

# Semantic Segmentation Deep learning

Gianni FRANCHI  
ENSTA-Paris



"Lately it seems like nothing but zeroes."



# Semantic segmentation

Most images come from [web1] [web2].

For a better understanding of the field please read : <http://d2l.ai/>

# What is Semantic segmentation ?



## Input

segmented

- 1: Person
  - 2: Purse
  - 3: Plants/Grass
  - 4: Sidewalk
  - 5: Building/Structure

3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	3	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	3	5	5	5	5	5
3	3	3	3	3	1	1	3	3	3	3	3	3	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	5	5	5	5
4	4	4	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

## Semantic Labels

# Why do we do semantic segmentation ?



To have a better understanding of the scene.

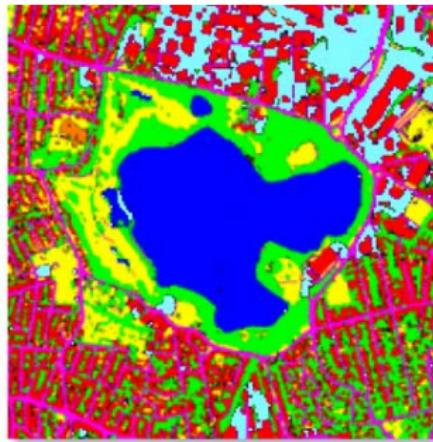
# Where do we apply semantic segmentation?

## Autonomous driving



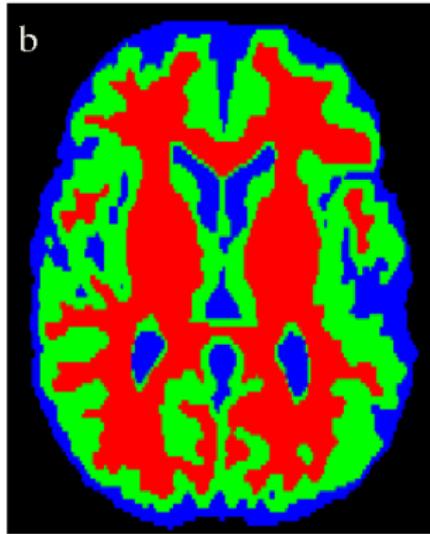
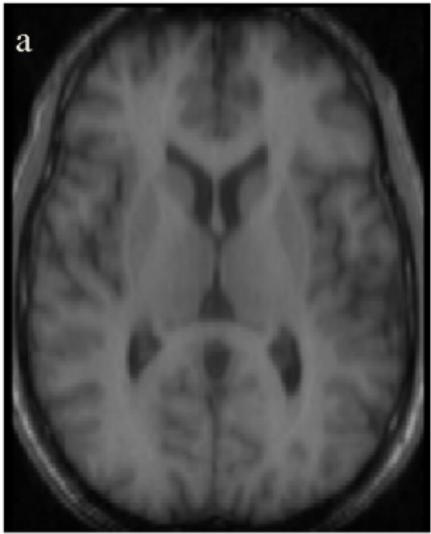
# Where do we apply semantic segmentation?

Remote sensing images



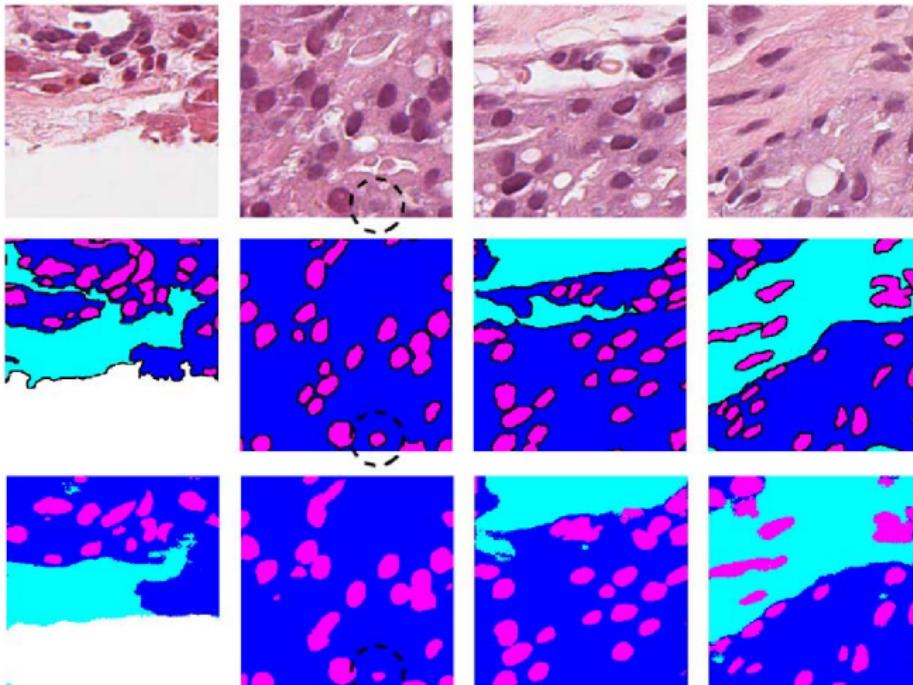
# Where do we apply semantic segmentation?

Medical images [Withey2008]



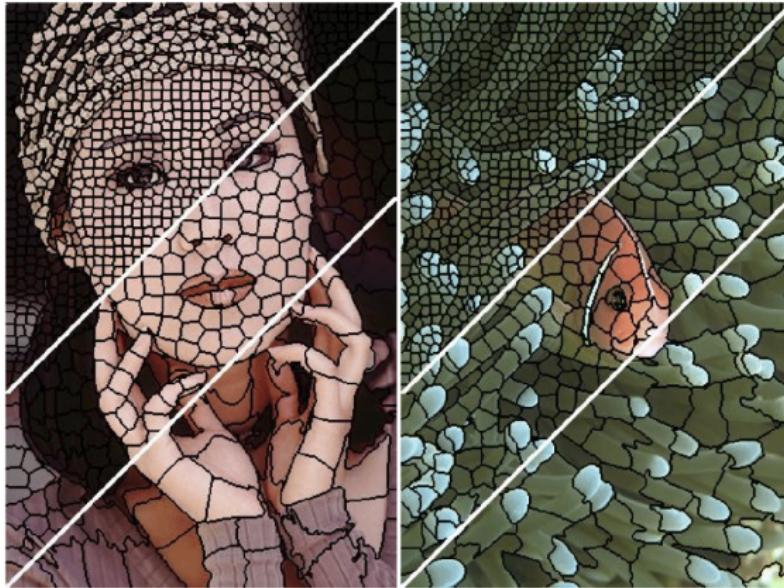
# Where do we apply semantic segmentation?

microscopic images [Isaksson IJCNN2017]



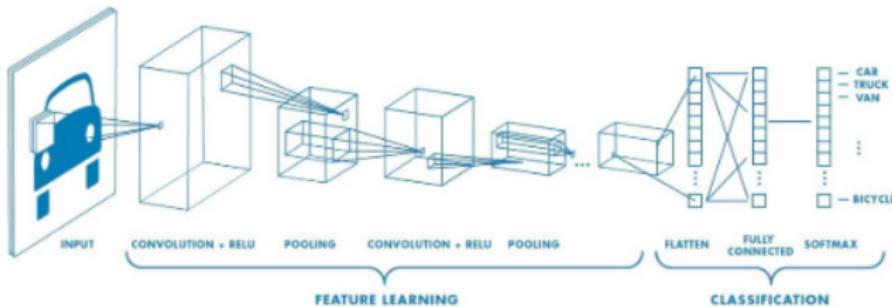
# Traditional semantic segmentation

- Pixel based
- Superpixel based



# How to do Semantic segmentation with DL?

Remember how a Deep Neural network works?

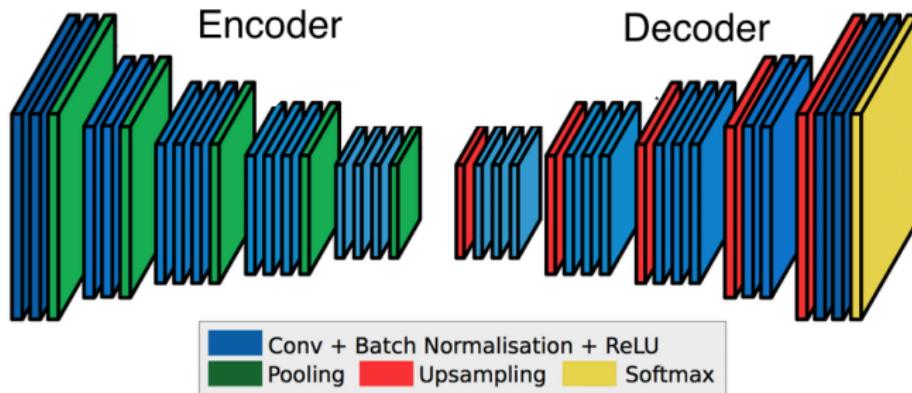


# What are the problems?

- the classical DNN just encode the image
- we loose all the spatial information
- we must be able to handle change on the spatial size of images

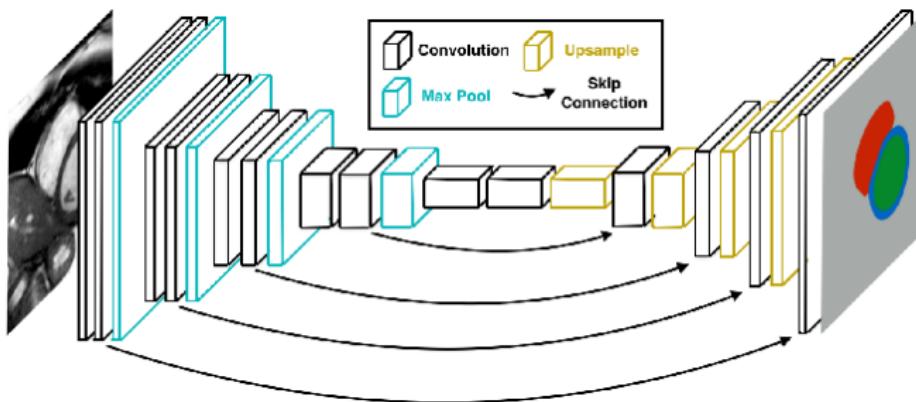
# How to do Semantic segmentation with DL?

Solution : An auto encoder composed of a **Fully Convolutional Network (FCN)**



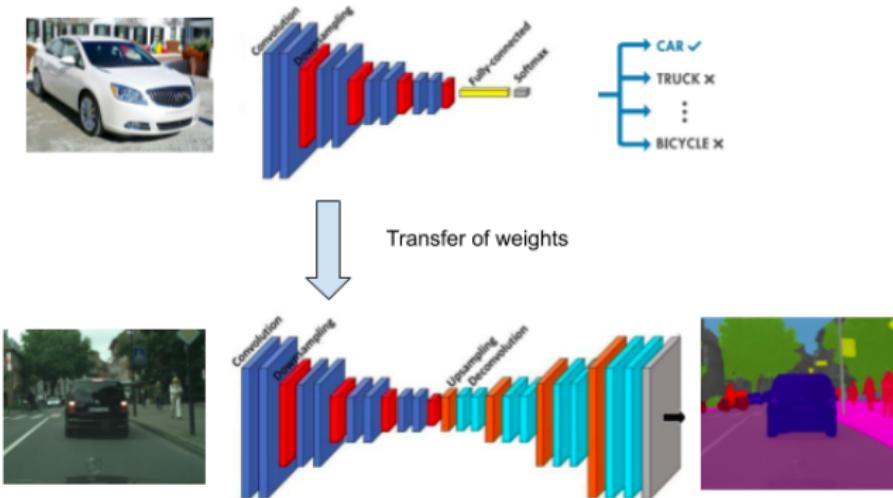
# How to do Semantic segmentation with DL?

We might loose spatial and low-level information. A solution use Skip connection



# How to do Semantic segmentation with DL?

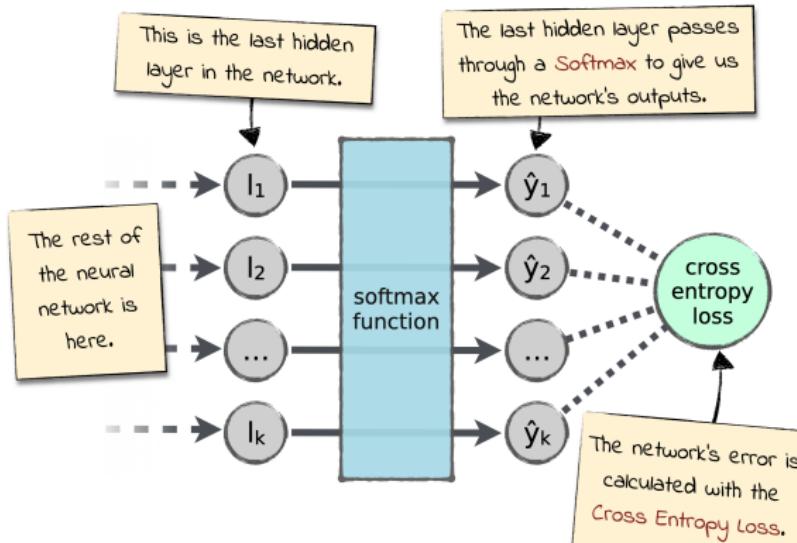
Training a DNN from scratch might be problematic  
A solution use **pre-trained** DNNs



# Which loss do we choose?

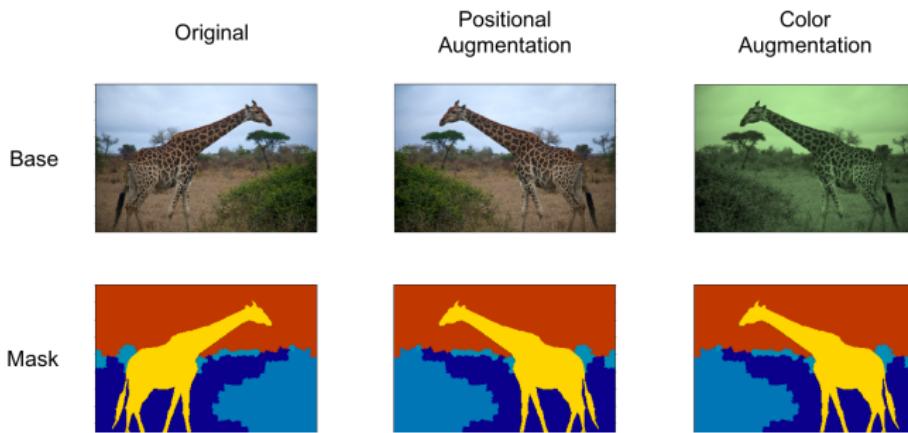
A solution use the **cross entropy loss**.

$$\mathcal{L}(\hat{y}, y) = - \sum_i^C y_i \log(\hat{y}_i) \quad (1)$$



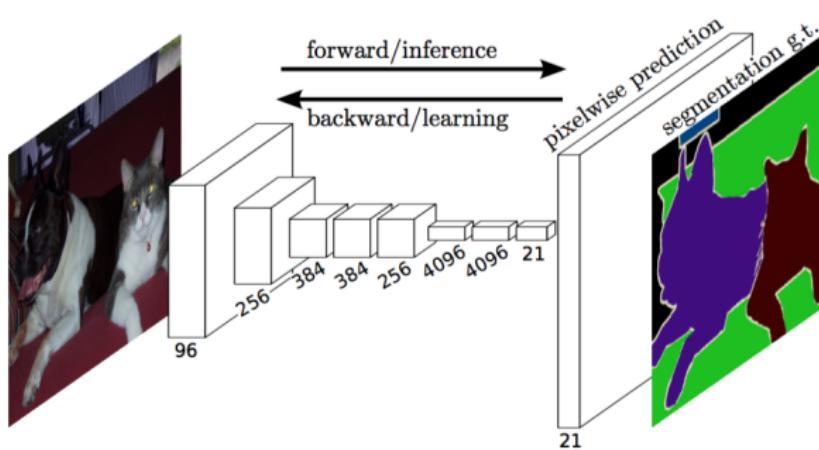
# How to train Semantic segmentation with DL?

Training a DNN from scratch might be problematic  
A solution use **data augmentations**.



# FCN

FCN is the first proposed models for end-to-end semantic segmentation, at the end a big transposed convolution was used.



# FCN + CRF

FCN is followed by a CRF to refine the labels.

3

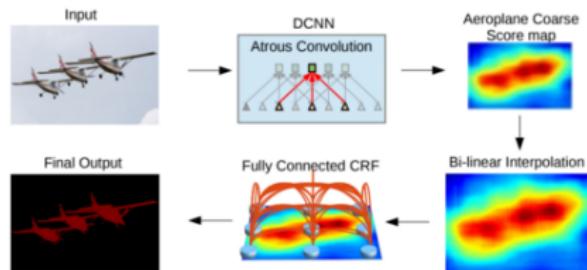
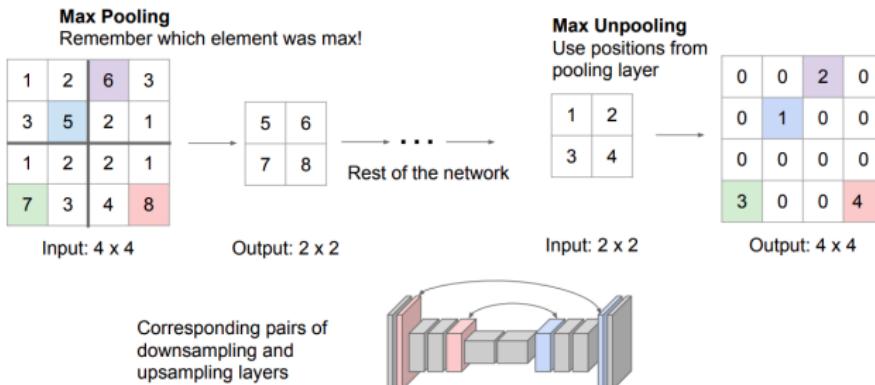


Fig. 1: Model Illustration. A Deep Convolutional Neural Network such as VGG-16 or ResNet-101 is employed in a fully convolutional fashion, using atrous convolution to reduce the degree of signal downampling (from 32x down 8x). A bilinear interpolation stage enlarges the feature maps to the original image resolution. A fully connected CRF is then applied to refine the segmentation result and better capture the object boundaries.

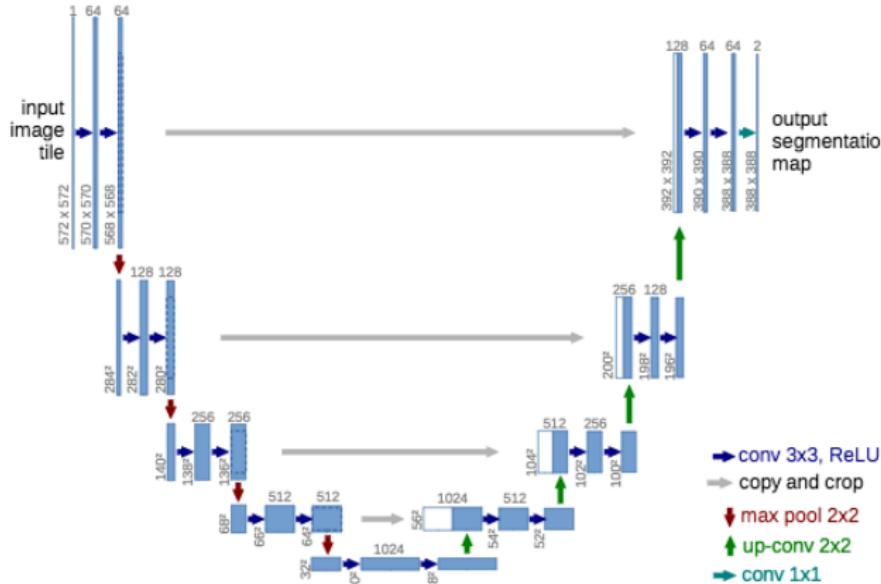
# SegNet

For SegNet, the encoder and decoder layers are symmetrical to each other without any skip connections. The upsampling operation of the decoder layers use the max-pooling indices of the corresponding encoder layers. Unlike FCN, no learnable parameters are used for upsampling.



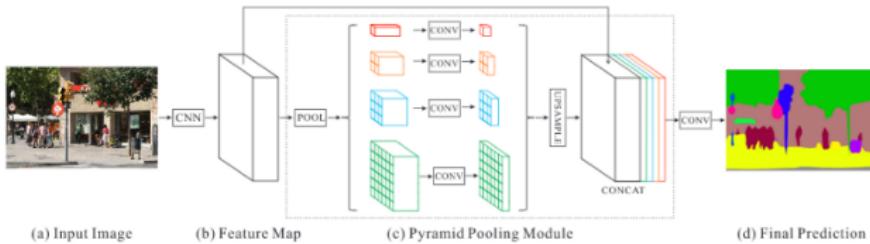
# U-Net

For U-Net, the encoder and decoder layers are symmetrical to each other **with skip connections**. The U-Net uses interpolation followed by convolution operation on the decoder.

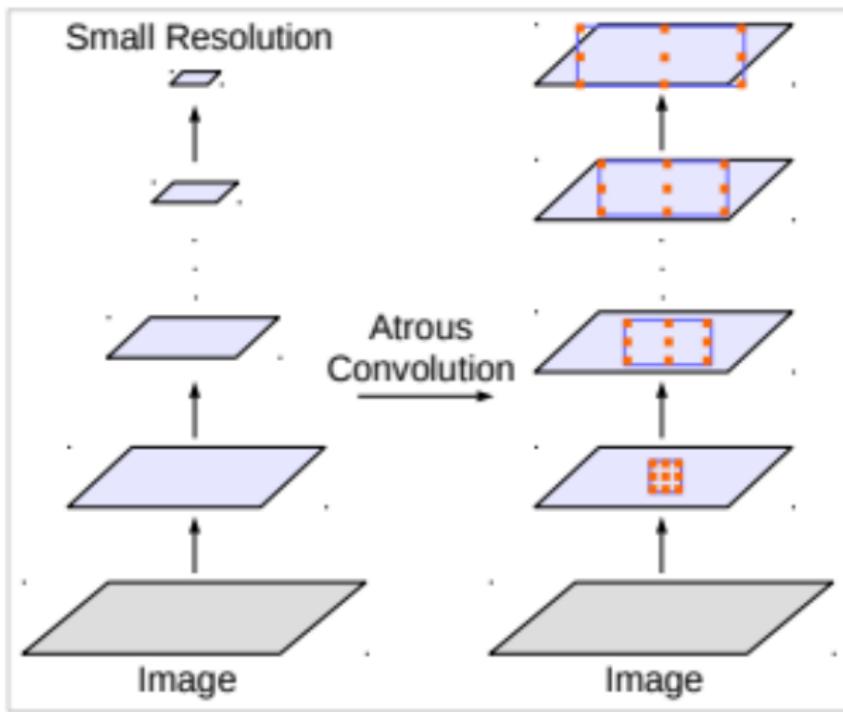


# PSPNet

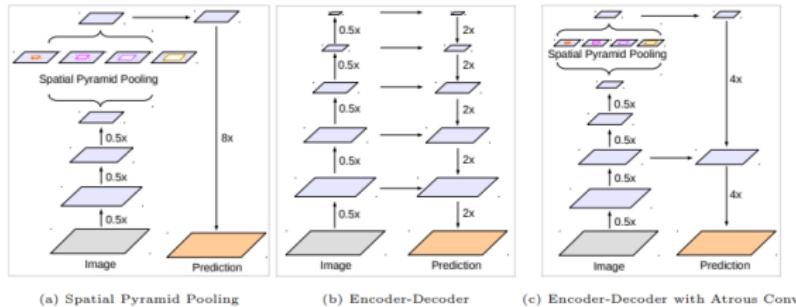
For PSPNet, first use encoder CNN to get the feature map of the last convolutional layer. Then, it uses a pyramid parsing module to improve the representation. This is followed by upsampling and concatenation layers to form the final feature representation.



# Deeplab



# Deeplab v3+



**Fig. 1.** We improve DeepLabv3, which employs the spatial pyramid pooling module (a), with the encoder-decoder structure (b). The proposed model, DeepLabv3+, contains rich semantic information from the encoder module, while the detailed object boundaries are recovered by the simple yet effective decoder module. The encoder module allows us to extract features at an arbitrary resolution by applying atrous convolution.

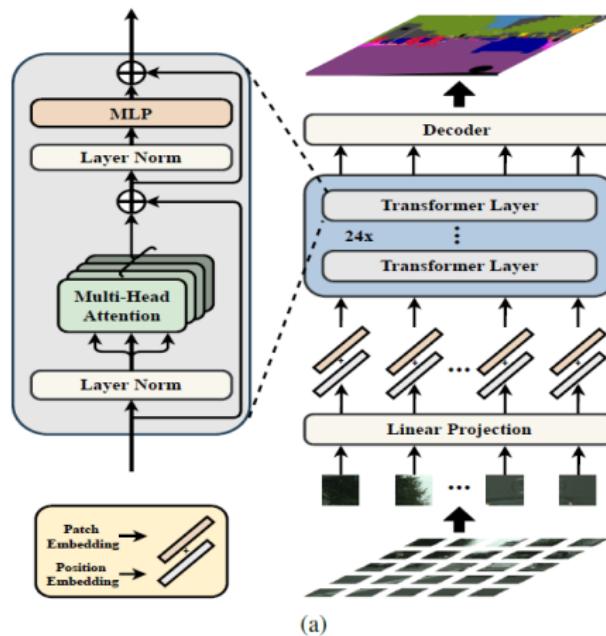
# SEgmentation TTransformers (SETR) [Zheng CVPR2021]

A standard Semantic segmentation model has an **encoder-decoder** architecture:

- **encoder**: for feature representation learning
- **decoder**: for pixel-level classification of the feature representations yielded by the encoder

They propose to use for the encoder a DNN inspired by VIT. For the decoder they propose 3 techniques.

# SEgmentation TTransfomers (SETR) [Zheng CVPR2021]



# SEgmentation TRansformers (SETR) ENCODER [ZhengCVPR2021]

typically the encoder is designed for semantic segmentation would downsample a 2D image  $x \in \mathbb{R}^{H \times W \times 3}$  into a feature map

$x - f \in \mathbb{R}^{\frac{H}{16} \times \frac{W}{16} \times C}$  with  $C$  the number of channels. They decided to set the transformer input sequence length  $L$  to  $\frac{H}{16} \times \frac{W}{16}$

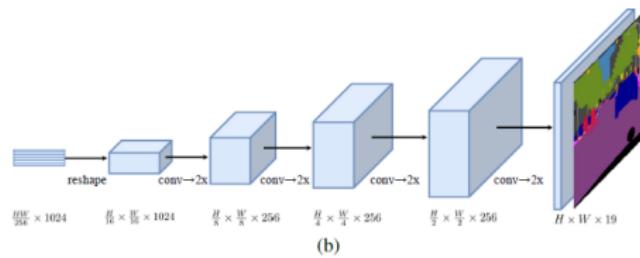
# SEgmentation TRansformers (SETR) DECODER

## [ZhengCVPR2021]

The decoder is a function  $(\frac{H}{16} \times \frac{W}{16} \times C) \rightarrow (H \times W \times K)$ , with  $K$  the number of classes. They propose 3 kind of decoder:

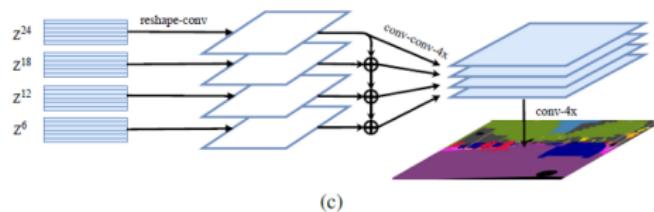
- Naive upsampling (Naive) :  $1 \times 1$  conv + sync batch norm (w/ ReLU) +  $1 \times 1$  conv + bilinear upsample
- Progressive UPsampling (PUP) : Not just on big upsample but a progressive upsampling strategy that alternates conv layers and upsampling operations.
- Multi-Level feature Aggregation (MLA) : a progressive upsampling strategy + multi-level feature aggregation.

# SEgmentation TRansformers (SETR) PUP DECODER [ZhengCVPR2021]



(b)

# SEgmentation TRansformers (SETR) MLA DECODER [ZhengCVPR2021]



(c)

# SEgmentation TRansformers (SETR) [Zheng CVPR2021]

Method	Pre	Backbone	#Params	40k	80k
FCN [38]	1K	R-101	68.59	73.93	75.52
Semantic FPN [38]	1K	R-101	47.51	-	75.80
<i>Hybrid-Base</i>	R	T-Base	112.59	74.48	77.36
<i>Hybrid-Base</i>	21K	T-Base	112.59	76.76	76.57
<i>Hybrid-DeiT</i>	21K	T-Base	112.59	77.42	78.28
SETR- <i>Naïve</i>	21K	T-Large	305.67	77.37	77.90
SETR- <i>MLA</i>	21K	T-Large	310.57	76.65	77.24
SETR- <i>PUP</i>	21K	T-Large	318.31	78.39	79.34
SETR- <i>PUP</i>	R	T-Large	318.31	42.27	-
SETR- <i>Naïve-Base</i>	21K	T-Base	87.69	75.54	76.25
SETR- <i>MLA-Base</i>	21K	T-Base	92.59	75.60	76.87
SETR- <i>PUP-Base</i>	21K	T-Base	97.64	76.71	78.02
SETR- <i>Naïve-DeiT</i>	1K	T-Base	87.69	77.85	78.66
SETR- <i>MLA-DeiT</i>	1K	T-Base	92.59	78.04	78.98
SETR- <i>PUP-DeiT</i>	1K	T-Base	97.64	<b>78.79</b>	<b>79.45</b>

Table 2. Comparing SETR variants on different pre-training strategies and backbones. All experiments are trained on Cityscapes train fine set with batch size 8, and evaluated using the single scale test protocol on the Cityscapes validation set in mean IoU (%) rate. “Pre” denotes the pre-training of transformer part. “R” means the transformer part is randomly initialized.

# Bibliography

-  [web1] <https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc>
-  [web2] <https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc>
-  [Withey2008] Withey, Daniel J., and Zoltan J. Koles. "A review of medical image segmentation: methods and available software." *International Journal of Bioelectromagnetism* 10.3 (2008): 125-148.
-  [IsakssonIJCNN2017] Isaksson, Johan, et al. "Semantic segmentation of microscopic images of H&E stained prostatic tissue using CNN." *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017.

# Bibliography



[ZhengCVPR2021] Zheng, S., Lu, J., Zhao, H., Zhu, X., Luo, Z., Wang, Y., ... & Zhang, L. (2021). Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 6881-6890).

# Uncertainty Quantification in Deep Learning

Gianni FRANCHI

*U2IS, ENSTA Paris*

Presentation 2023

# Plan

- 1 Context
- 2 Uncertainty and Deep learning
- 3 Aleatoric loss
- 4 Mixture density network
- 5 Calibration of Deep Neural Network
- 6 Learning loss
- 7 Bayesian Deep Neural Network and ensembling
- 8 Experiments
- 9 Adversarial Attacks
- 10 Bibliography

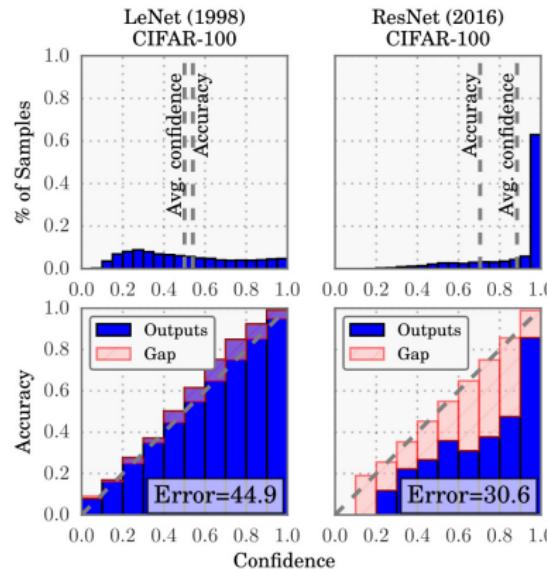
# What is uncertainty in machine/deep learning

- 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]

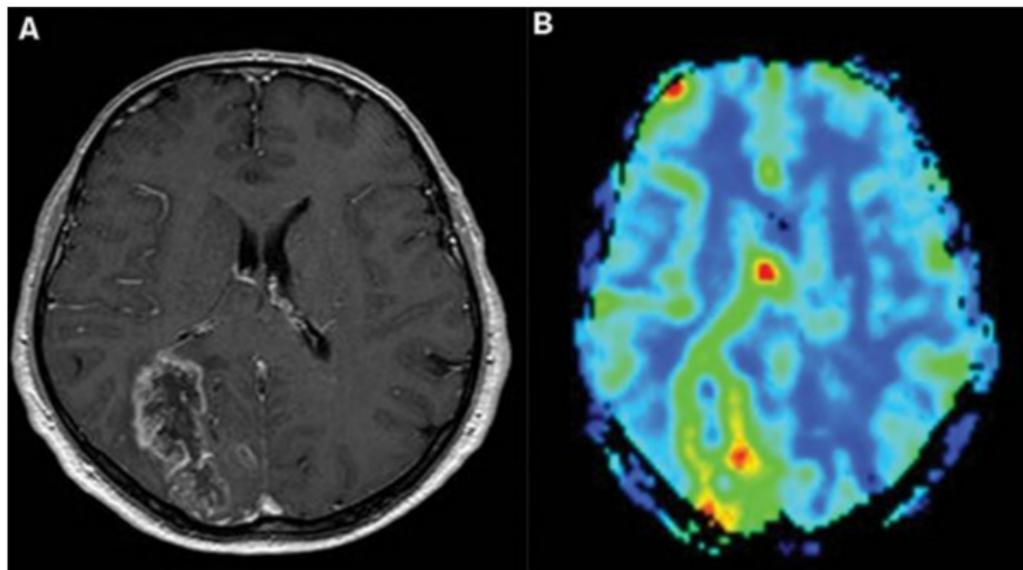
# Why Uncertainty is important?

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:

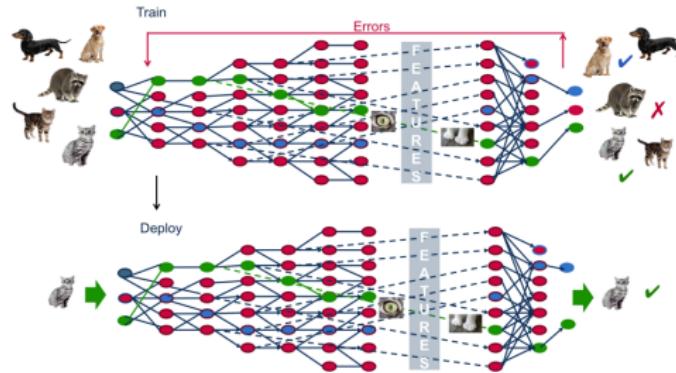


## Why Uncertainty is important?

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 innovations, software innovations, and larger data sets.



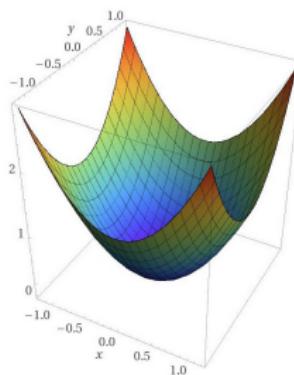
# Deep Learning notations

- Training/Testing sets are denoted respectively by  $\mathcal{D}_I = (x_i, y_i)_{i=1}^{n_I}$ ,  $\mathcal{D}_{\tau} = (x_i, y_i)_{i=1}^{n_{\tau}}$ . 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  $\mathcal{D}_I$  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  $\omega = \{\omega_k\}_{k=1}^K$ .
- During training,  $\omega$ , is iteratively updated for each mini-batch and we denote by  $\omega(t)$  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  $g_{\omega(t)}(x_i)$  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 can use a gradient descent to optimize  $\omega(t)$



Computed by WolframAlpha

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:

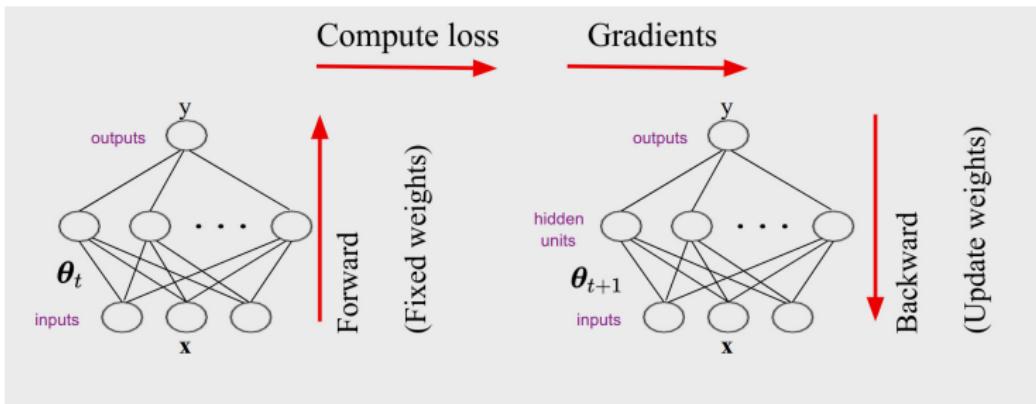
$$\nabla \mathcal{L}_{\omega_k(t)} = \frac{1}{|B(t)|} \sum_{(x_i, y_i) \in B(t)} \frac{\partial \mathcal{L}(\omega(t-1), y_i)}{\partial \omega_k(t-1)} \quad (1)$$

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

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

with  $\eta$  the learning rate.

# Deep Learning optimization



# Deep Learning optimization

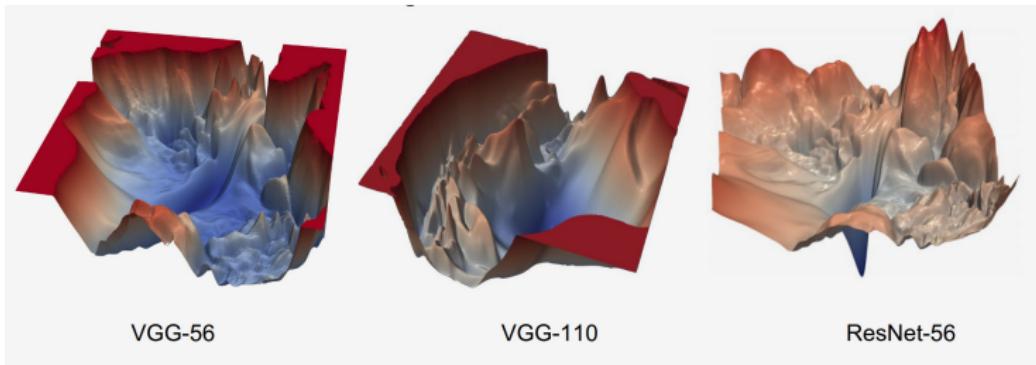
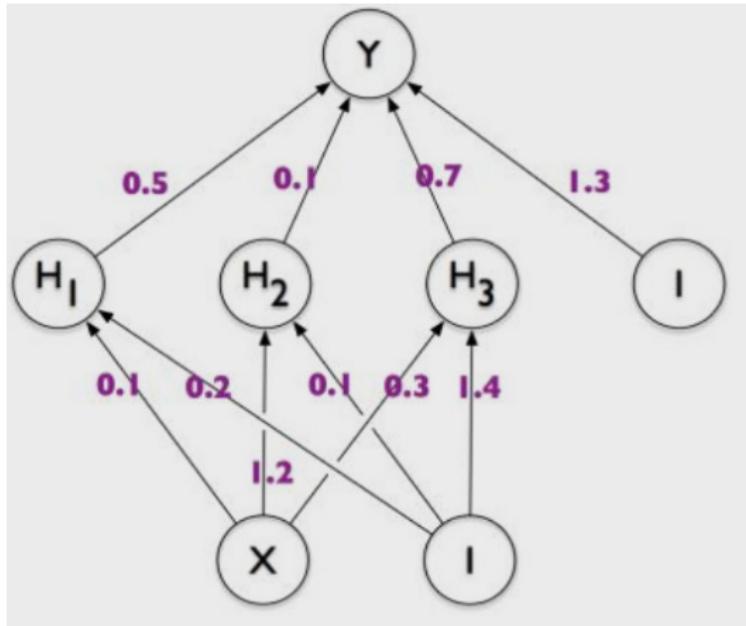


Figure: Visualizing the loss surfaces of modern DNN [3]

# 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<sup>1</sup>

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?

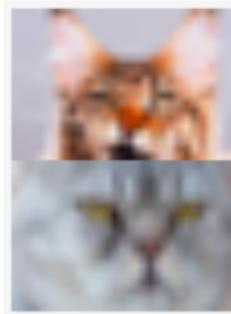


---

<sup>1</sup>Credits: Gille Louppe

## Types of Uncertainty: Case 2<sup>2</sup>

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.



---

<sup>2</sup>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:

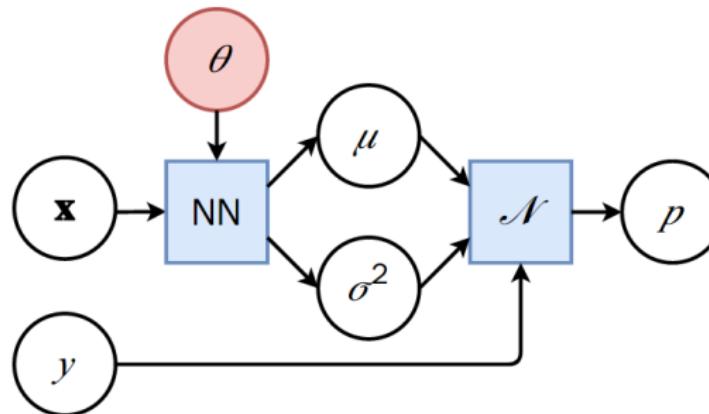
$$\mathcal{P}(Y|X, \omega) = \mathcal{N}(\mu(X, \omega); \sigma^2(\mu(X, \omega))) \quad (3)$$

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

# Heteroscedastic loss[6]<sup>3</sup>

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

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




---

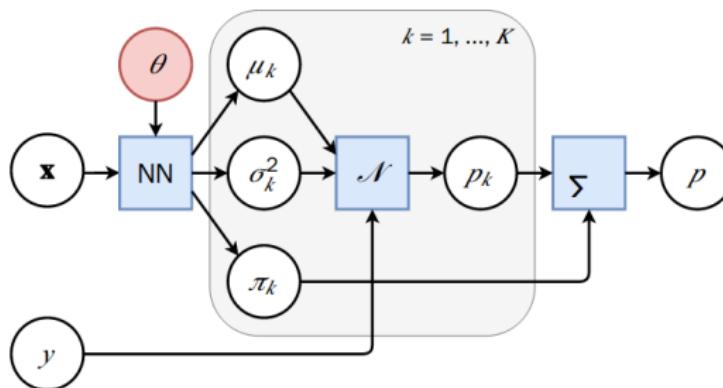
<sup>3</sup>Credits: Gille Louppe

# Mixture density network[12]<sup>4</sup>

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

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

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




---

<sup>4</sup>Credits: Gille Louppe

# Calibration

The classical loss used is the cross entropy:

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

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

## Calibration with temperature scaling

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

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

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

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

# 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  $\omega$  that maximize the likelihood.

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

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

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

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

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

With  $H$  the cross entropy.

# Bayesian approach and DNN

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

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

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

This leads to l2 regularization.

# Bayesian DNN

Bayesian DNN is based on marginalization instead of MAP optimization.

$$\mathcal{P}(Y|X) = \mathbb{E}_{\omega \sim \mathcal{P}(\omega|\mathcal{D}_I)} (\mathcal{P}(Y|X, \omega))$$

$$\mathcal{P}(Y|X) = \int \mathcal{P}(Y|X, \omega) \mathcal{P}(\omega|\mathcal{D}_I) d\omega$$

In practice:

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

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

## Variational inference

Variational inference approximates the posterior  $\mathcal{P}(\omega|\mathcal{D}_I)$  with a family of distributions  $q_\lambda(\omega|\mathcal{D}_I)$ . The variational parameter  $\lambda$  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  $\lambda_{x_i} = (\mu_{x_i}, \sigma_{x_i}^2)$ .

**Question :** How can we know how well our variational posterior  $q_\lambda(w|\mathcal{D}_I)$  approximates the true posterior  $\mathcal{P}(\omega|\mathcal{D}_I)$ ?

## 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  $\mathcal{P}$  :

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

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

## Variational inference

The optimal approximate posterior is thus

$$q_{\lambda}^*(\omega | \mathcal{D}_I) = \arg \min_{\lambda} \text{KL}(q_{\lambda}(\omega | \mathcal{D}_I) || \mathcal{P}(\omega | \mathcal{D}_I)).$$

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

$$\begin{aligned} ELBO(\lambda) &= E_q[\log \mathcal{P}(\omega, \mathcal{D}_I)] - E_q[\log q_{\lambda}(\omega | \mathcal{D}_I)] \\ &= - \int_{\omega} \left( q_{\lambda}(\omega | \mathcal{D}_I) \log \left( \frac{q_{\lambda}(\omega | \mathcal{D}_I)}{\mathcal{P}(\omega) \mathcal{P}(\mathcal{D}_I | \omega)} \right) \right) d\omega \\ &= E_q[\log \mathcal{P}(\mathcal{D}_I | \omega)] - \text{KL}(q_{\lambda}(\omega | \mathcal{D}_I) || \mathcal{P}(\omega)) \end{aligned}$$

Note that  $\text{KL}(q_{\lambda}(\omega | \mathcal{D}_I) || \mathcal{P}(\omega | \mathcal{D}_I)) = \log \mathcal{P}(\mathcal{D}_I) - ELBO(\lambda)$ .

## Variational inference: Reparametrization trick

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

$$\frac{\partial}{\partial \lambda} E_{q_\lambda(\omega | \mathcal{D}_I)} f(\omega, \lambda) = E_{q(\epsilon)} \left[ \frac{\partial f(\omega, \lambda)}{\partial \omega} \frac{\partial \omega}{\partial \lambda} + \frac{\partial f(\omega, \lambda)}{\partial \lambda} \right].$$

## 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

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

6. Calculate the gradient with respect to the standard deviation parameter  $\rho$

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

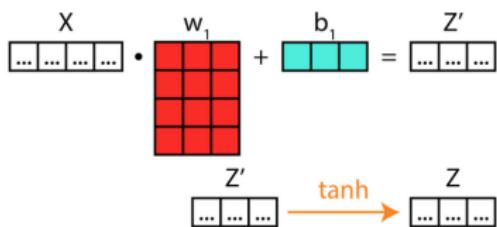
7. Update the variational parameters:

$$\mu \leftarrow \mu - \alpha \Delta_\mu \quad (5)$$

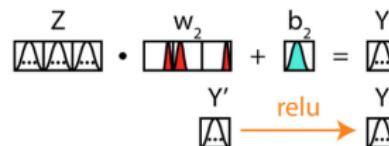
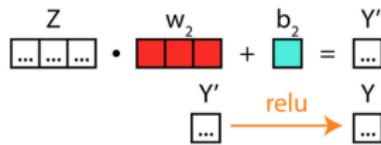
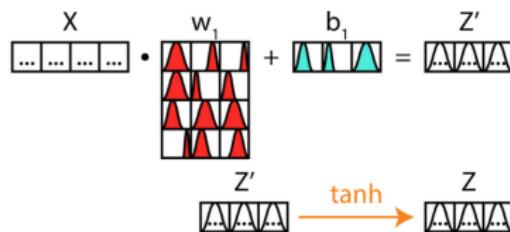
$$\rho \leftarrow \rho - \alpha \Delta_\rho. \quad (6)$$

# Weight Uncertainty in Neural Networks [9]<sup>5</sup>

Standard Neural Network



Bayesian Neural Network



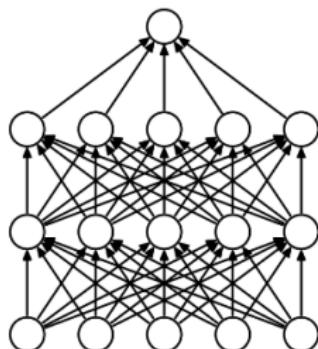
<sup>5</sup>Image credit: Eric Ma

# Dropout<sup>6</sup>

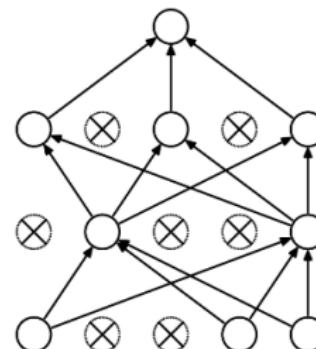
Dropout is an empirical technique that was proposed to avoid overfitting in CNN.

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

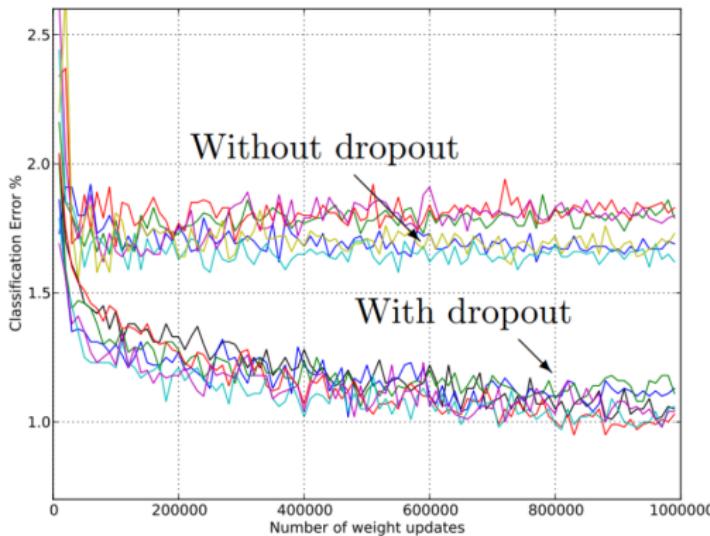


(b) After applying dropout.

---

<sup>6</sup>Image credit: G. Louuppe

# MC dropout<sup>7</sup>



Why does dropout work?

<sup>7</sup>Image credit: G. Louppe

## MC dropout [4]<sup>8</sup>

**Dropout does variational inference.**

Let us split the weights  $\omega$  per layer  $\omega = \{\omega_1, \dots, \omega_L\}$  ( $L$  is the number of Layer). Let us also further split each layer to unit  $\omega_I = \{\omega_{I,1}, \omega_{I,q_1}\}$ . Variational parameters  $\lambda$  are split similarly into  $\lambda = \{M_1, \dots, M_L\}$ , with  $M_I = \{m_{I,1}, m_{I,q_1}\}$ .

Then, the proposed  $q$  is :

$$q_\lambda(\omega / \mathcal{D}_I) = \prod_I^L q(\omega_I, M_I)$$

$$\text{with } q(\omega_I, M_I) = \prod_i^{q_I} q(\omega_{I,i}, m_{I,i})$$

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

---

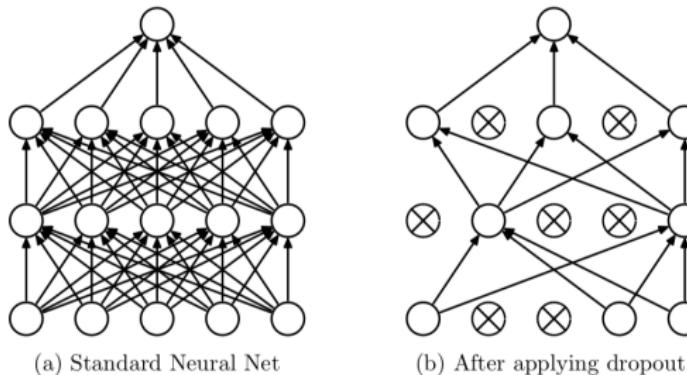
<sup>8</sup>Image credit: G. Louuppe

## MC dropout [4]

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

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

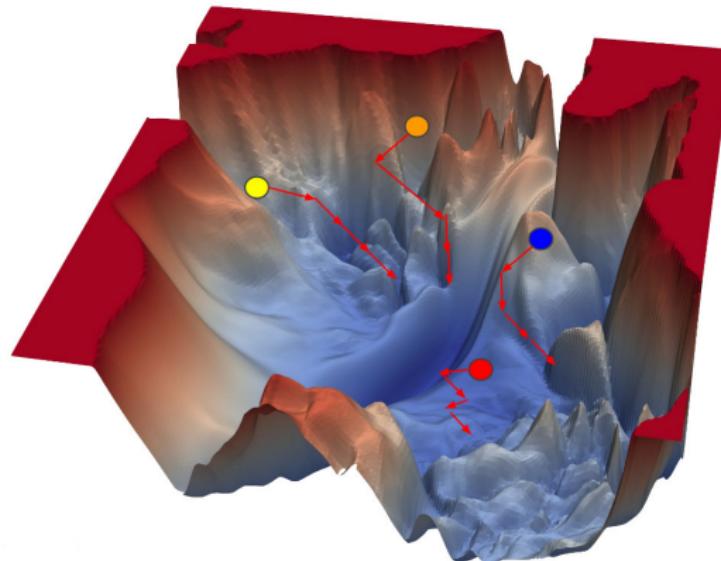
with  $b^j$  a vector of the same size of  $\omega(t^*)$  which is a realization of a binomial distribution.



## Deep Ensembles[5]

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

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



## Deep Ensembles[10]

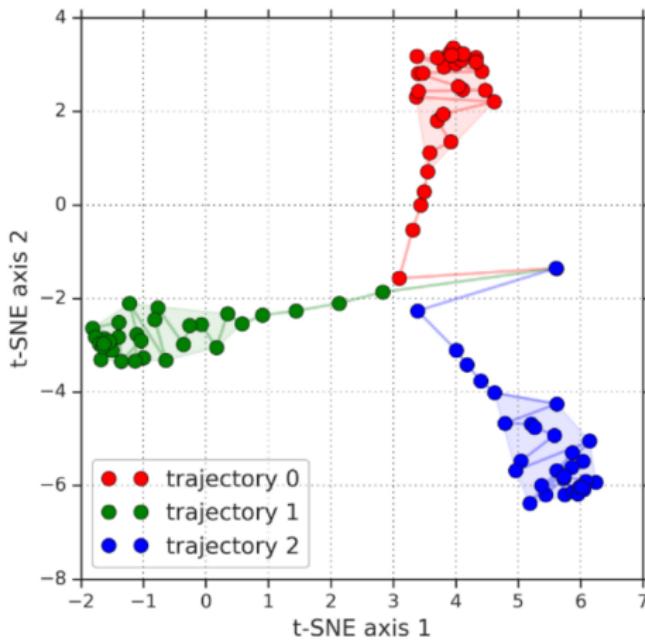
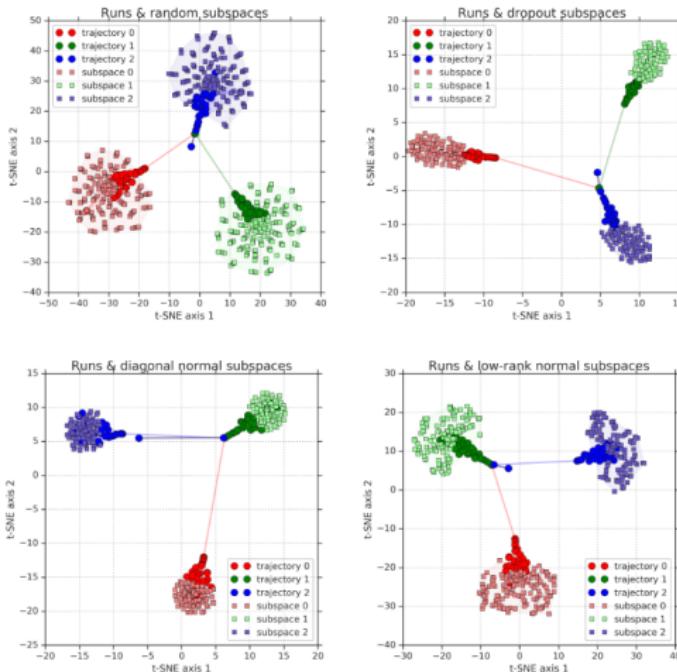


Figure: t-SNE plot of predictions from checkpoints corresponding to 3 different randomly initialized trajectories

# 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]

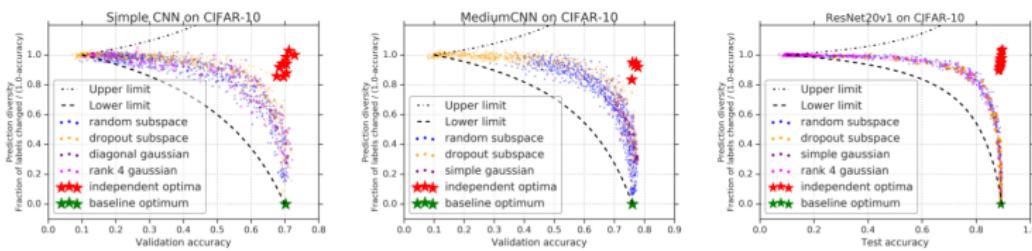
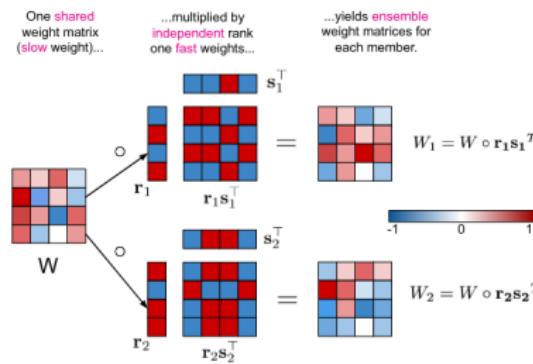


Figure: Diversity versus accuracy plots for 3 models trained on CIFAR-10

## 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 kind of weights. 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  $\omega^{\text{fast}} = \{F_j\}_{j=1}^{N_{\text{model}}}$ . 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  $\phi$  an activation function then when we apply the BatchEnsemble on an image we perform:

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

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

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

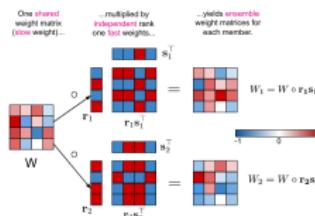


Figure: An illustration on how to generate the ensemble weights for two

# TRADI

- $\omega(0)$  is the initial set of weights  $\{\omega_k(0)\}_{k=1}^K$  following  $\mathcal{N}(0, \sigma_k^2)$ , where  $\sigma_k^2$  are fixed as in [2].
- $\mathcal{L}(\omega(t), y_i)$  is 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.
- Weights on different layers are assumed to be independent of one another at all times.
- Each weight  $\omega_k(t)$ ,  $k = 1, \dots, K$ , 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  $\mu_k(t)$ :

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

with  $\varepsilon_\mu$  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.

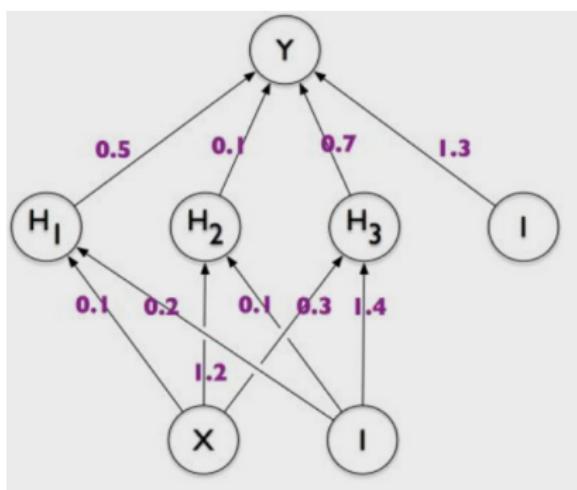
# TRADI

The state and measurement equations for the variance  $\sigma_k$  are given by:

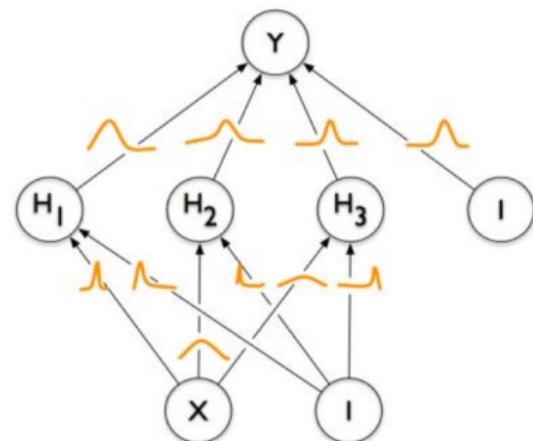
$$\begin{cases} \sigma_k^2(t) = \sigma_k^2(t-1) + (\eta \nabla \mathcal{L}_{\omega_k(t)})^2 - \eta^2 \mu_k(t)^2 + \varepsilon_\sigma \\ z_k(t) = \sigma_k^2(t) - \mu_k(t)^2 + \tilde{\varepsilon}_\sigma \\ \text{with } z_k(t) = \omega_k(t)^2 \end{cases} \quad (13)$$

with  $\varepsilon_\sigma$  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)

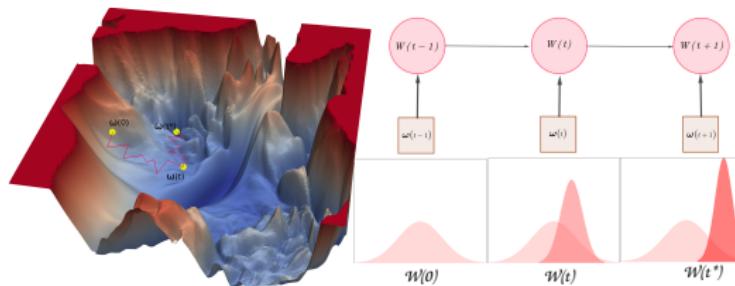
# TRADI

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

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

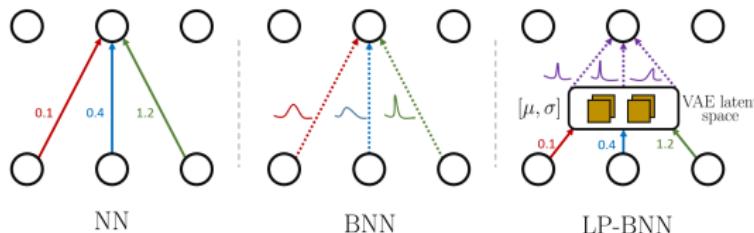
$\mathbf{m}_1$  is a realization of the multivariate Gaussian  $\mathcal{N}(0_K, \mathbf{I}_K)$ . Then we take the expectation over this distribution :

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

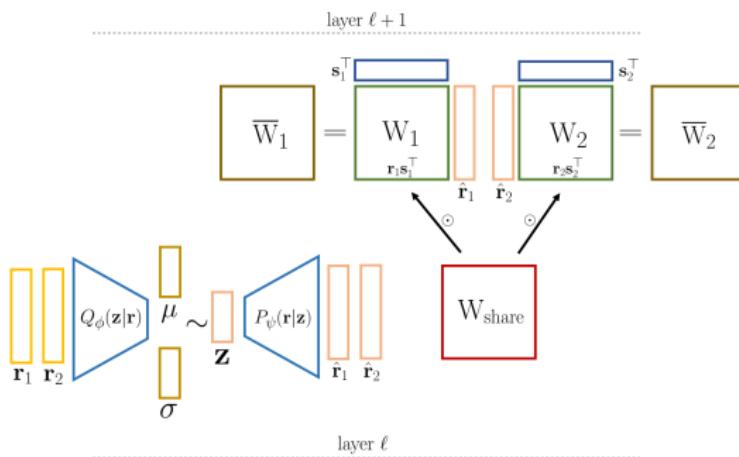


# LP-BNN [15]

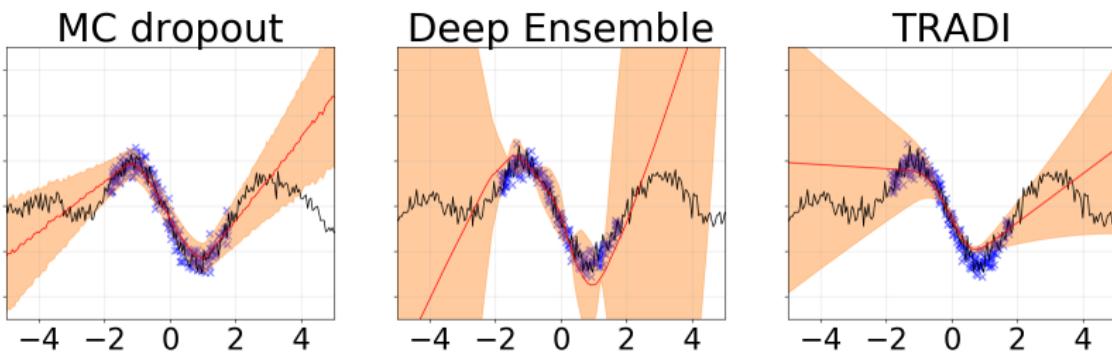
In classical BNN al



# LP-BNN [15]



# 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	<b>0.035</b>	<b>98.88</b>	<b>0.173</b>	<b>95.67</b>
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:

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

The average confidence in  $B_m$  is defined as:

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

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:

$$ECE = \sum_m^M 1/|B_m| |\text{acc}(B_m) - \text{conf}(B_m)| \quad (18)$$

## Metrics[11]

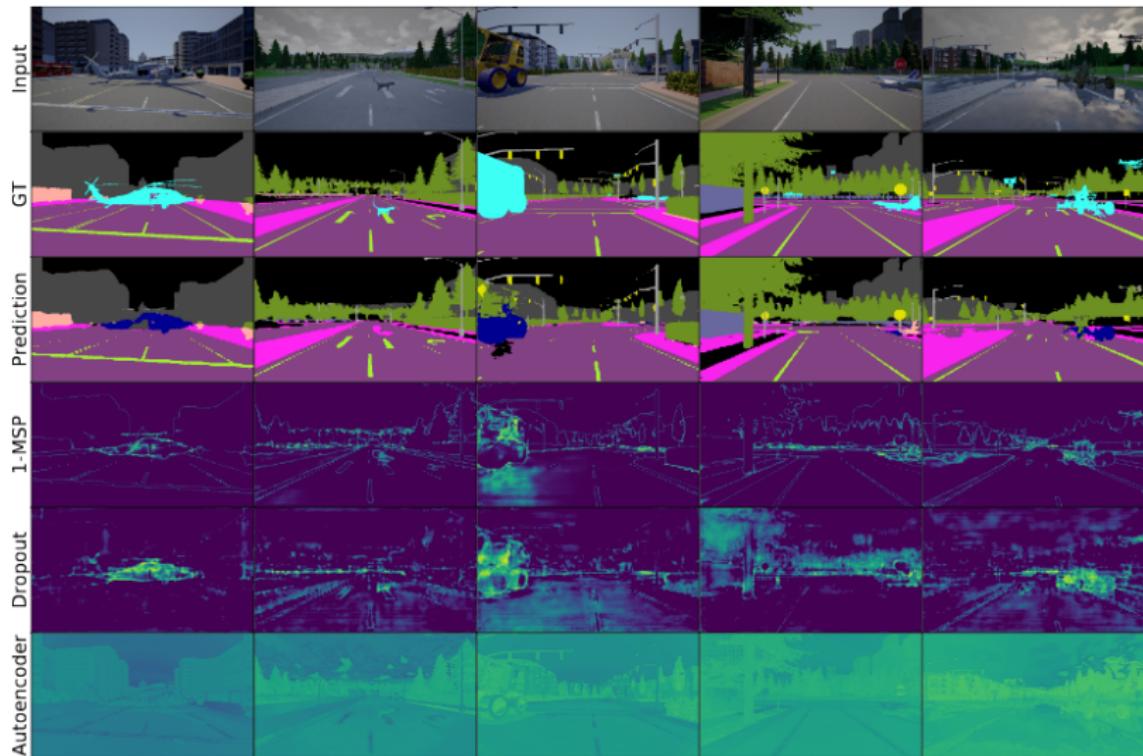
The dataset is divided in two:

- Out of distribution
- in distribution

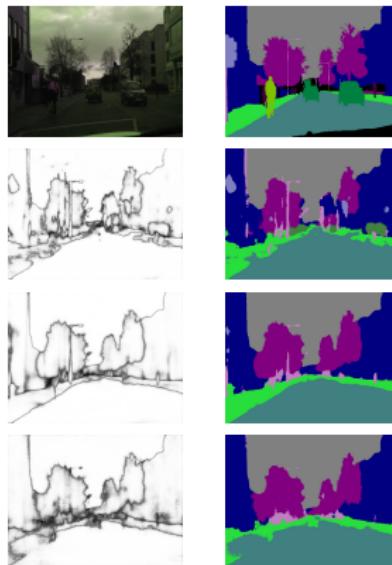
The confidence score  $\hat{p}_i$  for sample  $i$   $\hat{p}_i$  is used to detect OOD data. To evaluate the quality we can use :

- Area Under the ROC Curve → AUC
- Area Under the Average Precision Curve → 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  $\text{TPR} = \text{TP} / (\text{TP} + \text{FN})$  and , the false positive rate (FPR) can be computed by  $\text{FPR} = \text{FP} / (\text{FP} + \text{TN})$ .

# 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



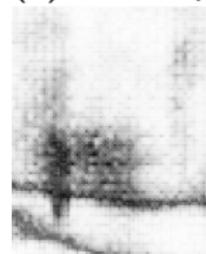
(a) input image



(b) MC dropout confidence



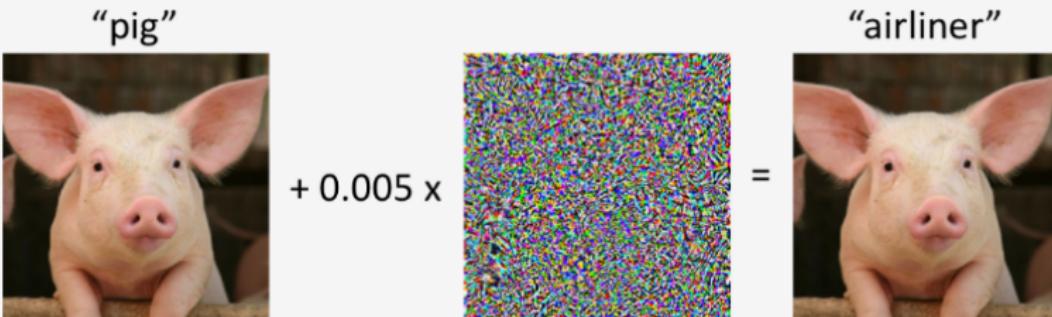
(c) Deep Ensembles confidence



(d) TRADI confidence

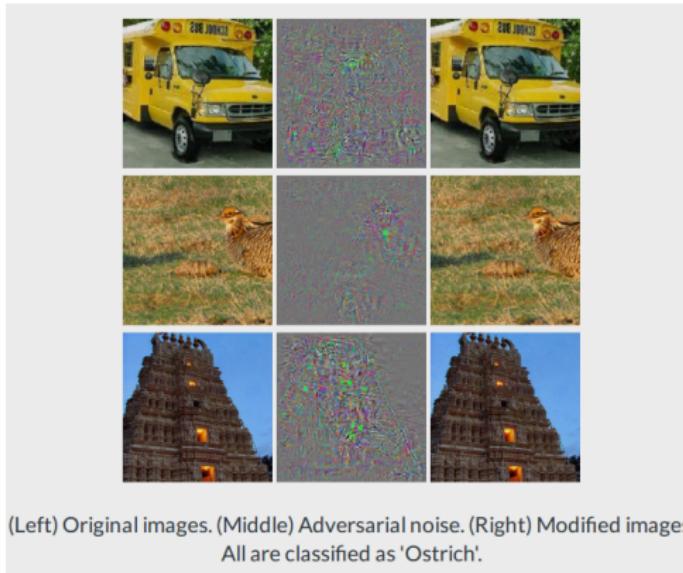
**Figure:** Zooms of the confidence results on the CamVid experiments. In the bottom left of the input image (a), there is a human, hence a pixel region of an unknown class for all the DNNs, since the pedestrian class was amongst the ones marked as unlabeled. Yet, only the TRADI DNN (d) is consistent.

# Adversarial Attacks<sup>9</sup>



<sup>9</sup>Credits: Gille Louppe

# Adversarial Attacks<sup>10</sup>

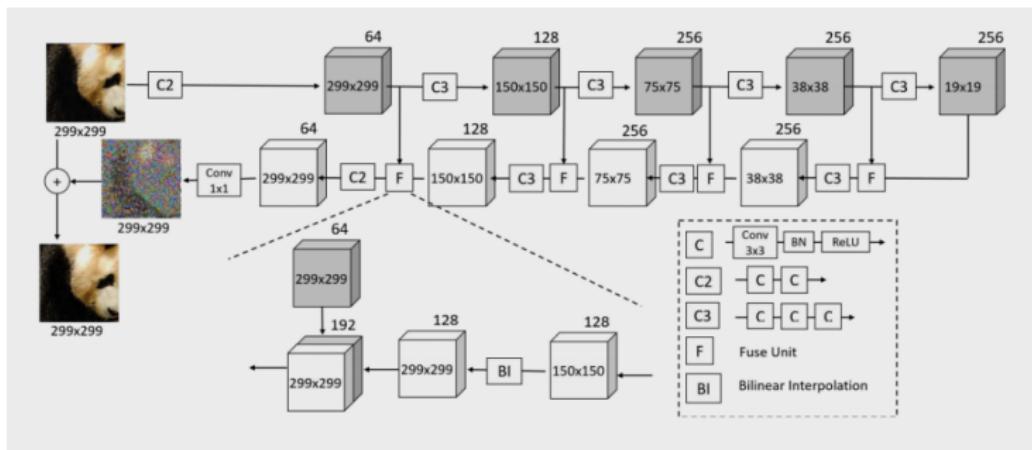


[https://www.youtube.com/watch?v=zQ\\_uMenoBCk](https://www.youtube.com/watch?v=zQ_uMenoBCk)

<sup>10</sup>Credits: Gille Louppe

# Adversarial Attacks<sup>11</sup>

Train the network to remove adversarial perturbations before using the input



<sup>11</sup>Credits: Gille Louppe

## Adversarial training [5]

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

$$\tilde{x} = x - \epsilon \text{sign}(-\nabla_x \log \text{softmax}_{\hat{y}}(x, T)) \quad (19)$$

New training loss :

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

## 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.