Uncertainty Quantification in Deep Learning

Gianni FRANCHI

U2IS, ENSTA Paris

Presentation 2023

Plan

What is uncertainty in machine/deep learning

- We make observations using the sensors in the world (e.g. camera)
- Based on the observations, we intend to learn a model that makes decisions
- Given the **same** observations, the decision should be the **same**

However,

- The world **changes**, observations **change**, our sensors **change**, the output should not change!
- We would like to know how confident we can be about the decisions

Why Uncertainty is important?

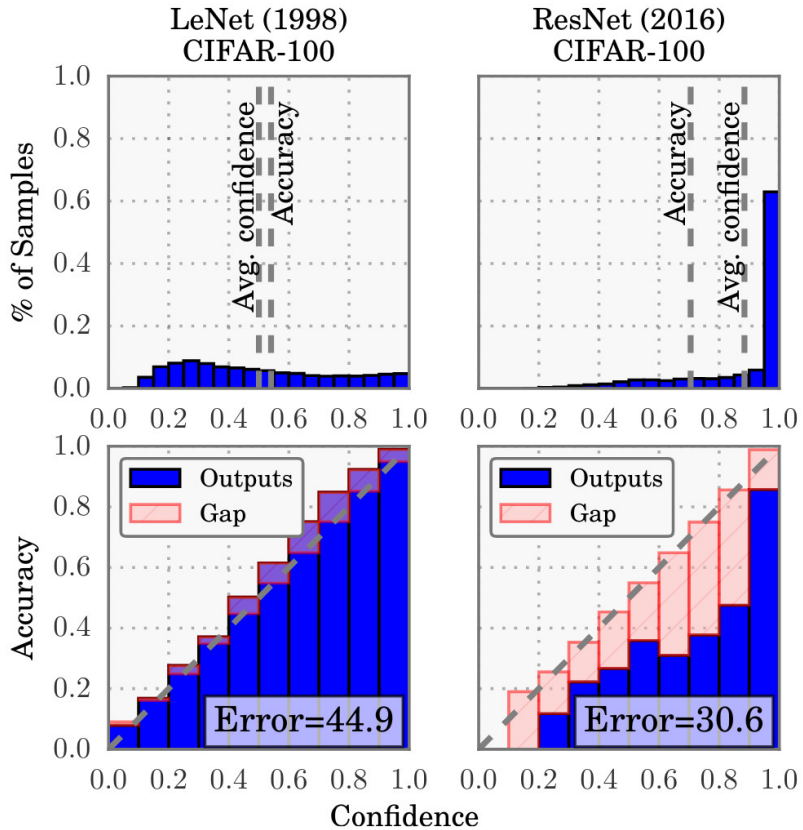
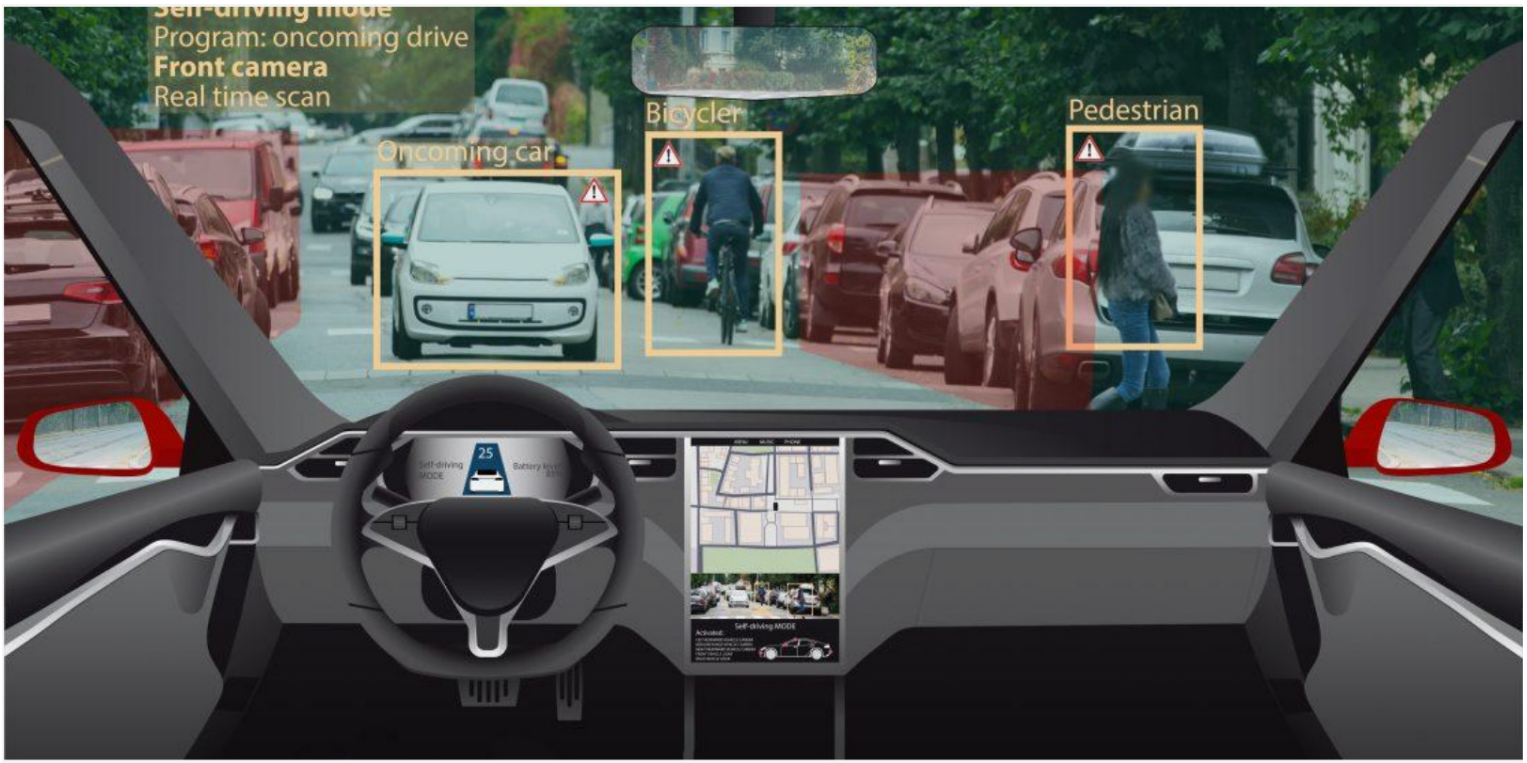


Figure: Confidence histograms (top) and reliability diagrams(bottom) for a 5-layer LeNet (left) and a 110-layer ResNet (right)on CIFAR-100. [1]

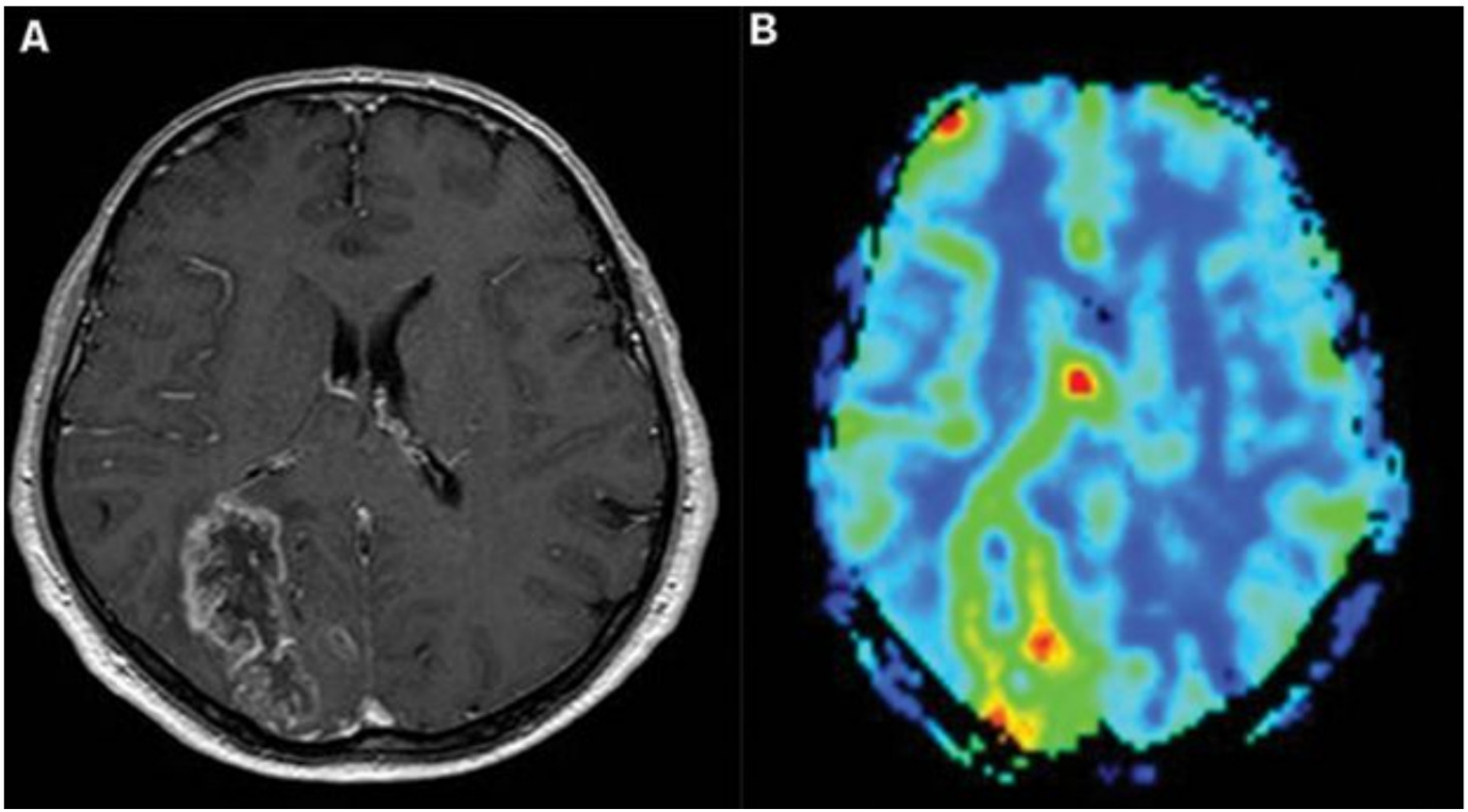
Why Uncertainty is important?

Imagine an autonomous car with a perception system based on Deep learning without Uncertainty:



Why Uncertainty is important?

Imagine a medical diagnostics based on Deep learning without Uncertainty:

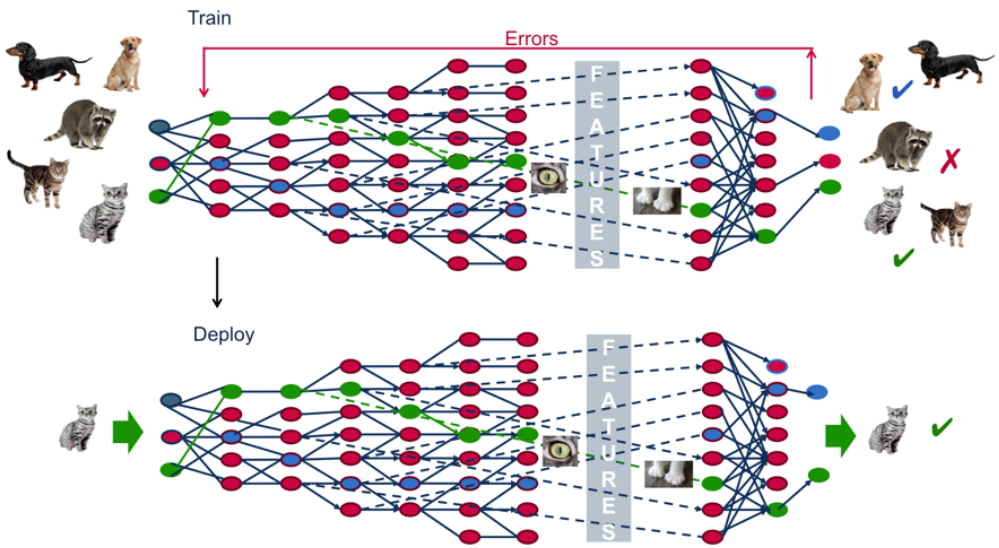


Why Uncertainty is important?

We build models for predictions, can we trust them? Are they certain?

Deep Learning

Deep learning systems are neural network (or convolutional neural network) models similar to those popular in the '80s and '90s, with algorithmic inovations, software inovations, and larger data sets.

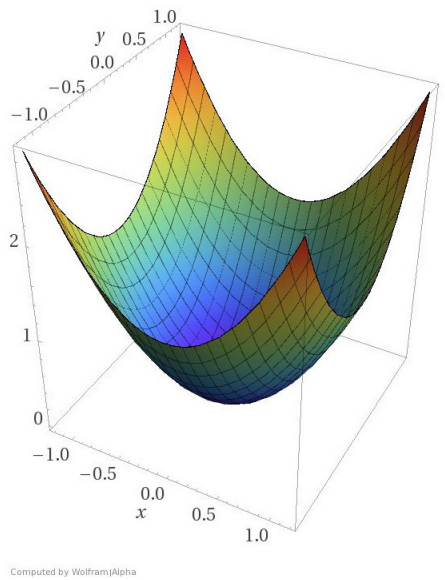


Deep Learning notations

- Training/Testing sets are denoted respectively by $\mathcal{D}_l = (x_i, y_i)_{i=1}^{n_l}$, $\mathcal{D}_\tau = (x_i, y_i)_{i=1}^{n_\tau}$. Without loss of generality we consider the observed samples $\{x_i\}_{i=1}^{n}$ and the corresponding labels $\{y_i\}_{i=1}^{n}$ as vectors. Data in $\mathcal{D}_l$ and $\mathcal{D}_\tau$ are assumed to be i.i.d. distributed according to their respective unknown joint distribution $\mathcal{P}_t$ and $\mathcal{P}_\tau$.
- The Deep Neural Networks (DNN) are function parameterized by a vector containing the K trainable weights $\boldsymbol{\omega} = \{\omega_k\}_{k=1}^{K}$.
- During training, $\boldsymbol{\omega}$, is iteratively updated for each mini-batch and we denote by $\boldsymbol{\omega}(t)$ the state of the DNN at iteration t of the optimization algorithm, and following the random variable $W(t)$.
- g represents the architecture of the DNN associated with these weights and $g_{\boldsymbol{\omega}(t)}(x_i)$ its output at t.

Deep Learning optimization

We denote: $\mathcal{L}(\omega(t), y_i)$ the loss function used to measure the dissimilarity between the output $g_{\omega(t)}(x_i)$ of the DNN and the expected output y_i. One can use different loss functions.
We use can use a gradient descent to optimize $\omega(t)$



However, for large neural networks with a large training set, computing the gradient is costly, and the loss is not convex.

Deep Learning optimization

For that, we consider **stochastic gradient descent** SGD algorithm on a mini-batch in order to optimize the loss between two weight realizations. The loss derivative with respect to a given weight $\omega_k(t)$ on a mini-batch $B(t)$ is given by:

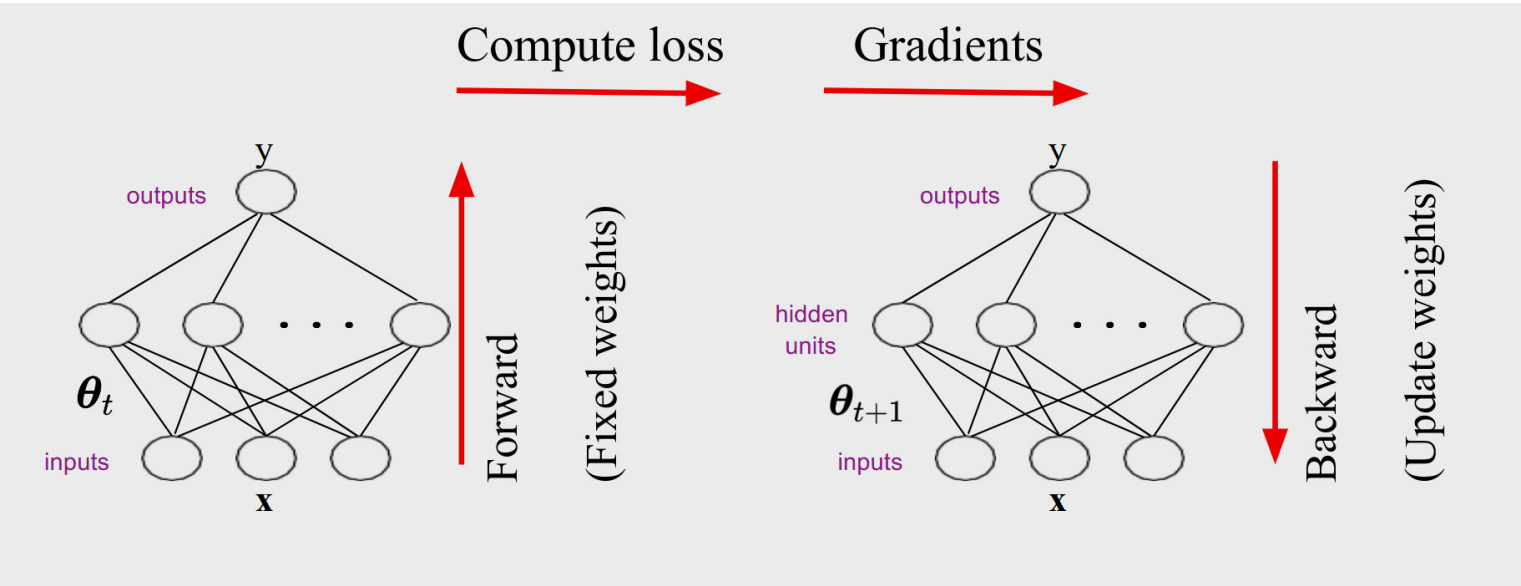
$$\nabla \mathcal{L}_{\omega_k(t)} = \frac{1}{|B(t)|} \sum_{(x_i, y_i) \in B(t)} \frac{\partial \mathcal{L}(\boldsymbol{\omega}(t-1), y_i)}{\partial \omega_k(t-1)} \tag{1}$$

Weights $\omega_k(t)$ are then updated as follows:

$$\omega_k(t) = \omega_k(t-1) - \eta \nabla \mathcal{L}_{\omega_k(t)} \tag{2}$$

with η the learning rate.

Deep Learning optimization



Deep Learning optimization

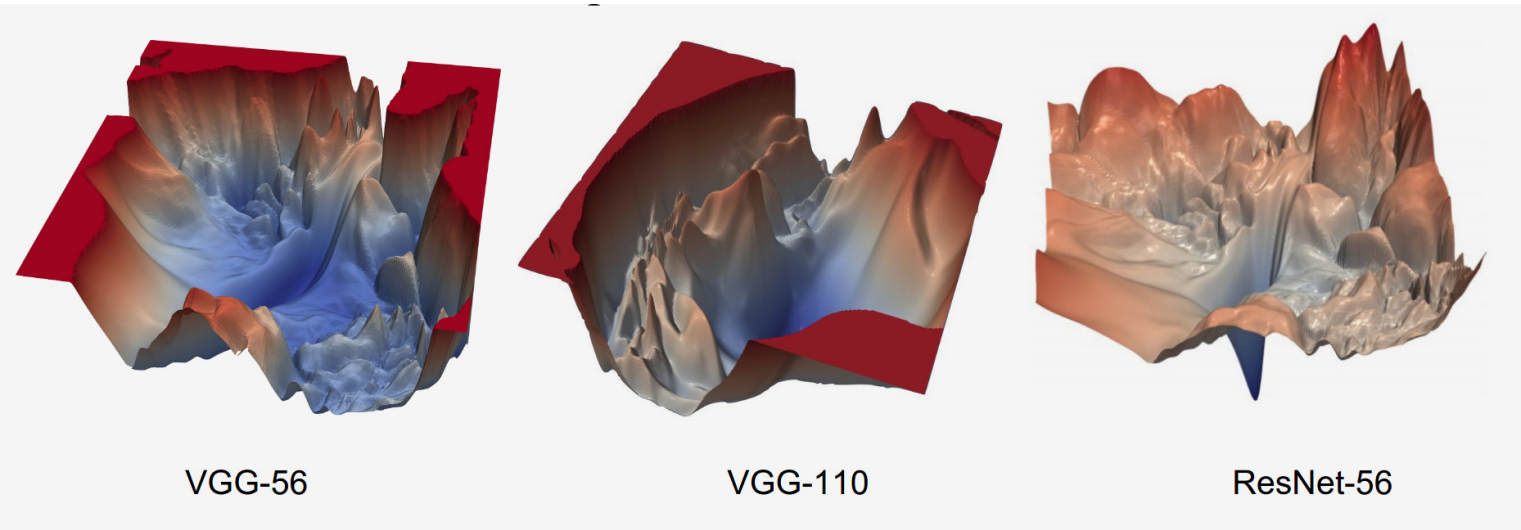
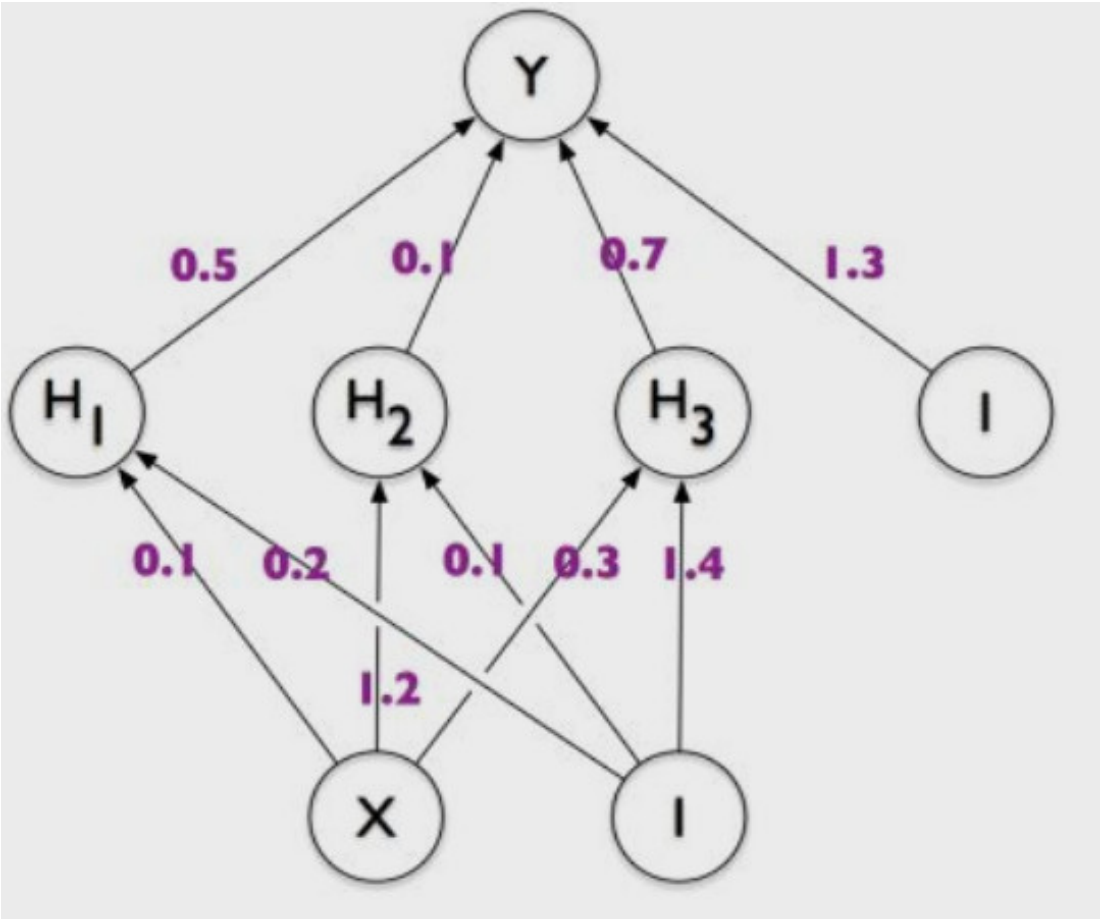


Figure: Visualizing the loss surfaces of modern DNN [3]

Deep Learning testing

When we test a DNN on new data we just test with the optimal $\omega(t^*)$ (one realisation $W(t^*)$)



Types of Uncertainty

- Aleatoric: Uncertainty inherent in the observation noise (problems caused by sensor quality, natural randomness, that cannot be explained by our data).
- Epistemic: Our ignorance about the correct model that generated the data (lack of knowledge about the process that generated the data).

Aleatoric uncertainty

Aleatoric uncertainty captures noise inherent in the observations:

- For example, sensor noise or motion noise result in uncertainty.
- This uncertainty cannot be reduced with more data.
- However, aleatoric could be reduced with better measurements.

Aleatoric uncertainty

Aleatoric uncertainty can further be categorized into homoscedastic and heteroscedastic uncertainties:

- Homoscedastic uncertainty relates to the uncertainty that a particular task might cause. It stays constant for different inputs.
- Heteroscedastic uncertainty depends on the inputs to the model, with some inputs potentially having more noisy outputs than others.

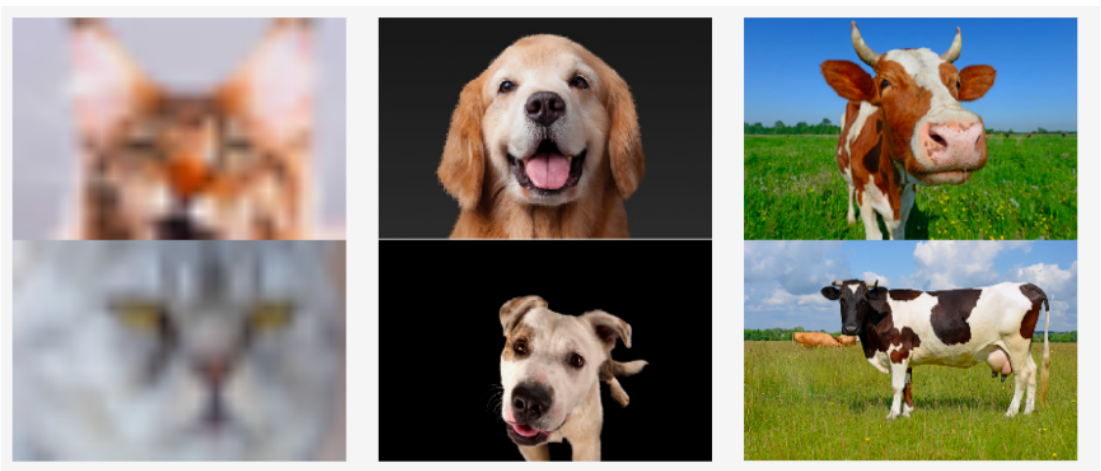
Types of Uncertainty: Case 1[1]

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

Aleatoric loss[6]

We model aleatoric uncertainty in the output by modeling the conditional distribution as a Normal distribution. We want the CNN to predict:

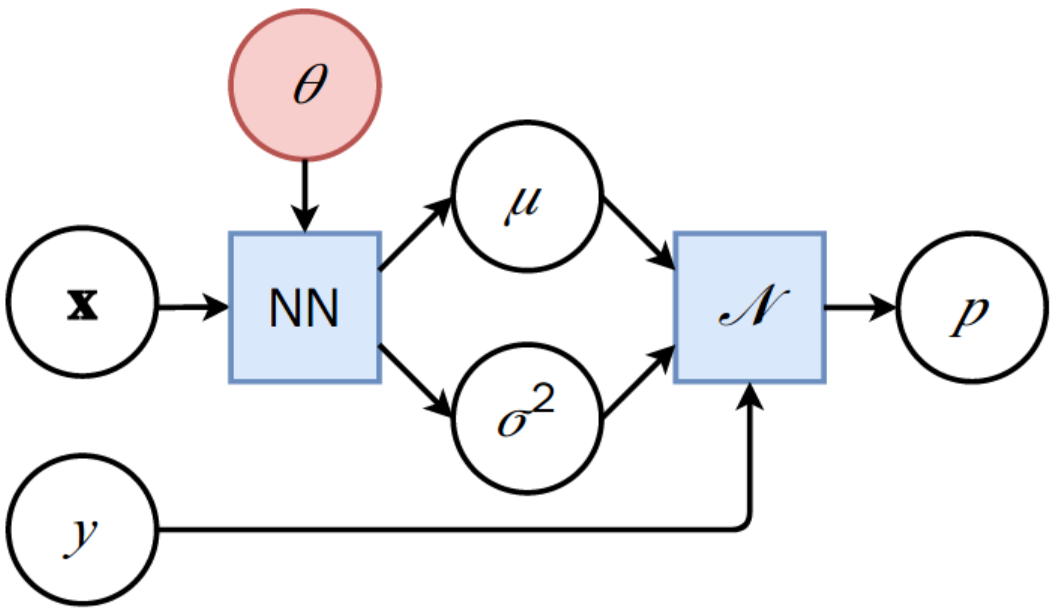
$$\mathcal{P}(Y|X,\boldsymbol{\omega}) = \mathcal{N}\left(\mu(X,\boldsymbol{\omega}); \sigma^2(\mu(X,\boldsymbol{\omega}))\right) \tag{3}$$

where $\mu(X,\boldsymbol{\omega})$ and $\sigma^2(\mu(X,\boldsymbol{\omega})$ are parametric functions to be learned by a CNN. We do not wish to learn just one function $f(X,\boldsymbol{\omega})$ that would only produce point estimates. If $\sigma^2(\mu(X,\boldsymbol{\omega}))$ is independent of x we deal with the Homoscedastic uncertainty.

Heteroscedastic loss[6][3]

We train ω such that $\mu(X, \omega)$ and $\sigma^2(\mu(X, \omega))$ optimize this loss

$$\mathcal{L}(Y|X, \omega) = \sum_i \frac{\|\mu(x_i, \omega) - y_i\|^2}{2\sigma^2(x_i, \omega)} + \log(\sigma^2(x_i, \omega)) + Cst \tag{4}$$



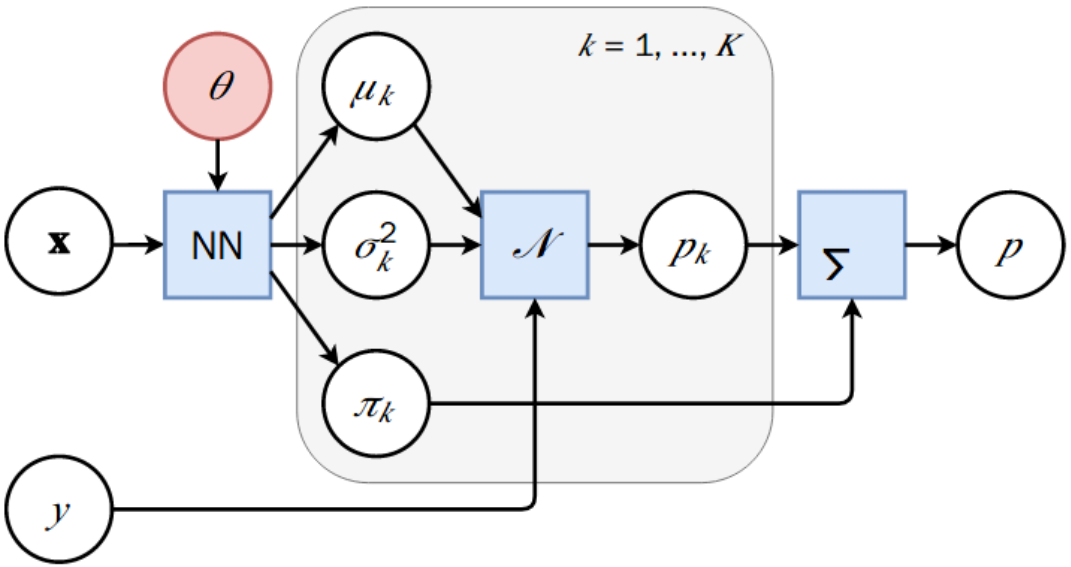
[3] Credits: Gille Louppe

Mixture density network[12][4]

A mixture density network is a neural network implementation of the Gaussian mixture model

$$\mathcal{P}(Y|X) = \sum_k \pi_k \mathcal{N}(y|\mu_k, \sigma_k^2) \tag{5}$$

With $0 \leq pi_k \leq 1$ and $\sum_k \pi_k = 1$



[4]Credits: Gille Louppe

Calibration

The classical loss used is the cross entropy:

$$\mathcal{L}_{\text{cross entropy}}(\mathcal{D}_I, \boldsymbol{\omega}) = -1/N \sum_{i}^{N} y_i \log(f(x_i, \boldsymbol{\omega})) \tag{6}$$

- y_i is the ground truth label corresponding to probability density expressed as one hot-vector
- $f(x_i, \boldsymbol{\omega})$ is the predicted class transformed into probability via softmax

Calibration with temperature scaling

$f(x_i, \boldsymbol{\omega}) \in \mathbb{R}^K$ is the predicted class transformed into probability via softmax let us write $g(x_i, \boldsymbol{\omega}) \in \mathbb{R}^K$ the logit just before the softmax.

$$f(x_i, \boldsymbol{\omega})_k = \frac{\exp(g(x_i, \boldsymbol{\omega})_k)}{\sum_k^K \exp(g(x_i, \boldsymbol{\omega})_k)} \tag{7}$$

Temperature scaling is to use a scalar parameter $T > 0$ called the temperature for all classe that soften the softmax:

$$\tilde{f}(x_i, \boldsymbol{\omega})_k = \frac{\exp(g(x_i, \boldsymbol{\omega})_k / T)}{\sum_k^K \exp(g(x_i, \boldsymbol{\omega})_k / T)} \tag{8}$$

Calibration

Some papers prefer to learn the loss or the confidence of the prediction.

- learning loss for active learning [7]
- learning confidence for OOD detection [8]

Bayesian approach and DNN

The Goal of DNN is to find $\mathcal{P}(Y|X,\omega)$, most of the classical approaches find ω that maximize the likelihood.

$$\omega = \arg\max_{\omega} \log \mathcal{P}(\mathcal{D}_l|\omega)$$

$$\omega = \arg\max_{\omega} \sum_{i=1}^{n_l} \log \mathcal{P}(Y_i|X_i,\omega)$$

$$\omega = \arg\max_{\omega} 1/n_l \sum_{i=1}^{n_l} \log \mathcal{P}(Y_i|X_i,\omega)$$

$$\omega = \arg\max_{\omega} \mathbb{E}_{(X,Y)\sim\mathcal{P}(\mathcal{D}_l)} \log \mathcal{P}(Y|X,\omega)$$

$$\omega = \arg\min_{\omega} H\left[\mathcal{P}(\mathcal{D}_l), \mathcal{P}(Y|X,\omega)\right]$$

With H the cross entropy.

Bayesian approach and DNN

The Goal of DNN is to find $\mathcal{P}(Y|X, \boldsymbol{\omega})$. In the classical bayesian approach we find $\boldsymbol{\omega}$ such that we have the maximum a posteriori (MAP).

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \log \mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_l)$$
$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \log \mathcal{P}(\mathcal{D}_l|\boldsymbol{\omega}) + \log \mathcal{P}(\boldsymbol{\omega})$$

This leads to l2 regularization.

Bayesian DNN

Bayesian DNN is based on marginalization instead of MAP optimization.

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

In practice:

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

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

Variational inference

Variational inference approximates the posterior $\mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_I)$ with a family of distributions $q_\lambda(\boldsymbol{\omega}/\mathcal{D}_I)$ The variational parameter λ indexes the family of distributions. For example, if q were Gaussian, it would be the mean and variance of the latent variables for each datapoint $\lambda_{x_i} = (\mu_{x_i}, \sigma^2_{x_i})$).
Question : How can we know how well our variational posterior $q_\lambda(w/\mathcal{D}_I)$ approximates the true posterior $\mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_I)$?

Variational inference

Question : How can we know how well our variational posterior $q_\lambda(\omega/\mathcal{D}_I)$ approximates the true posterior $\mathcal{P}(\omega|\mathcal{D}_I)$?
We can use the Kullback-Leibler divergence, which measures the information lost when using q to approximate $\mathcal{P}$:

$$\begin{aligned}\mathbb{KL}(q_\lambda(\omega/\mathcal{D}_I) \,||\, \mathcal{P}(\omega|\mathcal{D}_I)) &= \int_\omega \left(q_\lambda(\omega/\mathcal{D}_I) \log(\frac{q_\lambda(\omega/\mathcal{D}_I)}{\mathcal{P}(\omega|\mathcal{D}_I)}) \right) d\omega \\ &= \int_\omega \left(q_\lambda(\omega/\mathcal{D}_I) \log(\frac{q_\lambda(\omega/\mathcal{D}_I)}{\mathcal{P}(\mathcal{D}_I)\mathcal{P}(\mathcal{D}_I,\omega)}) \right) d\omega \\ &= \mathrm{E}_q[\log q_\lambda(\omega/\mathcal{D}_I)] - \mathrm{E}_q[\log \mathcal{P}(\omega,\mathcal{D}_I)] + \log \mathcal{P}(\mathcal{D}_I)\end{aligned}$$

Our goal is to find the variational parameters λ that minimize this divergence. The optimal approximate posterior is thus

Variational inference

The optimal approximate posterior is thus

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

This impossible to compute directly due to $\mathcal{P}(\mathcal{D}_l)$ that appears in the divergence. So, we consider the following function:

$$\begin{aligned} ELBO(\lambda) &= \mathrm{E}_q[\log \mathcal{P}(\boldsymbol{\omega}, \mathcal{D}_l)] - \mathrm{E}_q[\log q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l)] \\ &= -\int_{\boldsymbol{\omega}} \left(q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l) \log(\frac{q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l)}{\mathcal{P}(\boldsymbol{\omega})\mathcal{P}(\mathcal{D}_l|\boldsymbol{\omega})}) \right) d\boldsymbol{\omega} \\ &= \mathrm{E}_q[\log \mathcal{P}(\mathcal{D}_l|\boldsymbol{\omega})] - \mathbb{KL}(q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l) \,||\, \mathcal{P}(\boldsymbol{\omega})) \end{aligned}$$

Note that $\mathbb{KL}(q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l) \,||\, \mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_l)) = \log \mathcal{P}(\mathcal{D}_l) - ELBO(\lambda)$.

Variational inference: Reparametrization trick

theorem: Let ϵ be a random variable having a probability density given by $q(\epsilon)$ and let $\boldsymbol{\omega} = t(\lambda, \epsilon)$. Suppose that $q_\lambda(\boldsymbol{\omega}/\mathcal{D}_I)$, is such that $q(\epsilon)d\epsilon = q_\lambda(\boldsymbol{\omega}/\mathcal{D}_I)d\boldsymbol{\omega}$. Then for a function f with derivatives in $\boldsymbol{\omega}$:

$$\frac{\partial}{\partial \lambda} \mathsf{E}_{q_\lambda(\boldsymbol{\omega}/\mathcal{D}_I)} f(\boldsymbol{\omega}, \lambda) = \mathsf{E}_{q(\epsilon)} \left[\frac{\partial f(\boldsymbol{\omega}, \lambda)}{\partial \boldsymbol{\omega}} \frac{\partial \boldsymbol{\omega}}{\partial \lambda} + \frac{\partial f(\boldsymbol{\omega}, \lambda)}{\partial \lambda} \right].$$

Variational inference [9]

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

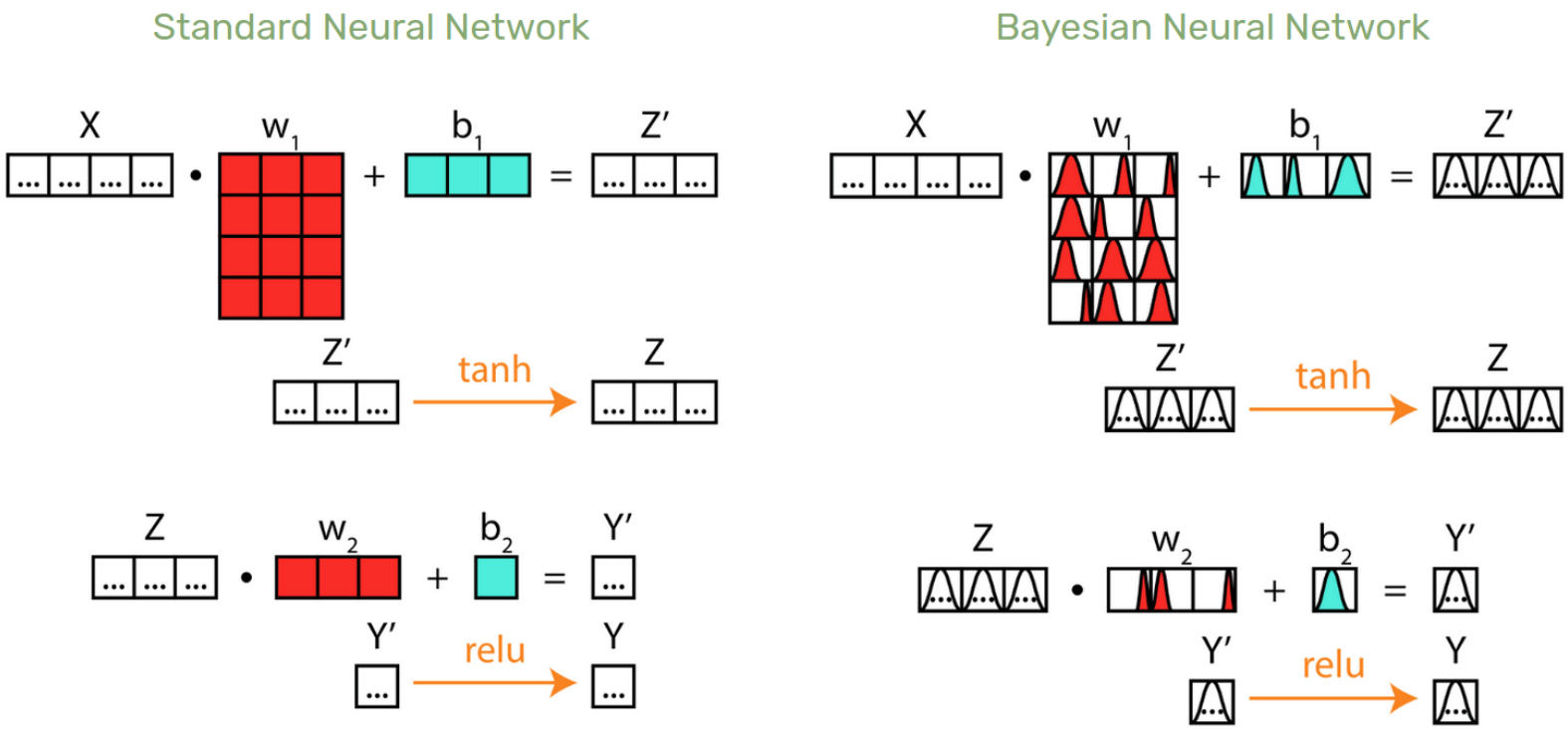
$$\Delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}. \tag{4}$$

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 [9][5]

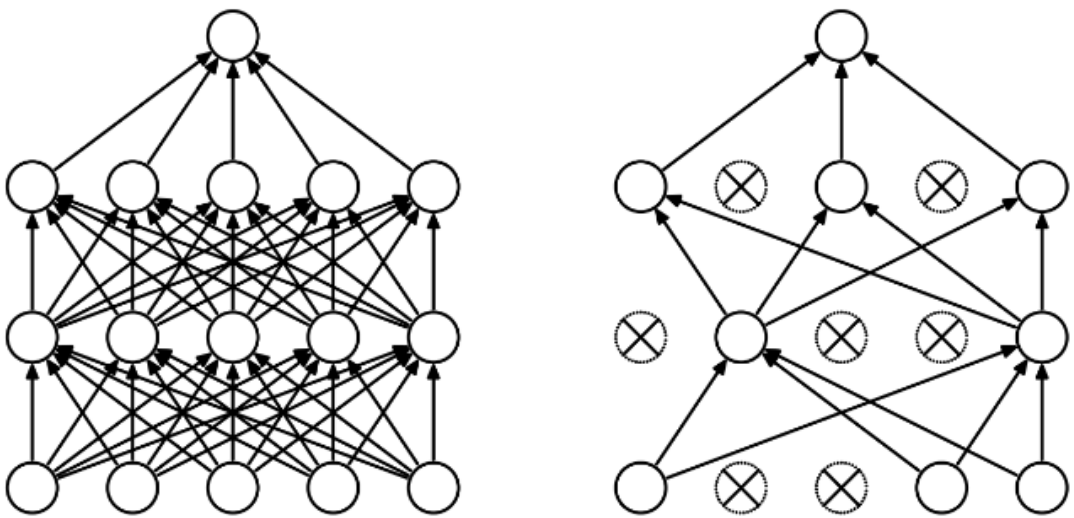


[5] Image credit: Eric Ma

Dropout[6]

Dropout is an empirical technique that was proposed to avoit overfitting in CNN.
At each training step (i.e., for each sample within a mini-batch)

- Remove each node in the network with a probability p
- Update the weights of the remaining nodes with backpropagation.

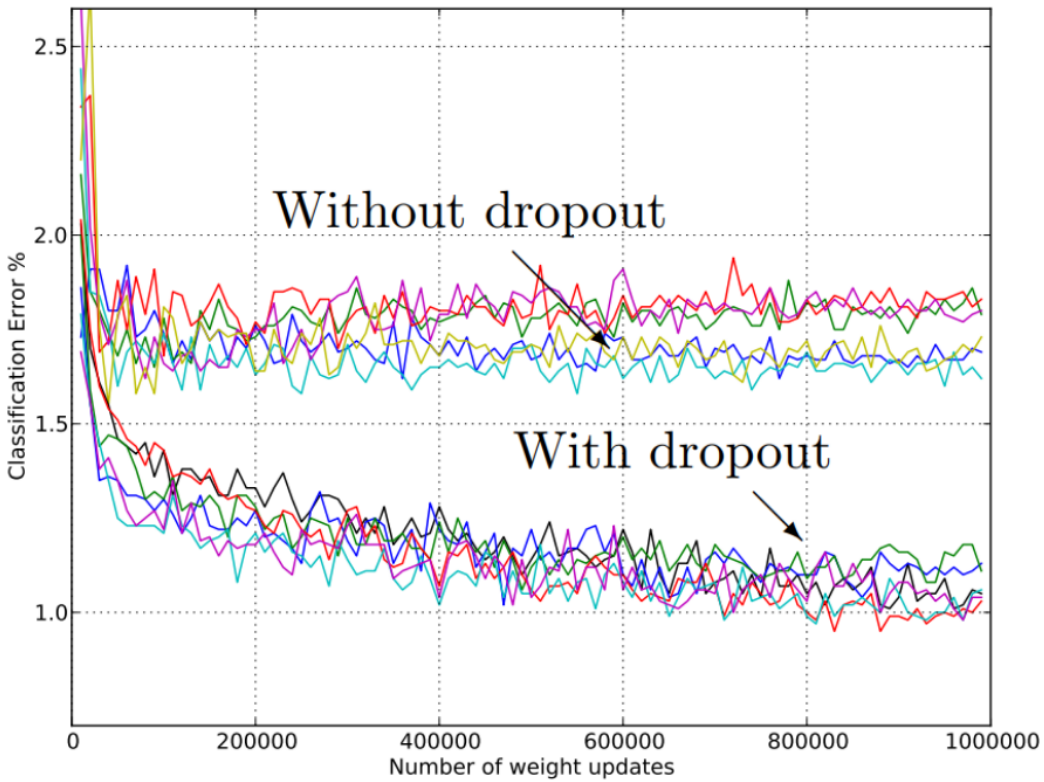


(a) Standard Neural Net

(b) After applying dropout.

[6]Image credit: G. Louppe

MC dropout[7]



Why does dropout work?

[7] Image credit: G. Louppe

MC dropout [4][8]

Dropout does variational inference.

Let us split the weights ω per layer $\omega = \{\omega_1, \ldots, \omega_L\}$ (L is the number of Layer). Let us also further split each layer to unit $\omega_l = \{\omega_{l,1}, \omega_{l,q_1}\}$. Variational parameters λ are split similarly into $\lambda = \{M_1, \ldots, M_L\}$, with $M_l = \{m_{l,1}, m_{l,q_1}\}$.
Then, the proposed q is :

$$q_\lambda(\omega/\mathcal{D}_l) = \prod_l^L q(\omega_l, M_l)$$

$$\text{with } q(\omega_l, M_l) = \prod_i^{q_l} q(\omega_{l,i}, m_{l,i})$$

$$\text{with } q(\omega_{l,i}, m_{l,i}) = p\delta_O(\omega_{l,i}) + (1-p)\delta_{m_{l,i}}(\omega_{l,i})$$

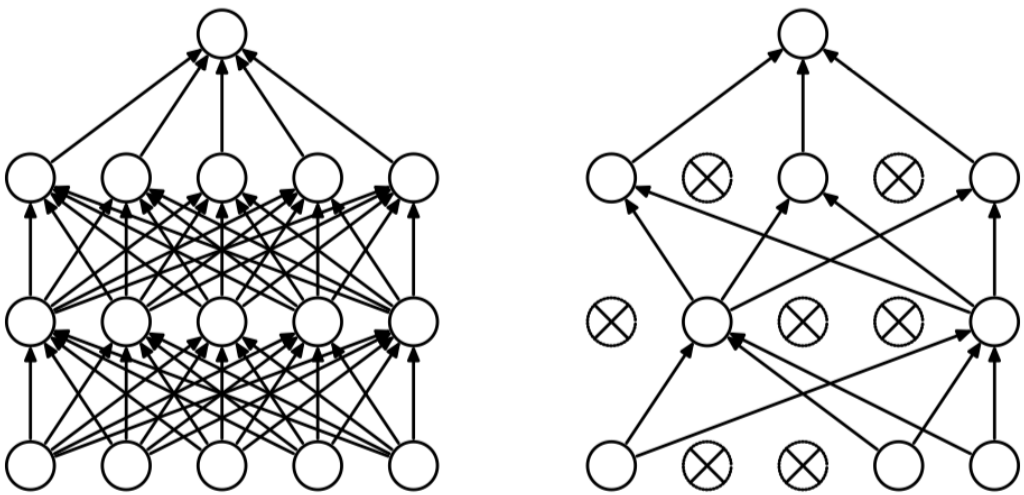
[8] Image credit: G. Louppe

MC dropout [4]

They [4] propose to average the predictions of several DNN where they apply the dropout:

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

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



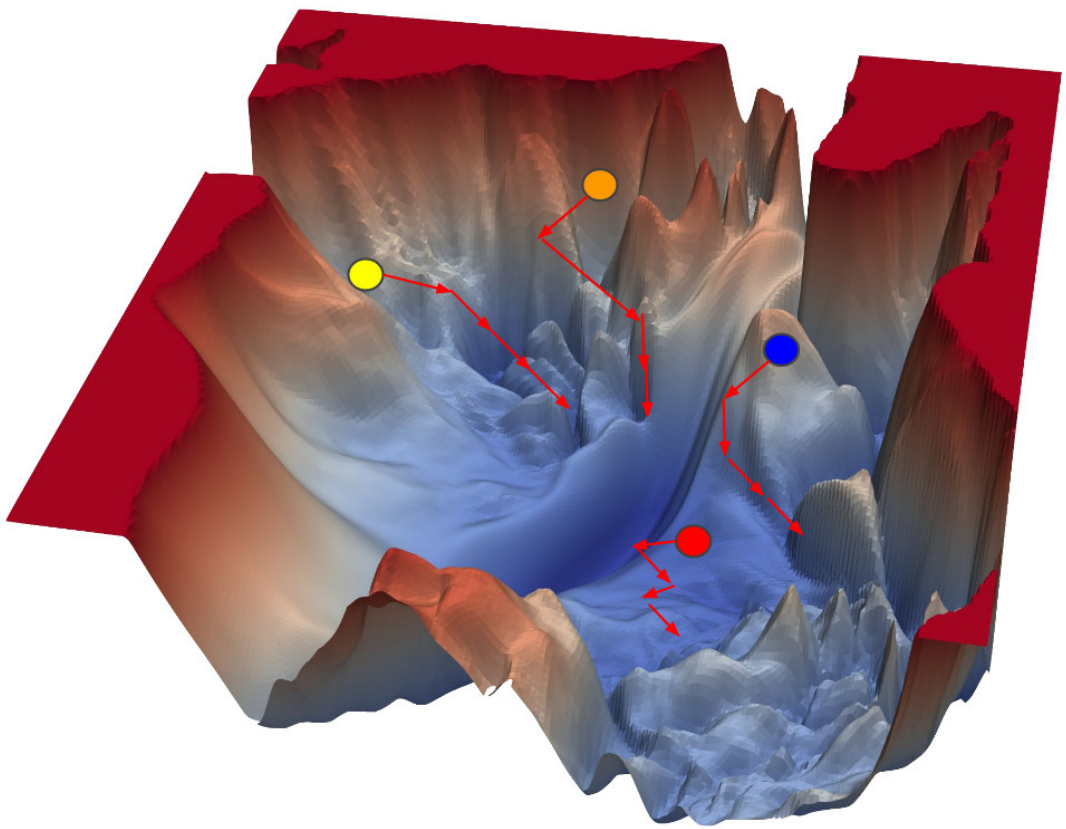
(a) Standard Neural Net

(b) After applying dropout.

Deep Ensembles[5]

They [5] propose to average the predictions of several DNN with different initial seeds:

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



Deep Ensembles[10]

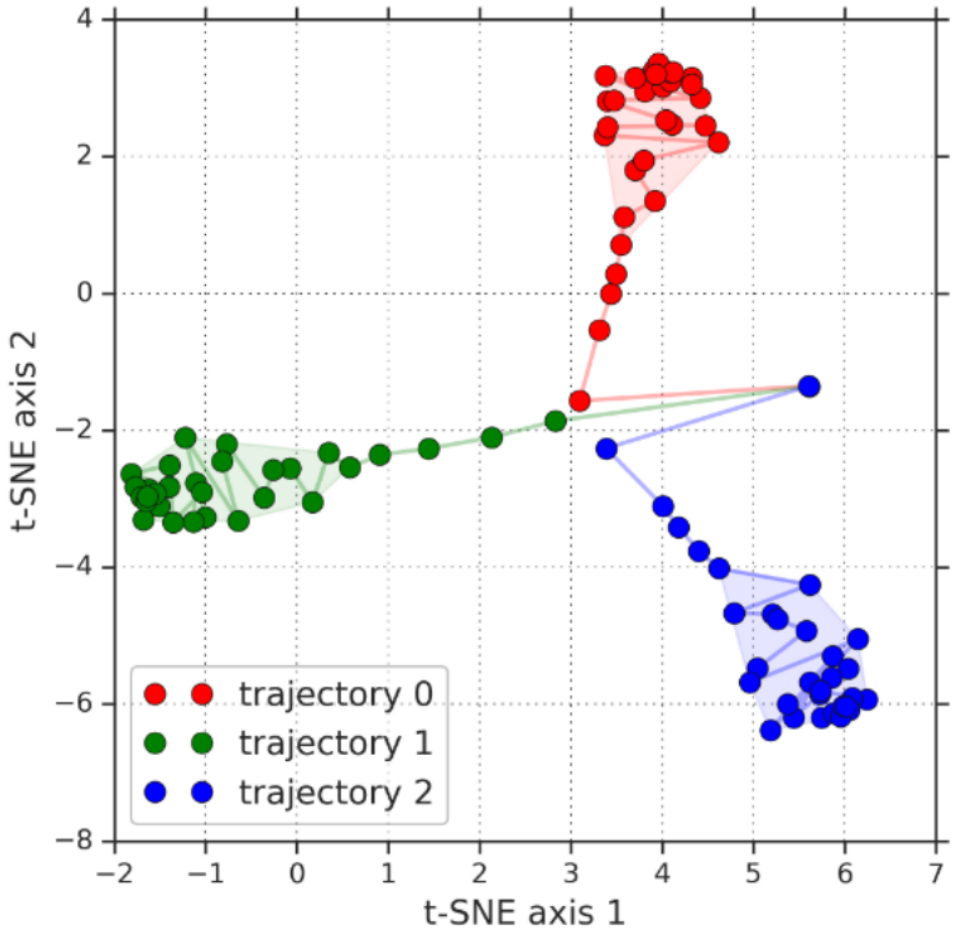


Figure: t-SNE plot of predictions from checkpoints corresponding to 3 different randomly initialized trajectorie

Deep Ensembles[10]

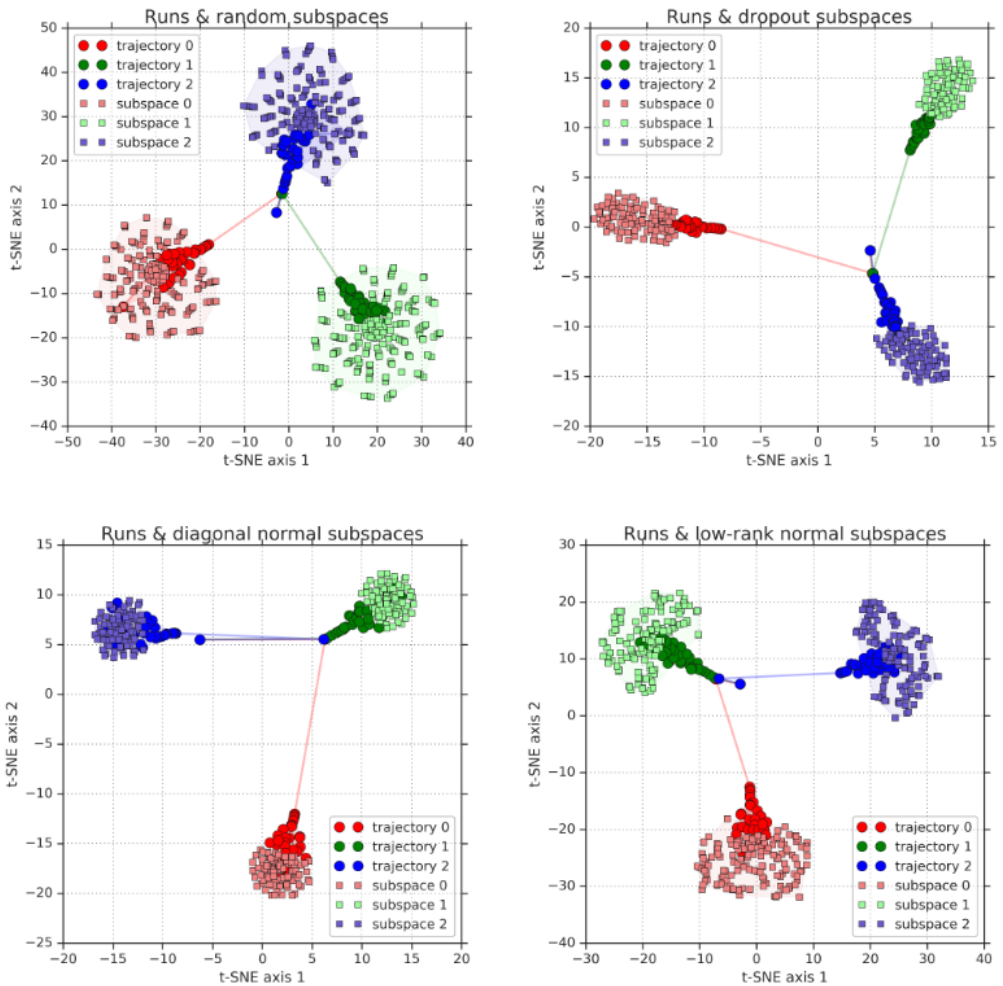


Figure: Results using SimpleCNN on CIFAR-10: t-SNE plots of validation set predictions for each trajectory along with four different subspace generation methods

Deep Ensembles[10]

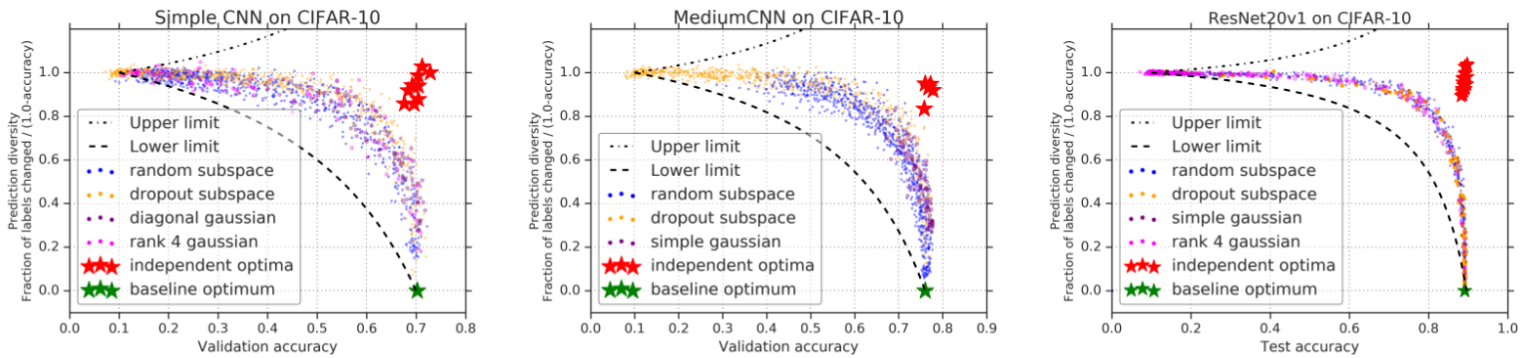


Figure: Diversity versus accuracy plots for 3 models trained on CIFAR-10

BatchEnsemble[14]

They [14] 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 $\omega = \{\omega_j\}_{j=1}^{N_{\text{model}}} = \{W_j\}_{j=1}^{N_{\text{model}}}$ and $\omega^{\text{slow}} = W$ and $\omega^{\text{slow}} = \{F_j\}_{j=1}^{N_{\text{model}}}$. We have $W_j = W \cdot F = W \cdot (r_j s_j^t)$

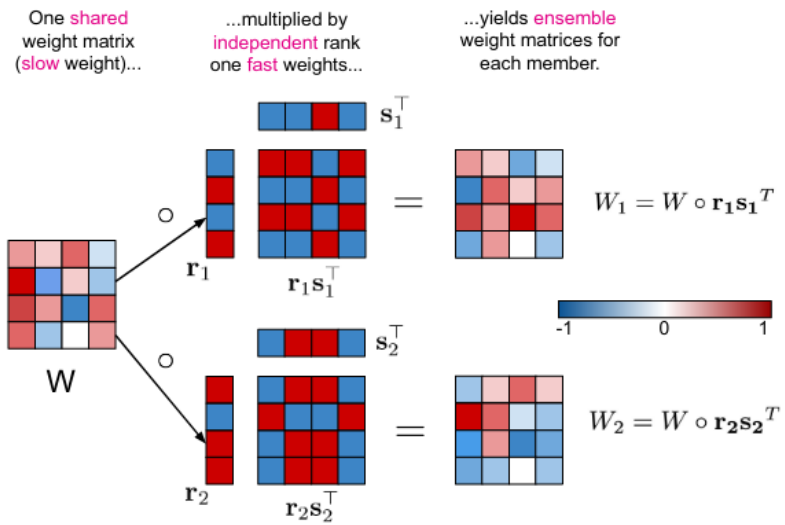


Figure: An illustration on how to generate the ensemble weights for two ensemble members

BatchEnsemble[14]

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:

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

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

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

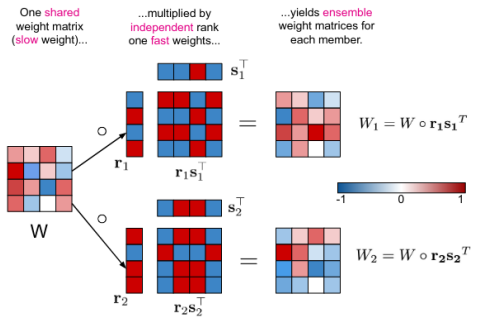


Figure: An illustration on how to generate the ensemble weights for two

TRADI

- $\boldsymbol{\omega}(0)$ is the initial set of weights $\{\omega_k(0)\}_{k=1}^K$ following $\mathcal{N}(0, \sigma_k^2)$, where σ_k^2 are fixed as in [2].
- $\mathcal{L}(\boldsymbol{\omega}(t), y_i)$ is the loss function used to measure the dissimilarity between the output $g_{\boldsymbol{\omega}(t)}(x_i)$ of the DNN and the expected output y_i. One can use different loss functions.
- Weights on different layers are assumed to be independent of one another at all times.
- Each weight $\omega_k(t)$, $k = 1, \ldots, K$, follows a non-stationary Normal distribution (e.g. $W_k(t) \sim \mathcal{N}(\mu_k(t), \sigma_k^2(t))$) whose two parameters are tracked.

TRADI

We had following state and measurement equations for the mean $\mu_k(t)$:

$$\begin{cases} \mu_k(t) = \mu_k(t-1) - \eta \nabla \mathcal{L}_{\omega_k(t)} + \varepsilon_\mu \\ \omega_k(t) = \mu_k(t) + \tilde{\varepsilon}_\mu \end{cases} \tag{12}$$

with ε_μ being the state noise, and $\tilde{\varepsilon}_\mu$ being the observation noise, as realizations of $\mathcal{N}(0, \sigma_\mu^2)$ and $\mathcal{N}(0, \tilde{\sigma}_\mu^2)$ respectively.

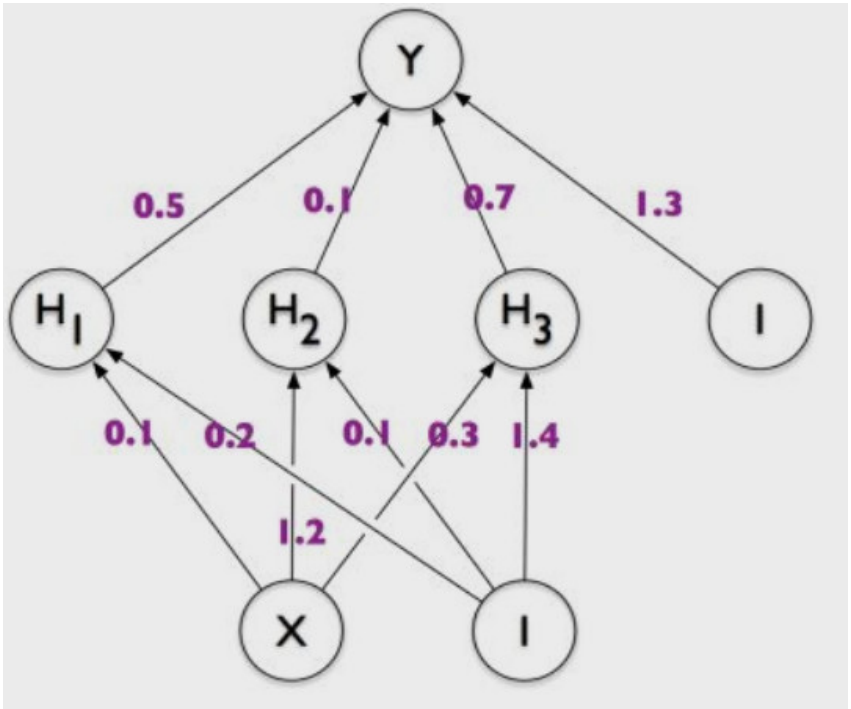
TRADI

The state and measurement equations for the variance σ_k are given by:

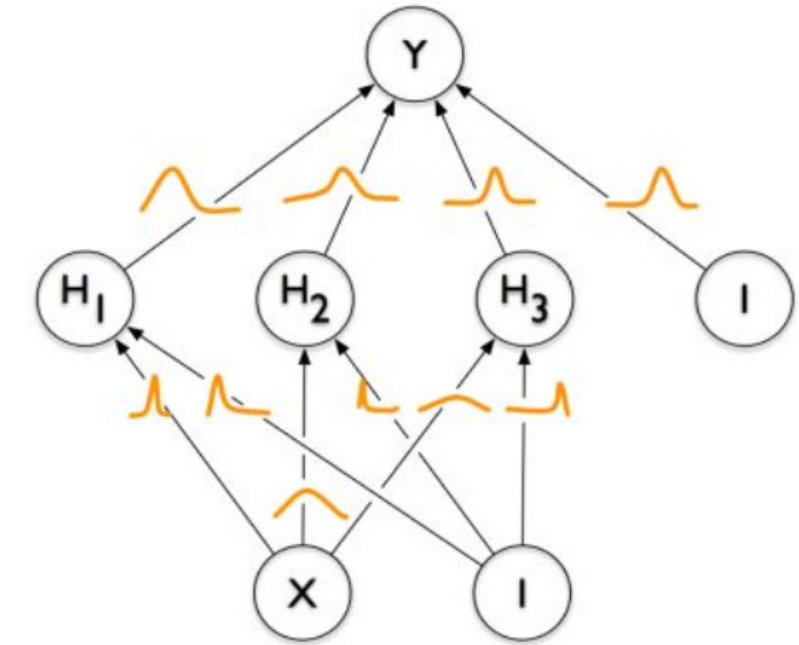
$$\begin{cases} \sigma_k^2(t) = \sigma_k^2(t-1) + \left(\eta \nabla \mathcal{L}_{\omega_k(t)}\right)^2 - \eta^2 \mu_k(t)^2 + \varepsilon_\sigma \\ z_k(t) = \sigma_k^2(t) - \mu_k(t)^2 + \tilde{\varepsilon}_\sigma \\ \text{with } z_k(t) = \omega_k(t)^2 \end{cases} \tag{13}$$

with ε_σ being the state noise, and $\tilde{\varepsilon}_\sigma$ being the observation noise, as realizations of $\mathcal{N}(0, \sigma_\sigma^2)$ and $\mathcal{N}(0, \tilde{\sigma}_\sigma^2)$, respectively.

TRADI



(Normal DNN)



(Bayesian DNN)

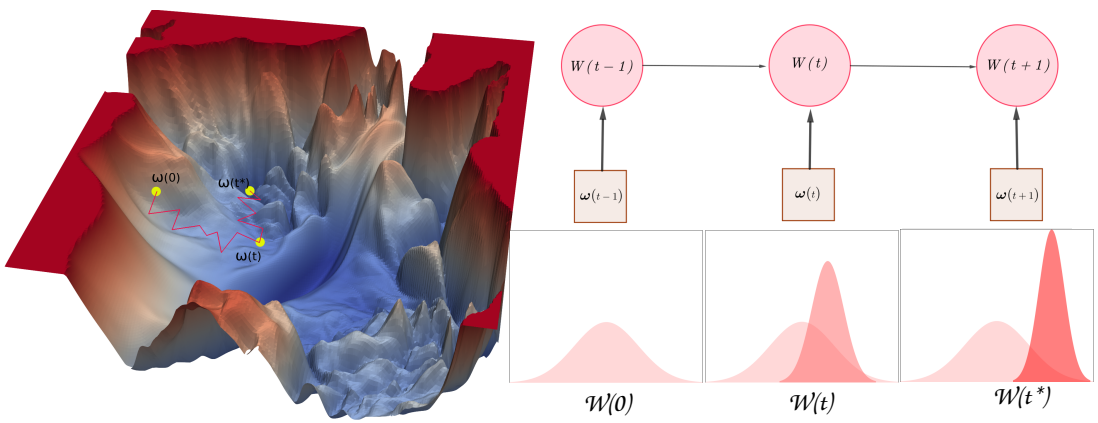
TRADI

We sample new realizations of $W(t*)$ using the following formula:

$$\tilde{\omega}(t^*) = \mu(t^*) + \Sigma^{1/2}(t^*) \times \boldsymbol{m}_1 \text{ with } \Sigma \text{ the covariance matrix.} \quad (14)$$

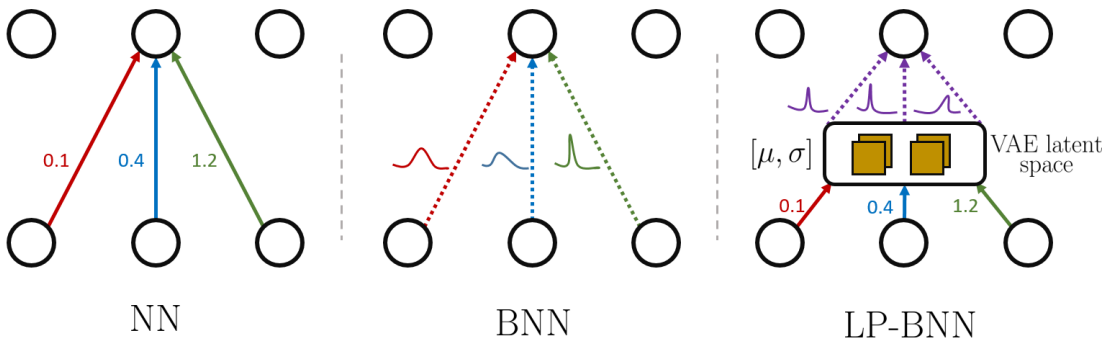
$\boldsymbol{m}_1$ is a realization of the multivariate Gaussian $\mathcal{N}(0_K, \boldsymbol{I}_K)$. Then we take the expectation over this distribution :

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\tilde{\omega}^j(t^*), x^*) \quad (15)$$

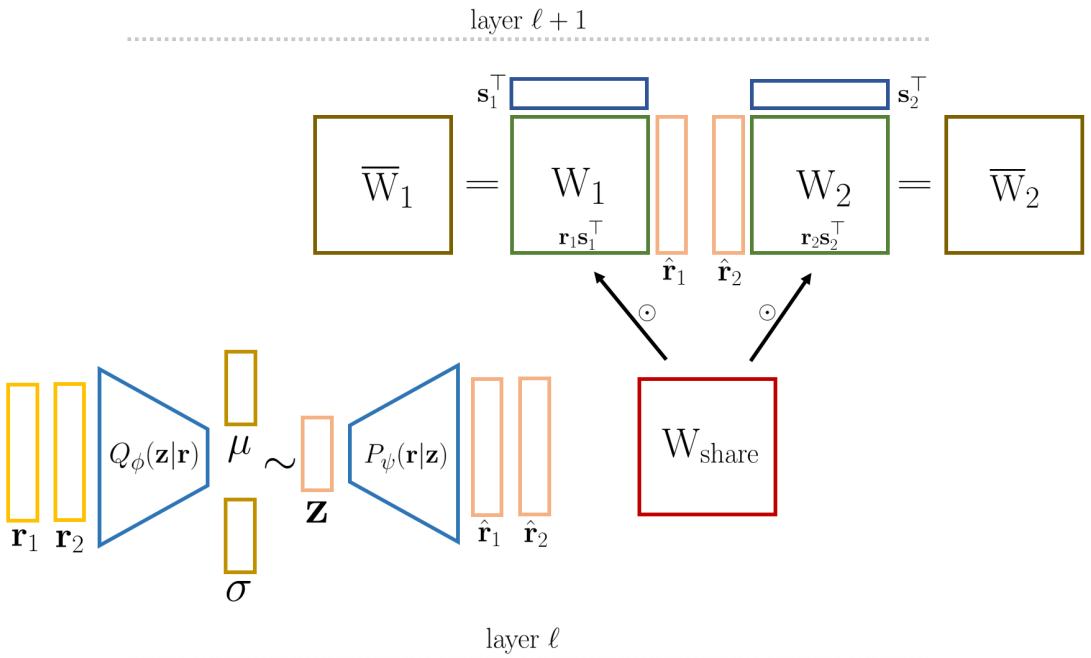


LP-BNN [15]

In classical BNN al



LP-BNN [15]


layer $\ell+1$
$\mathbf{s}_1^\top$
$\mathbf{s}_2^\top$
$\overline{\mathrm{W}}_1$
W_1
$\mathbf{r}_1\mathbf{s}_1^\top$
W_2
$\mathbf{r}_2\mathbf{s}_2^\top$
$\overline{\mathrm{W}}_2$
$\hat{\mathbf{r}}_1$
$\hat{\mathbf{r}}_2$
$\odot$
$\odot$
$\mathrm{W}_{\text{share}}$
$\mathbf{r}_1$ $\mathbf{r}_2$
$Q_\phi(\mathbf{z}|\mathbf{r})$
μ
$\sim$
$\mathbf{z}$
σ
$P_\psi(\mathbf{r}|\mathbf{z})$
$\hat{\mathbf{r}}_1$ $\hat{\mathbf{r}}_2$
layer ℓ

Regression

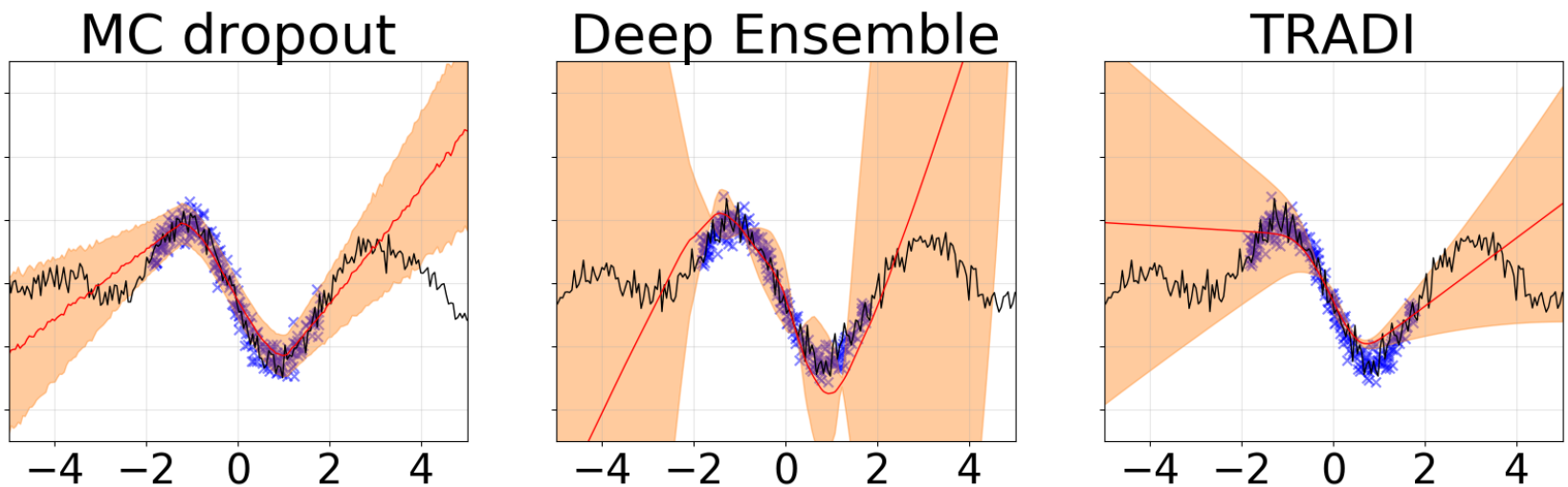


Figure: Results on a synthetic regression task with MC dropout, Deep Ensembles and TRADI algorithm. x-axis: spatial coordinate of the Gaussian process. Black lines: ground truth curve. Orange areas: estimated variance. Blue points represents the training points.

Classification

Table: Comparative results on image classification

Method	MNIST		CIFAR-10	
	NLL	ACCU	NLL	ACCU
Deep Ensembles	**0.035**	**98.88**	**0.173**	**95.67**
MC Dropout	0.065	98.19	0.205	95.27
TRADI	0.044	98.63	0.205	95.29

Metrics[1]

First we group predictions into M bins, each of size $1/M$. Let B_m be the set of indices of samples whose prediction confidence falls into the interval $I_m =]m - 1/M, m/M]$.
The accuracy of a set B_m is defined as:

$$\text{acc}(B_m) = 1/|B_m| \sum_{i \in B_m} \delta_{y_i}(\hat{y}_i) \tag{16}$$

The average confidence in B_m is defined as:

$$\text{conf}(B_m) = 1/|B_m| \sum_{i \in B_m} \hat{p}_i \tag{17}$$

where $\hat{p}_i$ is the confidence for sample i.

Metrics [1]

Expected Calibration Error (ECE) measures the difference in expected accuracy and expected confidence. It is defined as:

$$ECE = \sum_{m}^{M} 1/|B_m||\text{acc}(B_m) - \text{conf}(B_m)| \tag{18}$$

Metrics[11]

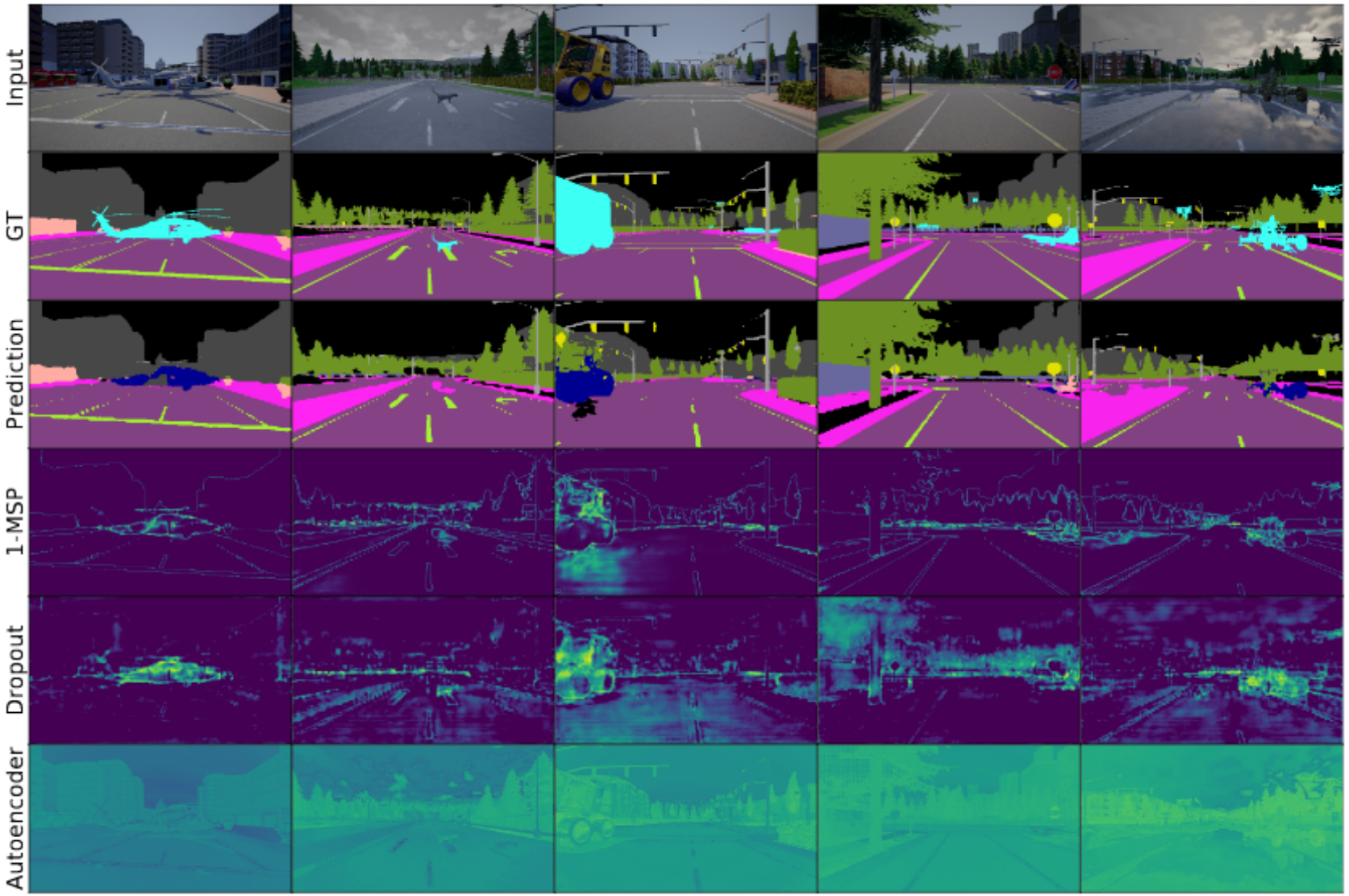
The dataset is divided in two:

- Out of distribution
- in distribution

The confidence score $\hat{p}_i$ for sample i $\hat{p}_i$ is used to detect OOD data. To eveluate the quality we can use :

- Area Under the ROC Curve $\rightarrow$ AUC
- Area Under the Average Precision Curve $\rightarrow$ AUPR
- FPR at 95% TPR can be interpreted as the probability that a negative (out-of-distribution)example is misclassified as positive (in-distribution) when the true positive rate (TPR) is as high as 95%. True positive rate can be computed by TPR = TP / (TP+FN) and , the false positive rate (FPR) can be computed by FPR =FP / (FP+TN).

Results [11]



Out of distribution (Results on the CamVid experiments)

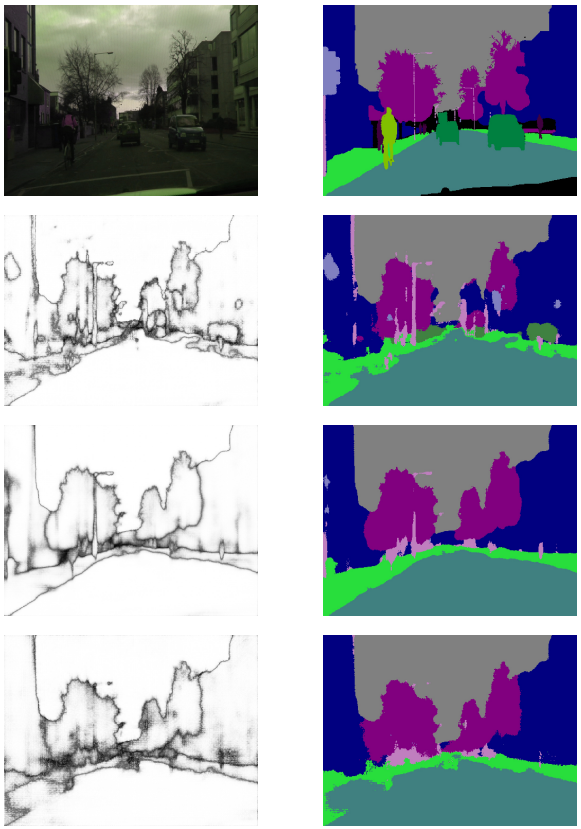
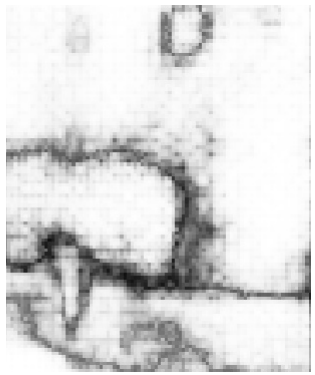


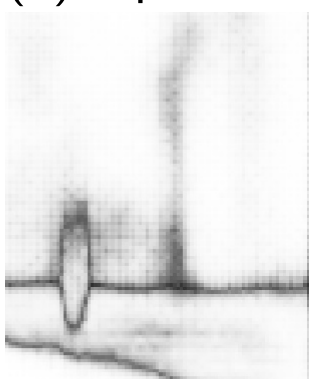
Figure: First row: input image and ground truth, second, third and fourth rows: output and confidence score given by MC dropout, Deep Ensembles and our TRADI, respectively.

Out of distribution

(a) input image



(b) MC dropout confidence

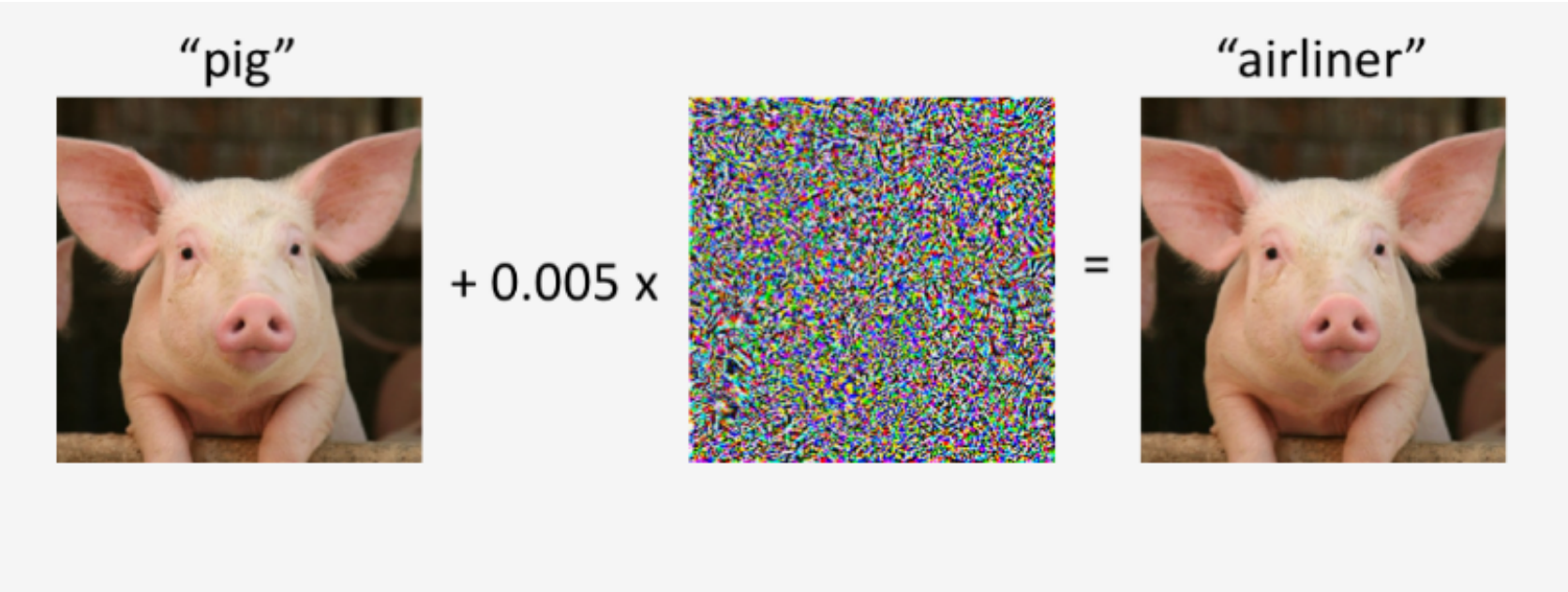


(c) Deep Ensembles confidence

(d) TRADI confidence

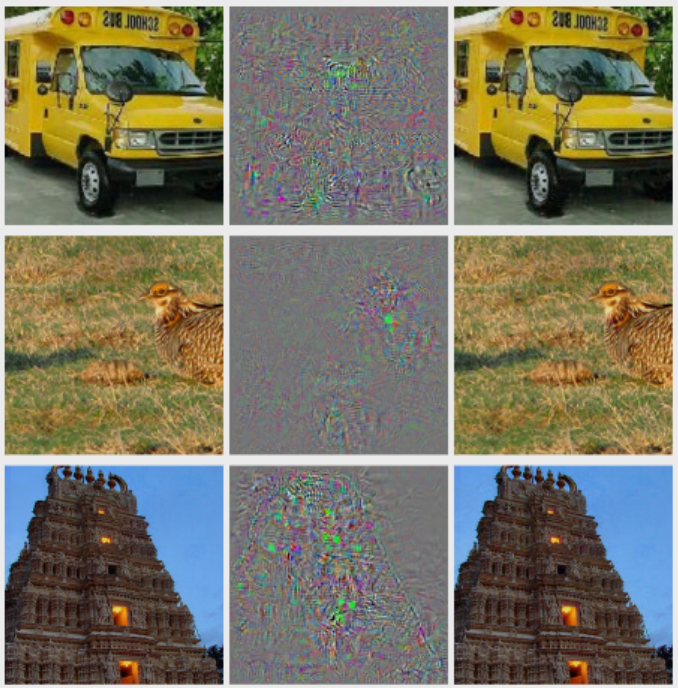
Figure: Zooms of the confidence results on the CamVid experiments. In the bottom left of the input image (a), there is a human, hence a pixel region of an unknown class for all the DNNs, since the pedestrian class was amongst the ones marked as unlabeled. Yet, only the TRADI DNN (d) is consistent.

Adversarial Attacks[9]



[9] Credits: Gille Louppe

Adversarial Attacks[10]



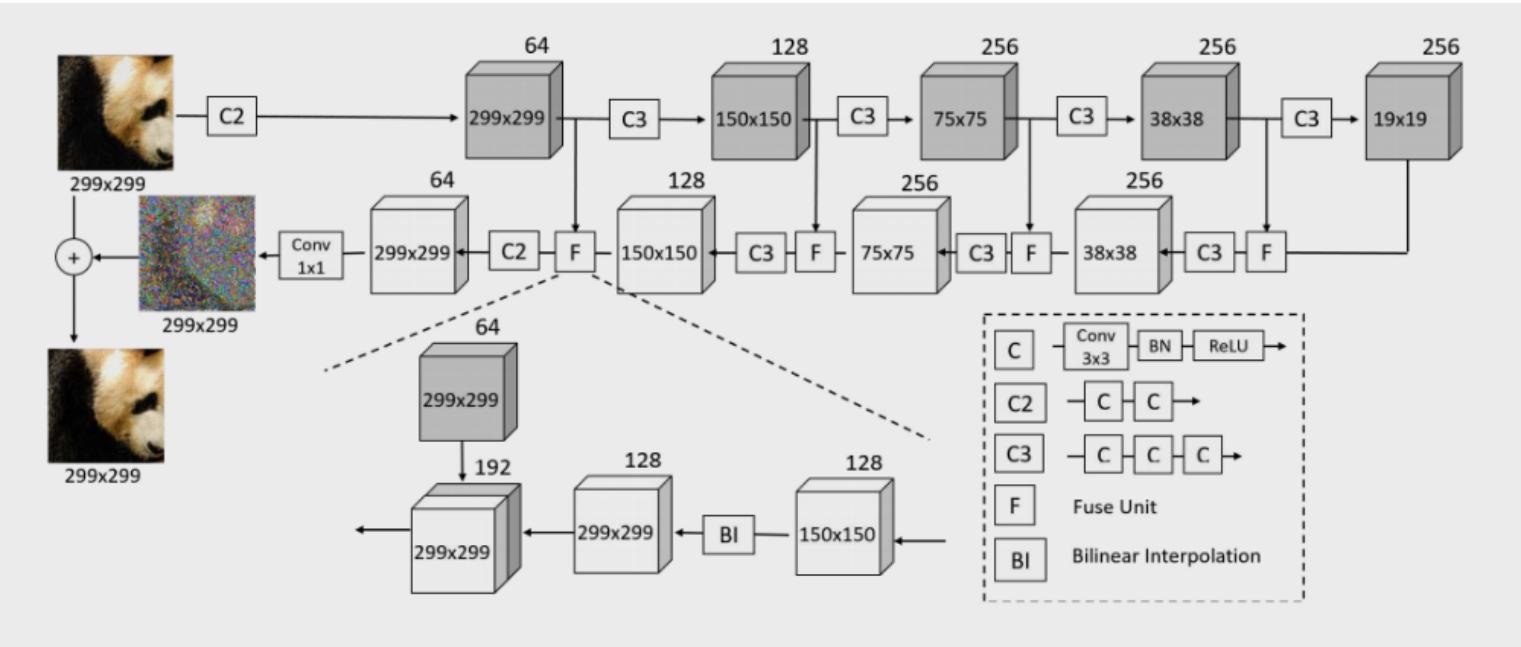
(Left) Original images. (Middle) Adversarial noise. (Right) Modified images. All are classified as 'Ostrich'.

https://www.youtube.com/watch?v=zQ_uMenoBCk

[10] Credits: Gille Louppe

Adversarial Attacks[11]

Train the network to remove adversarial perturbations before using theinput



[11] Credits: Gille Louppe

Adversarial training [5]

Train the network with image x and $\tilde{x}$ a version of x with adversarial perturbations :

$$\tilde{\mathrm{x}} = \mathrm{x} - \epsilon \mathrm{sign}(-\nabla_{\mathrm{x}} \log \mathrm{softmax}_{\hat{y}}(\mathrm{x}, T)) \tag{19}$$

New training loss :

$$\mathcal{L}(g_{\omega}(\tilde{\mathrm{x}}), y) + \mathcal{L}(g_{\omega}(x), y) \tag{20}$$

Bibliography:

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

2 He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

3 Li, Hao, et al. "Visualizing the loss landscape of neural nets." Advances in Neural Information Processing Systems. 2018.

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

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

Bibliography:

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

7 Yoo, Donggeun, and In So Kweon. "Learning loss for active learning." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

8 Corbiere, Charles, et al. "Addressing Failure Prediction by Learning Model Confidence." Advances in Neural Information Processing Systems. 2019.

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

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

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

Bibliography:

12 Bishop, Christopher M. "Mixture density networks." (1994).

13 Liao et al, Defense against Adversarial Attacks Using High-Level Representation Guided Denoiser, 2017

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

15 Franchi, G., Bursuc, A., Aldea, E., Dubuisson, S., & Bloch, I. (2020). Encoding the latent posterior of Bayesian Neural Networks for uncertainty quantification. arXiv preprint arXiv:2012.02818.