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 23, 2019

Place, Date

Signature

München, April 23, 2019

Place, Date

Signature

Abstract

Deep learning based classifiers have achieved tremendous success on different tasks. However, this kind of classifiers cannot provide a reliable uncertainty estimation and express excessive overconfidence, which means that the models are always highly certain even on unfamiliar data. This behavior can easily lead to severe consequences in safety-critical applications such as diagnosis of diseases, perception of self-driving car or robotics. Robots should be aware of the correctness of their predictions to prevent unnecessary accidents and/or to detect failure cases. Furthermore, this kind of introspection can help the robot to recognize unfamiliar environments and to adapt gradually. However, uncertainties of independent data examples do not take into account uncertainty related to dependencies between data examples. Given a classifier with good uncertainty estimations the classification of objects with similar appearances should result in a high uncertainty about its prediction due to the appearance ambiguity, which can still lead to miss-classification. To overcome this issue, contextual information can be used to resolve the ambiguities and improve the initial prediction.

In this work, the uncertainty estimation of deep neural networks are improved by employing Bayesian neural networks. Within this context the applicability and performance of different inference techniques such as dropout variational inference and scalable Laplace approximation are evaluated. The resulting network is applied to a continuous learning use-case where the improved uncertainty estimation is used to select data samples for fine-tuning in an interactive (automatically and manually labeling) manner. For further improvements a conditional random field is employed to fuse the initial predictions with contextual information between objects. Extensive experiments are performed to show that the uncertainty estimation can be improved with Bayesian neural network in terms of different evaluation metrics. Besides, it is experimentally shown that manual efforts for collecting a dataset for fine-tuning a classifier can be reduced with the help of improved uncertainty estimation on two benchmark datasets. Last but not least, experiments also show that the conditional random field is able to further improve the performance by incorporating contextual information.

List of Abbreviations

BNN	Bayesian Neural Network.
CRF	Conditional Random Field.
DNN	Deep Neural Network.
KL-div	Kullbach-Leibler divergence.
LBP	Loopy Belief Propagation.
MAP	Maximum a posterior.
MCMC	Markov Chain Monte Carlo.
MLE	Maximum Likelihood Estimation.
MLP	Multi-layer Perceptron.
OOD	Out-of-distribution.
SGD	Stochastic Gradient Descent.
SGLD	Stochastic Gradient Langevin Dynamics.
VI	Variational Inference.
WRGBD	RGBD Dataset from Washington University[LBRF11].

Contents

Chapter 1

Introduction

1.1 What is uncertainty?

This question seems a little philosophical and difficult to answer. However, in the context of probabilistic modeling in machine learning, one reasonable and illustrative answer can be found in [KFB09]: ”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”. In spite of the innate nondeterminism, the uncertainty can be further categorized into two main types: the **epistemic uncertainty** and the **aleatoric uncertainty** [DKD09, SBD$^+$14, KG17].

The former one, epistemic or model uncertainty illustrates the the uncertainty in modeling, associated with **imperfect models** of the real world due to the insufficient or imperfect knowledge of reality, i.e., it refers to the epistemic state of the decision maker. Therefore it can reduced by collecting more knowledge. One simple example to illustrate this kind of uncertainty is a linear regression with a limited number of observed points, where different curves are obtained during training and then used to predict the unobserved points. After observing enough points in some intervals, the generated models or curves are more restricted, hence the uncertainty decreases. On the other hand, if there are insufficient points in specific intervals, upon which the large number of curves can be obtained to fit them during training, which induces high model uncertainty. In reality, it is common to have inadequate amount of data, and thus hard to model the underlying distribution well and precisely. Therefore the observed data can be explained by a large variety of different models, from where the model uncertainty comes.

The latter one, aleatoric or data uncertainty, accounts for the **inherent randomness** of an underlying phenomenon which is expressed as variability in the observed data and meanwhile associated with the limitation of our observing ability. It is treated as non-reducible within our ability to observe the data. Examples for this kind of uncertainty are the noise induced by limited resolution of sensors or the ambiguity from the content of the data which are unrelated to the model used to represent the data. One simple example can be the classification of images captured by cameras with different resolutions. Images with lower resolutions have the higher uncertainty because they are more ambiguous and less illustrative compared to those with higher resolutions.

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

1.2 Why do we need uncertainty?

The aforementioned model uncertainty represents the belief of a model about its predictions, which brings additional information about the randomness induced by the model compared with the deterministic counter part, which provides only a hard prediction. This kind of uncertainty plays an important role in the two following scenarios.

The first use-cases are those requiring **high safety guarantee** or **optimal decision-making**, in which false predictions can cause hazardous consequences, which is the so-called AI safety[AOS+16]. A reliable model uncertainty estimation can equip the model with the ability to know when the it does not know. More than that, people or operator can take corresponding counter measures to avoid unnecessary accidents. With more and more machine learning algorithms applied in real life applications, this kind of ability meanwhile plays an more and more important role. Concrete examples include the steering control in self-driving cars[MGK+17], disease diagnosis in the medical domain[LAA+17], or even applications in aerospace or other domains requiring higher precision and stronger robustness. One could argue that if the trained model can perform perfectly, thus having achieved 100% accuracy, there is no need for the uncertainty information. [DL91] argue that, it is hard or even impossible to have enough training data to define a precise model, which is in coincidence with reality because it is hard to define a task very precisely and get access to enough samples drawn from its real distribution. The uncertainty information is even more valuable and important when algorithms are confronted with multiple tasks[KGC]. Other relevant tasks such as misclassified and Out-of-distribution (OOD) data detection[HG16] and adversarial attacks [KGB16, FCSG17] have also attracted more and more research interests in uncertainty estimation, aiming to address these challenges.

The second scenario are cases requiring **interaction between algorithms and people or environments**. The model uncertainty can be treated as a bridge between machine learning algorithms and humans. Such interaction or the exploration of the environment can establish a more robust and more data-efficient machine learning system. This scenario involves different kind of fields such as active learning[GIG17], reinforcement learning[BCKW15][OBPVR16][GMR] , automatic labeling, industrial components inspection and so on. The idea behind that is, with reliable model uncertainty estimation we know which data sample the model is unfamiliar with and thus providing more information for the model . In active or reinforcement learning, theses unfamiliar data samples can be used to improve the corresponding model more efficiently. In tasks involving human robot interaction, the uncertainty estimation indicates when the model requires help from human experts. For instance, in industrial component inspection task, firstly a model which can perform quite well on some evaluation datasets is trained. With the assumption that all predictions made by the model are correct, the system is deployed to the real application. However, there is no guarantee that this model represents the real distribution of data perfectly and is robust against a slight or even large domain gap. If the distributions of the real world data vary a lot because of unexpected factors e.g. equipment aging or changes of weather condition, the model could 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 assists 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 a 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 a semantic view point, which we can be also incorporated into training and yield more robust models. This kind of uncertainty is not quantified explicitly but can be incorporated conceptually with the help of specifically designed models and high level statistic cues or features to **disambiguate predictions** directly. One example of this is the modeling contextual information among pixels by a so-called Conditional Random Field(CRF) in the semantic segmentation task[KK11][SM+12][LSVDHR16]. The idea behind that is simple and CRF provides a principle way to achieve this idea. Since uncertainty caused by the appearance ambiguity represents strong correlation between different categories, CRF can model the correlations or dependencies between random variables by taking into account high order potentials. In the task of semantic segmentation, for 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 already known, deep learning is a powerful black box model while it has achieved tremendous success in different tasks. Therefore to obtain and handle uncertainty can help to get a better understanding about this black box model and make deep learning based applications 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, due to the intractability of the real model uncertainty in Deep Neural Network (DNN), there are some **ad-hoc approaches** that try to obtain such a quality measure on specific tasks. [LLS17] combine temperature scaling and input preprocessing which is called adversarial training together and yields a simple but effective OOD sample detector, but this approach requires an additional OOD dataset for training which is hard to collect in reality. [DT18] modify the loss function to train a classifier against OOD data, however this approach is hard to generalize in case of slight domain transfers. [LLLS17] treat a uniform distribution as ground truth distribution of the OOD data and show that OOD data can be generated via a so-called Generative Adversarial Network(GAN). This approach seems practical, but training a GAN effectively does not always work for images because of the high dimensionality of the data distribution.

On the other hand, there are some other more theoretically sound approaches which employ Bayesian Neural Network (BNN)[Mac92][Nea12], bootstrapping[OBPVR16], ensemble methods[LPB17] or even ensemble of [SG18]. Although a BNN provides a principal way to obtain model uncertainty by considering distribution on model parameters, it is difficult to infer the exact posterior distribution over model parameters of DNN because of its complex architecture and large scale training set. Therefore approximate inference plays an important role in addressing this issue. The golden standard for the 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 byStochastic Gradient Langevin Dynamics(SGLD)[WT11], the latter one attracts a lot of research interests like [BRMW15][WVL$^+$18]. However, since it is 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 (KL-div) between the approximate posterior distribution and real posterior. Therefore approximate distribution family needs to be chosen and thus has a large influence on the performance. [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 ”re-parameterization trick”[KW13]. HLA15 also used the same approximate distribution but with expectation propagation[Min01] instead of VI. Since the assumption of fully factorization may impose an strong restriction on the flexibility of the approximate distribution and the correlations between weights can not be captured. 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 with non-diagonal covariance matrix [KSW15] which can capture variances of rows in weight matrices. Another more expressive one is matrix variate Gaussian distribution[LW16][SCC17][ZSDG17], which capture both rows and columns correlations in weight matrices. 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 with help of ”local re-parameterization trick”[KSW15].

Another alternative is the Laplace approximation[Mac92], which puts a Gaussian centered at Maximum a posterior (MAP) estimate of the model parameters with a covariance determined by the inverse of the Hessian matrix of the log posterior distribution. The main issue of this approach for large neural networks is infeasible computation and storage cause by large size of the Hessian matrix which is quadratic of the number of parameters and inversion operation on it whose complexity is cubic over the number of elements of Hessian matrix. Recently [RBB18] address this problem by approximating the Hessian or the Fisher matrix of log-likelihood with kronecked-factorization approximation which reduces the storage size and complexity of inversion operation. The resulting posterior can also be treated as a 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 many practical applications. Therefore the so-called dropout inference[GG16] looks promising considering this restriction and thus is adopted to obtain model uncertainty in this work. On the other hand, scalable Laplace approximation [RBB18] has the potential to resolve 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 a deep neural network training. Therefore another model, **CRF** [LMP01], is adopted based on the predictions of a discriminative neural network classifier. As mentioned in the previous section, the choice of features of second order potentials is a key factor in this problem. In this work, addressing the scene object recognition task, the correlation between objects in specific scene are of importance. The correlation can be represented by a co-occurrence matrix [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 BNN to obtain reliable and calibrated uncertainty. In order to keep the ability of powerful feature extraction of DNN, we modify a classifier with ResNet50[HZRS16] as backbone to incorporate different aforementioned inference techniques including dropout variational inference and scalable Laplace approximation. We evaluate prediction and calibration performance of BNN and then select predictions with low uncertainty/high confidence as automatically labeled data based on improved uncertainty estimation. This labeled dataset collected with little manual efforts is used for training a more accurate classifier while the performance of original one drops a lot in case of slight domain transfer caused by possibly by changes of light condition, changes of appearance and even changes of recording equipment. 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 more accurate classifier in case of slight domain transfer during deployment of robots.

The second part is to handle the uncertainty caused by appearance ambiguity properly. 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 this end, CRF is selected to capture the contextual relationship between objects with similar appearance but different contexts and at the same time utilize the information brought by better uncertainty estimation obtained in the first part. Key factor to make CRF to work is to design informative pairwise features which should not contradictory to unary feature. In this work, binary co-occurrence matrix of categories as pairwise features are employed.

The main contributions of this work are:

- to show that dropout variational inference and scalable Laplace approximation can obtain better uncertainty information from a large and complex network on a multi-class classification problem with 51 classes, which can be further improved by learning dropout rates during training and their ensemble.
- to show that manual effort in data collection in a slight domain transfer learning task can be reduced by making use of reliable uncertainty estimation. With help of this, classifier can learn continuously in test environment which has domain gap to the training set.
- to show that CRF can improve the results from discriminative classifier by handling uncertainty caused by appearance ambiguity and utilizing information brought by better uncertainty estimation.

The structure of this work is as follows. In order to give the reader a clear theoretical background of techniques applied in this work, the basic theory and ideas of Bayesian neural network and inference techniques such as dropout variational inference and scalable Laplace approximation, and the theory of CRF are briefly reviewed in chapter 2. In chapter 3, technical approaches proposed and employed in this work, including proposed variants of concrete dropout, modified network architecture, combination of BNN and CRF as well as pipeline of continuous learning are introduced. In chapter 4, extensive experiments are performed to evaluate performance of uncertainty estimation of BNN with different inference techniques. In the following, experiments on performance of continuous learning and improvements brought by CRF are performed. Then summary of this work, conclusion and possible future work are presented in chapter 5.

Chapter 2

Theoretical Background

2.1 Bayesian neural network

In this chapter, a brief review on theory of BNN is given. Since calculating the exact posterior distribution over the parameters of large and complex networks is intractable, proper techniques to perform approximate inference are required. Those techniques can be categorized into two main mainstreams, VI and MCMC. However, since larger dataset and more complex neural network are popular nowadays, the traditional approximate inference techniques are difficult to scale to large datasets and complex architectures. Therefore modern approximate inference techniques need to be adopted. In this section, two modern approximate inference techniques for BNN including dropout variational inference[GG16] and scalable Laplace approximation[RBB18] are introduced.

2.1.1 Introduction

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

In supervised learning, our goal is to learn a distribution of the output conditioned on the input, $\boldsymbol{p}(\mathbf{y}|\mathbf{x})$ that can explain the underlying data distribution based on the observables $\boldsymbol{\mathcal{D}}$ which are samples drawn from the underlying data distribution. Then this 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:

$$\boldsymbol{p(y = d|\mathrm{x}, \omega) = \frac{exp(f^{\omega}_{d})}{\sum_{d'} exp(f^{\omega}_{d'})}} \tag{2.1}$$

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

In the regression case, the likelihood is the Gaussian distribution:

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

with the model precision $\boldsymbol{\tau}$, 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 the prior distribution of the model parameters $\boldsymbol{p(\omega)}$. The probability distribution of the model parameters is updated from the prior into the posterior after observing the training dataset $\boldsymbol{\mathcal{D}}$ via Bayes' theorem:

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

where $\boldsymbol{p}(\mathbf{Y}|\mathbf{X}) = \int \boldsymbol{p}(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})\boldsymbol{p}(\boldsymbol{\omega})\boldsymbol{d\omega}$ is the so-called model evidence or marginal likelihood, whose integration is always intractable when the dimensionality of parameters is too high.

After obtaining the posterior distribution over the model parameters, predictions can be obtained by marginalizing the likelihood of unseen input points such as $\boldsymbol{x}^{\star}$ over the model parameters, which leads to a predictive distribution over the output:

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

Explaining the difference between the deterministic neural network and the BNN can help understanding the mechanism of BNN better. In Fig.2.1, graphical model is used to express these two kinds of neural network, in which the solid point denotes the deterministic variable, the circle denotes the random variable, while the shaded circle denotes the observed random variable. The plate notation denotes $\boldsymbol{N}$ observed data pairs $(\boldsymbol{x},\boldsymbol{y})$. Fig.2.1 demonstrates that parameters $\boldsymbol{\omega}$ in the deterministic neural network is normal variable which is a point estimate obtained by MAP:

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

On the other hand, in BNN, the model parameters $\boldsymbol{\omega}$ are random variables with the distribution function parameterized by $\boldsymbol{\theta}$. To note that, for explanation of concept here, no simplified assumptions are made on the function parameterized by $\boldsymbol{\theta}$, which means it can be a arbitrary function. This distribution over model parameters can be inferred based on the 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. An approximation needs to be employed to solve this problem. It's worth noting that if we choose the approximate distribution to be a delta function, $\boldsymbol{q_{\theta}(\omega) = \delta(\omega - \theta)}$, the corresponding BNN is recovered into the deterministic neural network. Because the computation of model evidence $\boldsymbol{p(\mathbf{Y}|\mathbf{X})}$ is always intractable in a complex model and a large dataset, approximate inference needs to be applied, which is briefly introduced in next section.

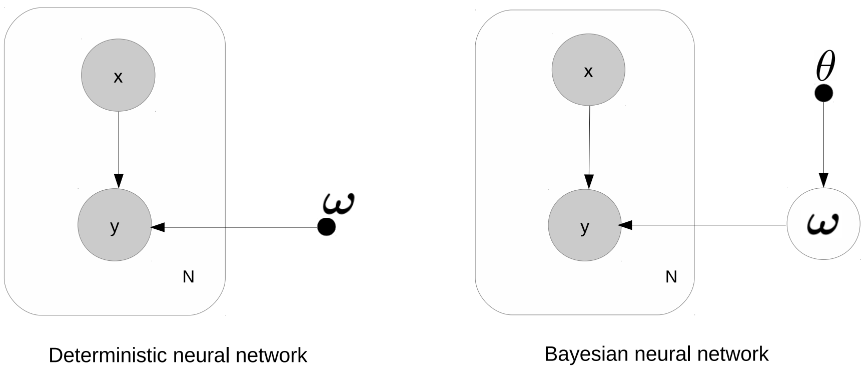

Figure 2.1: Differences between parameter estimation of the deterministic neural network and the BNN. The solid point denotes deterministic variable, and the circle denotes random variable, while the shaded circle denotes observed random variable. The plate notation denotes $\boldsymbol{N}$ observed data pairs $\boldsymbol{(x, y)}$.

2.1.2 Dropout variational inference

VI casts inference into optimization by minimizing the KL-div between the approximate posterior distribution and the real posterior distribution. However, there is no analytical definition of this KL-div if the real posterior distribution is unknown. A lower bound can be derived with Jensen's inequality for tractable optimization, which is the so-called evidence lower bound($\boldsymbol{ELBO}$) which bounds the log marginal likelihood. And from that, the marginal likelihood is the sum of $\boldsymbol{ELBO}$ and KL-div between the approximate posterior and the real posterior. The derivation is given in the following:

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

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

The $\log(\boldsymbol{p(Y)|X})$ can be obtained by calculating the sum of $\boldsymbol{ELBO}$ and KL-div between the approximate posterior $\boldsymbol{q_\theta(\omega)}$ and the real posterior $\boldsymbol{p(\omega|X,Y)}$:

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

When maximizing the $\boldsymbol{ELBO}$ w.r.t the parameter of approximate posterior $\boldsymbol{\theta}$, which is also called **variational parameter**, it is equivalent to minimizing the KL-div because the log marginal likelihood is not a function of $\boldsymbol{\theta}$. Then a well-defined objective $\boldsymbol{ELBO}$ comes out, 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 the regularization term which ensures the approximate posterior does not deviate too far from the prior. Now the inference problem has been cast into an optimization problem:

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

which is equivalent to

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

However, there are still difficulties solving this optimization problem with this objective. The first one is to deal with large size datasets, which induce too much computations in the expected log likelihood term. [Gra11] shows that this can be solved by data sub-sampling which is similar as the stochastic optimization. Another one is the need of the derivatives of $\boldsymbol{ELBO}$ w.r.t. the variational parameters $\boldsymbol{\theta}$. Since the model parameters are samples from the approximate posterior, a good estimator for the derivatives is required which are introduced in the next subsection.

Dropout variational inference

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

The mechanism of dropout is straightforward, each units of specific layer is multiplied by a random variable under Bernoulli distribution with $\boldsymbol{p}$ as its parameter, where $\boldsymbol{1-p}$ is the so-called dropout rate. In each iteration during training, the 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 within the same iteration. In the testing, the sampling phase is turned off, only one forward propagation is needed to produce predictions. Normally, in order to avoid rescaling weights in the testing, which is used to keep the output magnitudes in the same scale when dropout is off, rescaling of the output of the dropout is always done in training. In Fig.2.2, there are two figures showing how the dropout is turned on and off.

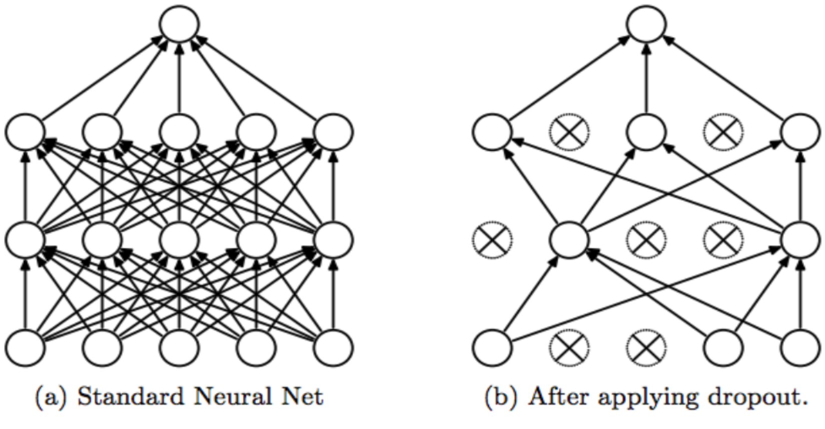

Figure 2.2: Schematic illustration of dropout in a three-layer fully connected neural network. (a) the standard neural network with all neurons switched on. (b) after applying dropout only a subset of neurons are used[SHK$^+$14].

Bayesian interpretation of dropout: In VI, the goal is to minimize the KL-div between the approximate posterior and the real posterior over the model parameters. This objective is equivalent to the maximization of the evidence lower bound($\boldsymbol{ELBO}$). 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 $\boldsymbol{ELBO}$ 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 Fig.2.3, in which we define the hidden layer as the first layer and output layer as the second layer and assume a fully factorized Gaussian prior over parameters.

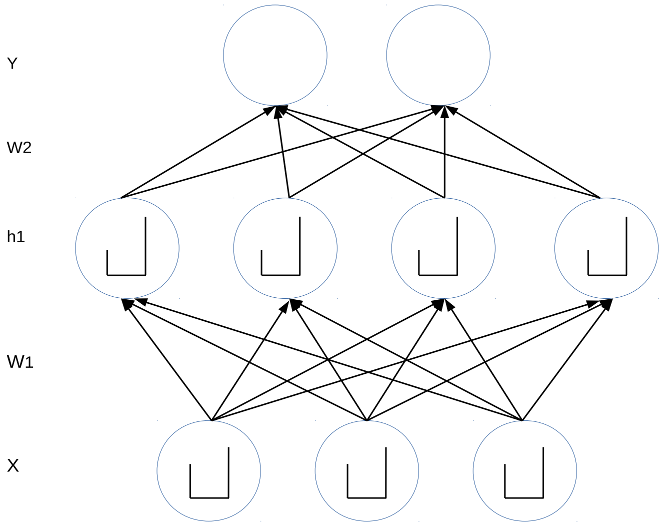

Figure 2.3: An example of two layer neural network with dropout, a Bernoulli random variable is imposed on each unit of input layer and hidden layer.

Approximate distribution: Fig.2.3 shows a simple example of DNN with dropout, in which the following variables are defined: $\boldsymbol{y} \in \mathbb{R}^{\boldsymbol{m} \times \boldsymbol{D_2}}$ as the output layer, $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{m} \times \boldsymbol{D_0}}$ as the input, $\boldsymbol{h_1} \in \mathbb{R}^{\boldsymbol{D_1}}$ as the response of the hidden layer, where $\boldsymbol{m}$ represents the number of data instances, and $\boldsymbol{D_i}$ is the dimensionality of the $\boldsymbol{i}$-th layer, where the $\boldsymbol{0}$-th layer represents input layer and $\boldsymbol{i} \in \{\boldsymbol{1}, .., \boldsymbol{L}\}$, $\boldsymbol{L} = \boldsymbol{2}$. Further $\boldsymbol{\omega} = \{(\boldsymbol{W_i})_{\boldsymbol{i=1}}^{\boldsymbol{L}}\}$ is defined as model parameters, and $\boldsymbol{\epsilon_i} \in \mathbb{R}^{\boldsymbol{D_{i-1}}}$ as the Bernoulli distributed random vector parameterized by $\boldsymbol{p_i} \in \mathbb{R}^{\boldsymbol{D_{i-1}}}$, for $\boldsymbol{i}$-th layer, which we call **keeping rate** in this report. The corresponding dropout rate is equal to $\boldsymbol{1} - \boldsymbol{p_i}$. It is assumed that a fully factorized Gaussian prior distribution is placed over the weights. These assumptions can easily be generalized and transfered to a multi-layer neural network easily.

In naive version of the dropout, each element of vector $\boldsymbol{p_i}$ has the same value as $\boldsymbol{p_i}$, which means that there is only one keeping rate for each layer. In the following, if there is no further specification, $\boldsymbol{p_i}$ and $\boldsymbol{p_i}$ are used interchangeably depending on the context. Because the weights in weight matrix are treated as random variables, $\boldsymbol{M_i} \in \mathbb{R}^{\boldsymbol{D_{i-1}} \times \boldsymbol{D_i}}$ is used to denote position of non-zero element in Bernoulli distribution for $\boldsymbol{W_i}$. Additionally, the bias $\boldsymbol{b_i} \in \mathbb{R}^{\boldsymbol{D_i}}$ is absorbed into $\boldsymbol{W_i}$ by appending a new row at the end of weight matrix and 1 at the end of each data input vector, also called homogeneous coordinate. Another assumption is that the approximate weight distribution is factorized over the layer, which yields

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

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

$$\begin{aligned} \boldsymbol{p(y|x,\omega)} &= \boldsymbol{\sigma((h_1 \odot \epsilon_2)M_2)} \\ &= \boldsymbol{\sigma(h_1(diag(\epsilon_2)M_2))} \\ &= \boldsymbol{\sigma(h_1 W_2)} \\ &= \boldsymbol{\sigma(\phi(x(diag(\epsilon_1)M_1) + b_1)W_2)} \\ &= \boldsymbol{\sigma(\phi(xW_1)W_2)} \end{aligned} \tag{2.10}$$

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

From the equation above, it can be derived:

$$\begin{gathered} \boldsymbol{W_i = g(M_i, \epsilon_i) = diag(\epsilon_i)M_i} \\ \text{with } \boldsymbol{\epsilon_i \sim p(\epsilon_i) = Bernoulli(p_i)} \end{gathered}$$

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

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

with

$$\begin{aligned} q_{\theta}(W_i|\epsilon) &= \delta(W_i - g(M_i, \epsilon_i)) \\ &= \delta(W_i - diag(\epsilon_i)M_i) \end{aligned} \tag{2.12}$$

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

$$\mathbb{E}_{q_{\theta}}(W_i) = M_i \odot P_i \tag{2.13}$$

where $P_i = [p_i, ..., p_i] \in \mathbb{R}^{D_{i-1} \times D_i}$.

The covariance matrix of the parameter matrix is:

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

where

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

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

which is the linear mapping between the element index before and after the matrix vectorization. The element of the l-th row and the q-th column in matrix M_i is defined as m^i_{lq}, which applies to m^i_{mn} as well. The indicator function is denoted as $\mathbb{1}\big[i = j\big]$, which is equal to 1 only when i is equal to j and otherwise 0. Because $vec(\cdot)$ operation converts matrix $W_i \in \mathbb{R}^{D_{i-1} \times D_i}$ into a column vector $vec(W_i) \in \mathbb{R}^{D_{i-1}D_i \times 1}$ by stacking the columns of matrix on top of each other. Then it is easy to obtain covariance matrix $Cov_{q_{\theta}}(vec(W_i)) \in \mathbb{R}^{D_{i-1}D_i \times D_{i-1}D_i}$.

From Eq.2.14, it can be seen 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 the covariance matrix of the parameter matrix W_i w.r.t. the approximate posterior $q_{\theta}(W_i)$, it is known that the samples of row vectors in the weight matrix are drawn independently and thus the covariance between rows are zero. On the other hand, the 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 the weights within the same row are induced. Therefore, weights within the same row can be learned by fitting this approximate distribution to the real posterior. This means that the approximate posterior family has more flexibility to approximate the real posterior when compared with other common approximate posterior distribution family such as fully factorized Gaussian that assumes distribution for each parameter is independent.

Training objective: Up to now, only the approximate distribution of dropout inference is explained. As mentioned in introduction section of this chapter, it is known that by performing an optimization of $\boldsymbol{ELBO}$ w.r.t. variational parameters, a good approximation to the true posterior distribution over parameters can be obtained. This posterior can be used in testing to produce more reliable uncertainty estimation. In the following, training a DNN with dropout is equivalent to optimize the $\boldsymbol{ELBO}$ w.r.t. the variational parameters is illustrated.

At first, the training objective of DNN with dropout is defined as the cross entropy between the predictive distribution and the target distribution plus L2 regularization:

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

where $\boldsymbol{N}$ represents the size of entire dataset, $\boldsymbol{\lambda}$ is L2 regularization coefficient. The likelihood over the entire dataset should be maximized w.r.t. the model parameter $\boldsymbol{\omega}$. This estimation of the model parameters is known as Maximum Likelhood Esitmation (MLE). Equipped with L2 regularization and Gaussian prior over parameters, a MAP estimation is attained by minimizing Eq.2.15.

Normally gradient descent is used to tune the parameters during training, which requires first derivative of the objective w.r.t. the model parameters $\boldsymbol{\omega}$. Since large size of dataset is ubiquitous, which means $\boldsymbol{N}$ in Eq.2.15 is too large, exact gradients of whole batch (entire training set) is hard to calculate exactly and efficiently. Therefore many mini-batches of data are used to estimate gradients. This is the so called Stochastic Gradient Descent (SGD). Apart from making computation tractable, the noise of gradients estimations in each mini-batch is helpful for the optimization procedure to escape from the poor local minimum. The expression of gradients required in each iteration of SGD is given in the following:

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

where $\boldsymbol{S}$ is a random subset of the entire dataset, and $|\boldsymbol{S}| = \boldsymbol{K}$.

On the other hand, when looking at the $\boldsymbol{ELBO}$ in Eq.2.6, to maximize $\boldsymbol{ELBO}$ is equivalent to minimize negative $\boldsymbol{ELBO}$ with SGD w.r.t. the variational parameter $\boldsymbol{\theta}$. The gradients are computed with the following expression:

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

In order to calculate two terms in Eq.2.17, re-parameterization trick[KW13] needs to be introduced for the first term and the condition called KL condition[Gal] is introduced for the second term.

Re-parameterization trick: when taking a close look at the first term in Eq.2.17, it's required to compute the gradients of expectation w.r.t. the parameters of distribution to which this expectation is subject. That means, the gradients of variational parameters are random because our objective is evaluated with the realizations of these parameters. Fortunately, there are different approaches to estimate the gradients of variational parameters such as score function or REINFORCE estimator[Wil92], re-parameterization trick [KW13] and so on. As is stated in [KW13], the re-parameterization trick has lower variance than the score function estimator. There is a quick recap of this technique given and it can be seen that it is already a built-in part in DNN with dropout.

To identify the problem, a more general form is given in the following, the expression of derivatives of expectation w.r.t. the parameters upon which the distribution is defined:

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

Assuming that $p_\theta(x)$ can be re-parameterized as $p(\epsilon)$ which is a parameter-free distribution such that the 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 the estimator of the gradients w.r.t. the distribution parameters θ is as follows, the derivation of it can be referred in A.2:

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

From practical point of view, the expectation of last line in Eq.2.19 can be approximated with the Monte Carlo integration. In the dropout inference, it is known that if the dropout rate is fixed to $1 - p_i$ in Eq.2.12 during training, ϵ_i in Eq.2.12 is a parameter-free random variable and $g(\cdot)$ is a differentiable function w.r.t. its input arguments M_i. Now the variational parameters θ only contain $\{(M_i)_{i=1}^{L}\}$, which are exactly the weights to be learned in training DNN with dropout. As a consequence, if the gradient of $ELBO$ w.r.t. $\{(M_i)_{i=1}^{L}\}$ can be computed, which is the first term in Eq.2.17, it is equivalent to calculate the gradients of the dropout loss w.r.t. the model parameters ω 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:

$$p(\omega) = \prod_{i=1}^{L}(p(W_i))$$

with

$$p(vec(W_i)) = \mathcal{N}(0, l^{-2} I_{D_{i-1}D_i})$$

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

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

where $\mathrm{w}_{i,j} \in \mathbb{R}^{D_i}$ is the j-th row of weight matrix 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-div between mixture of Gaussian and a single Gaussian. Reader who has interest can refer to derivation of KL-div between mixture of Gaussian and single Gaussian in [Gal]. KL condition is given in the following:

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

with

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

for large enough D_i. If we **fix dropout rate** in training and compute the gradients of KL divergence w.r.t. variational parameters $\theta = \{(M_i)_{i=1}^{L}\}$, which is now exactly the same as the model parameters $\omega = \{(W_i)_{i=1}^{L}\}$. 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):

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

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

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

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

where $\omega \sim q_\theta(\omega)$ and $\hat{\omega}_i$ is one of K realizations of ω, 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 contain $\{(M_i)_{i=1}^{L}\}$, which are equivalent to $\omega = \{(M_i)_{i=1}^{L}\}$ in DNN with dropout. Therefore optimizing any DNNs with dropout is equivalent to perform approximate inference on BNN with the same architecture. Thus the optimal weights learned during the optimization of DNN with dropout are the same as the optimal variational parameters of the corresponding BNN.

However, selecting dropout rate before training is not a trivial task for several reasons. Firstly, as shown in [SHK+14], different dropout rates have different influences on model capacity and thus model performance. To choose an optimal dropout rate manually requires repeating tedious experiments which induces a lot of time consumptions and computation efforts. Secondly, if we want the model not only to achieve satisfied accuracy but also to produce reliable uncertainty estimation. The dropout rate should matter a lot because it can influence the flexibility of approximate distribution family from the perspective of its 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 is not easy and intuitive to estimate gradients of expectation w.r.t. parameters of a discrete distribution. Before when the approximate distribution is continuous, gradients can be estimated 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 or categorical distribution, 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, which is called concrete distribution, categorical re-parameterization or Gumbel-softmax trick. In this trick, "max" operation in Gumbel-Max trick is replaced by "softmax" function, which can yield 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: Gumbel-Max trick[MTM14] is shown in Fig.2.1.2, which is used for drawing samples from a discrete distribution which is parameterized by set of unnormalized probability $\{\boldsymbol{\alpha_i}\}_{i=1}^{n}$ via inverse cumulative distribution function of Gumbel distribution, where $\boldsymbol{\alpha_i} \in \mathbb{R}_{>0}$ and $\boldsymbol{n}$ denotes the number of class. Assuming that we use one-hot encoding vector for label representation, that is $\boldsymbol{d} \in \{0,1\}^{n}$ and $\sum_{i=1}^{n} \boldsymbol{d_i} = 1$. The Gumbel-Max trick proceeds as follows(cf. Fig.2.1.2):

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

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

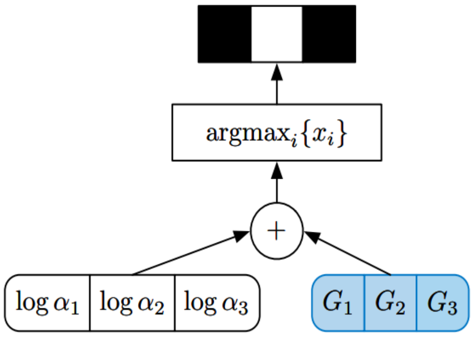

Figure 2.4: Example of Gumbel-max trick for drawing samples from a discrete distribution whose random variable has 3 states and $\{\boldsymbol{\alpha_i}\}_{i=1,2,3}$ as class parameters representing the possibility of occurrence of that class. $\{\boldsymbol{G_i}\}_{i=1,2,3}$ are i.i.d Gumbel$(\mathbf{0,1})$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

As observed in Gumbel-Max trick, the sampling step is re-parameterized 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 $\boldsymbol{argmax}$ operation. However, $\boldsymbol{argmax}$ operation is not differential w.r.t. distribution parameter $\boldsymbol{\alpha_i}$.

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

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

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

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:

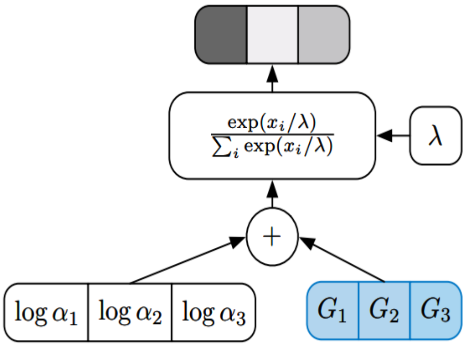

Figure 2.5: Example of continuous approximation to Gumbel-max trick for drawing samples from a discrete distribution whose random variable has 3 states and $\{\boldsymbol{\alpha_i}\}_{\boldsymbol{i=1,2,3}}$ as class parameters representing the possibility of occurrence of that class. $\{\boldsymbol{G_i}\}_{\boldsymbol{i=1,2,3}}$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

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

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

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

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

In order to make explanation more clear and validate the continuous approximate re-parameterization, one plot (Fig.2.6) is given to show 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 increasing step of 0.1. It can be seen that the empirical average of samples drawn from Eq.2.25 is close to expectation of Bernoulli distribution of corresponding parameter. Additionally, one single sample in the figure is shown. As can be seen, most of samples are lying on the boundary of range $[\boldsymbol{0,1}]$ and only few of them lie in the interior.

Dropout regularization: Because keeping rate of dropout is being optimized in concrete dropout, the variational parameters $\boldsymbol{\theta}$ contain both $\{(\boldsymbol{M_i})_{\boldsymbol{i=1}}^{\boldsymbol{L}}\}$ and $\{(\boldsymbol{p_i})_{\boldsymbol{i=1}}^{\boldsymbol{L}}\}$. In order to compute gradients of ELBO in Eq.2.17, categorical re-parameterization is employed to estimate the gradients w.r.t. Bernoulli distribution parameter $\boldsymbol{p_i}$ in the first

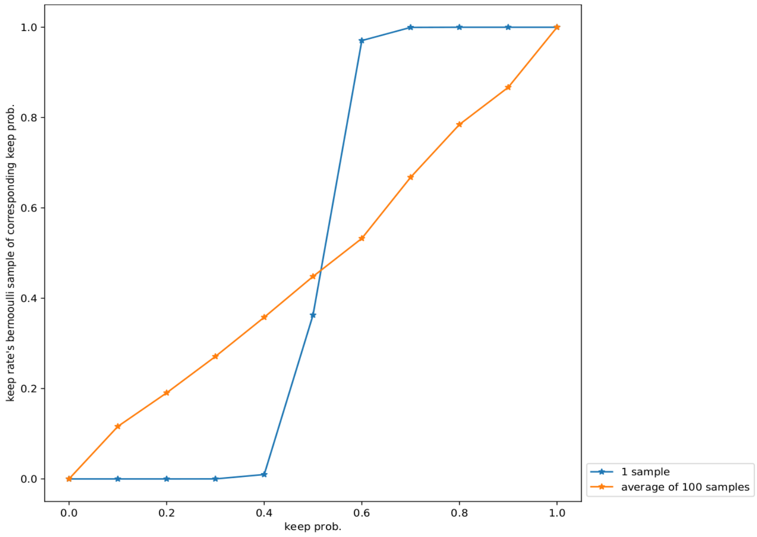

Figure 2.6: One sample and average value of 100 samples drawn from continuous approximation of Bernoulli distribution with parameter $\boldsymbol{p = [0.1, 0.2, ..., 1.0]}$ and temperature $\boldsymbol{\lambda = 0.1}$.

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-div term should be:

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

where $\boldsymbol{\lambda}$ and $\boldsymbol{\beta}$ 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, it can be seen 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 regularization 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 the weights is introduced, which is called Laplace approximation[Bis06]. It requires only the MAP point estimation of the model parameters, thus there is no need to re-train the existing model which has already obtained an point estimate. This characteristic makes this approach more appealing because it is not required to modify the model and thus can be applied to any existing models. The idea behind Laplace approximation is to put a Gaussian approximation on the MAP point estimate of the model parameters, which is one mode of the true posterior. The reasons why we choose this approach are as following:

- it has the easy compatibility to the existing network. To perform Laplace approximation, we only need one point estimation of the model parameters which is already available for all DNNs. There is no need to modify the training phase which saves a lot of computation and time.
- it can capture the relationship between model parameters. As mentioned in the first chapter, most of approximation approaches make an assumption that parameters are independent to each other for reasons like simplicity, scalability as well as efficient computation. This could be a quite strong restrictions on the approximation which would lead to bad performance.

In the following,the basic idea of Laplace approximation is introduced and further its scalable version for DNN based on Kronecker factor approximation is explained.

Assume there is already a point estimate of the model parameters via MAP or MLE:

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

After taking a second order Taylor expansion of the logarithm of of the posterior distribution $\boldsymbol{p}(\boldsymbol{\omega}|\boldsymbol{Y},\boldsymbol{X})$ around its mode $\boldsymbol{\omega}^{\star}$ (cf. A.3), the negative Hessian can be expressed as follows:

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

, $\boldsymbol{p}(\boldsymbol{\omega})$ is the prior distribution over model parameters. When exponentiating the second order Taylor expansion of log posterior, the following Gaussian-like functional form can be attained:

$$p(\omega|\boldsymbol{Y},\boldsymbol{X}) \propto p(\omega^\star|\boldsymbol{Y},\boldsymbol{X}) \exp\{-\frac{1}{2}(\omega-\omega^\star)^T \boldsymbol{H}(\omega-\omega^\star)\} \tag{2.28}$$

which means $\boldsymbol{\omega} \sim \mathcal{N}(\omega^\star, \boldsymbol{H}^{-1})$. 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 Gaussian prior is chosen, it's easy to know that the second order derivative of the prior distribution term $\frac{\partial^2 p(\omega)}{\partial \omega^2}$ 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, the negative Hessian of log likelihood is defined as $\hat{\boldsymbol{H}}$ as follows:

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

However, for a large training set, it is always infeasible to analyze the gradients or Hessian exactly. To resolve this, normally the expectation of gradients or Hessian is estimated with a mini-batch of data. It means that the Hessian can also be estimated with the empirical average Hessian computed in mini-batche:

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

where $\boldsymbol{N}$ is size of training samples, and

$$\mathbb{E}_{p(\boldsymbol{Y},\boldsymbol{X})}[\hat{\boldsymbol{H}}] \approx -\frac{1}{K}\sum_k \Big[\frac{1}{M}\sum_i \frac{\partial^2 \log\big(p(y_{ik}|x_{ik},\omega)\big)}{\partial \omega^2}\Big] \tag{2.29}$$

where $\boldsymbol{K}$ is the total number of mini-batch, and (y_{ik}, x_{ik}) is the $\boldsymbol{i}$-th training data sample in $\boldsymbol{k}$-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 the Fisher information matrix can be used as replacement of the 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:

$$\begin{aligned}\hat{\boldsymbol{F}} &= \mathbb{E}_{p(\boldsymbol{Y},\boldsymbol{X})}\Big[\frac{\partial}{\partial \omega}\log p(\boldsymbol{Y}|\boldsymbol{X},\omega)\frac{\partial}{\partial \omega}\log p(\boldsymbol{Y}|\boldsymbol{X},\omega)^T\Big] \\ &\approx \frac{1}{K}\sum_k \Big[\frac{1}{M}\sum_i \Big(\frac{\partial}{\partial \omega}\log p(y_{ik}|x_{ik},\omega)\frac{\partial}{\partial \omega}\log p(y_{ik}|x_{ik},\omega)^T\Big)\Big]\end{aligned} \tag{2.30}$$

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. The relationship between Fisher information matrix and expected negative Hessian when considering the entire training set is as follows:

$$\boldsymbol{\hat{H}} \approx \boldsymbol{N\hat{F}} \tag{2.31}$$

and after taking the Gaussian prior into account:

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

To compute the empirical average of outer product, large storage space should be used to save values with amount quadratic to the number of model parameters. More than that, the computation complexity of matrix inversion is cubic of dimensionality of the matrix which is the number of parameters of the whole network. This is also infeasible because the number of parameters in DNN can exceed million easily nowadays. 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 Fisher information matrix is only required after training, this little expensive computation is need to perform only once.

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

$$\boldsymbol{s_i} = \boldsymbol{a_i W_i} \text{ and } \boldsymbol{a_i} = \boldsymbol{\phi(s_i)} \tag{2.33}$$

We define $\boldsymbol{\tilde{w}}$ to be the vector consisting of all of the network parameters concatenated together, i.e. $\boldsymbol{\tilde{w}} = \left[\boldsymbol{vec(W_1)^T \; vec(W_2)^T \; ... \; vec(W_L)^T}\right]^{\boldsymbol{T}}$, where $\boldsymbol{vec}$ is the operator which vectorizes matrices by stacking their columns together. For simplicity the following notations are defined:

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

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

$$\boldsymbol{\partial \tilde{w}} = \left[\boldsymbol{vec(\partial W_1)^T \; vec(\partial W_2)^T \; ... \; vec(\partial W_L)^T}\right]^{\boldsymbol{T}} \tag{2.34}$$

From Eq.2.30, $\boldsymbol{\hat{F}} = \mathbb{E}[\boldsymbol{\partial \hat{w} \partial \hat{w}^T}]$ can be viewed as $\boldsymbol{L}$ by $\boldsymbol{L}$ block matrix, with the $\boldsymbol{(i,j)}$-th block $\boldsymbol{\hat{F}_{i,j}}$ given by $\boldsymbol{\hat{F}_{i,j}} = \mathbb{E}[\boldsymbol{vec(\partial W_i) vec(\partial W_j)^T}]$.

Noting that $\boldsymbol{\partial W_i} = \boldsymbol{a_{i-1} g_i^T}$ and $\boldsymbol{vec(uv^T)} = \boldsymbol{v} \otimes \boldsymbol{u}$, then we have:

$$\boldsymbol{vec(\partial W_i)} = \boldsymbol{vec(a_{i-1} g_i^T)} = \boldsymbol{g_i} \otimes \boldsymbol{a_{i-1}}$$

Then by rewriting it into:

$$\begin{aligned}
\boldsymbol{\hat{F}_{i,j}} &= \mathbb{E}[\boldsymbol{vec(\partial W_i) vec(\partial W_j)^T}] \\
&= \mathbb{E}[\boldsymbol{(g_i \otimes a_{i-1})(g_j \otimes a_{j-1})^T}] \\
&= \mathbb{E}[\boldsymbol{(g_i \otimes a_{i-1})(g_j^T \otimes a_{j-1}^T)}] \\
&= \mathbb{E}[\boldsymbol{(g_i g_j^T) \otimes (a_{i-1} a_{j-1}^T)}]
\end{aligned}$$

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

$$\begin{aligned}
\boldsymbol{\hat{F}_{i,j}} &\approx \boldsymbol{\tilde{F}_{i,j}} \\
&= \mathbb{E}[\boldsymbol{g_i g_j^T}] \otimes \mathbb{E}[\boldsymbol{a_{i-1} a_{j-1}^T}] \\
&= \mathbb{E}[\boldsymbol{G_{i,j}}] \otimes \mathbb{E}[\boldsymbol{A_{i,j}}]
\end{aligned} \tag{2.35}$$

where $\boldsymbol{G_{i,j}} = \boldsymbol{g_i g_j^T}$ and $\boldsymbol{A_{i,j}} = \boldsymbol{a_{i-1} a_{j-1}^T}$. 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 large diagonal block matrix, which means $\boldsymbol{\tilde{F}_{i,j}}$ is non-zero only for $\boldsymbol{i} = \boldsymbol{j}$. For $\boldsymbol{i}$-th layer, the Fisher information matrix can be computed with Eq.2.35. The two factors in Kronecker are outer product of gradients of pre-activation $\boldsymbol{G_i} = \boldsymbol{g_i g_j^T}$ and outer product of activation from previous layer $\boldsymbol{A_i} = \boldsymbol{a_{i-1} a_{i-1}^T}$, respectively:

$$\boldsymbol{\tilde{F}_i} = \boldsymbol{G_i} \otimes \boldsymbol{A_i}$$

If they are treated as covariance matrix of resulting Gaussian. That means each Gaussian has a Kronecker factored covariance matrix, 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 $\boldsymbol{i}$-th layer is:

$$\boldsymbol{W_i \sim \mathcal{MN}(W_i^\star, \mathbb{E}[A_i]^{-1}, \mathbb{E}[G_i]^{-1})} \tag{2.36}$$

If Gaussian prior and scale of Fisher information matrix(cf. Eq.2.32) are taken into account, then the resulting posterior can be written in the following form:

$$\boldsymbol{W_i \sim \mathcal{MN}(W_i^\star, (\sqrt{N}\mathbb{E}[A_i] + \sqrt{\tau}I)^{-1}, (\sqrt{N}\mathbb{E}[G_i] + \sqrt{\tau}I)^{-1})} \tag{2.37}$$

where $\boldsymbol{N}$ is the size of training set and $\boldsymbol{\tau}$ 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 $\boldsymbol{N}$. This will lead to unreasonably high precision for Gaussian and underestimate the uncertainty significantly. Second, Laplace approximation requires positive definite Hessian and $\boldsymbol{\tau}$ 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. 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 $\boldsymbol{p(y|x)}$. In the scope of scene object recognition problem, assume we have multiple objects in one scene denoted by $\boldsymbol{x} = [\boldsymbol{x_1}, ..., \boldsymbol{x_n}]$ and their labels $\boldsymbol{y} = [\boldsymbol{y_1}, ..., \boldsymbol{y_n}]$. We want to learn $\boldsymbol{p(y|x)}$ 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 $\boldsymbol{\mathcal{Y}}$. CRF for one scene $\boldsymbol{x}$ is defined by a graph $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}})$, where a set of nodes $\boldsymbol{\mathcal{V}}$ representing the random variables in $\boldsymbol{y}$, and a number of undirected edges $\boldsymbol{\mathcal{E}} \subseteq \boldsymbol{\mathcal{V}} \times \boldsymbol{\mathcal{V}}$ representing the dependencies between random variables. The output domain of CRF is the product of individual variable domains $\boldsymbol{\mathcal{Y}_i}$ so that we have $\boldsymbol{\mathcal{Y}} = \underset{i \in \boldsymbol{V}}{\times} \boldsymbol{\mathcal{Y}_i}$, where $\boldsymbol{\mathcal{Y}_i} = \boldsymbol{\mathcal{L}}$, which is a set of all possible labels. With these notations, the probability distribution of $\boldsymbol{y}$ conditioned $\boldsymbol{x}$ with model parameter $\boldsymbol{\theta}$ can be defined as

$$\boldsymbol{p(y|x;\theta)} = \frac{\boldsymbol{1}}{\boldsymbol{Z(x,\theta)}} \prod_{\boldsymbol{c \in \mathcal{C}}} \boldsymbol{\psi_c(y_c, x_c, \theta)} \tag{2.38}$$

where $\boldsymbol{\mathcal{C}}$ is the set of cliques of graph $\boldsymbol{\mathcal{G}}$, and $\boldsymbol{Z(\cdot)}$ is normalizer to make sure $\sum_{\boldsymbol{y \in \mathcal{Y}}} \boldsymbol{p(y|x;\theta)} = \boldsymbol{1}$, which is also called partition function. $\boldsymbol{\theta}$ stands for the parameters of CRF to be tuned during training phase. $\boldsymbol{\psi_c(\cdot)}$ is called potential function or factor which encodes the contribution of clique $\boldsymbol{c}$ to the total density. The order of number of random variables in clique $\boldsymbol{c}$ can be arbitrary number. However, for computational reason, only factors with number of random variables up to two are considered in common 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 $\boldsymbol{\psi_i(y_i, x_i, \theta)}$. 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 $\boldsymbol{\psi_{i,j}(y_i, y_j, x_i, x_j, \theta)}$ characterized by edges in the graph. This kind of potential states the compatibility of two random variables $\boldsymbol{(y_i, y_j)}$ being assigned to a certain pair of categories.

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

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

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

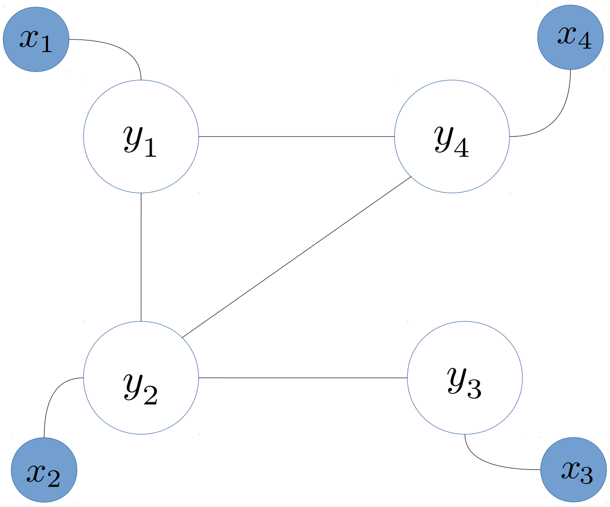

Figure 2.7: Simple example of conditional random field.

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:

$$\begin{aligned} \boldsymbol{p(y|x;\theta)} &= \frac{\mathbf{1}}{\boldsymbol{Z(x,\theta)}} \prod_{\boldsymbol{c \in \mathcal{C}}} \mathbf{exp}(\langle \boldsymbol{\phi(x_c, y_c), \theta} \rangle) \\ &= \frac{\mathbf{1}}{\boldsymbol{Z(x,\theta)}} \mathbf{exp}(\sum_{\boldsymbol{i \in V}} \langle \boldsymbol{\phi_u(x_i, y_i), \theta_u} \rangle + \sum_{\boldsymbol{(i,j) \in \mathcal{E}}} \langle \boldsymbol{\phi_p(x_i, x_j, y_i, y_j), \theta_p} \rangle) \end{aligned} \tag{2.39}$$

being $\langle \cdot, \cdot \rangle$ the inner product, and $\boldsymbol{\phi(x_c, y_c)}$ the sufficient statistic of the factor over clique $\boldsymbol{c}$. Because we only consider the potentials up to second order, only feature function for unary potential $\boldsymbol{\phi_u}(\cdot)$ and feature function for pairwise potential $\boldsymbol{\phi_p}(\cdot)$ 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.2.39. 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 $\boldsymbol{p(y_i|x_i)}$ 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:

$$\boldsymbol{p(y|x;\theta) = \frac{1}{Z(x,\theta)} \exp(\theta_u \sum_{i\in V} p(y_i|x_i) + \theta_p \sum_{(i,j)\in\mathcal{E}} H(y_i, y_j))} \tag{2.40}$$

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

2.2.2 Learning

The most popular approach is estimate $\boldsymbol{\theta}$ is MLE. The likelihood for one scene is defined in Eq.2.40. Normally we have large size of scenes in training set where each of them contains multiple objects. We denote training set by $\boldsymbol{\mathcal{D} = [d_1, ..., d_m]}$, where each scene is represented by $\boldsymbol{d_i = (x^i, y^i)}$. 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 $\boldsymbol{\mathcal{G}_i = (V_i, \mathcal{E}_i)}$. This is consistent to previous notations when we make little modifications for $\boldsymbol{x}$ and $\boldsymbol{y}$: $\boldsymbol{x^i} = [\boldsymbol{x^i_1}, ..., \boldsymbol{x^i_n},]$ and $\boldsymbol{y^i} = [\boldsymbol{y^i_1}, ..., \boldsymbol{y^i_n}]$. 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:

$$\boldsymbol{\mathcal{NLL}(\theta;\mathcal{D}) = -\sum_{i=1}^{m}\big(\theta_u \sum_{j\in V_i} p(y^i_j|x^i_j) + \theta_p \sum_{(j,k)\in\mathcal{E}_i} H(y^i_j, y^i_k) - Z_l(x^i;\theta)\big)} \tag{2.41}$$

where

$$\boldsymbol{Z_l = \log\big(\sum_{y^i\in\mathcal{Y}}\big(\theta_u \sum_{j\in V_i} p(y^i_j|x^i_j) + \theta_p \sum_{(j,k)\in\mathcal{E}_i} H(y^i_j, y^i_k)\big)\big)} \tag{2.42}$$

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

$$\boldsymbol{\theta^\star = \underset{\theta}{argmin}\big[\mathcal{NLL}(\theta;\mathcal{D})\big]} \tag{2.43}$$

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 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:

$$\boldsymbol{\frac{\partial}{\partial\theta}\mathcal{NLL}(\theta;\mathcal{D}) = -\sum_{i=1}^{m}\big(\phi(x^i, y^i) - \mathbb{E}_{p(y|x^i;\theta)}\big[\phi(x^i, y)\big]\big)} \tag{2.44}$$

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

$$\boldsymbol{\mathbb{E}_{p(y|x^i;\theta)}\big[\phi(x^i, y)\big] = \sum_{y\in\mathcal{Y}} p(y|x^i;\theta)\phi(y, x^i)} \tag{2.45}$$

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 $\boldsymbol{p(y|x;\theta)}$. 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.2.44 and Eq.2.45, to calculate the gradients in each step of SGD, we need to compute $\boldsymbol{p(y|x;\theta)}$, which requires to compute the partition function $\boldsymbol{Z(\cdot)}$. 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 (LBP) and introduced in next subsection. This approach can provide a approximation to the log-partition function $\boldsymbol{Z_l(\cdot)}$, which is required in Eq.2.45 and to the marginal probability distribution of random variables in $\boldsymbol{y}$ to be plugged in Eq.2.44 in the graph with cycles and exact solution in graph without cycle.

2.2.3 Inference

LBP 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 $\boldsymbol{k}$-dimensional vector $\boldsymbol{b(y_i)}$ if the node is discrete random variable where $\boldsymbol{k}$ is the number of class. The $\boldsymbol{i}$-th entry of this vector indicates the probability that $\boldsymbol{i}$-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 $\boldsymbol{y_i}$ to node $\boldsymbol{y_j}$ encodes belief of $\boldsymbol{y_i}$ about what class node $\boldsymbol{y_j}$ 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:

$$\boldsymbol{m_{ij}(y_j) = \sum_{y_i \in \mathcal{Y}_i} \psi_{ij}(y_i, y_j, x_i, x_j, \theta_p)\psi_i(y_i, x_i, \theta_u) \prod_{y_k \in \mathcal{N}(y_i)\backslash y_j} m_{ki}(y_i)} \tag{2.46}$$

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

$$\boldsymbol{b(y_i) = \frac{1}{Z_i}\psi_i(y_i, x_i, \theta_u) \prod_{y_j \in \mathcal{N}(y_i)} m_{ji}(y_i)} \tag{2.47}$$

where $\boldsymbol{Z_i}$ is the normalizer of unnormalized belief of node $\boldsymbol{y_i}$ to ensure the sum of marginal probability of $\boldsymbol{y_i}$ is 1.

Chapter 3

Technical Approach

In this chapter, the approaches employed in this work are explained. Firstly, confronted with the limitations of concrete dropout, an adapted version of concrete dropout is proposed called multiple dropout (multi-drop). Secondly, a modified architecture of ResNet50 is introduced in order to incorporate different inference techniques for BNN with this powerful feature extraction network. Thirdly, a pipeline which combines BNN and CRF is presented. Thereby, The CRF works as a post-processor to capture the relationship between random variables. Last but not least, the approach combining aforementioned techniques for continuous learning is introduced.

3.1 Multiple dropout

Based on Fig.2.2 and the expression of the approximate distribution in equation 2.11, only one probability of Bernoulli distributed random variable for each layer is chosen. Each element of random vector $\boldsymbol{p_i} = [\boldsymbol{p_i}]^{\boldsymbol{D_{i-1}}}$ for $\boldsymbol{i}$-th layer is the same. While the dropout regularization term pushes the probability of Bernoulli to 0.5 to maximize the entropy, the expected likelihood term tries to increase the probability since a decreasing probability will lead to a different model with lower capacity and thus lower performance. Thus, an equilibrium state between them should be achieved in training. With the concrete dropout introduced above, this approach can be extended for each hidden unit instead of each layer(cf. Fig.3.2), which results in the random vector $\boldsymbol{p_i} = [\boldsymbol{p_i^k}]_{\boldsymbol{k=1}}^{\boldsymbol{D_{i-1}}}$. The difference between multiple dropout and concrete dropout can be easily seen by comparing the figures Fig. 3.2 and Fig. 2.3. While the first term in the gradients computation stays the same, the second term should be modified to:

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

The reasons for multi-drop are as following:

- to increase flexibility in tuning variational parameters. The tunability of the parameter of the Bernoulli random variable in the likelihood term is low since there is only one single parameter controlling the entire layer. As observed in the experiments(cf. Fig.3.1), the parameter is 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 is optimal to assign the same probability for each hidden unit, this can be recovered in training with the multi-drop. Otherwise, other optimal solutions of assigning different probabilities to different hidden units could be considered.
- 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 better by the approximate posterior.

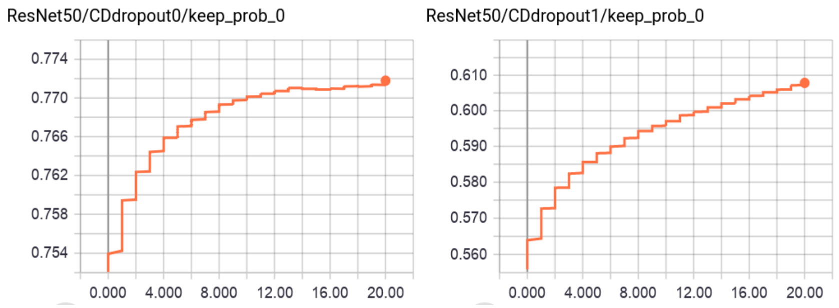

Figure 3.1: Changes of keep probability of the first two concrete dropout layers during training.

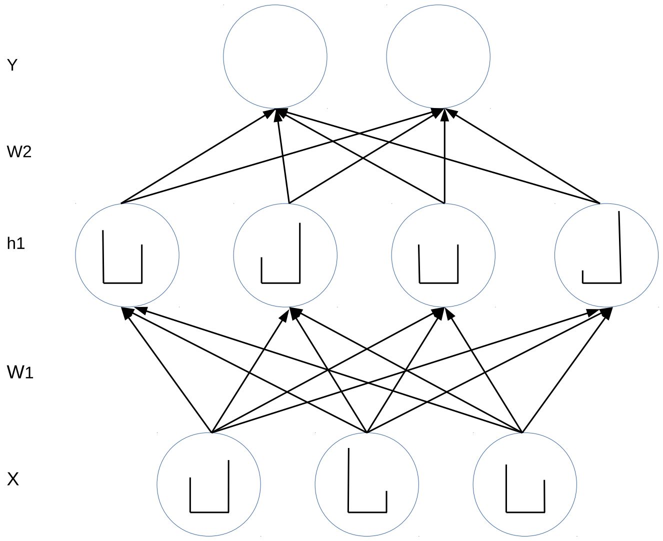

Figure 3.2: Different dropout rates for different hidden units in multi-drop.

3.2 Modified network architecture

After introducing dropout and concrete dropout variational inference, in this subsection the modified network architecture is described. As already mentioned, the fundamental task in this work is object classification. Thus, due to its strong ability to learn powerful representation for images a ResNet50[HZRS16] pre-trained on ImageNet as backbone for fine-tuning is selected. However, there is no dropout in the original version of this network. In order to employ dropout variational inference for a reliable uncertainty estimation, the dropout concept should be inserted into the network. In this work, three fully connected layers with 1024 hidden units are added before the output layer – whose dimension needs to be set to the number of classes – which are initialized from scratch. Next, a concrete dropout at flatten layer is added, and at the three new fully connected layers, respectively. There are three reasons why the network is modified this way:

- By inserting dropout in pre-trained layers, the pre-trained features will get destroyed. Since the major part of the network is initialized with pre-trained weights, hence this would lead to a significant drop of performance after fine-tuning.
- 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 three additional fully connected layers are added to ensure that the model possesses a large enough model capacity.
- As introduced in previous sub-sections, weights are a major part of variational pa-

rameters. Therefore, more weights can enhance the flexibility and capacity of the approximate distribution family, which improves the quality of approximation.

Fig.3.3 shows the sketch of the modified network architecture. The major part of the network without dropout can be interpreted as deterministic feature extractor. Following this, the four layers with dropout 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 the size of number of categories is appended. The version of dropout inserted can be normal dropout, concrete dropout or multiple dropout. In this report, only results of concrete dropout and multiple dropout are presented since normal dropout significantly underperforms.

During training, these two parts are trained together in order to achieve a better balance between uncertainty estimation and accuracy. As known to all, freezing the 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, an ablation study is conducted shown in experiment part. In testing, possible parameters according to posterior distribution have to be marginalized. 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 during testing can be achieved by different techniques, such as parallel computing or distillation. But this surpass the scope of this work.

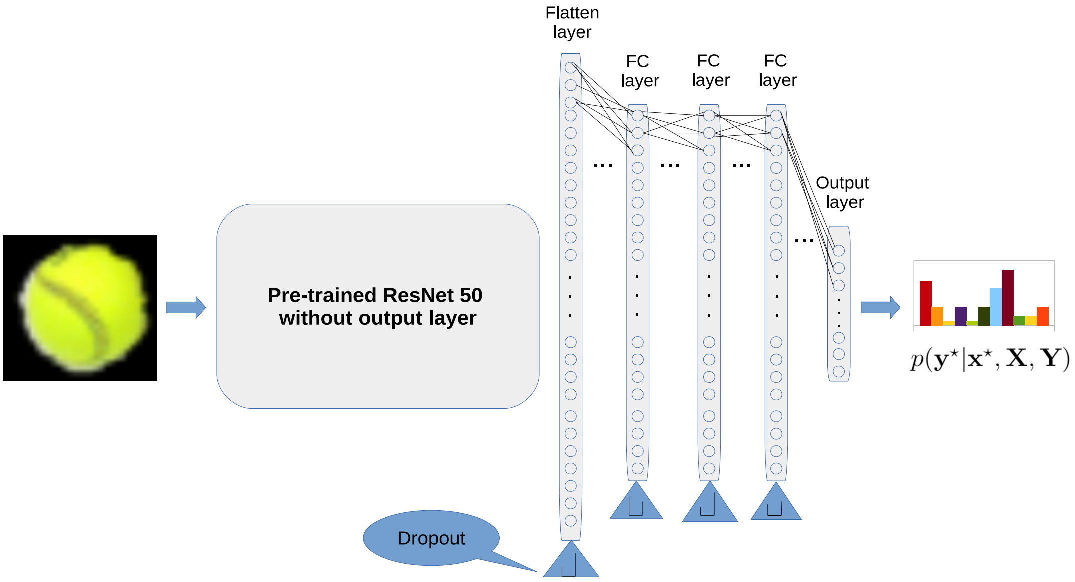
Flatten layer
FC layer
FC layer
FC layer
Output layer
Pre-trained ResNet 50 without output layer
Dropout
$p(\mathbf{y}^\star|\mathbf{x}^\star, \mathbf{X}, \mathbf{Y})$

Figure 3.3: Modified network architecture of ResNet50.

3.3 Combination with CRF

After employing BNN, the model can provide more reliable uncertainty estimation in a probabilistic manner. In other words, the predicted probability distribution contains more valuable information than before. In order to make use of this information another probabilistic model is applied. The idea is to combine the output from BNN and relationships between the objects in a scene via a CRF. In detail, the output distribution from BNN is used as unary feature and a binary co-occurrence matrix for pairwise feature (introduced in 2.2.1). Fig.3.4 demonstrates the flow chart how BNN and CRF are combined together. As can be seen, objects in one scene are firstly fed into BNN and their output distributions are fed into the CRF as unary feature for each node. In the scene, their contextual relationship is represented by the binary co-occurrence matrix which is forwarded to the CRF as pairwise feature. Then LBP is applied 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 the unary and pairwise feature have, the more improvement CRF can achieve. One thing worth noting, in experiment part, this approach is tested only on the T-LESS dataset, since constructing scenes for objects in the RGBD Dataset from Washington University(WRGBD)[LBRF11] dataset is non-trivial. The reason therefore is that most objects there have not only similar appearances but also similar contexts. Therefore it is hard to disambiguate miss-classification by incorporating their contextual information.

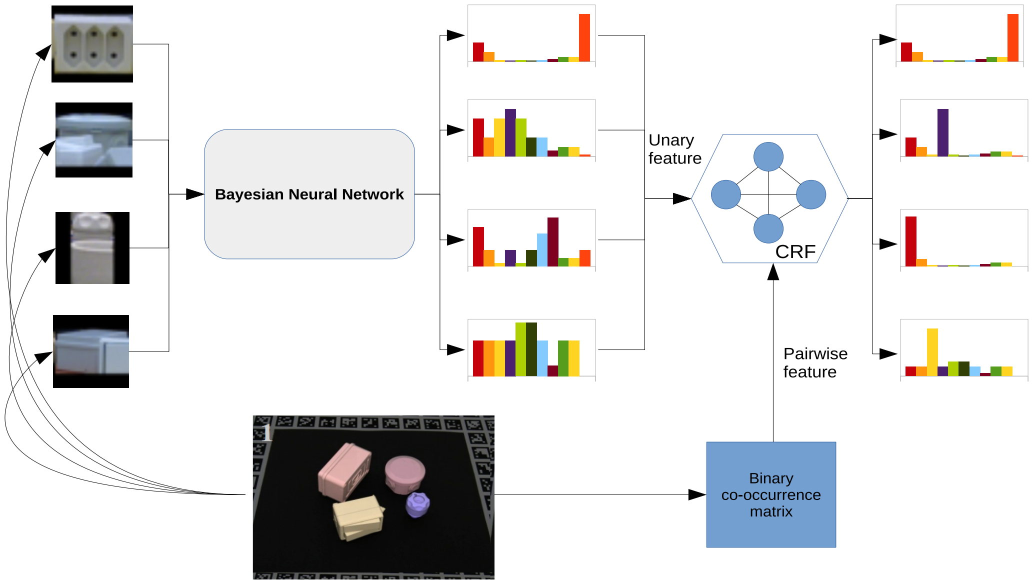

Figure 3.4: Combination of BNN and CRF.

3.4 Approach for continuous learning

With a 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 is common that the test data in real environment does not have exactly the same distribution as the training set, which induces a significant performance drop. In this situation, the classifier is expected to adapt itself to the real environment by fine-tuning itself with as little manual effort for the user as possible.

This idea is visualized in Fig.3.5. The initialization phase includes the training of the model with an easily obtainable dataset, which can be a public or synthetic dataset. Before deploying to 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 the real environment by fine-tuning with the dataset collected. The annotations for the collected dataset are generated automatically by the initial network and manually by the user. In order to prevent a high amount of miss-classified training data for fine-tuning, only high confident predictions are used for the automatic labeling. Therefore, a reliable uncertainty estimation is essential. If the relationship between objects in real environment is complementary to the discriminative classifier and can be encoded well, the CRF can be employed to capture the relationship to further improve the performance (cf. section 3.3). After that, the classifier is deployed to the test data in the real environment. In the experiment part, this approach is evaluated on both UniHB(WRGBD) and T-LESS(synthetic T-LESS) data.

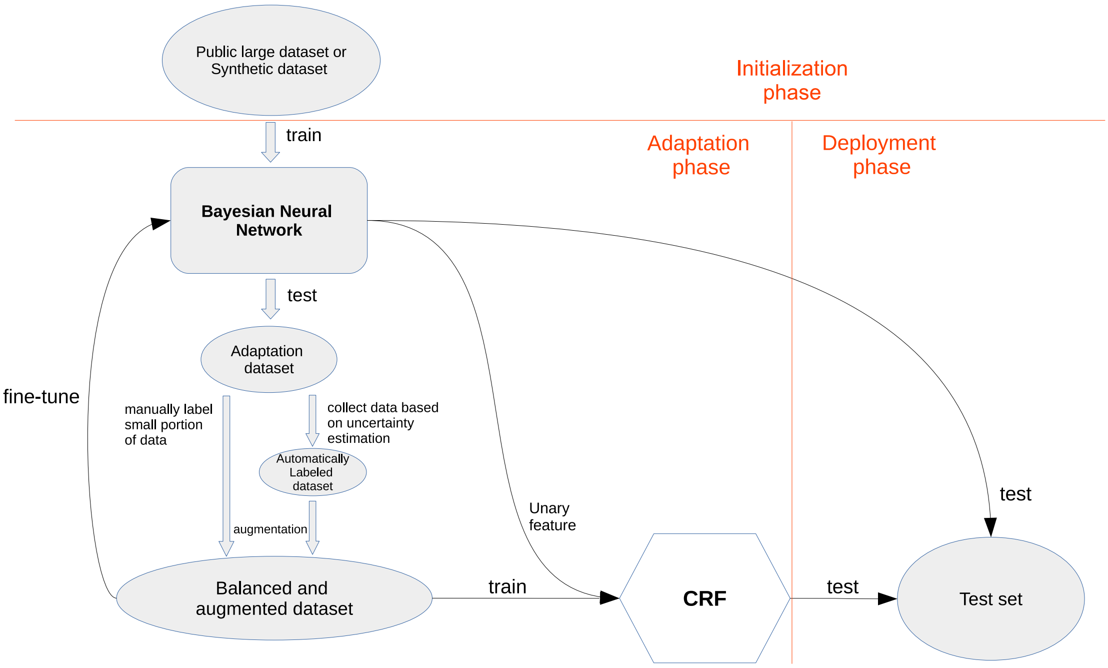
Public large dataset or Synthetic dataset
Initialization phase
train
Adaptation phase
Deployment phase
Bayesian Neural Network
test
Adaptation dataset
fine-tune
manually label small portion of data
collect data based on uncertainty estimation
Automatically Labeled dataset
augmentation
Unary feature
test
Balanced and augmented dataset
train
CRF
test
Test set

Figure 3.5: Approach for continuous learning.

Chapter 4

Experiment and Discussion

4.1 Background

In this chapter, different experiments with the following purposes are conducted:

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

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

Regarding the implementation of CRF, the C++ library from [RSGGJ15] is employed. However, the weights there are assumed to be category-independent, which means that the weights are defined as a weight matrix to build a linear mapping from the feature function to the potential function. Therefore the type of weights in this library are modified in order to incorporate the different type of weights defined in Eq. 2.40. These weights 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 the parameters are initialized with a Gaussian prior with a small standard deviation (0.01) around 0. Besides, the optimization is performed with SGD with momentum and set size of mini-batch as 16. The learning rate is set to $\boldsymbol{1e^{-}4}$ and the maximum number of iteration is $\mathbf{15 \times 10^3}$. LBP is used for inference both during training and testing. The maximum number of iterations for computing messages in LBP is 100, and the convergence threshold is $\boldsymbol{1e^{-4}}$.

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. Fig.4.1). The total size of entire dataset is around ∼160.9K. To note that the category level recognition involves classifying objects with similar semantic appearance such as cereal boxes with different texture into the 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 feature spaces 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 situations in the deployment of robots, a similar dataset is recorded by putting objects on a turn table in front of the robot and recording partial views of the objects. Thereby, the same methodology as [LBRF11] suggests for WRGBD is applied, but with only one object instance for each category in the dataset. This analogue of the WRGBD dataset is called the IAI-ODU dataset of UniHB, which in the following is called UniHB due to simplicity. The size of data with the elevation **45°** is around ∼8.6K and that of data with the elevation **30°** and **60°** is around ∼17.1K.

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

Original 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, an 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 ∼38.0K training images (cf. on the left of each thumbnail in Fig.4.3) and ∼10.0K test scene images (cf. on the right of each thumbnail in Fig.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 only the cropped images of objects in testing scenes are considered, resulting in a testing set with size ∼69.5K. Nevertheless, the context information in the scenes can be exploited with CRF which is showed in the experiment part.

Synthetic T-LESS

Since the data collection of real objects requires a high amount of manual efforts, the idea is to first train the classifier on a synthetic dataset which is easy to obtain. AFor this end, a synthetic T-LESS dataset is generated which contains multi-view images of objects rendered from 3D reconstructed CAD models with similar augmentations(cf. on the right of each thumbnail in Fig.4.4). There is a significant large domain gap between the synthetic objects and real objects (cf. on the left of each thumbnail in Fig.4.4) such as texture and geometries on the surface. This domain gap can induce a large performance drop when the model trained with synthetic objects is tested on real objects.

Augmentation is employed for both original and synthetic T-LESS dataset. Because objects in the test scenes are mostly occluded and have different backgrounds and lightness, this practical technique can improve the generalization of the model and thus the performance. In augmentation, the real and synthetic single objects are put onto different background images from the VOC2012 dataset[EVGW^{+}](cf. Fig.4.4) and augmented in different ways such as Gaussian noise, color changes, rotations and so on. One characteristic of the augmentation is that each background image can be treated as one scene, in which a random number of objects are placed randomly, inducing large variety of occlusions and combinations of different objects. This tries to simulate the situation in the test set, where different objects are placed randomly in each scene. More than that, the relationships between objects in one scene can be utilized by the CRF for a further performance gain.

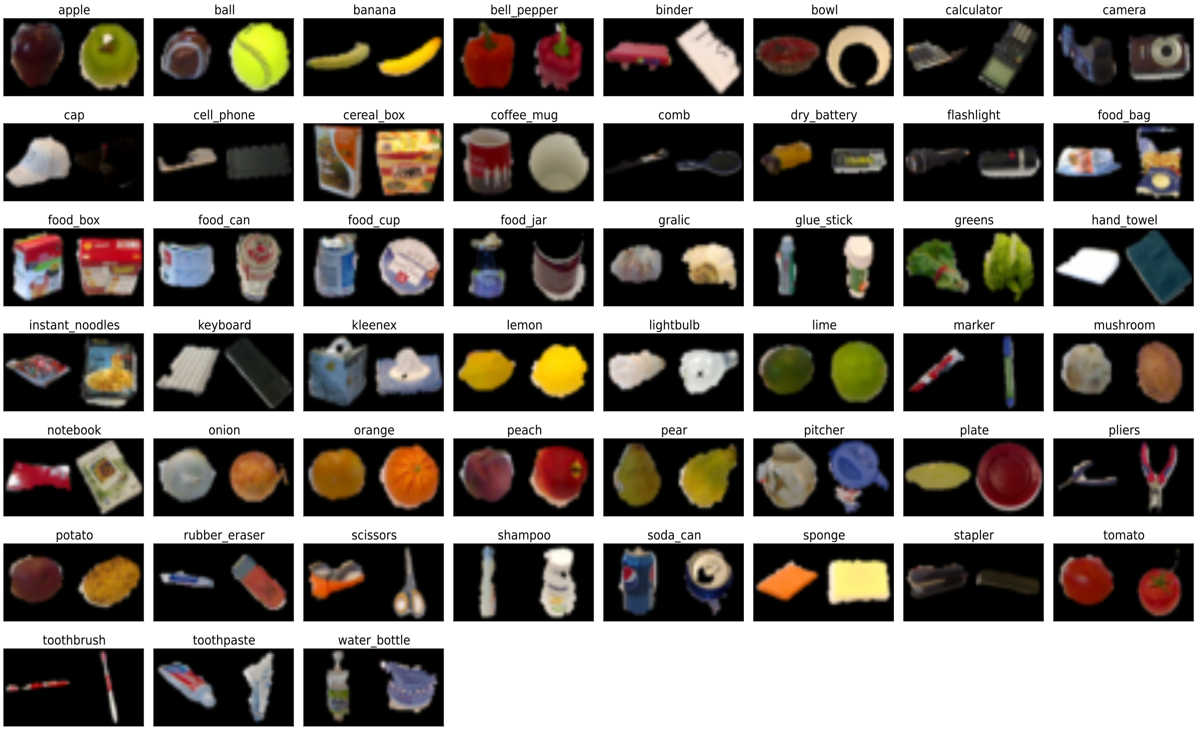

Figure 4.1: Example of masked images of objects from 51 categories in WRGBD and UniHB dataset. In each category, the left is from WRGBD and the right is from UniHB. We randomly pick one instance for the objects of WRGBD. Some light and appearance differencse between objects can be seen in these two datasets.

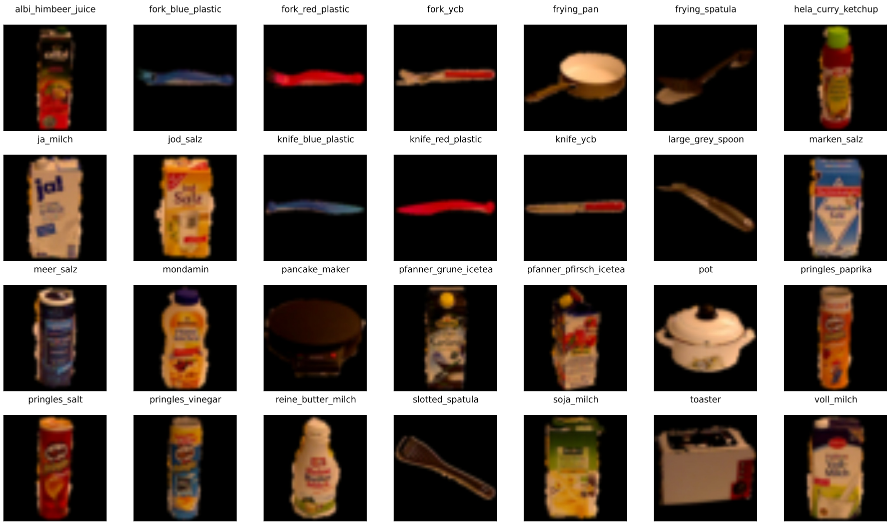

Figure 4.2: Example of masked images of objects from 28 categories which are not belonging to WRGBD categories, which are treated as OOD data samples.

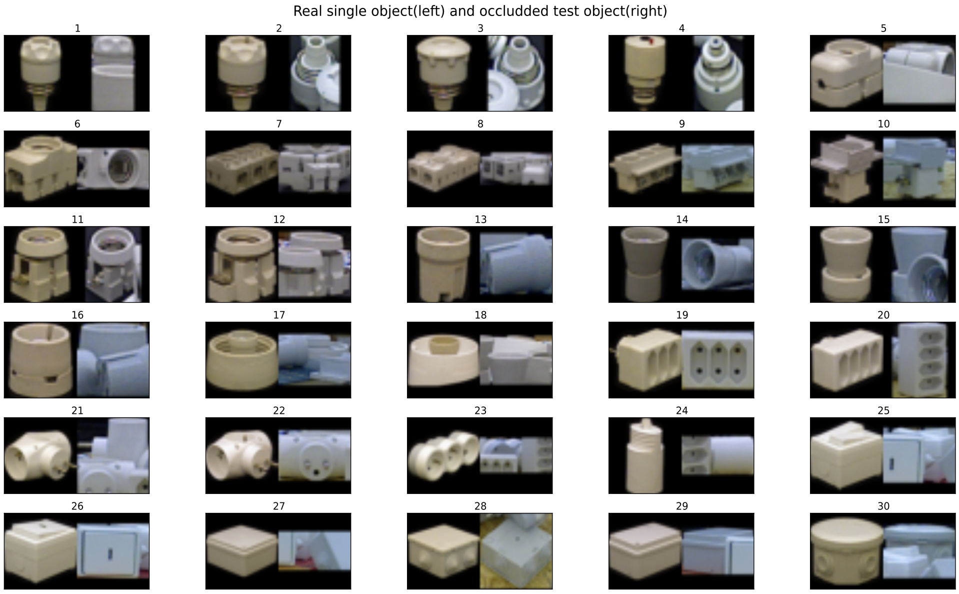

Figure 4.3: Example of object of each category in original training set and test set. In each thumbnail, the left image is real single object with black background and the right image is cropped image of object in test scene. (label index starts from 1.)

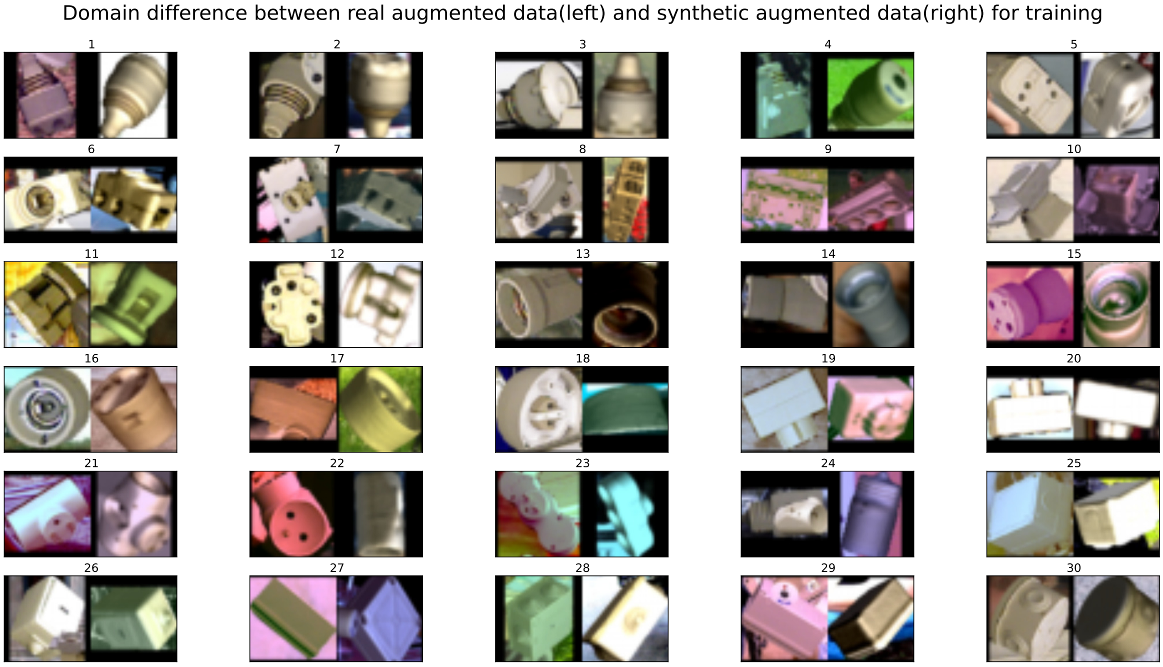
Domain difference between real augmented data(left) and synthetic augmented data(right) for training
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

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 measure

For each prediction, a predictive probability distribution from the model based on the equation 2.23 is obtained. In order to quantify the uncertainty of the prediction, there are different measures for uncertainty of a prediction, which are introduced in the following. A test data sample is defined by $\boldsymbol{x}^{\star}$ and $\boldsymbol{\mathcal{D}}$ as the training set.

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

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

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

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

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

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

Mutual information between the prediction $\boldsymbol{y}$ 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:

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

where $\boldsymbol{\mathcal{I}(\cdot)} \in [\boldsymbol{0}, \mathrm{log}|\boldsymbol{\mathcal{P}}|]$. This quantity considers the effect of an 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 w.r.t. the weights posterior. It is only low if the predictive distribution agrees with most of the possible models (weights realizations), which means that the model is sure about its prediction. Otherwise, it is high because most of the possible models do not agree with the other models and thus the predictive distribution is more uniform and thus has a higher entropy.

4.1.3 Evaluation metric

As stated in [GBR07], the goal of the probabilistic prediction is to maximize the sharpness of the 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 different metrics including both the accuracy and other quantities related to calibration as well as summary of accuracy and calibration are employed. Additionally, a histogram and diagram to express the results visually are applied. While the visual one can show the results more intuitively, the quantitative one allows to evaluate the results more objectively. The comparison between them may help to examine if the visual metrics correspond to the numerical metrics, which may provide more insights in evaluating uncertainty estimation.

Uncertainty histogram is an intuitive visual tool for analyzing the statistics of the uncertainty estimation. Compared with normal histograms, there is one difference to stress on. In order to make the visual effect more clear and hence the analysis easier, the **normalizer** of each type of prediction is the size of this type of predictions instead of size of all predictions. In detail, there are three types of predictions for plotting the histogram, which are:

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

The range of y axis is $[\mathbf{0}, \mathbf{1}]$ since the bins are normalized, and the range of x axis is set by the range of the corresponding type of the uncertainty measure.

Reliability diagram(Calibration curve) is another visual tool for expressing the **calibration** performance of the model[GPSW17], which plots the frequency of the success(accuracy of predictions in specific bin) as a function of confidence, which is the predictive likelihood of the prediction. If the model is perfectly calibrated, this function should be exactly overlapping with the diagonal line. The closer this curve is to diagonal curve, the better the calibration performance is. In order to draw the curve, firstly the predictions are grouped into $\boldsymbol{M}$ (here: $\boldsymbol{M} = \mathbf{20}$) interval bins w.r.t. the confidence. Next, the accuracy of predictions in each bin is calculated. To quantify the proximity to the diagonal curve, two metrics are used:

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

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

  where $\boldsymbol{n}$ is the number of samples, $\boldsymbol{acc(B_m)}$ represents the accuracy of the samples, and $\boldsymbol{conf(B_m)}$ the average predicted confidence of samples in the $\boldsymbol{m}$-th bin. One 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 the accuracy. One drawback of this metric is that, because the weights are computed based on number of samples in each bin, the metric would be 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, this metric is used. Additionally, in high risk applications where reliable confidence measures are absolutely necessary, one might take into account the worst case. The MCE is defined as follows:

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

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 the predictive distribution and real one.

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). The scoring rule can be defined as a function $\boldsymbol{\mathcal{S}(p(y|x), (y|x))}$ that evaluates the quality of the predictive distribution $\boldsymbol{p(y|x)}$ relative to an event $\boldsymbol{y|x \sim q(y|x)}$ where $\boldsymbol{q(y|x)}$ represents the true distribution over $\boldsymbol{(y|x)}$. Consequently, the expected scoring rule is:

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

$\boldsymbol{\mathcal{S}(p, q)}$ is proper if $\boldsymbol{\mathcal{S}(p, q) \leq S(q, q)}$, with equality holds if and only if $\boldsymbol{p(y|x) = q(y|x)}$. 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 the predictive uncertainty [QCRS+05]. This metric considers the aforementioned confidence as a likelihood. The smaller this metric is, the better the predictive distribution is.

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

  where $\boldsymbol{c_i}$ 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 the predictive distribution:

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

  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. It is also interesting to see 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, the correct predictions are treated as positive samples in the two following metrics.

- **Area under Receiver Operating Characteristic curve(AUROC)**: Since one of the goals is to choose automatically labeled data based on the uncertainty estimation, it is necessary to evaluate the separability between correct predictions and false predictions or out-of-distribution data predictions. The ROC curve describes the relationship between the true positive rate($\boldsymbol{tpr} = \frac{\boldsymbol{tp}}{\boldsymbol{tp}+\boldsymbol{fn}}$) and the false positive rate($\boldsymbol{fpr} = \frac{\boldsymbol{fp}}{\boldsymbol{fp}+\boldsymbol{tn}}$). 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 rates in the 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 the positive one, the AUROC could still achieve a relatively high value although there are already many false positives.

- **Area under Precision Recall curve(AUPR)**: Considering the downside of AUROC, another evaluation metric is AUPR. This metric describes the relationship between the precision($\boldsymbol{pr} = \frac{\boldsymbol{tp}}{\boldsymbol{tp}+\boldsymbol{fp}}$) and the recall($\boldsymbol{tpr} = \frac{\boldsymbol{tp}}{\boldsymbol{tp}+\boldsymbol{fn}}$), which resolves the problem of different base numbers. 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, the performance of accuracy and uncertainty estimation as well as the summarized performance of different inference techniques for BNN on WRGB and UniHB dataset is evaluated. Before that, the shorthands of different techniques should be explained. In the following the original ResNet50 is depicted as "**ori**", ResNet50 with concrete dropout as "**cdp**",and ResNet50 with multiple dropout without specifications as "**mdp**".

In the first experiment, they are evaluated on a relatively simple task, namely instance recognition. However, the 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, namely category recognition. Besides comparing them, an ablation study is conducted to investigate how the feature extractor influences the predictive uncertainty.

4.2.1 Experiments I: uncertainty estimation on instance recognition

The appearances of different classes as well as the out-of-distribution data are highly discriminable. To this end, the WRGBD dataset is separated into two subsets based on the instance classes. The Subset I contains objects with the instance classes from 0 to 199 (assuming that here the index denotes the instance label) and the Subset II contains objects with the instance classes from 200 to 299.

The model is trained with objects captured in the elevation **30°** and **60°** of the Subset I (in which 20% are split off as validation set for model selection in the training). The size of the training set and validation set is around ∼71.0K. Then the model is tested on objects captured in the 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 the test set is around ∼35.6K and that of the OOD dataset is around ∼17.8K.

In this experiment, the performance of different approaches should be evaluated relatively. Since the OOD data is highly different to the training set, the model should express high uncertainties 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 is confident with its incorrect predictions, which can induce hazardous consequence in some safety-critical applications. As can be seen in the Fig.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 can be seen, different uncertainties show similar a trend in this experiment.

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

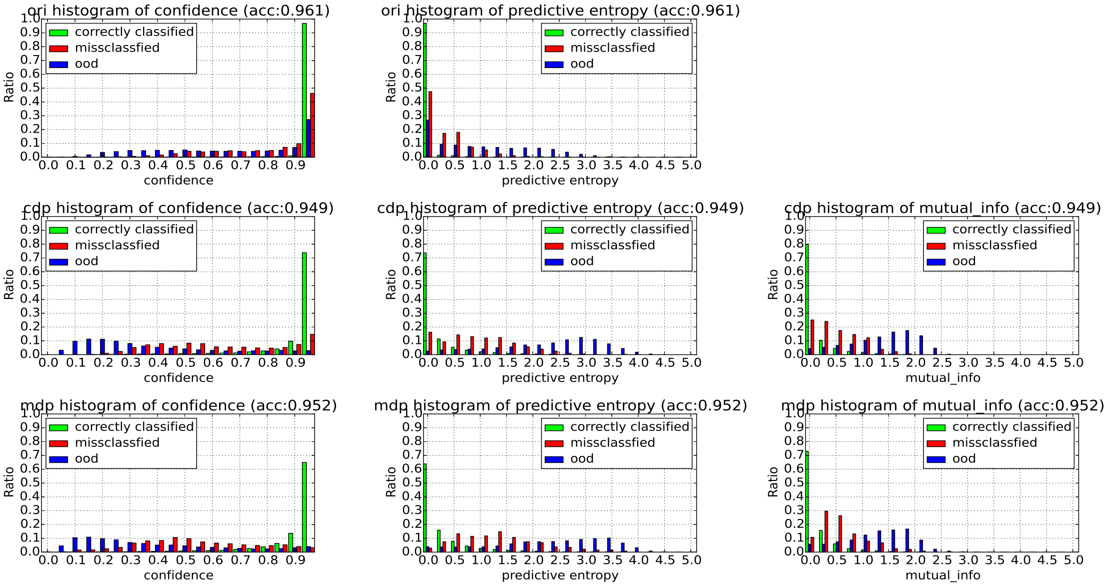

Figure 4.5: Uncertainty(confidence, predictive entropy, mutual information) histograms of original ResNet50, ResNet50 with concrete dropout and ResNet50 with multi-dropout.

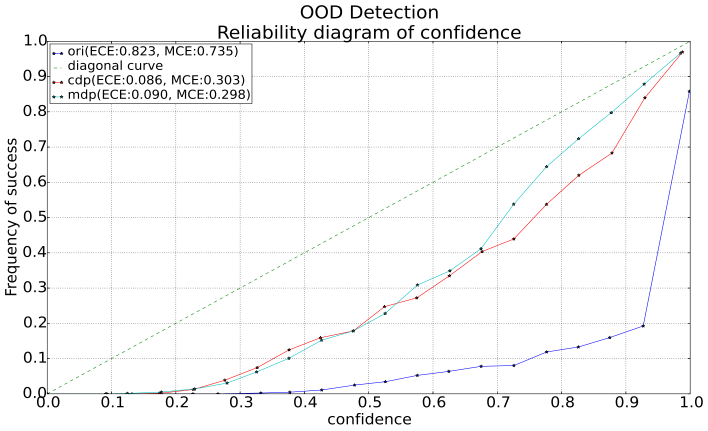

Figure 4.6: Calibration curves of original ResNet50, ResNet50 with concrete dropout and ResNet50 with multi-dropout evaluated on both test set and OOD dataset.

4.2.2 Experiments II: uncertainty estimation on category recognition

In this experiment, the model is evaluated on a category recognition task. The difficulties are expressed in two-fold. First, in this task, the model is confronted with classes which overlap more. Second, in order to simulate the situation of deploying robots in real world scenario and facing data in real world, the UniHB dataset with a domain gap(cf. Fig.4.1) to WRGBD is used to achieve this goal. This means, the uncertainty estimation should not only be able to perform well on the dataset with the exactly same distribution to training set, but also to generalize well to the dataset with domain gap which may decrease the accuracy significantly.

Accordingly, the entire WRGBD dataset is used including all view points, whose size is around ∼160.9K, to train the model (in which the ratio between training and validation set is 2:8). When it comes to UniHB dataset, objects captured in elevation **30°** and **60°** are treated as **adaptation set**, whose size is around ∼17.1K. The performance of uncertainty estimation is tested on this adaptation dataset. The images of **45°**, whose size is around ∼8.6K , are used for the final evaluation after being fine-tuned with a subset of the adaptation set to obtain a domain specific model. The latter part will be experimented in next section. Besides, the uncertainty estimation on OOD dataset(cf. Fig.4.2) whose size is around ∼6.0K is also evaluated.

In the following, firstly the protocols for different kinds of metric are explained, which can help understanding the plots and extracting useful information more easily and quickly.

- The following visual metrics are chosen in one of the three runs of different random seeds. The average quantitative results are given in tables following the visual metrics.
- The calibration curve with the title ”Classification” on the left is plotted **only** on the predictions of the test set. The one with the title ”OOD detection” on the right is plotted on the predictions of **both** OOD dataset and test dataset.
- The ROC curve and the PR curve measure the separability between the two types of prediction. The ones with the title ”classification” on the left measure the separability between **correct predictions** and **miss-classifications**. The ones with title ”OOD detection” on the right measure the separability between **correct predictions** and **OOD predictions**. In all the ROC and PR curves, correct predictions are always chosen to be positive.
- As shown in the uncertainty histogram, three uncertainty measures are applied. Each one has its own ROC curve and PR curve of the correct predictions versus the miss-classifications or OOD predictions. In the following plots of the ROC curve and the PR curve, only the one with highest area under curve is depicted. In detail, the **Confidence** is chosen in case of the correct predictions versus the miss-classifications. The **Mutual information** is chosen in case of the correct predictions versus the

OOD predictions.

Comparison with Ensemble

In this subsection, the results are shown in not only a qualitative (visual) way, but also in a quantitative way. The approaches compared in this part 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, the results are averaged from three different random seeds and report the mean and the standard deviation.

In the Fig.4.8, on the left there are calibration curves of different approaches, plotted from predictions on test dataset. It can be seen 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, the robustness against OOD data of cdp, mdp and their ensembles are better than the original version.

In the Fig.4.7, uncertainty histograms for different approaches and different uncertainty measures. Firstly, by comparing them vertically, the improvement can be observed 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 predictions is decreased. But it is known 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 by comparing them horizontally, it's shown that different uncertainty measures have similar trends in the histogram. Therefore other quantitative metrics for them are requiredm in order to evaluate different approaches as well as metrics more accurately and objectively.

When it comes to the separability metrics in Fig.4.9, as stated before, the separability between correct prediction and miss-classification (on the left) and that between correct prediction and OOD prediction (on the right) are measured. 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 observed with this metrics.

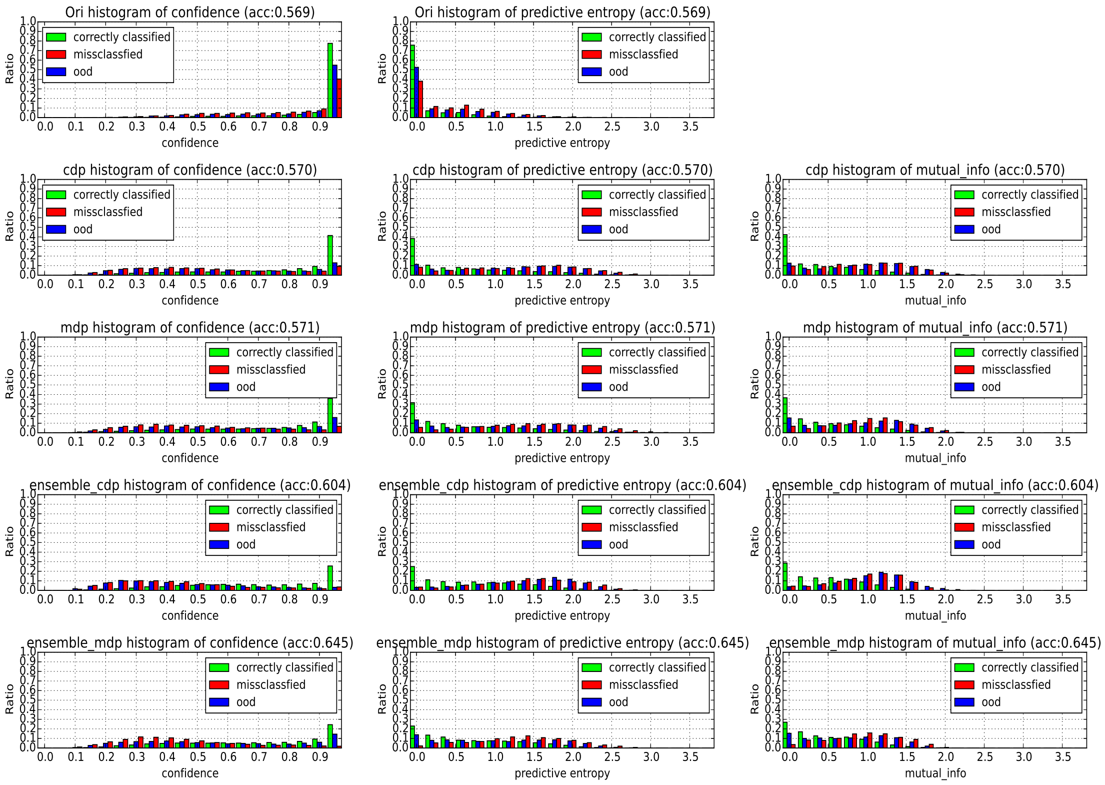

Figure 4.7: Uncertainty histograms of confidence, predictive entropy, mutual information for ori, cdp, mdp, ensemble of cdp, ensemble of mdp in one of three runs.

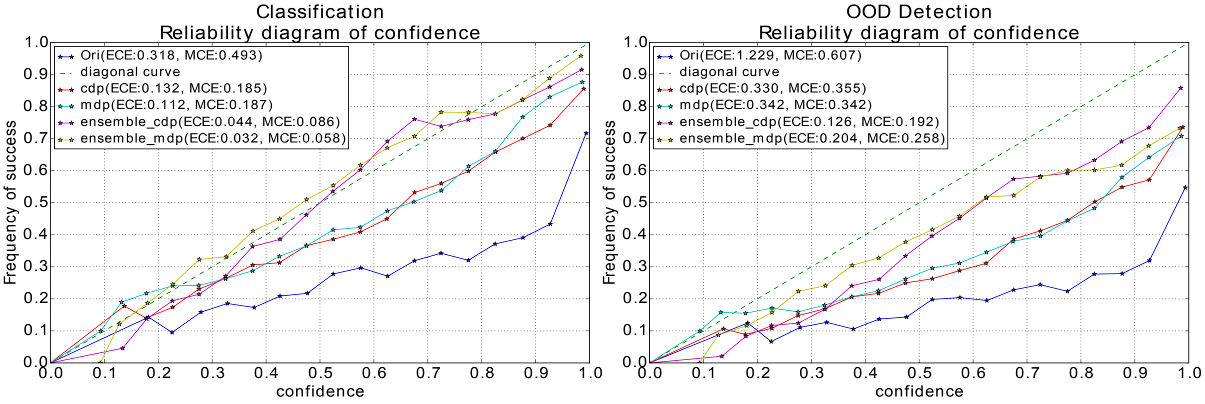

Figure 4.8: Calibration curve for ori, cdp, mdp, ensemble of cdp, ensemble of mdp in one of three runs.

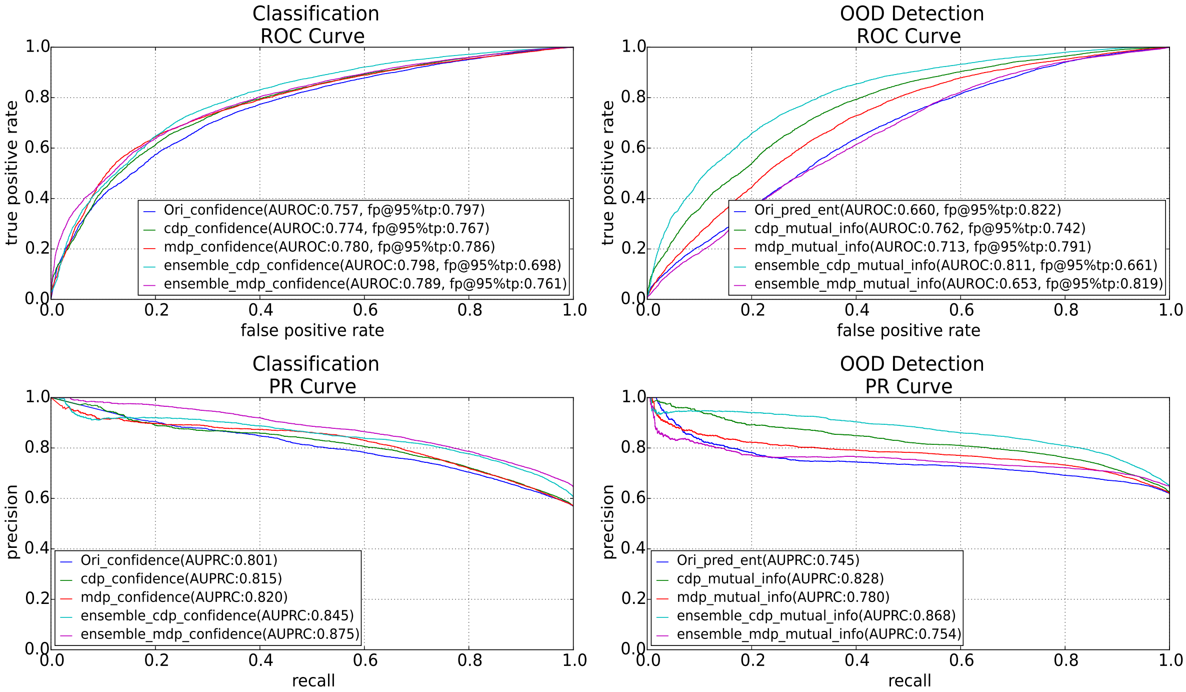

Figure 4.9: ROC and PR curve of ori, cdp, mdp, ensemble of cdp, ensemble in one of three runs.

Before only visual metrics for evaluation are shown, it's better to quantify the performance quantitatively. To this end, different aforementioned metrics are presented in the following table 4.1. In order to exclude the influence of random initialization, three models of different random seeds are trained and the mean and standard deviation for each metric are reported. It can be seen the improvements is achieved by cdp, mdp and their ensemble versions along all 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, the following conclusion can be drawn, 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 the already trained model of different approaches are taken as MAP point estimate. The approximation of Kronecker factors are computed with half of training set. The scale parameter of Kronecker factors $\sqrt{\boldsymbol{N}}$ is set to **1** and dump factor $\sqrt{\boldsymbol{\tau}}$ to **15** in equation 2.37 based on grid search on the validation set.

Different visual metrics such histograms, calibration curves, ROC curve and PR curve of cdp, mdp and their Laplace approximation versions are showed in the following. It can be seen 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.

By comparing the Laplace approximation between cdp and mdp, it can be seen 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.

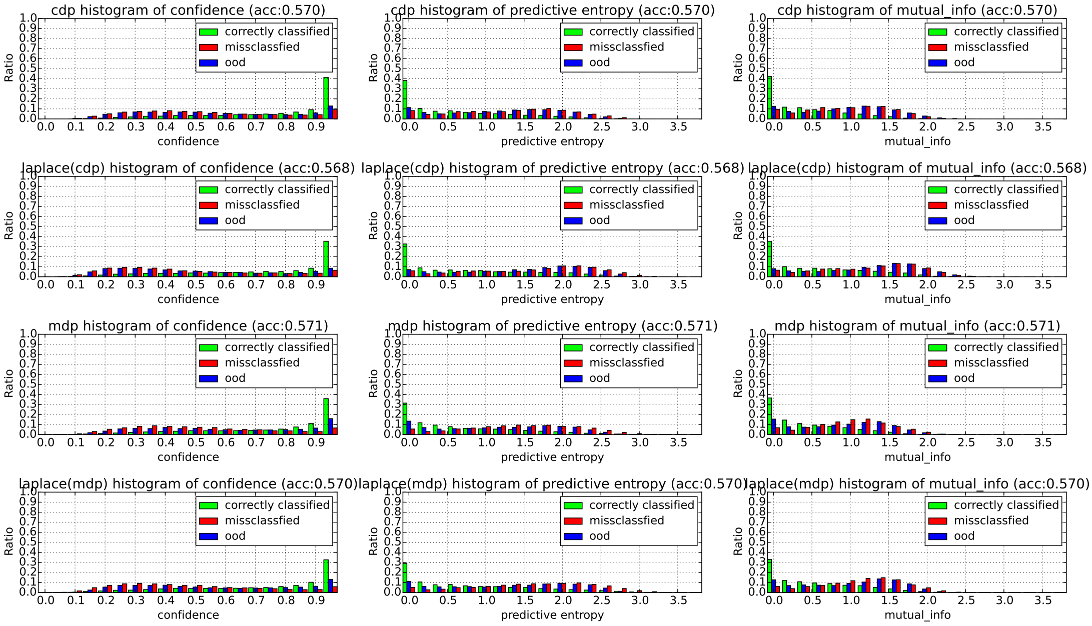

Figure 4.10: Uncertainty histogram of cdp, mdp and their Laplace approximation versions with confidence, predictive entropy and mutual information as uncertainty measure in one of three runs.

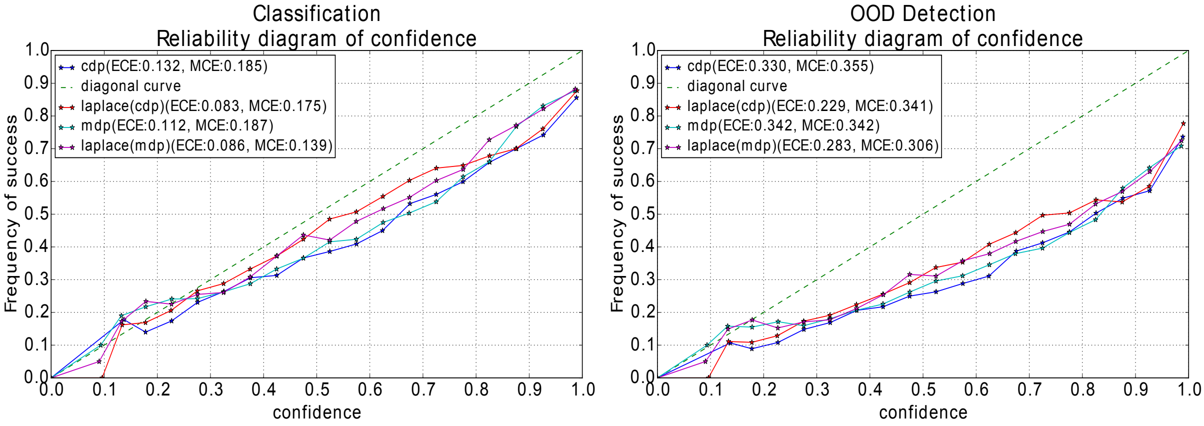

Figure 4.11: Calibration curve of cdp, mdp and their Laplace approximation versions in one of three runs.

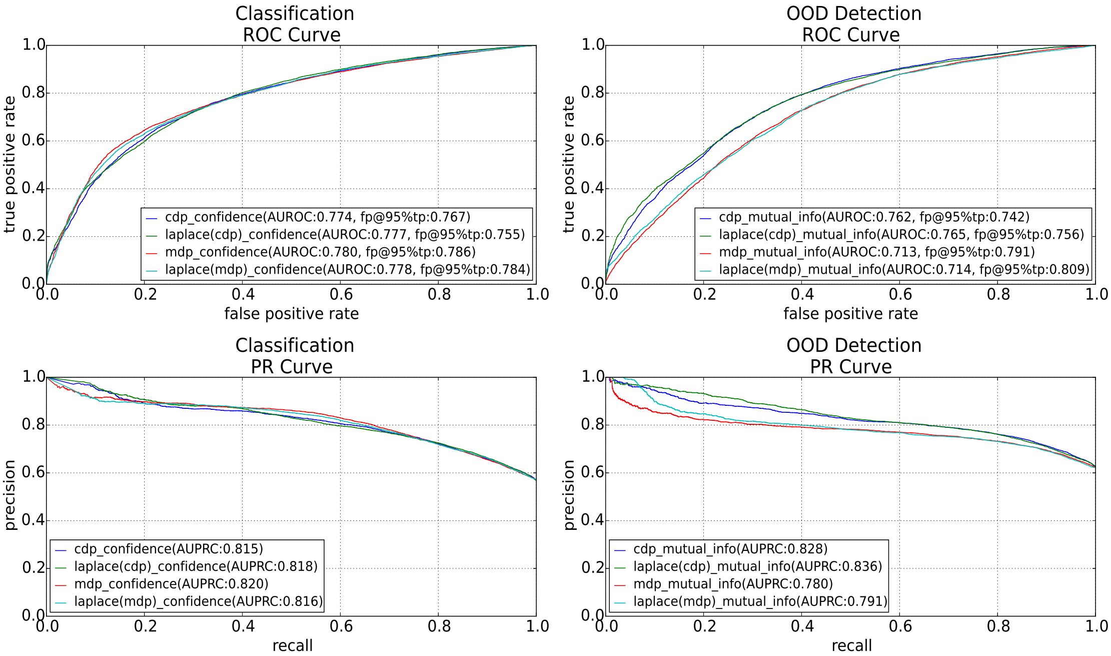

Figure 4.12: ROC and PR curve of cdp, mdp and their Laplace approximation versions in one of three runs.

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 can be seen 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, a ablation study is conducted to investigate the influence of aleatoric uncertainty which comes from mainly the feature extractor part. To this end, assumption is made that the feature extractor part has a dominant effect on the aleatoric uncertainty, because it's deterministic compared with the following Multi-layer Perceptron (MLP) classifier.

Considering this, the parameters of feature extractor are frozen 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 the results of this ablation study of the same metrics used before are shown. It's clear that the accuracy would drop when the feature extractor is frozen. However, it can be seen 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, it can inferred 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.

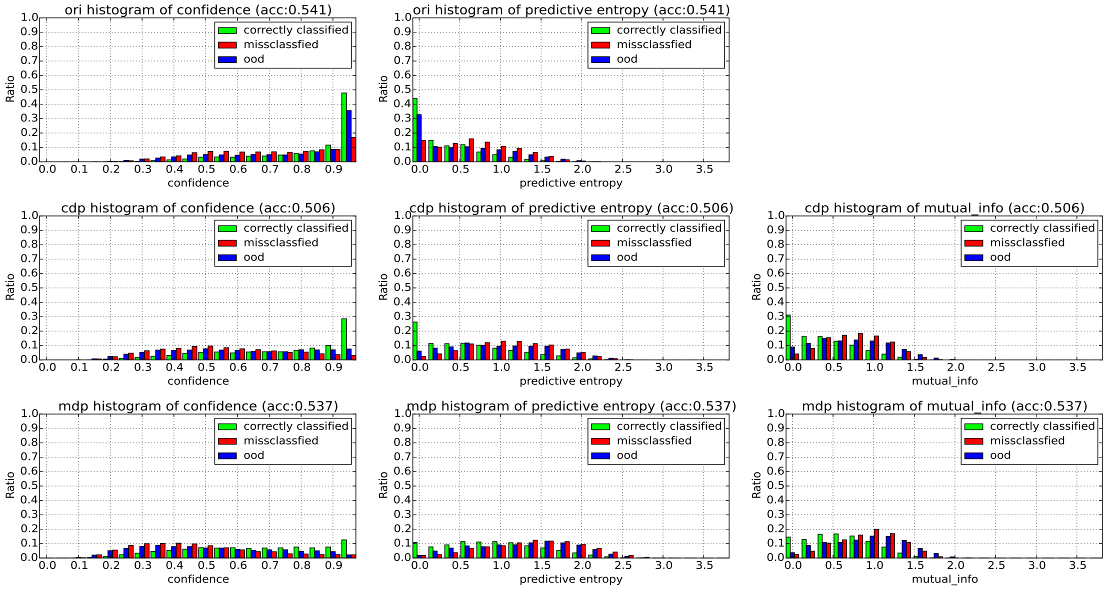

Figure 4.13: Uncertainty histograms of ori, cdp, mdp trained with frozen features in one of three runs.

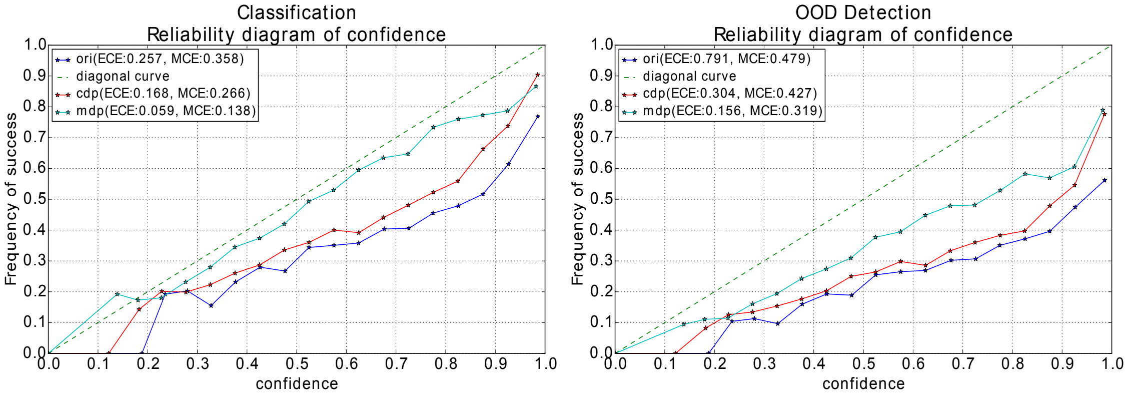

Figure 4.14: Calibration curves of ori, cdp, mdp trained with frozen features one of three runs.

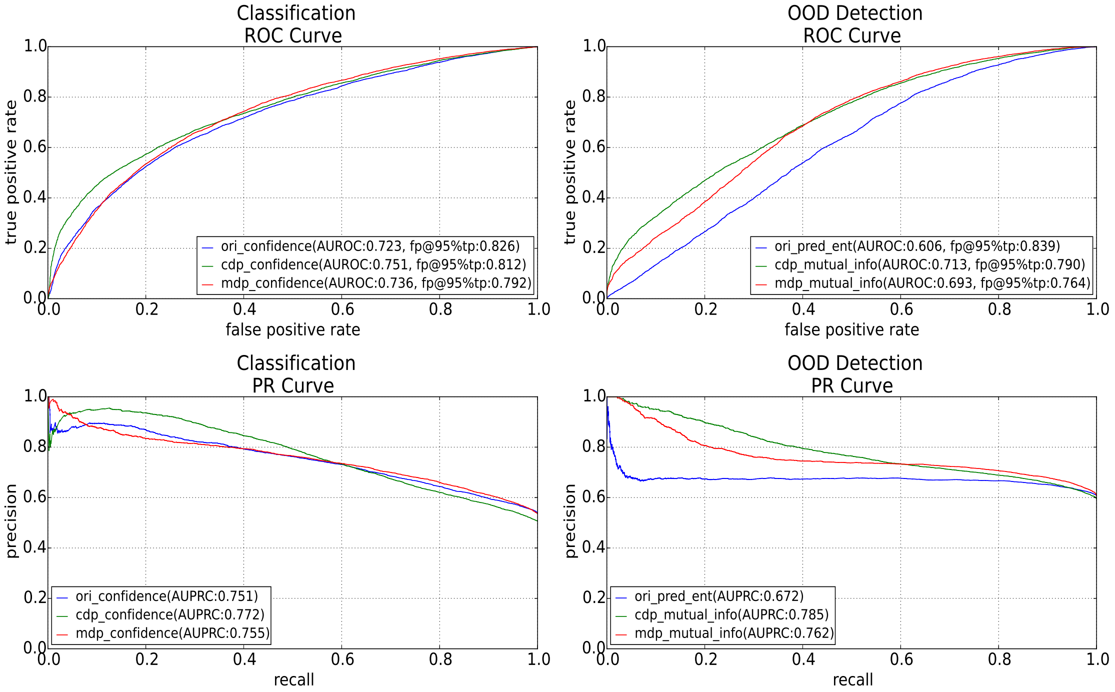

Figure 4.15: ROC and PR curves of ori, cdp, mdp trained with frozen features one of three runs.

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, the 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 (cf. 3.5). The automatically labeled data is collected based on improved uncertainty estimation. Because ResNet50 with concrete dropout shows more stable results than other version, it's used 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, experiment are firstly performed on WRGBD and UniHB dataset with restriction on the size of collected dataset.

Then the experiment is extended to another dataset, T-LESS dataset, to test its generalization ability in second part. 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. Data augmentation is employed 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, BNN with entire WRGBD dataset are trained and evaluated on objects of elevation **30°** and **60°** in UniHB dataset, which is called **adaptation dataset**. The size of adaptation dataset in this experiment is around ∼17.1K. The objects of **45°** with size ∼8.6K 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 used in this part is confidence because OOD data is not considered in this part and confidence performs better in separating correct predictions and miss-classifications based on the results of previous experiments.

Automatic Labeling: One major restriction in this experiment is the size of dataset for fine-tuning. Because if the size is not restricted, 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, which is fixed to 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 how two factors may influence the performance of fine-tuning can be investigated. 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, different settings are described and capital letters are used to denote them.

The steps of **selecting automatically labeled data** in this experiment is firstly explained, which is a little different to the strategy in experiment II.

For C, D, E, F, G, the following steps are performed:

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 the data of each class are randomly chosen according to 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.It is assumed that all the predictions except for automatic labeling have been manually labeled and a *proof-of-concept* experiment is performed with a **as balanced as possible** dataset in F and G.

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%

Fine-tuning: After collecting the dataset with different settings, the results of BNN fine-tuned with those dataset on the final test set which contains only objects of **45°** from UniHB data is shown in the table 4.3.1. The following observations can drawn from the results:

- 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.

- by comparing performance of E, F and G, it is known 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, 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, it is can be seen 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, a conclusion can be drawn 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, original ResNet50 and BNN are trained on synthetic dataset introduced in section 4.1.1. Then automatically labeled data is collected by testing the trained model on real single objects (original training set of T-LESS) which is called **adaptation dataset**. The difference between the strategy of selecting data in experiment I is that threshold used for selecting data based on uncertainty is chosen on validation set. The accuracy of predictions with uncertainty smaller than this threshold is set as 95%. Validation set is split off from training set with ratio 2:8. The reason why this way is selected to choose threshold is that it's more practical and closer to the real world applications. In this experiment predictive entropy is selected 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: The comparisons of uncertainty estimation are shown in term of different evaluation metrics with BNN and original ResNet50 in the following figures. As can be seen 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 Fig.4.17, BNN expresses better performance in terms of both calibration and separability metrics.

With the same procedure of selecting automatically labeled data, the size of dataset is around ∼1.05K with only 93% accuracy using original ResNet50, but around ∼1.6K with

96% accuracy using BNN. Because original ResNet50 produces less automatically labeled data with lower quality, the dataset for fine-tuning is collected with BNN.

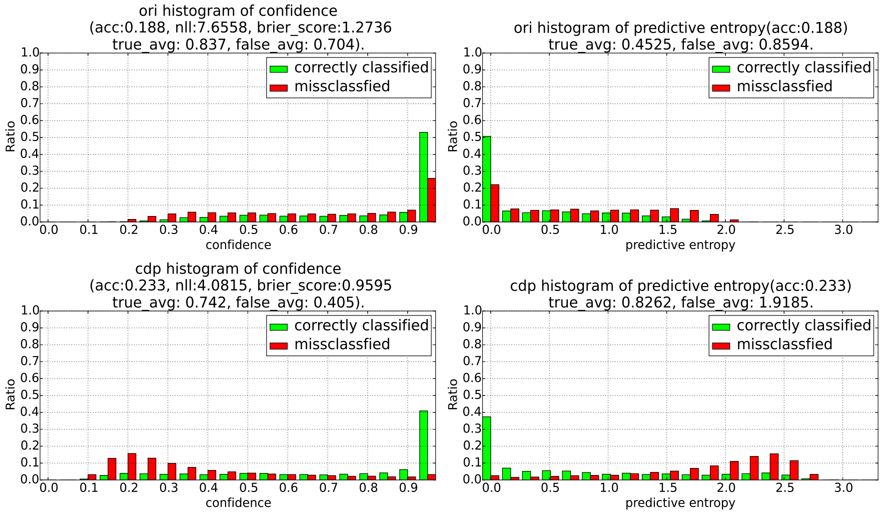

Figure 4.16: Uncertainty histograms of original ResNet50 (top) and BNN (down).

However, as found in experiment I, the problem of imbalance in number of data samples of each class still appears on this dataset and has a large influence on the performance. The horizontal histogram in Fig.4.18 reveals this problem, in which most of confident predictions cluster in few classes which are class 6, 7, 28 and so on. To mitigate this problem, two simple ways are adopted here:

- to manually label data uniformly with size 3% of the adaptation set(around ∼0.9K data samples).
- to employ augmentations described in section 4.1.1 to the objects to balance the dataset as much as possible.

The balance of dataset with/without manually labeled data after augmentations are showed in Fig.4.18. By using augmentations, datasets are more balanced than before. But the balance before augmentations has significant effect on the performance, which is discussed in the following.

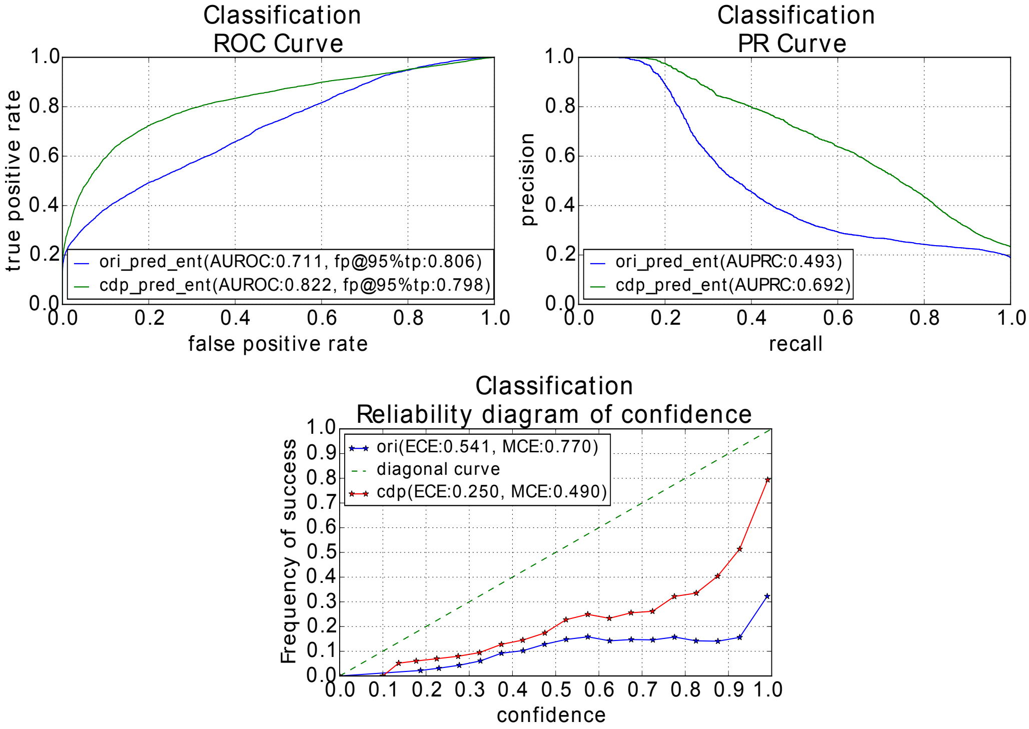

Figure 4.17: Calibration curves, ROC curves, PR curves of original ResNet50 and BNN.

Fine-tuning: BNNs trained with only synthetic data previously are fine-tuned on dataset collected with different strategies. Table 4.3.2 gives the results of BNN trained with different dataset. They are all tested on the test set of T-LESS dataset. By comparing A and B, the performance drop caused by the domain gap between synthetic objects and real ones is quite large, around 35%. By fine-tuning model on dataset collected based on uncertainty estimation, the accuracy increases a lot. With more balanced dataset, the accuracy even exceed the model trained with augmented entire real dataset by around 3%. By comparing C and D, it's obvious that the balance of data samples in each class before augmentation is important to obtain more accuracy gain. The reason behind that is the diversity of data samples before augmentation. Although augmentation can help the model generalize better by increasing the number of data samples, if the distribution of training set is poorly represented by few training samples, the way does not help much. This can also be observed in comparison of the corresponding confusion matrices in Fig.4.19, even though the number of data samples in some classes are similar after augmentation, their accuracies differ a lot (35% vs. 65% in class 0).

Table 4.5: Results of BNN trained with different dataset (size of dataset before augmentations is showed in the bracket).

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

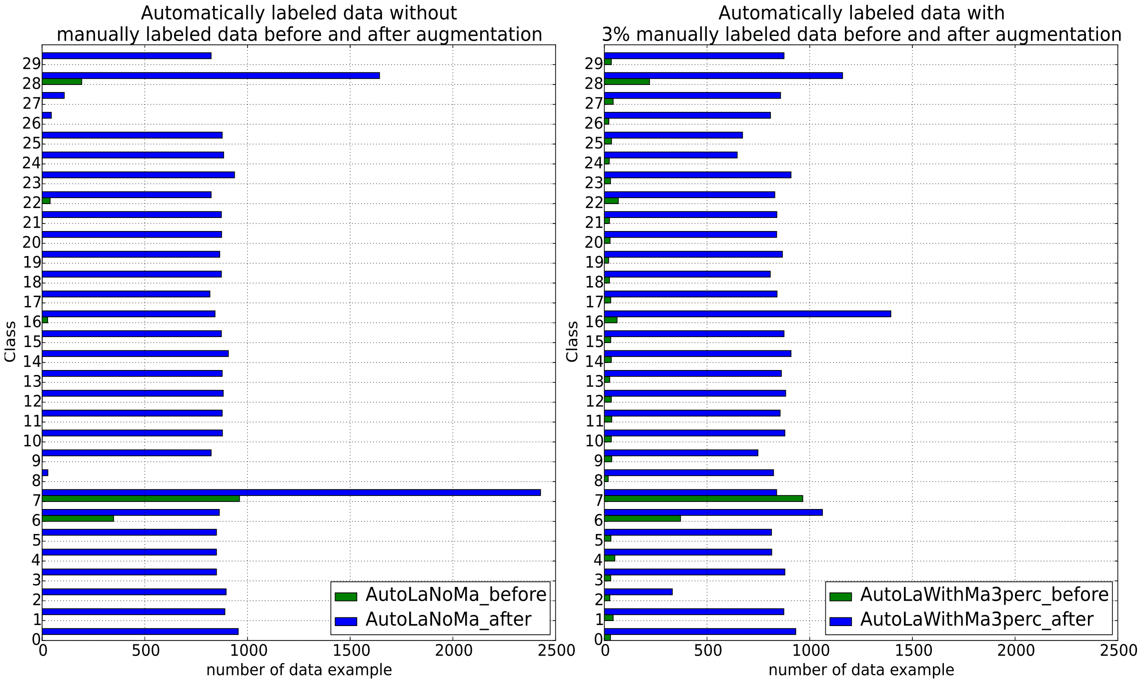

Figure 4.18: Horizontal histograms of automatically labeled data without (left) and with (right) 3% manually labeled data before/after augmentations.

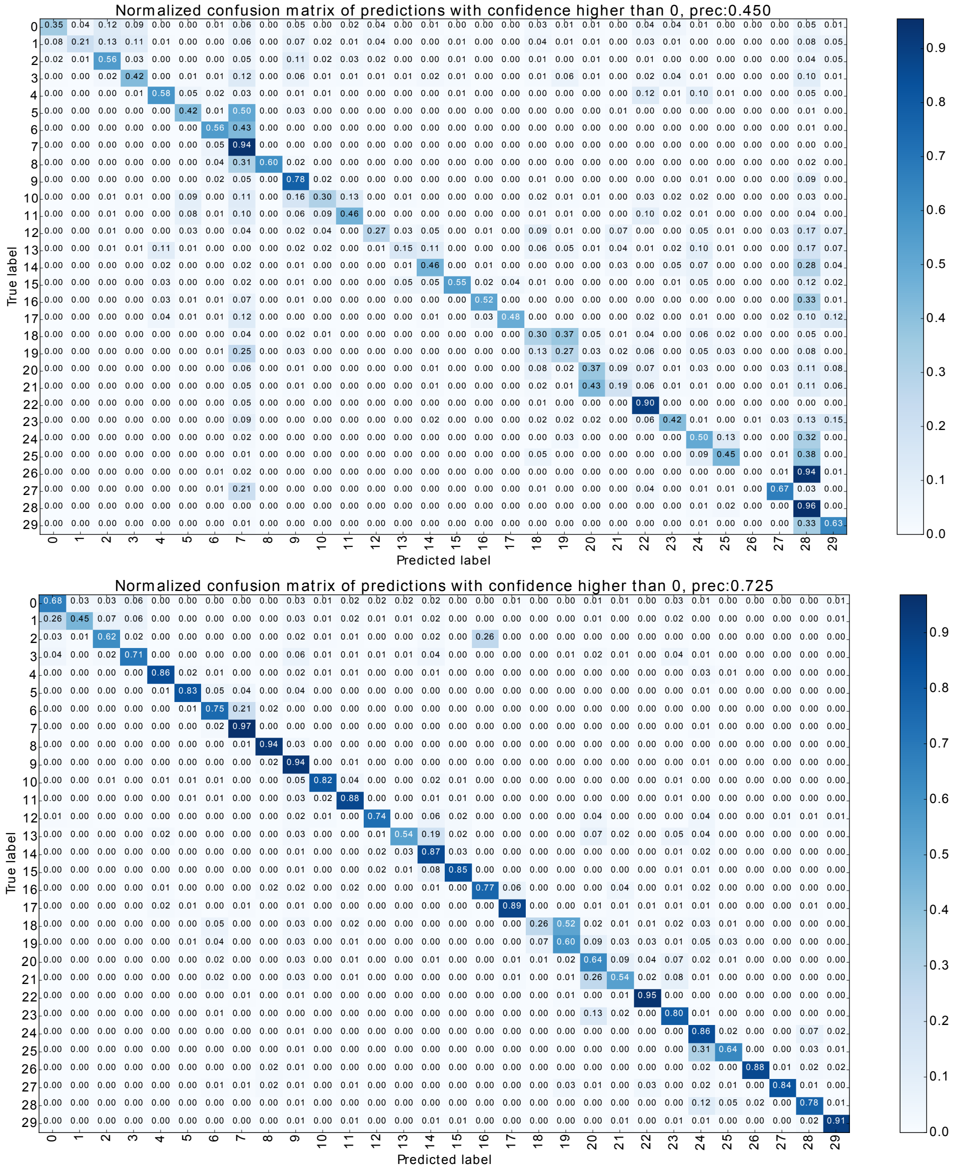

Figure 4.19: Confusion matrices of BNN fine-tuned with dataset without (top) and with (down) small portion of manually labeled data.

4.4 Context-based improvement experiments

In this section, experiments are performed to investigate the effect of combining improved uncertainty estimation with a probabilistic graphical model in down-stream task. In detail, a CRF is employed to capture the dependencies between objects by exploiting contextual information in the scene(cf. Fig.3.3). The intuition behind this idea is that, if the predictive distribution from BNN is more reliable and can represent the inherent randomness better.

Then this bit of information can be utilized by CRF for further performance gain. Similar to previous experiments, ResNet50 with concrete dropout is used to improve uncertainty estimation in the following experiments and we call it **BNN** without specifications.

Firstly a simpler experiment is performed on subset of T-LESS dataset to show that CRF can capture the dependencies and improve the accuracy, which can be boosted with better uncertainty estimation from BNN. Secondly, the same idea is tested on the entire T-LESS dataset to show the generalization ability of this approach.

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

In this experiment, original ResNet50 and BNN are trained on the real single objects with black background (original training set of T-LESS dataset). Then a subset of test scenes of original T-LESS dataset are used for training and testing CRF. In detail, scene 1, 2, 3, 4 (∼8K objects and ∼2K scenes) are used for training CRF, and scene 5, 6, 7, 8 (∼12K objects and ∼2K scenes) for testing. To note that **no augmentations** are applied in this experiment. To note that, based on the prior knowledge about the testing set, that no objects repeat in the test scenes, the diagonal entries of binary co-occurrence matrix is set to -1 to penalize the outcome with two same objects.

In the table 4.6, results of CRF trained with two different unary features from original ResNet50 and BNN are given. As can be seen, CRF trained and tested with unary feature from BNN can yield more improvement(13.17%) compared with that from deterministic network (12.42%).

Table 4.6: 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 CRF model act as coefficients for two kinds of features, the magnitude of these two weights represents their contributions to overall performance. In the table 4.7, the weight of unary feature $\boldsymbol{\theta_u}$ from BNN is larger than the one of edge feature $\boldsymbol{\theta_p}$. The situation is inverse in unary feature from original ResNet50. It's evident that the unary feature from BNN contributes more than the pairwise one while the one from original ResNet50 contributes less. It also shows that the predictive distribution from BNN contains more information than its counterpart from original ResNet50. More importantly, this information can be utilized to improve the performance which is shown in the table 4.6.

Table 4.7: Learned weights of CRF trained with unary feature from different versions of ResNet50

	node weight $\boldsymbol{\theta_u}$	edge weight $\boldsymbol{\theta_p}$
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, approach of combination of BNN and CRF is tested on the whole T-LESS dataset. Based on the results obtained in subsection 4.3.2, the accuracy should be improved further by exploiting contextual information with CRF (cf. Fig.3.4). In details, CRF are trained with unary features extracted from BNN before fine-tuning with size around ∼24.5 objects and ∼5K scenes (cf. Fig.3.5). In testing, the whole testing set of original T-LESS dataset with size ∼70K objects and ∼10K scenes, whose unary features are extracted from fine-tuned BNN.

Because the combinations in the whole test set are more complex than its subset (scene 5, 6, 7, 8) in experiment I, different values on the diagonal are explored for they encode different prior knowledge. There are many objects occurring repeatedly in other scenes of test set besides scene 5, 6, 7, 8. Therefore there is a trade-off between different kind of prior knowledge. Filling diagonal entries with 1 indicates that co-occurrence of same objects is more likely to happen and it suits better on the whole test set. As can be seen in the table 4.4.2, CRF trained with pairwise matrix with 1 on diagonal has most improvement (4%) when comparing with matrices with other diagonal values.

Table 4.8: Results of CRF with different type of potentials

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 Summary

In this work, different approaches for the improvement of a classification system related to uncertainty are investigated.

Firstly, in order to enable a classifier to adapt itself to a test environment which has a domain gap with its training set, dataset for fine-tuning should be collected with as little human efforts as possible. Therefore it is proposed to gather automatically labeled data based on the uncertainty estimation of the classifier. Considering the overconfident behavior of deep learning classifiers, a Bayesian neural network is employed to improve the uncertainty estimation. For easy compatibility with existing neural network architectures, dropout variational inference and scalable Laplace approximation are chosen for this task. In addition to the naive version, advanced variants of them such as concrete dropout, multiple dropout and ensemble of them are further taken into account. The original ResNet50 is modified to incorporate these techniques and keep the ability of extracting powerful features at the same time. Extensive experiments on WRGBD and UniHb dataset are performed to evaluate the performance of the uncertainty estimation of different approaches in terms of different metrics including accuracy, proper scoring rules, calibration metrics, and separability metrics. The results show that Bayesian neural network can improve the performance of both accuracy and calibration as well as uncertainty estimation when compared with the original ResNet50.

Secondly, the improved uncertainty estimation is used for collecting automatically labeled dataset, which is used for fine-tuning the model trained on a easily obtainable dataset. The goal is to obtain a more accurate domain-specific classifier with as little human efforts as possible. Experiments of this part are performed on both WRGBD/UniHB dataset and original T-LESS/synthetic T-LESS dataset. During the experiments, it is found out that the balance of data samples in each class has a significant influence on the performance after fine-tuning. Augmentations are employed to mitigate this problem and the experiments results show that, a more accurate domain-specific can be obtained with much less human efforts, while its performance can reach or even exceed the performance of the classifier obtained with much more human efforts.

Thirdly, the improved uncertainty estimation should be utilized further with a proper probabilistic model. On the other hand, uncertainty induced by appearances similarity should be handled by modeling the contextual information between objects. On account of these two points, a conditional random field is employed to further improve the performance by utilizing more information from the output of the Bayesian neural network and capturing the contextual relationship between objects in the same scene. For the end, experiments are performed to validate this idea but only on the T-LESS dataset because designing pairwise features in WRGBD dataset is non-trivial. The experimental results show that the CRF can improve the accuracy further by not only making use of the information from better uncertainty estimation but exploiting the contextual relationship between objects.

Although the experimental results show that these approaches can achieve better performance compared to the original version, the problems occurring when constructing approaches and performing experiments should require more attention and efforts towards solving them. There are several main problems listed in the following:

- inefficiency in testing with BNN: testing each input requires sampling the posterior many times. Different approaches are already available such as network[HVD15] distillation and parallel computation.
- Uncertainty estimation should be defined more clearly and explored with more efforts. For example, the difference of separability metrics and calibration metrics are used to evaluate the uncertainty estimation in literatures[HG16][GPSW17]. However, they should account for different abilities which should not be summarized with one word. As can be seen in the experiments, well calibrated models may have similar or even worse separability between correct predictions versus miss-classifications or OOD predictions than the badly-calibrated model.
- Other approaches to improve uncertainty estimations should be considered. A Bayesian neural network is a principal way for this purpose. But the computation behind that is too high. It is like using a generative model to solve a problem which can be solved by a discriminative model. Maybe there is a more efficient way to achieve this goal such as different post-processing methods [GPSW17].
- the way to choose highly confident predictions as automatically labeled data for continuous learning sounds practical and meaningful. However, from the perspective of active learning, the confident predictions contain little information required by the classifier. To train a classifier with a better performance requires a training set with more information. To label predictions with more information (low confidence) sounds more meaningful. Therefore a better technique to combine them should be taken into account.

- modeling dependencies between random variables is a promising and natural way to improve the results because the i.i.d. assumption does not hold all the time. However, employing probabilistic graphical models requires hand-crafted edge feature which is non-trivial to have and design. Therefore, graph neural networks which can learn the edge feature based on data is promising to work on.

5.2 Conclusion

- Bayesian neural network can improve uncertainty estimation in terms of different evaluation metrics including accuracy, proper scoring rule, calibration and separability between correct predictions versus miss-classifications or OOD predictions, while the ensemble of them can achieve even better performance.
- Multiple dropout can work better without considering OOD data, while concrete dropout can offer more stable performance on separating correct predictions and miss-classifications.
- Scalable Laplace approximation with Kronecker-factored approximation can achieve a similar performance as the concrete dropout and multiple dropout, indicating that the performance is highly related to the point estimate obtained during training.
- Manual efforts in collecting datasets for fine-tuning to enable classifiers to adapt itself to test environments can be reduced with the help of better uncertainty estimation. Nevertheless, the imbalance of a collected dataset has a negative effect on fine-tuning, which can be mitigated by data augmentation and adding little manual labeling.
- A conditional random field can improve the performance further by utilizing information brought by better uncertainty estimation and exploiting contextual information between objects in the same scene.

5.3 Future work

There are interesting works related to the approaches used in this work:

- Uncertainty distributions of different view points of the object can be checked. On the one hand, to understand the behavior of the neural network better with uncertainty. On the other hand, if the uncertainty can represent the actual discriminability of different view points, this information is useful in down-stream tasks such as optimal view point planing for grasping and so on.
- As mentioned in summary, inefficiency in testing is an important problem. It would be interesting to try a distillation method to decrease the runtime during testing.
- Although in this work, dropout variational inference and Laplace approximation are evaluated, the approximation distribution of them seems restrictive and simple.

Interesting future work would be to try more complex and expressive approximate posterior distributions such as normalizing flow .

- In the experiments of evaluating uncertainty estimation, multiple dropout can achieve a better performance without considering OOD data. The ablation study also shows that multiple dropout can have a better performance. As stated in [KSW15], covariances between instances increase the variances of estimating derivatives of the variational parameters. One way to improve the performance of the multiple dropout would be to decrease the variance of estimating variational parameters by sampling masks for each instance separately because there are more parameters than concrete 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 $\boldsymbol{\theta}$ is a vector. Fisher information $\boldsymbol{\mathcal{I}(\theta)}$ is defined as follows:

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

Where $\boldsymbol{f(X;\theta)}$ is probability density function or mass function of random variable $\boldsymbol{X}$. 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 $\boldsymbol{\theta}$ is the true parameter. Based on the fact that $\boldsymbol{f(X;\theta)}$ is density, we can have:

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

Differentiating the above with respect to $\boldsymbol{\theta}$ 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. $\boldsymbol{\theta}$ 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 Derivation of Re-parameterization Trick

Let $p_{\theta}(x)$ denote the density function of x parameterized by θ, $g(\cdot)$ is a differential function, ϵ is a random variable with parameter-free density function $p(\epsilon)$. The estimator of the gradients w.r.t. θ with $p(x|\epsilon) = \delta(x - g(\theta,\epsilon))$ can be derived as:

$$\begin{aligned}
I'(\theta) &= \frac{\partial}{\partial\theta}\int f(x)p_{\theta}(x)dx \\
&= \frac{\partial}{\partial\theta}\int f(x)(\int p_{\theta}(x|\epsilon)p(\epsilon)d\epsilon)dx \\
&= \frac{\partial}{\partial\theta}\int(\int f(x)p_{\theta}(x|\epsilon)dx)p(\epsilon)d\epsilon \\
&= \frac{\partial}{\partial\theta}\int(\int f(x)\delta(x-g(\theta,\epsilon))dx)p(\epsilon)d\epsilon \\
&= \frac{\partial}{\partial\theta}\int f(g(\epsilon,\theta))p(\epsilon)d\epsilon \\
&= \int \frac{\partial}{\partial\theta}f(g(\epsilon,\theta))p(\epsilon)d\epsilon \\
&= \int f'(g(\epsilon,\theta))\frac{\partial}{\partial\theta}g(\theta,\epsilon)p(\epsilon)d\epsilon \\
&= \mathbb{E}_{p(\epsilon)}\left[f'(g(\epsilon,\theta))\frac{\partial}{\partial\theta}g(\theta,\epsilon)\right]
\end{aligned} \tag{A.2}$$

□

A.3 Second Order Taylor Expansion of Log Posterior

$\boldsymbol{p(\omega|Y,X)}$ denotes the posterior over model parameters $\boldsymbol{\omega}$. $\boldsymbol{\omega^\star}$ is one optimizer obtained after training. Then the second order Taylor expansion of posterior distribution around $\boldsymbol{\omega^\star}$ is defined as follows:

$$\begin{aligned}\mathrm{log}\boldsymbol{p(\omega|Y,X)} \approx &\mathrm{log}\boldsymbol{p(\omega^\star|Y,X)}+\\ &\boldsymbol{(\omega-\omega^\star)^T\frac{\partial\mathrm{log}p(\omega|Y,X)}{\partial\omega}}+\\ &\boldsymbol{\frac{1}{2}(\omega-\omega^\star)^T\frac{\partial^2\mathrm{log}p(\omega|Y,X)}{\partial\omega^2}(\omega-\omega^\star)}\\ =&\mathrm{log}\boldsymbol{p(\omega^\star|Y,X)-\frac{1}{2}(\omega-\omega^\star)^T H(\omega-\omega^\star)}\end{aligned} \tag{A.3}$$

where the term of first derivatives vanishes because the gradients around an optimizer is zero and

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

, where $\boldsymbol{p(Y|X,\omega)}$ is the likelihood function, $\boldsymbol{p(\omega)}$ is the prior distribution over model parameters.

□

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.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[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.