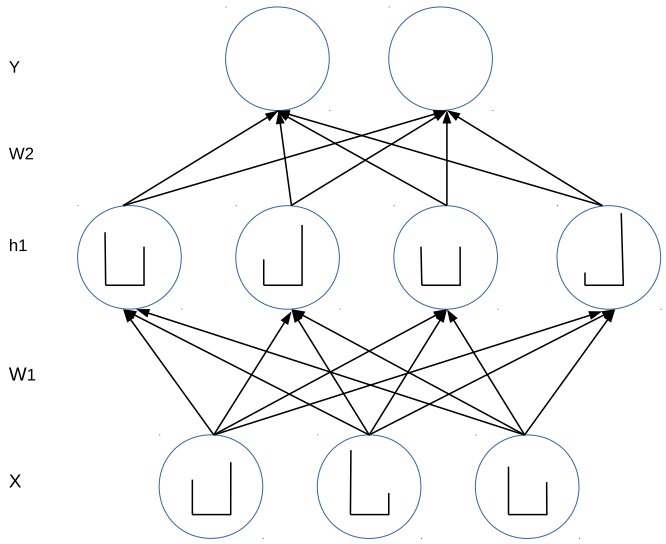


Figure 3.3: Different dropout rates for different hidden units in multi-drop.

3.2 Modified network architecture

After introducing dropout and concrete dropout variational inference, we will describe the modified network architecture in this subsection. The fundamental task in this work is object classification. We choose ResNet50[HZRS16] pre-trained on ImageNet as backbone for fine-tuning because of its strong ability to learn powerful representation for images. However, there is no dropout in original version of ResNet50. If we want to employ dropout variational inference to obtain reliable uncertainty estimation, dropout should be inserted into the network. In this work, we add three fully connected layer with 1024 hidden units , which are initialized from scratch, before the output layer whose dimension needs to be set to the number of classes. Then we add concrete dropout at flatten layer, and these three new added fully connected layer, respectively. There are three reasons why we modify the network in this way:

- Inserting dropout in layers initialized with pre-trained weights will destroy pre-trained features. Because we initialize major part of network with pre-trained weights, hence it would lead to significant drop of performance after fine-tuning if we insert dropout into them.
- According to the suggestions from [SHK+14], insertion of dropout reduces the capacity of the model and thus a model with dropout should have larger capacity than one without dropout. Therefore we add three more fully connected layers to make sure that our model possesses large enough model capacity.

- As we have introduced in previous sub-sections, weights are major part of variational parameters. Therefore to have more weights can enhance the flexibility and capacity of approximate distribution family, which improves the quality of approximation.

Figure 3.4 shows the sketch of our modified network architecture. We can interpret major part of network, which do not have dropout inserted, as a deterministic feature extractor. Following this, there four layers with dropout inserted work as a probabilistic classifier. These four layers include one flatten layer with 2048 hidden units, and three fully connected layer with 1000 hidden units each. At the end, an output layer with size of number of categories is appended. The version of dropout inserted can be normal dropout, concrete dropout or multiple dropout. In this report, we only show results of concrete dropout and multiple dropout because normal dropout underperfroms when compared with them.

In training, these two parts are trained together in order to achieve a better balance between uncertainty estimation and accuracy. As known to all, freezing feature extraction part can lead to worse performance, but it can also reduce the influence on predictive uncertainty from aleatoric uncertainty. To investigate this effect, we have done a ablation study which is shown in experiment part. In testing, we need to marginalize possible parameters according to posterior distribution. Layers with dropout should be turned on, sampled and run multiple times to perform Monte Carlo integration to approximate predictive distribution. The reduction of run time in testing can be achieved with different techniques, such as parallel computing or distillation. But this is out of the scope of this work.

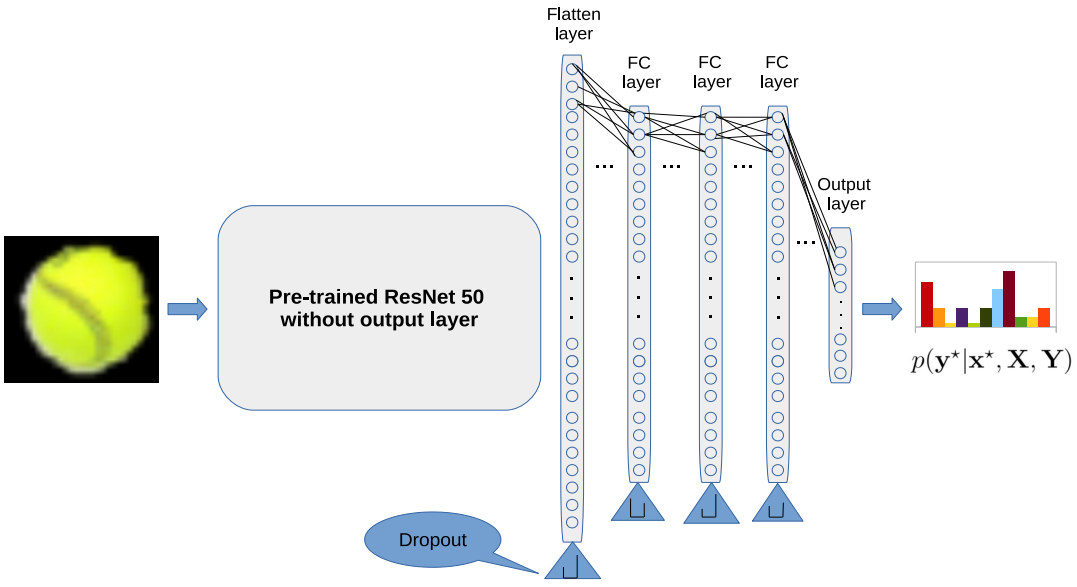


Figure 3.4: Modified network architecture of ResNet50.

3.3 Combination with CRF

After employing Bayesian neural network, the model can provide more reliable uncertainty estimation in a probabilistic manner. In other words, the output distribution can bring more information than before. It would be better to make use of this information with another probabilistic model. In this work, we explore to combine output from BNN and relationship between objects in scene via CRF. In detail, we use output distribution from BNN for unary feature and binary co-occurrence matrix for pairwise feature (introduced in 2.2.1). Figure 3.5 demonstrate the flow chart how BNN and CRF is combined together. As we can see, objects in one scene are firstly fed into BNN and their output distributions are fed into CRF as unary feature for each node. In the scene, their contextual relationship is represented by the binary co-occurrence matrix which is fed into CRF as pairwise feature. Then loopy belief propagation is used to infer the marginal distribution of each node.

One key factor in this case is that the pairwise feature is required to be designed by hand. The choice and design of it have significant impact on the improvement achieved by CRF. The information brought by pairwise feature should be complementary instead of contradictory to that of unary feature. Once this point is fulfilled, the more information unary feature has, the more improvement CRF can achieve.

One thing worth noting, in experiment part, we test this approach only on T-LESS dataset. Because to construct scene for objects in WRGBD dataset is non-trivial. The reason for this is that most objects there have not only similar appearances but also similar contexts. Therefore it's hard to disambiguate miss-classification by incorporating their contextual information.

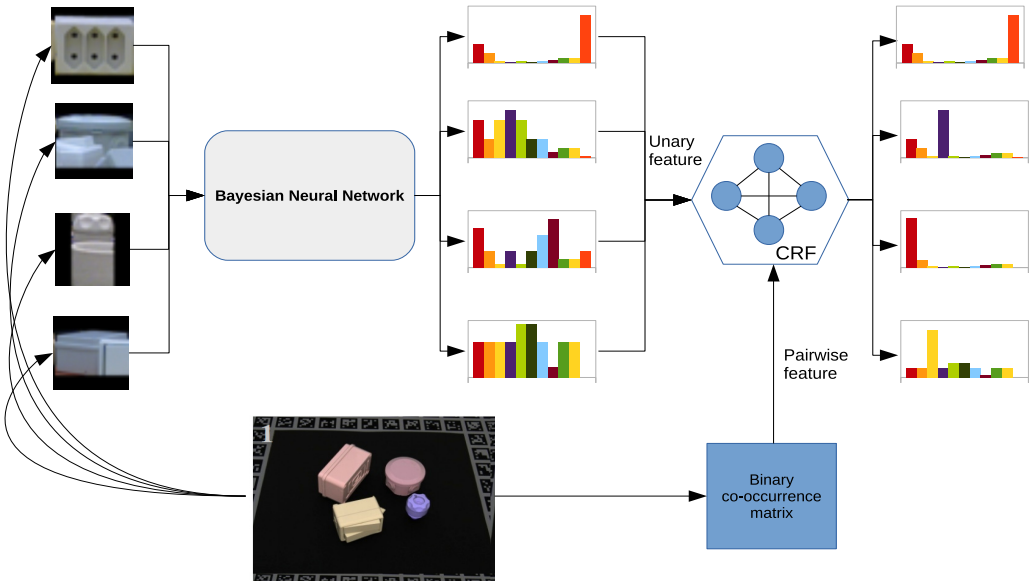


Figure 3.5: Combination of BNN and CRF.

3.4 Approach for continuous learning

With reliable uncertainty estimation, one of our goals is to enable the classifier or robot to learn continuously. The classifier should be introspective and express the confidence about its predictions. It's common that the test data in real environment does not have exactly same distribution as the training set, which induces a significant performance drop. In this situation, we expect our classifier to be able to adapt itself in real environment by fine-tuning itself with as little manual effort as possible.

This idea is visualized in figure 3.6. The initialization phase includes training model with easily obtainable dataset, which can public or synthetic dataset. Before deploying in the real environment, there is an adaptation phase which makes use of the aforementioned techniques. In this phase, the classifier should be able to adapt itself to the objects in real environment by fine-tuning with dataset collected with as little manual effort as possible. If the relationship between objects in real environment is complementary to the discriminative classifier and can be encoded well, CRF can be employed to capture the relationship to improve the performance further (cf. section 3.3). After that, the classifier is deployed to test data in real environment. In the experiment part, we evaluate this approach on both UniHB(WRGBD) and T-LESS(synthetic T-LESS).

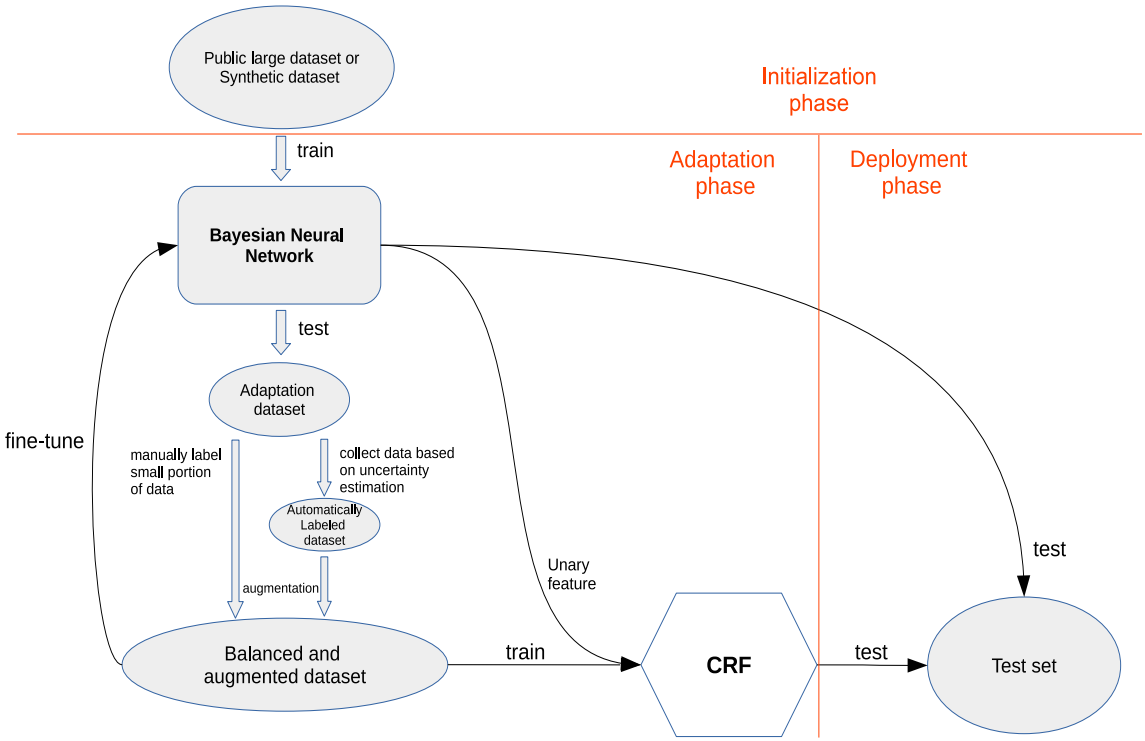


Figure 3.6: Approach for continuous learning.

Chapter 4

Experiments and Discussion

4.1 Preliminaries

In this chapter, we perform different experiments for the following purposes:

- to evaluate the performance of uncertainty estimation of dropout variational inference and its variants as well as Laplace approximation. Comparisons and analysis towards the results of these two kinds of approach are given.
- to evaluate performance of training a domain specific classifier, with different strategies for collecting training data including manual labeling and automatic labeling, where the latter one is chosen based on uncertainty estimation.
- to evaluate performance of classifier including context information via CRF.

Before looking into the results, we need to specify some details of experiments such as the dataset, evaluation metrics. Models of deep neural nets are implemented in Tensorflow[ABC+16], and optimization was performed using RMSprop with initial learning rate of $1e^{-5}$ and L2 regularization with coefficient of $3.5e^{-6}$ as well dropout regularization with coefficient of $1.0e^{-5}$. We used early stopping with all methods, where the amount of epochs to run was determined based on performance on a validation set. We set the number of maximum epoch as 20.

Regarding implementation of conditional random field, we employ C++ library from [RSGGJ15]. However, the weights there are assumed to be category-independent, which means that the weights are defined as weight matrix to build a linear mapping from feature function to potential function. Therefore we modify the type of weights in this library in order to incorporate different types of weight defined in Eq. 2.41, which serve as coefficients of different potential functions instead of a linear mapping inside the potential functions. To note that the messages are initialized uniformly during loopy belief propagation and parameter are initialized with Gaussian prior with small standard deviation (0.01) around 0.

Besides, we perform optimization with stochastic gradient descent(SGD) with momentum and set size of mini-batch as 16, learning rate as $1e^{-}4$ and maximum number of iteration as 15×10^3. Loopy belief propagation(LBP) is used for inference both during training and testing. The maximum number of iteration for computing messages in LBP is 100, and the convergence threshold is $1e^{-4}$.

4.1.1 Dataset

WRGBD[LBRF11] This dataset is a public large-scale dataset of 300 household objects captured from multi-viewpoint. The objects are organized into 51 categories and 300 instance classes in hierarchical structure. There are multiple instances in different categories. Each object was placed on a turn table and captured from a systematically sampled view hemisphere with 15° step in elevation (from 30°to 60°) and 2° step in azimuth (from 0° to 360°) (cf. figure 4.1). The total size of entire dataset is around 160.9×10^3. To note that category level recognition involves classifying objects with similar semantic appearance such as cereal boxes with different texture into same category, instead of just physical appearance. Instance level recognition is to classify objects with similar physical appearance to the same category. Therefore the overlapping in features space between classes in category recognition is larger than that in instance recognition. More abstract and semantic concepts are expected to be learned in category recognition.

UniHB To simulate application-specific situation in deployment of robots, we recorded a similar dataset by putting objects on a turn table in front of the robot and recording partial views of objects. We follow the same methodology as [LBRF11] suggests for WRGBD, but with only one object instance for each category in the dataset. This analogue of WRGBD dataset is called the IAI-ODU dataset of UniHB. In this work, we call it UniHB dataset for simplicity. The size of data with elevation 45° is around 8.6×10^3 and that of data with elevation 30° and 60° is around 17.1×10^3.

While the UniHB dataset's setup strives to mimic the WRGBD one, the differing capturing conditions such as different equipment and light conditions even appearances of objects, result in obvious changes in feature space, thus yielding a significant drop on classification accuracy. In addition to the existing 51 categories, there are 28 novel objects(cf. figure 4.2) that do not belong to the 51 class categories and these objects are treated as out-of-distribution(OOD) data for testing uncertainty estimation of the model. To note that data of these novel classes are recorded in a slight different way, that is, their view points are sampled with 15° step in elevation (from 30°to 60°) and 5° step in azimuth (from 0° to 360°). The size of this OOD dataset is around 6.0×10^3.

T-LESS[HHO$^+$17] This dataset features 30 commodity electrical parts which have no significant texture, discriminative color or distinctive reflectance properties, and often bear similarities in shape and/or size. Furthermore, a unique characteristic of the objects is that some of them are parts of others. All images were captured systematically sampled from a

view sphere, resulting in around 38.0×10^3 training images and 10.0×10^3 test scene images (cf. figure 4.3). The training images depict objects in isolation with a black background, while the test images are from 20 table-top scenes with arbitrarily arranged objects placed on table. In this work we only consider the cropped images of objects in testing scenes, resulting a testing set with size 69.5×10^3. Nevertheless, the context information in the scene can be exploited with conditional random field in the next experiment.

Because objects in test scenes are mostly occluded and have different backgrounds and lightness. Therefore we employ data augmentation to improve the performance, where the real single objects are augmented and put onto different background images from VOC2012 dataset[EVGW$^+$](cf. figure 4.4). Since collecting real data requires much manual effort, we want to first train classifier on synthetic data which are multi-view images of objects rendered from 3D reconstructed CAD models with similar augmentations as the real single objects(cf. figure 4.4). Although the real object and synthetic object do not look much different, there is significantly large domain gap between them such as texture and details on the object so that the performance of classifier trained on synthetic dataset on test set decreases a lot. One way to tackle this is that we can collect labeled data automatically based on improved uncertainty estimation, which can then be used to train a more accurate classifier.

Figure 4.1: Example of masked images of objects from 51 categories in WRGBD and UniHB dataset. In each category, the left is from WRGBD and the right is from UniHB. We randomly pick one instance for the objects of WRGBD. We can see some light and appearance difference between objects in these two datasets.

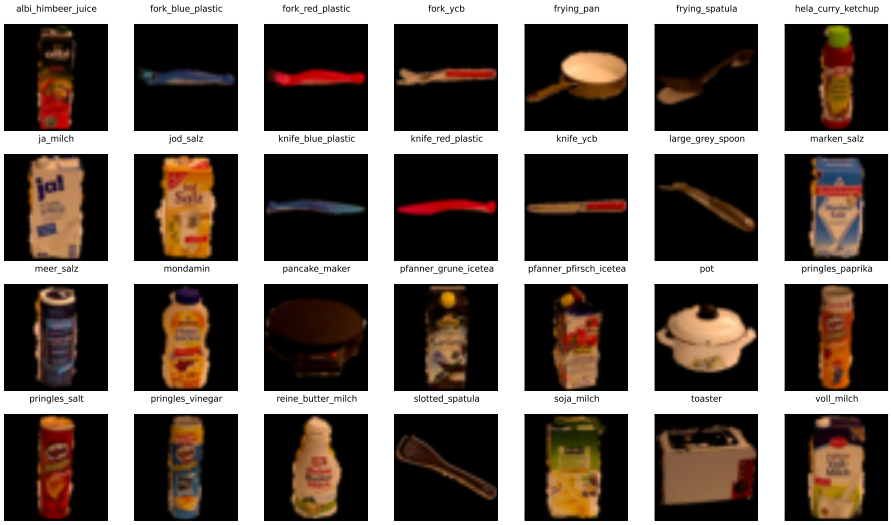


Figure 4.2: Example of masked images of objects from 28 categories which are not belonging to WRGBD categories, which are treated as OOD data samples.

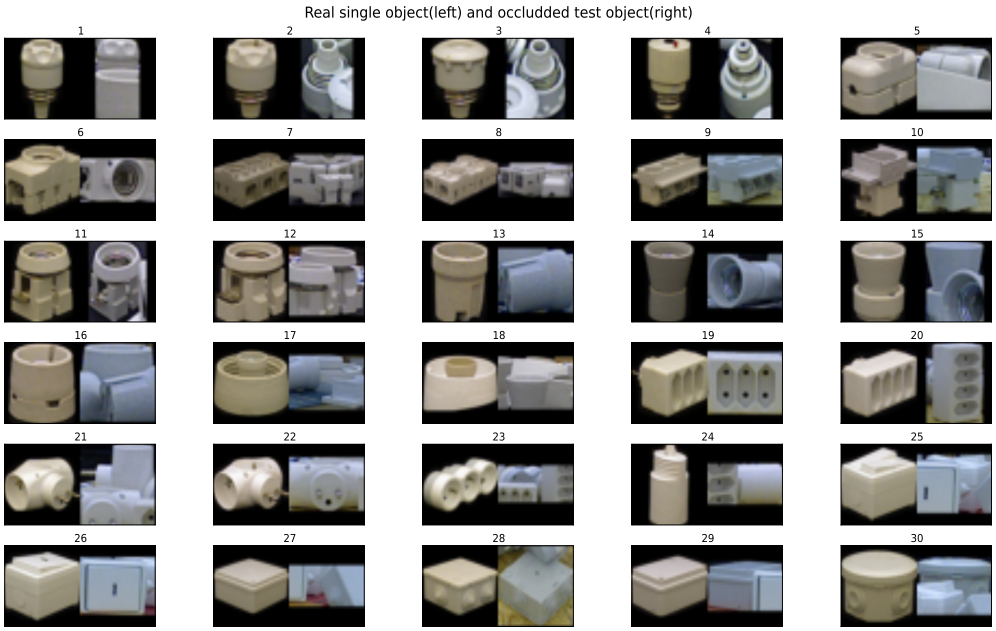


Figure 4.3: Example of object of each category in original training set and test set. In each thumbnail, the left image is real single object with black background and the right image is cropped image of object in test scene. (label index starts from 1.)

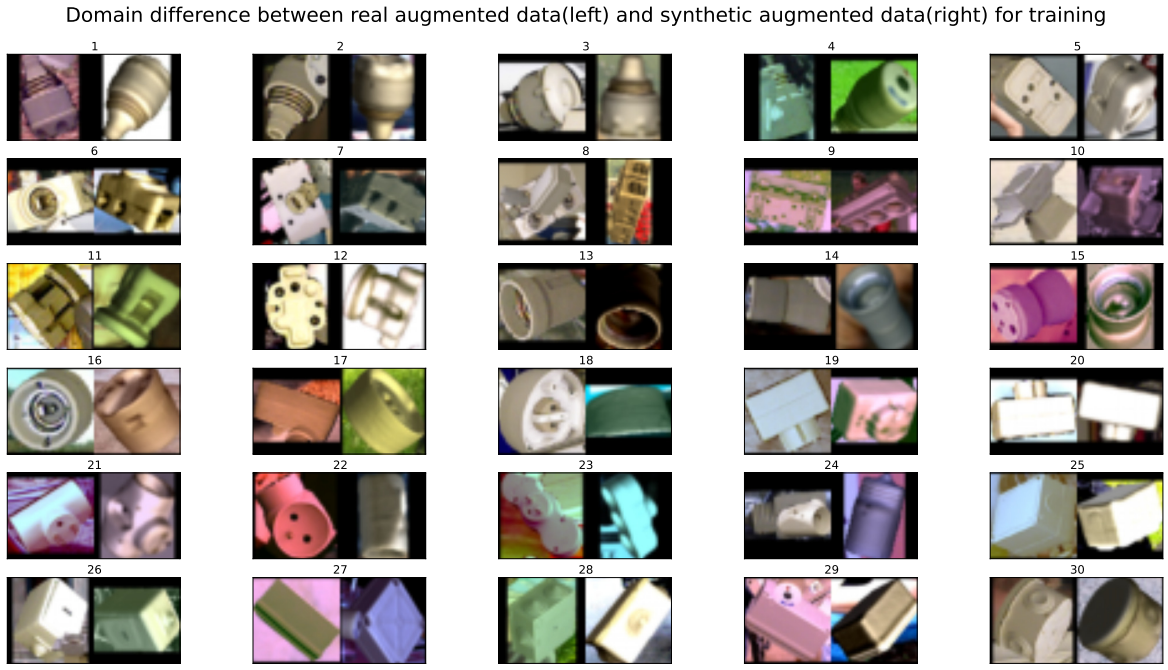


Figure 4.4: Example of object of each category with augmentation. In each thumbnail, the left image is real augmented object and the right image is synthetic augmented object.(label index starts from 1.)

4.1.2 Uncertainty measures

For each prediction, we can obtain one predictive probability distribution from our model with equation 2.23. In order to quantify uncertainty of the prediction, there are different metrics to measure the uncertainty of prediction, which are introduced in the following. We define $x^\star$ as one test data sample and $\mathcal{D}$ as our training set.

Confidence is defined as the maximum probability of the output predictive distribution, whose index is the class prediction.

$$conf = \max_c \left[p(y = c|x^\star, \mathcal{D})\right] \tag{4.1}$$

where $conf \in [0, 1]$, $c \in \mathcal{L} = \{0, ..., |\mathcal{L}| - 1\}$, and $\mathcal{L}$ represents output space in which label is expressed as number index which is transformed into one-hot representation in computation of objective function. The larger this quantity is, the less uncertain the prediction is.

Predictive entropy is quantity that captures the average amount information contained in predictive distribution[Sha48]:

$$\mathcal{H}[p(y|x^\star, \mathcal{D})] = -\sum_c p(y = c|x^\star, \mathcal{D}) \log p(y = c|x^\star, \mathcal{D}) \tag{4.2}$$

where c is the possible class y can take, $\mathcal{H}(\cdot) \in [0, \log|\mathcal{P}|]$, the larger this quantity is, the more uncertain the prediction is.

Mutual information between the prediction y and the weights posterior offers a different uncertainty measure when compared with the aforementioned ones. This measure is widely used in active learning tasks[HHGL11], which is also called Bayesian active learning disagreement(BALD). The definition of this measure is as follows:

$$\begin{aligned}
\mathcal{I}[y, \boldsymbol{\omega}|x^\star, \mathcal{D}] &= \mathcal{H}[y|x^\star, \mathcal{D}] - \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D})}\big[\mathcal{H}[y|x^\star, \boldsymbol{\omega}]\big] \\
&= -\sum_c p(y = c|x^\star, \mathcal{D}) \log p(y = c|x^\star, \mathcal{D}) \\
&\quad + \mathbb{E}_{p(\boldsymbol{\omega}|\mathcal{D})}[-\sum_c p(y = c|x^\star, \boldsymbol{\omega}) \log p(y = c|x^\star, \boldsymbol{\omega})] \\
&\approx -\sum_c p(y = c|x^\star, \mathcal{D}) \log p(y = c|x^\star, \mathcal{D}) \\
&\quad + \mathbb{E}_{q(\boldsymbol{\omega})}[-\sum_c p(y = c|x^\star, \boldsymbol{\omega}) \log p(y = c|x^\star, \boldsymbol{\omega})]
\end{aligned} \tag{4.3}$$

where $\mathcal{I}(\cdot) \in [0, \log|\mathcal{P}|]$. This quantity considers the effect of approximate posterior distribution more directly. Compared with aforementioned uncertainty measures, this one should be able to capture the model uncertainty more accurately. To think about it intuitively, this quantity measures the information gain between the entropy of predictive output distribution and the expected entropy of output distribution wr.t. weights posterior. It will be low only if the predictive distribution agrees with most of possible models(weights realizations), which means that the model is sure about its prediction. Otherwise, it will be high because most of possible models do not agree with the other models and thus the predictive distribution will be more uniform and thus has higher entropy.

4.1.3 Evaluation metrics

As is stated in [GBR07], the goal of probabilistic prediction is to maximize the sharpness of predictive distribution subject to calibration. Calibration refers to the statistical consistency between the predictive probability and the occurrence of observations, which is the frequency of the event. Therefore we employ different metrics including both the accuracy and other quantities related to calibration as well as summary of accuracy and calibration. Additionally, we also employ histogram and diagram to express the results visually. While the visual one can show us the results more intuitively, the quantitative one can allow us to evaluate the results more objectively. The comparison between them may help us to examine if visual metrics are corresponding to the numerical metrics, which may provide more insights in evaluating uncertainty estimation.

Uncertainty histogram is a intuitive visual tool for analyzing the statistics of the uncertainty estimation. Compared with normal histogram, there is one difference to stress

on. In order to make visual effect more clear and hence the analysis easier, **normalizer** of each type of prediction is the size of this type of predictions instead of size of all predictions. In detail, we have three types of prediction for plotting the histogram, which are:

- **correct prediction**
- **miss-classification**
- **out-of-distribution(OOD)**

The range of y axis is $[0, 1]$ because we normalize the number in each bin, and range of x axis is set by the range of corresponding type of uncertainty measure.

Reliability diagram(Calibration curve) is another visual tool for expressing **calibration** performance of the model[GPSW17], which plots the frequency of success(accuracy of predictions in specific bin) as a function of confidence, which is predictive likelihood of prediction. If the model is perfectly calibrated, this function should be overlapping with the diagonal line exactly. The closer this curve is to diagonal curve, the better the calibration performance is. In order to draw the curve, we firstly group the predictions into M interval bins w.r.t. confidence and then calculate the accuracy of predictions in each bin. We use $M = 20$ in this work. To quantify the proximity to the diagonal curve, we use two metrics:

- **Expectation calibration error(ECE)**: In order to obtain a more objective measure of calibration quality, we can compute the **expected calibration error** by computing the weighted average of difference between accuracy and confidence:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)| \tag{4.4}$$

  where n is the number of samples, and $acc(B_m)$ represents the accuracy of samples, $conf(B_m)$ the average predicted confidence of samples in m-th bin. We can see that this metric measures the inconsistency between the statistics and predictive distribution. To note that this metric only considers the calibration quality instead of accuracy. One drawback of this metric is that, because the weights are computed based on number of samples in each bin, the metric would bias if most of predictions are clustering in few bins. Then the weights of these bins are much larger than other bins and other bins with large error would be ignored.

- **Maximal calibration error(MCE)**: Considering the drawback mentioned in ECE, we make use of this metric. Additionally, in high risk applications where reliable confidence measures are absolutely necessary, we may wish to take the worst case into account. It is defined as follows:

$$MCE = \max_{m} |acc(B_m) - conf(B_m)| \tag{4.5}$$

Proper scoring rules: Scoring rules provides a **summary** measure in the evaluation of probabilistic forecasts. It assigns a numerical score based on the difference between predictive distribution and real distribution.

Because we want to make predictions for the future and also have a suitable measure of the uncertainty associated with them (see [GR07] for a review). We can define the scoring rule as a function $\mathcal{S}(p(y|\boldsymbol{x}), (y|\boldsymbol{x}))$ that evaluates the quality of predictive distribution $p(y|\boldsymbol{x})$ relative to an event $y|\boldsymbol{x} \sim q(y|\boldsymbol{x})$ where $q(y|\boldsymbol{x})$ represents the true distribution over $(y|\boldsymbol{x})$. Consequently, the expected scoring rule is:

$$\mathcal{S}(p, q) = \int q(y|\boldsymbol{x})\mathcal{S}(p, (y|\boldsymbol{x}))dy \tag{4.6}$$

$\mathcal{S}(p, q)$ is proper if $\mathcal{S}(p, q) \leq S(q, q)$, with equality holds if and only if $p(y|\boldsymbol{x}) = q(y|\boldsymbol{x})$. Here we adopt two simple and famous proper scoring rules in which the less it is, the better the performance is:

- **averaged negative log likelihood(NLL)** is a popular metric for evaluating predictive uncertainty [QCRS$^+$05]. This metric considers aforementioned confidence as likelihood. The smaller this metric is, the better the predictive distribution is.

$$NLL = -\frac{1}{|\mathcal{D}_{test}|} \sum_{i=1}^{|\mathcal{D}_{test}|} \log(p(y_i = c_i|\boldsymbol{x}_i)) \tag{4.7}$$

  where c_i is the ground truth label for $\boldsymbol{x}_i$.

- **Brier score** is the mean squared error between the target distribution(one-hot encoding label) and predictive distribution:

$$BS = -\frac{1}{|\mathcal{D}_{test}|} \sum_{i=1}^{|\mathcal{D}_{test}|} (\boldsymbol{y}_i^{gt} - p(\boldsymbol{y}_i|\boldsymbol{x}_i))^2 \tag{4.8}$$

  where $\boldsymbol{y}_i^{gt}$ is the one-hot encoding ground truth label for $\boldsymbol{x}_i$.

Separability metrics: In addition to the summary metrics of predictive probability. We are also interested in the separability between different types of predictions, which can assist the down-stream tasks if they are highly separable (e.g. separate correct predictions and false predictions or out-of-distribution data to improve robustness of system and provide more safety guarantee). To note that, we treat the correct predictions as positive samples in the two following metrics.

- **Area under Receiver Operating Characteristic curve(AUROC)**: Since one of our goals is to choose automatically labeled data based on uncertainty estimation, it's necessary to evaluate the separability between correct predictions and false

predictions or out-of distribution data predictions. ROC curve describes the relationship between true positive rate($tpr = \frac{tp}{tp+fn}$) and false positive rate($fpr = \frac{fp}{fp+tn}$). Moreover, AUROC can be interpreted as the probability that a positive samples has a greater score than a negative samples. However, the drawback of AUROC is that, the normalizers of two kinds of rate in ROC curve are independent to each other. When these two normalizers differs too much. AUROC can provide misleading conclusion. For example, if the number of negative predictions is much higher than positive one, the AUROC could still achieve a relatively high value, while in fact there are already many false positives.

- **Area under Precision Recall curve(AUPR)**: Considering the downside of AUROC, another evaluation metric is employed, that is AUPR. AUPR describes the relationship between precision($pr = \frac{tp}{tp+fp}$) and recall($tpr = \frac{tp}{tp+fn}$), which resolves the problem of different base number. In the previously mentioned case, though the AUROC is high, the AUPR will be low because the precision is low.

4.2 Uncertainty estimation experiments

In this part, we evaluate performance of accuracy and uncertainty estimation as well as their summarized performance of different inference techniques for BNN on WRGB and UniHB dataset. Before that, the shorthands of different techniques should be explained. We call original ResNet50 "ori", ResNet50 with concrete dropout "cdp", ResNet50 with multiple dropout "mdp" without specifications.

In the first experiment, we evaluate them on a relatively simple task, instance recognition. However, our main focus is in the second experiment, where different techniques including dropout, Laplace approximation and ensembles are evaluated and compared on a more difficult task, category recognition. Besides comparing them, we also conduct a ablation study to investigate how feature extractor influences the predictive uncertainty.

4.2.1 Experiments I: uncertainty estimation on instance recognition

In this experiment, we want to evaluate our model on a relatively easy task which is instance recognition at first. The appearances of different classes as well as the out-of-distribution data are highly discriminable. To this end, we separate WRGBD dataset into two subsets based on the instance class. Subset I contains objects with instance class from 0 to 199 (assuming that here we use index to denote the instance label) and Subset II contains objects with instance class from 200 to 299.

We train our model with objects captured in elevation 30° and 60° of Subset I (in which we split off 20% of training set as validation set for model selection in training). The size of training set and validation set is around 71.0×10^3. Then we test the model on objects

captured in elevation 45° of both Subset I and Subset II. In this experiment, the objects in Subset II serve as out-of-distribution samples because they are not present during training. The size of test set is around 35.6×10^3 and that of OOD dataset is around 17.8×10^3.

In this experiment, we just want to test and compare the performance of different approaches. Because the OOD data is highly different to the training set. The model should express high uncertainty when facing these data. Nevertheless, the miss-classifications should have higher uncertainty than the correct predictions because the model should be confident when it makes predictions correctly. Otherwise the model is overconfident when it's confident with its incorrect predictions, which can induce hazardous consequence in some safety-critical applications. As we can see in the figure 4.5, the original ResNet50 model is highly overconfident, which assigns high confidence to nearly 50% of miss-classification and nearly 30% of OOD data in the highest bin. While the model with concrete dropout and multi-drop can significantly decrease these two proportions. As we can see, different uncertainty show similar trend in this experiment.

In addition to uncertainty histogram, we plot the calibration curve of each model evaluated on both test set and OOD dataset in figure 4.6. As expected, the calibration performance of original model is much worse than those of the other two models. This result is corresponding to the results in [GPSW17], that the original ResNet50 is highly overconfident. And the concrete dropout and multi-dropout can mitigate the undesired problem significantly.

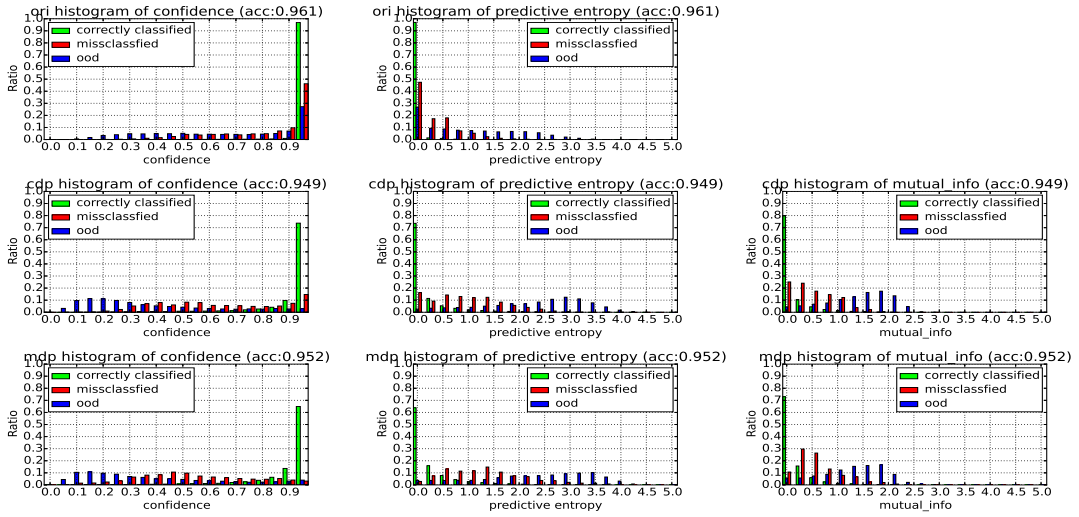


Figure 4.5: Uncertainty(confidence, predictive entropy, mutual information) histograms of original ResNet50, ResNet50 with concrete dropout and ResNet50 with multi-dropout.

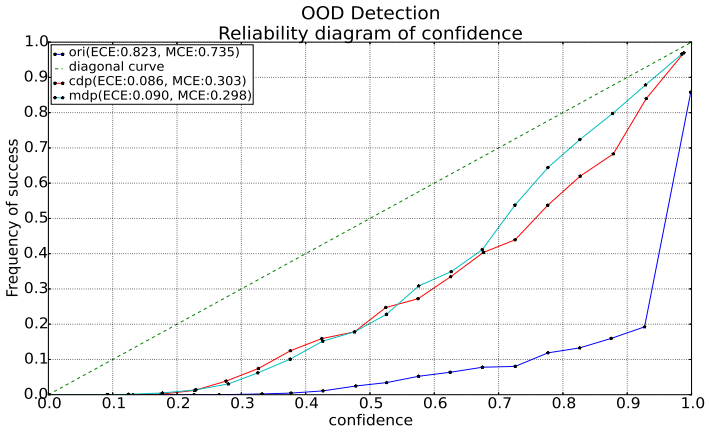


Figure 4.6: Calibration curves of original ResNet50, ResNet50 with concrete dropout and ResNet50 with multi-dropout evaluated on both test set and OOD dataset.

4.2.2 Experiments II: uncertainty estimation on category recognition

In this experiment, we evaluate our model on a category recognition task. The difficulties are expressed in two-fold. Firstly, in this task, the model is confronted with classes with more overlapping. Second, because we want to simulate situation of deploying robots in real world scenario and facing data in real world, the UniHB dataset with domain gap(cf. figure4.1) to WRGBD is used to achieve this goal. This means, the uncertainty estimation should not only be able to perform well on dataset with exactly same distribution to training set, but also to generalize well to dataset with domain gap which may decrease the accuracy significantly.

Accordingly, we use the entire WRGBD dataset including all view points, whose size is around 160.9×10^3, to train our model (in which we split off 20% of training set as validation set for model selection during training). When it comes to UniHB dataset, we treat objects captured in elevation 30° and 60° in this dataset as **adaptation set**, whose size is around 17.1×10^3. We test performance of uncertainty estimation on this adaptation dataset. The images of 45°, whose size is around 8.6×10^3 , are used for final evaluation after we fine-tune the model with subset of adaptation set to obtain a domain specific model. The latter part will be experimented in next section. Besides, we also evaluate the uncertainty estimation on OOD dataset(cf. figure 4.2) whose size is around 6.0×10^3.

We firstly explain protocols in the following for different kinds of metric, which can help understanding the plots and extracting useful information more easily and quickly.

- The following visual metrics are chosen in one of three runs of different random seeds. The average quantitative results are given in tables following the visual metrics.

- The calibration curve with title "Classification" on the left is plotted **only** on predictions of test set. The one with title "OOD detection" on the right is plotted on predictions of **both** OOD dataset and test dataset.
- The ROC curve and PR curve measure the separability between two types of prediction. The ones with title "classification" on the left measure the separability between **correct prediction** and **miss-classification**. The ones with title "OOD detection" on the right measure the separability between **correct prediction** and **OOD prediction**. In all ROC curve and PR curve, correct prediction is always chosen to be positive.
- As is shown in uncertainty histogram, we have used three uncertainty measures. Each one has its own ROC curve and PR curve of correct prediction versus miss-classification or OOD prediction. In the following plots of ROC curve and PR curve, we only show the one with highest area under curve. In detail, **Confidence** is chosen in case of correct prediction versus miss-classification. **Mutual information** is chosen in case of correct prediction versus OOD prediction.

Comparison with Ensemble

In this subsection, we will show the results in not only a qualitative (visual) way, but also in a quantitative way. The approaches we compare here include original version of ResNet50(ori), modified ResNet50 with concrete dropout(cdp), modified ResNet50 with multiple dropout(mdp) as well as their ensemble version. The members in ensemble are just initialized from different random seeds and no other techniques for enhancing the performance are used. In order to quantify the results more objectively, we average the results from three different random seeds and report the mean and the standard deviation.

In the figure 4.9, on the left it's figure of calibration curves of different approaches, plotted from predictions on test dataset. We can see that the calibration performance is improved a lot by the cdp and mdp. Additionally, their ensemble versions can give even better results which are tightly overlapping with the diagonal curve. On the right there are curves plotted from predictions on both test dataset and OOD dataset. Similar ranks and trends as the left curves appear here. After adding the OOD data into evaluation, all of the calibration curves are pulled down, which means that the calibration performance get worse. If all OOD data is assigned with low confidence, then curves between left and right should be similar because the accuracy in middle or high confidence interval would not change a lot or be pulled down a lot. By comparison these two figures, we can also see that the robustness against OOD data of cdp, mdp and their ensembles are better than the original version.

In the figure 4.8, we can see uncertainty histograms for different approaches and different uncertainty measures. Firstly, let's compare them vertically. We can see the improvement visually. The original version is still overconfident, which express low uncertainty to more than 50% of OOD data and nearly 40% of miss-classification. The cdp and mdp can lower this percentage to around 10%, although at the same time the percentage of correct pre-

dictions is decreased. But we can know that the predictions for which the model expresses low uncertainty are more likely to be correct. And the ensemble of cdp and mdp can perform even better with miss-classification. However, the ensemble of mdp does not work as well as cdp version on OOD data, which can be seen in the histogram on the last row of the figure. The reason for this will be investigated in a ablation study later. Then we can compare them horizontally, it's shown that different uncertainty measures have similar trends in the histogram. Therefore we need other quantitative metrics for them in order to evaluate different approaches as well as metrics more accurately.

When it comes to the separability metrics in figure 4.7, as stated before, we measure separability between correct prediction and miss-classification (on the left) and that between correct prediction and OOD prediction (on the right). On separability between correction prediction and miss-classification, cdp, mdp and their ensemble version can improve the results compared with original version in both ROC curve and PR curve on the left. On the other hand, the improvements on separability between correct prediction and OOD data are more obvious on the right. However, as observed before, the ensemble of mdp does not work well with OOD data, which can be observe with this metrics. We can know more from the curves that the worse performance with OOD data in mdp is enlarged by building ensemble of it.

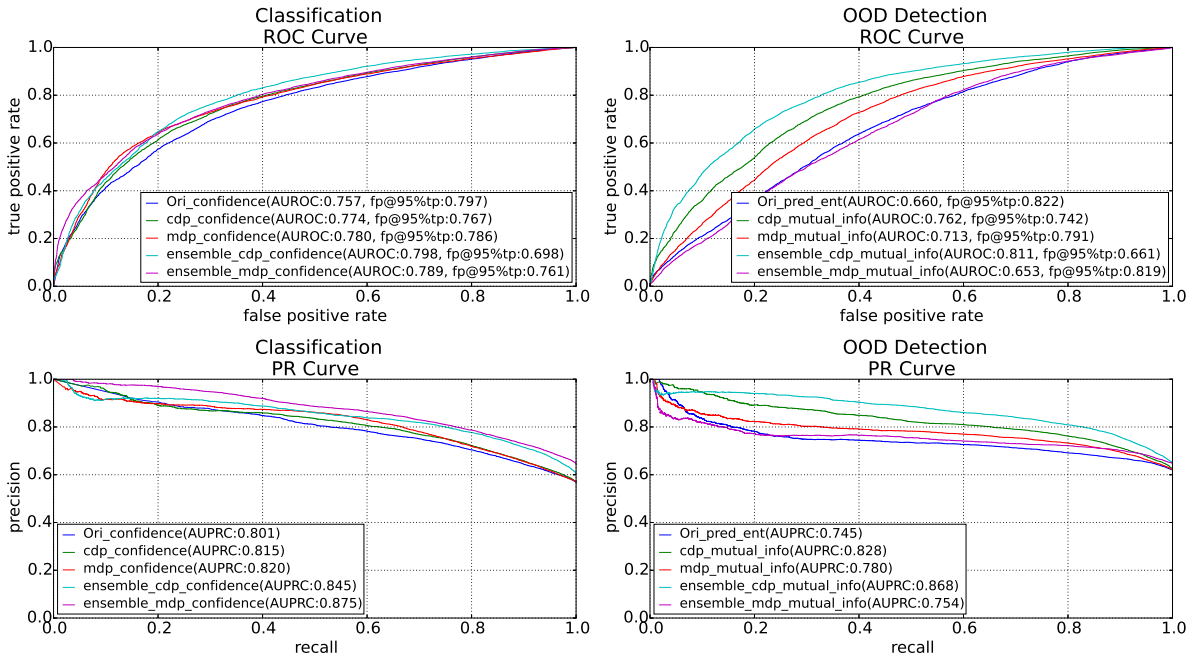


Figure 4.7: ROC and PR curve of ori, cdp, mdp, ensemble of cdp, ensemble in one of three runs.

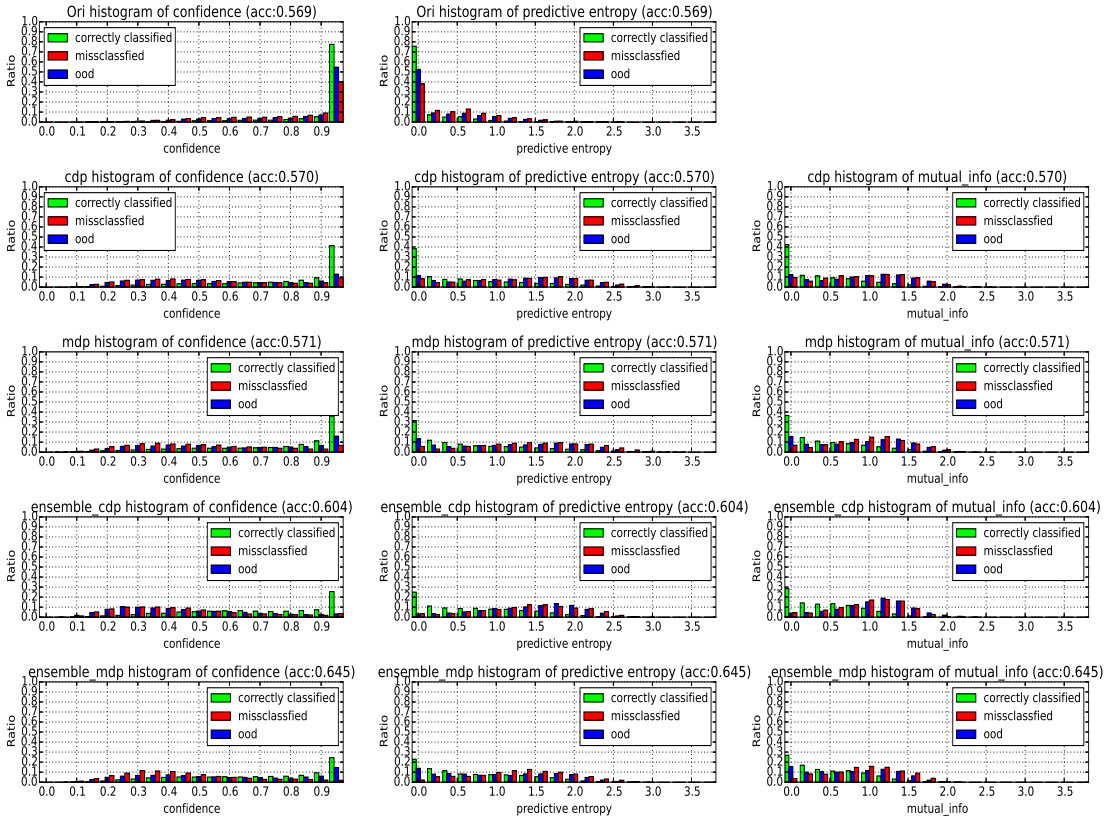


Figure 4.8: Uncertainty histograms of confidence, predictive entropy, mutual information for ori, cdp, mdp, ensemble of cdp, ensemble of mdp in one of three runs.

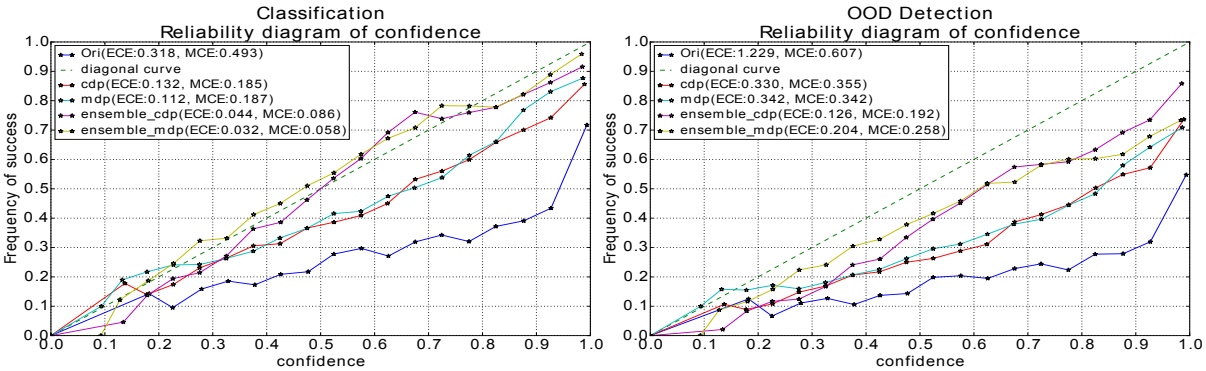


Figure 4.9: Calibration curve for ori, cdp, mdp, ensemble of cdp, ensemble of mdp in one of three runs.

Before we only show the visual metrics for evaluation, it's better to quantify the performance quantitatively. We adopt different aforementioned metrics to achieve this which are

presented in the following table 4.1. In order to exclude the influence of random initialization, we train three models of different random seeds and report the mean and standard deviation for each metric. We can see the improvements by cdp, mdp and their ensemble versions along all the metrics. The phenomenon that mdp does not work well with OOD and this effect enlarged by ensemble is also expressed in the table. Based on these experiment results, we can draw the conclusion that, concrete dropout and multiple dropout can improve both accuracy and calibration performance compared with original ResNet50. While the ensemble of concrete dropout and multiple dropout work even better without OOD data, the ensemble of multiple dropout work worse than that of concrete dropout when considering OOD data in this work. When comparing concrete dropout and multiple dropout, concrete dropout is more robust than multiple dropout because the latter one does not work well with OOD data.

Table 4.1: Quantitative results of acc, bs, nll, ece, mce, auroc, aupr averaged from 3 different random seeds

	accuracy↑	brier_score ↓	negative log likelihood ↓
ori	0.568±0.008	0.722±0.019	3.242±0.340
cdp	0.577±0.008	0.594±0.013	2.088±0.181
mdp	0.599±0.023	0.566±0.020	1.940±0.064
emsemble_cdp	0.604	0.534	1.452
emsemble_mdp	**0.645**	**0.496**	**1.389**

	expected calibration error(w/o. OOD/ w. OOD)↓	maximal calibration error(w/o. OOD/ w. OOD)↓	area under ROC (vs. Miss-classified/ vs. OOD)↑	area under PR curve (vs. Miss-classified/ vs. OOD)↑
ori	0.304±0.016 / 0.633±0.065	0.461±0.027 / 0.362±0.025	0.750±0.007 / 0.664±0.011	0.802±0.008 / 0.751±0.018
cdp	0.124±0.023/ 0.288±0.048	0.206±0.015/ 0.374±0.018	0.775±0.008/ 0.783±0.022	0.825±0.007/ 0.850±0.022
mdp	0.114±0.012/ 0.383±0.046	0.199±0.016/ 0.367±0.023	0.780±0.011/ 0.709±0.004	0.838±0.013/ 0.788±0.006
emsemble_cdp	0.044/ **0.042**	0.086/ **0.093**	**0.798**/ **0.811**	0.845/ **0.868**
emsemble_mdp	**0.032**/ 0.170	**0.058**/ 0.227	0.789/ 0.653	**0.875**/ 0.754

Comparison with Laplace approximation

Because Laplace approximation requires only MAP point estimate of model parameter. Therefore we take the already trained model of different approaches as our MAP point estimate and compute the approximation of Kronecker factors with half of training set. We set the scale parameter of Kronecker factors $\sqrt{N}$ as 1 and dump factor $\sqrt{\tau}$ as 15 in equation 2.38 based on grid search on the validation set.

In the following figures, we show the histograms, calibration curves, ROC curve and PR curve of cdp, mdp and their Laplace approximation versions. We can see that the Laplace approximation can achieve similar results as the cdp or mdp do in uncertainty histograms, calibration curves as well as ROC curve and PR curve. The histograms of different uncertainty measure show similar trend as before.

If we compare the Laplace approximation between cdp and mdp, we can see that the result of Laplace approximation is similar to the their dropout versions. This can be observed through that the percentage of OOD prediction with high confidence of Laplace approximation for network trained with mdp is higher than that of Laplace approximation for network trained with cdp in the histogram. Besides, in ROC curve/PR curve on correct prediction vs. OOD data prediction of mdp and cdp, the area under curve of Laplace approximation of mdp is obviously lower than that of cdp. This phenomenon also exists in the histogram and ROC as well as PR curve of mdp and cdp. In the table of quantitative results (cf. 4.2.2), the quantities of different metrics corresponding to the visual metrics. To summarize this, although Laplace can achieve similar results as concrete dropout and multiple dropout do, their performance is highly related to the point estimate of parameter obtained during training.

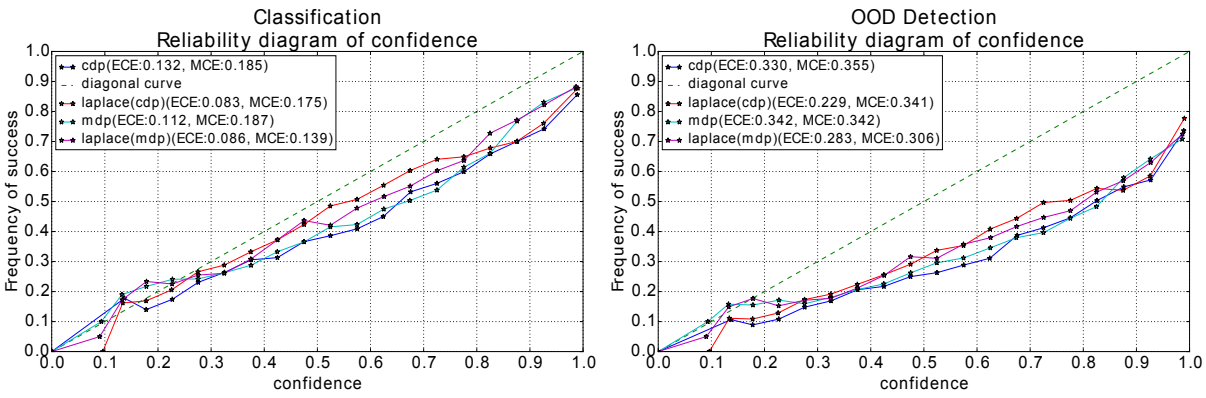


Figure 4.10: Calibration curve of cdp, mdp and their Laplace approximation versions in one of three runs.

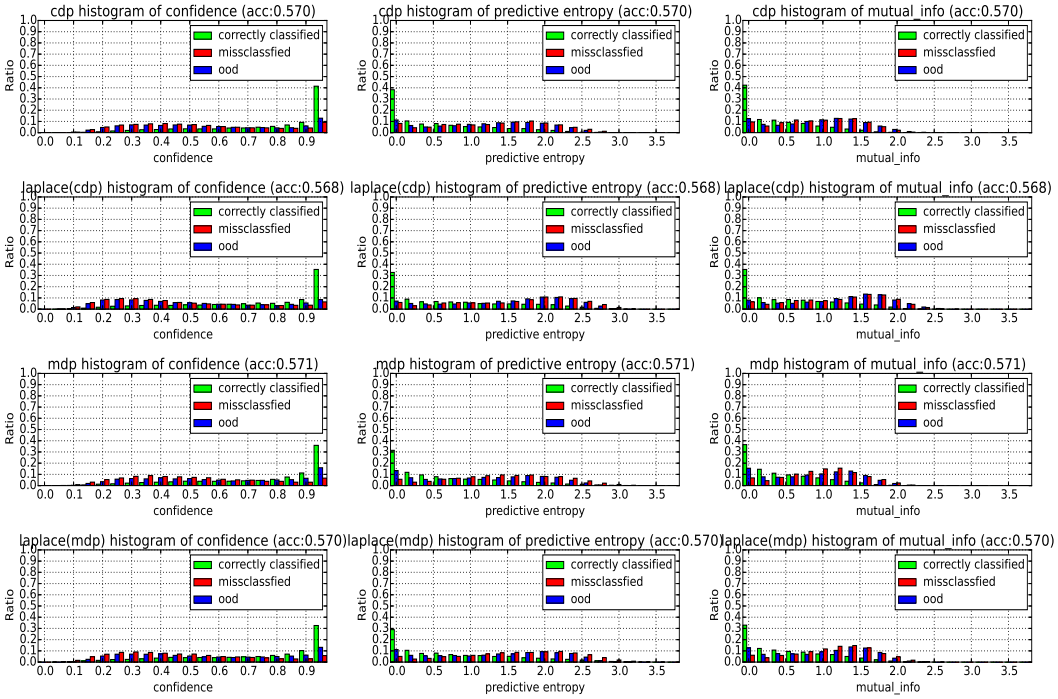


Figure 4.11: Uncertainty histogram of cdp, mdp and their Laplace approximation versions with confidence, predictive entropy and mutual information as uncertainty measure in one of three runs.

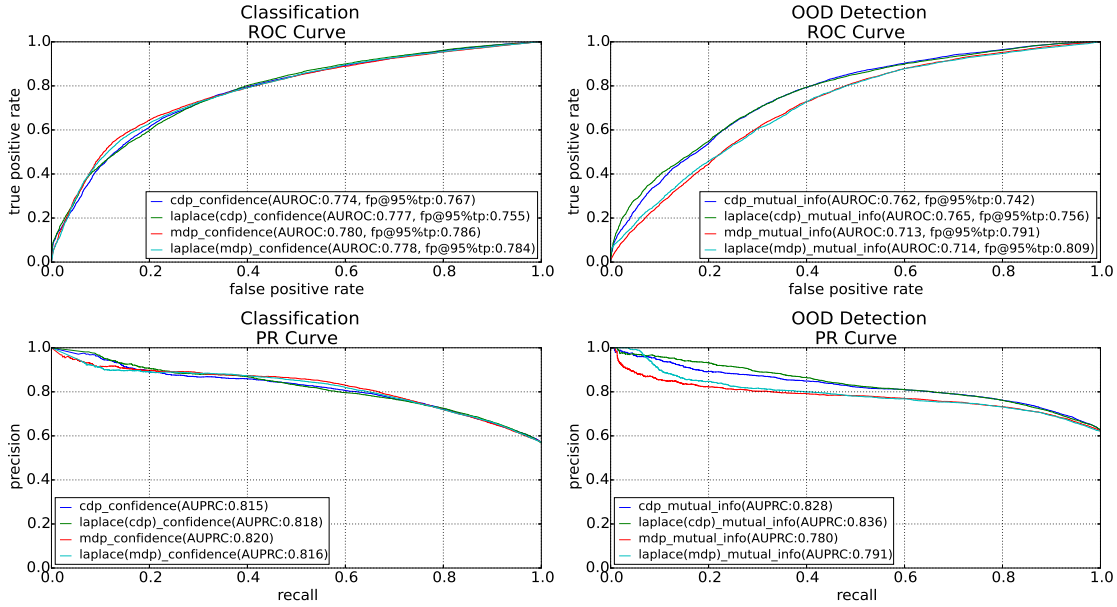


Figure 4.12: ROC and PR curve of cdp, mdp and their Laplace approximation versions in one of three runs.

Table 4.2: Quantitative results of acc, bs, nll, ece, mce, auroc, aupr averaged from 3 different random seeds

	accuracy ↑	brier score ↓	negative log likelihood ↓
cdp	0.577±0.008	0.594±0.013	2.088±0.181
laplace_cdp	0.576±0.009	0.602±0.011	2.322±0.350
mdp	**0.599±0.023**	**0.566±0.020**	**1.940±0.064**
laplace_mdp	0.598±0.024	0.567±0.018	1.970±0.117

	expected calibration error(w/o. OOD/ w. OOD) ↓	maximal calibration error(w/o. OOD/ w. OOD)↓	area under ROC (vs. Miss-classified/ vs. OOD)↑	area under PR curve (vs. Miss-classified/ vs. OOD)↑
cdp	0.124±0.023/ **0.288±0.048**	0.206±0.015/ 0.374±0.018	0.775±0.008/ **0.783±0.022**	0.825±0.007/ **0.850±0.022**
laplace_cdp	0.129±0.058/ 0.341±0.157	0.235±0.073/ 0.406±0.070	0.779±0.004/ 0.782±0.017	0.826±0.007/ 0.849±0.016
mdp	0.114±0.012/ 0.383±0.046	0.199±0.016/ 0.367±0.023	**0.780±0.011**/ 0.709±0.004	**0.838±0.013**/ 0.788±0.006
laplace_mdp	**0.104±0.018**/ 0.352±0.061	**0.179±0.029**/ **0.352±0.038**	0.776±0.012/ 0.711±0.005	0.837±0.015/ 0.798±0.005

Ablation study

As we can see from the previous results, mdp can work better than cdp without considering OOD data. However, it underperforms when OOD data is included. In order to investigate the reason behind this, we have done a ablation study to check the influence of aleatoric uncertainty which comes from mainly the feature extractor part. To this end, we firstly make assumption that the feature extractor part has a dominant effect on the aleatoric uncertainty, because it's deterministic compared with the following MLP classifier.

Considering this, we freeze the parameters of feature extractor trying to keep the dominant effect on aleatoric uncertainty fixed for different approaches and just train our probabilistic classifier which is a three-layer MLP. In the following we show the results of this ablation study with the same metrics used before. It's clear that the accuracy would drop when the feature extractor is frozen. However, we can see that mdp can have better calibration performance than cdp and much better than original ResNet50. Regarding the separability metrics, although mdp still doesn't work better than cdp, the gap between is much smaller than the version without freezing feature extractor. These observations are expressed quantitatively in the table 4.2.2.

Based on these observations, we know that the underperformance of mdp on OOD dataset can attribute to the influence of aleatoric uncertainty. If the feature extractor is not frozen, then the aleatoric uncertainty is trained to adapt to the dataset. The problem why optimizing dropout rates for each hidden units can change the point estimate of the model parameter to have low aleatoric uncertainty for OOD data arises here. One possible reason for this could be the variance of estimation of derivatives of variational parameters is too large because of increased number of variational parameters[KSW15]. And the larger variances are back propagated to the feature extractor which may make the point estimate deviate from the better local minimum. One possible solution to resolve this is to sample the realization of activations of hidden units independent to each data instance, which may decrease the variance and help to improve the results.

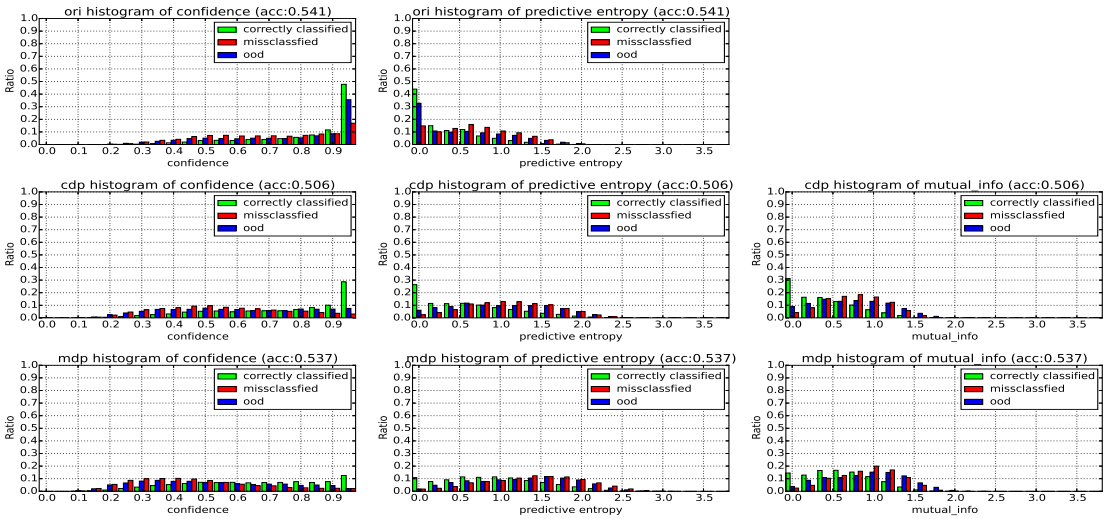


Figure 4.13: Uncertainty histograms of ori, cdp, mdp trained with frozen features in one of three runs.

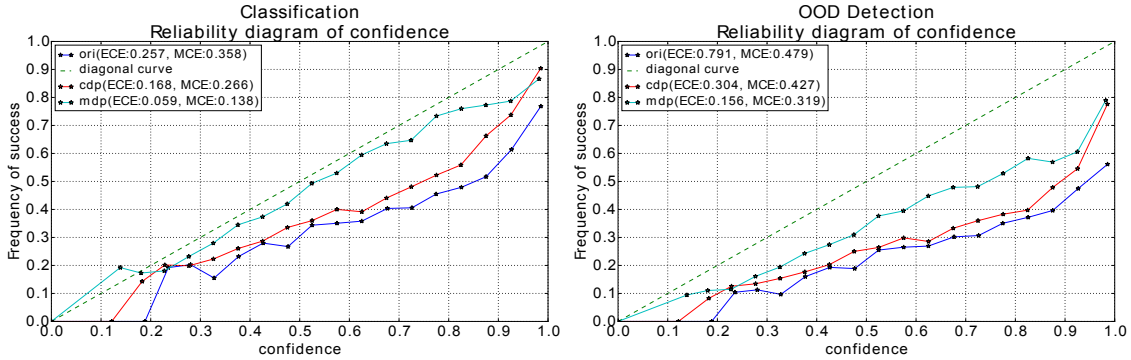


Figure 4.14: Calibration curves of ori, cdp, mdp trained with frozen features one of three runs.

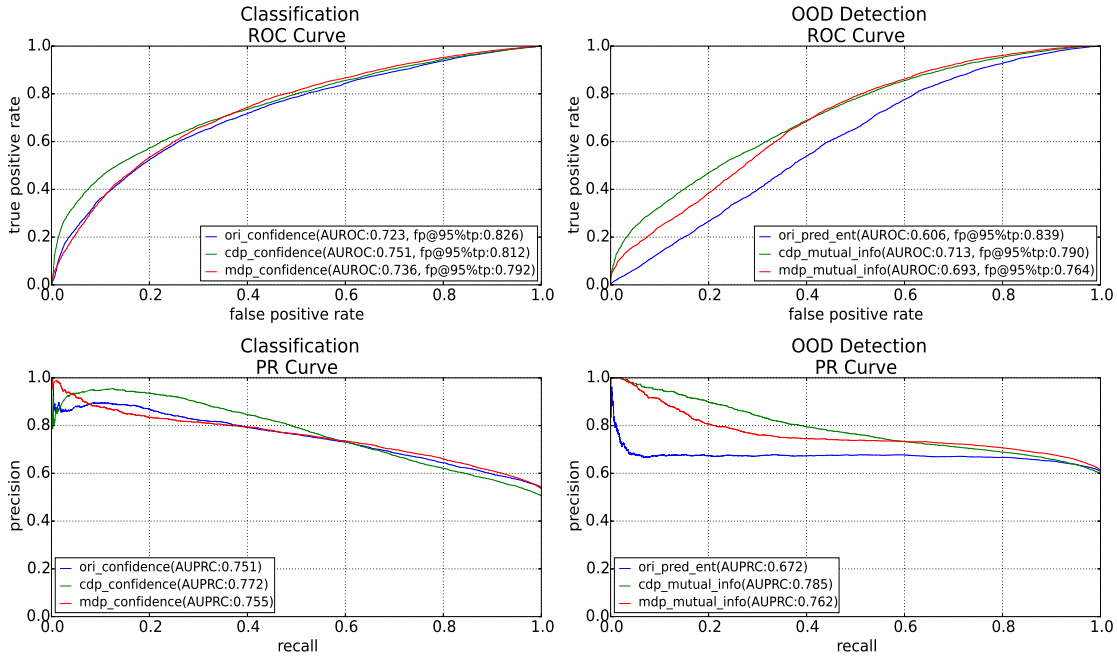


Figure 4.15: ROC and PR curves of ori, cdp, mdp trained with frozen features one of three runs.

Table 4.3: Quantitative results averaged from 3 random seeds.

	accuracy ↑	brier_score ↓	negative log likelihood ↓
ori	**0.532±0.015**	0.717±0.023	2.383±0.053
cdp	0.521±0.011	0.655±0.017	2.190±0.149
mdp	0.520±0.012	**0.641±0.014**	**1.804±0.073**

	expected calibration error(w/o. OOD/ w. OOD)↓	maximal calibration error(w/o. OOD/ w. OOD)↓	area under ROC (vs. Miss-classified/ vs. OOD)↑	area under PR curve (vs. Miss-classified/ vs. OOD)↑
ori	0.262±0.011/ 0.793±0.012	0.365±0.010/ 0.491±0.013	0.716±0.005/ 0.602±0.014	0.724±0.022/ 0.681±0.007
cdp	0.141±0.022/ 0.270±0.037	0.203±0.053/ 0.359±0.056	**0.748±0.002**/ **0.712±0.006**	**0.770±0.002**/ **0.782±0.004**
mdp	**0.068±0.009**/ **0.158±0.006**	**0.134±0.018**/ **0.311±0.008**	0.730±0.009/ 0.674±0.014	0.749±0.009/ 0.748±0.010

4.3 Automatic labeling experiments

In experiments of automatic labeling, our goal is to perform a proof-of-concept experiment, which simulates the situation that a classifier trained on a public large scale dataset or synthetic dataset is deployed in a real world environment and tries to fine-tune itself to improve performance with automatically labeled dataset and as little manually labeled data as possible. The automatically labeled data is collected based on improved uncertainty estimation. Because ResNet50 with concrete dropout shows more stable results than other version, we use it to improve uncertainty estimation in the following experiments and call it **BNN** without specifications.

This way to enable the classifier to learn continuously is seldom seen in the literatures. Therefore, in order to investigate the practicability of this method, we firstly perform experiments on WRGBD and UniHB dataset with restriction on the size of collected dataset.

Then we try to extend this way to another dataset, T-LESS dataset, to test its generalization ability in second experiment. In the second experiment, the situation is similar to the first experiment, where a classifier is firstly trained on synthetic dataset and then deployed in a real world environment which is simulated by the real objects in training set. We employ data augmentation to address the problem of imbalance in fine-tuning classifier found in the first experiment.

4.3.1 Experiment I: evaluation on WRGBD and UniHB dataset

The first experiment is performed on the WRGBD and UniHB dataset. As is mentioned in experiment II of uncertainty estimation experiments, we train BNN with entire WRGBD dataset and evaluate performance of uncertainty estimation on objects of elevation 30° and 60° in UniHB dataset, which we call **adaptation dataset**. The size of adaptation dataset in this experiment is around 17.1×10^3. The objects of 45° with size 8.6×10^3 are used for final evaluation after fine-tuning the classifier. The dataset used for fine-tuning are collected from the adaptation set with different settings in order to investigate the issues of this way for continuous learning.

The uncertainty measure we used is confidence because in this case we do not consider OOD data and confidence performs better in separating correct predictions and missclassifications based on the results of previous experiments.

Automatic Labeling: One major restriction we impose in this experiment is the size of dataset for fine-tuning. Because if we do not restrict the size, the performance is influenced by both the size of dataset and other factors such as imbalance or quality of dataset. In order to exclude the influence of dataset size, we fix it as 3% of the adaptation set which is around 510. If the dataset is balanced in number of data sample of category, then there are around 10 data samples in each category. Then we can investigate how two factors may influence the performance of fine-tuning. These two factors are:

- the balance in number of data sample of each class
- the amount of information of data samples

In the following table 4.3.1, we describe different settings and use capital letter to denote them.

Here we firstly illustrate the steps of **selecting automatically labeled data** in this experiment which is a little different to the strategy in experiment II.

For C, D, E, F, G, we conduct the following steps:

1. Automatic labeling: select the most confident predictions and label them with their predictions.
2. Manual labeling: select the data randomly or with least confidence required for manual labeling.
3. Combining: combine automatically labeled and manually labeled data, then check if any category does not have data sample, if yes add one random data sample of this category.
4. Balancing dataset: firstly check if number of data sample of any categories exceed the average number of data sample (which can be calculated beforehand), if yes, drop the data sample of this category randomly until its number reach the average number of category. After that, if augmentation is chosen, then use augmentation to increase of the number of data sample of category in which the number of data sample is less than average number of data sample.

To note that, the accuracy of automatically labeled data in C, E, F, G is around 96%, which means not all labels of them are correct. And "**randomly**" in this context means that we randomly choose the data of each class with number computed based on the difference between current number of data samples of each class and the average number of data samples of each class. We assume that we have manually labeled all the predictions after automatic labeling and conduct a *proof-of-concept* experiment with **as balanced as possible** dataset in F and G.

Fine-tuning: After collecting the dataset with different settings, we show the results of BNN fine-tuned with those dataset on the final test set which contains only objects of 45° from UniHB data in the table 4.3.1. We can draw the following observations from this table:

- diverse and balanced dataset can achieved best performance.
- the difference of performance between C and D may attribute to quality of dataset which include correct labeling and information of data sample.

Table 4.4: Results of fine-tuned network with different settings

Settings of dataset for fine-tuning	Accuracy (average over 3 random seeds)
A: 0% (no fine-tuning)	66.9%
B: 3% manually labeled data randomly (**balanced**)	91.7%
C: 3% automatically labeled data (**imbalanced**)	79.0%
D: 3% manually labeled data with least confidence (**imbalanced**)	83.3%
E: 2% automatically labeled data and 1% manually labeled data with least confidence, augmentation for balance (**balanced**)	83.8%
F: 2% automatically labeled data and 0.5% manually labeled data with least confidence and 0.5% manually labeled data randomly, augmentation for balance (**balanced**)	88.3%
G: 2% automatically labeled data and 1% manually labeled data randomly, augmentation for balance (**balanced**)	89.6%

- by comparing performance of E, F and G, we know that diverse data sample before augmentation is important to achieve better performance. When considering information of data sample, manually labeling of lest confident data sample sounds better. However, we also know that before augmentation dataset in E is more imbalanced than that in F, and F is more imbalanced than G because of the higher proportion of predictions with least confidence. Therefore it shows that data balance is more important than information of data sample in this case.

By comparing the results of other settings with that of A and B, we can see that the manual labeling effort can be reduced based on automatic labeling. The performance of fine-tuned domain specific classifier can nearly reach the the performance of classifier fine-tuned with all manually labeled data.

In the end, we can draw a conclusion that diversity of data and balance of number of data sample of each category play an significant role. Based on the last aforementioned observation, balance of number of data sample of each category is more important in than information of data sample.

4.3.2 Experiment II: evaluation on T-LESS dataset

In this experiment, we firstly train networks on synthetic and augmented dataset which is introduced previously. Then we select automatically labeled data by testing the trained model on real single objects (original training set of T-LESS) which we call **adaptation dataset**. The difference between the strategy of selecting data in experiment I is that we split off a small balanced validation set from the whole real single object dataset with ratio 2:8. Then we choose the threshold of uncertainty up to which the accuracy can reach 95% based on predictions on validation set. This threshold is employed in the rest of whole adaptation dataset for selecting automatically labeled data. This way to choose threshold makes the simulate situation more realistic and show the practicability of this approach. In this experiment we use predictive entropy as uncertainty measure, because it shows better performance in term of separability metrics on validation set.

The improved uncertainty estimation from BNN plays an important role here, because it not only provides a reliable uncertainty estimation, but also increases the separability between correct predictions and false predictions, which is more useful in this task.

Automatic labeling: We show the comparisons in term of different evaluation metrics between uncertainty estimation from BNN and original ResNet50 in the following figures. As we can see in the uncertainty histogram (cf.4.16), the mode of distribution of uncertainty estimation between correct prediction and miss-classification is further from each other when testing with BNN than with original ResNet50. In figure 4.17, BNN express better performance on both calibration and separability metrics.

With the same procedure of selecting automatically labeled data, we can only obtain around 1.05×10^3 data samples with only 93% accuracy using original ResNet50, but around 1.6×10^3 data samples with 96% accuracy using BNN. Because original ResNet50 produces less automatically labeled data with lower quality, we do not consider using original ResNet50 for collecting fine-tuning dataset and instead focus on using BNN.

However, as we found in experiment I, the problem of imbalance in number of data samples of each class still appears on this dataset. The confusion matrix in figure 4.21 shows this problem obviously, in which most of confident predictions cluster in few classes which are class 6, 7 and 28 in this work. To mitigate this problem, we adopt two simple ways following the procedures in experiment I:

- to manually label data uniformly with size 3% of the adaptation set(around 900 data samples).
- to employ augmentations including occlusions, different background images and so on to the objects to balance the dataset as much as possible.

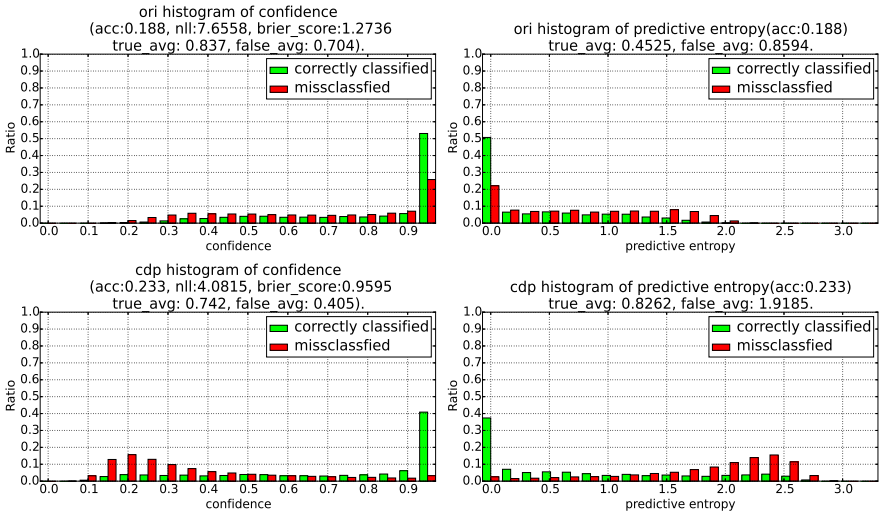


Figure 4.16: Uncertainty histograms of original ResNet50 (top) and BNN (down).

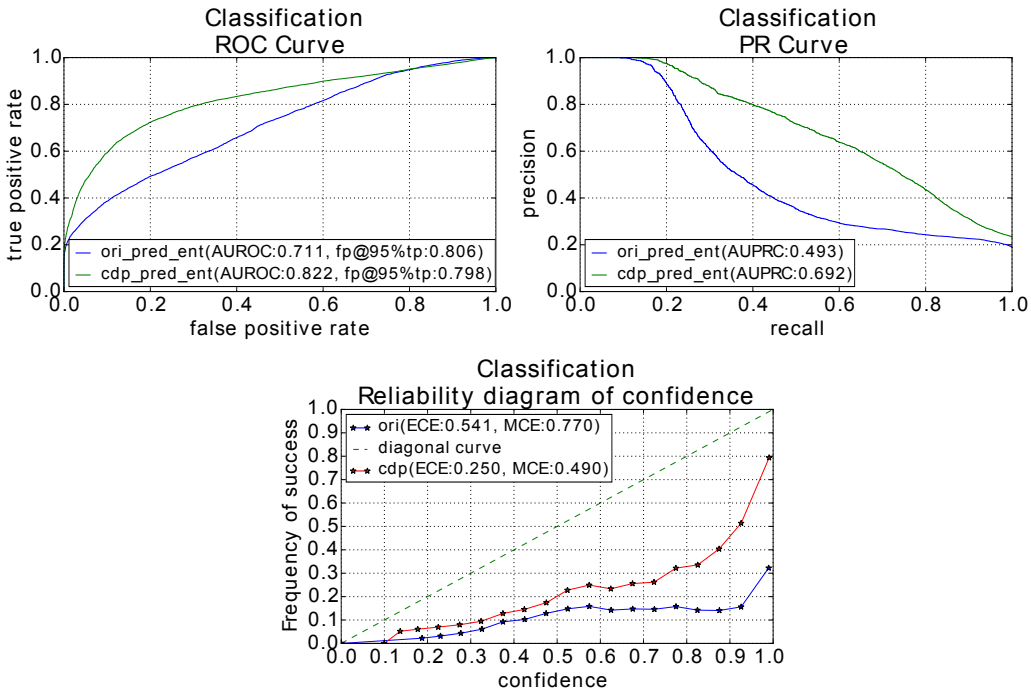


Figure 4.17: Calibration curves, ROC curves, PR curves of original ResNet50 and BNN.

Fine-tuning: After collecting dataset with automatic labeling and small portion of manually labeling, we can use it to fine tune BNN, which was trained previously with only augmented synthetic data. In the following, we show the results of fine-tuned BNN on the original test set of T-LESS dataset and analyze the results with additional tool such as confusion matrix.

Setting of dataset	Accuracy
A: augmented, synthetic dataset	34.91%
B: augmented, entire real dataset (∼30K)	69.58%
C: fine-tune A with augmented, automatically labeled and 3% manually labeled real dataset (∼2.5K)	72.48%
D: fine-tune A with augmented, only automatically labeled real dataset (∼1.6K)	44.99%

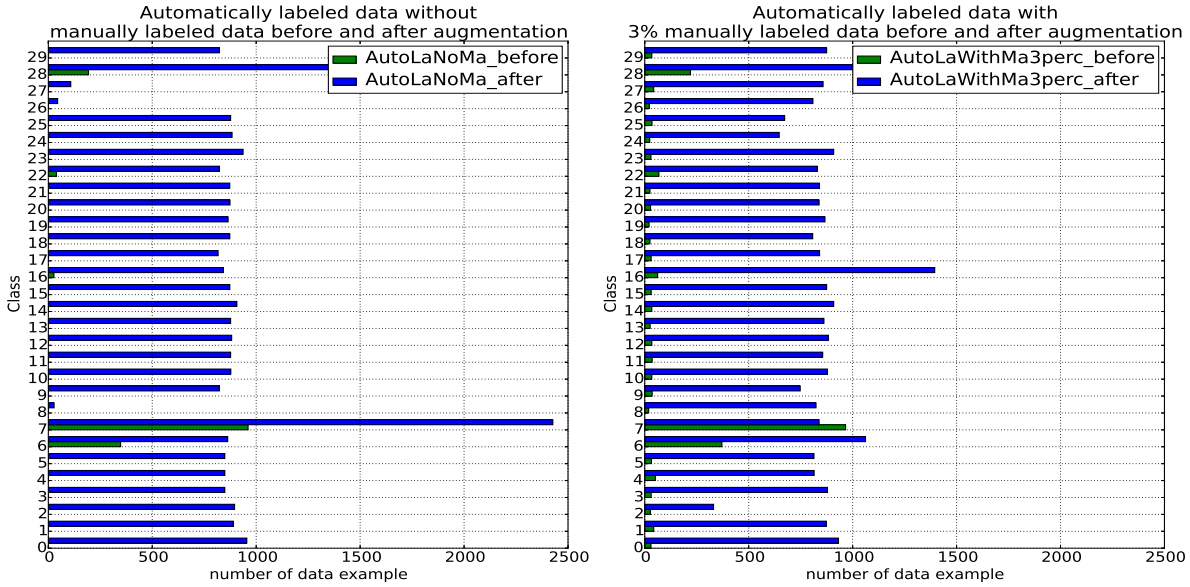


Figure 4.18: Confusion matrix of automatically labeled dataset(label index starts from 0).

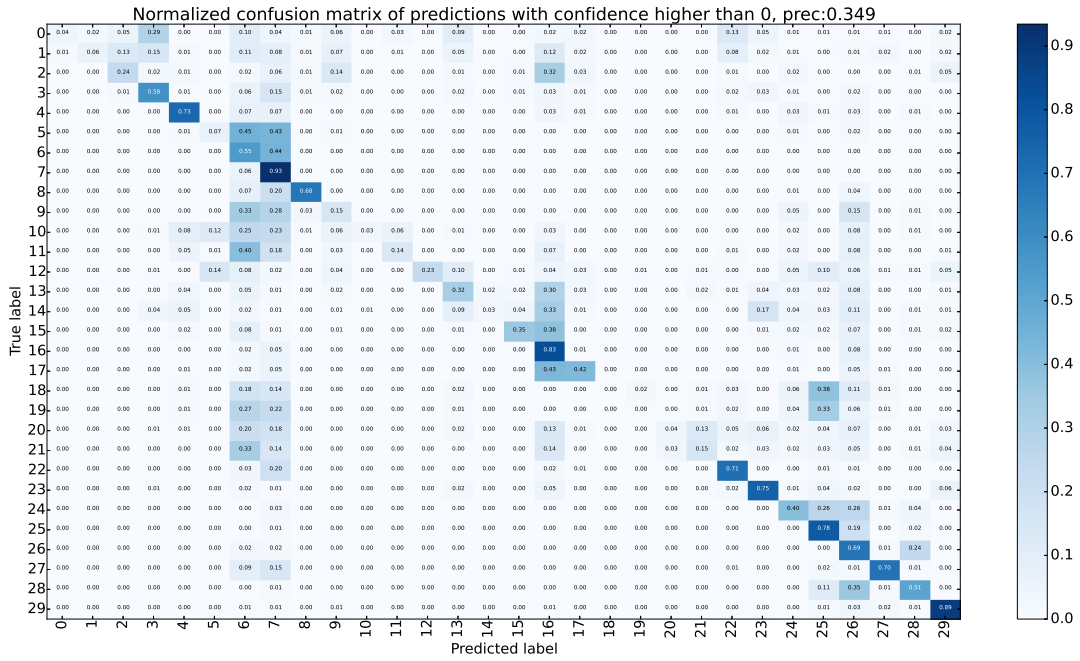


Figure 4.19: Confusion matrix of automatically labeled dataset(label index starts from 0).

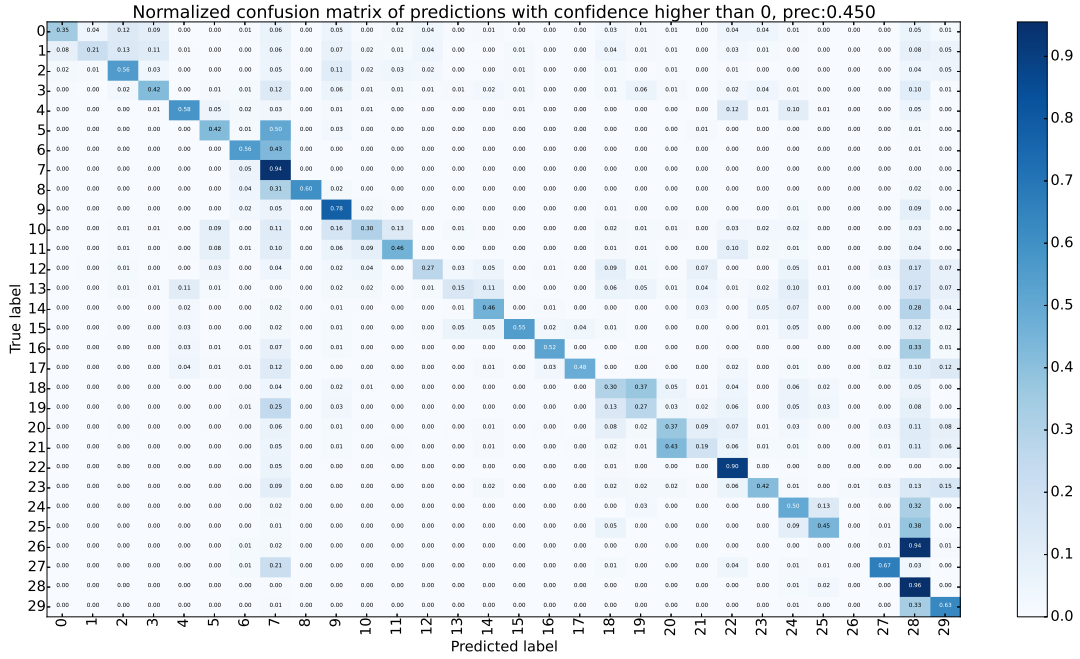


Figure 4.20: Confusion matrix of automatically labeled dataset(label index starts from 0).

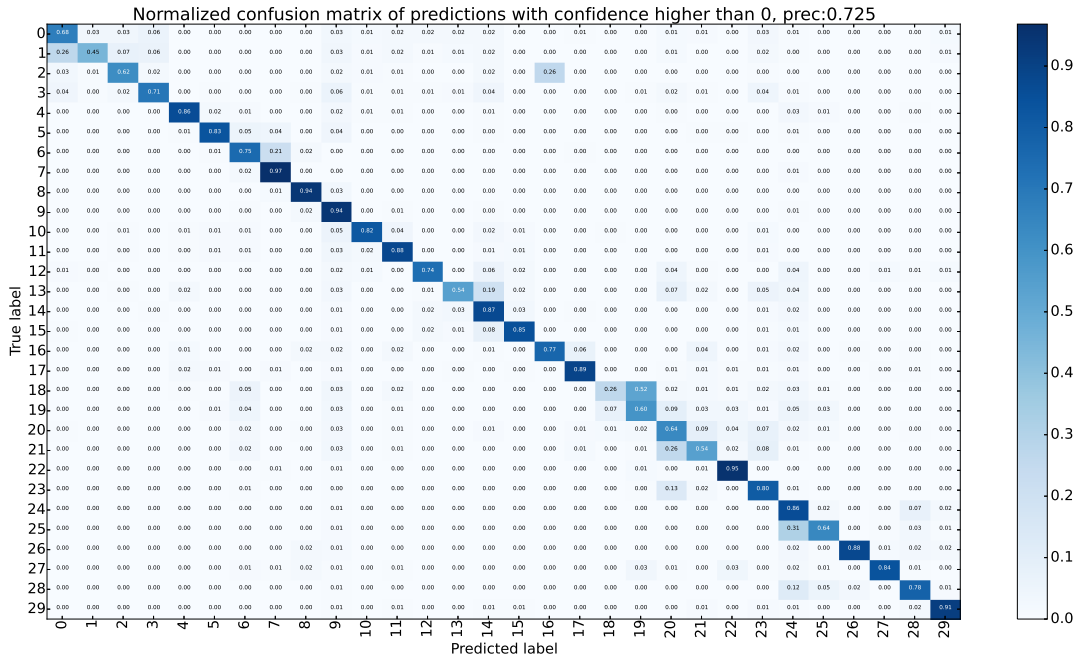


Figure 4.21: Confusion matrix of automatically labeled dataset(label index starts from 0).

4.4 Context-based improvement experiments

In this section, we perform experiments to investigate the effect of improved uncertainty estimation when incorporating them with a probabilistic graphical model in down-stream task. In detail, we employ CRF introduced in chapter 3 to capture the dependencies between objects by exploiting contextual information in the scene. The intuition behind this idea is that if the predictive distribution (used as one uncertainty measure) is more reliable and can represent the inherent randomness better. Then this bit of information can be better utilized by CRF in the down-stream task. Similar to previous experiment, we use ResNet50 with concrete dropout to improve uncertainty estimation in the following experiments and call it **BNN** without specifications.

We have performed mainly two experiments in this part. Firstly we perform a simpler experiment on subset of T-LESS dataset to show that CRF can capture the dependencies and then improve the performance. The performance can be boosted with improved uncertainty estimation from Bayesian neural network. Secondly, we test the same idea on the entire T-LESS dataset based on the output of classifier fine-tuned in the automatic labeling experiment.

4.4.1 Experiment I: evaluation on subset of T-LESS dataset

In this experiment, we train original ResNet50 and BNN on the real single objects with black background which is the original training set of T-LESS dataset. Then we use a subset of test scenes to train and test CRF. In detail, we use scene 1, 2, 3, 4 to train CRF, and scene 5, 6, 7, 8 to test CRF. The background of these scenes are black, thus we do not apply augmentations to the dataset for training the CRF. In experiments, the algorithm always reach the maximum number of iteration during optimization.

In the table 4.5, we show the results of CRF trained with two different unary features from two different versions of ResNet50 and tested on scene 5, 6, 7, 8. As we can see, the network trained and tested with unary feature from BNN can yield more improvement(13.17%) compared with that from deterministic network (12.42%).

Table 4.5: Results of CRF trained with unary feature from different versions of ResNet50

	accuracy with unary potential	accuracy with unary and pariwise potentials
unary feature from ori	48.79%	61.21%
unary feature from BNN	51.94%	65.11%

Additionally, because the weights in our CRF model act as coefficients for two kinds of features, we can check the magnitude of these two weights which can indicate the contribution to overall performance. In the table 4.6, we can see that in CRF trained with unary feature from BNN, the weight of unary feature θ_u is larger than the one of edge feature θ_p. The situation is inverse in CRF trained with unary feature from deterministic ResNet50. This means that the unary feature contributes more than the pairwise one if we use the predictive distribution from BNN. It also shows that the predictive distribution from BNN contains more information than its counterpart from deterministic one. More importantly, this information can be utilized to improve the performance which is shown in the table 4.5.

Table 4.6: Learned weights of CRF trained with unary feature from different versions of ResNet50

	node weight θ_u	edge weight θ_p
unary feature from ori	3.744	4.213
unary feature from BNN	4.635	3.882

4.4.2 Experiment II: evaluation on entire T-LESS dataset

In this experiment, we want to test the generalization ability of CRF on the whole T-LESS dataset.

Type of potential	Accuracy
Unary	72.48%
Unary and pairwise (-1s on diagonal)	73.40%
Unary and pairwise (0s on diagonal)	74.64%
Unary and pairwise (1s on diagonal)	76.50%

Chapter 5

Summary and Conclusion

5.1 Discussions

-

5.2 Conclusions

5.3 Future work

- check uncertainty distribution of different view points of the object.
- try distillation method to decrease runtime of testing.
- try more complex and expressive approximate posterior distribution such as normalizing flow.
- improve performance of multiple dropout.

Appendix A

Appendix

A.1 Fisher information

Here, we consider the case that parameter of likelihood function is scalar. This can easily be extended to the case that θ is a vector. Fisher information $\mathcal{I}(\theta)$ is defined as follows:

$$\mathcal{I}(\theta) = \mathbb{E}\big[\big(\frac{\partial}{\partial\theta}\log f(X;\theta)\big)^2\big] \tag{A.1}$$

Where $f(X;\theta)$ is probability density function or mass function of random variable X. We need to prove that $\mathbb{E}\big[\big(\frac{\partial}{\partial\theta}\log f(X;\theta)\big)^2\big] = \mathbb{E}\big[-\frac{\partial^2}{\partial\theta^2}\log f(X;\theta)\big]$. When we suppose that θ is the true parameter. Based on the fact that $f(X;\theta)$ is density, we can have:

$$\int f(x;\theta)dx = 1$$

Differentiating the above with respect to θ gives:

$$\frac{\partial}{\partial\theta}\int f(x;\theta)dx = 0$$

After moving differentiating operation into integration and making use of derivative of logarithm, we get:

$$\int \frac{\partial \log f(x;\theta)}{\partial\theta} f(x;\theta)dx = 0$$

Differentiating again w.r.t. θ and taking the derivative inside gives:

$$\begin{aligned}
&\int \frac{\partial^2 \log f(x;\theta)}{\partial \theta^2} f(x;\theta)dx + \int \frac{\partial \log f(x;\theta)}{\partial \theta}\frac{\partial f(x;\theta)}{\partial \theta}dx = 0 \\
\Rightarrow &\int \frac{\partial^2 \log f(x;\theta)}{\partial \theta^2} f(x;\theta)dx + \int \frac{\partial \log f(x;\theta)}{\partial \theta}\frac{\partial f(x;\theta)}{\partial \theta}\frac{1}{f(x;\theta)} f(y;\theta)dx = 0 \\
\Rightarrow &\int \frac{\partial^2 \log f(x;\theta)}{\partial \theta^2} f(x;\theta)dx + \int \frac{\partial \log f(x;\theta)}{\partial \theta}\frac{\partial \log f(x;\theta)}{\partial \theta} f(y;\theta)dx = 0 \\
\Rightarrow &- \mathbb{E}[\frac{\partial^2 \log f(x;\theta)}{\partial \theta^2}] = \mathbb{E}[\frac{\partial \log f(x;\theta)}{\partial \theta}\frac{\partial \log f(x;\theta)}{\partial \theta}]
\end{aligned}$$

which gives us the required result.

□

A.2 Implementation details

List of Figures

List of Tables

Bibliography

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[AOS+16] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[BCKW15] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[BRMW15] Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems*, pages 3438–3446, 2015.

[DKD09] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009.

[DL91] John S Denker and Yann Lecun. Transforming neural-net output levels to probability distributions. In *Advances in neural information processing systems*, pages 853–859, 1991.

[DT18] Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

[EVGW+] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[FCSG17] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[FRD18] Di Feng, Lars Rosenbaum, and Klaus Dietmayer. Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection. *arXiv preprint arXiv:1804.05132*, 2018.

[Gal] Yarin Gal. *Uncertainty in deep learning*. PhD thesis.

[GBR07] Tilmann Gneiting, Fadoua Balabdaoui, and Adrian E Raftery. Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(2):243–268, 2007.

[GG16] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[GHK17] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017.

[GIG17] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*, 2017.

[GMR] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models.

[GN99] AK Gupta and DK Nagar. *Matrix Variate Distributions*, volume 104. CRC Press, 1999.

[GPSW17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. *arXiv preprint arXiv:1706.04599*, 2017.

[GR07] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[Gra11] Alex Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356, 2011.

[GRB08] Carolina Galleguillos, Andrew Rabinovich, and Serge Belongie. Object categorization using co-occurrence, location and appearance. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[HC68] John M Hammersley and Peter Clifford. Markov fields on finite graphs and lattices. 1968.

[HG16] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[HHGL11] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

[HHO+17] Tomáš Hodaň, Pavel Haluza, Štěpán Obdržálek, Jiří Matas, Manolis Lourakis, and Xenophon Zabulis. T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[HLA15] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[HVC93] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JGP16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[KC16] Alex Kendall and Roberto Cipolla. Modelling uncertainty in deep learning for camera relocalization. In *2016 IEEE international conference on Robotics and Automation (ICRA)*, pages 4762–4769. IEEE, 2016.

[KFB09] Daphne Koller, Nir Friedman, and Francis Bach. *Probabilistic graphical models: principles and techniques*. 2009.

[KG17] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[KGC] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics.

[KK11] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117, 2011.

[KSW15] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[LAA+17] Christian Leibig, Vaneeda Allken, Murat Seçkin Ayhan, Philipp Berens, and Siegfried Wahl. Leveraging uncertainty information from deep neural networks for disease detection. *Scientific reports*, 7(1):17816, 2017.

[LBRF11] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *2011 IEEE international conference on robotics and automation*, pages 1817–1824. IEEE, 2011.

[LLLS17] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. *arXiv preprint arXiv:1711.09325*, 2017.

[LLS17] Shiyu Liang, Yixuan Li, and R Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[LMP01] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[LPB17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

[LRKT10] Lubor Ladicky, Chris Russell, Pushmeet Kohli, and Philip HS Torr. Graph cut based inference with co-occurrence statistics. In *European Conference on Computer Vision*, pages 239–253. Springer, 2010.

[LSVDHR16] Guosheng Lin, Chunhua Shen, Anton Van Den Hengel, and Ian Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3194–3203, 2016.

[LW16] Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning*, pages 1708–1716, 2016.

[LW17] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

[Mac92] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[MG15] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[MGK+17] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Vivian Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. International Joint Conferences on Artificial Intelligence, Inc., 2017.

[Min01] Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[MMT16] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[MTM14] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Advances in Neural Information Processing Systems*, pages 3086–3094, 2014.

[Nea12] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[OBPVR16] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

[Pea14] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[QCRS+05] Joaquin Quinonero-Candela, Carl Edward Rasmussen, Fabian Sinz, Olivier Bousquet, and Bernhard Schölkopf. Evaluating predictive uncertainty challenge. In *Machine Learning Challenges Workshop*, pages 1–27. Springer, 2005.

[RBB18] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. 2018.

[RSGGJ15] J. R. Ruiz-Sarmiento, Cipriano Galindo, and Javier González-Jiménez. Upgmpp: a software library for contextual object recognition. In *3rd. Workshop on Recognition and Action for Scene Understanding (REACTS)*, 2015.

[RV09] Nikhil Rasiwasia and Nuno Vasconcelos. Holistic context modeling using semantic co-occurrences. 2009.

[RVG+07] Andrew Rabinovich, Andrea Vedaldi, Carolina Galleguillos, Eric Wiewiora, and Serge Belongie. Objects in context. In *Computer vision, 2007. ICCV 2007. IEEE 11th international conference on*, pages 1–8. IEEE, 2007.

[SBD+14] Robin Senge, Stefan Bösner, Krzysztof Dembczyński, Jörg Haasenritter, Oliver Hirsch, Norbert Donner-Banzhoff, and Eyke Hüllermeier. Reliable classification: Learning classifiers that distinguish aleatoric and epistemic uncertainty. *Information Sciences*, 255:16–29, 2014.

[SCC17] Shengyang Sun, Changyou Chen, and Lawrence Carin. Learning structured weight uncertainty in bayesian neural networks. In *Artificial Intelligence and Statistics*, pages 1283–1292, 2017.

[SG18] Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*, 2018.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SM+12] Charles Sutton, Andrew McCallum, et al. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4(4):267–373, 2012.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[WT11] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[WVL+18] Kuan-Chieh Wang, Paul Vicol, James Lucas, Li Gu, Roger Grosse, and Richard Zemel. Adversarial distillation of bayesian neural network posteriors. *arXiv preprint arXiv:1806.10317*, 2018.

[ZSDG17] Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. *arXiv preprint arXiv:1712.02390*, 2017.