Torch-Uncertainty Overview

Gianni Franchi & Adrien Lafage & Firas Gabetni & Olivier Laurent

July 1, 2025

Introduction

Link to Example Notebook

- Focus: Small-scale image classification using the MNIST dataset.
- Dataset: MNIST - 70,000 grayscale images of handwritten digits (0 to 9).
- Goal: Classify the images into their respective digits.
- Metrics:
  - Accuracy
  - Brier Score
  - Calibration Error

Key Libraries

- **PyTorch Lightning**: Simplifies training and evaluation.
- **TorchUncertainty**: Tools to quantify Deep Learning models' uncertainty.
- **TorchVision**: For dataset and transformations.

Visualizing In-Distribution and Out-of-Distribution Data

- In-Distribution Data: MNIST
- OOD Data: FashionMNIST

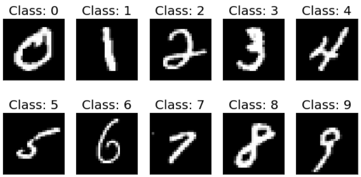


Figure: MNIST

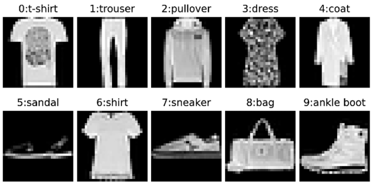


Figure: FashionMNIST

Uncertainty Evaluation

- Accuracy
- Negative log-likelihood
- OOD Detection:
  - AUPR, AUROC (Higher = better).
  - FPR95 (Lower = better).
- Calibration:
  - Expected Calibration Error (ECE).
  - Adaptive ECE.

Setting up a simple LeNet: Part 1

- Initialize the model using PyTorch.

```python
from torch import nn

class LeNet(nn.Module):
    def __init__(self, in_channels: int, num_classes: int):
        super().__init__()
        self.conv1 = nn.Conv2d(
            in_channels=in_channels, out_channels=6,
            ↪  kernel_size=(5,5)
        )
        self.conv2 = nn.Conv2d(
            in_channels=6, out_channels=16, kernel_size=(5,5)
        )
        self.fc1 = nn.Linear(in_features=256, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=84)
        self.fc3 = nn.Linear(in_features=84,
        ↪  out_features=num_classes)

        self.activation = nn.ReLU()
        self.pooling = nn.AdaptativeAvgPooling2d((4,4))
```

Setting up a simple LeNet: Part 2

- Define the `forward()` function.

```
from torch import nn, Tensor
import torch.nn.functional as F

class LeNet(nn.Module):
    # ...
    def forward(self, x: Tensor) -> Tensor:
        out = self.activation(self.conv1)
        out = F.max_pool2d(out, 2)
        out = self.activation(self.conv2)
        out = F.max_pool2d(out, 2)
        out = self.pooling(out)
        out = out.flatten(1)
        out = self.activation(self.fc1(out))
        out = self.activation(self.fc2(out))
        return self.fc3(out)

model = LeNet(in_features=1, num_classes=10)
```

Setting up Deep Ensembles

- Ensembles of LeNet for higher accuracy and better uncertainty estimation.
- Leverage the model wrapper from `TU`: `deep_ensembles()`.

```python
from torch_uncertainty.models import deep_ensembles
from torch_uncertainty.models.lenet import lenet

# Create ensemble of 4 LeNet models
model = deep_ensembles(
    lenet(in_channels=1, num_classes=10),
    num_estimators=4,
    task="classification",
    reset_model_parameters=True,
)
```

Setting up Packed-Ensembles

- Efficient ensembling of LeNet models for lower memory consumption.

```python
from torch_uncertainty.models.lenet import packed_lenet
# Create a Packed LeNet ensemble of 4 networks and a width
↪  expansion factor 2.
model = packed_lenet(
    in_channels=1, num_classes=10, num_estimators=4, alpha=2,
)
```

- Under the hood, we have only replaced the `Conv2d` and `Linear` layers by the `PackedConv2d` and `PackedLinear` layers (see example below). The `forward` method is left unchanged.

```python
from torch_uncertainty.layers import PackedConv2d, PackedLinear

self.conv2 = PackedConv2d(
    in_channels=6, out_channels=16, kernel_size=(5,5),
    num_estimators=4, alpha=2,
)
```

Setting up a Bayesian Neural Network

- Initialize a Bayesian LeNet which uses the layers: BayesLinear and BayesConv2d.

```python
from torch_uncertainty.models.lenet import bayesian_lenet

# Create bayesian LeNet model that uses 16 samples.
model = bayesian_lenet(
    in_channels=1, num_classes=10, num_samples=16,
)
```

- Under the hood, we have only replaced the Conv2d and Linear layers by the BayesConv2d and BayesLinear layers (see example below). The forward method is left unchanged.

```python
from torch_uncertainty.layers import BayesConv2d, BayesLinear

self.conv2 = BayesConv2d(
    in_channels=6, out_channels=16, kernel_size=(5,5),
)
```

A single training pipeline

- Train your models on classification tasks using `TUTrainer` and `ClassificationRoutine`.

```python
from torch_uncertainty.utils import TUTrainer
from torch_uncertainty.routines import ClassificationRoutine

# Create the trainer that will handle the training
trainer = TUTrainer(accelerator=device, max_epochs=max_epochs)

# The routine is a wrapper of the model that contains the
↪   training logic with the metrics, etc
routine = ClassificationRoutine(
    num_classes=10,
    model=model,
    loss=nn.CrossEntropyLoss(),
    optim_recipe=optim_recipe(model),
    eval_ood=True,
    is_ensemble=True  # False if not
)

# In practice, avoid performing the validation on the test set
trainer.fit(routine, train_dataloaders=train_dl,
↪   val_dataloaders=test_dl)
```

Define Post-Processing methods

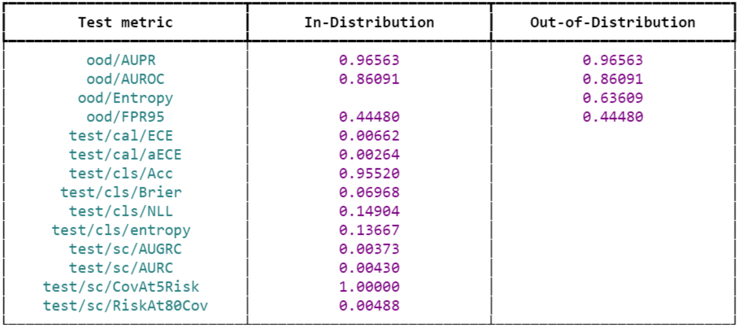
- `ClassificationRoutine` allows to pass post-processing methods.
- Example: Laplace approximation for uncertainty quantification.

```python
from torch_uncertainty.post_processing import LaplaceApprox

laplace_approx = LaplaceApprox(
    task="classification",
    weight_subset="last_layer",
    hessian_struct="kron",
    pred_type="glm",
    link_approx="probit",
    batch_size=256,
)

routine = ClassificationRoutine(
    # ...
    post_processing = laplace_approx,
)
```

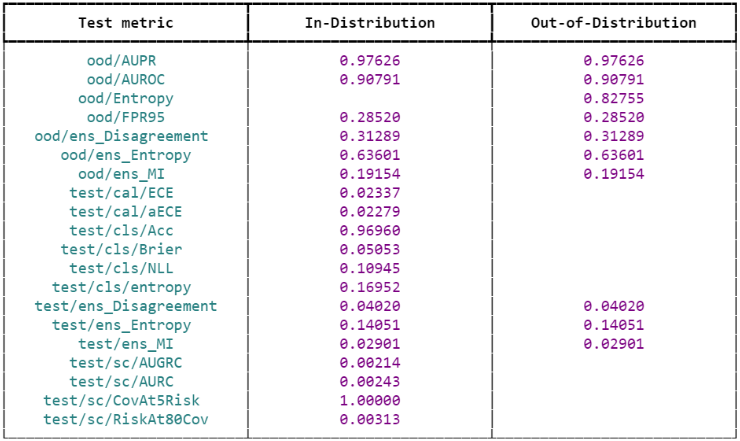
Results - LeNet



Test metric	In-Distribution	Out-of-Distribution
ood/AUPR	0.96563	0.96563
ood/AUROC	0.86091	0.86091
ood/Entropy		0.63609
ood/FPR95	0.44480	0.44480
test/cal/ECE	0.00662	
test/cal/aECE	0.00264	
test/cls/Acc	0.95520	
test/cls/Brier	0.06968	
test/cls/NLL	0.14904	
test/cls/entropy	0.13667	
test/sc/AUGRC	0.00373	
test/sc/AURC	0.00430	
test/sc/CovAt5Risk	1.00000	
test/sc/RiskAt80Cov	0.00488	

Figure: Performance on the evaluation set

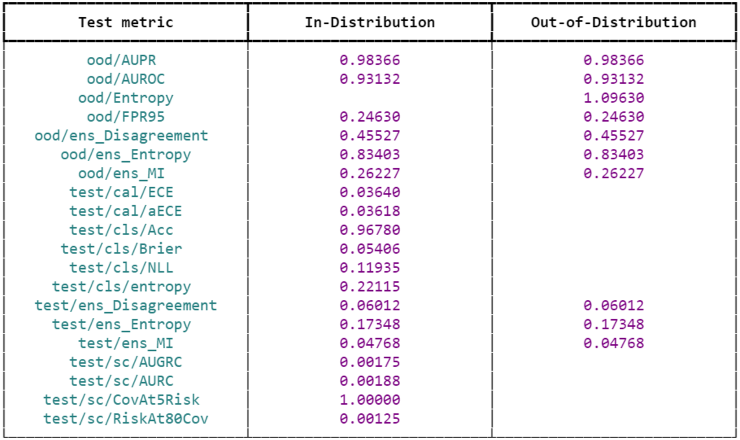
Results - Deep Ensembles LeNet



Test metric	In-Distribution	Out-of-Distribution
ood/AUPR	0.97626	0.97626
ood/AUROC	0.90791	0.90791
ood/Entropy		0.82755
ood/FPR95	0.28520	0.28520
ood/ens_Disagreement	0.31289	0.31289
ood/ens_Entropy	0.63601	0.63601
ood/ens_MI	0.19154	0.19154
test/cal/ECE	0.02337	
test/cal/aECE	0.02279	
test/cls/Acc	0.96960	
test/cls/Brier	0.05053	
test/cls/NLL	0.10945	
test/cls/entropy	0.16952	
test/ens_Disagreement	0.04020	0.04020
test/ens_Entropy	0.14051	0.14051
test/ens_MI	0.02901	0.02901
test/sc/AUGRC	0.00214	
test/sc/AURC	0.00243	
test/sc/CovAt5Risk	1.00000	
test/sc/RiskAt80Cov	0.00313	

Figure: Performance on the evaluation set

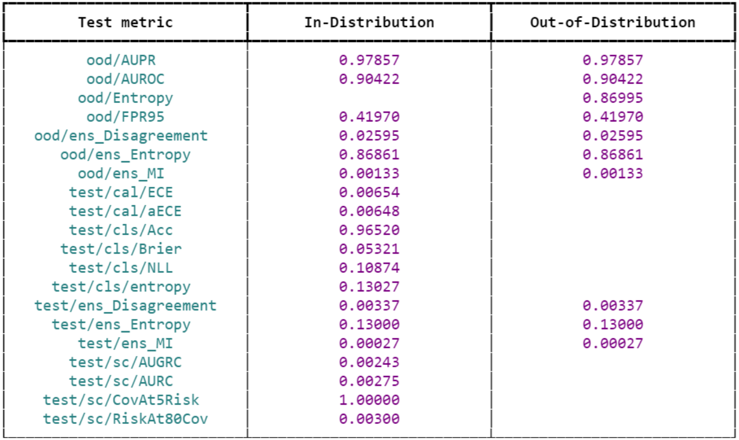
Results - Packed-Ensembles LeNet



Test metric	In-Distribution	Out-of-Distribution
ood/AUPR	0.98366	0.98366
ood/AUROC	0.93132	0.93132
ood/Entropy		1.09630
ood/FPR95	0.24630	0.24630
ood/ens_Disagreement	0.45527	0.45527
ood/ens_Entropy	0.83403	0.83403
ood/ens_MI	0.26227	0.26227
test/cal/ECE	0.03640	
test/cal/aECE	0.03618	
test/cls/Acc	0.96780	
test/cls/Brier	0.05406	
test/cls/NLL	0.11935	
test/cls/entropy	0.22115	
test/ens_Disagreement	0.06012	0.06012
test/ens_Entropy	0.17348	0.17348
test/ens_MI	0.04768	0.04768
test/sc/AUGRC	0.00175	
test/sc/AURC	0.00188	
test/sc/CovAt5Risk	1.00000	
test/sc/RiskAt80Cov	0.00125	

Figure: Performance on the evaluation set

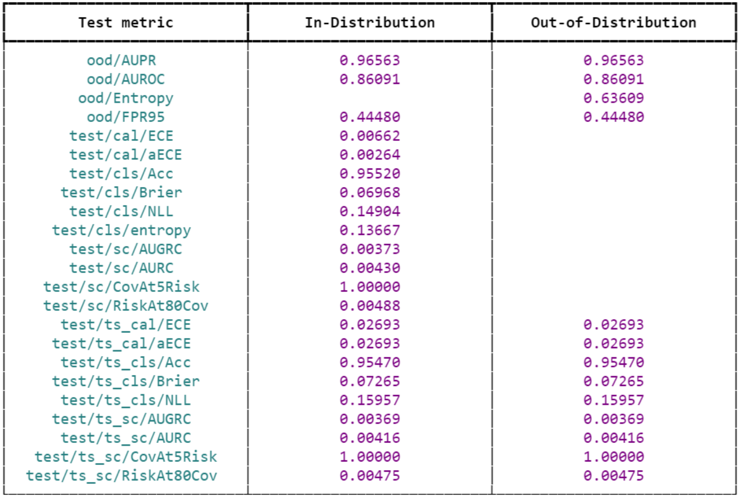
Results - Bayesian LeNet



Test metric	In-Distribution	Out-of-Distribution
ood/AUPR	0.97857	0.97857
ood/AUROC	0.90422	0.90422
ood/Entropy		0.86995
ood/FPR95	0.41970	0.41970
ood/ens_Disagreement	0.02595	0.02595
ood/ens_Entropy	0.86861	0.86861
ood/ens_MI	0.00133	0.00133
test/cal/ECE	0.00654	
test/cal/aECE	0.00648	
test/cls/Acc	0.96520	
test/cls/Brier	0.05321	
test/cls/NLL	0.10874	
test/cls/entropy	0.13027	
test/ens_Disagreement	0.00337	0.00337
test/ens_Entropy	0.13000	0.13000
test/ens_MI	0.00027	0.00027
test/sc/AUGRC	0.00243	
test/sc/AURC	0.00275	
test/sc/CovAt5Risk	1.00000	
test/sc/RiskAt80Cov	0.00300	

Figure: Performance on the evaluation set

Results - Laplace Approximation on LeNet



Test metric	In-Distribution	Out-of-Distribution
ood/AUPR	0.96563	0.96563
ood/AUROC	0.86091	0.86091
ood/Entropy		0.63609
ood/FPR95	0.44480	0.44480
test/cal/ECE	0.00662	
test/cal/aECE	0.00264	
test/cls/Acc	0.95520	
test/cls/Brier	0.06968	
test/cls/NLL	0.14904	
test/cls/entropy	0.13667	
test/sc/AUGRC	0.00373	
test/sc/AURC	0.00430	
test/sc/CovAt5Risk	1.00000	
test/sc/RiskAt80Cov	0.00488	
test/ts_cal/ECE	0.02693	0.02693
test/ts_cal/aECE	0.02693	0.02693
test/ts_cls/Acc	0.95470	0.95470
test/ts_cls/Brier	0.07265	0.07265
test/ts_cls/NLL	0.15957	0.15957
test/ts_sc/AUGRC	0.00369	0.00369
test/ts_sc/AURC	0.00416	0.00416
test/ts_sc/CovAt5Risk	1.00000	1.00000
test/ts_sc/RiskAt80Cov	0.00475	0.00475

Figure: Performance on the evaluation set

Torch-Uncertainty Overview: UQ Methods

- Deep Evidential (Regression & Classification)
- Beta NLL in Deep Regression
- Variational Inference BNN
- Deep Ensembles
- Monte-Carlo Dropout
- Stochastic Weight Averaging (SWA)
- Stochastic Weight Averaging Gaussian (SWAG)
- CheckpointEnsemble
- SnapshotEnsemble
- BatchEnsemble
- Masksembles
- MIMO
- Packed-Ensembles
- LPBNN
- Conformal Prediction
- OOD Detection Criteria

Torch-Uncertainty Overview: Metrics

- *Classification*
  - **Proper Scores**: `BrierScore`, `CategoricalNLL`.
  - **Out-of-Distribution Detection**: `AURC`, `FPRx`, `FPR95`.
  - **Selective Classification**: `AUGRC`, `RiskAtxCov`, `RiskAt80Cov`, `CovAtxRisk`, `CovAt5Risk`.
  - **Calibration**: `AdaptativaCalibrationError`, `CalibrationError`.
  - **Others**: `GroupingLoss`.
- *Regression*: `DistributionNLL`, `Log10`, `MeanAbsoluteErrorInverse`, `MeanGTRelativeAbsoluteError`, `MeanGTRelativeSquaredError`, `MeanSquaredErrorInverse`, `MeanSquaredLogError`, `SILog`, `ThresholdAccuracy`.
- *Segmentation*: `MeanIntersectionOverUnion`

Useful links

Example Notebook

TorchUncertainty

GitHub

Preprint

Thanks for your attention!

Uncertainty Quantification in Deep Learning

Gianni FRANCHI

U2IS, ENSTA Paris

Tutoriel PFIA 2025

Planning of the tutorial

Planning of the tutorial

- 90 min - Sections 1 to 3
- 90 min - Sections 4 to 5
- 30 min - TorchUncertainty demo
- 30 min - Question Answering

Tutorial's organisers

Gianni Franchi

Olivier Laurent

Adrien Lafage

Firas Gabetni

Plan

Uncertainty: The Physicist's Perspective

Foundation in Physics

- Physicists seek to model and understand complex phenomena, but its roots are deeply tied to the fundamental uncertainties present in the physical world.
- Physics provides the groundwork for our understanding of natural processes, emphasizing the need to deal with uncertainty and variability in experimental observations.

Linking Experiments to Models

- Physicists often perform experiments to collect data and formulate models that describe the behavior of the physical systems.
- These models inherently carry uncertainties due to measurement errors, experimental conditions, and the stochastic nature of certain phenomena.

Example with Hooke's Law

- Hooke's Law describes the behavior of materials when subjected to stretching or compressing forces.
- Named after Robert Hooke, a 17th-century physicist.
- Fundamental principle in elasticity.

$$F = -k \cdot \Delta L \tag{1}$$

- F is the restoring force.
- k is the spring constant.
- ΔL is the displacement from the equilibrium position.

Linear Elastic Region

- Hooke's Law is valid within the linear elastic region of a material.
- Beyond this region, permanent deformation or failure may occur.

Example with Hooke's Law

$$F = -k \cdot \Delta L \tag{2}$$

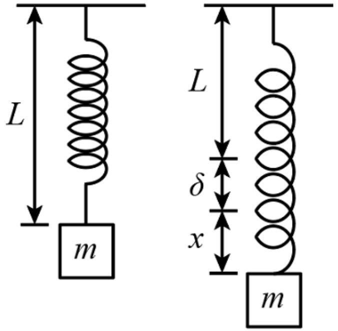


Figure: Experiment: Find the spring constant k

Uncertainty: The Physicist's Perspective

Sources of Uncertainty

- **Parameter Uncertainty:** Many physical models involve parameters that may not be precisely known. Uncertainty in these parameters can stem from variations in material properties, manufacturing tolerances, or imprecise calibration of instruments.
- **Initial Conditions Measurement:** The initial state of a system is often a critical factor in physical models. Uncertainty in the initial conditions can arise due to measurement errors or incomplete knowledge of the system's state.
- **Dynamic Measurement Error:** Experimental measurements are prone to errors, whether they are due to limitations in measurement instruments, environmental conditions, or human factors.
- **Model Simplifications:** Physical models often involve simplifications and assumptions to make the mathematical representation more tractable. These simplifications may not perfectly capture the complexity of the real-world system.

Uncertainty: The Physicist's Perspective

Sources of Uncertainty

- **Numerical Approximations:** When using numerical methods to solve physical equations, the choice of discretization methods and grid resolution introduces numerical errors. These errors can propagate and amplify, leading to uncertainty in the model results.
- **Model Formulation:** The choice of mathematical equations and the model structure itself introduces uncertainty. Different formulations may yield different predictions, and selecting the most appropriate model can be challenging.
- **Stochastic Nature of Physical Processes:** Some physical processes are inherently stochastic (random) rather than deterministic. Quantum mechanics, for example, involves inherent uncertainties at the fundamental level.

Sources of Uncertainty in Machine Learning

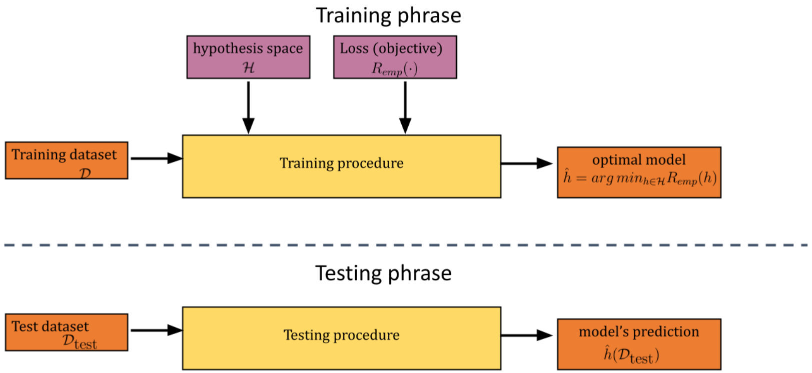
Uncertainty in Predictions

- Machine learning models, while powerful, are not immune to uncertainty.
- Understanding and quantifying sources of uncertainty is crucial for model reliability and decision-making.

Key Question

- What are the primary sources of uncertainty in machine learning, and how do they impact model performance and interpretability?

Sources of Uncertainty in Machine Learning


Training phrase
hypothesis space $\mathcal{H}$
Loss (objective) $R_{emp}(\cdot)$
Training dataset $\mathcal{D}$
Training procedure
optimal model $\hat{h} = arg\,min_{h \in \mathcal{H}} R_{emp}(h)$
Testing phrase
Test dataset $\mathcal{D}_{\text{test}}$
Testing procedure
model's prediction $\hat{h}(\mathcal{D}_{\text{test}})$

Sources of Uncertainty in Machine Learning

Data based

- **Data Quality:** Inaccurate or incomplete training data can introduce uncertainty, affecting model generalization.
- **Data Variability:** Natural variations in the data may lead to different model outcomes, especially in dynamic environments.

Model based

- **Algorithmic Complexity:** Complex models may capture noise in the data, leading to uncertainty in predictions.
- **Hyperparameter Sensitivity:** Small changes in hyperparameters may result in significantly different model behavior.

Sources of Uncertainty in Machine Learning

Deployment and External Factors

- **Distribution Shift:** Differences between training and deployment data can introduce uncertainty, especially in real-world applications.
- **External Influences:** Changes in the environment, user behavior, or external events can impact model performance.

Training Process: Limited Training Data

- **Small Datasets:** Insufficient data may lead to overfitting or underfitting, introducing uncertainty in the model's ability to generalize.
- **Data Imbalance:** Unequal representation of classes can result in biased models with uncertain predictions for underrepresented classes.

Sources of Uncertainty in Machine Learning

Training Dynamics

- **Non-Convex Optimization:** The non-convex nature of many optimization problems in machine learning can result in multiple local minima, impacting the model's final state.
- **Stochastic Processes:** Some machine learning tasks involve inherent randomness, leading to inherent uncertainty in predictions.
- **Stochastic Gradient Descent:** The randomness introduced by stochastic optimization algorithms contributes to training uncertainty.

Sources of Uncertainty in Machine Learning

Dataset notations

- We consider that we have a training dataset

$$\mathcal{D} := \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y} \ , \tag{3}$$

- $(\mathbf{x}_i, y_i)$ are assumed to be independent and identically distributed (i.i.d.) according to some unknown probability measure $P_{\mathcal{X}\times\mathcal{Y}}$ on $\mathcal{X} \times \mathcal{Y}$

Model notations

- We denote Ω the set of all possible models.
- We denote $\mathcal{H}$ a subset of possible models.
- We denote $h(\cdot) \in \mathcal{H}$ a possible model that could be a Deep Neural Network (DNN).
- We denote $f_\omega(\cdot) \in \mathcal{H}$ a DNN model with weights ω.

Sources of Uncertainty in Machine Learning

Model True objective

The goal of Machine Learning is to induce a hypothesis $h^* \in \mathcal{H}$ with low risk (expected loss)

$$R(h) := \int_{\mathcal{X}\times\mathcal{Y}} \ell(h(\boldsymbol{x}), y)\, d\, P_{\mathcal{X}\times\mathcal{Y}}(\boldsymbol{x}, y) \ . \tag{4}$$

$$h^* := \arg\min_{h\in\mathcal{H}} R(h)\, . \tag{5}$$

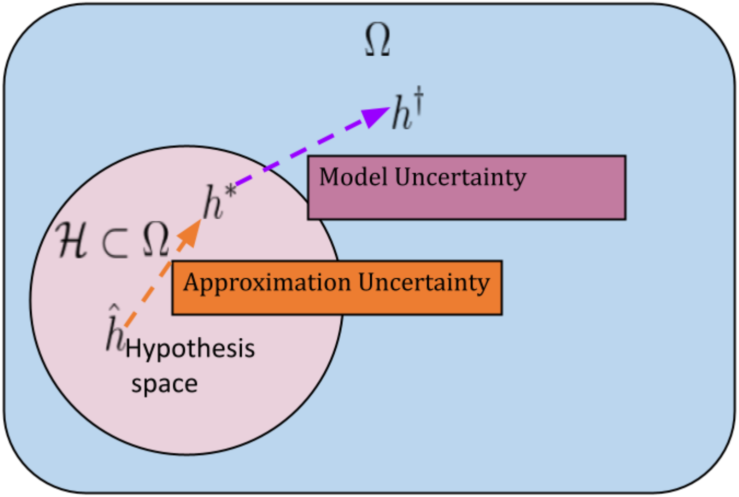
Model Empirical objective

The goal of machine learning is to induce a hypothesis $h^* \in \mathcal{H}$ with low risk (empirical loss)

$$R_{emp}(h) := \frac{1}{N}\sum_{i=1}^{N} \ell(h(\boldsymbol{x}_i), y_i) \ , \tag{6}$$

$$\hat{h} := \arg\min_{h\in\mathcal{H}} R_{emp}(h) \tag{7}$$

Sources of Uncertainty in Machine Learning [8]



Sources of Uncertainty in Machine Learning

Distribution shift

The dependency between $\mathcal{X}$ and $\mathcal{Y}$ is typically non-deterministic, the description of a new prediction problem in the form of an instance $\boldsymbol{x}$ gives rise to a conditional probability distribution

$$P(y|\boldsymbol{x}) = \frac{P_{\mathcal{X}\times\mathcal{Y}}(\boldsymbol{x}, y)}{P(\boldsymbol{x})} \tag{8}$$

Distribution shift can occur in either the marginal distribution $P(\boldsymbol{x})$, or both $P(y)$ and $P(\boldsymbol{x})$.

- Distribution shift can occur in the sensory hence we have $P(\boldsymbol{x}) \neq P'(\boldsymbol{x})$
- Distribution shift can occur in the semantic hence we have $P(y) \neq P'(y)$

Sources of Uncertainty in Machine Learning

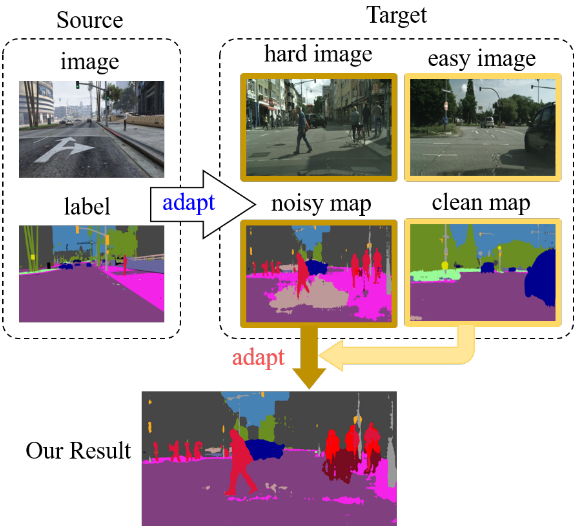


Figure: Sensory Distribution shift: Domain adaptation [1]

Sources of Uncertainty in Machine Learning

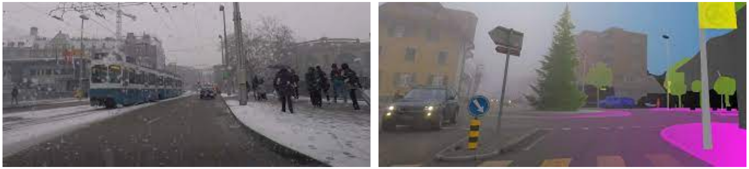


Figure: Sensory Distribution shift: ACDC Dataset [2]

Sources of Uncertainty in Machine Learning

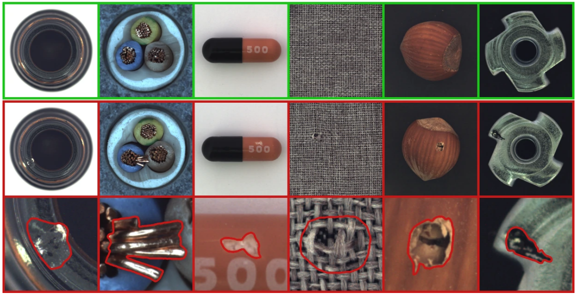


Figure: Sensory Distribution shift: MVTEC Dataset [3]

Sources of Uncertainty in Machine Learning

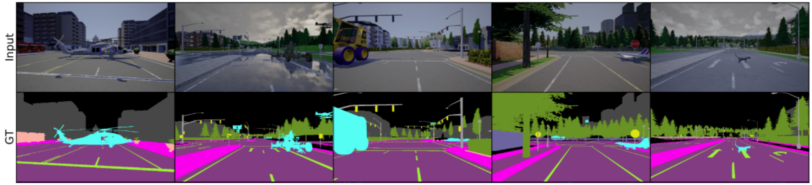


Figure: Semantic Distribution shift: StreetHazards [4]

Sources of Uncertainty in Machine Learning

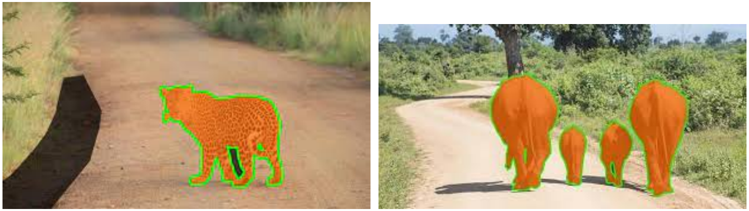


Figure: Sensory and semantic (mixed) Distribution shift, for example, the SegmentMeIfYouCan Challenge [5]

Sources of Uncertainty in Machine Learning

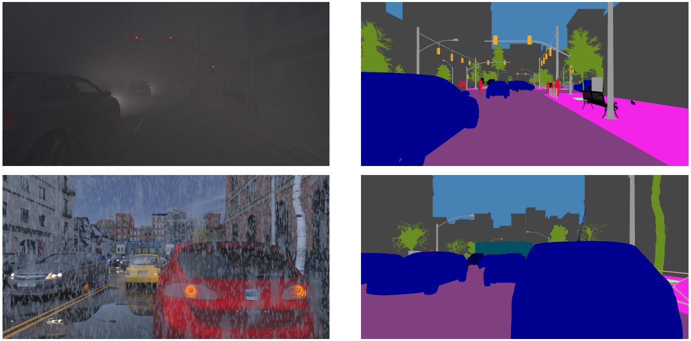


Figure: Sensory and semantic (mixed and not mixed) Distribution shift, for example, the MUAD dataset [6]

Sources of Uncertainty in Machine Learning

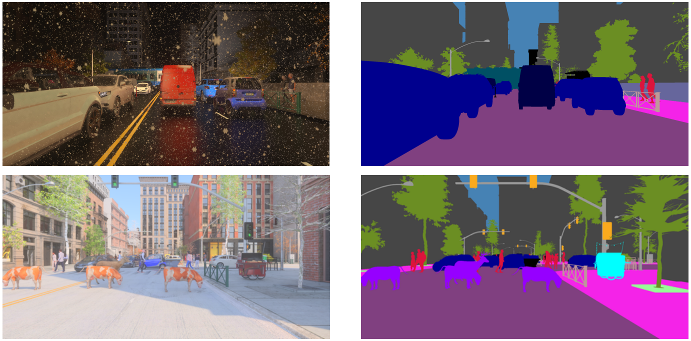
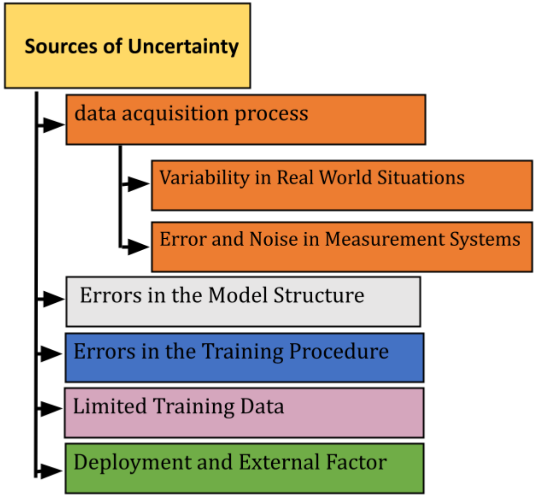


Figure: Sensory and semantic (mixed and not mixed) Distribution shift for example MUAD Dataset [6]

Sources of Uncertainty in Machine Learning


Sources of Uncertainty
data acquisition process
Variability in Real World Situations
Error and Noise in Measurement Systems
Errors in the Model Structure
Errors in the Training Procedure
Limited Training Data
Deployment and External Factor

Types of Uncertainty in Machine Learning

Epistemic Uncertainty

- **Model Uncertainty:** Arises from a lack of knowledge about the true model structure. It can be reduced with more data and better model architecture.
- **Inherent Model Limitations:** Uncertainty arising from the inability of the model to capture all relevant aspects of the underlying data distribution.
- **Parameter Uncertainty:** Related to uncertainty in the values of model parameters, often addressed through techniques like Bayesian modeling.

Aleatoric Uncertainty

- **Data Uncertainty:** Arises from inherent variability in the data. It can be further classified into homoscedastic (constant variance) and heteroscedastic (varying variance) uncertainty.
- **Measurement Uncertainty:** Associated with errors in the measurement process, impacting the reliability of observed data.

Types of Uncertainty in Machine Learning

Aleatoric vs. Epistemic Uncertainty

- **Aleatoric Uncertainty:** Irreducible uncertainty due to inherent randomness.
- **Epistemic Uncertainty:** Uncertainty that can be reduced with more information or improved models.

Limitation on the separation Aleatoric vs. Epistemic

- **Irreducible vs. Reducible:** A a practitioner can decide to extend the description of instances by additional features, which essentially means replacing the current instance space X with another space X'.
- **Distribution shift:** Can be considered as Aleatoric and/or Epistemic Uncertainty

Types of Uncertainty in Machine Learning

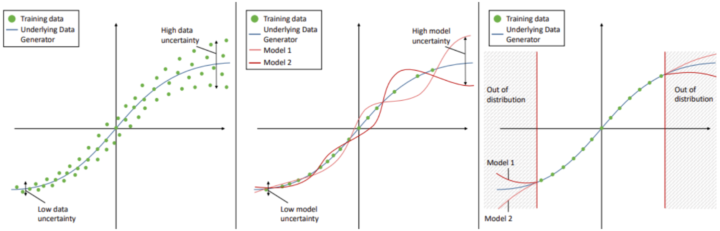


Figure: Visualization of the data, the model, and the distributional uncertainty for regression models. Taken from [7].

Types of Uncertainty in Machine Learning

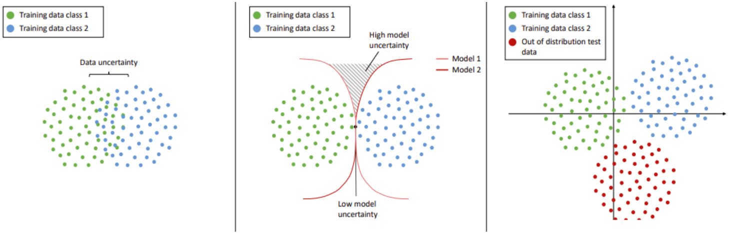


Figure: Visualization of the data, the model, and the distributional uncertainty for regression models. Taken from [7].

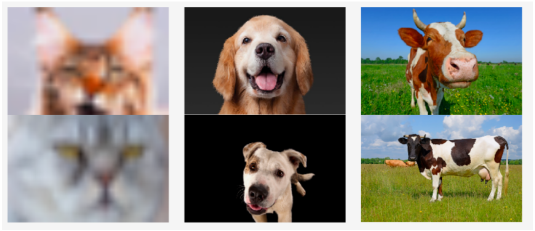
Types of Uncertainty: Case 1[1]

Let us consider a neural network model trained with several pictures of dogs. We ask the model to decide on a dog using a photo of a cat. What would you want the model to do?

[1]Credits: Gille Louppe

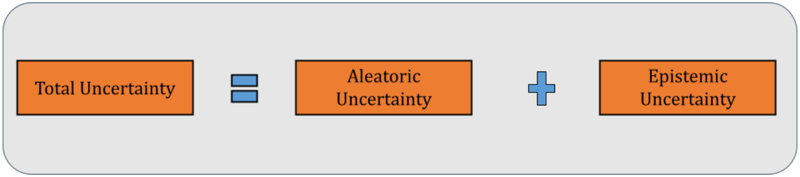
Types of Uncertainty: Case 2[2]

We have three different types of images to classify, cat, dog, and cow, some of which may be noisy due to the limitations of the acquisition instrument.

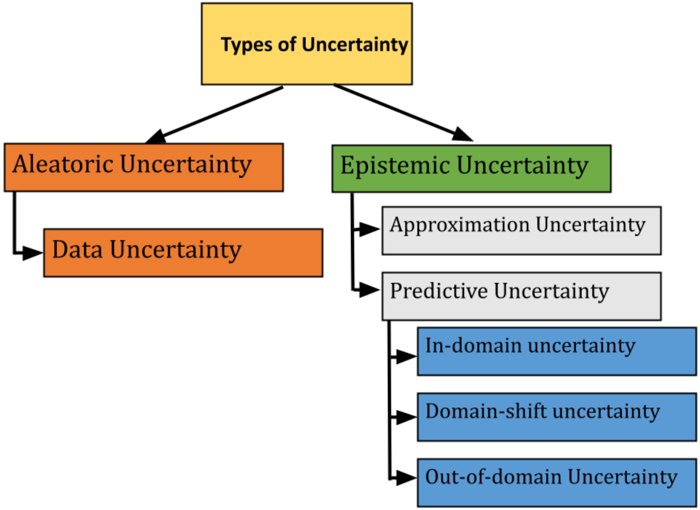


[2]Credits: Gille Louppe

Types of Uncertainty in Machine Learning



Types of Uncertainty in Machine Learning



The Need to Quantify Uncertainty in Deep Neural Networks

Context

- Deep Neural Networks (DNNs) have achieved remarkable success in various applications, but their predictions are not always infallible.
- Recognizing and quantifying uncertainty is crucial for enhancing the reliability and trustworthiness of DNNs.

Motivation

- **Real-world Consequences:** In critical applications such as healthcare or autonomous systems, incorrect predictions can have severe consequences.
- **Decision-Making:** Users and decision-makers need to understand the confidence levels associated with DNN predictions.

The Need to Quantify Uncertainty in Deep Neural Networks

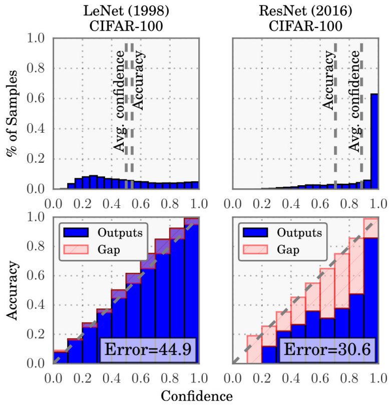


Figure: Confidence histograms (top) and reliability diagrams (bottom) for a 5-layer LeNet (left) and a 110-layer ResNet (right)on CIFAR-100. [9]

Addressing Epistemic Uncertainty

Epistemic Uncertainty

- DNNs often face uncertainty arising from limited training data and model complexity.
- **Quantifying Epistemic Uncertainty:** By estimating uncertainty related to model parameters and architecture, we can better gauge the reliability of predictions.

Benefits

- **Model Robustness:** Understanding epistemic uncertainty helps improve model robustness by highlighting areas where predictions are less certain.
- **Adaptive Decision-Making:** Decision-makers can adapt their actions based on the level of confidence provided by the model.

Addressing Aleatoric Uncertainty

Aleatoric Uncertainty

- Aleatoric uncertainty stems from inherent randomness and variability in the data.
- **Quantifying Aleatoric Uncertainty:** DNNs should account for uncertainty in the observed data, especially in scenarios with diverse and dynamic inputs.

Applications and Ethical Considerations

Applications

- **Medical Diagnosis:** In medical imaging, uncertainties in image quality and patient conditions must be considered for reliable diagnoses.
- **Autonomous Systems:** Vehicles and robots operating in dynamic environments require models that can handle uncertainties in sensor inputs.

Ethical Considerations

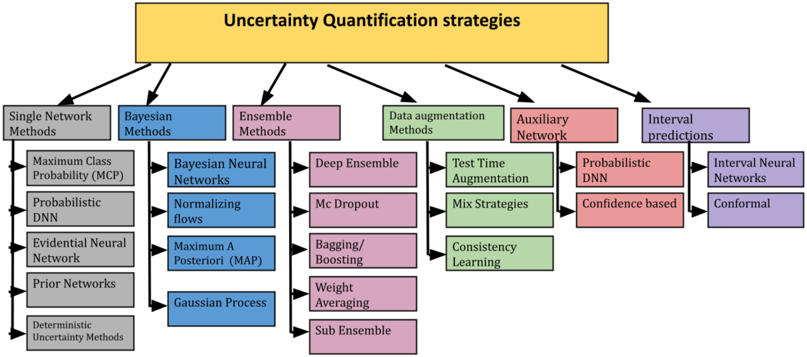
- **Avoiding Overconfidence:** Quantifying uncertainty helps prevent overconfident predictions that may lead to biased or unfair decisions.
- **Transparency and Accountability:** Understanding uncertainty enhances the accountability of AI systems and ensures ethical use.

Diverse Applications of Uncertainty in Machine Learning

Other applications

- **Active Learning:** Utilizing uncertainty in active learning aids in selecting images that benefit the most from annotation, optimizing the learning process.
- **Semi-Supervised Learning:** Uncertainty is valuable in semi-supervised learning, allowing for pseudo-annotation of data without explicit human labeling.
- **Domain Adaptation:** Incorporating uncertainty is beneficial in domain adaptation, providing insights into the effectiveness of knowledge transfer between different domains.
- **Open World Object Detection:** Leveraging uncertainty in open-world object detection helps identify instances when the model encounters objects outside the scope of its training domain.

Uncertainty Quantification Strategies



Uncertainty Quantification Strategies

Single Deep Neural Network (DNN)

- **Basic Approach:** Training a single DNN and utilizing inherent uncertainty in its predictions.
- **Limitations:** May not capture diverse sources of uncertainty, especially in complex scenarios.
- **Application:** Commonly used in straightforward tasks where model complexity is not a significant concern.

Ensemble Methods

- **Idea:** Combine predictions from multiple models (ensemble) to enhance overall performance and quantify uncertainty.
- **Diversity:** Ensemble models can be diverse in architectures, initializations, or training data, capturing different aspects of uncertainty.
- **Benefits:** Generally more robust and provide better uncertainty estimates.

Uncertainty Quantification Strategies

Bayesian Methods

- **Principle:** Bayesian methods incorporate prior knowledge and update beliefs through observed data.
- **Uncertainty Estimation:** Bayesian neural networks explicitly model uncertainty by assigning probability distributions to weights.
- **Advantages:** Capture epistemic uncertainty effectively but computationally demanding.

Data Augmentation-Based Methods

- **Approach:** Introduce variations in training data through data augmentation techniques.
- **Uncertainty Incorporation:** Augmented data provides a broader view of the input space, aiding in uncertainty estimation.
- **Applications:** Commonly used in image classification and computer vision tasks.

Uncertainty Quantification Strategies

Auxiliary-Based Networks

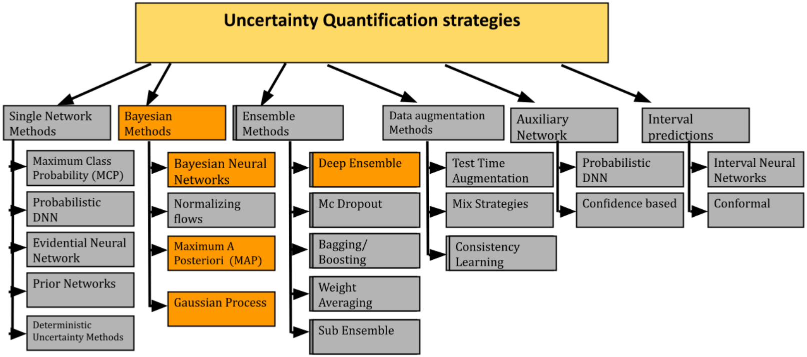
- **Concept:** Augment primary task networks with auxiliary tasks to enhance model understanding and uncertainty estimation.
- **Benefits:** Improves model generalization and robustness by learning diverse features.
- **Applications:** Useful in scenarios with limited labeled data.

Interval-Based Methods

- **Approach:** Define prediction intervals to quantify uncertainty.
- **Benefits:** Provide a straightforward and interpretable way to communicate uncertainty to users.
- **Applications:** Widely used in regression tasks.

Uncertainty Quantification Strategies

We will focus today just on the Orange blocks.



Bayesian Approach in Deep Neural Networks

The goal of a Deep Neural Network (DNN) is to model the conditional probability:

$$\mathcal{P}(Y|X, \boldsymbol{\omega})$$

Most classical approaches optimize the parameters $\boldsymbol{\omega}$ by maximizing the likelihood:

$$\begin{aligned}
\boldsymbol{\omega} &= \arg\max_{\boldsymbol{\omega}} \log \mathcal{P}(\mathcal{D}_l|\boldsymbol{\omega}) \\
&= \arg\max_{\boldsymbol{\omega}} \sum_{i=1}^{n_l} \log \mathcal{P}(Y_i|X_i, \boldsymbol{\omega}) \\
&= \arg\max_{\boldsymbol{\omega}} \frac{1}{n_l} \sum_{i=1}^{n_l} \log \mathcal{P}(Y_i|X_i, \boldsymbol{\omega}) \\
&= \arg\max_{\boldsymbol{\omega}} \mathbb{E}_{(X,Y)\sim\mathcal{P}(\mathcal{D}_l)} \log \mathcal{P}(Y|X, \boldsymbol{\omega}) \\
&= \arg\min_{\boldsymbol{\omega}} H[\mathcal{P}(\mathcal{D}_l), \mathcal{P}(Y|X, \boldsymbol{\omega})]
\end{aligned}$$

where H denotes the cross-entropy.

Bayesian Optimization and MAP Estimation

In the Bayesian framework, we seek the maximum a posteriori (MAP) estimate:

$$\begin{aligned}\boldsymbol{\omega} &= \arg\max_{\boldsymbol{\omega}} \log \mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_I) \\ &= \arg\max_{\boldsymbol{\omega}} [\log \mathcal{P}(\mathcal{D}_I|\boldsymbol{\omega}) + \log \mathcal{P}(\boldsymbol{\omega})]\end{aligned}$$

This results in L2 regularization, where prior knowledge about parameters is incorporated.

Bayesian DNN [17]

Bayesian DNN is based on marginalization instead of MAP optimization.

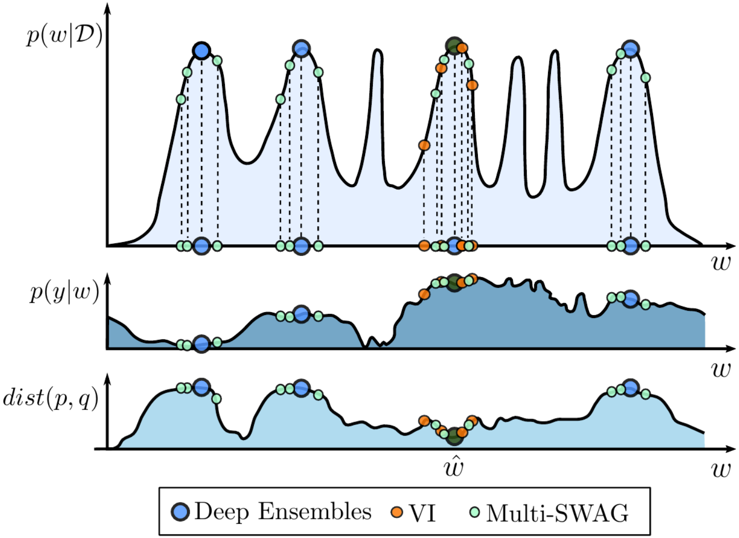
$$\mathcal{P}(Y|X) = \mathbb{E}_{\omega \sim \mathcal{P}(\omega|\mathcal{D})} \left(\mathcal{P}(Y|X,\omega)\right)$$
$$\mathcal{P}(Y|X) = \int \mathcal{P}(Y|X,\omega)\mathcal{P}(\omega|\mathcal{D})d\omega$$

In practice:

$$\mathcal{P}(Y|X) \simeq \sum_i \left(\mathcal{P}(Y|X,\omega_i)\right) \text{ with } \omega_i \sim \mathcal{P}(\omega|\mathcal{D})$$

Different techniques to estimate $\mathcal{P}(\omega|\mathcal{D})$.

Posterior and ensemble [21]



Ensemble Methods Overview

Ensemble Methods Overview

- Ensemble methods combine multiple base models to create a stronger, more robust predictive model.
- They are widely used to improve generalization and performance in various machine learning tasks.
- Can be used to quantify the uncertainty

Ensemble Methods Overview

Ensemble Methods Overview

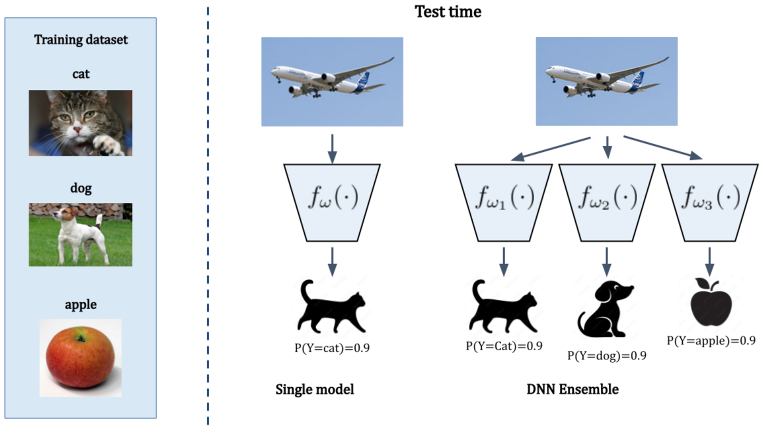
- **Bagging (Bootstrap Aggregating):** Involves training multiple instances of the same model using different subsets of the training data and averaging their predictions.
- **Random Forests:** Adapts the traditional random forest concept to neural networks, creating an ensemble of decision trees or models.
- **Boosting:** Sequentially trains multiple weak learners, giving more weight to misclassified instances in each iteration to improve overall model performance.
- **Stacking:** Involves training multiple diverse models and combining their predictions using another model (meta-learner).
- **Dropout:** During training, it randomly drops out neurons, effectively training an ensemble of slightly different models

Ensemble Methods Overview

Ensemble Methods Overview

- **Snapshot Ensembles:** Involves saving multiple snapshots of a model during training and using these snapshots as an ensemble for making predictions.
- **Bayesian Neural Networks (BNNs):**Introduces uncertainty by treating weights as probability distributions, providing a Bayesian approach to ensembling.
- **Deep Ensembles:** It ensembles multiple independently trained neural networks to improve generalization and reduce overfitting.
- **Weak Ensembles:** Methods that extend the ensemble concept to the batch dimension during training, and that ensemble smaller DNNs.

Motivation for Ensemble Methods



Bagging (Bootstrap Aggregating)

Bagging Overview

- **Idea:** Bagging involves training multiple instances of the same model using different subsets of the training data.
- **Bootstrap Sampling:** Each model is trained on a bootstrap sample (a random sample with replacement) from the original dataset.
- **Aggregation:** Predictions are aggregated (e.g., averaged for regression, majority vote for classification) to form the final prediction.

Advantages of Bagging

Benefits of Bagging

- **Reduced Variance:** Bagging helps reduce model variance by combining diverse models trained on different subsets of the data.
- **Improved Stability:** By averaging predictions, bagging creates a more stable and robust model, especially in the presence of outliers or noisy data.
- **Parallelization:** Training each model independently allows for easy parallelization, improving computational efficiency.

Boosting

Boosting Overview

- **Idea:** Boosting focuses on sequentially training multiple weak learners and giving more weight to misclassified instances.
- **Weighted Training:** Misclassified instances are given higher weights, emphasizing their importance in subsequent models.
- **Combining Weak Models:** Predictions are combined, with more emphasis on models that perform well on challenging instances.

Advantages of Boosting

Benefits of Boosting

- **Sequential Improvement:** Boosting builds models sequentially, each focusing on correcting errors made by the previous ones.
- **Adaptability:** Adaptive learning allows boosting to perform well even with weak base models, leading to strong overall performance.
- **Handling Imbalanced Data:** Boosting is effective in handling imbalanced datasets by giving more weight to minority class instances.

Comparison: Bagging vs. Boosting

Bagging and Boosting Differences

- **Bagging:** Parallel training of diverse models with equal weight; reduces variance.
- **Boosting:** Sequential training of models with adaptive weights; focuses on correcting errors.
- **Diversity:** Bagging relies on diverse subsets; boosting emphasizes diversity through adaptive training.

Deep Ensembles [13]

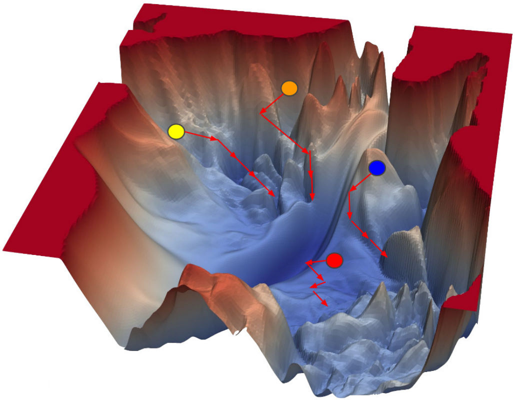
Deep Ensembles

- **Idea:** Deep Ensembles involves training multiple instances of the same DNN model using the same training data.
- **Diversity:** Contrary to Bagging and Boosting Deep Ensemble relies on 3 sources of stochasticity:
    - Stochastic Optimisation
    - Random Initialisation
    - Non-deterministic backpropagation studied in [25]
- **Aggregation:** Predictions are averaged.

Deep Ensembles [13]

They [13] propose to average the predictions of several DNNs with different initial seeds:

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\textbf{model}}} \sum_{j=1}^{N_{\textbf{model}}} \mathcal{P}(y^*|\omega^j(t^*), x^*) \tag{9}$$



Deep Ensembles [18]

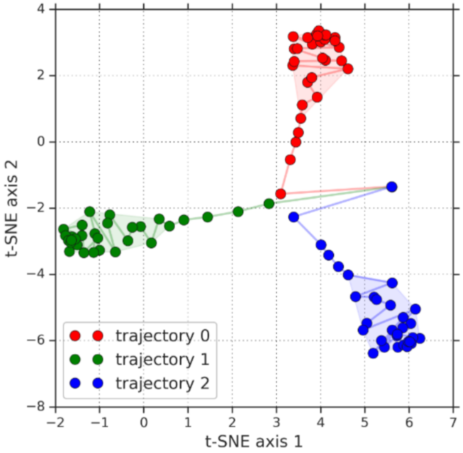


Figure: t-SNE plot of predictions from checkpoints corresponding to 3 different randomly initialized trajectorie

Deep Ensembles [18]

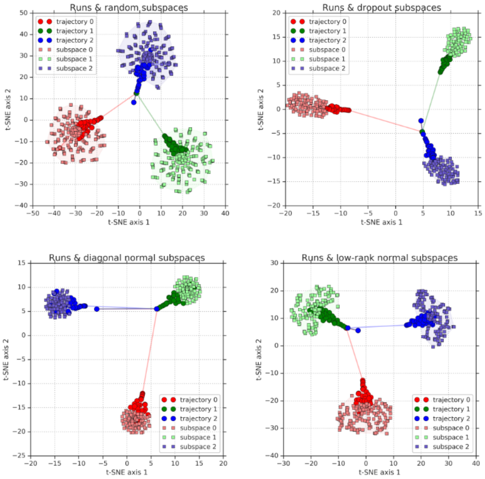


Figure: Results using SimpleCNN on CIFAR-10: t-SNE plots of validation set predictions for each trajectory along with four different subspace generation methods

Deep Ensembles [18]

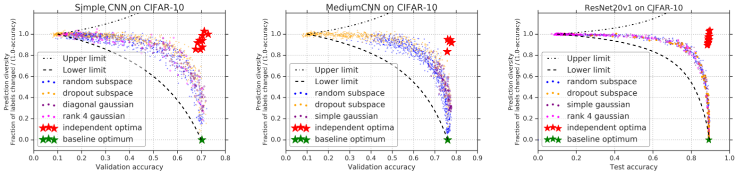
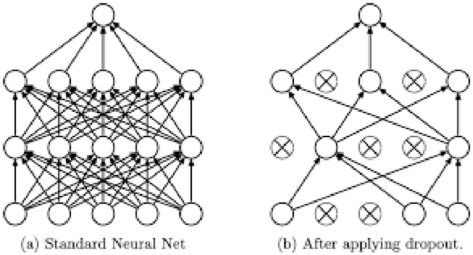


Figure: Diversity versus accuracy plots for 3 models trained on CIFAR-10

Introduction to Dropout[10]

What is Dropout?

- **Definition:** Dropout is a regularization technique used in neural networks to prevent overfitting.
- **Idea:** During training, randomly "drop out" (ignore) a fraction of neurons, forcing the network to be more robust and preventing reliance on specific neurons.



How Dropout Works[10]

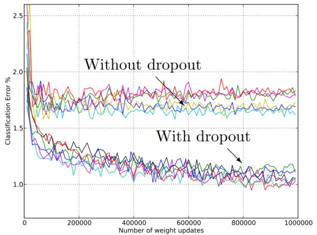
Mechanism of Dropout

- **Training Phase:** In each training iteration, random neurons are dropped out with a specified probability.
- **Variability:** Dropping out neurons introduces variability, making the network less sensitive to the presence of any individual neuron.
- **Ensemble Effect:** Dropout can be seen as training an ensemble of multiple subnetworks, each missing different neurons.

Benefits and Considerations

Advantages and Considerations of Dropout

- **Regularization:** Dropout helps prevent overfitting, improving the model's generalization to unseen data.
- **Ensemble Training:** Provides an implicit way to train multiple models simultaneously, enhancing robustness.
- **Hyperparameter:** The dropout rate is a hyperparameter that needs to be tuned based on the specific task and dataset.
- **Impact on Training Time:** While dropout is beneficial during training, it is typically turned off during inference.

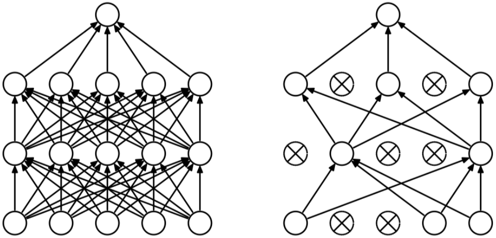


MC dropout [12]

They [12] propose to average the predictions of several DNNs where they apply the dropout:

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\textbf{model}}} \sum_{j=1}^{N_{\textbf{model}}} \mathcal{P}(y^*|\omega(t^*) \odot b^j, x^*) \tag{10}$$

with b^j a vector of the same size of $\omega(t^*)$ which is a realization of a binomial distribution.


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

Overview of Weak Ensemble Methods

Overview of Weak Ensemble Methods

- While Deep Ensemble is frequently considered state-of-the-art (SOTA), it comes with significant computational demands.
- Weak Ensemble methods offer a faster alternative to achieve comparable results.
- Weak Ensemble methods can be performed on a reduced dataset, or/and with fewer neurons, or/and for a shorter duration.

BatchEnsemble Overview [22]

What is BatchEnsemble?

- **Definition:** BatchEnsemble is an ensemble learning technique designed for improving the performance and robustness of neural networks.
- **Inspiration:** Inspired by ensemble methods, BatchEnsemble extends the concept to the batch dimension during training.

How BatchEnsemble Works

- **Batch-Level Ensembling:** Instead of ensembling models across different training runs, BatchEnsemble ensembles within the same training batch.
- **Variability Across Batches:** Introduces diversity by training multiple instances of the model within each batch, enhancing robustness.

BatchEnsemble Overview [22]

They [22] propose to approximate the average of the predictions of several DNN with different initial seeds by using a DNN with two king of weights. For simplicity is the ω has two set of weight ω^{slow} , ω^{fast}
For simplicity let us consider a DNN with just one fully connected layer and let us write $\omega = \{\omega_j\}_{j=1}^{N_{\textbf{model}}} = \{W_j\}_{j=1}^{N_{\textbf{model}}}$ and $\omega^{\text{slow}} = W$ and $\omega^{\text{slow}} = \{F_j\}_{j=1}^{N_{\textbf{model}}}$. We have $W_j = W \cdot F = W \cdot (r_j s_j^t)$

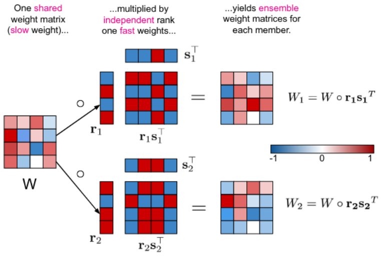

One shared weight matrix (slow weight)...
...multiplied by independent rank one fast weights...
...yields ensemble weight matrices for each member.
W
$\circ$
$\mathbf{r}_1$
$\mathbf{s}_1^\top$
$\mathbf{r}_1\mathbf{s}_1^\top$
$=$
$W_1 = W \circ \mathbf{r_1}\mathbf{s_1}^T$
-1
0
1
$\circ$
$\mathbf{r}_2$
$\mathbf{s}_2^\top$
$\mathbf{r}_2\mathbf{s}_2^\top$
$=$
$W_2 = W \circ \mathbf{r_2}\mathbf{s_2}^T$

Figure: An illustration on how to generate the ensemble weights for two ensemble members

BatchEnsemble Overview [22]

We have a set of weight $W_j = W \cdot F = W \cdot (r_j s_j^t)$ with W that sees all images and $(r_j s_j^t)$ that does not see all the same images. If we denote ϕ an activation function then when we apply the BatchEnsemble on an image we perform:

$$y = \phi\left(W_j^t x\right) = \phi\left(\left(W^t \cdot (r_j s_j^t)\right)^t x\right) = \phi\left(\left(W^t (x \cdot r_j) \cdot s_j\right)\right)$$

Similarly to Deep Ensembles, to perform inference we just perform ensembling :

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\textbf{model}}} \sum_{j=1}^{N_{\textbf{model}}} \mathcal{P}(y^*|\omega_j, x^*) \tag{11}$$

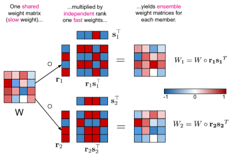

One shared weight matrix (slow weight)...
...multiplied by independent rank one fast weights...
...yields ensemble weight matrices for each member.
$\mathbf{s}_1^\top$
$\mathbf{r}_1$
$\mathbf{r}_1\mathbf{s}_1^\top$
$W_1 = W \circ \mathbf{r_1 s_1}^T$
W
-1
0
1
$\mathbf{s}_2^\top$
$\mathbf{r}_2$
$\mathbf{r}_2\mathbf{s}_2^\top$
$W_2 = W \circ \mathbf{r_2 s_2}^T$

Figure: An illustration on how to generate the ensemble weights for two

MIMO Overview [11]

What is MIMO?

- **Definition:** MIMO stands for Multiple Input Multiple Output .
- **Objective:** MIMO aims to utilize a single model's capacity to train multiple subnetworks that independently learn the task at hand.

Key Concepts of MIMO [11]

How MIMO Works

- **MIMO principle:** The lottery ticket hypothesis shows that one can prune away 70-80% of the connections in a DNN without adversely affecting performance
- **MIMO Idea:**The basic Idea is that a neural network has sufficient capacity to fit 3-4 independent subnetworks simultaneously. Hence they just need to modify the input and output to handle this 3-4 subnetworks.

Key Concepts of MIMO [11]

```
class MIMOModel(nn.Module):
    def __init__(self, hidden_dim: int = 784, ensemble_num: int = 3):
        super(MIMOModel, self).__init__()
        self.input_layer = nn.Linear(hidden_dim, hidden_dim * ensemble_num)
        self.backbone_model = BackboneModel(hidden_dim, ensemble_num)
        self.ensemble_num = ensemble_num
        self.output_layer = nn.Linear(128, 10 * ensemble_num)

    def forward(self, input_tensor: torch.Tensor):
        input_tensor = input_tensor.transpose(1, 0).view(
            batch_size, self.ensemble_num, -1)
        # (batch_size, ensemble_num, hidden_dim)
        input_tensor = self.input_layer(input_tensor)
        # (batch_size, ensemble_num, hidden_dim * ensemble_num)
        # usual model forward
        output = self.backbone_model(input_tensor)  # (batch_size, ensemble_num, 128)
        output = self.output_layer(output)  # (batch_size, ensemble_num, 10 * ensemble_n
        output = output.reshape(
            batch_size, ensemble_num, -1, ensemble_num
        )  # (batch_size, ensemble_num, 10, ensemble_num)
        output = torch.diagonal(output, offset=0, dim1=1, dim2=3).transpose(2, 1)
# (batch_size, ensemble_num, 10)
        output = F.log_softmax(output, dim=-1)  # (batch_size, ensemble_num, 10)
        return output
```

Key Concepts of MIMO [11]

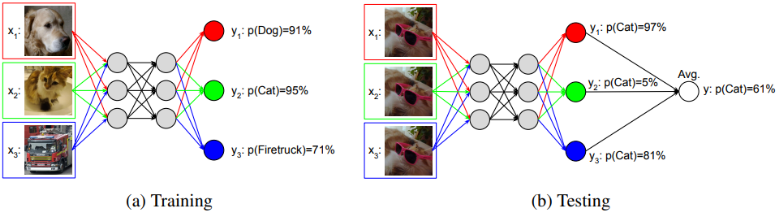


Figure: The multi-input multi-output (MIMO) configuration, the network takes M = 3 inputs and gives M outputs [11]

Packed-Ensembles Overview [25]

Seamless training of ensembles with Packed-Ensembles

- **Definition:** Packed-Ensembles estimate the posterior distributions restraining their support to smaller networks in a computationally efficient manner with grouped convolutions.
- **Objective:** Get the benefits of *deep ensembles* with reduced costs.

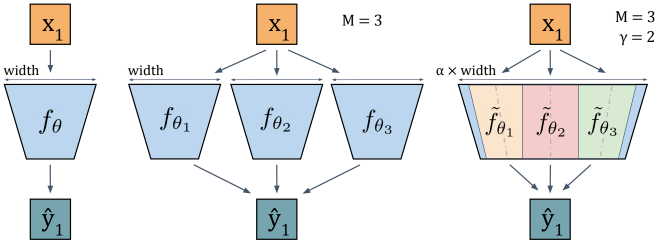


Figure: **Left:** A standard network, **Center:** A *deep ensembles*, **Right:** The corresponding Packed-Ensembles

How well does Packed-Ensembles perform? [25]

Performance of Packed-Ensembles

- **Performance:** For sufficiently large networks, Packed-Ensembles is equivalent to *deep-ensembles* in performance and UQ.
- **Computational efficiency:** Use Packed-Ensembles with *float16* to benefit from grouped-convolutions better.

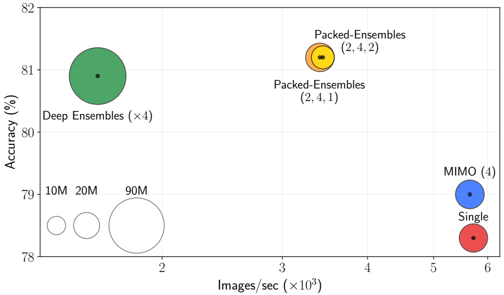


Figure: Performance (accuracy) wrt. the image throughput

Sources of stochasticity in deep ensembles [25]

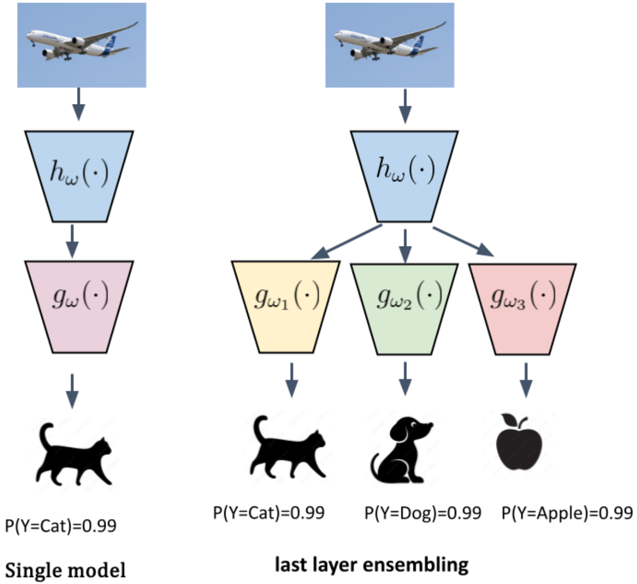
Stochasticity			ResNet-50				
ND	**DI**	**DB**	**Acc** (↑)	**ECE** (↓)	**AUPR** (↑)	IDMI	OODMI
-	-	-	77.63±0.23	0.0825±0.0018	89.19±0.65	0±0	0±0
✓	-	-	80.94±0.10	0.0179±0.0010	90.23±0.62	0.1513	0.4022
-	✓	-	81.01±0.06	0.0202±0.0011	91.10±0.39	0.1524	0.4088
-	-	✓	80.87±0.10	0.0178±0.0010	90.80±0.30	0.1505	0.4115
✓	✓	✓	81.08±0.08	0.0198±0.0013	90.68±0.25	0.1534	0.4031

Figure: Impact of the three sources of stochasticity, non-deterministic backdrop. kernels (**ND**), different initialization (**DI**), and different batches (**DB**).

Uncertainty-sources are equivalent!

No source of stochasticity during training seems to single out. Having one source is sufficient, and adding more does not seem to affect the performance or the quantitative functional diversity (Mutual information).

Last Layer sampling [29]



Last Layer sampling [29]

Last Layer Sampling

- **Idea:** Instead of estimation the posterior for the entire model, only we estimate the posterior only for the last layer.
- **Benefits:** Reduces computational cost and memory, while still benefiting from ensemble-based uncertainty estimation.
- **Diversity:** Each ensemble member has a unique final layer, leading to diverse predictions while sharing common features from earlier layers.
- **Aggregation:** Predictions are averaged across the last layer ensemble members, similar to full ensembles.

Mixture Density Networks Overview [26]

What are Mixture Density Networks?

- **Definition:** Mixture Density Networks (MDNs) are neural networks producing the parameters of mixture density models.
- **Objective:** Relaxing the inductive bias enforced onto the model when assuming $P(y|\mathbf{x}, \omega)$ follows unimodal distributions, by considering mixtures of them.

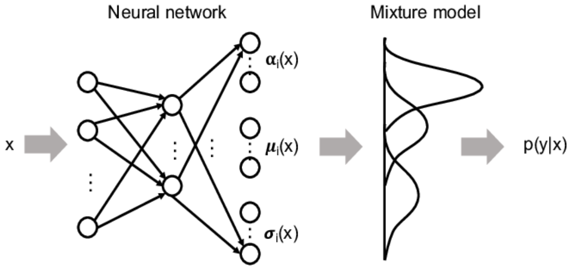


Figure: The output of a neural network parametrizes a Gaussian mixture model (from [27]).

Mixture Density Networks Overview [26]

MDNs can approximate any probability density function up to a certain point, assuming mixing coefficients and Gaussian parameters are carefully chosen.

Building Mixture Density Networks

- MDNs can be implemented using ensembles of networks producing unimodal distribution parameters.
- Further in that direction we can consider mixtures of mixtures using ensembles of MDNs, enabling different levels of modes. [28]

By doing so, each ensemble member can contribute to modeling the aleatoric uncertainty, while the ensemble helps to estimate epistemic uncertainty.

Bayesian Neural Networks (BNN)

Unlike MAP optimization, Bayesian Neural Networks perform marginalization over parameters:

$$\begin{aligned}\mathcal{P}(Y|X) &= \mathbb{E}_{\omega\sim\mathcal{P}(\omega|\mathcal{D}_l)}\left[\mathcal{P}(Y|X,\omega)\right] \\ &= \int \mathcal{P}(Y|X,\omega)\mathcal{P}(\omega|\mathcal{D}_l)d\omega\end{aligned}$$

In practice, this is approximated using Monte Carlo sampling:

$$\mathcal{P}(Y|X) \approx \frac{1}{M}\sum_{i=1}^{M}\mathcal{P}(Y|X,\omega_i), \quad \omega_i \sim \mathcal{P}(\omega|\mathcal{D}_l)$$

Several methods exist for approximating $\mathcal{P}(\omega|\mathcal{D}_l)$.

Variational Inference in Bayesian Neural Networks

Variational inference approximates the posterior $\mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_l)$ using a family of distributions $q_\lambda(\boldsymbol{\omega})$:

- λ represents the variational parameters defining the distribution.
- If q is Gaussian, $\lambda = (\mu, \sigma^2)$ represents its mean and variance.

Key Question: How well does $q_\lambda(\boldsymbol{\omega}|\mathcal{D}_l)$ approximate $\mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_l)$?

Measuring Approximation: KL Divergence

To quantify the approximation quality, we use Kullback-Leibler (KL) divergence:

$$\begin{aligned}\mathbb{KL}(q_\lambda(\omega|\mathcal{D}_l)||\mathcal{P}(\omega|\mathcal{D}_l)) &= \int q_\lambda(\omega|\mathcal{D}_l)\log\frac{q_\lambda(\omega|\mathcal{D}_l)}{\mathcal{P}(\omega|\mathcal{D}_l)}d\omega \\ &= \mathbb{E}_q[\log q_\lambda(\omega|\mathcal{D}_l)] - \mathbb{E}_q[\log \mathcal{P}(\omega,\mathcal{D}_l)] + \log \mathcal{P}(\mathcal{D}_l)\end{aligned}$$

Evidence Lower Bound (ELBO)

Since $\mathcal{P}(\mathcal{D}_l)$ is intractable, we focus change the problem to find the optimal approximate posterior:

$$q^*_\lambda(\omega/\mathcal{D}_l) = \arg\min_\lambda \mathbb{KL}(q_\lambda(\omega/\mathcal{D}_l) \mid\mid \mathcal{P}(\omega|\mathcal{D}_l)).$$

Hence, we optimize the Evidence Lower Bound (ELBO):

$$\text{ELBO}(\lambda) = \mathbb{E}_q[\log \mathcal{P}(\mathcal{D}_l|\omega)] - \mathbb{KL}(q_\lambda(\omega/\mathcal{D}_l)||\mathcal{P}(\omega))$$

Minimizing KL divergence is equivalent to maximizing ELBO.

Reparameterization Trick

Theorem: If $\boldsymbol{\omega} = t(\lambda, \epsilon)$ with $\epsilon \sim q(\epsilon)$ and $q(\epsilon)d\epsilon = q_\lambda(\boldsymbol{\omega})d\boldsymbol{\omega}$, then:

$$\frac{\partial}{\partial \lambda}\mathbb{E}_{q_\lambda(\boldsymbol{\omega})} f(\boldsymbol{\omega}, \lambda) = \mathbb{E}_{q(\epsilon)}\left[\frac{\partial f(\boldsymbol{\omega}, \lambda)}{\partial \boldsymbol{\omega}}\frac{\partial \boldsymbol{\omega}}{\partial \lambda} + \frac{\partial f(\boldsymbol{\omega}, \lambda)}{\partial \lambda}\right]$$

This trick enables gradient-based optimization of variational parameters.

Bayesian Neural Networks: Practical Example

1. Sample $\epsilon \sim \mathcal{N}(0, I)$.
2. Let $\mathbf{w} = \mu + \log(1 + \exp(\rho)) \circ \epsilon$.
3. Let $\theta = (\mu, \rho)$.
4. Let $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathbf{w})P(\mathcal{D}|\mathbf{w})$.
5. Calculate the gradient with respect to the mean

$$\Delta_\mu = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu}. \tag{3}$$

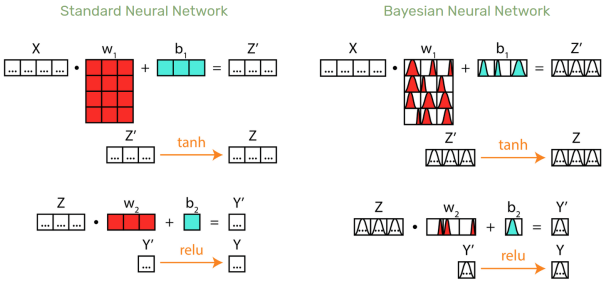
6. Calculate the gradient with respect to the standard deviation parameter ρ

$$\Delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}. \tag{4}$$

7. Update the variational parameters:

$$\mu \leftarrow \mu - \alpha \Delta_\mu \tag{5}$$
$$\rho \leftarrow \rho - \alpha \Delta_\rho. \tag{6}$$

Weight Uncertainty in Neural Networks[3]



[3]Image credit: Eric Ma

TRADI: a Bayesian DNN

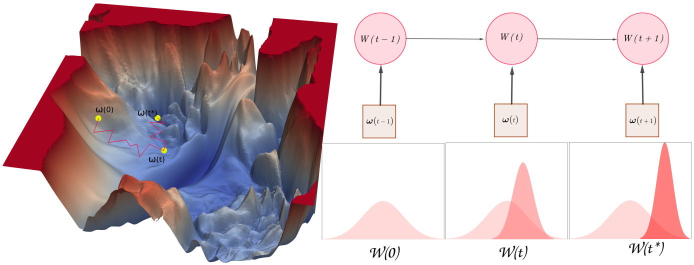


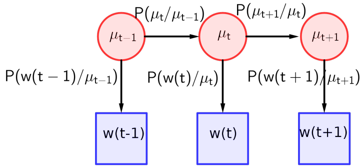
Figure: TRADI uses Kalman filtering for tracking the distribution $\mathcal{W}$ of all DNN weights across training steps from a generic prior $\mathcal{W}(0)$ to the final estimate $\mathcal{W}(t^*)$.

TRADI: Mean State and Measurement Equations

Equations for the Mean $\mu_k(t)$

$$\mu_k(t) = \mu_k(t-1) - \eta \nabla \mathcal{L}_{\omega_k(t)} + \varepsilon_\mu,$$
$$\omega_k(t) = \mu_k(t) + \tilde{\varepsilon}_\mu,$$

- $\varepsilon_\mu \sim \mathcal{N}(0, \sigma_\mu^2)$: State noise.
- $\tilde{\varepsilon}_\mu \sim \mathcal{N}(0, \tilde{\sigma}_\mu^2)$: Observation noise.



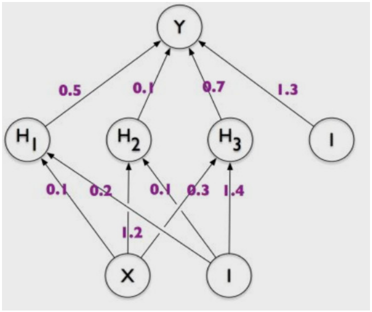
TRADI: Variance State and Measurement Equations

Equations for the Variance $\sigma_k^2(t)$

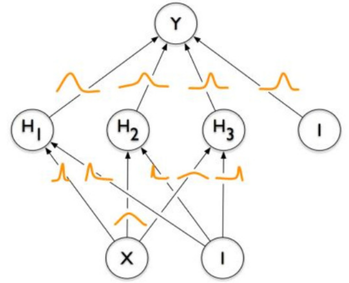
$$\begin{aligned}
\sigma_k^2(t) &= \sigma_k^2(t-1) + \left(\eta \nabla \mathcal{L}_{\omega_k(t)}\right)^2 - \eta^2 \mu_k(t)^2 + \varepsilon_\sigma, \\
z_k(t) &= \sigma_k^2(t) - \mu_k(t)^2 + \tilde{\varepsilon}_\sigma.
\end{aligned}$$

- $\varepsilon_\sigma \sim \mathcal{N}(0, \sigma_\sigma^2)$: State noise.
- $\tilde{\varepsilon}_\sigma \sim \mathcal{N}(0, \tilde{\sigma}_\sigma^2)$: Observation noise.

TRADI: Comparison - Normal DNN vs Bayesian DNN



Normal DNN



Bayesian DNN

TRADI: Sampling New Realizations of Weights

Weight Sampling Equation

$$\tilde{\boldsymbol{\omega}}(t^*) = \boldsymbol{\mu}(t^*) + \boldsymbol{\Sigma}^{1/2}(t^*) \times \boldsymbol{m}_1,$$

where $\boldsymbol{\Sigma}$ is the covariance matrix, and $\boldsymbol{m}_1 \sim \mathcal{N}(0_K, \boldsymbol{I}_K)$.

Prediction

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\tilde{\omega}^j(t^*), x^*).$$

TRADI: Efficient BNN Strategy

Key Idea

- TRADI is a Bayesian Neural Network (BNN) strategy that tracks the **posterior distribution** of the Deep Neural Network (DNN) during training.

Benefits of TRADI

- **Lightweight Approach**: Unlike traditional BNNs, TRADI does not introduce heavy computational overhead.
- **Non-intrusive**: It does not perturb the DNN training process, allowing it to run as efficiently as a standard DNN.
- **Posterior Tracking**: By monitoring the posterior, TRADI enhances the uncertainty estimation while keeping the model's original structure intact.

How to estimate the Posterior of BNN?

Classical VI-BNN

Using the "reparametrization trick", a layer j of an MLP can be written:

$$\begin{aligned} \boldsymbol{u}_j &= \text{norm}\left(\left[W_\mu^{(j)} + \epsilon_j W_\sigma^{(j)}\right] \boldsymbol{h}_{j-1}, \beta_j, \gamma_j\right), \text{ and} \\ \boldsymbol{a}_j &= a(\boldsymbol{u}_j), \end{aligned} \tag{12}$$

where the matrices $W_\mu^{(j)}$ and $W_\sigma^{(j)}$ denote the mean and standard deviation of the posterior distribution of layer j, $\epsilon_j \sim \mathcal{N}(0, \mathbb{1})$ and the operator norm$(\cdot, \beta_j, \gamma_j)$, of trainable parameters β_j and γ_j, can refer to any batch, layer, or instance normalization.

How to turn a DNN into a BNN?

ABNN

Our objective differs from VI-BNN, which requires training the posterior distribution parameters from scratch. Instead, our approach entails leveraging and converting an existing DNN into a BNN.

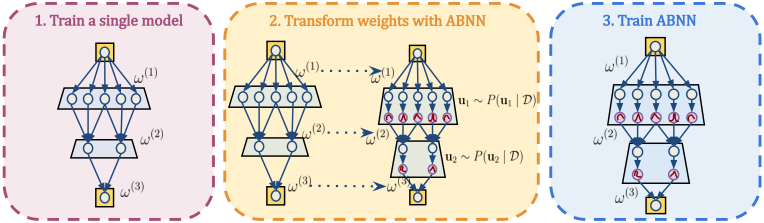


Figure: **Illustration of the training process for the ABNN.** The procedure begins with training a single DNN $\boldsymbol{\omega}_{\mathrm{MAP}}$, followed by architectural adjustments to transform it into an ABNN. The final step involves fine-tuning the ABNN model.

Converting DNNs into BNNs

Post-hoc Bayesian Strategy

- **Base Strategy:** Start with pre-trained DNNs with normalization layers like Batch, Layer, or Instance normalization.
- **Bayesian Adaptation:** Replace deterministic normalization layers with Bayesian Normalization Layers (BNL) that add Gaussian perturbation.
- **Goal:** Efficiently convert pre-trained DNNs into Bayesian Neural Networks (BNNs) with minimal modifications.

Bayesian Normalization Layer (BNL)

Transforming Normalization Layers

- **Key Equation:**

$$\boldsymbol{u}_j = \textbf{BNL}\left(W^{(j)}\boldsymbol{h}_{j-1}\right), \text{ and } \boldsymbol{a}_j = a(\boldsymbol{u}_j), \text{ with}$$

$$\textbf{BNL}(\boldsymbol{h}_j) = \frac{\boldsymbol{h}_j - \hat{\mu}_j}{\hat{\sigma}_j} \times \gamma_j(1+\epsilon_j) + \beta_j,$$

  where $\epsilon_j \sim \mathcal{N}(0, \mathbb{1})$

- **Explanation:** Gaussian perturbation is applied to normalization layers to introduce stochasticity like **gaussian dropout**, transforming deterministic layers into Bayesian layers.
- **Parameters:** γ_j and β_j are learnable vectors, retrained for a limited number of epochs.

Training ABNN

Training Strategy

- **Process:** - After replacing normalization layers with BNL, retrain the parameters for a few epochs.
- **Multi-Modality:** - Instead of training a single model, we train multiple ABNNs, each with different weight configurations $\omega_1, \ldots, \omega_M$.
- **Benefit:** This approach helps in improving the generalization and reliability of the BNN.

Inference in ABNN

Inference Strategy

- **Sampling from ABNN:** - For each ABNN sample, multiple ϵ_j are drawn independently from $\mathcal{N}(0, \mathbb{1})$.
- **Marginalization:** The prediction for a new sample $\boldsymbol{x}$ is the expected outcome from a finite ensemble of models and weight configurations:

$$P(y \mid \boldsymbol{x}, \mathcal{D}) \approx \frac{1}{ML} \sum_{l=1}^{L} \sum_{m=1}^{M} P(y \mid \boldsymbol{x}, \boldsymbol{\omega}_m, \epsilon_l).$$

Benefits of ABNN

Key Advantages

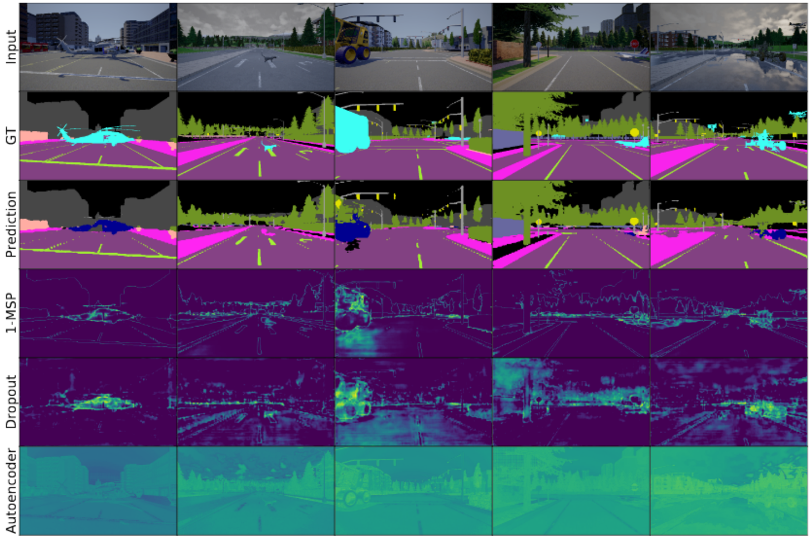
- **Uncertainty Estimation:** BNNs provide a probabilistic interpretation of model predictions.
- **Efficient Conversion:** Pre-trained DNNs can be easily adapted into BNNs with minimal retraining.
- **Scalability:** The use of Bayesian Normalization Layers (BNL) allows leveraging modern architectures like ResNet and Vision Transformers.

Quantifying Uncertainty in DNNs

Criteria for Uncertainty Assessment

- **Single DNN:**
  - **Maximum Class Probability (MCP):**
    - $max_{k \subset \mathcal{Y}} P(Y = k|x, w)$
    - Higher MCP implies higher confidence, while lower MCP indicates increased uncertainty.
  - **Entropy:**
    - $\mathrm{H}(P(Y|x, w)) := -\sum_{k \in \mathcal{Y}} P(Y = k|x, w) \log P(Y = k|x, w))$
    - Higher entropy signifies higher uncertainty as it reflects a more uniform distribution of probabilities. (related to aleatoric uncertainty)
- **Ensemble of DNNs:**
  - **Maximum Class Probability (MCP):** Similar to single DNN, but now considering the MCP across the marginalized distribution $P(Y = k|x) = \int \mathcal{P}(Y|X, \boldsymbol{\omega})\mathcal{P}(\boldsymbol{\omega}|\mathcal{D})d\boldsymbol{\omega}$.
  - **Entropy:** $\mathrm{H}(P(Y|x)) := -\sum_{k \in \mathcal{Y}} P(Y|x) \log P(Y|x))$
  - **Mutual Information:** Measures the shared information between predictions of individual models, offering insights into epistemic uncertainty: $I(Y|x) = \mathrm{H}(P(Y = k|x)) - E_{\mathcal{P}(\boldsymbol{\omega}|\mathcal{D})}\mathrm{H}(P(Y = k|\boldsymbol{\omega}, x))$

Some Results [23]



Some Results [24]

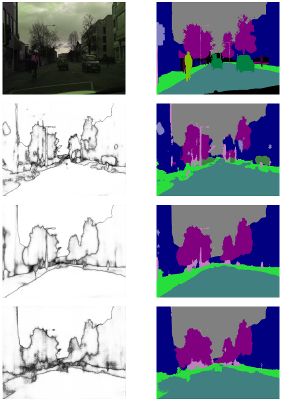
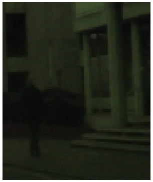
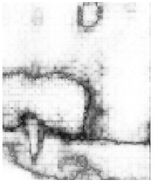
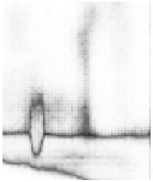


Figure: First row: input image and ground truth, second, third, and fourth rows: output and confidence score given by MC dropout, Deep Ensembles, and our TRADI, respectively.

Some Results [23]


(a) input image


(b) MC dropout confidence


(c) Deep Ensembles confidence

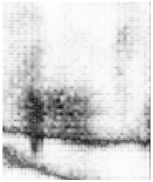

(d) TRADI confidence

Figure: Zooms of the confidence results on the CamVid experiments. In the bottom left of the input image (a), there is a human, hence a pixel region of an unknown class for all the DNNs, since the pedestrian class was amongst the ones marked as unlabeled. Yet, only the TRADI DNN (d) is consistent.

Evaluating Uncertainty Quantification in DNNs

Evaluating Uncertainty in DNNs

- Evaluating the quality of Uncertainty quantification is crucial for reliable deep learning models.
- We distinguish between aleatoric uncertainty, epistemic uncertainty, and distribution shift, each requiring specific evaluation metrics.

Introduction to ECE

Expected Calibration Error (ECE) for classification

- **Definition:** The Expected Calibration Error (ECE) is a metric used to assess the calibration of predicted probabilities in classification tasks.
- **Importance:** Calibration is crucial for models that provide probability estimates, ensuring that predicted confidence scores align with actual outcomes.

Introduction to ECE

ECE Formula and Interpretation

- **Formula:** We begin by partitioning the data into m bins based on the confidence scores of the DNN predictions. B_i represents the collection of samples whose predicted probabilities fall within the i-th bin.
$$ECE = \sum_{i=1}^{m} \frac{|B_i|}{N} \cdot |\text{accuracy}(B_i) - \text{confidence}(B_i)|$$
- **Interpretation:**
  - A perfectly calibrated model has ECE $= 0$, indicating precise alignment between predicted and actual probabilities.
  - Higher ECE values suggest miscalibration, revealing discrepancies between predicted confidence and true outcomes.

Introduction to ECE

ECE: Usage and Considerations

- **Usage:**
  - ECE provides a global measure of calibration across the entire range of predicted probabilities.
  - Visualization through a reliability diagram aids in understanding calibration performance.
- **Considerations:**
  - ECE is sensitive to bin sizes; proper binning is crucial for meaningful results.
  - Lower ECE values indicate better-calibrated models with more accurate confidence scores.

Introduction to Sparsification Error

Sparsification Error

- **Definition:** The Sparsification Error (SE) is a metric used to assess the correspondence between the uncertainty and the errors.
- **Importance:** Sparsification Error is crucial for models that provide confidence score estimates, ensuring that predicted confidence scores align with actual error.

Introduction to Sparsification Error

Sparsification Error Formula and Interpretation

- **Formula:** We begin by choosing an error metric (MSE, MAE, ...) then we partition the data into 2 sets of m bins:
  - one set is based on the confidence scores of the DNN predictions. B_i represents the collection of samples whose predicted probabilities fall within the i-th bin.
  - one set is based on the error scores of the DNN predictions. O_i represents the collection of samples whose predicted probabilities fall within the i-th bin.

$$\text{Sparsif. error} = \sum_{i=1}^{N} |\text{error}(\text{concat}(B_i, ..., B_N)) - \text{error}(\text{concat}(O_i, ..., O_N))|$$

- **Interpretation:**
  - A perfectly calibrated model has Sparsification error $= 0$, indicating precise alignment between the predicted confidence score and the DNN error.

Aleatoric Uncertainty Evaluation (Classification)

Quantifying Aleatoric Uncertainty (Classification)

- **Negative Log Likelihood (NLL):** Measures the likelihood of the true class under the predicted probability distribution.
- **Expected Calibration Error (ECE):** Measures the calibration of predicted probabilities against true outcomes.
- **Accuracy:** Essential for assessing the correctness of predictions.

Aleatoric Uncertainty Evaluation (Regression)

Quantifying Aleatoric Uncertainty (Regression)

- **Accuracy:** Essential for assessing the correctness of predictions.
- **Sparsification Error:** Measures the error introduced by discarding uncertain predictions.

Epistemic Uncertainty Evaluation (Classification)

Quantifying Epistemic Uncertainty (OOD Detection)

- Quantifying Epistemic Uncertainty is hard so often we consider just Out-of-Distribution (OOD) Detection.
- **Out-of-Distribution (OOD) Detection:** Evaluates the model's ability to detect samples outside the training distribution.
- **Transform to 2-Class Classification:** Detecting ID vs. OOD samples, based on DNN confidence scores.

Epistemic Uncertainty Evaluation (Classification)

Quantifying Epistemic Uncertainty (metrics)

We use metrics for binary classification assessment:

- **AUROC** (Area Under the Receiver Operating Characteristic Curve): Measures the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) across different probability thresholds. A higher value (closer to 1) indicates better performance.
- **AUPR** (Area Under the Precision-Recall Curve): Focuses on the precision-recall trade-off, emphasizing positive class prediction performance, especially in imbalanced datasets. A higher value (closer to 1) indicates better performance.
- **FPR95** (False Positive Rate at 95% True Positive Rate): Evaluates the model's performance at a high sensitivity level (95% true positive rate). A lower value (closer to 0) indicates better performance.

Epistemic Uncertainty Evaluation (regression)

Quantifying Epistemic Uncertainty

- We apply a similar evaluation approach as in the classification task to assess epistemic uncertainty in regression.
- It's important to note that defining Out-of-Distribution (OOD) instances can be challenging in regression tasks compared to classification.

Distribution Shift Evaluation (Classification)

Quantifying Distribution Shift

- **Accuracy, ECE and Sparsification Error:** Standard metrics for assessing classification and regression performance.
- **Transform to 2-Class Classification:** Detecting shifted vs. non-shifted samples.

Considerations and Challenges

Considerations and Challenges

- **Trade-offs:** Balancing between different metrics to obtain a comprehensive evaluation.
- **Context:** Metrics may vary based on the application, dataset, and specific requirements.
- **Interpretability:** Understanding the implications of uncertainty metrics for decision-making.

Importance of Quantifying Uncertainty in DNNs

Enhancing Reliability and Safety

- **Reliability:** Quantifying uncertainty in DNNs is crucial for improving the reliability of their predictions.
- **Safety:** Knowing the level of uncertainty allows for more cautious decision-making, contributing to safer and more robust AI.

Utilizing Uncertainty for Various Tasks

- **Anomaly Detection:** Uncertainty quantification aids in identifying unusual or unexpected patterns, enhancing anomaly detection capabilities.
- **Domain Adaptation:** Understanding uncertainty facilitates smoother adaptation of models to different domains.
- **Semi-Supervised Learning:** Uncertainty information can guide the pseudo-labeling of unlabeled data, enhancing semi-supervised learning approaches.
- **Open Set Classification:** Quantifying uncertainty is valuable for distinguishing between known and unknown classes in open set classification scenarios.

Key Takeaways

Summary of Insights

- **Understanding Sources:** We explored various sources impacting DNNs, acknowledging the inherent uncertainties.
- **Distinguishing Types:** The distinction between aleatoric and epistemic uncertainty provided clarity on different uncertainty manifestations.
- **Quantification Techniques:** We delved into diverse methods for quantifying uncertainty in DNNs.
- **Evaluation Approaches:** Different techniques for evaluating the effectiveness of uncertainty quantification were discussed.

Conclusions

Next Steps: Implementation

- **Practical Application:** Moving forward, we will explore practical implementations of uncertainty quantification.
- **Link for the Practical Application:** please visit the following link:

  `https://drive.google.com/file/d/1GpeHCq5bQDEusUtYHroGNIXDNW4fKMf1/view?usp=sharing`

Exploring Further

- **Contribute to Torch Uncertainty:** If you are interested in advancing the field, consider contributing to Torch Uncertainty.

  `https://github.com/ENSTA-U2IS/torch-uncertainty`

- **Explore Our Resources:** Check out our curated list of resources on Uncertainty, available at our "awesome" repository.

  `https://github.com/ENSTA-U2IS/awesome-uncertainty-deeplearning`

Conclusions

Tutorial Collab link

TorchUncertainty

Thanks for your attention!

Bibliography:

1 Pan, F., Shin, I., Rameau, F., Lee, S., & Kweon, I. S. (2020). Unsupervised intra-domain adaptation for semantic segmentation through self-supervision. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 3764-3773).

2 Sakaridis, Christos, Dengxin Dai, and Luc Van Gool. "ACDC: The adverse conditions dataset with correspondences for semantic driving scene understanding." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021.

3 Bergmann, P., Fauser, M., Sattlegger, D., & Steger, C. (2019). MVTec AD–A comprehensive real-world dataset for unsupervised anomaly detection. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 9592-9600).

4 Hendrycks, D., Basart, S., Mazeika, M., Zou, A., Kwon, J., Mostajabi, M., ... & Song, D. (2019). Scaling out-of-distribution detection for real-world settings. arXiv preprint arXiv:1911.11132.

Bibliography:

5 Chan, R., Lis, K., Uhlemeyer, S., Blum, H., Honari, S., Siegwart, R., ... & Rottmann, M. (2021). Segmentmeifyoucan: A benchmark for anomaly segmentation. arXiv preprint arXiv:2104.14812.

6 Franchi, G., Yu, X., Bursuc, A., Tena, A., Kazmierczak, R., Dubuisson, S., ... & Filliat, D. (2022). MUAD: Multiple Uncertainties for Autonomous Driving, a benchmark for multiple uncertainty types and tasks. arXiv preprint arXiv:2203.01437.

7 Gawlikowski, J., Tassi, C.R.N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R. and Shahzad, M., 2023. A survey of uncertainty in deep neural networks. Artificial Intelligence Review, 56(Suppl 1), pp.1513-1589.

8 Hullermeier, Eyke, and Willem Waegeman. "Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods." Machine Learning 110 (2021): 457-506.

Bibliography:

9 Guo, Chuan, et al. "On calibration of modern neural networks." Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017.

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

11 Havasi, M., Jenatton, R., Fort, S., Liu, J. Z., Snoek, J., Lakshminarayanan, B., ... & Tran, D. (2020). Training independent subnetworks for robust prediction. arXiv preprint arXiv:2010.06610.

12 Gal, Yarin, and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." international conference on machine learning. 2016.

13 Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles." Advances in neural information processing systems. 2017.

Bibliography:

14 Kendall, Alex, and Yarin Gal. "What uncertainties do we need in bayesian deep learning for computer vision?." Advances in neural information processing systems. 2017.

15 A.G. Wilson, P. Izmailov. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. Advances in Neural Information Processing Systems, 2020.

16 Ilg, E., Cicek, O., Galesso, S., Klein, A., Makansi, O., Hutter, F., & Brox, T. (2018). Uncertainty estimates and multi-hypotheses networks for optical flow. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 652-667).

17 Blundell, Charles, et al. "Weight uncertainty in neural networks." arXiv preprint arXiv:1505.05424 (2015).

18 Fort, Stanislav, Huiyi Hu, and Balaji Lakshminarayanan. "Deep Ensembles: A Loss Landscape Perspective." arXiv preprint arXiv:1912.02757 (2019).

Bibliography:

19 Yu, Xuanlong, Gianni Franchi, and Emanuel Aldea. "SLURP: Side learning uncertainty for regression problems." arXiv preprint arXiv:2110.11182 (2021).

20 Upadhyay, U., Karthik, S., Chen, Y., Mancini, M., & Akata, Z. (2022, October). Bayescap: Bayesian identity cap for calibrated uncertainty in frozen neural networks. In European Conference on Computer Vision (pp. 299-317). Cham: Springer Nature Switzerland.

21 A.G. Wilson, P. Izmailov. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. Advances in Neural Information Processing Systems, 2020.

22 Wen, Yeming, Dustin Tran, and Jimmy Ba. "Batchensemble: an alternative approach to efficient ensemble and lifelong learning." arXiv preprint arXiv:2002.06715 (2020).

Bibliography:

22 Wen, Yeming, Dustin Tran, and Jimmy Ba. "Batchensemble: an alternative approach to efficient ensemble and lifelong learning." arXiv preprint arXiv:2002.06715 (2020).

23 Hendrycks, Dan, et al. "A Benchmark for Anomaly Segmentation." arXiv preprint arXiv:1911.11132 (2019).

24 Franchi, G., Bursuc, A., Aldea, E., Dubuisson, S., & Bloch, I. (2020). TRADI: Tracking deep neural network weight distributions. In ECCV 2020.

25 Laurent, O., Lafage, A., Tartaglione, E., Daniel, G., Martinez, J. M., Bursuc, A., and Franchi, G. (2022). Packed-Ensembles for Efficient Uncertainty Estimation. In ICLR 2023.

26 Bishop, C. M. (1994). Mixture density networks. Aston University.

27 Vossen, J., Feron, B., Monti and A. (2018). Probabilistic Forecasting of Household Electrical Load Using Artificial Neural Networks.

28 Lafage, A., Barbier, M., Franchi, G., and Filliat, D. (2024). Hierarchical Light Transformer Ensembles for Multimodal Trajectory Forecasting.