FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

LTE
Lehrstuhl für Technische Elektronik

Lehrstuhl für Technische Elektronik
Prof. Dr.-Ing. Dr.-Ing. habil. Robert Weigel
Prof. Dr.-Ing. Georg Fischer

Master's Thesis

im Studiengang
"Elektrotechnik, Elektronik und Informationstechnik (EEI)"

von

Raza Ul Azam

zum Thema

Interpretation and Classification of Bayesian Uncertainty for Vision based Classification Tasks

Betreuer:	Prof. Dr.-Ing. Dr.-Ing. habil. Robert Weigel William H. Beluch Anand Dubey
Beginn:	25.02.2020
Abgabe:	25.08.2020

Declaration

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 25.08.2020

Raza Ul Azam

Acknowledgements

Firstly, I would like to thank William Harris Beluch, Anand Dubey and Prof. Robert Weigel for their valuable time, guidance and supervision during the process of completion associated with this Master's thesis. I highly appreciate their meaningful comments and discussions. Secondly, I would like to thank my family and friends, especially my parents, who have been a source of endless support and help during the time when I was working on my thesis.

Lastly, I would like to thank Robert Bosch GmbH for providing me with the opportunity to write my thesis in joint collaboration with my university (FAU Erlangen-Nuremberg) and for funding six months of my thesis.

Abstract

The advent of strong tools in Deep learning have gained the attention of researchers from a wide variety of fields such as Computer vision, Natural language processing and so forth. Recently, these tools have started being used for some Medical applications as well. Typically, we train a Neural network such as Convolutional neural network with strong regularization techniques like dropout for achieving the desired results. We try our best in collecting the extensive amount of data for making our network acquainted with every possible scenario. However, in real-world settings we can come across some unwanted situations that make our network behave unreliably. For applications such as Autonomous driving or Medical diagnosis where the lives of humans could be at stake, we can not afford to have our network make unreliable decisions. Thus, estimation of the uncertainty in the predictions is important. In the literature, a couple of strategies have been proposed for this purpose that make the estimation of the different uncertainties scalable in real-world applications. However, we still need to decide the way in which we should process this uncertainty information for enabling our networks to make smart decisions.

In this work, we make use of the work presented by the authors in [1] for the root task of uncertainty estimation. Initially, we conduct experiments in the Semantic segmentation setting where we validate the empirical observations of the authors with two different modern state-of-the-art network architectures. These networks possess some complexities that could potentially interfere with this method. Thus, we conduct different experiments and perform an in-depth analysis of the method for deriving strong conclusions. Additionally, we measure the quality of the uncertainty estimates with the metric presented by the authors in [2] and comment on its power by performing a correlation analysis with the performance metric of our networks. Further, we present our insights on different aspects based on the results of these experiments.

Apart from that, we perform a set of experiments in the Image classification setting with the main goal of observing whether our selected method can produce uncertainty estimates that correlate with the complexities in the input data space. For this purpose, we rely on our inductive belief and create artificial samples that could probably make the network produce unreliable decisions. We conduct several experiments with various datasets and closely observe the uncertainty estimates for different data samples. Following that, we introduce a new uncertainty based classifier which could be used for identifying the unreliable samples, just based on their estimates for the different uncertainties produced by our network. Additionally, we formulate a regularized loss function which helps in improving the performance of our classifier. We give a detailed overview of the efficiency of this classifier in different experimental settings.

Contents

Abbreviations

AUROC Area Under the Receiver Operating Characteristic

Bayesian CNN Bayesian Convolutional Neural Network

Bayesian NN Bayesian Neural Network

CNN Convolutional Neural Network

DNN Deep Neural Network

e.g. Exempli gratia (“for example”)

ECE Expected Calibration Error

ELBO Evidence Lower Bound

GP Gaussian Process

GPU Graphical Processing Unit

HMC Hamiltonian Monte Carlo

i.e. Id est (“that is”)

ID In-Distribution

KL Kullback-Liebler

MC Monte Carlo

MCE Maximum Calibration Error

MCMC Markov Chain Monte Carlo

mIoU mean Intersection over Union

MLP Multi-layer Perceptron

OOD Out-of-Distribution

PAvsPU Patch Accuracy vs Patch Uncertainty

PSPNet Pyramid Scene Parsing Network

ReLU Rectified Linear Unit

ResNet Residual Network

SGLD Stochastic Gradient Langevin Dynamics

VI Variational Inference

1 Introduction: The Importance of Uncertainty Estimation

Deep learning is a rapidly advancing field which is being integrated in diverse domains for achieving better results. Based on the type of the application, we use a particular network architecture that is capable of extracting meaningful features from our input data for producing the desired outputs. Over the past few years, a lot of research has been done for improving the capability of these networks for the task of prediction however we still commonly encounter situations where our network fails to produce the reliable results even though it has been trained to the point of perfection. This mostly happens when our network processes the data which it does not have any knowledge of or when the data is very noisy. In such situations, we get the predictions that can prove to be disastrous in applications such as Autonomous driving. Considering that, if we can enable our network to make the predictions along with estimating their corresponding confidence levels (uncertainty estimates) then we can avoid the undesirable consequences of these uncertain predictions.

In standard deep learning, we represent the predictions as well as the parameters of a network as *point estimates*. Considering the supervised training process, we aim to learn a *deterministic function* mapping between our input and the output data. Such deterministic mappings are not able to represent the uncertainty information. For incoporating this, there is another direction of work which falls under the domain of *Bayesian machine learning*. In this field, we deal with the probabilistic networks that are capable of producing the predictions as well as the uncertainty information at the same time. The motivation is derived from the strong models such as Gaussian processes that define a probability distribution over the function mappings which enables them to incorporate the uncertainty information.

Considering the importance of the uncertainty estimation, recently, a couple of strategies have been proposed that make it possible to extract the uncertainty information from the neural networks along with the predictions in a scalable way. In this work, we select one of these methods, identify some loopholes for proposing the possible research questions, and perform several experiments for deriving strong answers to these questons. During this process, we closely observe the patterns in our uncertainty estimation task and exploit them to propose a classifier design which could be used for identifying the unreliable data, if it being fed to our main network. The application area which we put focus on is Autonomous driving which makes use of the common Computer vision tasks such as Image classification and Semantic segmentation. The core of these tasks is the neural network design which is named as *Convolutional neural network*. We will shed some light on this particular network in the next section.

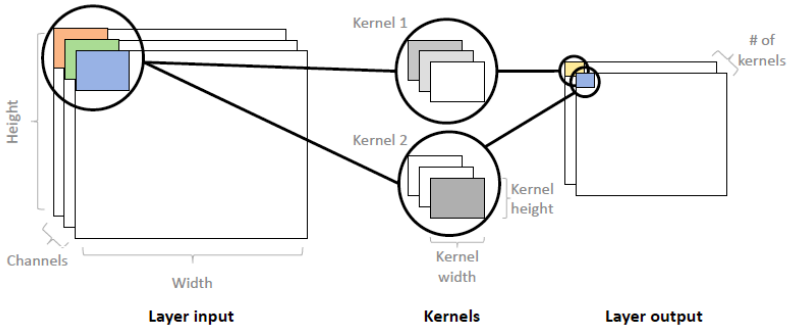


Figure 1.1: Visualization of a convolutional layer [4].

1.1 Convolutional neural network

Convolutional neural networks (CNNs) [3] are a class of deep neural networks that are commonly used in the applications where we have to process the images for making an informed decision. These networks exploit the spatial correlation of the pixels in an image for extracting the fine-grained features that represent some meaningful information. As the name indicates, we use the mathematical operation called *convolution*, which is linear. In comparison with a simple feed-forward neural network, where every neuron in the next layer is connected to the neurons in the current layer with a unique weight, CNNs possess a shared-weight network architecture. We define a number of weight matrices called *kernels* that we separately traverse on our input image for extracting the unique feature maps. We refer to the size of these kernels as the receptive field of a neuron which defines a certain portion of the image which that neuron observes. Following the extraction of the feature maps, we stack all of them together for representing the output from a convolutional layer. In a deep CNN, we stack multiple such layers together where the information extracted by one layer is passed on to the other for learning some high-dimensional concepts that the network make use of for giving its output. Figure 1.1 shows the visualization of a convolutional layer in a CNN.

Typically, in a CNN, we use a pooling layer in combination with the convolutional layer. The main purpose of this layer is to subsample the feature maps in the spatial dimension so that our convolutional layers can only focus on processing the essential information. Intuitively speaking, this pooling layer also help in making a CNN invariant to the translations such that it does not get confused with the rotated version of an image.

In this work, we explicitly work with the Computer vision applications that employ CNNs. We focus on the task of the uncertainty estimation in these applications using a state-of-the-art method. But, the question is, what are the different types of uncertainty estimates that are important in Computer vision? We would present some details about it in the next section.

1.2 Important uncertainties in Computer vision

In Computer vision, there are two different type of uncertainties that we commonly deal with. They are named as the *epistemic* uncertainty and the *aleatoric* uncertainty. Combining these two different uncertainties together, we get the *predictive* uncertainty. For explaining the epis-

temic uncertainty, let us consider an example: Imagine that we train a CNN on the images of cats and dogs. It is a binary classification task where we only have two output classes. Now, at the test-time, if we feed our network an image of a cat/dog, it would most probably be able to classify it accurately. But, what if we feed it an image of a leopard? In this situation, our network has encountered an input which it has not ever seen before, so it would make a random prediction and assign it either the dog or the cat class with high confidence. However, it should assign an equal confidence level to both of the classes instead of just being highly confident about one. This is where the epistemic uncertainty plays its role. We observe a high value of the epistemic uncertainty for the situations which our model has not seen before. In real-world applications such as Autonomous driving, there are high chances of such scenarios to appear so measuring this uncertainty can significantly help in making smart decisions.

The second type of the uncertainty which we have is the aleatoric uncertainty. This uncertainty estimate basically captures the noise inherent in our data. For training the networks that are deployed in the Autonomous cars, we firstly capture the data from the sensors such as camera. As the quality of the images is limited according to the resolution of the camera, we are definitely bound to observe some noise. Although, this noise might not be that evident for a normal human eye but, for the neural networks, it can potentially interfere with their inner working. Thus, the aleatoric uncertainty quantifies the noise in the data, which is induced from a measurement sensor - in this case, a camera. For the images where our network observe a high amount of noise and encounter difficulties while making the predictions, it would produce a high estimate for the aleatoric uncertainty.

1.3 Main contributions

As mentioned before, we concentrate on the task of uncertainty estimation in Computer vision problems. Out of all the possibilites, in this thesis, we specifically focus on the Image classification [5] and the Semantic segmentation [6] task. Instead of focusing on the development of the uncertainty estimation tools, we rather select an aleady existing state-of-the-art approach, perform an in-depth analysis of that method in some complex situations and subsequently propose some new ideas.

In the bayesian machine learning community, the researchers have proposed a couple of methods that let us compute the different uncertainty estimates in a scalable way. However, most of these approaches either focus on computing the epistemic uncertainty alone or the aleatoric uncertainty. Nevertheless, the authors in [1] introduced a method which lets us compute both of these uncertainties at the same time. For that purpose, they present a probabilistic interpretation of the regularization technique called dropout and combine it with an observation noise model. The authors demonstrate the efficacy of their method in the Semantic segmentation and the Depth regression [7] settings. However, there are some ambiguities in their work which we identify and propose some possible research questions out of them. For answering these questions, we conduct a separate set of experiments in the Semantic segmentation and the Image classification settings.

In the Semantic segmentation task, we work with two different state-of-the-art network architectures called Pyramid Scene Parsing Network [8] and DeepLabv3+ [9]. We quantify the different uncertainties using our selected method and use some common metrics for estimating the quality of these uncertainties. However, these quality metrics generally help in analyzing

the epistemic uncertainty quality only. For getting an idea of the quality for both the aleatoric and the epistemic uncertainty, we measure an additional metric presented by the authors in [2]. We observe whether the quality of the uncertainty estimates actually correlate with the performance of our network.

For the Image classification task, we work with three different datasets, named as MNIST [10], FashionMNIST [11] and CIFAR10 [12], and network architectures, named as LeNet5 [13], VGG-7 [14] and ResNet50 [15], respectively. We put samples, that might potentially have either the high epistemic or the aleatoric uncertainty, in our dataset and observe whether the method presented by the authors in [1] produce uncertainty estimates that correlate with these complexities. Following that, we propose a new classifier design which could be used to identify the high uncertainty samples. Finally, we introduce a regularized loss function which helps in improving the performance of our classifier, if we train our main network with it.

2 Uncertainty Estimation in Deep Learning

In order to get an estimate of uncertainties in deep learning, we need to resort to Bayesian neural networks which are derived in the Bayesian probability theory framework. For performing inference in these networks, we use a method called Variational inference. This chapter presents a brief overview of the main ideas underlying Bayesian probability theory framework, Bayesian neural networks and Variational inference. In addition, this chapter discusses different type of uncertainties which exist in deep learning and some popular state-of-the-art approaches which can be used for getting an estimate of these uncertainties.

2.1 Bayesian probability theory framework

In supervised learning settings, we are given some observed input data $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ and the output data $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\}$, which can concisely be written as $D_N = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$, where "$N$" denotes the total number of data points. In the literature, this dataset D_N is commonly referred to as the *training* data. In the context of deep learning, we are interested in learning a mapping $\boldsymbol{f : X \rightarrow Y}$ which is parametrized by the weights $\boldsymbol{w}$ of a neural network. However, the way in which this mapping is learned, differs for deterministic deep learning and its bayesian counterpart. In the deterministic formulation, we randomly initialize the set of parameters $\boldsymbol{w}$ followed by iteratively optimizing a loss function (dependant on the parameters), by using stochastic optimization techniques, for learning the set of parameters which best represent the mapping $\boldsymbol{f}$. On the other hand, we follow a completely different approach in the bayesian framework.

Taking into account the bayesian probability theory framework, we formulate everything in the probabilistic domain and rely on *Bayes' theorem* for deriving the main result. Given input data $\mathbf{X}$, we are interested in finding the set of parameters $\boldsymbol{w}$ that are likely to have generated the corresponding output data $\mathbf{Y}$ [4]. For this purpose, we impose a *prior* probability distribution over the parameter space $p(\boldsymbol{w})$, which captures our initial belief of the likely parameter set before observing any data point from the dataset D_N. We then proceed with defining a *likelihood* probability distribution $p(\mathbf{y}|\mathbf{x}, \boldsymbol{w})$, which serves the purpose of generating the outputs given a set of parameters $\boldsymbol{w}$ and inputs. This likelihood distribution also helps in evaluating the quality of our initial belief and updating it as we keep on observing more data. The choice of the likelihood probability distribution depends purely on the application but, commonly, we consider softmax and gaussian likelihood for classification and regression settings respectively [4].

With the dataset $\{\mathbf{X}, \mathbf{Y}\}$ and already evaluated prior and likelihood distributions at hand, we then make use of the Bayes' theorem to compute the *posterior* probability distribution over the

parameter space:

$$p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{w})p(\boldsymbol{w})}{p(\mathbf{Y}|\mathbf{X})}. \tag{2.1}$$

This posterior distribution represents our updated prior belief, after observing all the data points in the dataset, and serves as a major tool for computing the predictions on new data points, along with the likelihood distribution. Given a new data point $\mathbf{x}^*$, we can compute the predicted output $\mathbf{y}^*$ by a process called *inference*[1], which can be mathematically written as:

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})\, \mathrm{d}\boldsymbol{w}. \tag{2.2}$$

Going back to the posterior probability distribution formulation (equation 2.1), there is a term in the denominator, $p(\mathbf{Y}|\mathbf{X})$ - In the literature, this term is coined as *marginal likelihood* distribution or *model evidence* [4] and can be computed by evaluating the integral:

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \boldsymbol{w})p(\boldsymbol{w})\, \mathrm{d}\boldsymbol{w}. \tag{2.3}$$

In simple cases, where likelihood distribution is *conjugate* to the prior e.g. Gaussian likelihood and Gaussian prior or Categorical likelihood and Dirichlet prior etc. it is possible to evaluate the above integral analytically, since, under such settings, the posterior falls under the same family of probability distributions as the prior. But, in most practical situations, we do not have such ease at hand and thus, computing the integral in equation 2.3 by considering every possible set of the parameters, renders it as an highly intractable task.

In order to overcome this difficulty of intractability, we use approximate posterior inference techniques, for determining $p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})$ and consequently the predictions on new data points - $\mathbf{x}^*$.

2.2 Bayesian neural networks

Before we look into the approximate inference techniques, let us first talk about Bayesian Neural Networks (Bayesian NNs), which are derived in the probability theory framework described in section 2.1. Bayesian NNs combine the strengths of neural networks and probabilistic modelling techniques to provide a tool, which is capable of computing uncertainty estimates and gives the additional benefits of learning from small datasets and robustness to overfitting [4]. It places a prior distribution on the weights of a network, which may also be viewed as an implicit prior distribution over functions since every set of weights drawn from its prior distribution correspond to a different representation of the function [16]. Commonly, we choose to impose standard Gaussian prior distributions over each element of the set of weights ($\boldsymbol{w}$) and consider point estimates for the values of the bias ($\boldsymbol{b}$) in order to keep the problem simple [4].

Tracing back into the history of Bayesian NNs, a lot of work has been done in this direction, with the earliest ones being the contributions of MacKay [17] and Neal [18]. It is difficult to

[1] In deep learning literature, the term "inference" is referred to as the test time evaluation of the trained model whereas in bayesian framework, it corresponds to the evaluation of integral over the model parameters. But, calculating this integral is often computationally intractable so we need some kind of approximation, which basically means that the process of inference can involve some kind of "optimization" at training time [4].

perform inference in these networks hence, the main common goal of every work was to devise the methods which could make this problem easy to solve.

As discussed in section 2.1, we need to approximate posterior probability distribution over the set of parameters - $p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})$, for performing inference in Bayesian neural networks. The next section presents an overview of some techniques which have been proposed in the literature for achieving this task.

2.3 Approximate inference techniques

Evaluating posterior probability distribution $p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})$ in Bayesian NNs is a computationally intractable task due to the lack of available closed form solutions. Thus, we use approximate inference methods, for *approximating* this distribution, that can be divided into two major categories: 1) *Sampling* based approaches and 2) different variants of the *Variational inference* method. Most of the modern research falls into the second category hence, we would dig deeper into the elements of this category with only providing a comprehensive overview of the first category.

2.3.1 Sampling based approaches

Among the first contributions, in this direction, is the work of Neal [18] where he developed posterior probability distribution approximations based on Monte Carlo (MC) techniques. In particular, the use of Hamiltonian Monte Carlo[2] (HMC) [19] algorithm was proposed, which does not require any prior assumption on the form of the posterior distribution, for generating samples $\hat{\boldsymbol{w}}$ which would have been difficult to sample otherwise due to the hidden nature of the target (posterior) distribution [4]. HMC makes use of an integrator which takes multiple (user defined) *leapfrog* step sizes, while moving in the space of states. In practice, the performance of the HMC algorithm is highly variable to the selection of these step sizes and does not scale well to large data [4]. For mitigating these issues, the Langevin method was introduced, which requires the use of a single leapfrog stepsize and can also be used with large data. In the literature, this method has been formally named as Stochastic Gradient Langevin Dynamics (SGLD) [20]. Although, SGLD helps us in avoiding the issues observed with HMC, its practicality is still limited due to some reasons. SGLD often collapses to a single mode of the target distribution, and generates correlated samples $\hat{\boldsymbol{w}}$ around it, which puts a limit on the exploration of the state space and further requires generating more samples in order to get around with the problem of correlation (among samples)[4].

As another improvement to the Hamiltonian monte carlo method, the No-U-Turn Sampler (NUTS) [21] strategy was suggested, which sets the number of leapfrog step sizes by itself and also helps in dealing with the correlation problems. But still, it did not find much use in the literature. In Bayesian machine learning community, most of the research on approximate

[2]Hamiltonian Monte Carlo (HMC) is a Markov Chain Monte Carlo (MCMC) method which relies on the Hamiltonian dynamics for moving between states in the state space, in the process of approximating a target distribution which is the posterior probability distribution in the case of Bayesian neural networks.

inference techniques is centered around Variational inference method, which we discuss next.

2.3.2 Variational inference

In order to approximate the posterior probability distribution $p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})$, we choose an *variational* distribution $q_\theta(\boldsymbol{w})$, parametrized by θ, from a convenient family of distributions whose structure is easy to evaluate, and can express the posterior distribution well [22, 23]. For matching this variational distribution to the posterior in a best possible way, we minimize the Kullback-Liebler (KL) divergence [24] between the two distributions, which can be mathematically written as:

$$\mathrm{KL}(q_\theta(\boldsymbol{w}) \parallel p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})) = \int q_\theta(\boldsymbol{w}) \log \frac{q_\theta(\boldsymbol{w})}{p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})} \, \mathrm{d}\boldsymbol{w}. \tag{2.4}$$

Substituting equation 2.1 and applying the properties of logarithms, we can thus rewrite the preceding equation as:

$$\mathrm{KL}(q_\theta(\boldsymbol{w}) \parallel p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})) = \int q_\theta(\boldsymbol{w}) \left[\log q_\theta(\boldsymbol{w}) + \log \frac{p(\mathbf{Y}|\mathbf{X})}{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{w}) p(\boldsymbol{w})} \right] \mathrm{d}\boldsymbol{w}. \tag{2.5}$$

which can further be reduced to:

$$\int q_\theta(\boldsymbol{w}) \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{w}) \, \mathrm{d}\boldsymbol{w} - \mathrm{KL}(q_\theta(\boldsymbol{w}) \| p(\mathbf{w})) = \mathbb{E}_{q_\theta(\boldsymbol{w})} \left[\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{w}) \right] - \mathrm{KL}(q_\theta(\boldsymbol{w}) \| p(\boldsymbol{w})). \tag{2.6}$$

In the literature, the above quantity is identified as *Evidence Lower Bound* (ELBO) [4] and it is upper bounded by the logarithm of the model evidence - $\log p(\mathbf{Y}|\mathbf{X})$, which gives us the final mathematical representation of ELBO:

$$\underbrace{\mathbb{E}_{q_\theta(\boldsymbol{w})} \left[\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{w}) \right]}_{Expected\ log\ likelihood} - \underbrace{\mathrm{KL}(q_\theta(\boldsymbol{w}) \| p(\boldsymbol{w}))}_{Prior\ KL} \leq \log p(\mathbf{Y}|\mathbf{X}). \tag{2.7}$$

Thus, minimizing the KL diveregence between the posterior and the variational distribution results in an optimization objective where we maximize the quantity ELBO with respect to the parameters θ of the variational distribution. Our main goal is to maximize the expected log likehood of the data, under a given probabilistic model, and minimize the prior KL which acts as a *regularization* term and forces the variational distribution to stay closer to the prior.

The above process of choosing a variational distribution $q_\theta(\boldsymbol{w})$ and subsequently maximizing ELBO, with respect to its parameters θ, is popularly known as *Variational Inference* (VI) [25]. VI provides us with a practical framework for performing inference in Bayesian NNs. Instead of evaluating the intractable integral in equation 2.3, we solve an optimization problem, which is easy due to the large availability of strong convex/non-convex optimization problem solving tools in mathematics. However, this approach could still create problems, if we have large datasets at hand, since the expected log likelihood term in equation 2.7 requires solving the integral over the whole dataset [4]. Apart from that, it is difficult to compute the expected value of the log likelihood if we have a complex model. In the literature, a couple of remedies have been proposed for dealing with these problems as well as some improvements to the existing VI method, which falls under the category of *modern scalable variational inference*. We present an overview of the elements in this category next.

2.3.2.1 Modern scalable variational inference

Variational inference gives us a practical tool for performing inference in Bayesian neural networks. But, evaluating the integral $\int q_\theta(\boldsymbol{w}) \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{w})\, \mathrm{d}\boldsymbol{w}$ poses some serious difficulties if:

1. we have to deal with large datasets $\{\mathbf{X}, \mathbf{Y}\}$
2. and work with complex networks, which makes the computation of the integral intractable and demand a smart representation of the variational distribution $q_\theta(\boldsymbol{w})$.

In the past, starting from 1993, a couple of solutions were proposed for these problems. Hinton *et al.* [26] attempted to define a Gaussian variational distribution, which factorized over each value of the weight in the set $\boldsymbol{w}$ [4]:

$$q_\theta(\boldsymbol{w}) = \prod_{i=1}^{L} q_\theta(\mathbf{W}_i) = \prod_{i=1}^{L} \prod_{j=1}^{K_i} \prod_{k=1}^{K_{i+1}} q_{m_{ijk}, \sigma_{ijk}}(w_{ijk}) = \prod_{i,j,k} \mathcal{N}\left(w_{ijk}; m_{ijk}, \sigma_{ijk}^2\right). \tag{2.8}$$

However, the use of the above variational distribution in the VI optimization objective was only shown for a single hidden layer Bayesian NN, since the expected log likelihood term is difficult[3] to evaluate for complex networks. In addition, this variational distribution assumes independence over each value of the weight, which can lead to bad performance since the weight correlations might encode some useful information. In order to study the importance of these correlations, Barber *et al.* [27] computed covariance matrices and showed an improvement over the previous method. However, it comes with an increased computational complexity which makes the use of this method impractical for most of the networks.

In order to provide a solution[4] to both of the problems concerning large datasets and the intractability of the expected log likelihood term, Graves [28] presented the *data sub-sampling* technique which provides a promising solution to the first problem. Denoting our optimization objective in equation 2.6 as $\mathscr{L}_{\mathrm{VI}}(\theta)$, we can thus rewrite it as:

$$\mathscr{L}_{\mathrm{VI}}(\theta) := \sum_{i=1}^{N} \int q_\theta(\boldsymbol{w}) \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{w})\, \mathrm{d}\boldsymbol{w} \; - \; \mathrm{KL}(q_\theta(\boldsymbol{w}) \| p(\mathbf{w})). \tag{2.9}$$

Applying the data sub-sampling approach, the above objective can be modified to:

$$\hat{\mathscr{L}}_{\mathrm{VI}}(\theta) := \frac{N}{B} \sum_{i \in S} \int q_\theta(\boldsymbol{w}) \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{w})\, \mathrm{d}\boldsymbol{w} \; - \; \mathrm{KL}(q_\theta(\boldsymbol{w}) \| p(\mathbf{w})), \tag{2.10}$$

where $S \subseteq \{\mathbf{X}, \mathbf{Y}\}$ and has a cardinality of B. As shown by Hoffman *et al.* [29], we can solve the above optimization objective to get the optimum values θ^* for the parameters of the variational distribution, since it offers an unbiased approximation to the original VI optimization objective (equation 2.9). The data sub-sampling technique helps in simplifying the computations with large datasets however, the problem related to the intractability of the expected log likelihood

[3] The computation of the expected log likelihood requires marginalization with respect to the variational distribution $q_\theta(\boldsymbol{w})$, which is intractable in case of complex networks.

[4] The solutions presented by Graves [28] as well as Hinton *et al.* [26] were derived in the context of information theory but they result in an optimization objective, which is similar to VI's ELBO.

integral still remains. For alleviating this issue, Graves [28] used Monte Carlo (MC) estimation. In addition, he suggested the use of delta as well as fully factorized Gaussian distribution (similar to Hinton *et al.* [26]) as $q_\theta(\boldsymbol{w})$. Besides offering a practical solution to the underlying problems in VI, Grave's work [28] did not perform very well, which might probably be due to the ignorance of weight correlations.

Perhaps, one of the biggest contributions, in the context of modern scalable variational inference, has been the work of Gal [4]. He took inspiration from the work of Graves [28] and Hinton *et al.* [26] for developing a practical inference technique, from the bayesian perspective. He suggested the use of data sub-sampling approach for dealing with large datasets however, for MC approximation of the expected log likelihood integral, he presented an overview of the different stochastic estimators, and analyzed their variance in the process of selecting the best one for his technique.

For solving the VI optimization objective, we are interested in estimating the derivatives of the expected log likelihood term with respect to the variational distribution parameters θ. We can write a general mathmetical equation for it as:

$$I(\theta) = \frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x, \tag{2.11}$$

where $f(x)$ is a real differentiable function and $p_\theta(x)$ is a probability density function (pdf) parametrized by θ. Considering the above equation, Gal [4] gives an overview and variance analysis of particularly three MC estimators, which are used in the VI literature.

Firstly, he tells about the *score function* estimator [30] which relies on the identity $\frac{\partial}{\partial\theta}p_\theta(x) = p_\theta(x)\frac{\partial}{\partial\theta}\log p_\theta(x)$ and parametrizes the equation 2.11 as:

$$\begin{aligned}\frac{\partial}{\partial\theta}\int f(x)p_\theta(x)\mathrm{d}x &= \int f(x)\frac{\partial}{\partial\theta}p_\theta(x)\mathrm{d}x \\ &= \int f(x)\frac{\partial\log p_\theta(x)}{\partial\theta}p_\theta(x)\mathrm{d}x.\end{aligned} \tag{2.12}$$

This estimator results in an unbiased solution - $\mathbb{E}_{p_\theta(x)}\left[\hat{I}_1(\theta)\right] = I(\theta)$, but with with high variance.

Secondly, he presents the mathematical details of the *pathwise derivative* estimator, which is alternatively known as the *re-parametrization trick* [31]. He shows that, by re-parametrizing $p_\theta(x)$ as $p(\epsilon)$ such that $x = g(\theta, \epsilon)$, with $g(\cdot,\cdot)$ being a differentiable bivariate transformation, the estimator which results is:

$$\hat{I}_2(\theta) = f'(g(\theta,\epsilon))\frac{\partial}{\partial\theta}g(\theta,\epsilon), \tag{2.13}$$

where $f'(\cdot)$ denotes the first derivative of the fuction $f(\cdot)$. This estimator also forms an unbiased solution - $\mathbb{E}_{p(\epsilon)}\left[\hat{I}_2(\theta)\right] = I(\theta)$ and (in most of the cases) gives less variance as compared to the score function estimator.

Lastly, he gives an overview of the *characteristic function* estimator, which was introduced by Opper *et al.* [32]. The mathematical formulation of this estimator is done by restricting the pdf $p_\theta(x)$ to Gaussian $\mathcal{N}(\mu,\sigma)$. This results in two derivative estimators, where the μ derivative estimator can be mathematically written as:

$$\frac{\partial}{\partial\mu}\int f(x)p_\theta(x)\mathrm{d}x = \int f'(x)p_\theta(x)\mathrm{d}x, \tag{2.14}$$

with $\hat{I}_3(\mu) = f'(x)$ and $\mathbb{E}_{p_\theta(x)}\big[\hat{I}_3(\mu)\big] = I(\mu)$, and the σ derivative estimator as:

$$\frac{\partial}{\partial\sigma}\int f(x)p_\theta(x)\mathrm{d}x = 2\sigma\cdot\frac{1}{2}\int f''(x)p_\theta(x)\mathrm{d}x, \tag{2.15}$$

with $\hat{I}_3(\sigma) = \sigma f''(x)$ and $\mathbb{E}_{p_\theta(x)}\big[\hat{I}_3(\sigma)\big] = I(\sigma)$. Here, $f''(\cdot)$ denotes the second derivative of the function $f(\cdot)$. The characteristic function estimator results in even lower variance than the pathwise derivative estimator however, its use is only limited to the cases where we use a Gaussian distribution.

Making use of the findings, related to the stochastic estimators, presented above and addressing the solution to the problems associated with other approaches, Gal [4] makes use of the pathwise derivative estimator instead of the chracteristic function estimator, since it allows us to explore a wide variety of variational distribution families rather than just restricting it to Gaussian, and preserves weight correlations by factorizing over each weight row $\mathbf{w}_{l,i}$ in the weight matrix $\mathbf{W}_l$ instead of factorizing over every single *scalar* weight value in all the rows, as done in previous works [26, 28]. Here, $\mathbf{w}_{l,i}$ denotes the i^{th} row of the weight matrix $\mathbf{W}_l$, which belongs to the layer l of a Bayesian NN. In line with the principles of the pathwise derivative estimator, Gal [4] re-parametrizes the variational distribution $q_{\theta_{l,i}}\big(\mathbf{w}_{l,i}\big)$ as $\mathbf{w}_{l,i} = g\big(\theta_{l,i}, \epsilon_{l,i}\big)$ and defines $p\big(\epsilon_{l,i}\big)$ according to his method (described in the next paragraph). For keeping the mathematical notation consistent with the previous optimization objective formulations, we would denote $\boldsymbol{w} = g(\theta, \epsilon)$ and $p(\epsilon) = \prod_{l,i} p\big(\epsilon_{l,i}\big)$. Now, we can rewrite the data sub-sampling approach inspired optimization objective in equation 2.10 as:

$$\begin{aligned}\hat{\mathscr{L}}_{\text{VI}}(\theta) &= \frac{N}{B}\sum_{i\in S}\int q_\theta(\boldsymbol{w})\log p(\mathbf{y}_i|\boldsymbol{f}^{\boldsymbol{w}}(\mathbf{x}_i))\mathrm{d}\boldsymbol{w} - \mathrm{KL}(q_\theta(\boldsymbol{w})\|p(\boldsymbol{w})) \\ &= \frac{N}{B}\sum_{i\in S}\int p(\boldsymbol{\epsilon})\log p\big(\mathbf{y}_i|\boldsymbol{f}^{g(\theta,\epsilon)}(\mathbf{x}_i)\big)\mathrm{d}\boldsymbol{\epsilon} - \mathrm{KL}(q_\theta(\boldsymbol{w})\|p(\boldsymbol{w}))\end{aligned} \tag{2.16}$$

and apply pathwise derivative estimator for resolving the log likelihood integral and getting a new optimization objective:

$$\hat{\mathscr{L}}_{\text{MC}}(\theta) = \frac{N}{B}\sum_{i\in S}\log p\big(\mathbf{y}_i|\boldsymbol{f}^{g(\theta,\epsilon)}(\mathbf{x}_i)\big) - \mathrm{KL}(q_\theta(\boldsymbol{w})\|p(\boldsymbol{w})), \tag{2.17}$$

which follows $\mathbb{E}_{S,\epsilon}\Big(\hat{\mathscr{L}}_{\text{MC}}(\theta)\Big) = \mathscr{L}_{\text{VI}}(\theta)$. In the above equation, the expected log likelihood integral is replaced by its stochastic counterpart - MC integration.

For defining the parameter-free distribution $p(\boldsymbol{\epsilon})$, Gal *et al.* [4, 33, 34] make a connection between the standard *regularization*[5] techniques and the approximate variational inference with distribution $q_\theta(\boldsymbol{w})$. Particularly, they do this with *dropout* [35], which is, by far, the most popular regularization technique used in deep learning. In dropout, we add stochasticity in the feature space and subsequently set some of the features in the intermediate layers to zero. Denoting the activation feature vector, for a single hidden layer neural network, as $\boldsymbol{h}$, we sample a binary vector $\hat{\boldsymbol{\epsilon}}$, whose elements take zero value with probability p, where $0 \leq p \leq 1$, and demonstrate the effect of dropout as: $\hat{\boldsymbol{h}} = \boldsymbol{h}\odot\hat{\boldsymbol{\epsilon}}$. Here, $\hat{\boldsymbol{h}}$ denotes the modified activation feature vector, which we get after applying dropout.

[5]In deep learning, regularization techniques are seen as an essential tool for improving the performance and generalization capability of the neural networks, which are commonly used in different application areas.

In standard deep learning, dropout is considered as adding stochasticity in the feature space. However, if it could be mathematically interpreted as adding stochasticity over the space of network parameters, then we can possibly make its connection with the approximate variational inference in Bayesian NNs. Gal *et al.* [4, 33, 34] study this transformation[6] of dropout from the feature space to the parameter space and show that if we train a standard neural network with dropout, it is identical to performing approximate VI in it and, thus, we can obtain uncertainty estimates from such networks.

Considering a single hidden layer neural network, as shown by Gal *et al.* [4, 33, 34], we can write the dropout optimization objective, in case of regression [36] as:

$$\hat{\mathscr{L}}_{\text{dropout}}\left(\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\right) = -\frac{1}{M\tau}\sum_{i\in S}\log p\Big(\mathbf{y}_i|\boldsymbol{f}^{g(\theta,\hat{\epsilon}_i)}(\mathbf{x}_i)\Big) + \lambda_1\|\mathbf{M}_1\|^2 + \lambda_2\|\mathbf{M}_2\|^2 + \lambda_3\|\mathbf{b}\|^2. \tag{2.18}$$

In the above equation, $\mathbf{M}_1$ and $\mathbf{M}_2$ are deterministic weight matrices, which are used to define the differentiable bivariate transformation $g(\theta, \hat{\epsilon}_i)$. Injecting the dropout noise to these matrices, we obtain their stochastic version, which we write as $\widehat{\mathbf{W}}_1$ and $\widehat{\mathbf{W}}_2$. Here, $\widehat{\mathbf{W}}$ denotes a realization of $\mathbf{W}$, which is a stochastic random variable defined over the set of real weight matrices. Now, we can collect these stochastic weight matrices in a set and relate it with the bivariate transformation as:

$$\hat{\boldsymbol{w}}_i = \Big\{\widehat{\mathbf{W}}_1^i, \widehat{\mathbf{W}}_2^i, \mathbf{b}\Big\} = \big\{\text{diag}\big(\hat{\boldsymbol{\epsilon}}_1^i\big)\mathbf{M}_1, \text{diag}\big(\hat{\boldsymbol{\epsilon}}_2^i\big)\mathbf{M}_2, \mathbf{b}\big\} =: g(\theta, \hat{\boldsymbol{\epsilon}}_i), \tag{2.19}$$

where $\theta = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\}$, $\hat{\boldsymbol{\epsilon}}_1^i \sim p(\boldsymbol{\epsilon}_1)$, and $\hat{\boldsymbol{\epsilon}}_2^i \sim p(\boldsymbol{\epsilon}_2)$. Here i denotes the total number of training data points i.e. $1 \leq i \leq N$ and $p(\epsilon_l)$ $(l = 1, 2)$ represents a product of *Bernoulli* distributions with probabilities $1 - p_l$. In addition, as given in [4], τ denotes the constant observation noise, in case of regression, where we consider the likelihood distribution as Gaussian. Further, λ is a regularization hyperparameter and is used for controlling the strength of the regularization. Rewriting the optimization objective in equation 2.17 as:

$$\hat{\mathscr{L}}_{\text{MC}}(\theta) = -\frac{N}{B}\sum_{i\in S}\log p\Big(\mathbf{y}_i|\boldsymbol{f}^{g(\theta,\epsilon)}(\mathbf{x}_i)\Big) + \text{KL}(q_\theta(\boldsymbol{w})\|p(\boldsymbol{w})), \tag{2.20}$$

Gal *et al.* [4, 33, 34] make its comparison with the dropout optimization objective in equation 2.18 and show that, if we choose a $q_\theta(\boldsymbol{w})$ with $\boldsymbol{w} = \{\text{diag}(\boldsymbol{\epsilon}_1)\mathbf{M}_1, \text{diag}(\boldsymbol{\epsilon}_2)\mathbf{M}_2, \mathbf{b}\}$ as Bayesian NN's variational distribution, the two optimization objectives are identical and a standard neural network trained with dropout can be seen as performing approximate VI in a probablistic interpretation of the same network. Consequently, uncertainty information can be extracted from the networks trained with dropout layers without even changing their architecture. They refer to this variational distribution $q_\theta(\boldsymbol{w})$ as *dropout variational distribution*. A detailed mathematical analysis of these findings can be found in [4, 33, 34].

So far, we have put emphasis on approximate VI in Bayesian NNs by considering uncertainty in the parameter space. However, there is an alternative direction of research, which focuses on performing VI by considering uncertainty in the *function* space. Instead of imposing a prior distribution on the space of parameters - $p(\boldsymbol{w})$, we do it on the space of functions and denote it as $p(\boldsymbol{f})$. The probability distribution $p(\boldsymbol{f})$ represents a joint distribution [37] over a set of N

[6] For the mathematical details of this transformation, the reader is recommended to look at [4].

random variables, each holding the function's output for a single data point, and can be mathematically represented as $p(\boldsymbol{f}^{\mathbf{X}})$, where $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ and $\boldsymbol{f}^{\mathbf{X}} = (f(x_1), f(x_2), \dots, f(x_n))$. For defining an optimization objective, we minimize the KL divergence between the variational $q(\boldsymbol{f})$ and the posterior distribution over the function space, for a given dataset $\{\mathbf{X}, \mathbf{Y}\}$. In the literature, various methods have been presented that consider this function space view of Bayesian NNs. A detailed overview of these methods can be found in [38–40].

In this thesis, we would consider performing approximate VI over the parameter space. In the next chapter, we would present an in-depth analysis of a method which exploits the connection between dropout training and approximate VI for computing uncertainty estimates. We would discuss the reasons for the selection of this particular method in one of the later sections. But before we look into that, let us talk about different types of uncertainties which exist in deep learning.

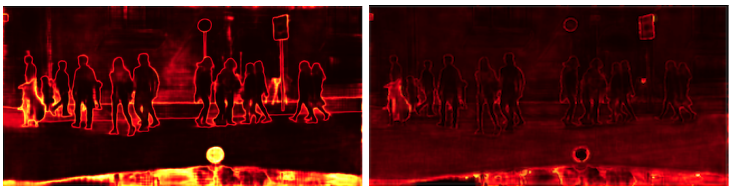
2.4 Uncertainties in deep learning

Uncertainty quantification is an important task especially in safety critical applications, such as Autonomous driving, Facial recognition systems and so forth, which deploy Deep Neural Networks (DNNs) for achieving the desired result. Considering the deep learning based vision tasks, there are two major types of uncertainties which one may encounter: *epistemic* and *aleatoric* [41]. Aleatoric uncertainty measures noise inherent in the data which could be associated to the noise in the sensor (e.g. camera for images), used for capturing that data [1]. It could be further classified into two different categories: *homoscedastic* and *heteroscedastic* aleatoric uncertainty [1]. In terms of regression settings (e.g. depth regression [7]), homoscedastic aleatoric uncertainty is a constant scalar value for every sample in the dataset whereas its heteroscedastic counterpart is a vector of length equal to the size of the dataset, such that each sample gets a different value for the uncertainty. On the other hand, epistemic uncertainty captures our ignorance of the right choice of the DNN which could be used for a particular application. Combining aleatoric and epistemic uncertainty, we get *predictive* uncertainty [4] which basically represents the overall confidence we have in the predictions from our network.

Epistemic uncertainty becomes significant when the network has to make predictions on the input data points that highly differ from the training data. We refer to such kind of samples as: *Out-of-Distribution (OOD)* input data [38] i.e. it does not belong to the data distribution on which our network was trained. In case of applications where we have sufficient amount of training data available, epistemic uncertainty gets remarkably reduced [1] however, it is still safe to keep a record of the measure of this uncertainty. In contrast, aleatoric uncertainty becomes notable, when we have an input example which belongs to the high noise region or (in case of classification settings) lies at the boundary of two different class clusters in the input data space. Aleatoric uncertainty can not be reduced by obtaining more training data thus, it is effective to model this uncertainty in safety critical applications.

Considering the addition of stochasticity in the space of parameters, for obtaining an estimate of epistemic uncertainty, we resort to bayesian modelling, where we impose a prior probability distribution on the network parameters (see section 2.1-2.2) and observe the change in this distribution as we keep on processing more data. On the other hand, for measuring aleatoric uncertainty, we place a distribution over the outputs of the network [1]. For getting a clear

(a) Image (b) Ground Truth (c) Prediction



(d) Epistemic Uncertainty (e) Aleatoric Uncertainty

Figure 2.1: Visual representation of aleatoric and epistemic uncertainty for semantic segmentation on Cityscapes dataset [43]. In (b), black pixels are the ones which are ignored while training the network thus, we can consider these pixels to be OOD while making predictions (b). In (d), we can see a high epistemic uncertainty for these pixels as well as for some other pixels which are semantically challenging for the network. Epistemic uncertainty tends to be high for the misclassified pixels as well. In (e), we have a visual representation of the aleatoric uncertainty, which tends to be higher for objects far away in the image. We will comment more on the visual interpretation of the aleatoric and epistemic uncertainty in the later chapters.

understanding of this point, consider the case of regression, where we typically model aleatoric uncertainty by imposing a Gaussian distribution on the outputs:

$$p(\mathbf{y}|\boldsymbol{f}^{w}(\mathbf{x})) = \mathcal{N}\left(\mathbf{y}; \boldsymbol{f}^{\boldsymbol{w}}(\mathbf{x}), \tau^{-1}I\right), \tag{2.21}$$

where $\tau = \frac{1}{\sigma}$ quantifies the observation noise. Here σ denotes the standard deviation of the distribution. This is an example of measuring homoscedastic aleatoric uncertainty. In practical situations, heteroscedastic aleatoric uncertainty makes more sense so we would discuss methods, which are focused on measuring it.

Uncertainty estimation can prove to be useful in a lot of applications. In this thesis, we would focus on this task in the context of Computer vision classification problems particulary, *Image classification* [5] and *Semantic segmentation* [6], that are an essential part of Autonomous driving [42]. There are some state-of-the-art methods which can be used for quantifying uncertainties in these applications. But, before we look into these methods, we would shed some light on the way in which we represent uncertainties in these computer vision tasks, in the next two subsections.

2.4.1 Uncertainty representation in Image classification

In Image classification, we are interested in finding a mapping between the input images and their corresponding labels. For an input example, we compute a single value for the aleatoric

and epistemic uncertainty, which can be combined to get a value for the predictive uncertainty. Consider a class with 500 samples from an arbitrary dataset; we compute uncertainty estimates for every sample in this class but instead of representing values for every single sample separately, we rather compute an average (*mean*) and represent a single value for the different uncertainty estimates for the whole class. For getting an idea of the variability in the values of individual samples, we can compute *standard deviation*. In this way, we can study the overall behaviour of the network to the samples of different classes.

2.4.2 Uncertainty representation in Semantic segmentation

Semantic segmentation is an extended version of image classification where, instead of assigning a single label to the whole image, we perform pixel-level classification and learn a mapping which is powerful enough to assign a single label to every pixel in an image. Similarly, for uncertainty representation, instead of computing a single value of different uncertainties, we compute an uncertainty map where we get a value for every pixel in an image. Figure 2.1 gives an illustration of how these uncertainty maps look like for a sample image from the Cityscapes [43] dataset. We can derive different intuitions from these visual maps about the pixels which are semantically challenging or noisy for the network.

Now, we have an idea of how we represent uncertainties in different computer vision applications. In the next section, we would look at a couple of state-of-the-art approaches which could be used for quantifying these uncertainty estimates.

2.5 Methods for computing uncertainty estimates

In Image classification and Semantic segmentation, *conditional random fields* [44] method was largely used for modelling uncertainties. However, with the increasing use of deep learning in these vision tasks, a couple of approaches have been introduced, which makes it possible to quantify uncertainties from the neural networks employed in these applications.

Considering the bayesian treatment of parameters, as shown in section 2.1, we are interested in finding the posterior distribution - $p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})$, which induces the predictive distribution (equation 2.2), while performing inference on a new input sample $\mathbf{x}^*$. It is intractable to compute the posterior distribution thus, we perform variational inference and compute optimal parameters θ^* of an approximating posterior distribution $q_\theta(\boldsymbol{w})$. Subsequently, for making predictions, we use $q^*_\theta(\boldsymbol{w})$ which results in a predictive distribution [4]:

$$q^*_\theta(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{w}) q^*_\theta(\boldsymbol{w}) \mathrm{d}\boldsymbol{w}. \tag{2.22}$$

In order to approximate the above integral, Monte Carlo (MC) techniques are commonly used which gives us the result [4]:

$$q^*_\theta(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) := \frac{1}{T} \sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \hat{\boldsymbol{w}}_t) \underset{T \to \infty}{\longrightarrow} p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}), \tag{2.23}$$

with $\hat{\boldsymbol{w}}_t \sim q^*_\theta(\boldsymbol{w})$. Here T denotes the number of MC samples. Consequently, this approximating predictive distribution $q^*_\theta(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})$ can be used to compute predictive uncertainty.

Following this approach, Korattikara *et al.* [45] compute predictive uncertainty by using a teacher-student network configuration, where they train a student to reproduce the behaviour of a teacher network. They compute the predictive distribution by MC integration however, for computing the posterior probability distribution over the space of parameters $\boldsymbol{w}$, they combine online MCMC methods with model distillation, instead of performing approximate VI. This method requires two separate networks which limits its practicality for complex tasks, where we need to use extremely deep models. Bouchacourt *et al.* [46] use a different approach and introduce DISCO networks that are trained by minimizing the dissimilarity coefficient between the network model distribution and the true data distribution. In addition to these methods, Lakshminarayanan *et al.* [47] train an ensemble of M networks for obtaining predictive uncertainty in non-bayesian settings. They show that their method works equivalently well (or better) in comparison to obtaining predictive uncertainty estimate with Bayesian NNs. Although simple, their approach requires training multiple networks which does not seem to be a viable solution for computer vision tasks that demand the use of DNNs. Gal *et al.* [4, 33, 34] make a connection between the regularization technique dropout and Bayesian NNs for computing predictive uncertainty. They show that the standard networks trained with dropout perform implicit approximate VI and, thus, can be used to extract uncertainty information. Placing dropout layers in a network is equivalent to imposing the dropout variational distribution on the weights of the probabilistic version of the similar network. At test time, they use dropout with MC integration for computing predictive uncertainty and they name this method as Monte Carlo dropout (MC dropout). Here, MC integration is done by performing multiple stochastic forward passes through the network, while keeping the dropout on, and averaging the results. Performing a stochastic forward pass through the network is equivalent to extracting a sample from the variational posterior distribution. Since we need to perform multiple forward passes, this method increases the computational burden at test time. A solution to this sampling based approach is put forward by Postels *et al.* [48]. They rely on approximated variance propagation for estimating predictive uncertainty and prove that their technique performs faster than the MC dropout. Although fast, their method makes independence assumptions while propagating variances and consequently does not show much quantitative improvements. At this point, it is worth mentioning that an alternative line of research exists which focuses on the addition of stochasticity in the function space. As an example, we would state the work of Carvalho *et al.* [37]. They leverage the functional VI framework from [39] to compute scalable predictive uncertainty estimates. They define a Gaussian Process (GP) prior, based on Bayesian Convolutional Neural Networks (Bayesian CNNs) priors with a closed-form covariance kernel, and formulate an optimization objective (ELBO) for computing the approximate posterior distribution over the space of functions. Later, while making predictions, they use this distribution to compute the predictive distribution and hence, the predictive uncertainty. Although scalable, their method does not work well if we have *pooling* layers in the network. Further, it is highly sensitive to the choice of the non-linearity (activation function) used in a Bayesian NN.

So far, we have discussed methods that aim for computing predictive uncertainty. This type of uncertainty combines both aleatoric and epistemic uncertainty into one value. For adressing the computation of epistemic and (heteroscedastic) aleatoric uncertainty separately, Kendall *et al.* [1] combine MC dropout with a noise model. They claim that the epistemic uncertainty induces

predictive uncertainty and thus interpret this predictive uncertainty as epistemic. However, the interesting thing is that they use a single network for computing these different uncertainties. In addition, their method does not require making any network architectural changes except the fact that we only need to add the dropout layers in our network. For obtaining epistemic uncertainty, they apply MC dropout whereas for aleatoric uncertainty, they place a distribution over the output logits[7] of the network. The downside of this method is that, during training, it requires sampling from the logit space distribution which increases the computational time and requires more memory on the hardware. Further, MC dropout increases the computational burden at test-time. Apart from this method, Gast *et al.* [49] introduce an uncertainty propagating architecture for computing aleatoric uncertainty (only). They add white Gaussian noise to the input, which is equivalent to imposing a Gaussian distribution, and propagate it through the network using Expectation Propagation [50] based Assumed Density Filtering method [51], in a single forward pass. They name this approach as Lightweight Probabilistic Deep Networks. But, this method has some limitations. Firstly, it assumes independence between different neurons activation distributions in an intermediate layer, which makes it difficult to propagate the distributions through the non-linear layers, such as *pooling* since it introduces correlation between activations. Secondly, this method increases the number of network parameters since, at every layer, we now need to compute the mean as well as variance of the propagating Gaussian distributions (for every activation).

As discussed in this section, a couple of approaches can be used for computing uncertainty estimates in classification based vision tasks. Every method has its own pros and cons and one needs to study them well, before choosing a particular method for a certain application. In the next subsection, we would highlight the method which we use in this work for computing uncertainty estimates. Further, we give arguments supporting our decision of the selection of this particular method.

2.5.1 Selected approach for uncertainty estimation

For the task of computing uncertainties in Image classification and Semantic segmentation, a couple of approaches exist. Out of all the methods explained in the previous section, in this thesis, we use the work of Kendall *et al.* [1] for computing uncertainty estimates. This is because:

1. it considers uncertainty estimation by the addition of stochasticity in the space of parameters which aligns with the formulation of Bayesian NNs, is easier to comprehend, and powerful enough at the same time.

2. the method introduced in this work provides a unified framework for computing both aleatoric and epistemic uncertainty as compared to the other approaches,

3. it is a baseline method that everyone compares with, while introducing their work on uncertainty estimation,

[7] Here, logits denote the activation values in the layer just before the final *softmax* layer, which computes the class confidence scores, in a neural network.

4. the method does not require making any changes to the network but only demands the addition of the dropout layers at appropriate places in the network,

5. the performance of this method depends on a couple of variables but has been only been studied with varying only few,

6. and it has not been shown, whether this method computes uncertainty estimates that correlate well with the complexities in the data.

Clearly, reason 3 and 4 have a huge potential of providing some important research questions, which could be answered by performing an analysis on this method. These reasons are often used [37, 49] for commenting on the goodness of this method and, subsequently, introducing new approaches. So, we perform an in-depth analysis of the work of Kendall *et al.* [1] for answering some research questions which we derive first, and for performing some other uncertainty based tasks.

As dicussed in section 2.5, the method proposed by Kendall *et al.* [1] poses some difficulties. Firstly, during training, we need to marginalize over the logit space distribution. Since there does not exist any closed-form solution for it, we perform the sampling operation. We do it for learning the parameters of this logit space distribution which subsequently leads to computing predictions as well as aleatoric uncertainty (at test-time). As a matter of fact, this sampling occurs only at the final layer of the network which does not add much to the computational resources required. Secondly, during test-time, we need to perform multiple forward passes, while keeping the dropout layers, for getting an estimate of the epistemic uncertainty. Gal [4] empirically shows that, for computing an unbiased estimate of this uncertainty, performing a few forward passes ($\sim$ 50) is enough. Hence, with modern computing resources such as Graphical Processing Units (GPUs) at hand, we can easily parallelize this operation and perform it in a constant run-time.

In the next chapter, we will explain the details of the method of Kendall *et al.* [1] and the follow-up work which was done using this method.

3 The Power of Uncertainty Estimation Tools

For achieving state-of-the-art performance in computer vision tasks, particularly Image classification [5] and Semantic segmentation [6], deep learning is being actively used. These tasks play their role in accomplishing the overall objective of Autonomous driving which is a highly safety critical application. Thus, uncertainty estimation is important. There are two uncertainties named as epistemic and aleatoric which one can model. In this chapter, we would give details of a practical framework which can be used for quantifying these uncertainty estimates in vision tasks. Later, firstly, in the context of Semantic segmentation, we would give details of the practical variations in this framework that might have an effect on the quality of the estimates. In addition, we would present some metrics for quantitatively evaluating the computed uncertainties. Secondly, in the context of Image classification, we would introduce some possible input data space complexities, for making a clear distinction between the cases, that might have a significant impact on the respective uncertainty estimates and perform a correlation analysis. Following this study, we would propose a new classifier design which could be used to partition the predictions from a main network into different categories. Alongside, we will also provide the details of these categories. Afterwards, in the next chapter, we would validate our arguments by conducting different experiments.

3.1 Unified framework for estimating uncertainties

In the Bayesian machine learning literature, there exist a couple of approaches which could be used for computing both epistemic as well as aleatoric uncertainty. However, as already indicated in the subsection 2.5.1, we would use the method introduced by Kendall *et al.* [1] for this purpose.

In vision tasks, for extracting uncertainty information, we resort to the bayesian representation of the convolutional neural network. We place a prior distribution on the space of parameters $\boldsymbol{w}$, define a likelihood distribution $p(\mathbf{y}|\mathbf{x}, \boldsymbol{w})$, and apply bayesian probability theorem for computing the posterior distribution $p(\boldsymbol{w}|\mathbf{X}, \mathbf{Y})$. Due to the intractable computation of this distribution, we perform variational inference by choosing a simple approximating variational distribution $q_\theta(\boldsymbol{w})$ and optimizing its parameters θ. As discussed in sub-subsection 2.3.2.1, evaluating the expected log likelihood integral in VI's optimization objective is intractable thus, we perform approximate VI. Kendall *et al.* [1] make use of dropout variational distribution [4, 33, 34] for approximating inference in Bayesian NNs, which gives us the optimal parameters θ^* of this variational distribution. For obtaining the predictive distribution, the process of marginalization over the approximating posterior distribution is performed by extracting samples from this distribution. This process is known as MC integration. In classification settings,

we commonly pick *softmax likelihood* - thus, we can rewrite the equation 2.23 for obtaining the predictive distribution for a new input example $\mathbf{x}^*$ as:

$$p(\mathbf{y}^* = c|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) \approx \frac{1}{T}\sum_{t=1}^{T} \text{softmax}\Big(\boldsymbol{f}^{\hat{\boldsymbol{w}}_t}(\mathbf{x}^*)\Big), \tag{3.1}$$

with $\hat{\boldsymbol{w}}_t \sim q_\theta^*(\boldsymbol{w})$. Here, T denotes the number of samples extracted from the approximating posterior distribution and c refers to a class label in the dataset. In addition, $\boldsymbol{f}$ represents a Bayesian CNN. According to [1], in order to quantify the epistemic uncertainty (which induces predictive uncertainty), we compute the entropy[1] of this predictive distribution $p(\mathbf{y}^* = c|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})$. As given in sub-subsection 2.3.2.1, performing approximate VI with the dropout variational distribution is equivalent to training a standard neural network with dropout layers. Thus, finding the optimal set of parameters $\boldsymbol{w}^*$ in this setting is the same as obtaining optimal parameters θ^* of the approximating variational distribution [4]. Following this approach, for obtaining the epistemic uncertainty at test-time, we keep the dropout layers[2], perform multiple stochastic forward passes through the network, and compute the entropy of the resulting probability vector $\mathbf{p}$. This technique is popularly known as MC dropout [4, 33, 34]. We can obtain the mathematical notation of this operation by rewriting the equation 3.1 as:

$$\mathbf{p} = \frac{1}{T}\sum_{t=1}^{T} \text{softmax}\Big(\boldsymbol{f}^{\hat{\boldsymbol{w}}_t{}^*}(\mathbf{x}^*)\Big), \tag{3.2}$$

where T denotes the number of stochastic forward passes and $\boldsymbol{f}$ represents a Convolutional Neural Network (CNN). Here, $\hat{\boldsymbol{w}}_t{}^*$ is a realization of $\boldsymbol{w}^*$ at the t^{th} forward pass. This is equivalent to extracting samples from the approximating posterior distribution in the case of Bayesian CNNs. In addition, $\boldsymbol{f}^{\hat{\boldsymbol{w}}_t^*}(\cdot)$ gives a vector of unaries[3] the logit space for an input example, which is passed through the softmax operation for obtaining the probability vector $\mathbf{p}$. The entropy of this probability vector can be denoted as:

$$H(\mathbf{p}) = -\sum_{c=1}^{C} p_c \log p_c, \tag{3.3}$$

where C represents the total number of possible class labels in a dataset. In Semantic segmentation, we predict a class label for every pixel in an input image thus we compute the epistemic uncertainity for every pixel as well. However, in case of Image classification, we just get a single number for this uncertainty.

As discussed above, we need to learn the optimal parameters of a CNN during training, for computing the epistemic uncertainty at test-time. However, as shown by Kendall *et al.* [1], the same network can be used for computing the aleatoric uncertainty as well. It would be more meaningful to compute the input-dependent uncertainty thus, we would deal with the formulation of the heteroscedastic aleatoric uncertainty. For this purpose, we derive intuitions from the regression settings and extend them to a classification model. We rely on this assumption

[1] Entropy is a concept which is defined in the context of information theory. It is used to capture the randomness of a random variable's (x) probability distribution $p(x)$.

[2] In standard deep learning, dropout layers are freezed at test-time. However, it is not the case with MC dropout.

[3] In regression, a multi-dimensional input is mapped to a unary output [1]. We use the same logic for interpreting the mapping of an image to the logit space and consequently use this word "unaries".

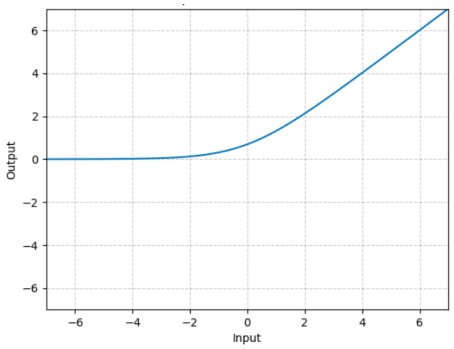


Figure 3.1: Softplus function.

that the noise in an input image sample gets regressed through the network, all the way up to the logit space. Hence, by making use of this analogy, we split the output head of the network to predict both the vector of unaries as well as the variance σ^2 of this noise. In order to learn the parameters of the network, we place a multivariate Gaussian distribution, over the unaries vector in the logit space, with the mean of the distribution equal to the vector of unaries and covariance matrix equal to a diagonal matrix with each entry on the diagonal corresponding to the variance in a particular logit value in the unaries vector. Consider the case of Semantic segmentation, where we predict a vector of unaries $\boldsymbol{f}_i^{\boldsymbol{w}}$ for every pixel i in the image, we modify the network by placing the Gaussian distribution as:

$$\hat{\mathbf{x}}_i|\boldsymbol{w} \sim \mathcal{N}\left(\boldsymbol{f}_i^{\boldsymbol{w}}, \left(\sigma_i^{\boldsymbol{w}}\right)^2\right). \tag{3.4}$$

Here $\hat{\mathbf{x}}_i$ denotes the realization of the unaries vector which is passed through the softmax operation for obtaining the probability vector $\hat{\mathbf{p}}_i$. Mathematically, we denote this as:

$$\hat{\mathbf{p}}_i = \text{softmax}(\hat{\mathbf{x}}_i). \tag{3.5}$$

In equation 3.4, $\boldsymbol{f}_i^{\boldsymbol{w}}$ and $\left(\sigma_i^{\boldsymbol{w}}\right)^2$ are the outputs from our network that represent the vector of unaries and variances, obtained with weights $\boldsymbol{w}$ of the network, respectively. We interpret these variances as the aleatoric uncertainty (for each value in the unaries vector). The vector of variances is diagonalized to obtain a diagonal covariance matrix. Since the individual values of the variance in the variances vector can not be negative, we resolve them to the positive domain by applying the *softplus* function [52]. Mathematically, this function could be written as:

$$\text{softplus}(\mathbf{x}) = \ln(1 + e^{\mathbf{x}}) \tag{3.6}$$

Figure 3.1[4] shows a graphical visualization of the softplus function. Based on the formulations above, we could define the expected likelihood for out network to be:

$$\mathbb{E}_{\mathcal{N}\left(\hat{\mathbf{x}}_i; \boldsymbol{f}_i^{\boldsymbol{w}}, \left(\sigma_i^{\boldsymbol{w}}\right)^2\right)}\left[\hat{\mathbf{p}}_{i,c}\right], \tag{3.7}$$

where c represents the true class label for the pixel i. In order to solve the above objective, it would be ideal to have the closed-form solution instead of marginalizing over the Gaussian

[4] `https://pytorch.org/docs/master/generated/torch.nn.Softplus.html`

distribution. But, there does not exist any such solution so we approximate the objective by MC integration. We extract unaries vector from the Gaussian distribution, pass them through the softmax operation, and subsequently approximate the expected value of the likelihood. In order to sample unaries from the distribution, we apply the re-parametrization trick [31] and eventually write the final loss function for the task of Semantic segmentation as [1]:

$$\begin{aligned} \hat{\mathbf{x}}_{i,t} &= \boldsymbol{f}_i^{\boldsymbol{w}} + \sigma_i^{\boldsymbol{w}} \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I) \\ \mathscr{L}_x &= \sum_i \frac{1}{T} \sum_t \hat{x}_{i,t,c} - \log \sum_{c'=1}^{C} \exp \hat{x}_{i,t,c'}, \end{aligned} \tag{3.8}$$

where T represents the total number of samples extracted from the Gaussian distribution in the logit space. In addition, C represents the total number of possible class labels in the dataset being used for training the network. Further, $\hat{x}_{i,t,c}$ represent the value in the unaries vector corresponding to the actual class label of the pixel i at the t^{th} realization of this vector. Using the above loss function, we can train our model to predict both the class label as well as the aleatoric uncertainty for all the pixels in an image. As we can observe, we do not need any uncertainty labels for learning the uncertainty [1]. Rather, we only need the ground truth labels for the pixels so that we can supervise the training process using the above loss function, for learning the aleatoric variances mapping implicitly.

We have formulated the loss function in 3.8 for the task of Semantic segmentation. However, in the same way, we can formulate a loss function for the task of Image classification as well. Instead of computing the vector of unaries for every pixel in the image, we rather compute a single vector for the whole image and place a Gaussian distribution over it in the logit space. Afterwards, we sample unaries from this distribution and compute the loss, which is propagated backwards for learning the best parameters for predicting both the vector of unaries and aleatoric variances.

So far, we have seen the details of the work of Kendall *et al.* [1] - a practical method which we would use for quantifying different uncertainties as well as for doing some follow-up work. In the next subsection, we would present the empirical evidence given in [1] and suggest some important variations, which we would explore for determining the quality of the uncertainty estimates obtained using this method. Further, we would comment on some loopholes in the work presented in [1] that we try to address with our work.

3.1.1 Possible research questions

For quantifying both epistemic and aleatoric uncertainty with a neural network, we can use the framework presented in section 3.1. Kendall *et al.* [1] use this framework for estimating uncertainties in Depth regression and Semantic segmentation tasks. Since our point of focus is uncertainty estimation in classification settings, we would concentrate on the empirical observations provided for the latter problem. In addition, for the rest of this work, all the comments which we give, would be in the context of classification tasks as well - particularly Image classification and Semantic segmentation. We would clearly state the research questions which we answer in the context of both of these problems, separately.

For achieving the desired goal of uncertainty estimation in Semantic segmentation, Kendall *et al.* [1] use the DenseNet architecture [53] introduced in [54] with the CamVid [55] dataset.

This dataset consists of different images representing outdoor street scenes. The DenseNet architecture has an encoder-decoder like structure where the encoder is responsible for extracting meaningful feature maps from an image by passing it through multiple convolutional layers at different stages, as well as reduce the spatial dimensions of these feature maps by applying the pooling operation along the way. On the other hand, the decoder module is responsible for gradually recovering the spatial resolution of the feature maps (obtained from the encoder) by applying the de-convolution operation along with incorporating the positional information shared by the encoder via skip connections, for obtaining the fine-grained segmentation results. For preventing the overfitting problem, this architecture places dropout with probability $p = 0.2$ after every convolutional layer in the encoder module. With this network, the particular positions of the dropout layers are already defined and thus, we just need to perform MC dropout on the trained network for extracting the epistemic uncertainty. In [1], Kendall *et al.* claim that, applying MC dropout for estimating epistemic uncertainty results in the improved performance of the network in terms of the *mean Intersection over Union* (mIoU) [56] accuracy metric, which is popularly used in Semantic segmentation tasks. However, modern state-of-the-art architectures which have already surpassed DenseNet's performance on this task of image segmentation, does not come with the predefined dropout layers since they use other regularization techniques such as Batch normalization [57] and so forth. So, going along with the formulation of MC dropout, we need to include the dropout layers ourselves. However, as stated in [35] adding dropout layer after every parameter layer in the network can potentially lead to the problem of underfitting, so we need to be careful with the number of dropout layers as well with their locations in the network. Thus, the addition of dropout layers in the modern[5] state-of-the-art architectures for performing MC dropout raises an interesting hypothesis: Would the performance of these networks improve with MC dropout? Further, it raises questions on the exact locations in these network where we should add dropout layers as well as the strength of the dropout probability which we should use. Alongside, since the modern network architectures can process complex datasets competently, it would be interesting to observe if the method presented in [1] transfer well to some other complex outdoor street scenes images dataset.

Along with analyzing the performance of a model in terms of an accuracy metric, it is very important to access the quality of the uncertainty estimates as well since it would directly impact the future decisions which we take based on the strength of these estimates. Kendall *et al.* use calibration plots [58] for their network on the test set. Generally, for obtaining the calibration plot for networks performing Semantic segmentation, we discretize our network's predicted confidences for all the pixels in the test set into a number of bins, and compute the average of all the confidences in each bin. Following that, we compute the classification accuracy for the pixels in each bin and plot it against the average of the confidences computed before. Such type of plots help in understanding the confidence calibration of the model, with the best calibration plot being the straight line: $y = x$, which can be somehow related with the quality of the epistemic uncertainty. Further, there are some popular metrics named as *Expected Calibration Error* (ECE) [59] and *Maximum Calibration Error* (MCE) [59], that can be directly calculated

[5]Modern network architectures use a particular configuration of layers and modules for improving their performance on the accuracy metric. Thus, the addition of dropout layers could potentially interfere with the inner working of these network architectures.

from the calibration plots. Mathematically, ECE is written as:

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} |\text{acc}(B_m) - \text{conf}(B_m)|, \tag{3.9}$$

whereas MCE is denoted as:

$$\text{MCE} = \max_{m \in \{1,\ldots,M\}} |\text{acc}(B_m) - \text{conf}(B_m)| \tag{3.10}$$

In both of the equations 3.9 and 3.10, M represents the total number of bins whereas the keywords $\text{acc}(\cdot)$ and $\text{conf}(\cdot)$ stands for the accuracy and the average confidence of the pixels in a particular bin. Additionally, in equation 3.9, n represents the total number of samples which, in the case of Semantic segmentation, is the total number of pixels in all the images of the test-set. In the bayesian machine learning community, researchers commonly use ECE and MCE metrics for accessing the quality of the (epistemic) uncertainty estimates since they summarize the information contained in the calibration plot. However, the problem is that these calibration plots as well as the ECE and MCE metrics, can only be used for commenting on the quality of the epistemic uncertainty. The quality of the aleatoric uncertainty has nothing to do with the confidence calibration of the network since, as stated in [1], the aleatoric uncertainty only captures the noise inherent in the observations and does not vary much for the out-of-distribution (OOD) examples (whereas the epistemic uncertainty does). Our trained network would probably assign varying confidence to the possible class labels for most of the pixels in such examples instead of giving a high confidence for only one of the classes. Therefore, we would see a high epistemic uncertainty for the pixels in the OOD samples. This connection between the epistemic uncertainty and the confidence vector aligns with the formulation of the calibration plot which could then be used to comment on the quality of the epistemic uncertainty. However, if the aleatoric uncertainty does not have to differ significantly for the OOD examples and has no direct contribution in deciding the class label of the pixels in an input sample, we could not use such plots for analyzing its quality. So, some interesting questions that arise from the above discussion are: 1) Does there exist any alternative method (or metric) for analyzing the quality of both aleatoric and epistemic uncertainty collectively? 2) Does the aleatoric uncertainty vary only slightly for the OOD input samples?

For the computation of the aleatoric uncertainty, Kendall *et al.* [1] use the loss function given in equation 3.8 for training their DenseNet architecture with two output heads. At test-time, out of these two heads, one of them give us the class label and the other head gives us the input dependent aleatoric uncertainty information. In Semantic segmentation, as stated in section 3.1, we get a vector of probabilities (through the softmax operation) and a vector of aleatoric variances for every pixel. Now, the question is that, how exactly should we combine the information from the aleatoric uncertainty vector to a single scalar? There are two options which we could opt for: 1) we can either pick up the value from the aleatoric uncertainty vector at the index which corresponds to the class label for every pixel and get an aleatoric uncertainty map, or 2) we can take the average of the values in the aleatoric uncertainty vector for every pixel and consequently obtain an uncertainty map. In [1], there is not any clear indication of which option was chosen therefore, it would be interesting to filter out the best option out of these two, in terms of the quality of the aleatoric uncertainty map which they give. In addition, Kendall *et al.* [1] empirically show that the networks (DenseNet) capable of capturing

aleatoric uncertainty alone (no dropout layers) or both aleatoric and epistemic uncertainty, show a significant improvement in terms of the mIoU metric in Semantic segmentation. Hence, it would be meaningful to validate this argument in such settings, as well.

As mentioned in section 2.4, predictive uncertainty captures both the epistemic and aleatoric uncertainty information. Kendall *et al.* [1] declare the predictive uncertainty estimate as the epistemic uncertainty. They claim that the epistemic uncertainty captures the stochasticity in the parameters of a network which induces the predictive uncertainty. Thus, they interpret the predictive uncertainty as epistemic and compute the aleatoric uncertainty separately by training their network with the loss function given in the equation 3.8. In addition, in their Semantic segmentation experiments, they have proven that their method produce high epistemic uncertainty estimates (at test-time) for the OOD input examples. From this empirical evidence, we get the idea that the MC dropout produce epistemic uncertainty estimates that correlate with the nature of the input data. But, what about the aleatoric uncertainty? Kendall *et al.* [1] show that their method produce high aleatoric uncertainty either for the objects far away in the camera's coordinate frame or at the boundaries of different objects. However, aleatoric uncertainty has the capacity to capture more difficult concepts which could significantly help in the way we make future decisions based on these estimates. So, it gives birth to a very important question: Is the method presented in [1] capable of capturing the correlations between the aleatoric uncertainty and the complexities in the input data space? Further, based on the above discussion, we could also think of exploring some other metric which could capture the epistemic uncertainty so that we can separate its connection with the predictive uncertainty.

Finally, Kendall *et al.* [1] use precision-recall curves for showing the correlation between the mIoU accuracy metric (in Semantic segmentation) and the uncertainty estimates. They show that the accuracy metric increases if we remove the uncertain pixels, declared by either the aleatoric or the epistemic uncertainty, from the input samples. In essence, precision-recall curves are computed by looking at the classification accuracy and the value of the uncertainty estimates for every pixel separately. However, in Semantic segmentation, it would be more reasonable to look at the uncertainty values in several small regions comprising of pixels in close neighbourhood instead of looking at all the pixels separately. So, some interesting questions that arise from this point are: 1) Does there exist any method which evaluates the uncertainty information by looking at several small regions in an image? and 2) Does that same method give quality metrics (or curves) for the uncertainty estimates that correlate with the performance metric mIoU?

In summary, based on the empirical evidence presented in [1], we have formulated a couple of questions in this subsection which we would want to answer by conducting different experiments. Collectively, these questions are:

1. Does the epistemic uncertainty computation with the technique MC dropout help in improving the performance of modern state-of-the-art vision architectures that do not come with predefined dropout layers?

2. Does the value of the dropout probability affect the quality of the epistemic uncertainty as well as the performance of the network?

3. How do we identify the exact locations for adding the dropout layers in modern network architectures?

4. Does the method given in [1] transfer well to other complex street scenes images datasets?
5. What is the best way of summarizing the information contained in an aleatoric variance vector? Should we pick up the value at the index corresponding to the class label or should we just take the average of this vector?
6. Does the aleatoric uncertainty computation alone or both the aleatoric and the epistemic uncertainty estimation together affect the performance of our network positively?
7. Is there any alternative to the calibration plots which makes it possible to evaluate the quality of both the aleatoric and the epistemic uncertainty collectively?
8. Does there exist any method which can evaluate the uncertainty information for multiple pixels in close neighbourhood rather than just doing it separately for every pixel, in Semantic segmentation?
9. Does this same method produce uncertainty quality metrics (or curves) that correlate with the mIoU accuracy metric?
10. Is there any other metric which could be used for quantifying the epistemic uncertainty?
11. Does the aleatoric uncertainty vary only slightly for the OOD input examples?
12. Does the method presented in [1] capture correlations between the aleatoric uncertainty and the possible complexities in the input data space?
13. What could be the possible input data space complexities that could lead to high value of the aleatoric uncertainty?

In the next sections, we would specify the particular questions, formulated above, that we answer in the context of both the Semantic segmentation and the Image classification tasks.

3.2 Questions answered with Semantic segmentation

In the previous subsection, based on the empirical evidence presented by Kendall *et al.* [1], we have formulated a couple of questions that could potentially clear out the ambiguities in their work. In the context of Semantic segmentation, we would answer the first nine (3.1.1(1-9)) questions. This is because, these questions directly relate with the concrete observations given in [1] and contribute more in validating these observations in the context of new challenges. In addition, some of these questions (3.1.1(7-9)) can only be answered in the Semantic segmentation setting. The rest of the questions (3.1.1(10-13)) are more of an additional work and requires rigorous experimentation for building up strong arguments. Since, Image classification is a simpler version of Semantic segmentation and the networks used in this task are easier to train (requires less time), we would answer the remaining questions by conducting experiments in this setting.

Considering the first nine questions formulated in the previous subsection, our first question (3.1.1(1)) explore the performance of MC dropout with modern state-of-the-art vision CNN

architectures that do not have predefined dropout layers. In this context, initially, Srivastava *et al.* [35] claim that the NN architectures where we use Standard dropout show similar results to using MC dropout with the same network. However, this claim was verified in the context of the Image classification network architectures with dropout layers added only at the final stages of the network, where we have inner-product (fully connected) layers. Note, Standard dropout here refers to the actual definition of dropout in [35] and the way it is used in the literature, where, at test-time, we just scale the activations instead of dropping the weight units, which we do in MC dropout. In response to the empirical evidence provided against MC dropout in [35], Gal [4] shows that the convolutional neural networks which have a dropout layer after every parameter layer in their architecture tend to perform better with MC dropout as compared to the Standard dropout. But, he gives this argument in the context of Image classification networks as well. In Semantic segmentation, we have to deal with a dense prediction task so we suspect that adding a dropout layer after every parameter layer in the network might probably lead to the bad performance. Going over to the second (3.1.1(2)) question, we are interested in finding if the value of the dropout probability affects the performance of MC dropout and thus the quality of the epistemic uncertainty. In [4], it is given that the bigger networks should use a higher value of the dropout probability for getting a better quality of the epistemic uncertainty estimate as well as the performance of the network in terms of the accuracy metric. But again, this claim is given in the context of the Image classification networks. In addition, the author in [4] claims that the quality of the epistemic uncertainty is also dependent on the type of the non-linearity used in a NN. But, for our experiments we would stick to the non-linearity named as Rectified Linear Unit (ReLU)[6] since it commonly used in almost all of the modern networks. For the choice of the dropout probability in Semantic segmentation network architectures, the authors in [60] introduce a new dropout technique called *Concrete dropout*, where they optimize the dropout probability for every layer during the training process. However, for simplicity, we conduct our experiments with a set of two different probabilities: $p = 0.2$ and $p = 0.5$, instead of using Concrete dropout. We choose these two particular values because they are commonly used with the vision architectures. Our third question (3.1.1(3)) where we wish to identify the exact locations in the network where we should add dropout layers, directly answers our first question as well. In [61], the authors claim that for Semantic segmentation network architectures, which possess an encoder-decoder like structure, it is best to add the dropout layers in the middle flow of the network i.e. somewhere between the last modules of the encoder and the decoder. This is because, the initial layers of the network extract very low level features which are more like the same for every input image they process whereas the final layers of the network possess higher level information which is necessary for performing the task of prediction. Therefore, adding the dropout layers after the initial layers of the network might not prove to be that beneficial and their addition after the final layers of the network might potentially lead to the loss of important knowledge. In the wake of supporting their claims regarding the positioning of the dropout layers, the authors in [61] perform a set of multiple experiments so we choose a policy similar to theirs for our experiments too. Thus, considering the first three questions, we conduct experiments where we we add dropout layers in the middle flow of the networks and perform a separate set of experiments with two different dropout probabilities. We would mention the details of the networks that we use in one of the later subsections.

[6] ReLU(x) = max(0, x).

Moving on to the fourth (3.1.1(4)) question, we are interested in observing whether the method presented by Kendall *et al.* [1] works well with the other datasets similar to the one used by them. In their Semantic segmentation experiments, they use CamVid dataset which consists of multiple images representing outdoor street scenes. However, we have a couple of other datasets which are of the similar nature. In this context, Gustafsson *et al.* [62] perform experiments with KITTI [63] and the CityScapes [43] dataset but they quantify the epistemic uncertainty only using MC dropout. Contrary to that, in our experiments, we work with the CityScapes dataset but estimate both the epistemic and the aleatoric uncertainty. As the authors in [62] have shown promising results with the this dataset for the epistemic uncertainty, we believe that the same would apply for the aleatoric uncertainty estimates as well. Considering the fifth question (3.1.1(5)), we aim to classify the best way for summarizing the information contained in the aleatoric variance vector for every pixel. For this purpose, we conduct some experiments and use quantitative metrics for reaching the conclusion. In order to answer the sixth (3.1.1(6)) question, we follow an experimental setup similar to Kendall *et al.* [1]. We train our networks in four different settings: 1) Standard implementation without the dropout layers and the loss function in equation 3.8 which we use as a baseline, 2) Using the loss function in equation 3.8 for estimating the aleatoric uncertainty only (no dropout layers), 3) Adding the dropout layers with different dropout probabilities in the middle flow of the network and using MC dropout for capturing the epistemic uncertainty only, and 4) Combining the addition of the dropout layers with different probabilities (MC dropout) and the usage of the loss function in equation 3.8 for estimating both the epistemic and the aleatoric uncertainty. The experimental setup explained above uses the arguments given for the second as well as the third question and helps in providing a solid answer to not only the sixth but also the first question. In summary, we formulate an experimental setting where we train networks in different configurations with the CityScapes dataset. Further, for all the different network architectures, we look for the best way to process the aleatoric variance vector in one of the experimental configurations and adopt it for the rest.

Considering the remaining questions (3.1.1(7-9)), we want to explore if there exists a method for the Semantic segmentation task, as an alternative to the calibration plots, which could be used to analyze the quality of both the aleatoric and the epistemic uncertainty collectively. In addition, if such a method exists, we want it to look at the uncertainties of the pixels in a close neighbourhood, defined by a small window, for commenting on the quality instead of looking at a single pixel at one time. Subsequently, we want to observe if the uncertainty quality estimates predicted by such a method (if it exists) correlate with the mIoU accuracy metric in the Semantic segmentation task. Fortunately, in this context, Mukhoti *et al.* [2] propose a method, for quantifying uncertainty quality estimates, which integrates all the features that we desire. They propose a framework which looks at both the predictions and the uncertainty maps together, for all the images in a test-set, in order to report an average quality metric for the uncertainty estimates computed by a particular network configuration. Although, they use this framework for reporting the quality of the predictive and the epistemic uncertainty only, we extend it to predict the quality of the aleatoric uncertainty as well. We do it because, this method computes the quality estimates by keeping an eye on both the prediction capabilities as well as the uncertainty information produced by a network. Since, a high aleatoric uncertainty can potentially interfere with the predictions from a network, the use of the method presented in [2], for quantifying the aleatoric uncertainty quality, makes sense. Additionally, since we

know that the calibration plots as well as the ECE and MCE metrics are well known tools for observing the quality of the epistemic uncertainty, we compute them along with the new quality estimates, using the method in [2], for observing the correctness of our new method, specifically by looking at the epistemic uncertainty quality estimate. The authors in [2] name the metric, which holds the quality information of an uncertainty estimate, as *Patch Accuracy vs Patch Uncertainty* (PAvsPU). In the next subsection, we will present the steps required for computing this metric.

3.2.1 Method for estimating the uncertainty quality

As mentioned in the previous section, we want our uncertainty quality estimation method to satisfy some of the constraints. Fortunately, the work presented by the authors in [2] comply with all of them. In order to formulate a metric for quantifying the uncertainty quality, they rely on two intuitive desiderata:

1. **Desideratum 1**: if a network is certain about its predictions, it should be accurate on the same.

2. **Desideratum 2**: if a network produces inaccurate predictions, it should be uncertain about the same.

Note, that the converse of these desiderata might not hold in some situations [2]. In this context, let us imagine we have pixels in an image for which our network produces a very high value of the epistemic uncertainty. It could happen if the pixels in a certain image are not that frequent in the other images. So, even though our network tends to be uncertain on such pixels, it still has the capacity to classify them accurately.

Following the above give desiderate, the authors in [2] define two different conditional probabilities that they declare as:

1. $p(\textbf{accurate}|\textbf{certain})$: Given that the network is certain about its predictions, what is the probability that it is accurate on the same?

2. $p(\textbf{uncertain}|\textbf{inaccurate})$: Given that the network has produced inaccurate predictions, what is the probability that it is uncertain about the same?

For computing the above probabilities, we start with first choosing a patch size w and subsequently defining a window with dimensions $w \times w$. Following that, we traverse the predicted labels (from the network) and the true labels synchronously, in the same way as we do in the convolution operation but with non-overlapping windows, and compute the *pixel accuracy* [56] of all the patches selected by our window. Here, the pixel accuracy metric is like the standard accuracy metric which we use in the classification problems, where just divide the number of accurate predictions by the total number of predictions. For the patches where we observe the pixel accuracy greater than a certain threshold, we mark them as *accurate* or otherwise *inaccurate* [2].

In a similar way, we traverse the uncertainty map and compute the average uncertainty for every patch selected by our window. For the patches where we observe the average uncertainty less than a certain threshold, we mark them as *certain* or otherwise *uncertain* [2]. In Semantic

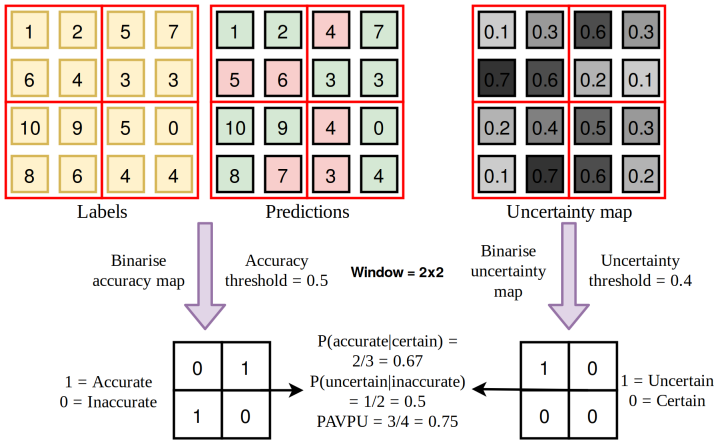


Figure 3.2: Example demonstrating the calculation of the metrics required for quantifying the quality of the uncertainty estimates [2].

segmentation, the size of the uncertainty map, in either of the dimensions, is the same as the prediction map so the patches selected by our window for predictions and the true label are the same as the ones selected in the uncertainty map.

Once the above steps have been performed, we calculate the number of patches which are accurate and certain (n_{ac}), accurate and uncertain (n_{au}), inaccurate and uncertain (n_{iu}) and inaccurate and certain (n_{ic}). Following that, we compute the conditional probabilities $p(\textbf{accurate}|\textbf{certain})$ and $p(\textbf{uncertain}|\textbf{inaccurate})$ as [2]:

$$\begin{aligned} p(\textbf{accurate}|\textbf{certain}) &= \frac{n_{ac}}{(n_{ac}+n_{ic})} \\ p(\textbf{uncertain}|\textbf{inaccurate}) &= \frac{n_{iu}}{(n_{ic}+n_{iu})}, \end{aligned} \tag{3.11}$$

and combine them to obtain the final quality metric *Patch Accuracy vs Patch Uncertainty* (PAvsPU) as:

$$\text{PAvPU} = \frac{(n_{ac} + n_{iu})}{(n_{ac} + n_{au} + n_{ic} + n_{iu})} \tag{3.12}$$

Figure 3.2 shows an example for the process of computing the above formulated metrics.

In [2], the authors use the PAvsPU metric for commenting on the quality of the predictive as well as the epistemic uncertainty estimates and demonstrate very promising results with the CityScapes dataset. As mentioned before, we need to compare the pixel accuracy and the average uncertainty of the patches with a certain threshold. In [2] the authors use a threshold of **0.5** for the pixel accuracy and for the uncertainty threshold they compute an average of the uncertainty for all the pixels in the images of the test-set. For our experiments, we follow the similar policy for deciding the thresholds as well. We perform the traversal step with both the aleatoric and the epistemic uncertainty separately and comment on the quality by looking at their PAvsPU metrics respectively.

Based on the information in section 3.2 and in this subsection, we now have tools which would help us build our experimental setup. In the next subsection, we would look at the network architectures that we would use in our Semantic segmentation experiments.

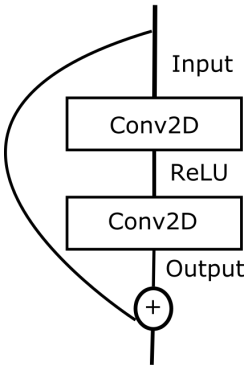


Figure 3.3: Residual learning block in ResNets. Here, Conv2D refers to the convolutional layers in a CNN.

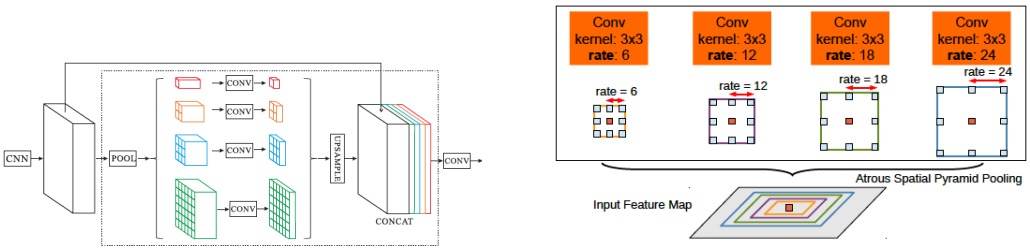
3.2.2 Network architectures used

For our Semantic segmentation experiments, we use two different modern state-of-the-art network architectures named as *Pyramid Scene Parsing Network* (PSPNet) [8] and *DeepLabv3+* [9]. Both of these networks use the main building blocks of *Residual Networks* (ResNets) [15] and introduce some new modules for performing the dense prediction task of Semantic segmentation. ResNets are considered to be one of the biggest advancements in the field of deep learning. As deep neural networks are prone to the vanishing gradient problem, the residual learning blocks[7] in ResNets help in avoiding that problem greatly. The concept of the residual learning block is pretty simple - let us imagine that we have a module in our network consisting of two convolutional layers. According to the concept of residual learning, we just introduce a branch which connects the input of this module to its output by the summation operation. This simple branching concept, as shown in figure 3.3, safeguards DNNs from the vanishing gradient problem. In essence, ResNets combine the concept of branching with a structure of layers similar to the VGG [14] network where multiple convolutional layers are stacked on top of one another.

PSPNet is a state-of-the-art architecture for performing the task of Semantic segmentation. In [8], the authors have shown promising results for this network with multiple datasets including CityScapes, the dataset which we use in our experiments. PSPNet has a fully convolutional network structure [56] and uses either ResNet50 or ResNet101 [15] as its backbone network. On top of this backbone network, PSPNet incorporates a *Pyramid Pooling Module* (PPM) [8] which pools the information contained in the feature maps at different scales in a pyramidal structure, upsample the variable size pooled features via bilinear interpolation and finally concatenate these upsampled feature maps. In terms of the similarity of the PSPNet with the encoder-decoder network structure, the ResNet backbone in PSPNet can be interpreted as an encoder which extracts the important features from an image whereas part of the PPM which performs the upsampling operation can be considered as a decoder.

In our experiments, we use PSPNet with ResNet50 as its backbone. For obtaining a Bayesian version of this network, we insert dropout layers after every *Bottleneck* block [15] in the ResNet50 backbone since it forms the middle flow of the network. As there are four bottleneck blocks

[7]For a more detailed explanation of this concept, the reader is recommended to look at [15].



(a) Pyramid Pooling Module in PSPNet [8]

(b) Atrous Spatial Pyramid Pooling Module in DeepLabv3+ [9]

Figure 3.4: Illustration of the Pyramid Pooling Module in PSPNet and Atrous Spatial Pooling Module in DeepLabv3+. In (a), we can see how the Pyramid Pooling Module in PSPNet summarizes the information from the feature maps obtained from the backbone network. In (b), we have an illustration of the Atrous Spatial Pooling Module in DeepLabv3+ which processes the information obtained from the backbone network at different scales using dilated convolutions. Here, rate refers to the dilation rate in different dilated convolutional layers.

in total, we add four dropout layers and perform our set of experiments. Additionally, there is another interesting thing to note about the PSPNet architecture. In the final block, immediately after the PPM module, we have a 1 x 1 convolutional layer, for reducing the number of channels in the feature map, and a *Dropout2D* layer [64] with dropout probability $p = 0.1$ which excludes one of the feature maps completely. Hence, we conduct another set of experiments by excluding this *Dropout2D* layer for observing its effects on the performance of the network as well as on the quality of the aleatoric and epistemic uncertainty estimates.

Along with PSPNet, we have DeepLabv3+ which is another state-of-the-art architecture for Semantic segmentation tasks. Just like PSPNet, DeepLabv3+ uses either VGG-16 [14], ResNet101 [15] or Xception [65] as its backbone network. On top of that, it places a *Atrous Spatial Pyramid Pooling* (ASPP) [9] module which uses dilated convolutions [66] for summarizing the information contained in the feature maps. In addition, these dilated convolutions help with dealing with the problems of objects present at different scales within an image as well. Further, this ASPP module performs *depth-wise separable* [9] convolutions in order to save the number of parameters. As a final comment, DeepLabv3+ network architecture also possesses an encoder-decoder like structure similar to the PSPNet.

For our experiments, we use DeepLabv3+ with ResNet101 as its backbone. In order to make it capable of capturing the epistemic uncertainty, we add dropout layers after the bottleneck blocks in the backbone network. Since, there are four bottle neck blocks, we add a total of four dropout layers. Note, that the bottleneck blocks in ResNet50 and ResNet101 are different in terms of the number of convolutional layers which each bottleneck block have. Clearly, the number of convolutional layers in ResNet101 bottleneck blocks are more as compared to the bottleneck blocks in ResNet50.

In this subsection, we gave a comprehensive overview of the network architectures which we use in our Semantic segmentaion experiments. Now, in order to have a clear picture of the experimental setup which we use, we will give some details in the next subsection.

3.2.3 Experimental setup

In section 3.2, we explained our approach towards finding the answers to the questions which we formulated in the context of Semantic segmentation task. Further, in subsections 3.2.1 and 3.2.2, we gave an overview of a metric for quantifying the uncertainty estimates quality and the network architectures which we use in our experiments, respectively. Now, we would explain some details of our overall experimental setup.

Notation remark: In this subsection and for the rest of this work, we would use the word **Deterministic** for referring to the standard implementation of the network which does not use either MC dropout or the loss function in the equation 3.8. We use standard cross-entropy loss function [67] for training this network configuration. Apart from that, we would use the word **Aleatoric** for referring to the network implementation which use the loss function in the equation 3.8 but does not use MC dropout. In addition, we would refer to the loss function in the equation 3.8 as the **Aleatoric loss**. Further, we would use the word **Epistemic** for referring to the network configuration which uses only MC dropout with the standard cross-entropy loss function for training purposes. For a short representation of MC dropout, we would use the word **MC**. Additionally, for referring to the Standard dropout, we would use the word **Normal** which means that the network does not use MC dropout at test-time for quantifying the uncertainty estimates and uses dropout in a way it is used in the literature. We would represent the dropout probability as **p** and use expressions such as **p = 0.2** for denoting its strength. Along with that, we would use the words **PAvsPU(epistemic)** and **PAvsPU(aleatoric)** for representing the PAvsPU metric for the epistemic and the aleatoric uncertainty respectively. Finally, we would use the phrase **Aleatoric + Epistemic** for referring to the network implementation which uses both the aleatoric loss and MC dropout.

For both PSPNet and DeepLabv3+, we train and evaluate the following network configurations:

- Deterministic.
- Aleatoric.
- Epistemic (p = 0.2 and p = 0.5).
- Aleatoric + Epistemic (p = 0.2 and p = 0.5).

Additionally, for the PSPNet, we also train and evaluate these same network configurations but without the Dropout2D layer in its final block. We refer to the network with the Dropout2D layer as **Spatial Dropout** and the network without the Dropout2D layer as **No Spatial Dropout**.

In order to evaluate the performance of the above mentioned network configurations, we use the mIoU [56] metric which is popularly used for the task of Semantic segmentation. For quantifying the epistemic uncertainty, we compute the entropy of the probability vector, obtained after MC dropout, as demonstrated in the equation 3.3. However, for the aleatoric uncertainty estimate we first look for the best way to summarize the information in the aleatoric variance vector using our *Aleatoric* network configuration and then use the same for the others. Note, in the Aleatoric + Epistemic configuration, we perform MC dropout by computing multiple forward passes through the network which means that we get an average probability vector as

well as an average aleatoric variance vector. For measuring the quality of the epistemic uncertainty we compute ECE (equation 3.9), MCE (equation 3.10) and PAvsPU metric (equation 3.12) wheras for quantifying the aleatoric uncertainty quality we only use the PAvsPU metric. We provide different plots for the graphical visualization of some of these uncertainty quality metrics as well. Finally, we give visual results obtained from our networks but only for a single experiment type, which gives the best result in terms of the mIoU metric, within each of the above mentioned network configurations. We do this because, it is very difficult to spot the differences in the visual results as compared to the differences in the numerical figures.

3.3 Questions answered with Image classification

Out of all the questions formulated in subsection 3.1.1, we answer the last four (10-13) in the context of the Image classification task. While answering these questions, we discovered a way in which we could process the uncertainty estimates further by designing a classifier, which partitions the input image samples to different categories based on their respective uncertainty values. Thus, in this section we would present the way in which we answer the desired questions but present the details of this classifier later.

Considering the first question (3.1.1(10)), we want to explore some other metric which could be used for quantifying the epistemic uncertainty. As already discussed, epistemic uncertainty is computed by first imposing a prior distribution over the space of parameters and later observing the change in this distribution as we keep on observing more data. For capturing our updated belief of this prior distribution, called the posterior, we perform approximate variational inference. Later, during the inference, we use this posterior distribution for computing a predictive distribution over our new input samples. In our work, we use the dropout variational inference method for performing these set of operations. This inference method can be interpreted as training a standard neural network with the dropout layers as described in detail in the subsubsection 2.3.2.1. In [4], the author mentions some information theoretic metrics that could be used for quantifying different uncertainty estimates. For predictive uncertainty, the author suggests computing the entropy of the predictive distribution, which is equivalent to computing the entropy of the average probability vector obtained after MC dropout, as explained in section 3.1. But, for estimating the epistemic uncertainty the author suggests computing another metric named as *Mutual information*. This metric basically estimates the variance of the results produced by the networks in an ensemble. In machine learning, *ensembling* [68] is a technique where we train a same network architecture several times but with different weight initializations. At test-time, we take an average of the result produced by these networks (in an ensemble) for reporting the final prediction. In our case, MC dropout could be considered as an ensembling technique as well. Every stochastic forward pass through a network selects a different set of parameters (posterior sampling), which induces a different function mapping, that could be interpreted as a separate network. Rewriting equation 3.3 with the help of the equation 3.2 as:

$$H(\mathbf{p}) = -\sum_{c=1}^{C}\left(\frac{1}{T}\sum_{t}\text{softmax}\left(\boldsymbol{f}^{\hat{\boldsymbol{w}}_t{}^*}(\mathbf{x}^*)\right)\right)\log\left(\frac{1}{T}\sum_{t}\text{softmax}\left(\boldsymbol{f}^{\hat{\boldsymbol{w}}_t{}^*}(\mathbf{x}^*)\right)\right), \tag{3.13}$$

we write the equation for the mutual information as:

$$I(\mathbf{p}, \boldsymbol{w}) = H(\mathbf{p}) + \frac{1}{T} \sum_{c,t} \Big(\text{softmax}\Big(\boldsymbol{f}^{\hat{\boldsymbol{w}}_t{}^*}(\mathbf{x}^*) \Big) \Big) \log\Big(\text{softmax}\Big(\boldsymbol{f}^{\hat{\boldsymbol{w}}_t{}^*}(\mathbf{x}^*) \Big) \Big). \tag{3.14}$$

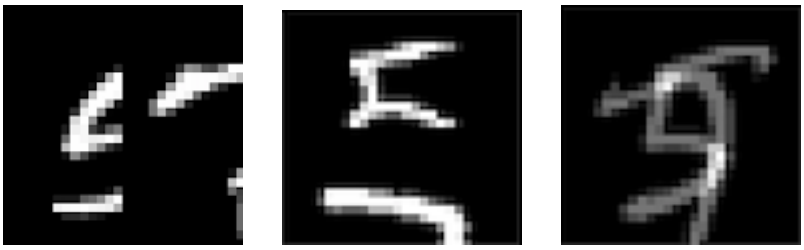
In the above two equations, C represents the total number of possible class labels whereas T denotes the number of stochastic forward passes made through the network. In addition, $\boldsymbol{f}$ represents a CNN and $\hat{\boldsymbol{w}}_t{}^*$ represents a realization of $\boldsymbol{w}^*$ at the t^{th} forward pass. For our experiments, we compute the mutual information metric for representing the epistemic uncertainty. In addition, based on Kendall *et al.* [1] argument of considering the predictive uncertainty as epistemic, we also keep a track of the predictive uncertainty. Since the epistemic uncertainty becomes significant for the OOD input samples, we add some of these samples in our test-set and observe the value of both the mutual information metric and the predictive uncertainty for deciding which one detects such samples with the best accuracy.

Moving on to the next question (3.1.1(11)), we are interested in observing whether the aleatoric uncertainty vary only slightly or significantly for the OOD input samples. As mentioned before, we add such samples to our test-set for differentiating between the power of the mutual information metric and the predictive uncertainty estimate to detect such samples. Thus, we compute the aleatoric uncertainty estimate of these OOD examples as well and exploit it to reach the answer to our question. In all our experiments, we have some OOD input samples (in the test-set) which makes us capable of reaching strong conclusions.

Considering the last two questions (3.1.1(12-13)), we want to explore the possible complexities in the input data space, that could lead to a higher value of the aleatoric uncertainty, and see if the method presented by Kendall *et al.* [1] is capable of detecting these complexities or not. For possible complexities that result in a significant epistemic uncertainty estimate, we have a well developed claim for the OOD input samples and the misclassified examples[8]. However, for the aleatoric uncertainty, it is hard to find the literature which gives some empirical observations. Therefore, we take on this challenge and conduct multiple experiments with different network architectures and datasets for giving our contribution. We conduct our experiments in the Image classification task setting, since it is an easier task to perform and enables one to conduct rigorous experimentation by training and evaluating the networks in a shorter time as compared to the network architectures used for the Semantic segmentation task.

As mentioned in the section 2.4, we could potentially observe a high aleatoric uncertainty for the input data samples that lie at the boundary of the two different class clusters in the input data space. We refer to such samples as *boundary data samples*. Exploiting this fact, we design various experiments where we create these samples ourselves. The main goal is to produce the boundary data samples that confuse our network the most. Since, aleatoric uncertainty is mainly modelled for the *In-Distribution* (ID) samples for detecting their noise levels, we need to include these (potentially higher aleatoric uncertainty) boundary data samples in the network's training distribution, such that our network learns to predict a high aleatoric uncertainty for such samples at the test-time. Therefore, while training the network, we create these samples from our training data and while testing, we create them from our test-data. However, the question is, how exactly do we produce these boundary data samples? There are several ways in which

[8]We get a higher value of the epistemic uncertainty for the OOD samples since our network does not possess enough capacity to assign accurate class labels to them. In that sense, we should expect a high epistemic uncertainty for the misclassified examples as well and the authors in [69] have already proven this fact.



(a) Horizontal combination of an image with class label 5 and an image with class label 7 from the MNIST dataset

(b) Vertical combination of an image with class label 5 and an image with class label 7 from the MNIST dataset

(c) Equal weight combination of an image with class label 5 and an image with class label 7 from the MNIST dataset

Figure 3.5: Illustration of the possible combinations of an image with the class label 5 and the class label 7 from the MNIST dataset [10] that can represent a boundary data sample.

it could be done. For our experiments, we randomly select half of the images belonging to two different classes and combine them either horizontally, vertically or just add them with equal weights. In order to make it clear, let us say we have an arbitrary dataset where we have 1000 images belonging to the class label 5 and 1000 images belonging to the class label 7. In addition, let us assume that every image in both of these classes has a spatial resolution of 28 x 28. According to our experimental policy, we randomly select 500 images with the class label 5 and the same amount with the class label 7. For combining these images horizontally, we take one half of the 500 images with the class label 5 (spatial resolution: 28 x 14) and the other half from the 500 images with the class label 7 (spatial resolution: 28 x 14) and concatenate them together. In that way, we obtain 500 boundary data samples. We perform the same set of operations for combining the images vertically. However, for combining the images with equal weights, we just add the pixels of the randomly sampled images together rather than performing the above mentioned operations for the horizontal and vertical combinations. So, if we call an image with the class label 5 as x_1 and an image with class label 7 as x_2, we get the boundary data sample by: $0.5 * x_1 + 0.5 * x_2$. Figure 3.5 shows how a boundary data sample, created from an image with the class label 5 and an image with the class label 7 from the MNIST dataset [10] looks like based on the above mentioned combinations. Here, it is important to note that the images which we select from two different classes for creating the boundary data samples are the copies of the original images. This is done to avoid the misbalance of different class images.

In the previous paragraph, we have formulated different ways in which we could create the boundary data samples. However, since Image classification is a supervised learning task, we need to come up with a policy for assigning the labels to the training boundary data samples as well. For that purpose, we first train our networks with different labelling policies. Later, at test-time, we identify the labelling policies that managed to confuse our networks successfully by looking at the aleatoric uncertainty estimate of the test boundary data samples. In that way, we come up with a set of labelling techniques that could be used for assigning the labels to the training boundary data samples.

For our experiments, we work with three different datasets named as MNIST [10], FashionMNIST [11] and CIFAR10 [12]. MNIST dataset consists of the images representing handwritten digits from one to ten whereas FashionMNIST contains images representing different pieces of clothing. Finally, CIFAR10 dataset is a mixture of images belonging to different categories such as cats, dogs, cars, ships and so forth. In all of these datasets, we have a total of ten possible class labels for all the images in the particular datasets. To start off, we conduct all of our experiments with the MNIST dataset and identify the experiments which produce the best results. Later, we conduct these best performing experiments with the other datasets for observing whether we get the similar results with other datasets or not. Following are the details of all the experiments which we initially perform with the MNIST dataset:

- Combining half of the class 5 and half of the class 7 images from the training data horizontally. We assign half of these combined images the class label of 5 and half of them the class label 7. For evaluation purposes, from the test-set, we combine half images from the class label 5 and half images from the class label 7 in the same way in order to produce the test boundary data samples.

- Combining half of the class 5 and half of the class 7 images from the training data vertically. We assign half of these combined images the class label of 5 and half of them the class label 7. For evaluation purposes, from the test-set, we combine half images from the class label 5 and half images from the class label 7 in the same way in order to produce the test boundary data samples.

- Combining half of the class 1 and half of the class 3 images from the training data horizontally. We assign half of these combined images the class label of 1 and half of them the class label 3. For evaluation purposes, from the test-set, we combine half images from the class label 1 and half images from the class label 3 in the same way in order to produce the boundary data samples.

- Combining half of the class 1 and half of the class 3 images from the training data vertically. We assign half of these combined images the class label of 1 and half of them the class label 3. For evaluation purposes, from the test-set, we combine half images from the class label 1 and half images from the class label 3 in the same way in order to produce the boundary data samples.

- Combining half of the class 5 and half of the class 7 images from the training data with equal weights of 0.5. We assign half of these combined images the class label of 5 and half of them the class label 7. For evaluation purposes, from the test-set, we combine half images from the class label 5 and half images from the class label 7 in the same way in order to produce the boundary data samples.

- Combining half of the class 5 and half of the class 7 images from the training data horizontally. In this case, we get a one-hot encoding representation of the labels for all the images in the training data including the combined images. For the combined images, in the one-hot encoded label vector, we assign a soft label of 0.5 at the class index 5 and the same soft label at the class index 7. For evaluation purposes, from the test-set, we combine half images from the class label 5 and half images from the class label 7, in the same way as the training data, in order to produce the boundary data samples.

In the experiments given above, we have mentioned two different class combinations for creating boundary data examples: (5, 7) and (1, 3). We do that just for verifying if the results on one class combination hold for some other class combination or not.

In this section, we highlighted the way in which we tend to answer the questions formulated in the context of the Image classification task. We gave details of the set of experiments which we initially perform with the MNIST dataset. In addition, we mentioned some other datasets with which we conduct the best performing experiments from the MNIST dataset, in order to further validate our empirical observations. For all of the datasets, we use a different network architecture. We would mention the details of these architectures in one of the later subsections. But before that, we would give explanation of some additional work which we do after observing the results from our experiments.

3.3.1 Additional work

In this section, we plan on giving details of some additional work which we did apart from answering the research questions that we formulated in the context of the Image classification task.

As mentioned in the previous section, we keep a track of the mutual information metric and the predictive uncertainty estimate as the representative candidates for the epistemic uncertainty. We know that the epistemic uncertainty becomes significant for the OOD input examples - the fact that we leverage for commenting on the capability of the mutual information metric to capture the epistemic uncertainty. Based on this information, we add some OOD input samples to our test-data stream and observe the predictive uncertainty as well as the mutual information metric for these examples.

Additionally, in the previous section, we also gave details of the possible ways in which we could produce the boundary data examples and subsequently the different set of experiments, for observing the capability of the method presented by Kendall *et al.* [1] to give a high aleatoric uncertainty for these examples. While explaining the details of these experiments, we mentioned that we produce some boundary data examples from the test-set and include them in the overall test-data stream for evaluating the performance of our trained networks in different experimental configurations. Thus, based on the above mentioned details, the final test-set which we have, consists of the normal input examples, boundary data examples and the OOD input examples.

In order to evaluate the results of our experiments and consequently reach an answer to our research questions, we first looked at the absolute values of the uncertainty estimates[9] for the normal input examples, OOD examples and the boundary data examples. But later, we discovered that we could instead pose it as a different task which lets us compute some metric that we could use for commenting on the power of the mutual information metric, predictive uncertainty estimate as well as the results of our experiments. So, following this idea, we formulated three different tasks: 1) Detecting OOD input examples, 2) Detecting misclassified examples and 3) Detecting high aleatoric uncertainty boundary data examples. In all of these tasks, we chose the deciding factor to be only the values of the predictive uncertainty, mutual information

[9]Note, as mentioned in the section 2.4, for the Image classification task, we take an average of the uncertainties for all the samples belonging to a certain class and report only one value for the whole class.

and the aleatoric uncertainty. Considering the present literature, the authors in [69] perform the first two tasks of detecting the OOD and the misclassified examples based on the predicitve uncertainty, mutual information and some other information theoretic metrics. However, in our work, we additionally perform the task of detecting the boundary data examples by incorporating the aleatoric uncertainty estimate as well. In essence, the main goal is to observe the efficiency of different uncertainty estimates in helping us detect the OOD, misclassified and the boundary data examples. For this purpose, we follow a very simple approach, similar to the one used by the authors in [69]. We compute the *Area Under the Receiver Operating Characteristic* (AUROC) curve for every uncertainty estimate in all the above mentioned tasks and use it as a metric to gauge the power of the uncertainy estimates at hand. We know that we should see a high predictive uncertainty as well as the mutual information for the OOD and the misclassified examples whereas a high aleatoric uncertainty for the boundary data examples. Exploiting these facts, we compute the AUROC metric for all the uncertainty estimates in our different tasks as:

- **Detecting OOD input examples:** For this task, we give a class label of 1 to the OOD samples in our test-data and a class label of 0 to all the other examples. Following that, we compute the AUROC metric by using these true labels and applying different thresholds[10] on all the uncertainty estimates separately. In that way, we get the AUROC metric for all the available uncertainties.

- **Detecting misclassified input examples:** For this task, we give a class label of 1 to the misclassified samples from our test-data and a class label of 0 to all the other examples excluding the OOD and the boundary data samples. Following that, we compute the AUROC metric by using these true labels and applying different thresholds on all the uncertainty estimates separately. In that way, we get the AUROC metric for all the available uncertainties.

- **Detecting boundary data examples:** For this task, we give a class label of 1 to the boundary data examples in our test-set and a class label of 0 to all the other examples excluding the OOD samples. Following that, we compute the AUROC metric by using these true labels and applying different thresholds on all the uncertainty estimates separately. In that way, we get the AUROC metric for all the available uncertainties.

Thus, computing these AUROC metrics help us in summarizing our observations in a better way along with providing clear answers to the research questions that we formulated in the context of the Image classification task.

For the computation of the above mentioned metric, we need to assign the true labels to different examples ourselves. However, in practical situations where our network is live in the production environment, we would not know about the nature of the data beforehand. In this case, based on the uncertainty estimates, our network should be able to detect the different kinds of data by itself and accordingly make an intelligent decision. So, keeping this practical problem in mind, we propose a simple classifier design that can process the uncertainty estimates from our main network for classifying the data into different categories. In the next

[10]For a detailed description of this operation, the reader is recommended to look at the documentation on this link: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html`

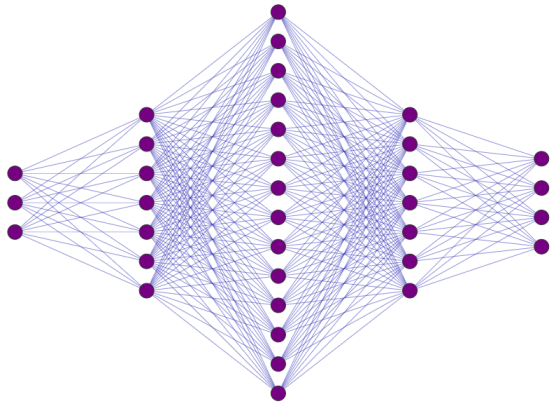


Figure 3.6: Visualization of the structure of our MLP classifier.

sub-subsection, we would give the details of this classifier along with the different categories in which it classifies our data into.

3.3.1.1 Uncertainty estimates based classifier

As mentioned in the previous subsection, we compute the AUROC metric for detecting the OOD, misclassifed and the boundary data examples in our test-set. However, the computation of this metric requires the knowledge of the nature of the data beforehand. In practical situations, where we do not have any clue about the nature of our input data (in production settings), we need to come up with an alternative way of making our network decide on the type of the data based on the uncertainty estimates produced by it, so that it could act intelligently in such situations. For this purpose, we introduce a simple classifier that could be attached at the end of our main network and solely performs the task of dividing the input data into different categories.

For formulating the structure of this new classifier, our test-data stream stays the same as in the previous cases i.e. it includes the normal data, OOD data and the boundary data examples. While performing our experiments, as described in the section 3.3, we noticed a certain pattern in the uncertainty estimates, especially for the best performing experiments, for different data kinds. Based on this observation, we designed a simple *Multi-layer Perceptron* (MLP) classifier with only three hidden layers. The number of neurons in these three hidden layers are 50, 100 and 50 respectively. Figure 3.6 shows how the structure of our MLP classifier looks like. We train this classifier using the standard cross-entropy loss and obtain the training data as the uncertainty estimates of our training data, in different experimental configurations (given in section 3.3), computed from our main network. However, the training data which we use in our experiments does not consist of the OOD examples since we only add them during the test-time, while evaluating the performance of our networks in different experiments, so for the purpose of training our new classifer we include them separately. In summary, the training data of our new classifer contains the numerical uncertainty estimates for the correctly classified, misclassified, OOD and the boundary data examples. So, based on this data distribution, we come up with four different classes that our classifier partitions the data into:

1. Correctly classified data.

2. Misclassified data.

3. Out-of-distribution (OOD) data.

4. High aleatoric uncertainty data (or boundary data examples).

. However, the interesting thing to note is, for a main network which has been trained to the point of convergence, we would see quite a few misclassified examples as compared to the other data kinds. Here, we consider that we have an equal amount of the OOD and the boundary data examples in our classifier training data as the normal input examples. Thus, getting a few misclassified examples would lead to the problem of the misbalanced dataset which would severly affect the performance of our MLP classifier. In our experiments, we resolve this issue by using the *Synthetic Minority Over-sampling Technique* (SMOTE) [70] for the misclassified examples and *Tomek link* (T-link) [71] downsampling technique for the rest of the examples. SMOTE upsamples the minority class by creating the synthetic k-nearest neighbours whereas T-link downsamples the majority class by dropping the samples that are very close to the samples of the minority class. Based on the application of these techniques, we get a balanced dataset which can now be used for training our classifier.

We train our new classifier for the uncertainty estimates, produced by the main network, only in the best performing experimental configurations, for all the datasets. Initially, while training our new classifier we used the uncertainty estimates, i.e. predictive uncertainty, mutual information and the aleatoric uncertainty, for all the input data samples. But, we got average results with this configuration since we noticed that our classifier confuses the misclassified and the OOD examples quite a lot. So, in order to improve the performance, we came up with an alternative strategy. We decided to separate the OOD and the ID (high aleatoric data, correctly classified and misclassified) data first and then use our new classifier only on the ID data for partitioning it into three different categories: 1) Correctly classified data, 2) Misclassified data and 3) High aleatoric data. Thus, our classifier now finds a decision boundary between these three different classes as compared to the four before. For separating the OOD and the ID data in the very initial stage, we follow a very simple principle. In theory, for a confidence-calibrated network, we should observe a uniform distribution of the class probabilities for the OOD data samples, that we get after applying the softmax operation on their corresponding logit vectors. If we have ten possible class labels then, in the best case, we should get a probability of 0.1 for every class. This is because, our network has not seen this data during its training phase so it should be uncertain about its decision rather than adopting a high bias towards one of the classes. Practically speaking, it is nearly impossible to get such a perfect uniform distribution but still, we exploit this fact and compute the KL divergence of all the input data samples probability vectors with a uniformly distributed probability vector. We get a high value of the KL divergence for the ID samples but for the OOD samples the value is not that significant. Thus, we fit a Gaussian distribution $\mathcal{N}(\mu_{ID}, \sigma^2_{ID})$ over the KL divergence values of the ID samples and make the following rule based on this distribution: *if the KL divergence of a sample is one standard deviation (measured from the mean) less, then that sample is OOD*. Making use of this new approach, we saw better classification results for all of our categories listed above as compared to what we observed with our initial technique.

As mentioned in the section 3.3, we conduct all of our designed experiments with the MNIST dataset first and then conduct the best performing experiments with the other datasets. In order to show the efficiency of our new classifier, we perform the above given classification task with the best performing experimental configurations as well, even in the case of the MNIST dataset. However, when we experimented our new classifier with the FashionMNIST and the CIFAR10 dataset, we noticed an average performance for the ID and the OOD data separation task. In this regard, in order to make an effort for bringing in some improvements, we re-trained our main networks with a new loss function which includes the entropy of a sample's probability vector as a regularization term. This way, our network learns to give peaky distributions for the ID examples so that it becomes easy to separate them from the OOD samples at the later classification stage. Mathematically, we could re-write the loss function given in 3.8 for deriving our new loss function for training the main network in the Image classification settings as:

$$\mathscr{L}_x = \left(\frac{1}{T} \sum_t \hat{x}_{t,c} - \log \sum_{c'=1}^{C} \exp \hat{x}_{t,c'} \right) + \left(- \sum_{c'=1}^{C} \hat{p}_{t,c'} \log \hat{p}_{t,c'} \right), \tag{3.15}$$

where T represents the number of samples extracted from the logit space Gaussian distribution (for learning the aleatoric uncertainty mapping) and C represents the total number of possible class labels. In addition, $\hat{p}_{t,c'} = \text{softmax}(\hat{x}_{t,c'})$. We demonstrate the performance of our new loss function for training the main network, which subsequently helps in improving the performance of our new classifier, for only a few experimental configurations in the case of the FashionMNIST and the CIFAR10 dataset.

In this sub-subsection, we explained the details of a new classifier which we designed for partitioning the data into different class categories based on the uncertainty estimates. We explained the significance of this classifier in real-time practical situations. Now, in the next section, we will provide the details of the main network architectures, for every dataset, that we use in our experiments where we create boundary data examples and use different labelling techniques for identifying the ones which confuse our network the most.

3.3.2 Network architectures used

As mentioned in the section 3.3, we work with three different datasets, for conducting our experiments in the context of the Image classification task, named as: MNIST [10], FashionMNIST [11] and CIFAR10 [12]. For every dataset, we use a different network architecture and we would mention the details of these architectures separately in this section.

While conducting experiments with the MNIST dataset, we work with the *LeNet5* [13] network architecture. It consists of just two convolutional layers and two inner-product (fully connected) layers at the end. In order to perform MC dropout with this network for capturing the epistemic uncertainty, we introduce two dropout layers - one after the second convolutional layer and the other after the first inner-product layer.

For performing experiments with the FashionMNIST dataset, we work with the *VGG-7* [14] network architecture. This architecture is composed of four convolutional layers and three inner-product layers at the end. In order to capture the epistemic uncertainty by performing MC dropout with this network, we add two dropout layers - one after the second convolutional

layer and the other after the first inner-product layer. Initially, we used LeNet5 with the FashionMNIST dataset but we did not get a reasonable performance so we shifted to this VGG-7 network architecture.

Finally, for performing experiments with the CIFAR10 dataset, we use the *ResNet50* [15] network architecture. As already mentioned in the subsection 3.2.2, this network architecture is composed of a couple of bottleneck blocks that are formed by a combination of multiple convolutional layers. In addition, this network employs the concept of residual learning which provides protection against the vanishing gradient problem. Consequently, it makes the network robust to the over-fitting problems as well. For the purpose of capturing the epistemic uncertainty by using MC dropout, we add a dropout layer after every bottleneck block in this network. Since, there are four bottleneck blocks, we add a total of four dropout layers.

In this subsection, we gave details of the network architectures that we use with the different datasets. We mentioned the particular locations of the dropout layers within our networks for performing MC dropout. However, we did not comment on the strength of the dropout probability. Thus, in the next section we will give details of the experimental setup that we use in the context of the Image classification experiments. While giving these details, we would comment on the strength of the dropout probability as well.

3.3.3 Experimental setup

For performing the set of initial experiments, mentioned in the section 3.3, with the MNIST dataset and later conducting the best performing experiments with the other datasets, we would train only one network configuration i.e. Aleatoric + Epistemic. We choose this particular configuration since we are interested in computing the predictive uncertainty, mutual information as well as the aleatoric uncertainty. For capturing the epistemic uncertainty, we add the dropout layers at different locations in our networks. Following the author's empirical evidence in [4], we choose a dropout strength of 0.5 for all the dropout layers in every network. Hence, based on the above comments, the network configuration which we use in all of our experiments is:

- Aleatoric + Epistemic (p = 0.5).

We train all of our networks with the loss function given in the equation 3.8. However, as mentioned in the sub-subsection 3.3.1.1, we train some of our networks with the loss function given in the equation 3.15 as well. We will explicitly mention the experimental configurations where we use this loss function in the chapter where we provide our empirical observations.

As we mentioned in the section 3.3, we need to include the OOD examples in our test-data stream. For that purpose, with the MNIST dataset, we use the FashionMNIST test-set as the OOD data. In a similar way, in the case of the FashionMNIST dataset, we use the MNIST test-set as the OOD data. Finally, for the CIFAR10 dataset, we use the validation-set of the TinyImageNet [72] dataset as the OOD data.

For measuring the predictive uncertainty, we use the equation 3.13 whereas for computing the mutual information we use the equation 3.14. For the aleatoric uncertainty, we just take the average of our aleatoric variances vector. In some of our initial experiments, we noticed that the average of the aleatoric variances vector represents the aleatoric uncertainty better as compared to picking up the value corresponding to the class label index. Additionally, in all

of our experiments in the Image classification setting, we do not care about the accuracy of our network since our goal here is to work around with the different patterns in the uncertainty estimates for accomplishing meaningful tasks.

Apart from that, for all the different experimental configurations that we try with our datasets, we compute the AUROC metric and analyze the power of different uncertainty estimates for some tasks highlighted in the subsection 3.3.1. In addition, we demonstrate the working of our new classifier with the best performing experimental configurations for every dataset. For training our classifier to partition the ID samples into different categories, after separating the ID data from the OOD data, we assign a class label of 0 to the high aleatoric data samples features, class label of 1 to the correctly classified samples features and a class label of 2 for the misclassified samples features. For analyzing the performance of our classifier, we compute the confusion matrices. Before ending this subsection, let us give short remark on the notation which we use for labelling the entries in the confusion matrices. We use the word **HighALEA** for the high aleatoric data samples, **CORRECT** for the correctly classified samples and **MISCLASSIFICATION** for the misclassified samples.

In this chapter, we gave details of a framework which can be used to compute both the epistemic and the aleatoric uncertainty at the same time. Afterwards, we formulated some research questions in the context of the Semantic segmentation and the Image classification settings. Additionally, we explained our approach towards answering these questions which enabled us to derive the experimental setups that we utilize for giving our empirical evidence and consequently providing the answers to our questions. Now, in the next chapter, we would show the results of our experiments. Further, we would give our comments on these results and relate them with the research questions that we have already formulated.

4 Experimental Observations and Analysis with Semantic Segmentation

In this chapter[1], we would show the results of our experiments that we conduct in the Semantic segmentation setting. As mentioned in the subsection 3.2.2, we use two different network architectures, named as PSPNet and DeepLabv3+, with the Cityscapes dataset, in our experiments. Thus, we would present the results with both of these network architectures separately. Additionally, we would follow the experimental setup explained in the subsection 3.2.3 for providing different results. However, before presenting any empirical results, we would provide the training settings that we use for our different network architectures.

Comments on the dataset: As mentioned above, we use the Cityscapes dataset in our Semantic segmentation experiments. This dataset consists of different images representing outdoor street scenes that have been captured by mounting a camera on a car. In total, we have 5000 finely-annotated (19 classes) images, with a spatial resolution of 1024x2048 that are divided into the training and the validation data. In our experiments, we use the validation data for monitering the training process on every epoch[2]. Further, since we do not have any finely-annotated testing images, we use the same validation data as the testing data too.

4.1 Results obtained from PSPNet

In this section, we will show the results that we achieved for our Semantic segmentation experiments with the PSPNet. For training different configurations of this network, we use the deep learning framework called *PyTorch* [73] with the following settings:

- Batch size (both training and validation): 8.
- Training epochs: 230.
- Optimizer: Stochastic gradient descent with its momentum set equal to 0.9 and the L2 weight regularization strength set equal to 0.0005.

[1] For getting an idea of the notation which we use while demonstrating the results of our experiments here, please look at the subsection 3.2.3.

[2] In deep learning, every training iteration is referred to as an *epoch*.

- Learning rate: We start with an initial learning of 0.01 and proceed with the polynomial learning rate decay policy as used by the authors in [62]. In addition, we multiply the learning rate of the PPM module in this network by 10 in the same way as stated in the original formulation of the PSPNet in [8].
- Number of samples extracted from the logit space Gaussian distribution: 50
- Weights initialization: We use the default initialization techniques used by the different layers in PyTorch.
- Training data augmentations: Firstly, we randomly choose a scale in the range of 0.5 - 2.0 for every image and scale their spatial dimensions equally. Afterwards, we extract a random crop of the size 512x512 from these scaled images. Following that, we standardize these random crops by subtracting the mean and subsequently dividing by the standard deviation[3]. Finally, for some of the randomly selected standardized crops, we apply rotations in the range of -10° to 0° and for the other randomly selected sample, we apply the horizontal flipping operation.
- Validation data augmentations: Firstly, we extract center crops of the size 713x713 from the validation data images. After that, we just standardize these crops by subtracting the mean and subsequently dividing by the standard deviation. We use the value of the mean as well as the standard deviation obtained from the training data images.

At test-time, we use the following set of settings for obtaining our results:

- Batch size: 4.
- Number of stochastic forward passes for MC dropout: 30.
- Testing data augmentations: We use the full size validation data images at the test-time. However, before feeding these images to our network, we standardize them by subtracting the mean and subsequently dividing by the standard deviation. We use the value of the mean as well as the standard deviation obtained from the training data images.

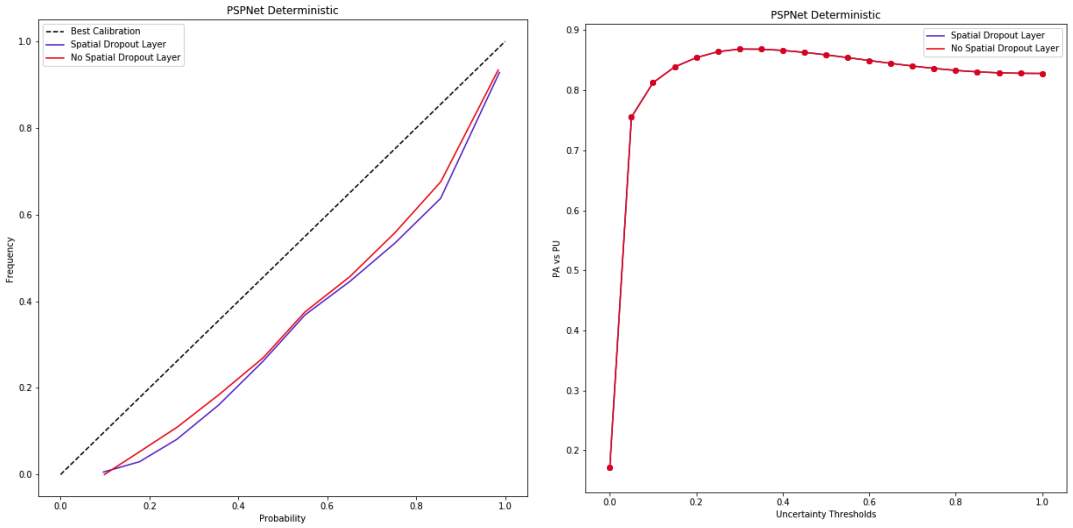
4.1.1 Deterministic network configuration

In this subsection, we show our results that we obtain with the deterministic configuration of the PSPNet. As mentioned in the subsection 3.2.2, PSPNet has a Dropout2D layer in its final block. Thus, we perform experiments with and without this layer. Remember, this deterministic configuration does not have any dropout layers and we use it as a baseline for our other configurations. However, we still compute the epistemic uncertainty by calculating the entropy of the probability vector for every pixel, obtained with a single forward pass through the network. Finally, we train this deterministic configuration with the standard cross-entropy loss.

[3]Before applying any augmentations, we compute the mean as well as the standard deviation of all the training data images pixels and use these values in the standardization process.

Network configuration	mIoU	ECE	MCE	PAvsPU (epistemic)
Deterministic (Spatial Dropout)	69.8	0.091	0.041	0.817
Deterministic (No Spatial Dropout)	67.1	0.082	0.038	0.807

Table 4.1: Results obtained with the deterministic configuration.



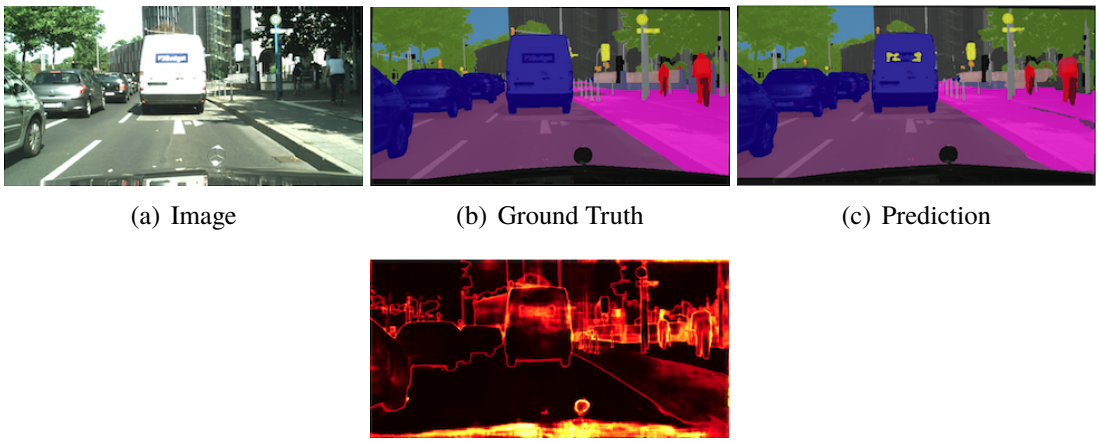
(a) Calibration plot

(b) Plot of the PAvsPU (epistemic) metric against the uncertainty thresholds

Figure 4.1: Plots for the uncertainty quality metrics obtained with the deterministic network configuration.

In the table 4.1, we can observe that the performance of our network is better with the Dropout2D layer (in its final block) in terms of the mIoU metric. However, in terms of the quality of the epistemic uncertainty estimate, we have a little clash between the ECE, MCE and the PAvsPU metric. According to the ECE and the MCE metric, the quality of the estimate produced by the network without the Dropout2D layer seems to be better however, in terms of the PAvsPU metric, the quality of the estimate produced by the network with the Dropout2D layer appears to be better. But, the interesting thing is that the PAvsPU (epistemic) metric does correlate with the mIoU metric which is one of the things that we want our uncertainty quality estimation method to have. The reason behing this correlation is the way in which we compute this metric where we keep an eye on both the prediction as well as the uncertainty estimation capabilities of our network simaltaneously. Figure 4.1(a) shows a graphical interpretation of the ECE and the MCE metric in form of a calibration plot. As evident, it is difficult to get an idea of the uncertainty quality from this plot. However, it can provide some intuitions on the confidence calibration of our network. We can observe that for the higher probability averages (in the bins), our network without the Dropout2D layer seems to be over-confident as compared to the network with the Dropout2D layer.

In the figure 4.1(b) we have a plot of the PAvsPU metric against the different uncertainty thresholds. Both of the network configurations produce almost the same plot with a very minute difference which is visible in the numerical values. However, we still do not derive any strong



(a) Image (b) Ground Truth (c) Prediction

(d) Epistemic Uncertainty

Figure 4.2: Illustration of the visual results obtained with the deterministic (spatial dropout) configuration.

conclusions from this experiment since we are neither using MC dropout nor the aleatoric loss function.

Figure 4.2 shows the visual results obtained with our deterministic (spatial dropout) configuration. We can see that we get a high epistemic uncertainty for the pixels that have been misclassified by our network.

4.1.2 Aleatoric network configuration

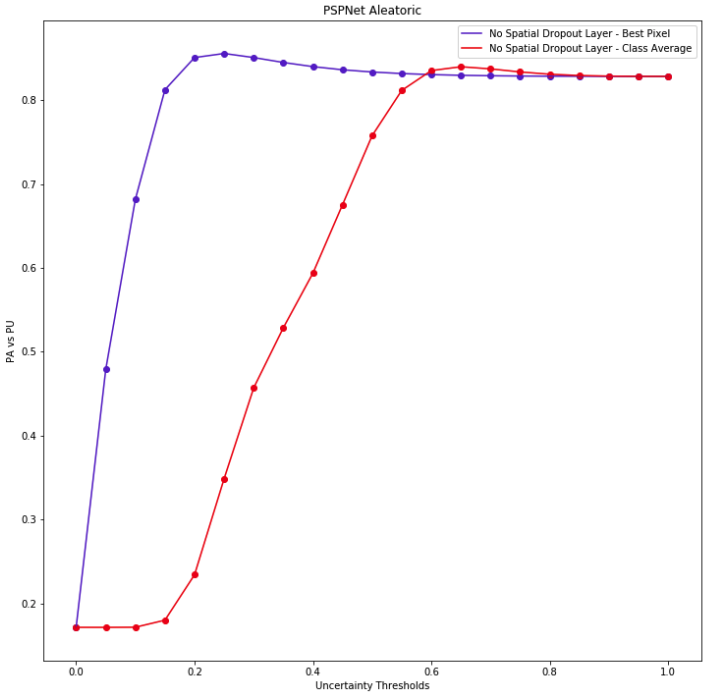
In this subsection, we show the results that we obtain with our aleatoric configuration of the PSPNet. Similar to the deterministic configuration, we give results for the network with the Dropout2D layer and without the Dropout2D layer. We train our networks in the aleatoric configuration with the aleatoric loss function but without any dropout layers. However, as with the deterministic configuration, we still measure the epistemic uncertainty. Along with computing the epistemic uncertainty, we also estimate the aleatoric uncertainty.

As mentioned in the subsection 3.1.1, we want to figure out the best method for summarizing the information contained in an aleatoric variances vector. For this purpose, we evaluate the two possible tricks: 1) Picking up the value in the vector corresponding to the class label index and 2) Taking the average of the aleatoric variances vector. We test these two choices with our Aleatoric (No Spatial Dropout) configuration, choose the best strategy and then adopt it for the rest of our experiments in the aleatoric configuration as well as in the other configurations.

Notation remark: We would use the word **PAvsPU (aleatoric - best pixel)** for representing the PAvsPU (aletoric) metric for our first aleatoric variances vector summarization strategy mentioned above. For denoting the PAvsPU (aleatoric) metric for our second strategy, we would use the word **PAvsPU (aleatoric - class average)**.

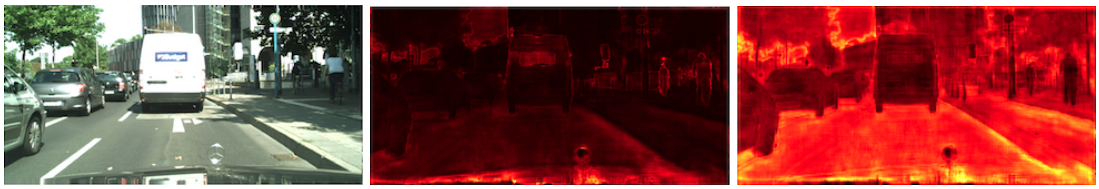
Network configuration	PAvsPU (aleatoric - best pixel)	PAvsPU (aleatoric - class average)
Aleatoric (No Spatial Dropout)	0.652	0.567

Table 4.2: Values of the PAvsPU (aleatoric) metric obtained with two different ways of summarizing the information contained in an aleatoric variances vector.



(a) PAvsPU (aleatoric) plot against the different uncertainty thresholds

Figure 4.3: Plot of the PAvsPU (aleatoric) metric against various uncertainty thresholds for our two different aleatoric variances vector summarization strategies.



(a) Image (b) Aleatoric Uncertainty (Best Pixel) (c) Aleatoric Uncertainty (Class Average)

Figure 4.4: Illustration of the visual results for the aleatoric uncertainty map obtained with two different strategies of summarizing the information contained in the aleatoric variances vector for every pixel of an image.

In the table 4.2, we can observe that the PAvsPU (aleatoric) metric has a higher value when we use the trick where we pick the value in the aleatoric variances vector corresponding to the class label index. This observation is also visible in the graph given in the figure 4.3. For

having an idea of the visual representation of the aleatoric uncertainty maps that we get with our two different aleatoric variances vector summarization strategies, we can look at the figure 4.4. It is clear that the aleatoric uncertainty map obtained with our best performing strategy is less noisy as compared to the one obtained from the other strategy. Relying on these observations, we pick this best performing trick for quantifying the aleatoric uncertainty in all of our experimental configurations.

Important result: As we have seen above, when we quantify the aleatoric uncertainty by picking the value in the aleatoric variances vector corresponding to the class label index, we get the best result. Thus, whenever we write PAvsPU (aleatoric) now, in any experimental configuration, it would essentially correspond to the PAvsPU (aleatoric - best pixel) metric. In addition, the visual results for the aleatoric uncertainty would represent the estimates obtained by this quantification process as well.

Network configuration	mIoU	ECE	MCE	PAvsPU (epistemic)	PAvsPU (aleatoric)
Aleatoric (Spatial Dropout)	64.4	0.090	0.044	0.824	0.684
Aleatoric (No Spatial Dropout)	68.6	0.080	0.036	0.817	0.652

Table 4.3: Results obtained with the aleatoric configuration.

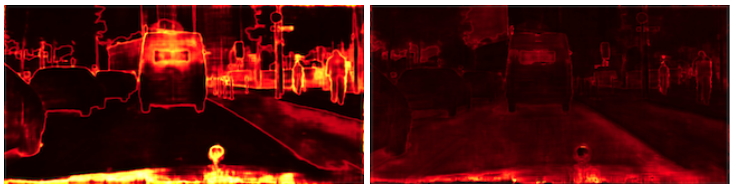
Moving on to the performance of the networks in this configuration, in the table 4.3, we can see that we get a huge improvement over the mIoU metric for the aleatoric configuration without the Dropout2D layer as compared to the one with the Dropout2D layer. If we compare these results with the deteministic configuration (which is our baseline), we can observe that using the aleatoric loss function with our network containing the Dropout2D layer severely disturbs the performance. However, when we use the same loss function with the network, which does not contain this Dropout2D layer, we see a very decent performance improvement over the baseline. Hence, these empirical observations suggest that, it is better to use the aleatoric loss with the networks that do not have the Dropout2D layer. Figure 4.5 shows the visual results obtained with our aleatoric (no spatial dropout) configuration.

Considering the uncertainty estimates quality metrics[4], our values for the ECE and the MCE metrics seem to correlate with the performance (mIoU) of our networks in the aleatoric configuration. However, the PAvsPU (epistemic) and the PAvsPU (aleatoric) metrics do not seem to follow the same correlation. For the aleatoric (no spatial dropout) configuration, where we have a higher value for the mIoU metric, we do not observe a similar trend for both of the PAvsPU metrics. However, the difference between the mIoU metric still overrules the difference between the PAvsPU metrics since it is not desirable to have a disastrous effect on the prediction capibilites of a network while making it capable of producing the different uncertainty estimates.

Summarizing the observations stated in the above paragraphs, we can conclude that it is better to use the loss function, given in the equation 3.8 (aleatoric loss), with the networks that do

[4]Note, in order to save up the space, we only give the numerical results for our uncertainty quality metrics. We show the graphical result when we figure out the best trick for summarizing the information contained in an aleatoric variances vector.

(a) Image (b) Ground Truth (c) Prediction



(d) Epistemic Uncertainty (e) Aleatoric Uncertainty

Figure 4.5: llustration of the visual results obtained with the aleatoric (no spatial dropout) configuration.

not have the Dropout2D layer. In addition, it seems like there is a trade-off between the performance of the network and the quality of the uncertainty estimates obtained, if we consider the PAvsPU metric to be giving the reliable quality results. In the rest of our experiments with the different network configurations, we will observe whether we get a similar pattern of results or not.

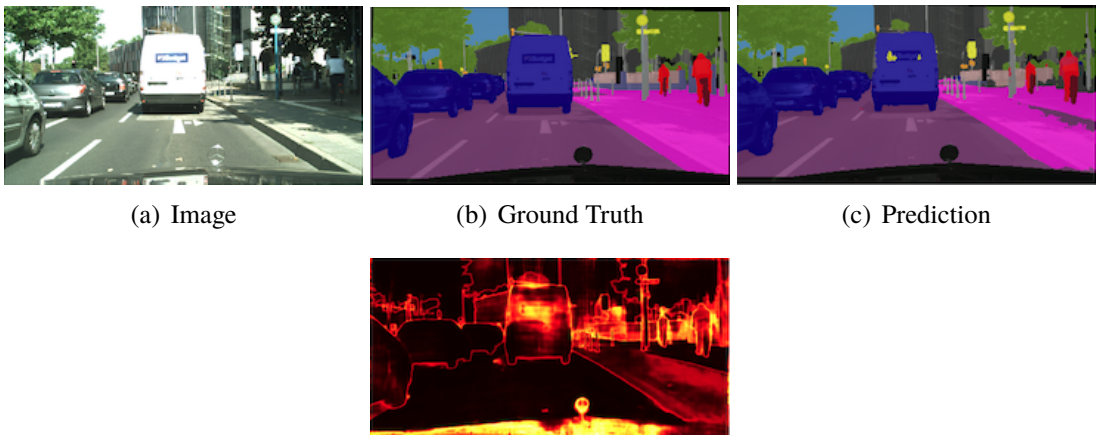
4.1.3 Epistemic network configuration

In this section, we demonstrate the results that we obtain with our epistemic network configuration. For the networks in this configuration, we add the dropout layers, as mentioned in the subsection 3.2.2, with two different dropout probabilities: p = 0.2 and p = 0.5. We obtain the results by performing the MC dropout (short form: MC) as well by using the dropout in a standard way (short form: Normal) as it is used in the literature. In addition, we perform the experiments with the networks containing the Dropout2D layer and the same networks without this layer for every value of the dropout probability. For training all the networks in this experimental configuration, we use the standard cross-entropy loss.

Network configuration	mIoU	ECE	MCE	PAvsPU (epistemic)
Epistemic (p = 0.2 + Spatial Dropout + Normal)	58.3	0.062	0.014	0.811
Epistemic (p = 0.2 + Spatial Dropout + MC)	58.0	0.055	0.011	0.811
Epistemic (p = 0.2 + No Spatial Dropout + Normal)	67.4	0.082	0.035	0.824
Epistemic (p = 0.2 + No Spatial Dropout + MC)	67.5	0.077	0.032	0.822
Epistemic (p = 0.5 + Spatial Dropout + Normal)	60.8	0.070	0.027	0.784
Epistemic (p = 0.5 + Spatial Dropout + MC)	61.2	0.053	0.016	0.777
Epistemic (p = 0.5 + No Spatial Dropout + Normal)	57.5	0.093	0.042	0.815
Epistemic (p = 0.5 + No Spatial Dropout + MC)	57.1	0.081	0.033	0.807

Table 4.4: Results obtained with the epistemic configuration using different dropout probabilities.

In the table 4.4, for the dropout probability p = 0.2, we can see that, for both the Standard and the MC dropout, our network with the Dropout2D layer has a big performance drop as compared to the baseline. In addition, the performance drop with the MC dropout is more as compared to the Standard dropout. However, for the network without the Dropout2D layer, our performance is better, for both the MC and the Standard dropout, as compared to the same deterministic baseline network. The performance increase is more with the MC dropout as compared to the Standard dropout. Talking about the quality of the epistemic uncertainty estimate, we can observe that the ECE and the MCE metrics have a lower value for the network with the Dropout2D layer but they do not correlate with the mIoU metrics obtained with the Standard and MC dropout respectively. However, for the network without the Dropout2D layer, our ECE and MCE metrics do correlate with its corresponding mIoU metrics. In terms of the PAvsPU (epistemic) metric, we get the highest value with the network which does not contain the Dropout2D layer and uses the Standard Dropout. However, since the difference is not that great as compared to the PAvsPU (epistemic) metric obtained with the MC dropout, using the same network, we consider that the network which does not have the Dropout2D layer and uses MC dropout with probability p = 0.2, gives the best performance and the epistemic uncertainty quality. Figure 4.6 shows the visual results obtained with the epistemic (p = 0.2 + no spatial dropout + MC) configuration.

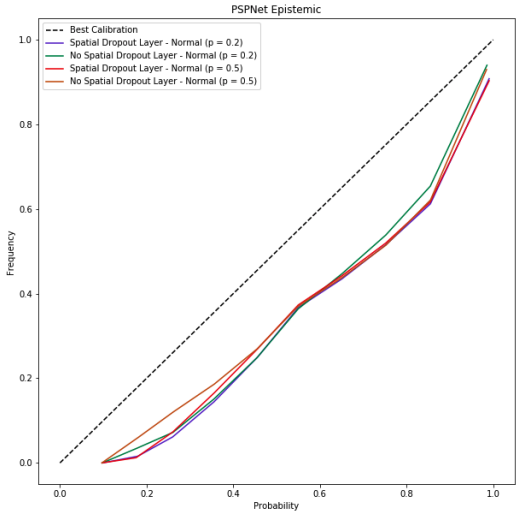


(a) Image (b) Ground Truth (c) Prediction

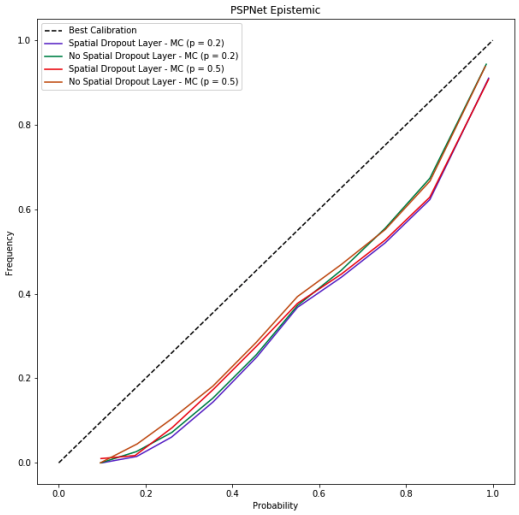
(d) Epistemic Uncertainty

Figure 4.6: llustration of the visual results obtained with the epistemic (p = 0.2 + no spatial dropout + MC) configuration.

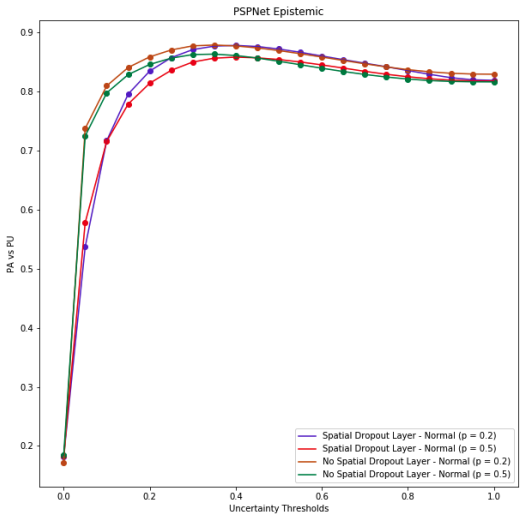
Having a look at the results obtained with the dropout probability p = 0.5, we can observe that the network without the Dropout2D layers suffers a huge performance loss as compared to the network with the Dropout2D layer. We observe better results by using MC dropout, with the network containing the Dropout2D layer, as compared to the Standard dropout. However, if we take a look on the uncertainty quality, we can see that the values of the ECE and the MCE metrics are higher for the network without the Dropout2D layer as compared to the network with this layer, which is actually the same pattern that we observe with the dropout probability p = 0.2. In terms of the correlation of these metrics with the mIoU metric, for the dropout probability p = 0.5 we do observe it, for the Standard as well as the MC dropout, with both of the networks. However, as evident, the PAvsPU (epistemic) metric shows the correlation



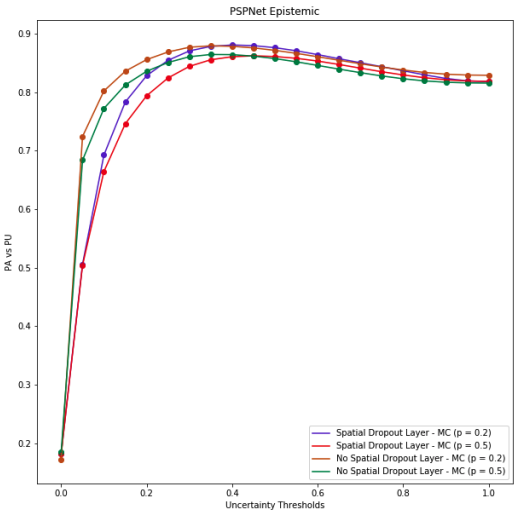
(a) Calibration plot obtained with the Standard dropout using different probabilities



(b) Calibration plot obtained with the MC dropout using different probabilities



(c) Plot of the PAvsPU (epistemic) metric against different uncertainty thresholds obtained with the Standard dropout using different probabilities



(d) Plot of the PAvsPU (epistemic) metric against different uncertainty thresholds obtained with the MC dropout using different probabilities

Figure 4.7: Plot for the uncertainty quality metrics obtained with the epistemic network configuration using different dropout probabilities.

for both of the dropout probabilities p = 0.2 and p = 0.5 used by our different networks in either MC dropout or the Standard dropout style. Figure 4.7 shows the calibration and the PAvsPU (epistemic) (against different uncertainty thresholds) plots for the networks with and without the Dropout2D layer using either MC or Standard dropout. It is a bit difficult to analyze the calibrations plot however, looking at the PAvsPU (epistemic) plots, we can observe that the networks which does not have the Dropout2D layer produces a better quality uncertainty estimate.

Summarizing the above mentioned observations, we can conclude that the network which does not have the Dropout2D layer and uses MC dropout with a dropout probability p = 0.2 gives us the best performance in terms of the mIoU metric and a better quality epistemic uncertainty estimate. Although, as compared to the baseline, the performance increase is not that high as compared to what we achieved with the aleatoric (no spatial dropout) configuration, we get a better quality epistemic uncertainty estimate. Additionally, up until this point, we can conclude that, computing the epistemic uncertainty using MC dropout does result in some performance improvements but this is applicable only with the network which does not have Dropout2D layer in its final block. Looking back at the subsection 3.1.1, it is actually one of the things that we wanted to explore.

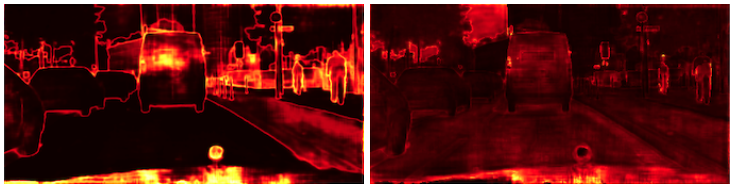
4.1.4 Aleatoric + Epistemic network configuration

In this subsection, we show the results that we obtain with our Aleatoric + Epistemic network configuration. For the networks in this configuration, we add the dropout layers, as mentioned in the subsection 3.2.2, with two different dropout probabilities: p = 0.2 and p = 0.5. We obtain the results by performing the MC dropout (short form: MC) as well by using the dropout in a standard way (short form: Normal) as it is used in the literature. In addition, we perform the experiments with the networks containing the Dropout2D layer and the same networks without this layer for every value of the dropout probability. For training all the networks in this experimental configuration, we use the aleatoric loss function.

In the table 4.5, for the dropout probability p = 0.2, we can see that, for both the Standard and the MC dropout, our network with the Dropout2D layer has a big performance drop as compared to the baseline. In addition, the performance drop with the MC dropout is more as compared to the Standard dropout. However, for the network without the Dropout2D layer, our performance is better, for both the Standard and the MC dropout, as compared to the same deterministic baseline network. The performance increase is more with the Standard dropout as compared to the MC dropout. Talking about the quality of the epistemic uncertainty estimate, we can observe that the ECE and the MCE metrics have a lower value for the network with the Dropout2D layer but they do not correlate with the mIoU metrics obtained with the Standard and MC dropout respectively. The same argument holds for the network without the Dropout2D layer using either the MC or the Standard dropout. In terms of the PAvsPU (epistemic) metric, we see a minimal correlation with the mIoU metric for both of the networks using either the MC or the Standard dropout. In addition, we get the highest value with the network which does not contain the Dropout2D layer. For the PAvsPU (aleatoric) metric, we can observe a correlation with the mIoU metric for both of the networks, with and without the Dropout2D layer, using either the Standard or the MC dropout. We see the highest value for this metric with the network containing the Dropout2D layer and using Standard dropout. However, the difference is not that high as compared to the PAvsPU (aleatoric) metric for the network without the Dropout2D layer. Interestingly, for both of the networks, the PAvsPU (aleatoric) quality drops with MC dropout whereas the PAvsPU (epistemic) metric stays the same. This fact demonstrates that the quality of the aleatoric uncertainty might suffer a bit with the MC dropout. From the results in the table 4.5, we can conclude, for the dropout probability p = 0.2, the network without the Dropout2D layer and using Standard dropout gives the best performance as well as the better

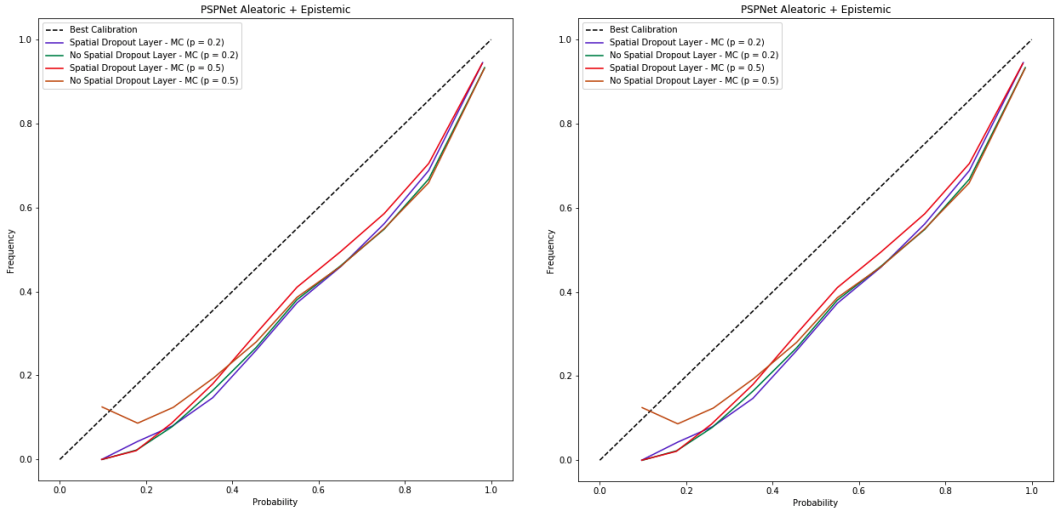
quality uncertainty estimates. Figure 4.8 shows the visual results obtained with the aleatoric + epistemic (p = 0.2 + no spatial dropout + Normal) configuration.

(a) Image (b) Ground Truth (c) Prediction



(d) Epistemic Uncertainty (e) Aleatoric Uncertainty

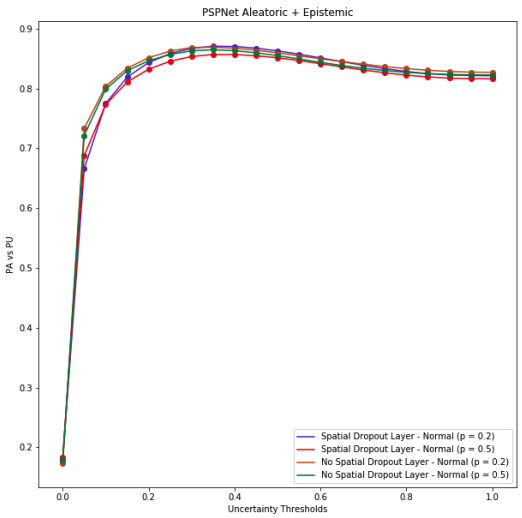
Figure 4.8: llustration of the visual results obtained with the aleatoric + epistemic (p = 0.2 + no spatial dropout + Normal) configuration.



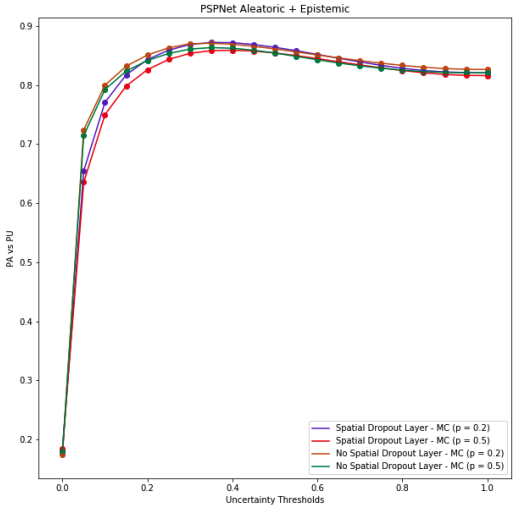
(a) Calibration plot obtained with the Standard dropout using different probabilities (b) Calibration plot obtained with the MC dropout using different probabilities

Figure 4.9: Calibration plots obtained with the aleatoric + epistemic network configuration using different dropout probabilities.

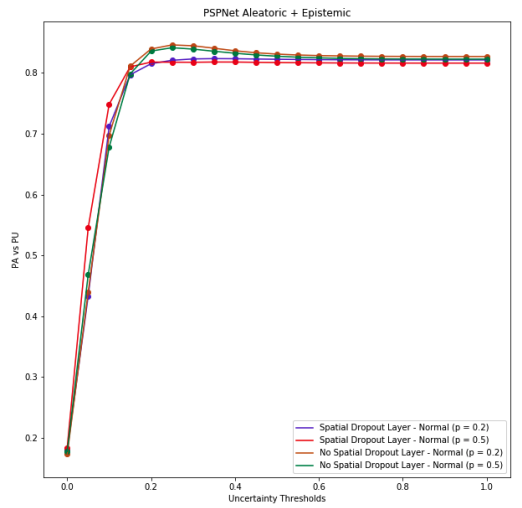
Having a look at the results obtained with the dropout probability p = 0.5, we can observe that both of the networks, with and without the Dropout2D layer, suffer a huge performance loss as compared to their deterministic counterparts. We observe better results by using MC dropout, with the network containing the Dropout2D layer, as compared to the Standard dropout. However, if we take a look on the epistemic uncertainty quality, we can see that the values of the



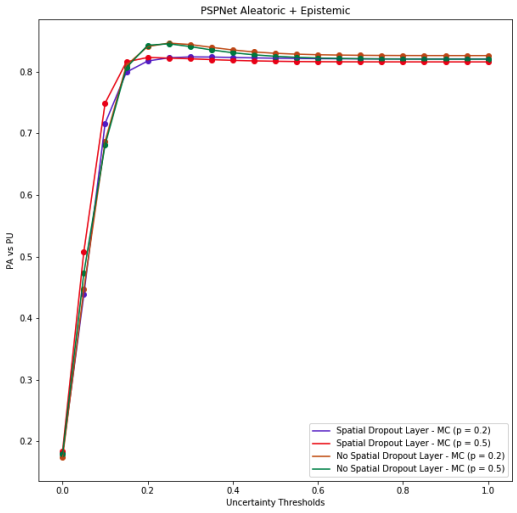
(a) Plot of the PAvsPU (epistemic) metric against different uncertainty thresholds obtained with the Standard dropout using different probabilities



(b) Plot of the PAvsPU (epistemic) metric against different uncertainty thresholds obtained with the MC dropout using different probabilities



(c) Plot of the PAvsPU (aleatoric) metric against different uncertainty thresholds obtained with the Standard dropout using different probabilities



(d) Plot of the PAvsPU (aleatoric) metric against different uncertainty thresholds obtained with the MC dropout using different probabilities

Figure 4.10: Plots for the PAvsPU metrics obtained with the aleatoric + epistemic network configuration using different dropout probabilities.

ECE and the MCE metrics are higher for the network without the Dropout2D layer as compared to the network with this layer, which is actually the same pattern that we observe with the dropout probability p = 0.2. In terms of the correlation of these metrics with the mIoU metric, for the network with the Dropout2D layer, we do observe it for the Standard as well as the MC dropout. However, for the network without the Dropout2D layer, we do not observe such correlation. Looking at the PAvsPU (epistemic) metric, we do observe some correlation (with the mIoU metric) for both of the networks in either MC dropout or the Standard dropout

style. For the PAvsPU (aleatoric) metric, we see the same correlation as what we observe with the PAvsPU (epistemic) metric. However, the interesting thing is that by using MC dropout with either of the networks, we see either a very small change or no change at all with the PAvsPU (epistemic) metric but for the PAvsPU (aleatoric) metric the change is a bit significant. Figure 4.9 shows the calibration plots and figure 4.10 shows the PAvsPU (epistemic) as well as the PAvsPU (aleatoric) (against different uncertainty thresholds) plots for the networks with and without the Dropout2D layer using either MC or Standard dropout. It is a bit difficult to analyze the calibrations plot however, looking at the PAvsPU plots, we can observe that the networks which does not have the Dropout2D layer produces better quality uncertainty estimates.

Summarizing the above mentioned observations, we can conclude that the network which does not have the Dropout2D layer and uses Standard dropout with a dropout probability p = 0.2 gives us the best performance in terms of the mIoU metric as well as better quality uncertainty estimates. With the same network, if we use the MC dropout the mIoU metric as well as the PAvsPU (aleatoric) metric drops a bit but the PAvsPU (epistemic) metric stays constant. Although, as compared to the baseline, the performance increase is not that high as compared to what we achieved with the aleatoric (no spatial dropout) configuration, we get a better quality for both the epistemic and the aleatoric uncertainty estimate. We can also notice this quality difference in the visual representation, especially of the aleatoric uncertainty, obtained with aleatoric (no spatial dropout) configuration (figure 4.5) and the one mentioned above (figure 4.8). Apart from that, we can conclude that computing the epistemic as well as the aleatoric uncertainty using Standard dropout results in some performance improvements. With the MC dropout, we see a performance gain over the baseline as well but it is lesser than the one achieved by using the Standard dropout. However, the important thing to note is that, this is applicable only for the network which does not have Dropout2D layer in its final block. Looking back at the subsection 3.1.1, the question, focusing on whether the performance of a network improves with computing both the epistemic as well as the aleatoric uncertainty, was actually one of the things that we wanted to explore.

4.1.5 Concluding remarks

In the above subsections, we demonstrated our results with the PSPNet in different experimental configurations. The main objective of conducting multiple experiments was to answer the research questions that we formulated in the subsection 3.1.1 for the Semantic segementation task. Looking back at those questions again, firstly we wanted to observe whether the performance of a network improve with the epistemic uncertainty computation using MC dropout or not. From our epistemic network configuration experiments, we figured out that the performance (in terms of the mIoU metric) indeed improves over the baseline deterministic network when we add the dropout layers in the middle flow of the PSPNet architecture, for capturing the epistemic uncertainty. However, there are a few conditions for this claim to be valid, specifically for the PSPNet. We should not have a Dropout2D layer in the final block of the network and we should not use a higher dropout probability like p = 0.5 - the claim which directly answers our second question related to the dropout probability strength as well. As demonstrated in the subsection 4.1.3, we get the best performance as well as the epistemic uncertainty quality when we use MC dropout with the network configuration which does not have the Dropout2D

layer in its final block. Importantly, we saw this result when we used a dropout probability p = 0.2.

Moving on, we wanted to filter out the best way for summarizing the information contained in an aleatoric variances vector as well observe whether the performance of a network improves when we make it capable of estimating the aleatoric uncertainty. In this regard, our aleatoric network configuration experiments in the subsection 4.1.2 proved that the best way of summarizing the information in an aleatoric variances vector is to pick up the value in the vector corresponding to the class label index. Additionally, we also observed a performance gain over the baseline deterministic network but without the Dropout2D layer in the final block of the network in this configuration.

Apart from that, we wanted to observe whether the computation of both the aleatoric and the epistemic uncertainty results in an performance improvement or not. From our aleatoric + epistemic network configuration results, demonstrated in the subsection 4.1.4, we showed that there is indeed a performance gain when we make the network capable of computing both the epistemic and the aleatoric uncertainty. However, there are some conditions for these arguments to be valid, specifically for the PSPNet. Firstly, there should not be the Dropout2D layer in the final block of the network and, secondly, we should not use a high value of the dropout probability such as p = 0.5. In addition, apart from the performance gain, we also observed a better epistemic as well as the aleatoric uncertainty quality estimate collectively as compared to the other network configurations which either compute the epistemic or the aleatoric uncertainty. Interestingly, we observed that the Standard dropout performs better in this configuration as compared to the MC dropout.

In all of the experiments above, we computed some metrics for quantifying the uncertainty estimate quality. We used a new metric, named as PAvsPU, apart from the other standard metrics commonly used in the literature. However, we wanted to observe whether the PAvsPU quality metric correlate with the performance of a network or not. In that regard, we discovered that this metric gives better quality estimates as compared to the standard metrics. Additionally, this metric is also capable of quantifying the aleatoric uncertainty quality as compared to the standard metrics which can only help with analyzing the epistemic uncertainty estimate quality. However, there is one interesting conclusion which we derived about the PAvsPU metric, by looking at the results of all the experimental configurations. For the PAvsPU (epistemic) metric, if we add the dropout layers in the middle flow of the network then regardless of using the Standard or MC dropout with either the network with or without the Dropout2D layer, the correlation with the performance metric almost follows the same pattern as what we observe without the dropout layers. However, for the PAvsPU (aleatoric) metric, the correlation pattern improves with the addition of the dropout layers. This can be confirmed by comparing the results in the subsection 4.1.2 and 4.1.4.

In summary, we conclude that it is always better to compute both the epistemic as well as the aleatoric uncertainty from a network for boosting its performance. Additionally, for the Semantic segmentation task, it is favourable to use a dropout probability p = 0.2 for observing the best results. Further, if we are estimating both the epistemic as well as the aleatoric uncertainty, the Standard dropout can provide better performance gains as compared to the MC dropout. For the PSPNet, since there is an additional complication of the Dropout2D layer, we can rely on these observation but still we feel it necessary to verify these claims with another network. For that purpose, we will present the results that we obtain with our other network

architecture called DeepLabv3+, in the next section.

Network configuration	mIoU	ECE	MCE	PAvsPU (epistemic)	PAvsPU (aleatoric)
Aleatoric + Epistemic (p = 0.2 + Spatial Dropout + Normal)	59.2	0.079	0.031	0.811	0.698
Aleatoric + Epistemic (p = 0.2 + Spatial Dropout + MC)	59.1	0.075	0.026	0.812	0.696
Aleatoric + Epistemic (p = 0.2 + No Spatial Dropout + Normal)	68.0	0.089	0.045	0.813	0.691
Aleatoric + Epistemic (p = 0.2 + No Spatial Dropout + MC)	67.7	0.086	0.040	0.813	0.679
Aleatoric + Epistemic (p = 0.5 + Spatial Dropout + Normal)	57.8	0.088	0.040	0.800	0.701
Aleatoric + Epistemic (p = 0.5 + Spatial Dropout + MC)	57.9	0.071	0.026	0.797	0.715
Aleatoric + Epistemic (p = 0.5 + No Spatial Dropout + Normal)	62.6	0.091	0.044	0.813	0.691
Aleatoric + Epistemic (p = 0.5 + No Spatial Dropout + MC)	62.3	0.086	0.041	0.809	0.685

Table 4.5: Results obtained with the aleatoric + epistemic configuration using different dropout probabilities.

4.2 Results obtained from DeepLabv3+

In this section, we will show the results that we achieved for our Semantic segmentation experiments on the Cityscapes dataset with the DeepLabv3+ network architure. For training different configurations of this network, we use the deep learning framework called *PyTorch* [73] with the following settings:

- Batch size (both training and validation): 8.
- Training epochs: 230.
- Optimizer: Stochastic gradient descent with its momentum set equal to 0.9 and the L2 weight regularization strength set equal to 0.0005.
- Learning rate: We start with an initial learning of 0.01 and proceed with the polynomial learning rate decay policy as used by the authors in [62]. In addition, we multiply the learning rate of the ASPP module in this network by 10 in the same way as stated in the original formulation of the DeepLabv3+ in [9].
- Number of samples extracted from the logit space Gaussian distribution: 50
- Weights initialization: We use the default initialization techniques used by the different layers in PyTorch.
- Training data augmentations: Firstly, we randomly choose a scale in the range of 0.5 - 2.0 for every image and scale their spatial dimensions equally. Afterwards, we extract a random crop of the size 512x512 from these scaled images. Following that, we standardize these random crops by subtracting the mean and subsequently dividing by the standard deviation[5]. Finally, for some of the randomly selected standardized crops, we apply rotations in the range of -10° to 0° and for the other randomly selected sample, we apply the horizontal flipping operation.
- Validation data augmentations: Firstly, we extract center crops of the size 713x713 from the validation data images. After that, we just standardize these crops by subtracting the mean and subsequently dividing by the standard deviation. We use the value of the mean as well as the standard deviation obtained from the training data images.

At test-time, we use the following set of settings for obtaining our results:

- Batch size: 4.
- Number of stochastic forward passes for MC dropout: 30.
- Testing data augmentations: We use the full size validation data images at the test-time. However, before feeding these images to our network, we standardize them by subtracting the mean and subsequently dividing by the standard deviation. We use the value of the mean as well as the standard deviation obtained from the training data images.

[5]Before applying any augmentations, we compute the mean as well as the standard deviation of all the training data images pixels and use these values in the standardization process.

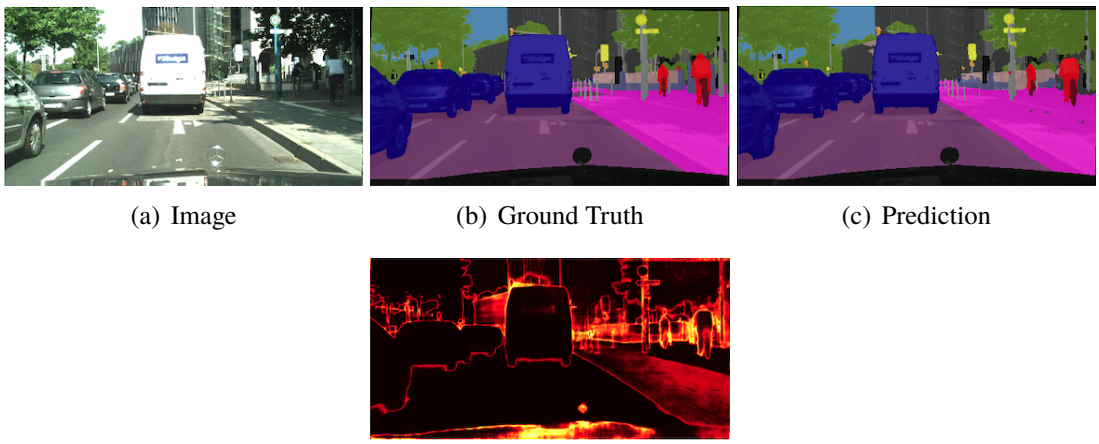
4.2.1 Deterministic network configuration

In this subsection, we demonstrate our results that we obtain with the deterministic configuration of the DeepLabv3+. As mentioned in the subsection 3.2.2, the ASPP module of this network performs separable dilated convolutions. So, out of curiosity, for observing whether this separable convolution has an effect on the performance of our network our not, we perform two different experiments with and without using separable convolutions in the ASPP module. We use the results from this configuration as a baseline for the other network configurations that we train. We do not perform either the MC or the Standard dropout since we do not add any dropout layers. For training, we use the standard cross-entropy loss. As for the PSPNet deterministic configuration, we report the epistemic uncertainty here as well, by simply computing the entropy of the probability vector (obtained with a single forward pass) associated with every pixel of an input image.

Notation remark: In this deterministic configuration, for the network which uses dilated separable convolutions we would use the word **Separable** and the word **No Separable** for the network which does not use it.

Network configuration	mIoU	ECE	MCE	PAvsPU (epistemic)
Deterministic (Separable)	72.5	0.109	0.074	0.811
Deterministic (No Separable)	68.7	0.104	0.064	0.824

Table 4.6: Results obtained with the deterministic configuration.

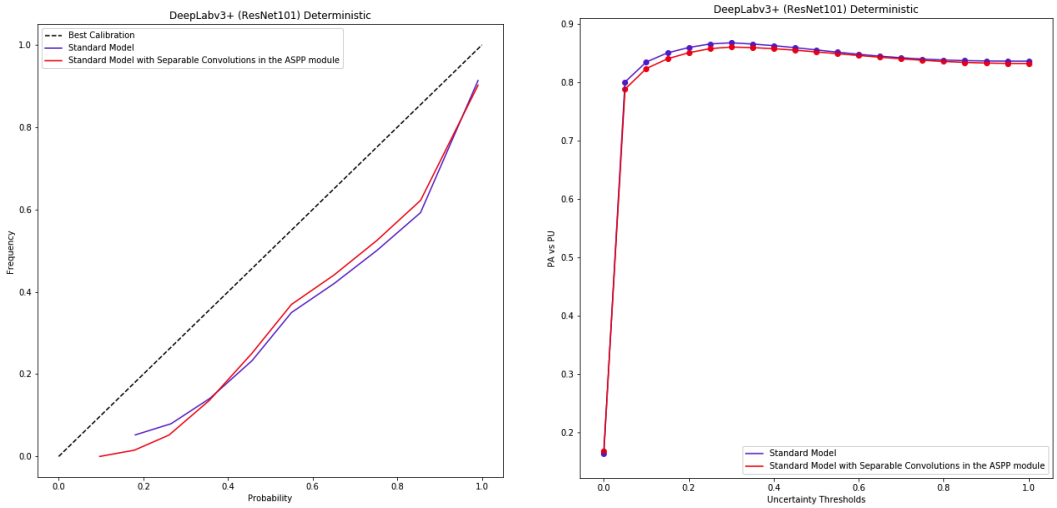


(a) Image (b) Ground Truth (c) Prediction

(d) Epistemic Uncertainty

Figure 4.11: Illustration of the visual results obtained with the deterministic (separable) configuration.

In the table 4.1, for our networks in the deterministic configuration, we demonstrate their performance in terms of the mIoU metric. Additionally, we report the quality metrics for the epistemic uncertainty. As evident, the network using the separable convolutions in the ASPP



(a) Calibration plot

(b) Plot of the PAvsPU (epistemic) metric against the uncertainty thresholds

Figure 4.12: Plots for the uncertainty quality metrics obtained with the deterministic network configuration.

module performs better in terms of the mIoU metric than the network which does not use it. However, the quality of the epistemic uncertainty estimate follows the opposite pattern as visible in the numerical figures in the table as well as in the graphs displayed on figure 4.12. Since, the performance with the separable convolutions is better as well as the difference between the quality metrics, from its other counterpart is not that significant, we continue with using separable convolutions in the ASPP module of the DeepLabv3+ network architecture for the rest of our experimental configurations. Figure 4.11 shows the visual results that we obtain with our deterministic (separable) network.

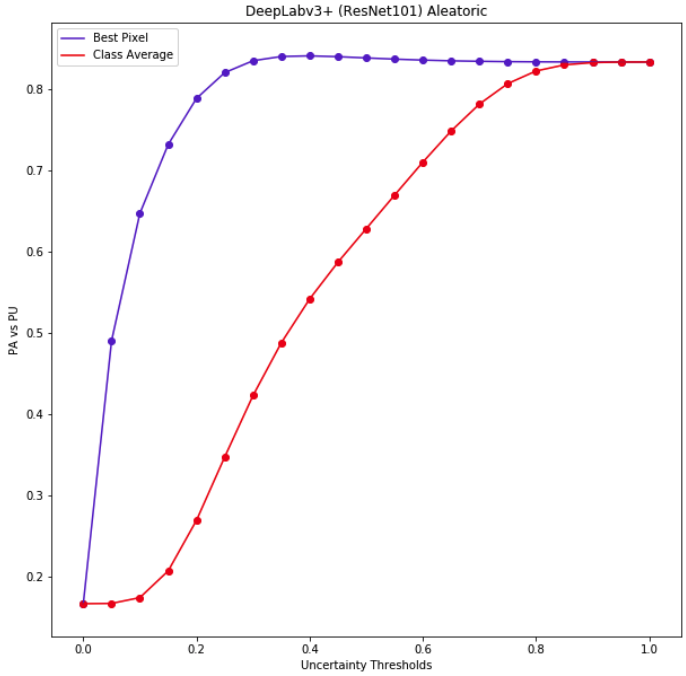
In terms of the correlation between the uncertainty quality metrics and the mIoU metric, we can observe that there is not any. However, we will not rely on these results since we do not have dropout layers in our network. Additionally, we are not using the loss function, given in the equation 3.8 (aleatoric loss), as well.

4.2.2 Aleatoric network configuration

In this subsection, we demonstrate the results that we achieve with our aleatoric network configuration. As with the deterministic networks, we do not add any dropout layers but we use the aleatoric loss function for training purposes. Additionally, in a similar way as done with the PSPNet, we start with figuring out the best way for summarizing the information contained in an aleatoric variances vector. For that purpose, we try two tricks: 1) Picking the value in the vector corresponding to the class label index and 2) Taking the average of the vector. We decide on the best strategy by getting the PAvsPU (aleatoric) metric values and then doing a comparison. After that, we proceed with the best strategy in this configuration as well as in all of the other experimental configurations.

Network configuration	PAvsPU (aleatoric - best pixel)	PAvsPU (aleatoric - class average)
Aleatoric	0.664	0.563

Table 4.7: Values of the PAvsPU (aleatoric) metric obtained with two different ways of summarizing the information contained in an aleatoric variances vector.



(a) PAvsPU (aleatoric) plot against the different uncertainty thresholds

Figure 4.13: Plot of the PAvsPU (aleatoric) metric against various uncertainty thresholds for our two different aleatoric variances vector summarization strategies.

In the table 4.7, we can observe that we get a higher value for the PAvsPU metric, when we summarize the information contained in the aleatoric variances vector by picking up the value in this vector corresponding to the class label index. We can see similar effect in the plot on the figure 4.13 as well. Hence, we would use this trick for our aleatoric variances vector in the rest of our experimental configurations. Figure 4.14 shows the aleatoric uncertainty maps that we obtain for a sample image with both of our summarization strategies. Clearly, the trick where we take an average of the aleatoric variances vector results in a very noisy uncertainty map.

Important remark: For the rest of our experiments, whenever we write **PAvsPU (aleatoric)**, it would essentially mean **PAvsPU (aleatoric - best pixel)**.

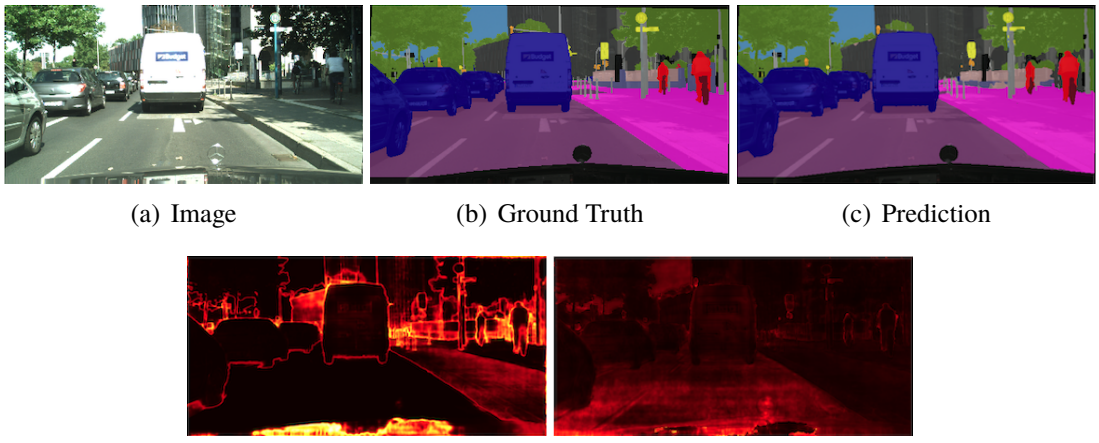
Network configuration	mIoU	ECE	MCE	PAvsPU (epistemic)	PAvsPU (aleatoric)
Aleatoric	72.1	0.103	0.068	0.812	0.665

Table 4.8: Results obtained with the aleatoric configuration.

(a) Image (b) Aleatoric Uncertainty (Best Pixel) (c) Aleatoric Uncertainty (Class Average)

Figure 4.14: Illustration of the visual results for the aleatoric uncertainty map obtained with two different strategies of summarizing the information contained in the aleatoric variances vector for every pixel of an image.

Moving on with the best strategy for summarizing the information in our aleatoric variances vector, we demonstrate the performance of our network in terms of the mIoU metric and the uncertainty quality estimates in the table 4.8. As compared to the deterministic baseline, the mIoU metric shows a bit of a drop whereas the uncertainty quality metrics[6] (excluding PAvsPU (aleatoric)) are almost the same. However, our aleatoric network configuration is additionally capable of providing the aleatoric uncertainty estimate. Figure 4.15 shows the visual results that we obtain this configuration.



(a) Image (b) Ground Truth (c) Prediction

(d) Epistemic Uncertainty (e) Aleatoric Uncertainty

Figure 4.15: Illustration of the visual results obtained with the aleatoric configuration.

[6]Note, in order to save up the space, we only give the numerical results for our uncertainty quality metrics. We show the graphical result when we figure out the best trick for summarizing the information contained in an aleatoric variances vector.

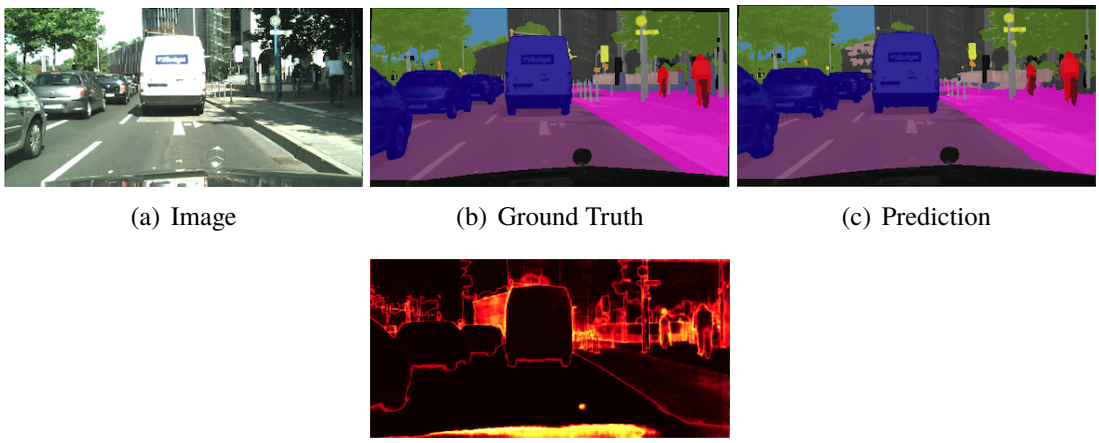
In summary, we can conclude that our peformance might drop a bit if we use the aleatoric loss function instead of the standard cross-entropy loss for training our network. However, if a minor performance drop is not an issue, then we should use the aleatoric loss function since it makes the network capable of producing the aleatoric uncertainty along with the epistemic uncertainty estimate. For the ECE, MCE and PAvsPU (epistemic) metrics, we can say that they improve with the aleatoric loss function but still we will not emphasize greatly on this point until and unless we observe some results with the networks that have dropout layers in their middle flow and perform either Standard or MC dropout for quantifying the epistemic uncertainty.

4.2.3 Epistemic network configuration

In this subsection, we show the results that we obtain with our networks in the epistemic configuration. For the networks in this configuration, we add the dropout layers in the middle flow of the network and conduct experiments with two different dropout probabilities, p = 0.2 and p = 0.5, separately. For obtaining the results, we use the MC dropout (short form: MC) as well as the Standard dropout (short form: Normal). In order to train the networks, we use the standard cross-entropy loss.

Network configuration	mIoU	ECE	MCE	PAvsPU (epistemic)
Epistemic (p = 0.2 + Normal)	72.1	0.106	0.069	0.815
Epistemic (p = 0.2 + MC)	72.1	0.103	0.066	0.815
Epistemic (p = 0.5 + Normal)	69.7	0.109	0.073	0.808
Epistemic (p = 0.5 + MC)	69.6	0.105	0.068	0.806

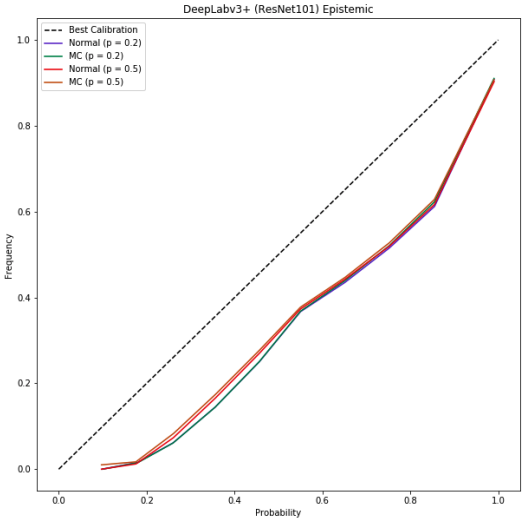
Table 4.9: Results obtained with the epistemic configuration using different dropout probabilities.



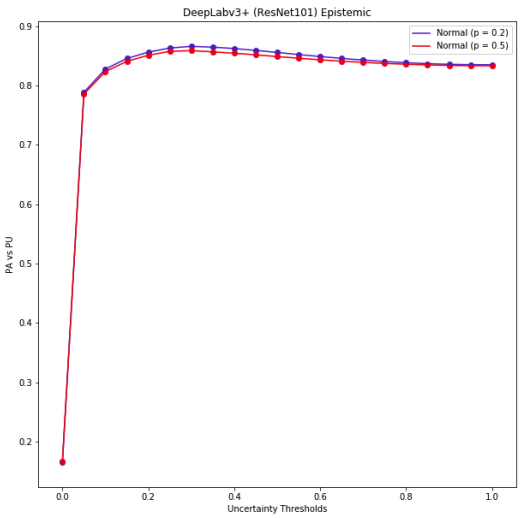
(a) Image (b) Ground Truth (c) Prediction

(d) Epistemic Uncertainty

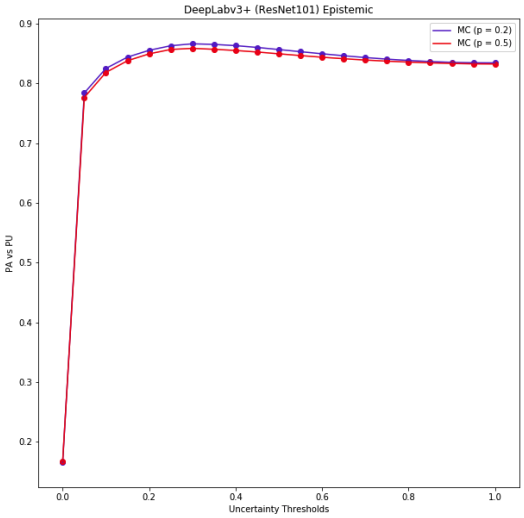
Figure 4.16: Illustration of the visual results obtained with the epistemic (p = 0.2 + MC) configuration.



(a) Calibration plot obtained with the Standard and MC dropout using different probabilities



(b) Plot of the PAvsPU (epistemic) metric against different uncertainty thresholds obtained with the Standard dropout using different probabilities



(c) Plot of the PAvsPU (epistemic) metric against different uncertainty thresholds obtained with the MC dropout using different probabilities

Figure 4.17: Plot for the uncertainty quality metrics obtained with the epistemic network configuration using different dropout probabilities.

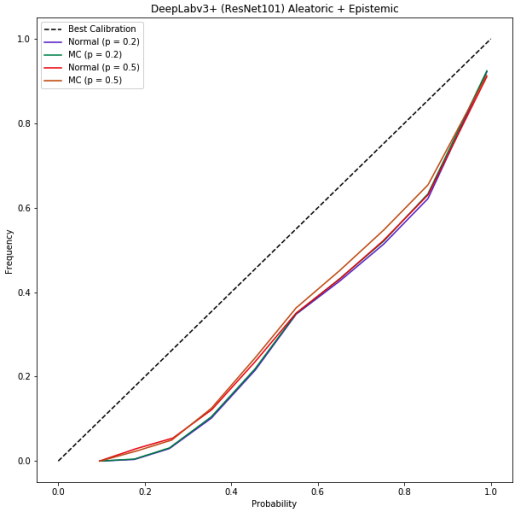
In the table 4.9, we can see the performance of our networks with different dropout probabilities. For the dropout probability p = 0.2, we can observe that the performance stays the same with both the Standard and the MC dropout. However, as compared to the deterministic baseline, there is a bit of a drop in the mIoU metric. In terms of the quality of the epistemic uncertainty estimate, MC dropout produces better ECE and MCE metrics but the same PAvsPU (epistemic) metric as compared to the Standard dropout. We can notice that in this configuration, the PAvsPU (epistemic) metric is better as compared its deterministic counterpart. Further, it shows a nice correlation with the mIoU metric as well.

Looking at the results with the dropout probability p = 0.5, we can observe that there is a significant performance drop as compared to the baseline network. As before, the ECE and the MCE quality metrics are better with the MC dropout but they do not show any correlation with the mIoU metric. However, with the PAvsPU (epistemic) metric, we do observe some correlation.

Based on the above mentioned observations, we can conclude that the network which uses a lower dropout probability i.e. p = 0.2 with either the Standard or the MC dropout performs better as compared to using a higher dropout probability. In terms of the epistemic uncertainty quality, using MC dropout yields better results than the Standard dropout. Hence, the epistemic network which uses a dropout probability p = 0.2 and performs MC dropout gives us the best results. Compared with the baseline, there is a minor performance drop but we see a better quality epistemic uncertainty estimate. Here, we make our quality argument based on the PAvsPU (epistemic) metric which shows a pretty decent correlation with the mIoU.

4.2.4 Aleatoric + Epistemic network configuration

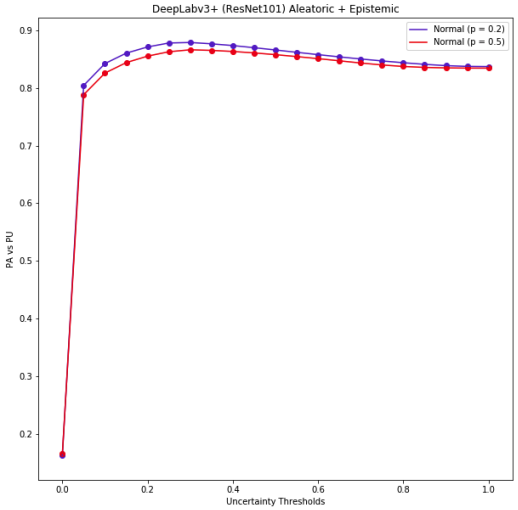
In this subsection, we show the results that we obtain with our aleatoric + epistemic configuration. For all the networks in this configuration, we add the dropout layers in the middle flow of the network and conduct separate experiments with two different probabilities: p = 0.2 and p = 0.5. For training the networks, we use the aleatoric loss function. Apart from that, we evaluate the networks using both the MC and the Standard dropout.



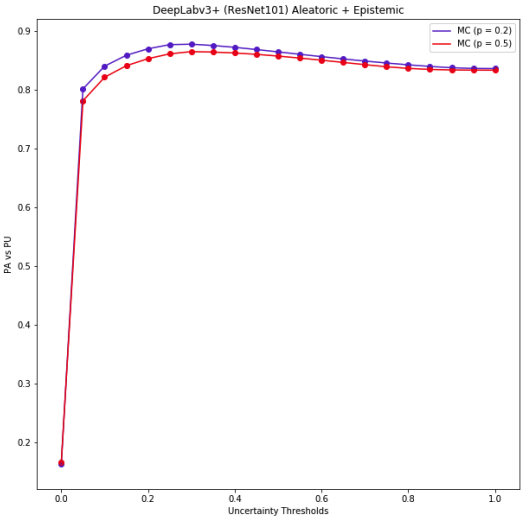
(a) Calibration plot obtained by using different dropout probabilities.

Figure 4.18: Calibration plots obtained with the aleatoric + epistemic network configuration using different dropout probabilities.

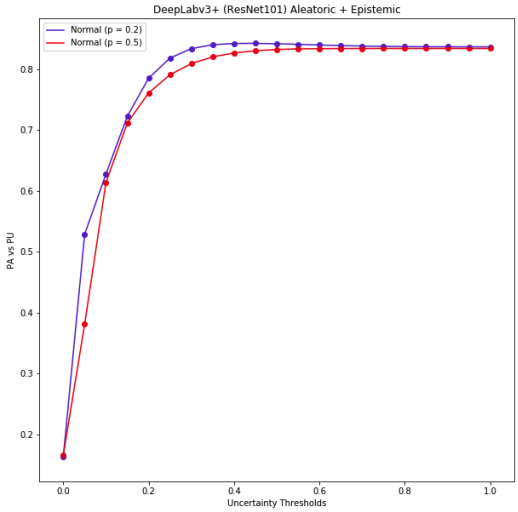
In the table 4.10, for the dropout probability p = 0.2, we can observe that our networks give a significant performance boost over the baseline deterministic configuration. The network which uses the Standard dropout performs better as compared to the one which uses the MC



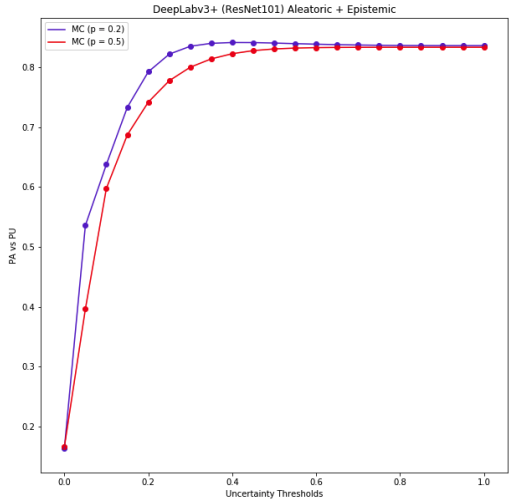
(a) Plot of the PAvsPU (epistemic) metric against different uncertainty thresholds obtained with the Standard dropout using different probabilities



(b) Plot of the PAvsPU (epistemic) metric against different uncertainty thresholds obtained with the MC dropout using different probabilities



(c) Plot of the PAvsPU (aleatoric) metric against different uncertainty thresholds obtained with the Standard dropout using different probabilities



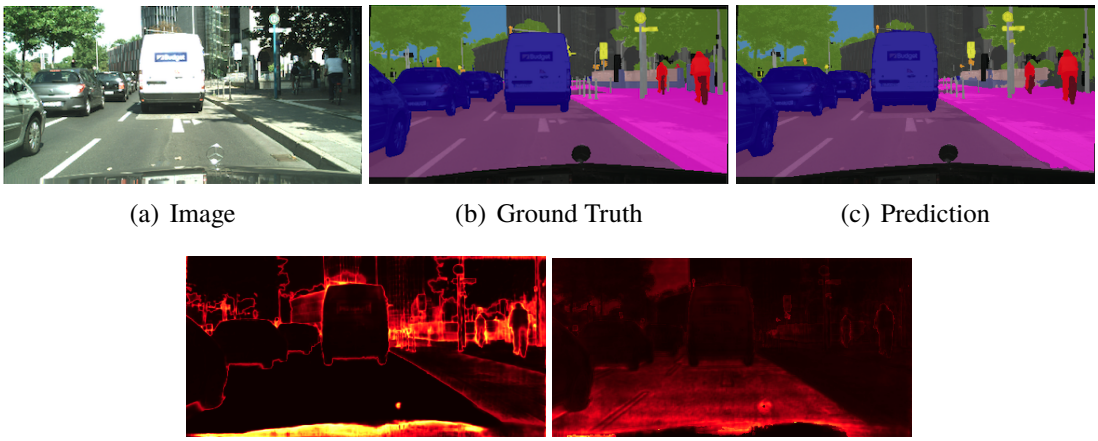
(d) Plot of the PAvsPU (aleatoric) metric against different uncertainty thresholds obtained with the MC dropout using different probabilities

Figure 4.19: Plots for the PAvsPU metrics obtained with the aleatoric + epistemic network configuration using different dropout probabilities.

dropout. In terms of the uncertainty quality, the ECE and the MCE metrics are better with the MC dropout however they do not show any correlation with the mIoU performance metric. For the PAvsPU (epistemic) metric, we do observe some correlation with the mIoU. Looking at the PAvsPU (aleatoric) metric, we can observe that it stays the same for both the Standard and the MC dropout and seem to be somehow correlating with the mIoU metric as well. But, if we compare it with the value which we achieved for our aleatoric configuration, there is a significant drop. In terms of the mIoU metric, our aleatoric + epistemic (p = 0.2) networks give

better results than the aleatoric configuration though. Figure 4.20 shows the visual results that we obtain with our aleatoric + epistemic (p = 0.2 + Normal) configuration.

Looking at the results for the dropout probability p = 0.5, we can observe that there is a performance drop over the baseline network, in terms of the mIoU metric. The values of the ECE and the MCE metrics seem to be positively correlating with the performance of the networks, however the PAvsPU metrics show the opposite trend. With the MC dropout, the network improves its performance but the PAvsPU metrics does not. In addition, the PAvsPU (aleatoric) metric is higher for the network which uses Standard dropout as compared to what we achieved for the networks with the dropout probability p = 0.2, but the drop in the mIoU metric overrules this increase. Figure 4.19 shows the plots for the PAvsPU metrics and figure 4.18 shows the calibration plots, for the networks using either the MC or the Standard dropout with different dropout probabilites, respectively.



(a) Image (b) Ground Truth (c) Prediction

(d) Epistemic Uncertainty (e) Aleatoric Uncertainty

Figure 4.20: Illustration of the visual results obtained with the aleatoric + epistemic (p = 0.2 + Normal) configuration.

In summary, we can conclude that the network which uses Standard dropout with a dropout probability p = 0.2 gives us the best results both in terms of the mIoU and the uncertainty quality metrics. If we take a closer look on the PAvsPU metrics and compare it with the previous configurations, we see a better value for the PAvsPU (epistemic) metric as compared to the PAvsPU (aleatoric) metric. This gives us the idea that measuring the aleatoric uncertainty, by training the network with the loss function given in the equation 3.8 first, focuses explicitly on improving the mIoU metric whereas using dropout for measuring the epistemic uncertainty focuses more on improving the quality estimates rather than enhancing the performance of the network significantly.

4.2.5 Concluding remarks

In the above subsections, we demonstrated the results that we obtained with our different experimental configurations of the DeepLabv3+ network architecture. The main objective of conducting these different experiments was to give comprehensive answers to the research

questions that we formulated in the subsection 3.1.1 for the Semantic segmentation task. If we take a look back at those questions, we firstly wanted observe whether the performance of a network improves when we make it capable of measuring the epistemic uncertainty by adding the dropout layers and subsequently performing the MC dropout. In our experiments with the DeepLabv3+ network architecture, we discovered that, by performing these steps we indeed get a better quality epistemic uncertainty but the performance of the network, in terms of the mIoU metric, drops slightly. In addition, we wanted to observe the effect of the dropout probability strength. Similar to the PSPNet, we observed a negative correlation between the dropout probability and the performance of the network. The network tends to perform much better when we use a lower dropout probability such as p = 0.2. However, interestingly, we figured out that with the lower probability, the Standard dropout performs the same or even better than the MC dropout.

Moving on, we wanted to figure out the best way for summarizing the information contained in an aleatoric variances vector. We already figured out the answer to this question with our PSPNet experiments but, for the sake of verification, we performed the same set of steps for reaching the answer with the DeepLabv3+ network as well. Luckily, our observations exactly matched with our previous same experiment with the PSPNet. Additionally, we wanted to explore whether the aleatoric uncertainty estimation helps improve the performance of the network or not. In that context, we saw a very slight decrease in the performance but at an advantage of getting a good aleatoric uncertainty estimate. However, when we estimated both the epistemic and the aleatoric uncertainty together, we observed a great performance boost over the baseline as well as appropriate quality uncertainty estimates.

Finally, we used a few metrics for specifying the quality of our uncertainty estimates. In the literature, it is common to see the use of the ECE and the MCE metrics but we additionally included the PAvsPU metric which lets us quantify both the epistemic and the aleatoric uncertainty quality. We observed that the PAvsPU metric correlates nicely with our performance metric (mIoU) which made us conclude that we should indeed use this metric for measuring the quality of the uncertainty estimates.

Network configuration	mIoU	ECE	MCE	PAvsPU (epistemic)	PAvsPU (aleatoric)
Aleatoric + Epistemic (p = 0.2 + Normal)	73.8	0.097	0.056	0.831	0.626
Aleatoric + Epistemic (p = 0.2 + MC)	73.6	0.095	0.054	0.830	0.626
Aleatoric + Epistemic (p = 0.5 + Normal)	70.7	0.100	0.067	0.816	0.632
Aleatoric + Epistemic (p = 0.5 + MC)	70.9	0.099	0.063	0.812	0.620

Table 4.10: Results obtained with the aleatoric + epistemic configuration using different dropout probabilities.

4.3 Deep insights

In this section, we will present some insights that we got from the results of our experiments in the context of the Semantic segmentation task. Our main goal was to verify the framework presented by the authors in [1] for some situations that we were not addressed in their work. In that context, we conducted multiple experiments with two different network architectures. We presented the results of these experiments in the above sections and performed an analysis which gave us a deep understanding of the working of the method [1].

Considering the performance of the network in terms of the mIoU metric, we believe that it is better to use the aleatoric loss instead of using the standard cross-entropy loss. The aleatoric loss makes a network capable of estimating the aleatoric uncertainty and, at the same time, makes the network robust to the misclassifications. In standard deep learning, it is a good practice to add the dropout regularization in DNNs to prevent them from over-fitting. Additionally, as mentioned in the sub-subsection 2.3.2.1, if we train a network with the dropout layers, it is the same as performing approximate VI in the probabilistic version of the same network. Keeping that in mind, we believe that we should add the dropout layers in our network but the selection of the dropout probability should be smart. For the Semantic segmentation, where we have a dense prediction task, we should use a lower dropout probability. By the addition of the dropout layers, we get a good quality epistemic uncertainty estimate as well as correlating mIoU metrics, however the performance hardly improves. Interestingly, we believe that a single forward pass in the Standard dropout settings is enough to get nice results.

Summing it all up, adding dropout layers in the network along with using the aleatoric loss function promises us a performance boost as well as good quality uncertainty estimates. The aleatoric loss function has more contribution in improving the mIoU performance metric whereas the dropout layers play a major role in the confidence calibration of our network and thus producing correlating uncertainty quality metrics. Finally, we think that using the Standard dropout instead of the MC dropout yields almost the same or sometimes even better performance atleast in the case of Semantic segmentation.

5 Experimental Observations and Analysis with Image Classification

In this chapter[1], we would show the results that we obtain with our Image classification experiments. As mentioned in the subsection 3.3, we start with performing our experiments on the MNIST dataset and identify the ones that give us the best results. Following that, we conduct these best performing experiments with the FashionMNIST and the CIFAR10 dataset. Remember, we designed these experiments based on our inductive belief of the input data space complexities that might result in a high aleatoric uncertainty estimate. So, our main objective is to validate our belief by using the method presented by the authors in [1]. This would additionally verify if this method produces uncertainty estimates, specifically the aleatoric uncertainty, that correlate with the input data space complexities.

Considering our empirical observations, we would present them separately for every dataset in different sections. Instead of giving absolute numbers for the uncertainty metrics, we would rather give the values of the AUROC metric for different detection tasks that would give us a solid idea of whether our experiments are performing accurately as desired or not. Further, we would give the results of our new classifier along with every qualifying experiment as well.

Comments on the datasets: As mentioned before, we work with three different datasets. Firstly, the MNIST dataset consists of gray-scale images, with 28x28 spatial resolution, representing hand-written digits from 1 to 10. We have 60,000 images in the training-set and 10,000 images in the test-set. Further, the class distribution is almost balanced for this dataset. In our experiments, we use the full training-set along with using the test-set as the validation-set for supervising the training process. At test-time, we use the test-set for evaluation purposes. Secondly, the FashionMNIST dataset consists of gray-scale images, with 28x28 spatial resolution, representing different pieces of clothing. In total, we have 10 distinct classes for this dataset that are balanced in the training as well in the test-set. In addition, the number of images in the training-set are 60,000 and the number of images in the test-set are 10,000. For training purposes, we use the whole training-set along with using the test-set for validation purposes. At test-time, we use the full test-set. Finally, the CIFAR10 dataset consists of colored images, with 32x32 spatial resolution, representing images belonging to different classes such as cars, cats, dogs and so forth. In total, we have 10 distinct classes that are balanced in the training and the test-set. The number of images in the training-set as well as the test-set are 50,000 and 10,000 respectively. We use the whole training-set along with using the test-set for validation purposes. At test-time, we use the whole test-set.

Notation remark: In all the tables that we present in this chapter, the phrase **High aleatoric**

[1] For getting an idea of the notation which we use while demonstrating the results of our experiments here, please look at the subsection 3.3.3.

would refer to the **High aleatoric uncertainty data**, **OOD** would refer to the **OOD data** and **Misclassified** would refer to the **Misclassified data**. In addition, we would use the short form **Aleatoric** for the **Aleatoric uncertainty**, **Mutual Info.** for the **Mutual information** and **Predictive** for the **Predictive uncertainty**.

5.1 Results obtained with MNIST dataset

In this section, we will present the results that we obtained with our different experiments by using the MNIST dataset. As mentioned in the subsection 3.3.2, we use the LeNet5 network architecture with this dataset. For training purposes, we use the deep learning framework called *PyTorch* [73] with the following settings:

- Batch size (both training and validation): 64.
- Training epochs: 35.
- Optimizer: Stochastic gradient descent with its momentum set equal to 0.9 and the L2 weight regularization strength set equal to 0.0001.
- Learning rate: We start with an initial learning of 0.01 and proceed with the exponential learning rate decay policy where we reduce the learning rate by 0.1 after every 20 epochs.
- Number of samples extracted from the logit space Gaussian distribution: 500
- Weights initialization: We use the default initialization techniques used by the different layers in PyTorch.
- Training data augmentations: We standardize the full size images by subtracting the mean and subsequently dividing by the standard deviation[2]. Following that, for some of the randomly selected images, we apply rotations in the range of -10° to 0°.
- Validation data augmentations: We just standardize the full size images by subtracting the mean and subsequently dividing by the standard deviation. The values of the mean as well as the standard deviation obtained are the ones that we obtain from our training data images.

At test-time, we use the following set of settings for obtaining our results:

- Batch size: 64.
- Number of stochastic forward passes for MC dropout: 25.

[2]Note, we compute the mean as well as the standard deviation of all the training data images pixels and use these values in the standardization process.

- Testing data augmentations: We use the full size test-set images at the test-time. Remember, this test-set additionally consists of the boundary data examples as well as the OOD data (FashionMNIST). Before feeding these images to our network, we standardize them by subtracting the mean and subsequently dividing by the standard deviation. We use the value of the mean as well as the standard deviation obtained from the training data images.

5.1.1 Horizontal combination of class label 5 and 7 images

In this subsection, we demonstrate the results of our experiment where we combine half class label 5 and 7 images, from the test-set, horizontally and include them along with the FashionMNIST testing data images in our overall test-data stream. The combined images represent the boundary data examples whereas the FashionMNIST images denote the OOD data. As mentioned in the section 3.3, we include these combined images in our training data as well where we assign the class label 5 to half of them and the class label 7 to the other half. We should see a high aleatoric uncertainty for these horizontally combined images and a high mutual information as well the predictive uncertainty for the OOD data in the test-set.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	95.8	57.7	97.3
OOD	99.2	95.3	97.6
Misclassified	98.0	97.6	91.0

Table 5.1: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.1, we can observe that we get a very high AUROC score for the aleatoric uncertainty while performing the high aleatoric data detection task. We see a high score for the predictive uncertainty since it is a combination of the epistemic and the aleatoric uncertainty. However, the mutual information (epistemic) score is low since we trained our network on the high aleatoric data as well. For the OOD data detection, we can observe that we see a high score for all the uncertainty estimates. The mutual information score is a bit lower than the predictive uncertainty but still it validates our choice of an alternative metric for representing the epistemic uncertainty. Additionally, the high score for the aleatoric uncertainty shows that the aleatoric uncertainty actually gets high for the OOD data. For the misclassified examples, we can see an almost similar high score for the predictive uncertainty and the mutual information. Interestingly, the aleatoric uncertainty score is high as well. Since we achieved a high AUROC aleatoric uncertainty score for our high aleatoric data detection task, we observed the performance of our new classifier in this setting as well.

In the figure 5.1(a), we can see the confusion matrix that we got for our OOD and ID data partioning task. We can observe that our OOD data gets separated pretty well but we observe some false negatives for our ID data. Apart from that, in the figure 5.1(b), we can observe the performance of our classifier on partitioning the ID data. Based on the uncertainty estimates, we are able to identify our boundary data as well as the misclassified data with a very good

accuracy. For the correctly classified examples, we can observe some false negatives but still the high number of true positives overrule them.

In summary, we consider this experiment where we use the labelling policy of assigning the two different hard labels to our horizontally combined boundary data images as being a part of the best performing experiments. We do that based on our AUROC scores in different data types detection task. Additionally, our well performing classifier strengthens this argument and proves that it can indeed be used to classify the input data into different categories in real practical settings.

5.1.2 Vertical combination of class label 5 and 7 images

In this subsection, we demonstrate the results of our experiment where we combine half class label 5 and 7 images, from the test-set, vertically and include them along with the Fashion-MNIST testing data images in our overall test-data stream. For the training boundary data examples we follow the same labelling policy as for the horizontal combination of the same images.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	96.0	54.6	98.4
OOD	98.5	96.0	96.4
Misclassified	98.2	98.1	88.5

Table 5.2: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.2, we can observe that we get a very high AUROC score for the aleatoric uncertainty while performing the high aleatoric data detection task. We see a high score for the predictive uncertainty since it is a combination of the epistemic and the aleatoric uncertainty. However, the mutual information (epistemic) score is low since we trained our network on the high aleatoric data as well. Additionally, the aleatoric uncertainty AUROC is a bit higher for the vertical combination as compared to the previous horizontal one. For the OOD and the misclassified data detection, we observe a similar pattern of the scores to the horizontal combination. Since we achieved a high AUROC aleatoric uncertainty score for our high aleatoric data detection task, we observed the performance of our new classifier in this setting as well.

In the figure 5.2(a), we can see the confusion matrix that we got for our OOD and ID data partioning task. We can observe that our OOD data gets separated pretty well with only a few false negatives but our ID data gives a bit more false negatives. Apart from that, in the figure 5.2(b), we can observe the performance of our classifier on partitioning the ID data. Based on the uncertainty estimates, we are able to identify our boundary data as well as the misclassified data with a very good accuracy. For the correctly classified examples, we can observe some false negatives but still the high number of true positives overrule them.

In summary, we consider this experiment where we use the same labelling policy as for the horizontally combined images as being a part of the best performing experiments. Additionally, we conclude that the value of the aleatoric uncertainty for the boundary data examples is

almost invariant to the type of the combination which we use (horizontal or vertical).

5.1.3 Horizontal combination of class label 1 and 3 images

In this subsection, we demonstrate the results of our experiment where we combine half class label 1 and 3 images, from the test-set, horizontally and include them along with the FashionMNIST testing data images in our overall test-data stream. For the training boundary data examples we assign the class label 1 to half of them and the class label 3 to the other half. The main purpose of performing this experiment with a different class label images combination was just to observe whether our results are dependent on a particular combination of classes or not. Considering that, we should see a high aleatoric uncertainty for these horizontally combined images and a high mutual information as well the predictive uncertainty for the OOD data in the test-set.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	96.1	55.5	99.1
OOD	99.2	96.1	96.6
Misclassified	98.2	97.7	87.5

Table 5.3: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.3, we can observe that we get a very high AUROC score for the aleatoric uncertainty while performing the high aleatoric data detection task. In addition, the score is a bit higher than what we observed for our class label 5 and 7 images horizontal/vertical combination. The reason for this could be that the class label 1 and 3 images horizontal combination gives us an image which represents a geometrical shape that manages to confuse our network a bit more. Further, we see a high score for the predictive uncertainty since it is a combination of the epistemic and the aleatoric uncertainty. However, the mutual information (epistemic) score is low since we trained our network on the high aleatoric data as well. For the OOD data detection, we can observe that we see a high score for all the uncertainty estimates. Here, the interesting thing is that we see a higher score for the aleatoric uncertainty than the mutual information which shows that we indeed get a high aleatoric uncertainty for the OOD data examples. The reason for the high value could be that our network is returning some random values since it is not trained to predict the aleatoric uncertainty for the OOD examples. However, if we rely on this fact, we believe that the network could throw a random lower value as well. But from the AUROC score for the aleatoric uncertainty, it seems like that the network is giving a higher value for almost 96% of the total OOD samples. In terms of the misclassified data detection, we see a high score for all the uncertainty estimates as well. However, the score for the predictive uncertainty and the mutual information is comparatively higher than the aleatoric uncertainty. Since we achieved a high AUROC aleatoric uncertainty score for our high aleatoric data detection task, we observed the performance of our new classifier in this setting as well.

In the figure 5.3(a), we can see the confusion matrix that we got for our OOD and ID data partioning task. We can observe that our OOD data gets separated pretty well but our ID

data gives some false negatives. However, we still do observe a high number of true positives for our ID samples. Apart from that, in the figure 5.3(b), we can observe the performance of our classifier on partitioning the ID data. As evident, the misclassified and the boundary data examples gets separated pretty well with a very few false negatives. Similarly, we get an appropriate classification accuracy for the correctly classified data as well.

In summary, we consider this experiment where we use the labelling policy of assigning the two different hard labels to our horizontally combined boundary data images as being a part of the best performing experiments. We do that based on our AUROC scores in different data types detection task. Additionally, we conclude that the aleatoric uncertainty estimate is not that invariant to the boundary data examples that are produced from some different class label images combination.

5.1.4 Vertical combination of class label 1 and 3 images

In this subsection, we demonstrate the results of our experiment where we combine half class label 1 and 3 images, from the test-set, vertically and include them along with the Fashion-MNIST testing data images in our overall test-data stream. For the training boundary data examples we follow the same labelling policy as for the horizontal combination of the same images.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	95.8	47.1	99.4
OOD	99.3	94.5	97.1
Misclassified	97.8	97.4	84.8

Table 5.4: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.4, we can observe that we get a very high AUROC score for the aleatoric uncertainty in the high aleatoric data detection. The low score for the mutual information shows the presence of our boundary data examples in the training data. Considering the OOD data detection, we observe a high value for almost all of the uncertainty estimates. The score for the mutual information proves it validity for quantifying the epistemic uncertainty whereas the high score for the aleatoric uncertainty shows that the OOD data examples indeed give us a high estimate for the aleatoric uncertainty. For the misclassified examples, we observe a comparatively higher score for the predictive uncertainty and the mutual information as compared to the aleatoric uncertainty. As our high aleatoric data detection task performed well, we observed the performance of our new classifier in this setting too.

Figure 5.4 shows the performance of our classifier on the partitioning of the data into different categories. In (a), we can observe that we get a very decent performance for the OOD and the ID data separation task. We observe some false negatives for the ID data but at an advantage of observing a significant amount of true positives. In (b), we can observe that our classifier partitions the high aleatoric uncertainty data with 100% accuracy and performs decently on the other data classes too.

In summary, we include this experiment in our best performing set of experiments. We use a labelling policy which is similar to the one that we used for our same horizontal combination.

5.1.5 Equal weights combination of class label 5 and 7 images

In this subsection, we demonstrate the results of our experiment where we combine half class label 5 and 7 images, from the test-set, with an equal weight of 0.5 and include them along with the FashionMNIST testing data images in our overall test-data stream. By equal weight combination, we mean that we just add the two different images together by first scaling their pixel values by 0.5. For the training boundary data examples we assign the class label 5 to half of them and the class label 7 to the other half. We expect to observe a high aleatoric uncertainty for our combined images and a high mutual information as well the predictive uncertainty for the OOD data in the test-set.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	96.7	73.2	98.3
OOD	98.9	95.2	98.2
Misclassified	98.0	97.3	89.7

Table 5.5: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.5, we can observe that we get a very high AUROC score for the aleatoric uncertainty in the high aleatoric data detection. However, we can see a bit higher score for the mutual information as well which means that our network considers the boundary data examples in this case as somewhat being OOD. Considering the OOD data detection, we observe a similar pattern of the scores to our previous experiments. For the misclassified examples, we observe a comparatively higher score for the predictive uncertainty and the mutual information as compared to the aleatoric uncertainty. As our high aleatoric data detection task performed well, we observed the performance of our new classifier in this setting too.

Figure 5.5 shows the performance of our classifier on the partitioning of the data into different categories. In (a), we can observe that we get a 100% accuracy for the OOD class with a decent performance for the ID class as well. In (b), we can observe that our classifier partitions the different ID data classes with an appropriate accuracy. We observe a significant number of false negatives for the correctly classified data however the number of true positives still overrule them.

In summary, we include this experiment in our best performing set of experiments. Here, we use a labelling policy which is similar to all of our previous experiments. Additionally, based on the results of this experiment and all the other previous experiments, we have got a sufficient proof that labelling the boundary data examples with two different hard labels would indeed confuse our network and make it produce high aleatoric uncertainty for such data samples.

5.1.6 Horizontal combination of class label 5 and 7 images with soft labels

In this subsection, we demonstrate the results of our experiment where we combine half class label 5 and 7 images, from the test-set, horizontally and include them along with the FashionMNIST testing data images in our overall test-data stream. For the training boundary data examples, we follow a different labelling policy as compared to our previous experiments. As mentioned in the section 3.3, we get a one-hot encoding representation of our labels for the data. For the boundary data samples, we assign a soft label of 0.5 to the index 5 and 7 in this one-hot encoded vector representation for its corresponding labels. We expect to observe a high aleatoric uncertainty for our combined images and a high mutual information as well the predictive uncertainty for the OOD data in the test-set.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	95.4	55.4	53.7
OOD	99.3	95.3	99.3
Misclassified	98.1	97.7	93.5

Table 5.6: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.6, we can observe that we get a pretty low AUROC score for the aleatoric uncertainty in the high aleatoric data detection task. However, the scores for the mutual information and the predictive uncertainty follow the similar pattern to our previous set of experiments. For the OOD detection task, we can see that the score for the aleatoric and the predictive uncertainty are similar and quite high. Finally, for the misclassified data detection task, we can observe that the scores are almost in line with our observations from all the other previous experiments. Since we do not get a good score for the aleatoric uncertainty in our high aleatoric data detection task, we do not train and observe the performance of our classifier in this setting.

In summary, we do not include this experiment in our best performing list of experiments. For the bad performance of the aleatoric uncertainty in the high aleatoric data detection task, we believe that the reason could the way we assigned labels to our boundary data examples during the training process. If we assign an equal soft label to two classes, then the network just tends to increase the confidence equally for both of the classes instead of actually getting confused, which it did when we assigned hard labels to our boundary data examples.

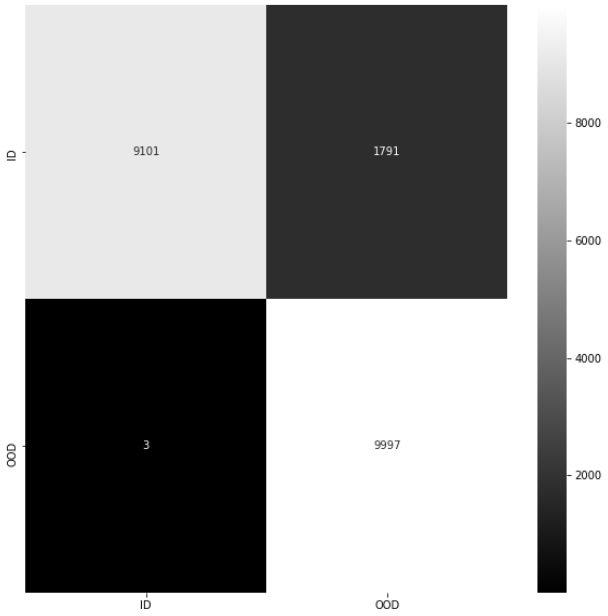
5.1.7 Best performing list of experiments

In the previous subsections, we demonstrated the results of our experiments that we conducted with the MNIST dataset. As mentioned in the section 3.3, we proceed with selecting the best performing experiments and later conducting them with some other datasets for observing whether we get a similar set of observations or not. Thus, based on the results presented in the above subsections, we select the following experiments for the other datasets:

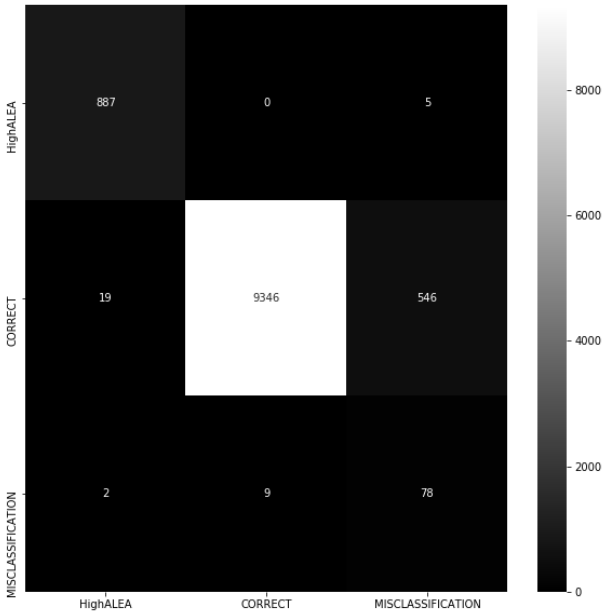
- Horizontal combination of two different class label images.

- Vertical combination of two different class label images.
- Equal weights combination of two different class label images.

We conduct the above mentioned experiments with two different datasets named as FashionMNIST and CIFAR10. In the next sections, we would present the results that we obtain for these datasets separately.

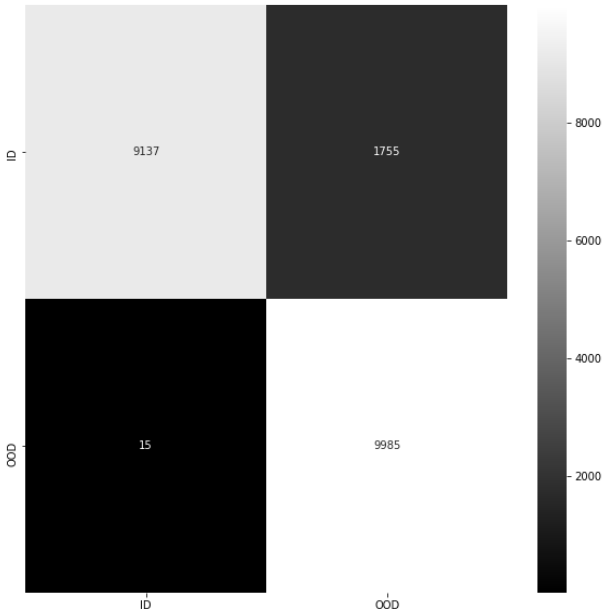


(a) Confusion matrix for the ID and the OOD data separation task

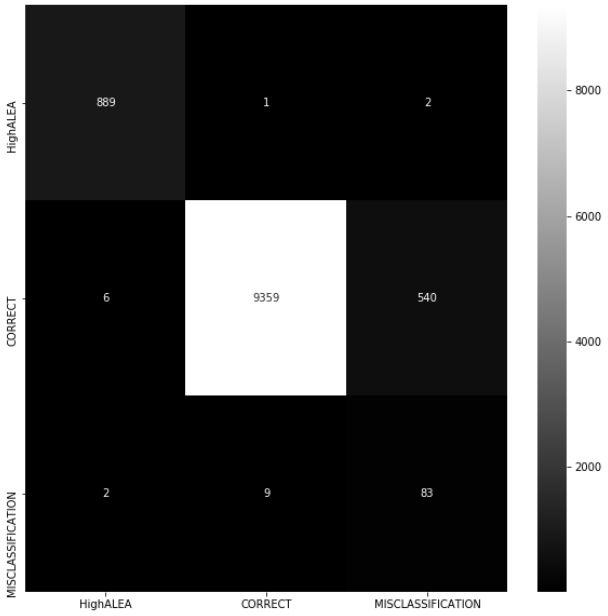


(b) Confusion matrix for the ID data partitioning task

Figure 5.1: Confusion matrices demonstrating the performance of our new classifier with class label 5 and 7 horizontally combined boundary data images.

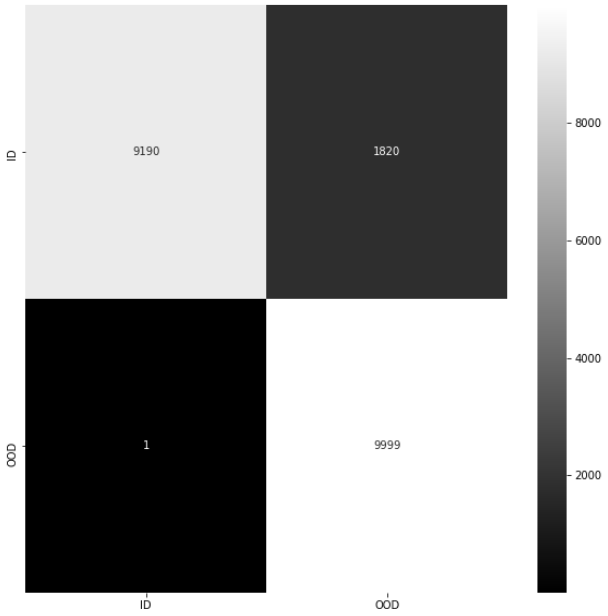


(a) Confusion matrix for the ID and the OOD data separation task

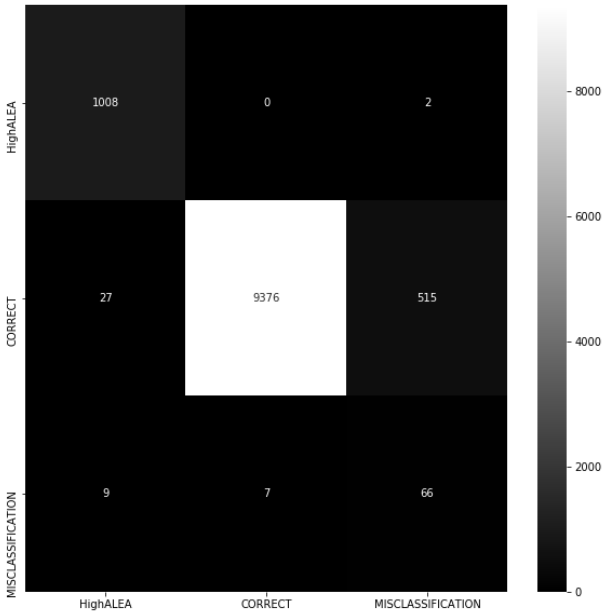


(b) Confusion matrix for the ID data partitioning task

Figure 5.2: Confusion matrices demonstrating the performance of our new classifier with class label 5 and 7 vertically combined boundary data images.

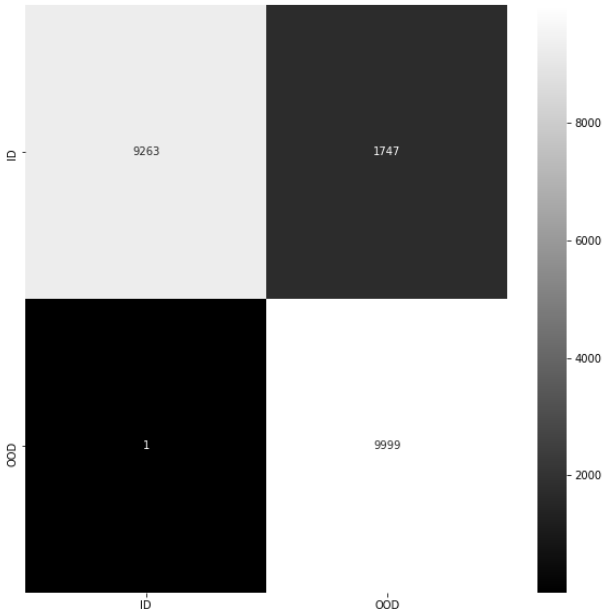


(a) Confusion matrix for the ID and the OOD data separation task

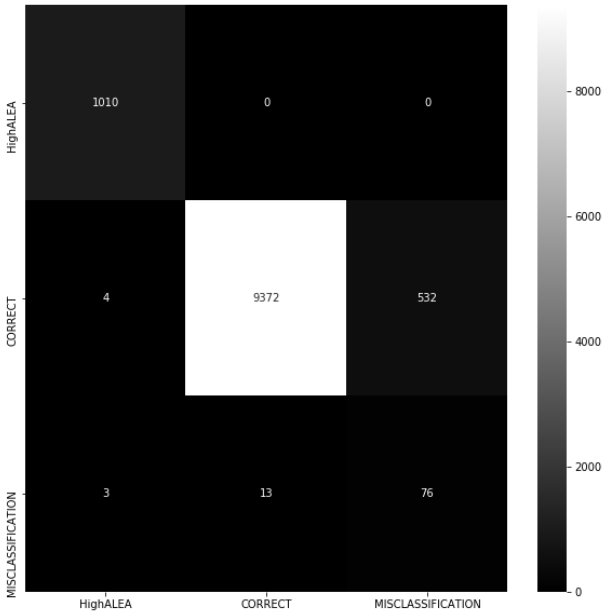


(b) Confusion matrix for the ID data partitioning task

Figure 5.3: Confusion matrices demonstrating the performance of our new classifier with class label 1 and 3 horizontally combined boundary data images.

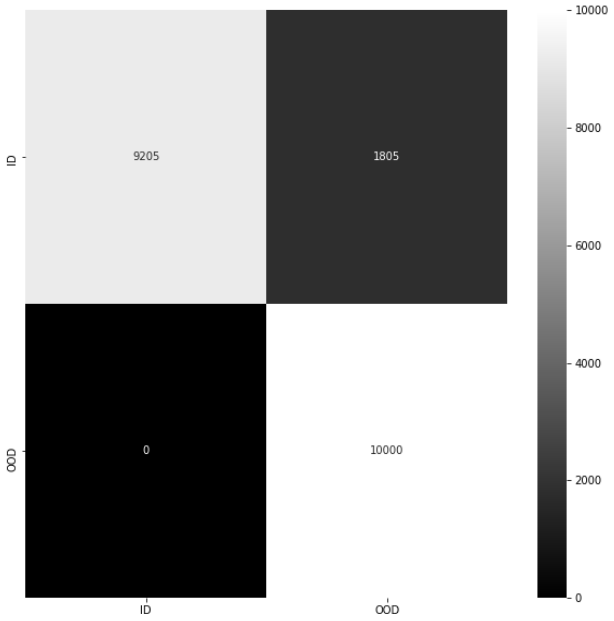


(a) Confusion matrix for the ID and the OOD data separation task

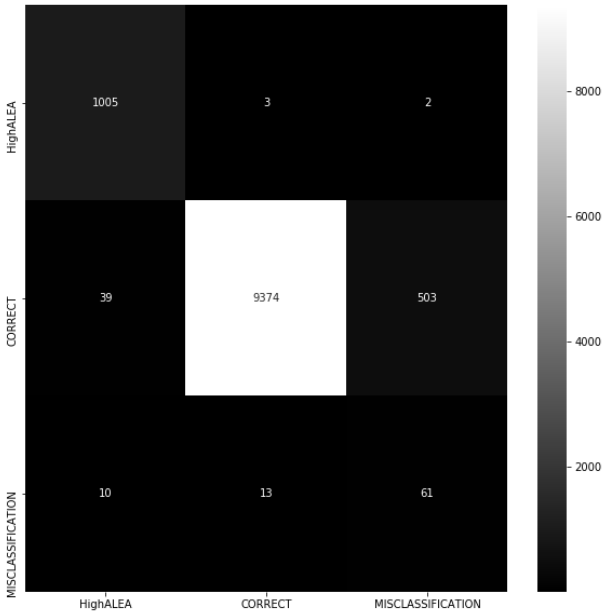


(b) Confusion matrix for the ID data partitioning task

Figure 5.4: Confusion matrices demonstrating the performance of our new classifier with class label 1 and 3 vertically combined boundary data images.



(a) Confusion matrix for the ID and the OOD data separation task



(b) Confusion matrix for the ID data partitioning task

Figure 5.5: Confusion matrices demonstrating the performance of our new classifier with class label 5 and 7 equally weighted boundary data images.

5.2 Results obtained with FashionMNIST dataset

In this section, we will present the results that we obtained with our different experiments by using the FashionMNIST dataset. As mentioned in the subsection 3.3.2, we use the VGG-7 network architecture with this dataset. For training purposes, we use the deep learning framework called *PyTorch* [73] with the following settings:

- Batch size (both training and validation): 64.
- Training epochs: 40.
- Optimizer: Stochastic gradient descent with its momentum set equal to 0.9 and the L2 weight regularization strength set equal to 0.0005.
- Learning rate: We start with an initial learning of 0.01 and proceed with the exponential learning rate decay policy where we reduce the learning rate by 0.1 after every 20 epochs.
- Number of samples extracted from the logit space Gaussian distribution: 500
- Weights initialization: We use the default initialization techniques used by the different layers in PyTorch.
- Training data augmentations: We standardize the full size images by subtracting the mean and subsequently dividing by the standard deviation[3]. Following that, for some of the randomly selected images, we apply rotations in the range of -10° to 0°.
- Validation data augmentations: We just standardize the full size images by subtracting the mean and subsequently dividing by the standard deviation. The values of the mean as well as the standard deviation obtained are the ones that we obtain from our training data images.

At test-time, we use the following set of settings for obtaining our results:

- Batch size: 64.
- Number of stochastic forward passes for MC dropout: 25.
- Testing data augmentations: We use the full size test-set images at the test-time. Remember, this test-set additionally consists of the boundary data examples as well as the OOD data (MNIST). Before feeding these images to our network, we standardize them by subtracting the mean and subsequently dividing by the standard deviation. We use the value of the mean as well as the standard deviation obtained from the training data images.

[3]Note, we compute the mean as well as the standard deviation of all the training data images pixels and use these values in the standardization process.

5.2.1 Horizontal combination of class label 7 and 9 images

In this subsection, we demonstrate the results of our experiment where we combined class label 7 and 9 images, in the test-set, horizontally. Here, the class label 7 is assigned to the images that represent "sneaker boots" whereas the class label 9 is assigned to the "ankle boots" images. We include these horizontally combined images (boundary data examples) in our test-data stream along with the normal FashionMNIST and the OOD MNIST dataset images. Since we create the same horizontal combination of images in our training data as well, we assign a hard label of 7 to half of them and a hard label of 9 to the other half, for training purposes. As with our same MNIST dataset experiment, we expect to see a high aleatoric uncertainty for our horizontally combined images and a high predictive uncertainty as well as the mutual information for the OOD data samples.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	91.4	38.2	99.7
OOD	85.2	92.8	23.2
Misclassified	93.1	91.2	82.0

Table 5.7: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.7, we have some interesting set of results. Firstly, for the high aleatoric uncertainty data detection task, we can observe that we get a quite high AUROC score for the aleatoric uncertainty. This means that the training boundary data examples that we created as well as the labelling policy which we used for them, managed to successfully make our network learn produce a high value of the aleatoric uncertainty for such examples at test-time. In addition, we can observe a low value of the mutual information which is due to the presence of our high aleatoric uncertainty data in the training set. Secondly, in comparison with the same MNIST dataset experiment, we can see that we get a very low score for the aleatoric uncertainty in our OOD data detection task. We will share our insights on this particular observation in the last section of this chapter. Additionally, we observe a higher value for the mutual information as compared to the predictive uncertainty which solidifies our belief on the use of the mutual information metric for representing the epistemic uncertainty. Finally, for the misclassified data detection task, we can observe that we get a similar pattern of the scores, for different uncertainties, to our same MNIST dataset experiment. Since we have got a high score for the aleatoric uncertainty in the high aleatoric data detection task, we observe the performance of our new classifier in this setting as well.

Figure 5.6 shows the performance of our classifier on the partitioning of the data into different categories. In (a), we can observe that we get a high number of false negatives while separating the OOD data whereas the number of false negatives with the ID data are almost similar to what we got with our classifier on the MNIST dataset. In (b), we can observe that our classifier partitions the different ID data classes with an appropriate accuracy. Since the presence of the OOD data is highly critical for a network running live in production settings, we tried to improve the performance of our OOD/ID data separation task by training the main network with our new regularized loss function (stated in the sub-subsection 3.3.1.1). Afterwards, we followed the same set of rules for separating the OOD/ID data and for training our new classifier.

Figure 5.7 shows the confusion matrices that demonstrate the performance of our new classifier after training the main network with the regularized loss function. In (a), we can observe that we get a significant improvement for the OOD data separation since our true positives have increased from 8716 to 9111. In addition, the number of false negatives for the ID data class have increased a bit but not significantly. This shows that our new regularized loss function indeed improves the OOD/ID data separation task quite significantly. In (b), we can observe the performance of our new classifier on the ID data separation task. As evident, the results are almost similar to what we got without training our main network with the regularized loss function.

In summary, we conclude that this experimental setting, which we classified as best performing from our MNIST dataset experiments, indeed transfers well to the other datasets too - in this case, FashionMNIST. Additionally, our new regularized loss function helps in improving the OOD/ID data separation task especially when we are working with deeper networks and a bit complex datasets.

5.2.2 Vertical combination of class label 7 and 9 images

In this subsection, we demonstrate the results of our experiment where we combined class label 7 and 9 images, in the test-set, vertically. We include these vertically combined images (boundary data examples) in our test-data stream along with the normal FashionMNIST and the OOD MNIST dataset images. For the training boundary data examples, we assign the class label 7 to half of them and the class label 9 to the other half. We expect to see a high aleatoric uncertainty for our vertically combined images and a high predictive uncertainty as well as the mutual information for the OOD data samples.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	91.6	46.6	99.7
OOD	85.5	92.5	29.2
Misclassified	93.1	91.3	81.4

Table 5.8: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.8, we can observe that for the high aleatoric uncertainty data detection task, we get a quite high AUROC score for the aleatoric uncertainty. In addition, we get a low score for the mutual information which is due to the presence of our high aleatoric uncertainty data in the training set. For the OOD and the misclassified data detection task, we can observe that we get similar pattern of the scores to our previous experiment where we created the boundary data examples by combining the images horizontally. Since we have got a high score for the aleatoric uncertainty in the high aleatoric data detection task, we observe the performance of our new classifier in this setting as well.

Figure 5.7 shows the confusion matrices that demonstrate the performance of our new classifier. As evident, we have got a decent performance for the OOD/ID as well as the ID data

separation task. Therefore, we do not demonstrate the performance of our classifier after training the main network with the regularized loss function[4], in this case.

In summary, we conclude that this experimental setting, which we classified as best performing from our MNIST dataset experiments, indeed transfers well to the other datasets too - in this case, FashionMNIST. Additionally, as we said it with our MNIST experiments, the estimate of the aleatoric uncertainty is almost invariant to the type of the combination (either horizontal or vertical) which we use to create our boundary data examples.

5.2.3 Equal weights combination of class label 7 and 9 images

In this subsection, we demonstrate the results of our experiment where we combine the class label 7 and 9 images, from the test-set, with equal weights of 0.5 and include them along with the MNIST testing data images in our overall test-data stream. By equal weight combination, we mean that we just add the two different images together by first scaling their pixel values by 0.5. For the training boundary data examples we assign the class label 7 to half of them and the class label 9 to the other half. We expect to observe a high aleatoric uncertainty for our combined images and a high mutual information as well the predictive uncertainty for the OOD data in the test-set.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	91.4	48.3	86.4
OOD	80.9	89.8	21.8
Misclassified	92.8	90.9	83.2

Table 5.9: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.9, we can observe that we get a comparatively lower score for the aleatoric uncertainty, in the high aleatoric data detection task, as compared to our previous experiments. In addition, for the mutual information as well as the predictive uncertainty score, we observe a similar pattern to our last experiments. For the OOD and the misclassified data detection task, we can observe a similar trend for the scores as well. Since we have achieved a considerable score for the aleatoric uncertainty in the high aleatoric data detection task, we demonstrate the performance of our new classifier in this setting as well.

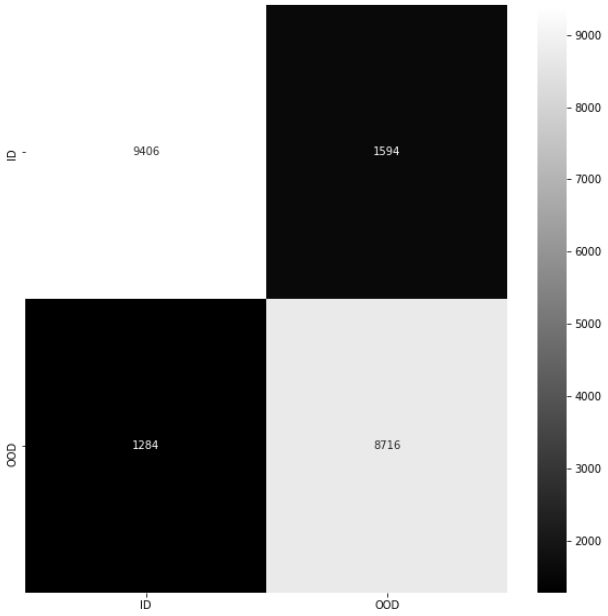
Figure 5.9 demonstrates the performance[5] of our new classifier in the form of confusion matrices. In (a), we can observe that we get some false negative both for the OOD and the ID data however the number of true positives are still very high. In (b), we can see the performance of our classifier for the ID data separation task. Due to the low aleatoric uncertainty score in the high aleatoric data detection task, we can observe that we get some false negatives as shown in the "HighALEA" row of the confusion matrix.

In summary, we conclude that this experimental setting performs comparatively bad as compared to the previous two settings. From the results, it seems like that our network gets confused

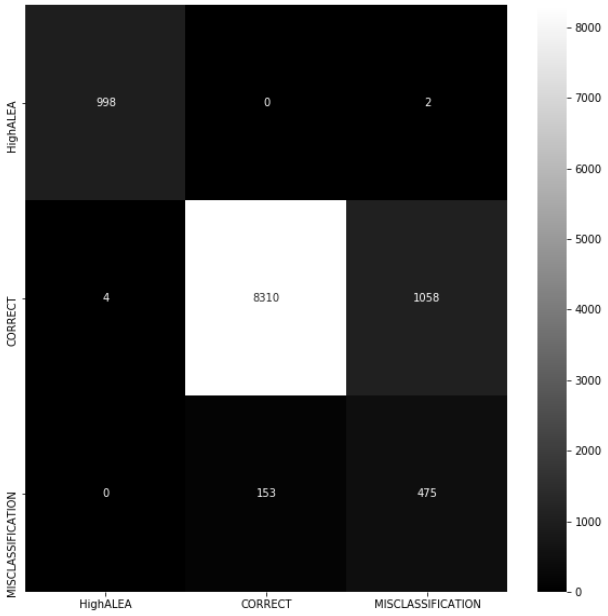
[4]Here, we believe that the performance might improve further by using the regularized loss function.

[5]For improving the OOD/ID data separation task, we could have tried our new regularized loss function but due to the shortage of time, we were not able to.

with equally weighted combined images but the contribution of confusion is more from the labelling policy than the images themselves.

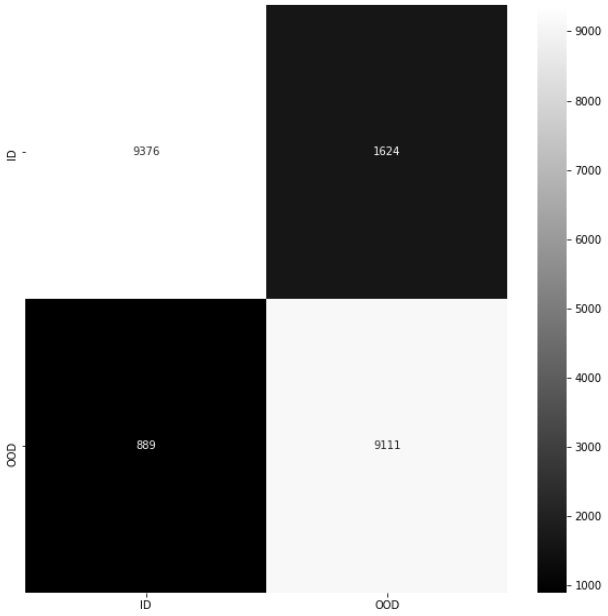


(a) Confusion matrix for the ID and the OOD data separation task

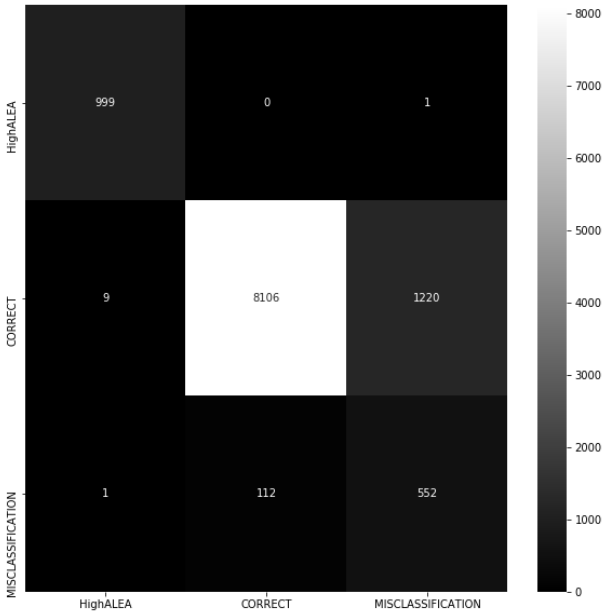


(b) Confusion matrix for the ID data partitioning task

Figure 5.6: Confusion matrices demonstrating the performance of our new classifier with class label 7 and 9 horizontally combined boundary data images.

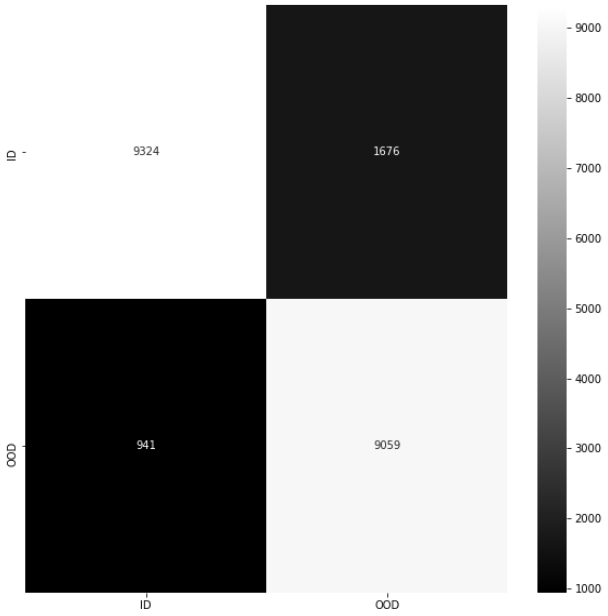


(a) Confusion matrix for the ID and the OOD data separation task

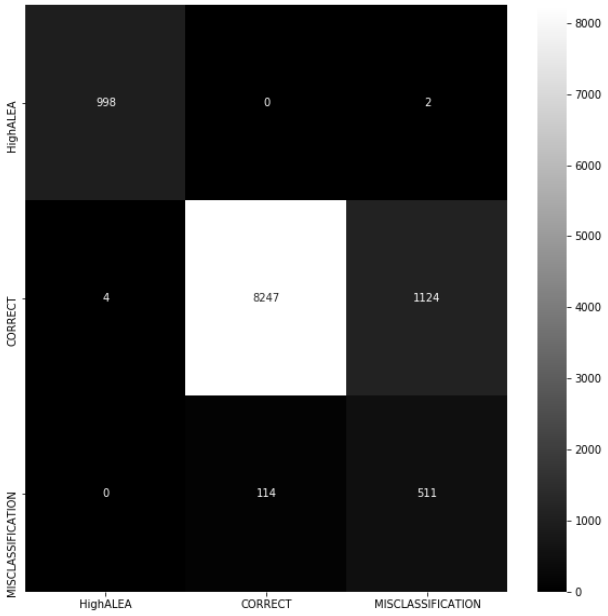


(b) Confusion matrix for the ID data partitioning task

Figure 5.7: Confusion matrices demonstrating the performance of our new classifier, after training the main network with the regularized loss function, with class label 7 and 9 horizontally combined boundary data images.

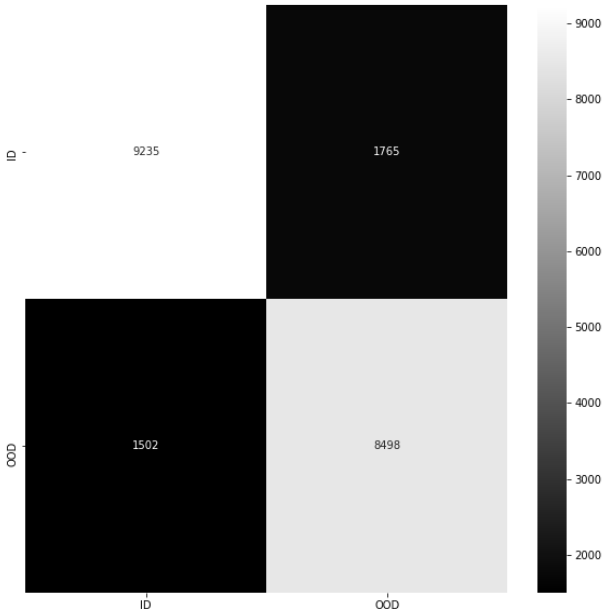


(a) Confusion matrix for the ID and the OOD data separation task

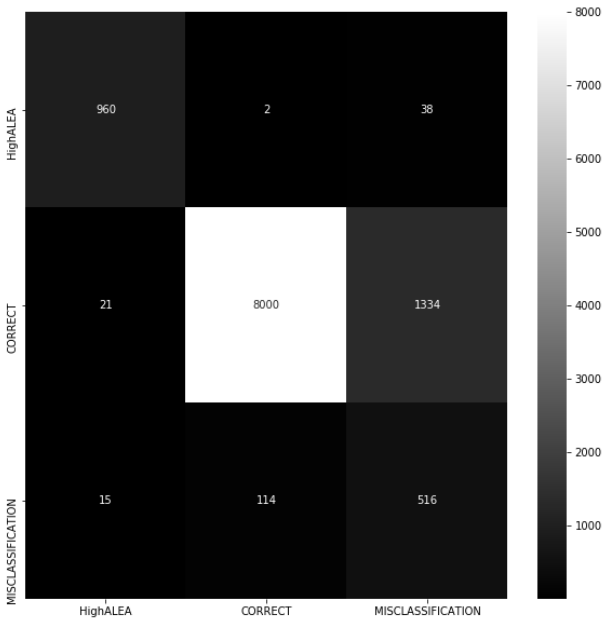


(b) Confusion matrix for the ID data partitioning task

Figure 5.8: Confusion matrices demonstrating the performance of our new classifier with class label 7 and 9 vertically combined boundary data images.



(a) Confusion matrix for the ID and the OOD data separation task



(b) Confusion matrix for the ID data partitioning task

Figure 5.9: Confusion matrices demonstrating the performance of our new classifier with class label 7 and 9 equally weighted boundary data images.

5.3 Results obtained with CIFAR10 dataset

In this section, we will present the results that we obtained with our different experiments by using the CIFAR10 dataset. As mentioned in the subsection 3.3.2, we use the ResNet50 network architecture with this dataset. For training purposes, we use the deep learning framework called *PyTorch* [73] with the following settings:

- Batch size (both training and validation): 64.
- Training epochs: 50.
- Optimizer: Stochastic gradient descent with its momentum set equal to 0.9 and the L2 weight regularization strength set equal to 0.0005.
- Learning rate: We start with an initial learning of 0.01 and keep it constant during the whole training process.
- Number of samples extracted from the logit space Gaussian distribution: 500
- Weights initialization: We use the default initialization techniques used by the different layers in PyTorch.
- Training data augmentations: We standardize the full size images by subtracting the mean and subsequently dividing by the standard deviation[6]. Following that, for some of the randomly selected images, we apply rotations in the range of -10° to 0°.
- Validation data augmentations: We just standardize the full size images by subtracting the mean and subsequently dividing by the standard deviation. The values of the mean as well as the standard deviation obtained are the ones that we obtain from our training data images.

At test-time, we use the following set of settings for obtaining our results:

- Batch size: 64.
- Number of stochastic forward passes for MC dropout: 25.
- Testing data augmentations: We use the full size test-set images at the test-time. Remember, this test-set additionally consists of the boundary data examples as well as the OOD data (TinyImageNet - rescaled to 32x32 spatial resolution). Before feeding these images to our network, we standardize them by subtracting the mean and subsequently dividing by the standard deviation. We use the value of the mean as well as the standard deviation obtained from the training data images.

[6]Note, we compute the mean as well as the standard deviation of all the training data images pixels and use these values in the standardization process.

5.3.1 Horizontal combination of class label 3 and 5 images

In this subsection, we demonstrate the results of our experiment where we combined class label 3 and 5 images, in the test-set, horizontally. Here, the class label 3 is assigned to the images that represent "cats" whereas the class label 5 is assigned to the "dogs" images. We include these horizontally combined images (boundary data examples) in our test-data stream along with the normal CIFAR10 and the OOD TinyImageNet dataset images. Since we create the same horizontal combination of images in our training data as well, we assign a hard label of 3 to half of them and a hard label of 5 to the other half, for training purposes. As with our same MNIST dataset experiment, we expect to see a high aleatoric uncertainty for our horizontally combined images and a high predictive uncertainty as well as the mutual information for the OOD data samples.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	76.4	68.5	87.7
OOD	80.7	80.7	69.4
Misclassified	88.9	88.3	68.9

Table 5.10: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.10, we can observe that for the high aleatoric uncertainty data detection task, we get a considerably high AUROC score for the aleatoric uncertainty. This means that the training boundary data examples that we created as well as the labelling policy which we used for them, managed to successfully make our network learn produce a high value of the aleatoric uncertainty for such examples at test-time. In addition, we can observe a low score for the mutual information which is due to the presence of our high aleatoric uncertainty data in the training set. For the OOD and the misclassified data detection task, we get a similar trend of the scores for the different uncertainties to our MNIST dataset experiments. Since we have got a good score for the aleatoric uncertainty in the high aleatoric data detection task, we observe the performance of our new classifier in this setting as well.

Figure 5.10 shows the performance of our classifier on the partitioning of the data into different categories. In (a), we can observe that we get a high number of false negatives while separating the OOD data whereas the number of false negatives with the ID data are almost similar to what we got with our classifier on the MNIST dataset. In (b), we can observe that our classifier partitions the different ID data classes with an appropriate accuracy. However, the number of false negatives for the high aleatoric uncertainty data are higher as compared to our previous experiments. We tried to improve the performance of our classification tasks by training the main network with our new regularized loss function (stated in the sub-subsection 3.3.1.1). Afterwards, we followed the same set of rules for separating the OOD/ID data and for training our new classifier.

Figure 5.11 shows the confusion matrices that demonstrate the performance of our classifier after training the main network with the regularized loss function. As evident, we now observe a better performance for the OOD and the ID data separation task. The number of false negatives for the OOD class have fallen down from 3310 to 2811 however they have increased a bit for the ID class. But, this increment is not that significant as compared to the decrement which we observe for our OOD class. In addition, we can see that the classification accuracy for the ID

classes has improved a bit too as compared to our previous classifier results that we obtained without training our main network with the regularized loss function.

In summary, we conclude that this experimental setting, which we classified as best performing from our MNIST dataset experiments, indeed transfers well to the other datasets too - in this case, CIFAR10. Additionally, our new regularized loss function shows promising results as what we saw with our same FashionMNIST dataset experiment.

5.3.2 Vertical combination of class label 3 and 5 images

In this subsection, we demonstrate the results of our experiment where we combined class label 3 and 5 images, in the test-set, vertically. We include these vertically combined images (boundary data examples) in our test-data stream along with the normal CIFAR10 and the OOD TinyImageNet dataset images. For the training boundary data examples, we assign the class label 3 to half of them and the class label 5 to the other half. We expect to see a high aleatoric uncertainty for our vertically combined images and a high predictive uncertainty as well as the mutual information for the OOD data samples.

Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	81.0	66.8	90.0
OOD	81.0	80.8	74.3
Misclassified	89.4	88.6	73.2

Table 5.11: AUROC scores obtained with various uncertainty estimates for different data types detection.

In the table 5.11, we can observe that we get a high aleatoric uncertainty AUROC score in the high aleatoric data detection task. Additionally, this score as well as the scores for the other uncertainty estimates are almost similar to what we achieved with our previous horizontal combination of the same class label images. Further, in the OOD and the misclassified data detection task, we observe a similar pattern of the scores to our previous experiment as well. Since we get a good aleatoric uncertainty score in the high aleatoric data detection task, we observe the performance of our classifier in this setting as well. Here, we only show the performance that we got by training our main network without the new regularized loss function[7].

Figure 5.12 shows the performance of our classifier. We can observe that the results are almost similar to our previous experimental setting.

In summary, we conclude that this experiment, which we included in the best performing list of experiments for the MNIST dataset, indeed transfers well to the other complex datasets - in this case, CIFAR10. Similar to our observations with the FashionMNIST and the MNIST dataset, here we can also claim that the aleatoric uncertainty is almost invariant to either the horizontal or the vertical combination of images.

[7]Due to the shortage of time, we could not train our network from scratch with the new regularized loss function.

5.3.3 Equal weights combination of class label 3 and 5 images

In this subsection, we demonstrate the results of our experiment where we combine the class label 3 and 5 images, from the test-set, with equal weights of 0.5 and include them along with the TinyImageNet testing data images in our overall test-data stream. By equal weight combination, we mean that we just add the two different images together by first scaling their pixel values by 0.5. For the training boundary data examples we assign the class label 3 to half of them and the class label 5 to the other half. We expect to observe a high aleatoric uncertainty for our combined images and a high mutual information as well the predictive uncertainty for the OOD data in the test-set.

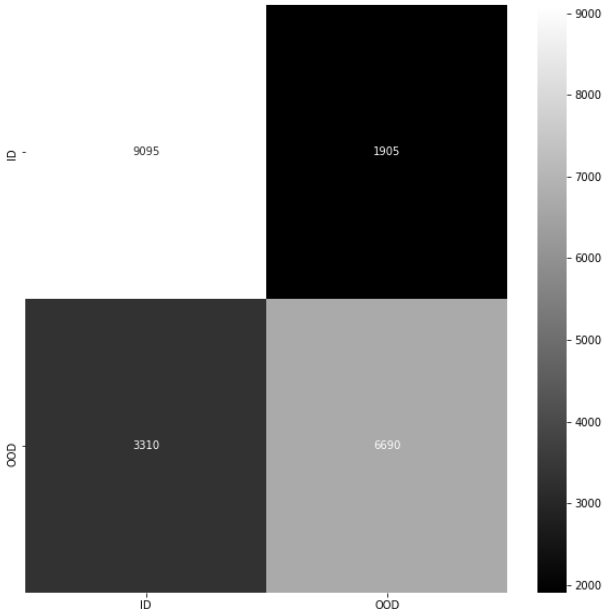
Detection type	Predictive	Mutual Info.	Aleatoric
High aleatoric	78.7	74.4	88.0
OOD	80.8	79.3	72.2
Misclassified	88.5	87.9	69.7

Table 5.12: AUROC scores obtained with various uncertainty estimates for different data types detection.

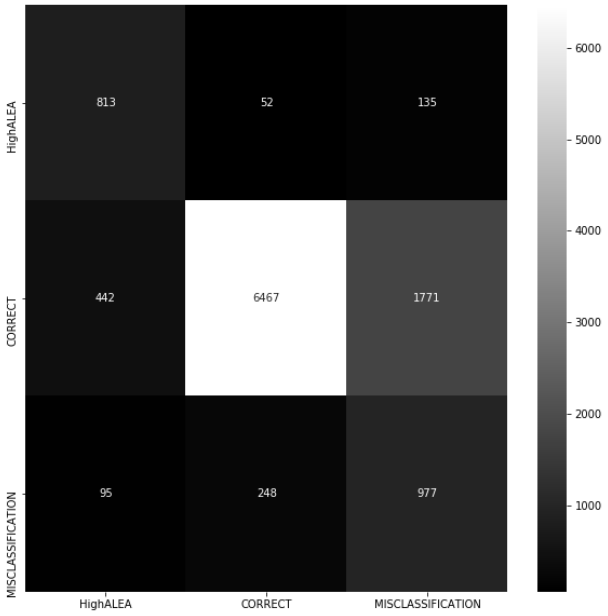
In the table 5.12, we can observe that we get a similar set of scores, for all the uncertainty estimates in our different data type detection tasks, to our previous experiments. Since we get a good aleatoric uncertainty score in the high aleatoric data detection task, we observe the performance of our classifier in this setting as well. Here, we only show the performance that we got by training our main network without the new regularized loss function.

Figure 5.13 shows the performance of our new classifier in a form of the confusion matrices. As evident, the results are in line with our observations from the same FashionMNIST experiment. In comparison to the previous CIFAR10 experimental settings, we can observe that the performance of our classifier is a bit bad comparatively.

In summary, we conclude that this experiment does not perform that well with the CIFAR10 dataset. We were able to get a good aleatoric uncertainty score for the high aleatoric data detection task but the pattern of the scores was not strong enough, as compared to what we got with our previous experimental settings, for the classifier to perform good.

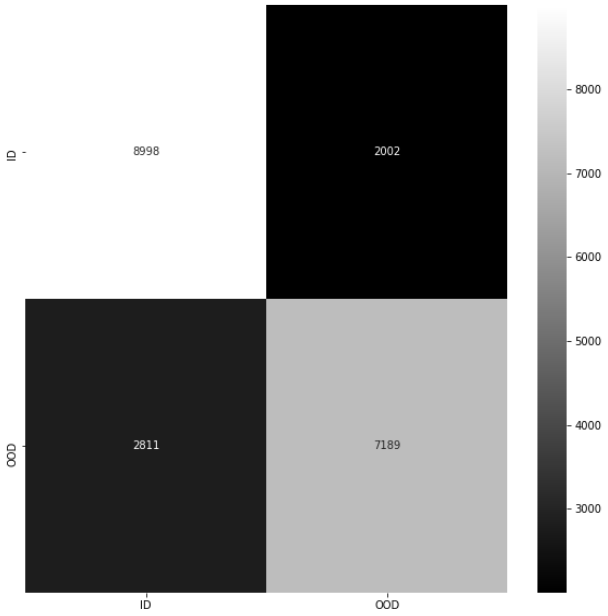


(a) Confusion matrix for the ID and the OOD data separation task

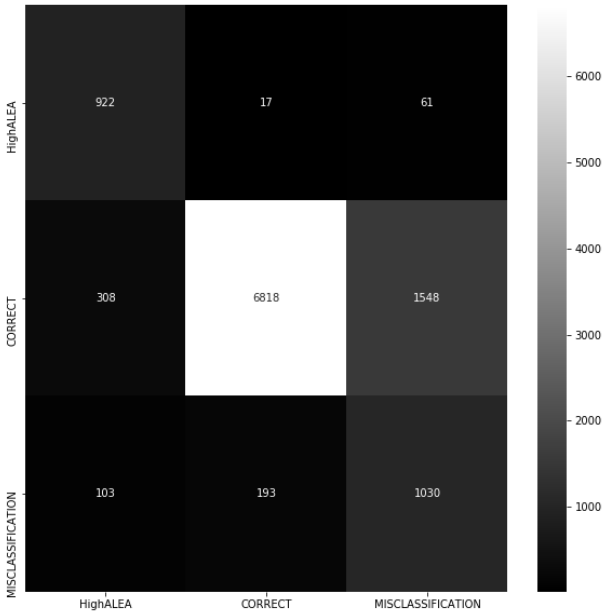


(b) Confusion matrix for the ID data partitioning task

Figure 5.10: Confusion matrices demonstrating the performance of our new classifier with class label 3 and 5 horizontally combined boundary data images.

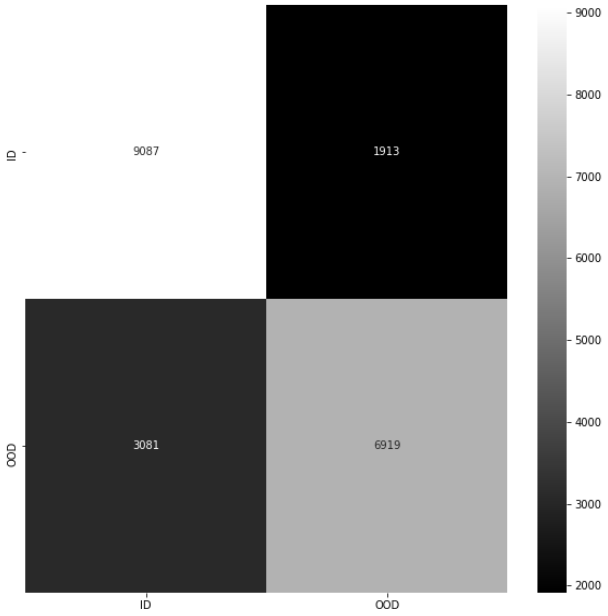


(a) Confusion matrix for the ID and the OOD data separation task

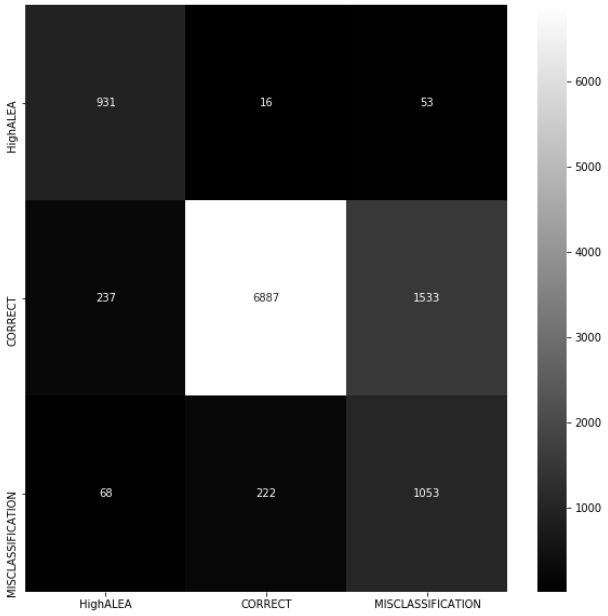


(b) Confusion matrix for the ID data partitioning task

Figure 5.11: Confusion matrices demonstrating the performance of our new classifier, after training the main network with the regularized loss function, with class label 3 and 5 horizontally combined boundary data images.

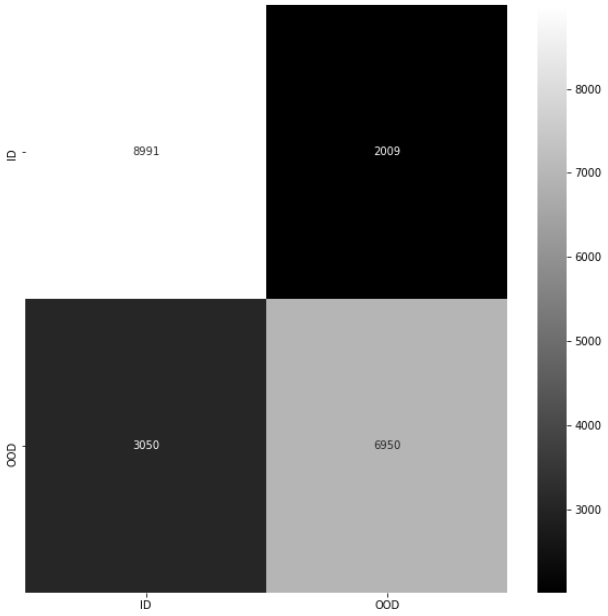


(a) Confusion matrix for the ID and the OOD data separation task

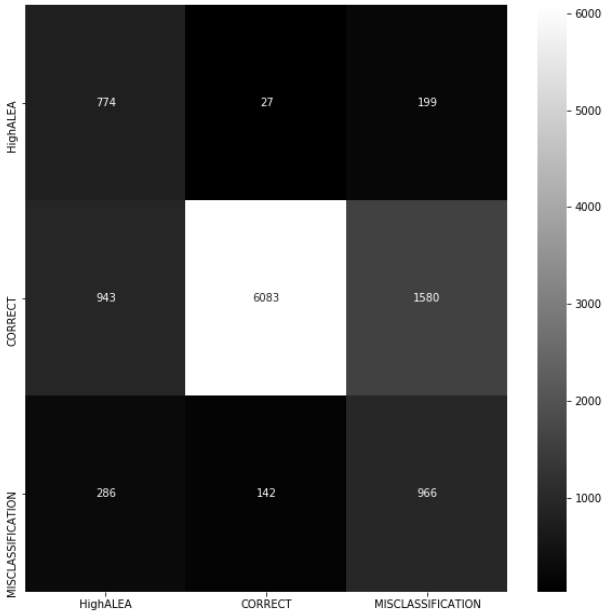


(b) Confusion matrix for the ID data partitioning task

Figure 5.12: Confusion matrices demonstrating the performance of our new classifier with class label 3 and 5 vertically combined boundary data images.



(a) Confusion matrix for the ID and the OOD data separation task



(b) Confusion matrix for the ID data partitioning task

Figure 5.13: Confusion matrices demonstrating the performance of our new classifier with class label 3 and 5 equally weighted boundary data images.

5.4 Deep insights

In the previous sections, we demonstrated the results that we got with the different datasets, in the context of the Image classification task. The main objective of performing these experiments was to answer some of the research questions that we formulated in the subsection 3.1.1. By performing these experiments, we not only got the answers to our questions but we also got a bit more in-depth idea of the working of the method presented by the authors in [1].

Considering the experiments conducted with the MNIST dataset, we showed that if we choose a correct labelling policy for the possible high aleatoric uncertainty data samples, while training our network, we can make our network learn produce a high aleatoric uncertainty for such samples at the test-time. In this regard, we conducted experiments by creating the boundary data images in various styles and trying the two different labelling policies for them. We observed that the vertical as well as the horizontal combination of two different class label images produce almost similar results, with a very minute difference that can be linked with the stochastic training process of the network. Additionally, we discovered that if we assign two different hard labels to these boundary data images, then our network gets confused the most and produce the aleatoric uncertainty estimates that can help us distinguish such samples very easily.

Apart from that, we validated the use of the mutual information metric for representing the epistemic uncertainty. As we know, the predictive uncertainty is a combination of the aleatoric as well as the epistemic uncertainty. Hence, in cases, where we observe a high value for this uncertainty, it gets difficult to determine whether the high value is due to a sample being OOD with the high epistemic uncertainty or ID with the high aleatoric uncertainty. Looking back at our high aleatoric uncertainty data detection task (with any dataset), we observed that we were getting a high score for the aleatoric uncertainty and a low score for the mutual information. But, the predictive uncertainty score was always high. In this case, we knew that it is due to the aleatoric uncertainty but, in practical situations, where we do not know about the nature of the data beforehand, the usage of predictive uncertainty for representing the epistemic uncertainty does not seem like a good idea. Thus, we think that the mutual information is a better metric for quantifying the epistemic uncertainty.

Moving on, we wanted to verify whether the method presented by the authors in [1] is capable of producing uncertainty estimates that correlate with the input data space complexities. For the boundary data examples where we expected this method to give the high aleatoric uncertainty, it indeed produced appropriate results but with a typical confusing labelling policy. In addition, we wanted to observe whether we get a high aleatoric uncertainty for the OOD samples or not. For all of our MNIST dataset experiments with the FashionMNIST OOD samples, we got a high value of this uncertainty for such samples but with our FashionMNIST dataset where we had the MNIST OOD samples, we did not observe the same in our OOD data detection task. The images in the FashionMNIST and the MNIST dataset are both grayscale and have the same spatial resolution however the network architecture which we used for the FashionMNIST dataset (VGG-7) is deeper than the LeNet5 architecture that we used with the MNIST dataset. A deeper network has always more representational power and because of that the VGG-7 network architecture might have produced peaky softmax distributions for the MNIST images. As the aleatoric uncertainty is quantified by the variances of the logit space values, we get to see a lower value of the variances for the logits in the case of peaky softmax distributions.

Finally, as an additional work we demonstrated the performance of our new classifier design with the different experiments. We proved that our uncertainty based classifier can be effectively used in live production settings for making the network aware of the nature of the data. In addition, we demonstrated the efficiency of our new regularized loss function for improving the performance of this classifier. As a side note, we suggest that the main networks must be trained to the point of convergence for getting the best performance from our new classifier design.

6 Future Research

In this work, we presented the results from our different experiments in the context of the Semantic segmentation and the Image classification task. For all of our experiments, we chose the method presented by the authors in [1]. The main purpose of performing these different experiments was to clear out some of the ambiguities that were left behind by the empirical observations of the authors.

For our Semantic segmentation experiments, we used two different state-of-the-art network architectures and combined their working with some of the other findings, made by different authors, for answering some important research questions. However, in our Image classification experiments, we firstly performed an in-depth analysis of the method [1] which we were using for quantifying the different uncertainties. Later, based on this analysis we proposed a new classifier design and demonstrated its working in different experimental settings. In addition, for the complex datasets and deeper networks, we formulated a new loss function for training the main network that could help in improving the performance of our classifier at later stages.

As a matter of future research, we believe that it would be interesting to observe whether the experiments that we performed in the case of the Image classification task would hold for the Semantic segmentation as well or not. This would additionally open up the question for the way we should create the boundary data examples, for this task, and choose an appropriate labelling policy for them. Apart from that, it would also be exciting to observe whether our new classifier is able to classify every pixel of an image, into different categories, based on just the different uncertainty estimates or not.

Bibliography

[1] A. Kendall and Y. Gal, "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" In *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 5574–5584.

[2] J. Mukhoti and Y. Gal, "Evaluating Bayesian Deep Learning Methods for Semantic Segmentation," *CoRR*, vol. abs/1811.12709, 2018.

[3] Y. LeCun, B. Boser, J. S. Denker, and D. Henderson, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Comput.*, MIT Press, vol. 1 (4), pp. 541–551, Dec. 1989.

[4] Y. Gal, "Uncertainty in Deep Learning," PhD thesis, University of Cambridge, 2016.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *Proceedings of the 2015 IEEE International Conference on Computer Vision*, ser. ICCV '15, USA: IEEE Computer Society, 2015, pp. 1026–1034.

[6] P.-Y. Huang, W. T. Hsu, C.-Y. Chiu, T.-F. Wu, and M. Sun, "Efficient Uncertainty Estimation for Semantic Segmentation in Videos," *CoRR*, vol. abs/1807.11037, 2018.

[7] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep Ordinal Regression Network for Monocular Depth Estimation," *CoRR*, vol. abs/1806.02446, 2018.

[8] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid Scene Parsing Network," *CoRR*, vol. abs/1612.01105, 2016.

[9] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," *CoRR*, vol. abs/1802.02611, 2018.

[10] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[11] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[12] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 (Canadian Institute for Advanced Research),"

[13] L. Jia and Y. Sun, "Digital Recognition Based on Improved LENET Convolution Neural Network," in *Proceedings of the 2018 International Conference on Machine Learning Technologies*, ser. ICMLT '18, Jinan, China: Association for Computing Machinery, 2018, pp. 24–28.

[14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, vol. abs/1512.03385, 2015.

[16] C. Ma, Y. Li, and J. M. Hernandez-Lobato, "Variational implicit processes," *International Conference on Machine Learning*, pp. 4222–4233, 2019.

[17] D. J. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Computation*, vol. 4 (3), pp. 448–472, 1992.

[18] R. M. Neal, "Bayesian learning for neural networks," PhD thesis, University of Toronto, 1995.

[19] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid Monte Carlo," *Physics Letters B*, vol. 195 (2), pp. 216–222, 1987.

[20] M. Welling and Y. W. Teh, "Bayesian Learning via Stochastic Gradient Langevin Dynamics," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11, Bellevue, Washington, USA: Omnipress, 2011, pp. 681–688.

[21] M. D. Hoffman and A. Gelman, "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo," *CoRR*, vol. abs/1111.4246, 2011.

[22] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 1613–1622.

[23] J. Shi, S. Sun, and J. Zhu, "Kernel Implicit Variational Inference," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.

[24] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22 (1), pp. 79–86, 1951.

[25] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An Introduction to Variational Methods for Graphical Models," *Mach. Learn.*, Kluwer Academic Publishers, vol. 37 (2), pp. 183–233, Nov. 1999.

[26] G. E. Hinton and D. 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*, ser. COLT '93, Santa Cruz, California, USA: Association for Computing Machinery, 1993, pp. 5–13.

[27] D. G. Barber and C. M. Bishop, "Ensemble learning in Bayesian neural networks," 1998.

[28] A. Graves, "Practical Variational Inference for Neural Networks," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, ser. NIPS'11, Granada, Spain: Curran Associates Inc., 2011, pp. 2348–2356.

[29] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic Variational Inference," *J. Mach. Learn. Res.*, JMLR.org, vol. 14 (1), pp. 1303–1347, May 2013.

[30] J. Paisley, D. M. Blei, and M. I. Jordan, "Variational Bayesian Inference with Stochastic Search," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12, Edinburgh, Scotland: Omnipress, 2012, pp. 1363–1370.

[31] D. P. Kingma, T. Salimans, and M. Welling, "Variational Dropout and the Local Reparameterization Trick," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 2575–2583.

[32] M. Opper and C. Archambeau, "The Variational Gaussian Approximation Revisited," *Neural Comput.*, MIT Press, vol. 21 (3), pp. 786–792, Mar. 2009.

[33] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, 2016, pp. 1050–1059.

[34] Y. Gal and Z. Ghahramani, "Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference," 2015.

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *The Journal of Machine Learning Research*, JMLR.org, vol. 15 (1), pp. 1929–1958, Jan. 2014.

[36] N. Tishby, E. Levin, and S. A. Solla, "Consistent inference of probabilities in layered networks: predictions and generalizations," *International 1989 Joint Conference on Neural Networks*, 403–409 vol.2, 1989.

[37] E. D. C. Carvalho, R. Clark, A. Nicastro, and P. H. J. Kelly, "Scalable Uncertainty for Computer Vision with Functional Variational Inference," 2020.

[38] D. Hafner, D. Tran, T. Lillicrap, A. Irpan, and J. Davidson, "Noise Contrastive Priors for Functional Uncertainty," 2018.

[39] S. Sun, G. Zhang, J. Shi, and R. B. Grosse, "Functional Variational Bayesian Neural Networks," *CoRR*, vol. abs/1903.05779, 2019.

[40] Z. Wang, T. Ren, J. Zhu, and B. Zhang, "Function Space Particle Optimization for Bayesian Neural Networks," 2019.

[41] A. Der Kiureghian and O. Ditlevsen, "Aleatoric or epistemic? Does it matter?" *Structural Safety*, Elsevier, vol. 31 (2), pp. 105–112, 2009.

[42] M. Bojarski, D. D. Testa, D. Dworakowski, and B. Firner, "End to End Learning for Self-Driving Cars," *CoRR*, vol. abs/1604.07316, 2016.

[43] M. Cordts, M. Omran, S. Ramos, and T. Rehfeld, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[44] A. Blake, R. Curwen, and A. Zisserman, "A Framework for Spatiotemporal Control in the Tracking of Visual Contours," *Int. J. Comput. Vision*, Kluwer Academic Publishers, vol. 11 (2), pp. 127–145, Oct. 1993.

[45] A. Korattikara, V. Rathod, K. Murphy, and M. Welling, "Bayesian Dark Knowledge," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 3438–3446.

[46] D. Bouchacourt, M. P. Kumar, and S. Nowozin, "DISCO Nets: Dissimilarity Coefficient Networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16, Barcelona, Spain: Curran Associates Inc., 2016, pp. 352–360.

[47] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6405–6416.

[48] J. Postels, F. Ferroni, H. Coskun, N. Navab, and F. Tombari, "Sampling-free Epistemic Uncertainty Estimation Using Approximated Variance Propagation," 2019.

[49] J. Gast and S. Roth, "Lightweight Probabilistic Deep Networks," *CoRR*, vol. abs/1805.11327, 2018.

[50] T. P. Minka and R. Picard, "A Family of Algorithms for Approximate Bayesian Inference," PhD thesis, Massachusetts Institute of Technology, USA, 2001, AAI0803033.

[51] X. Boyen and D. Koller, "Tractable Inference for Complex Stochastic Processes," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'98, Madison, Wisconsin: Morgan Kaufmann Publishers Inc., 1998, pp. 33–42.

[52] H. Zhao, F. Liu, L. Li, and C. Luo, "A Novel Softplus Linear Unit for Deep Convolutional Neural Networks," *Applied Intelligence*, Kluwer Academic Publishers, vol. 48 (7), pp. 1707–1720, Jul. 2018.

[53] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely Connected Convolutional Networks," *CoRR*, vol. abs/1608.06993, 2016.

[54] S. Jégou, M. Drozdzal, D. Vázquez, A. Romero, and Y. Bengio, "The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation," *CoRR*, vol. abs/1611.09326, 2016.

[55] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and Recognition Using Structure from Motion Point Clouds," in *ECCV*, 2008, pp. 44–57.

[56] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2015.

[57] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 448–456.

[58] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On Calibration of Modern Neural Networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, pp. 1321–1330.

[59] M. P. Naeini, G. F. Cooper, and M. Hauskrecht, "Obtaining Well Calibrated Probabilities Using Bayesian Binning," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15, Austin, Texas: AAAI Press, 2015, pp. 2901–2907.

[60] Y. Gal, J. Hron, and A. Kendall, "Concrete Dropout," in *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 2017, pp. 3581–3590.

[61] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding," *CoRR*, vol. abs/1511.02680, 2015.

[62] F. K. Gustafsson, M. Danelljan, and T. B. Schön, "Evaluating Scalable Bayesian Deep Learning Methods for Robust Computer Vision," *CoRR*, vol. abs/1906.01620, 2019.

[63] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Conference on Computer Vision and Pattern Recognition*, 2012.

[64] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient Object Localization Using Convolutional Networks," *CoRR*, vol. abs/1411.4280, 2014.

[65] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *CoRR*, vol. abs/1610.02357, 2016.

[66] A. Giusti, D. C. Ciresan, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Fast Image Scanning with Deep Max-Pooling Convolutional Neural Networks," *CoRR*, vol. abs/1302.1700, 2013.

[67] Z. Zhang and M. R. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels," *CoRR*, vol. abs/1805.07836, 2018.

[68] D. Opitz and R. Maclin, "Popular Ensemble Methods: An Empirical Study," *J. Artif. Int. Res.*, AI Access Foundation, vol. 11 (1), pp. 169–198, Jul. 1999.

[69] A. Malinin and M. Gales, "Predictive Uncertainty Estimation via Prior Networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, Montréal, Canada: Curran Associates Inc., 2018, pp. 7047–7058.

[70] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority over-Sampling Technique," *J. Artif. Int. Res.*, AI Access Foundation, vol. 16 (1), pp. 321–357, Jun. 2002.

[71] T. Elhassan, A. M, A.-M. F, and M. Shoukri, "Classification of Imbalance Data using Tomek Link (T-Link) Combined with Random Under-sampling (RUS) as a Data Reduction Method," *Global Journal of Technology and Optimization*, vol. 01, Jan. 2016.

[72] Y. Le and X. Yang, "Tiny ImageNet Visual Recognition Challenge," 2015.

[73] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Automatic differentiation in PyTorch," in *NIPS-W*, 2017.

List of Figures

List of Tables