UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU DE MATEMÀTIQUES

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

A Class-conditional Approach to Learning with Noisy Labels

Autor: Albert Catalán Tatjer

Director:	**Dra. Petia Radeva and Dr. Ricardo Marques**
Realitzat a:	**Departament de Matemàtiques i Informàtica.**

Barcelona, 24 de gener de 2022

Abstract

The main topic of this work is Learning with Noisy Labels (LNL). It is the field of Machine Learning concerned with training Neural Networks with noisy datasets. In particular, we have studied DivideMix, a method for LNL in the context of Computer Vision. After an extensive research we have discovered that it is unaware of the underlying class-conditional behaviour which consequently produces class imbalances. In this work, we present two class-conditional approaches to DivideMix. With this intent, we study approximate Bayesian Inference to quantify per-class uncertainty and leverage this extra information to improve the MixMatch step. In addition, we propose a class-aware policy that improves co-divide. Finally, improving DivideMix's predictive accuracy by up to 0.39% in certain noise settings.

Resum

El principal tema d'aquest treball és Learning with Noisy Labels (LNL), és el camp d'estudi de Machine Learning que es dedica a entrenar Xarxes Neuronals amb soroll a les dades d'entrenament. En concret hem estudiat el DivideMix, un mt̀ode de LNL en el context de Visió Artificial. Havent fet un estudi exhaustiu hem descobert que DivideMix no és conscient del comportament per classes adjacent i, conseqüentment, produeix desequilibris a escala de classe. En aquest treball presentem dues propostes per a equilibrar, i millorar, DivideMix. Amb aquest objectiu estudiem mètodes d'Infèrencia Bayesiana aproximats per a quantificar la incertesa de les classes. A més a més, proposem una política de divisió conscient del desequilibri que millora l'etapa co-divide de DivideMix. Finalment, aconseguim millorar la precisió de DivideMix fins a un 0.39%.

2020 Mathematics Subject Classification. 68T07, 62M45, 68U10, 65D19, 68T07

Acknowledgements

First of all, I would like to thank my supervisors, Dra. Petia Radeva and Dr. Ricardo Marques for their continuous support, wise guidance and enlightening meetings every Friday afternoon, without them this work would not have been possible. I would also like to thank Sergi Baena for his invaluable review at any given time.

Finally, I want to acknowledge my dear friends for the much-needed coffee breaks, laughs and fellowship. And my family, for the unconditional encouragement and care.

Contents

Chapter 1

Introduction

1.1 Motivation

Not long ago self-driving cars, virtual assistants, automatic environment descriptions, intelligent healthcare systems... were topics reserved exclusively for science-fiction. Although not completely solved, nowadays, it is not uncommon to watch and experience this "magical" technology. In the early days of Artificial Intelligence, the field was able to solve -in fractions of the time- tasks that are extremely hard for humans, such as matrix multiplications, but easy to formalise. However, the true challenge proved to be solving the tasks that are easy for people to perform, but hard for people to describe. Deep learning is a solution that allows computers to learn from experience -data- and understand the world in terms of a hierarchy of concepts by building complicated concepts out of simpler ones and avoiding the need for manual formalisation. A graph showing the relationships between these concepts would be deep with many layers, thus naming this field Deep Learning. Deep learning has achieved exceptional accomplishments in several fields and is now the leading edge for various important real-life applications like autonomous driving, medical diagnosis. Deep neural networks (DNNs) owe their success to the availability of extensive datasets with human annotated labels. Unfortunately, quality annotation at scale is extremely expensive and time-consuming. However, there exist automatic inexpensive methods for acquiring large-scale annotated datasets leveraging machine-generated labels [1], querying commercial search engines [2] or downloading social media data with tags [3]. Inevitably, these alternatives yield samples with *noisy labels* -samples with an annotation that does not correspond to its semantics. A recent study [4] indicates that DNNs are susceptible to overfit to noisy labels, resulting in underwhelming generalization performance. *Learning with noisy labels (LNL)* is committed to study the problem of training DNN in the presence of noise.

Needless to say, in addition to reliable predictions, a trustworthy uncertainty estimate is desirable and should be considered a key feature of any deep learning method, particularly in safety-critical application domains. *Bayesian learning* provides a mathematically grounded framework for modeling uncertainty in DNNs. Although exact Bayesian inference in DNNs is computationally intractable, there exist accepted methods of performing approximate Bayesian inference, such is Variational Inference (VI) [5]. Recently, Gal et al. [6] proved that training a neural network with dropout layers using stochastic gradient descent (SGD) is equivalent to performing VI. Therefore, providing an uncertainty quantification technique that does not require adjustments to the model - granted it has dropout layers.

Surprisingly, the connection between LNL and uncertainty quantification has been underexplored. With this in mind, the purpose of this work is to improve the state of the art by leveraging uncertainty quantification in a class-conditional manner in the context of LNL.

1.2 Thesis organisation

In the following sections of the Introduction, the state of the art for the related study fields is presented. Chapter 2 offers a brief introduction of a neural network and the different layers more relevant for this work. Chapter 3 explains approximate Bayesian Learning and introduces two well-known methods. Chapter 4 offers a detailed explanation on DivideMix. Our proposals are explained in 5. The implementation details of the experiments and the proposals are presented in Chapter 6. Chapter 7 lays out the relevant experiments conducted and their analysis. Finally, in Chapter 8 we give conclusions of the research done. And Chapter 8.1 lays out the next steps to follow.

1.3 Contributions

The main contributions that this work presents are:

- **Uncertainty Aware MixMatch**: a novel approach that extends DivideMix by leveraging the per-class uncertainty information.
- **Class-conditional co-divide**: a novel extension of DivideMix that assumes independence in the per-class loss distribution.
- **Experimental results and analysis** of both methods on two benchmark datasets, CIFAR-10 and CIFAR-100.

1.4 State of the art

1.4.1 Deep Learning and Convolutional Neural Networks

Based on *Neocognitron* by Fukushima in 1980 [8], Convolutional Neural Networks [9] have established as a core tool for image computer vision -in particular for image classification. On 2014, *VGGNet* [10] and *Inception-v1* [11] were an important breakthrough proving that the depth of a convolutional neural network is key to its performance inspiring the research on deeper networks. However, Deep Networks were hard to train because of the vanishing gradient problem, as the gradient is back-propagated repeated multiplication can make the gradient infinitively small. The *ResNet* [12] managed to leverage "identity shortcut connections" and "residual mappings" to solve this problem and boost learning in deeper networks, becoming one of the most popular architectures in computer vision. *ResNet152* was able to achieve a Top-1 Accuracy of 76.6% and Top-5 Accuracy of 93.1% on ImageNet.

The next breakthrough came with the introduction of the regularisation layer Batch-Normalisation [13]. *Inception-ResNet-v2*, an improvement on *Inception-v1* exploiting *ResNet* achieved Top-1 Accuracy of 80.01% and Top-5 Accuracy of 95.1% on Imagenet. Currently on 2021, the current Top-1 Accuracy on ImageNet is 90.88% by *CoAtNet-7* an architecture that combines convolution and transformers on top of the previous breakthroughs.

1.4.2 Learning with noisy labels

Before *DivideMix*[7], methods for training DNNs with noisy labels primarily took a loss correction approach. This correction can be categorized in two types.

One branch treated all the samples equally and then corrected the loss either explicitly of implicitly through relabeling the noisy samples. For the relabeling methods, they require access to a small set of clean samples to model the noisy samples, some examples are Conditional Random Fields [14], and knowledge graphs [15]. Iterative methods that use network predictions to relabel samples avoid the dependency on clean samples.

The second branch focuses on either reweighting the training samples or separating clean and noisy samples which results in a loss correction. The authors in [16] showed that CNNs tend to memorize simple samples first and then the remaining samples - including noise. A common policy is to consider samples with a low loss as clean ones. This measure of cleanliness is used by *Mentornet* [17] to train a mentor network to act as a guide for a student network. The authors in [18] calculate sample weights by modelling the per-sample loss with a mixture model. And [19] trains two networks separately to select the small samples for each mini-batch to train each other.

Following previous works, DivideMix [7] trains two networks and models the per-sample loss with a mixture model to split the clean and noisy data for the other network to train with. And for the first time, DivideMix discards the labels that are highly likely to be noisy and leverage the unlabeled samples to improve regularisation by training in a Semi-Supervised manner. It achieved a Top-1 accuracy of 74.76% on Clothing1M.

The current state of the art is PGDF [20] which suppresses noise by generating the samples' prior knowledge. This method achieved a Top-1 accuracy of 75.19% on Clothing1M.

Chapter 2

Neural networks

Artificial neural networks or neural networks (NN) are a class of machine learning models that have received an increasing level of interest in recent years. Such is the popularity of NNs that this work assumes a general understanding of them and only provides a brief description. *Deep Learning* [21] by Goodfellow, Bengio and Courville is a remarkable source for further information as to be seen below.

2.1 Definition

Once inspired by the biological brain, a neural network is the combination of connected units called artificial nodes - neurons that communicate sending signals. Now, 2.1 is a schema of a neural network. It is a collection of node layers, precisely of one input layer, multiple hidden layers and one output layer. Each neuron is connected to another and has an associated weight -W_i in the figure- and an activation function - σ_i. Therefore, a node performs a weighted sum over all the input signals and if its value is above the threshold, that node is activated and the signal is passed through. Otherwise, no data is passed. With this notation, the neural network from figure 2.1 can be modelled as:

$$\hat{y} = \sigma_2(\sigma_1(W_1 x + b_1)W_2 + b_2)W_3 + b3 \tag{2.1}$$

where $x = (x_1, x_2)$ is the input vector. And $\hat{y} = (\hat{y}_1, \hat{y}_2)$ is the associated output. Notice that σ is nonlinear. And therefore so is h. Then, all the learnable parameters are:

$$\theta = \{W_1, b_1, W_2, b_2, W_3, b_3\}. \tag{2.2}$$

The prediction from a neural network can be summarized as the result of a function h on x parametrized by θ. Generally, the task of a neural network consists of approximating some function f. This work is focused on the classification task - where the neural

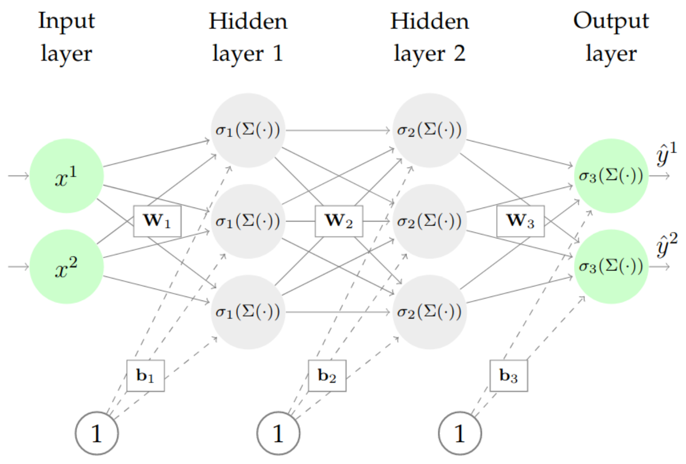

Figure 2.1: Schema of a 2 layer fully connected neural network. Inputs and outputs in green. Hidden layers in gray. Solid arrows represent the directional connections between the neurons.

network maps an input to a label of a predefined label space $\mathcal{Y}$. It is long known that NNs with at least one hidden layer are *universal approximators*, that is, given any continuous function $f(x)$ and some $\epsilon > 0$, there exists a neural network $h(x)$ with one hidden layer such that $\forall x, |f(x) - h(x)| < \epsilon$ -i.e. NNs can approximate any continuous function. However, in practice depth has proven an extremely important component, especially in Convolutional Networks - a type of neural network that realizes convolutions i.e. space sensitive processing.

2.2 Regularisation

A model is considered overfitted if it achieves great performance on training data, but the performance drops significantly on unseen data. Regularisation aims at controlling the capacity of NNs to prevent overfitting, thus preserving its generalisation performance. Amongst the regularisation techniques, Dropout and Batch-Normalisation stand out - also considered Stochastic Regularisation Techniques (SRTs) for their inherent stochastic component.

2.2.1 Dropout

Defined as a layer, the dropout [22] is implemented by only keeping a neuron active with some probability - the dropout rate - or setting it to zero otherwise. Figure 2.2 illustrates the dropout. On the left, one can see a standard neural network and on the right the same

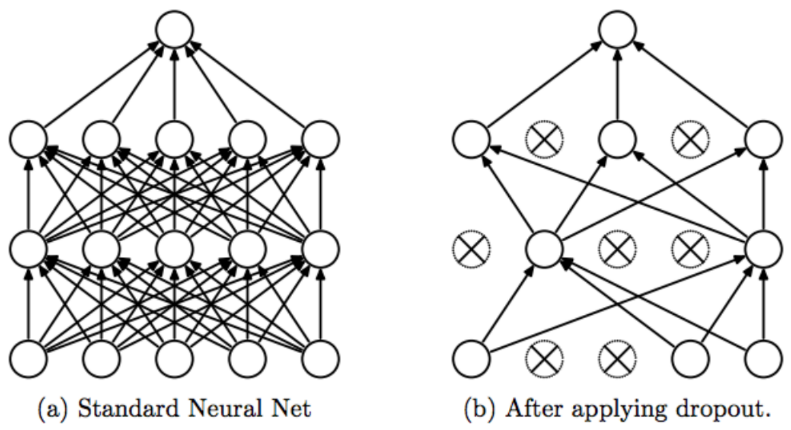

Figure 2.2: On the left, a fully connected neural network. On the right, a fully connected neural network with dropped neurons.

neural network at training after applying dropout, where only a subset of the nodes are activated. Training is done with the subnet. And on test time, the dropout is ignored and the full neural network is used. An explanation would be that it forces the neural network to not rely on a subset of the connections and encourages its full capacity by training different subnets - and connections - randomly. However, the source of its effectiveness is not fully understood.

2.2.2 Batch-Normalisation

Also implemented as a layer, Batch-Normalisation [13] is used to standardise each layer's input to have zero mean and unit variance. Formally, let g^l be the input for a fully connected layer l. Then, the BN operation uses the mini-batch mean μ_B^l and the mini-batch variance $(\sigma^2)_B^l$ to compute

$$\hat{g}^l = \left(\frac{g^l - \mu_b^l}{\sqrt{(\sigma^2)_B^l}} \right) \tau^l + \beta^l, \tag{2.3}$$

where τ^l and β^l are the scale and shift parameters learned through back propagation. Note that σ and μ are also learnable parameters.

At training time, Batch-Normalisation layers keep running batch statistics (σ, μ) that are updated with each different mini-batch. At testing time, the network is supposed to see only the test data point it is doing inference over. Hence, (σ, μ) used are those estimated during training rather than mini-batch statistics.

Chapter 3

Uncertainty Quantification. Approximate Bayesian Inference.

Imagine a binary image classification problem devoted to classify dogs and cats. If asked to classify a new cat photo, the prediction is expected to have high confidence. However, if forced to classify a picture of a submarine, it is desirable to get a low confidence prediction. Uncertainty is a measure about the confidence of a prediction. And this chapter is devoted to study Uncertainty Quantification in the context of DNNs.

3.1 Overview

At least two different types are distinguished, *aleatoric* and *epistemic*. Aleatoric or data uncertainty refers to the inherent variability in the outcome of the experiment. That is, its notion of randomness, a prototypical example is coin flipping, where no additional information can reduce its stochastic behaviour. Differently, epistemic or predictive uncertainty refers to the ignorance of the model and can be reduced with further knowledge. Now, machine learning essentially extracts models from data. Such models learn their predictive capacity by generalizing the available data based on a process of induction. Hence, the models are only hypothetical and consequently uncertain, such is a source of epistemic uncertainty.

3.2 Supervised learning

Supervised learning is concerned with training a learner -neural network- given a set of N data points and its associated labels,

$$\mathcal{D} := \{(x_i, y_i)\}_{i=1}^{N} \subset \mathcal{X} \times \mathcal{Y} \tag{3.1}$$

where $\mathcal{X}$ is the sample space and $\mathcal{Y}$ the set of outcomes that can be associated with a sample. With the goal that, given a new point, x_{n+1} to predict its associated outcome y_{n+1}.

Formally, it is assumed that the training examples (x_i, y_i) are independent and identically distributed (i.i.d) following an unknown probability P on $\mathcal{X}x\mathcal{Y}$. Let $\mathcal{H}$ be the hypothesis space consisting of hypothesis $h_\theta : \mathcal{X} \longrightarrow \mathcal{Y}$ mapping samples to labels, the subscript θ represents the underlying parameters inherent in h. And let $l : \mathcal{Y}x\mathcal{Y} \longrightarrow \mathbb{R}$ be a loss function mapping hypothesis h to a measure of its "cost", the higher it is, the worse the hypothesis. Then, the learner's goal is to induce a hypothesis with low risk (expected loss). That is:

$$\mathcal{R}(h) := \mathbb{E}[l(h_\theta(\mathcal{X}), \mathcal{Y})] = \int_{\mathcal{X}\times\mathcal{Y}} l(h_theta(x), y) dP(x, y).$$

To do so, the learner "guesses" a hypothesis $\hat{h}_\theta$ that minimizes the empirical risk given the training data $\mathcal{D}$ -the performance of a hypothesis on the training data-

$$\mathcal{R}_{emp} := \frac{1}{N} \sum_{i=1}^{N} l(h_\theta(x_i), y_i), \tag{3.2}$$

$$\hat{h} := \arg\min_{h_\theta \in \mathcal{H}} \mathcal{R}_{emp}(h). \tag{3.3}$$

Yet, $\mathcal{R}_{emp}(h_\theta)$ is only an estimation of the true risk, and therefore the $\hat{h}$ will typically not coincide with the true risk minimizer $h_\theta^* := \arg\min_{h_\theta \in \mathcal{H}} \mathcal{R}(h_\theta)$.

3.3 Uncertainty for Supervised learning

The standard procedure for supervised learning takes a frequentist approach, the unknown set of underlying parameters θ are treated as a set of fixed unknown values to estimate. Where the prediction $\hat{h}_\theta(x_{n+1})$ is in the form of a point estimation y_{n+1}. Given a trained learner $\hat{h}_\theta$ -that is an estimated set of θ-, the predicted outcome y_{n+1} of a data point x_{n+1}, can be the loss minimizer, $y_{n+1} = \arg\min_{y \in \mathcal{Y}} l(\hat{h}_\theta(x_{n+1}), y)$.

Generally, in classification, there is a softmax layer at the end of the pipeline:

$$\sigma(\hat{h}_\theta(x_i)) := \frac{\exp \hat{h}_\theta(x_i)^c}{\sum_{j \neq c}^{C} \exp \hat{h}_\theta(x_i)^j}, \ \forall c = 1, \ldots, C \tag{3.4}$$

that maps the output loss to the predictive probability.

However, it is often erroneously interpreted as model confidence. It is not useful to quantify uncertainty. Figure 3.1 from [6] displays the overconfidence of softmax. On the left, one can see an arbitrary function $f(x)$ of data x that is the softmax input. On the right, $\sigma(f(x))$ as a function of data x, the softmax output. The values between the gray dashed lines are used as training data. The shaded area represents unseen data. Notice that $f(x)$ has small fluctuation -that is uncertainty captured by f. However, softmax ignores it and outputs a prediction with confidence 1 for the shaded area.

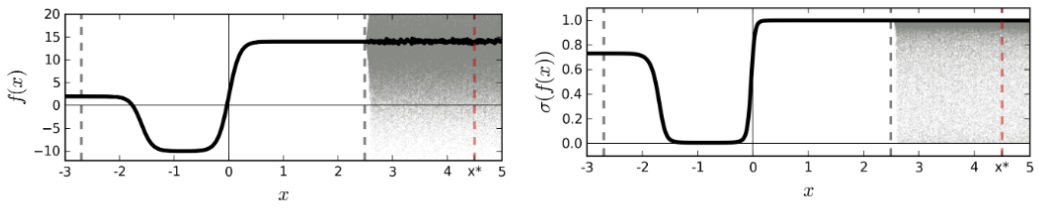

Figure 3.1: On the left, an arbitrary function $f(x)$ as a function of data x (the softmax input). On the right $\sigma(f(x))$ as a function of data x (softmax output). Sketches softmax input and output for an idealized binary classification problem. Training data is given between the grey dashed lines, roughly between -3 and 3. Function uncertainty is shown with a gray shaded area. A red dashed line marks a point x_{n+1} far from the training data. The point x_{n+1} is classified as class 1 with probability 1, completely ignoring the function uncertainty -the small fluctuation as $f(x)$ apparent in the shaded area-.

In conclusion, the standard approach to supervised-learning does not offer uncertainty quantification capability.

3.4 Bayesian Neural Networks

Bayesian Neural Networks (BNN) offer a different approach. Given a data point, x_{n+1} its prediction is a probability distribution instead of a point estimate. With this, the mean would be used as the prediction and some measure of variance would estimate its uncertainty. Figure 3.2 illustrates the difference between the standard approach and BNNs. To accomplish this, the unknown set of parameters θ are treated as random variables instead of unknown points. That is, given a training set, Bayesian learning aims to learn

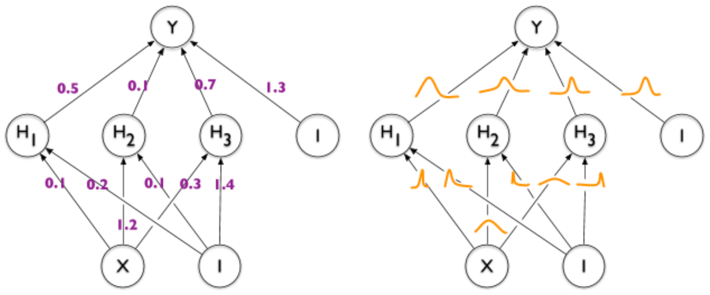

Figure 3.2: On the left, a diagram representing a deterministic NNs, where the underlying parameters are points. On the right, a representation of a Bayesian Neural Network with distributions placed over their weights.

the posterior distribution[1] of θ rather than a point estimate of it. It may seem strange to assume that the model is controlled by a set of random parameters. However, treating θ as random variables does not imply a random nature for them. The associated randomness encapsulates the model's uncertainty [23].

Therefore, to train a Bayesian Neural Network is to estimate the whole posterior distribution of θ by doing the full Bayesian inference. Full Bayesian inference uses the Bayes rule accounting the training dataset to estimate the posterior of θ, $p(\theta|\mathcal{D})$,

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}, \tag{3.5}$$

where $p(\theta)$ is the prior distribution and $p(\mathcal{D})$ is the evidence and $p(\mathcal{D}|\theta)$ is the data likelihood,

$$p(\mathcal{D}|\theta) = p(\mathcal{Y}|\mathcal{X},\theta) = \int_\theta p(\mathcal{Y}|\mathcal{X},\theta)p(\theta)d\theta. \tag{3.6}$$

Then, performing Bayesian inference on unseen data x_{n+1} is done taking the expectation of the output over the trained posterior distribution of the parameter, say $p(\hat{\theta}|\mathcal{D})$ as

$$p(y_{n+1}|x_{n+1}) = \mathbb{E}_{p(\hat{\theta}|\mathcal{D})}p(y_{n+1}|x_{n+1},\hat{\theta}). \tag{3.7}$$

Unfortunately, the exact computation of the posterior (3.5) is intractable, because there is no analytical solution to the integral (3.6) -except for a few special cases requiring the use of conjugate priors [24]- because the neural network is a complicated nonlinear function. Therefore, an exact Bayesian Neural Network cannot be found.

[1]Note that the posterior distribution of θ is its conditional probability distribution after the evidence of the training set is accounted for. And the prior distribution of θ is its probability distribution before the evidence of the training set.

3.5 Variational Inference for Approximate Bayesian Neural Networks

Fortunately, there are alternatives that approximate BNNs. VI is a well-known and accepted method for performing Bayesian inference [5]. It is used to approximate the true Bayesian posterior distribution over the model parameters θ (see equation (3.5)). Let $q_\phi(\theta)$ be some approximate distribution, parametrized by ϕ[2]. The goal of VI is to find the ϕ that minimizes the Kullback-Leibler divergence -a measure of similarity between two probability distributions- of $p(\theta|\mathcal{X},\mathcal{Y})$ with respect to $q_\phi(\theta)$. That is, to find:

$$\arg\min_\phi D_{KL}(q_\phi(\theta)\|p(\theta|\mathcal{D})), \tag{3.8}$$

which is equivalent to:

$$\arg\max_\phi \int_\phi q_\phi(\theta) ln(p(\mathcal{Y}|\mathcal{X},\theta))d\theta - D_{KL}.(q_\phi(\theta)||p(\theta) \tag{3.9}$$

This is often a simpler optimization problem than equation (3.8).

Now, with an approximate posterior distribution $q_\phi(\theta)$, it is possible to perform approximate Bayesian Inference by replacing $p(\theta|\mathcal{D})$ with $q_\phi(\theta)$ in (3.5).

3.6 Stochastic Regularization Techniques for VI

3.6.1 Montecarlo Dropout

Surprisingly, Gat et al. [6] proved that training a neural network with stochastic gradient descent (SGD) with dropout is equivalent to performing VI for a Bayesian model. Therefore, any neural network that uses dropout as a regularization technique (which is really common) is an approx. BNN - without the need for any change in its architecture.

To compute the distribution of a prediction $h_\theta(x_{n+1})$, it is only needed to keep the dropout layer in train mode and sample T different times for x_{n+1}, $h_\theta(x_{n+1})_t$ indicates the t-th stochastic forward pass. Note that the only change comes at test time, training is exactly the same. Now, the mean can be used as the prediction and some measure of variance can estimate its uncertainty.

[2]Note that $q_\phi(\theta)$ can only be optimized to the extent allowed by its functional form and parameter settings. This is the reason that makes VI an approximation technique.

Limitations and shortcomings

However, Montecarlo Dropout (MCDO) is dependent on dropout as a stochastic regularization technique (SRT). It is a significant limitation because the introduction of Batch-Normalisation has shifted the use of Dropout as SRT to Batch-Normalisation. In addition, Folgoc et al. [25] argue that MCDO is such a bad approximation that it is not meaningful to call it Bayesian uncertainty.

In conclusion even though it is an uncertainty quantification technique, MCDO is useful for the ability to acquire uncertainty at no cost rather than the quality of it.

3.6.2 Montecarlo Batch-Normalisation

In the introduction of Montecarlo Dropout [6], Gal et al. mentioned the possibility of using similar arguments with other SRTs. In 2018, Teye et al. [26] proved that Batch-Normalisation allows for the same outcome. That is, training a NN with stochastic gradient descent and Batch-Normalisation layer as SRT is equivalent to performing VI for a Bayesian Model.

This time, the stochastic component is not as straightforward as with MCDO. Recalling section 2.2.2 -on training mode- the BN layers depend on the mini-batch statistics of its input. Therefore, the stochastic component in the Monte Carlo Batch-Normalisation (MCBN) comes from randomly sampling the mini-batch. To compute the distribution of a prediction $h_\theta(x_{n+1})$, it is needed to keep the batch-normalisation layer in training mode and to sample T different times for x_{n+1}, $h_\theta(x_{n+1})^t$ with different mini-batch sample each time. To emphasize the importance of the randomness in the mini-batches, consider a naive problem with three samples per mini-batch. Let the first forward pass be $B_i^1 = \{1,7,4\}$, that is i-th mini-batch on the first forward pass. If for any other t, say $t = 39$ the mini-batch $B_j^{39} = \{1,7,4\}$, then the prediction of the samples $\{1,7,4\}$ will be exactly the same for $t = 1$ and $t = 39$.

In conclusion, this uncertainty quantification method is also "free", no modification is needed at training time. The only change needed is to set the BN layers to training mode at test time and sample T times with different mini-batch. Now, the mean can be used as the prediction and some measure of variance can estimate its uncertainty.

Limitations and shortcomings

MCBN is part of the same family of approximate Bayesian methods and therefore with the same shortcomings stated by folgoc. In addition, the Batch-Normalisation layers have learnable parameters σ, μ which represent the whole population statistics. Therefore, when used in training mode, the model is not leveraging these trained parameters and leads to a slightly worse predictive accuracy. Also, if the mini-batches are the size of all the dataset, there will be no randomness and MCBN will not provide uncertainty quantification. In conclusion, it is a technique used for its convenience rather than its quality.

Chapter 4

Learning with Noisy Labels. DivideMix.

This thesis is primarily inspired in the method **DivideMix** for image classification on LNL by Li et al. [7] and a later extension **Contrast to Divide (C2D)** proposed by Zheltonozhskii et al. [27].

4.1 The Challenge of Learning with Noisy Labels

Beforehand, the challenge of image classification is to train a model that is capable of assigning one of the predefined classes to any given unseen image. To do so, NNs are trained with large annotated datasets. The success of these models comes from the ability to find common patterns in the samples of a class. However, when the annotated dataset has noisy labels, the model might find patterns within the noisy class thus overfitting to the noisy dataset resulting in poor generalization performance. It is noteworthy that aleatoric uncertainty - data uncertainty - can be interpreted as a source of label noise. For example, food image classification is a domain that has to deal with this "implicit noise", where a food image is ambiguous and different humans can classify it differently. Therefore, the challenge of LNL goes beyond solving the data annotation cost and also learning in ambiguous domains.

4.2 Overview of DivideMix

DivideMix explores the connection of semi-supervised learning with LNL. C2D is an extension that leverages self-supervised training to minimize noise memorization during the warm-up stage. Broadly, DivideMix trains two networks simultaneously to filter errors for each other, avoiding confirmation bias where the model would accumulate its errors. At each epoch, and for each network, the co-divide step fits a Gaussian Mixture Model (GMM) on its per-sample loss distribution dividing the training samples into a labeled set ($\mathcal{X}$) and unlabeled set ($\mathcal{U}$), which are then used to train the other network. Afterwards, at each mini-batch, one network, guided by the other, leverages both labeled and unlabeled samples to perform MixMatch, a semi-supervised learning technique. An overview is shown in Figure 4.1, and in more detail on Algorithm 1.

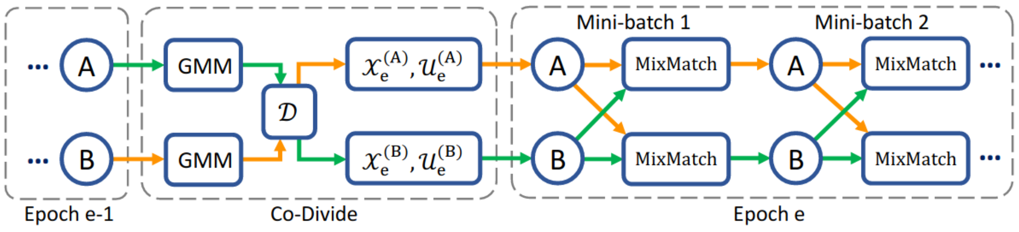
Mini-batch 1
Mini-batch 2
A
B
GMM
GMM
$\mathcal{D}$
$\mathcal{X}_e^{(A)}, \mathcal{U}_e^{(A)}$
$\mathcal{X}_e^{(B)}, \mathcal{U}_e^{(B)}$
MixMatch
Epoch e-1
Co-Divide
Epoch e

Figure 4.1: DivideMix trains two networks (A and B) simultaneously. At each epoch, a network models its per-sample loss distribution with a GMM to divide the dataset into a labeled set $\mathcal{X}$(mostly clean) and an unlabeled set $\mathcal{U}$ (mostly noisy), which is then used as training data for the other network. At each mini-batch, a network performs semi-supervised training using an improved MixMatch method that involves label co-refinement on $\mathcal{X}$ and label co-guessing on the $\mathcal{U}$. C2D's self-supervised pre-training is performed before epoch 0, as an extra step of initialization.

4.3 Co-divide by loss modeling

The main purpose of co-divide is to find ω_i, the probability of a sample being clean to classify it as labeled or as unlabeled (4.1). Fortunately, Arpit et al. in [28] discovered that deep networks tend to learn clean samples faster than noisy samples, therefore leading to lower loss for the clean samples [29],[30]. To find the probability of a sample being clean, a two-component Gaussian mixture model (GMM) is fit to the per-sample loss distribution [18]. Formally, let $\{D = (X, Y)\} = \{(x_i, y_i)\}_{i=1}^{N}$ be the training dataset, where x_i is an image, $y_i \in \{0, 1\}^C$ is the one-hot label over C classes, potentially noisy. Then, given a model with parameters θ, the cross-entropy loss $l(\theta)$ indicates how well the model fits the training samples:

$$\omega_i := P(x_i \in \mathcal{K}), \tag{4.1}$$

$$\ell(\theta) \;=\; \{\ell_i\}_{i=1}^{N} \;=\; \left\{ -\sum_{c=1}^{C} y_i^c \cdot log(p_{model}(x_i;\theta)) \right\}_{i=1}^{N}$$

where $\mathcal{K}$ is the clean set and p^c_{model} is the model's output softmax probability for class c.

Finally, the training data is divided into $\mathcal{X}$ and $\mathcal{U}$ by setting a threshold τ on w_i. However, training a model using the model divided by itself could lead to confirmation bias (i.e. the model is prone to confirm its mistakes, [31]), overfitting of the wrongly classified samples into the labeled set would keep their loss low. Therefore, the two networks are kept diverged from each other due to different parameter initialization, different training data division, different mini-batch sequence and different training targets. Thus, leveraging the ability to filter different types of error, makes the model more robust to noise.

The choice of GMM as the mixture model follows [32], they show that a GMM can better distinguish clean and noisy samples due to its flexibility in the sharpness distribution. In addition, the fitting of the two-component GMM to the per-sample loss is performed by the Expectation-Maximization algorithm. Therefore, the posterior probability $w_i = p(g|l_i)$ is the clean probability, where g is the Gaussian component with smaller loss (mean).

4.4 Warm-up challenge

For initial convergence, the algorithm depends on the NN's loss to be meaningful in order to provide a reasonable division. Therefore, the model needs to warm-up for a few epochs by training on all data using the standard cross-entropy loss. This warm-up is more effective for symmetric label noise (i.e. uniformly distributed across classes).

4.4.1 Confidence Penalty for Asymmetric Noise.

However, for asymmetric label noise (i.e. class-conditional), the network quickly overfits to noise during warm-up leading to most samples having near-zero normalized loss thus greatly diminishing the GMM's effectivity. To address this issue, confident predictions are penalized by adding a negative entropy term $-\mathcal{H}$ [33] to the cross-entropy loss during warm up:

$$\mathcal{H} = -\sum_{c} p^c_{model}(x;\theta) \cdot log(p^c_{model}(x;\theta)) \tag{4.2}$$

By maximizing the entropy, l becomes more evenly distributed and easier modeled by the GMM.

4.4.2 C2D: self-supervised pre-training.

Zheltonozhskii et al. [27] show that the model performance is constrained by the noise learned during the warm-up phase. The recent success in self-supervised learning [34, 35], i.e. learning without labels, encourages a new approach. It hypothesizes that performing a self-supervised contrastive learning and then proceeding with DivideMix can better detect noise in the data. Figure 4.2 shows that there is a strong correlation between linear accuracy and noise detection. Also, it is seen that C2D beats no pre-training (i.e. the standard approach) and pre-training on another annotated dataset (in this case, ImageNet), both in noise detection and in model accuracy.

In conclusion, C2D experimentally shows an improvement on DivideMix (and other LNL methods that rely on a warm-up phase), by self-supervised pretraining beforehand, followed by a 5 epoch warm-up and proceeding with the rest of DivideMix.

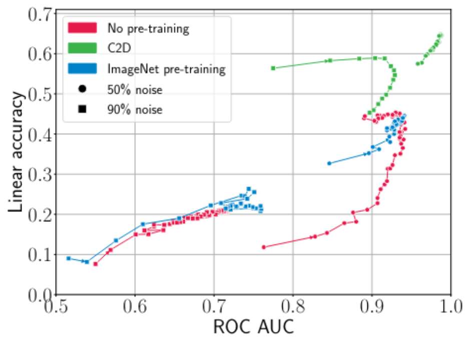

Figure 4.2: The ROC-AUC score of noise detection and the linear accuracy (calculated using clean labels), under various noise levels for CIFAR-100 for standard warm-up, ImageNet pre-training and C2D. Each point is one epoch of training, the arrowhead denotes time direction, markers denote noise level and colors denote the pre-training scheme.

4.5 MixMatch with label co-refinement and co-guessing

This step exploits MixMatch's [36] *consistency regularization*, that is, the model should output the same prediction across different augmentations of the same sample as well as *entropy minimization*, to encourage confident predictions on unlabeled data.

In detail, at each epoch, each neural network is trained while the other is kept fixed. Given a mini-batch of labeled samples $\{(x_b, y_b, w_b)\ s.t.\ b \in (1, \ldots, B)\}$ with their corresponding one-hot labels and clean probability, and a mini-batch of unlabeled samples $\{u_b\ s.t.\ b \in (1, \ldots, B)\}$, both undergo *data augmentation*, which applies input transformations assumed to leave the class semantics unaffected (thus, potentially providing a near infinite stream of new, modified data), (see lines 14, 15). Then, for the labeled set, *label co-refinement* is performed by linearly combining the label y_b with the network's prediction p_b (averaged across the multiple augmentations of x_b) weighted by the clean probability w_b produced by the other network, (4.5), Algorithm 1 lines 17, 18. Finally, *sharpening* its temperature to produce lower-entropy predictions, (4.5), Algorithm 1 line 19.

$$\overline{y}_b = w_b y_b \ + \ (1 - w_b) p_b \tag{4.3}$$

$$\hat{y}_b = \ Sharpen(\overline{y}_b, T) \ = \frac{\overline{y}_b^{c'\frac{1}{T}}}{\sum_{c=1}^{C} \overline{y}_b^{c\frac{1}{T}}}, \ for \ c' = 1, \ldots, C \tag{4.4}$$

Afterwards, for the unlabeled set, *label co-guessing* is performed by averaging the predictions from both networks across the augmentations of u_b, and then temperature sharpening is applied to the result, Algorithm 1 lines 20, 21.

We acquire $\hat{\mathcal{X}}$ ($\hat{\mathcal{U}}$), which consists of multiple augmentations of labeled (unlabeled) samples and their co-refined (co-guessed) labels. Following MixMatch, the data is "mixed", interpolating each sample with another sample randomly chosen from the combined mini-batch of $\hat{\mathcal{X}}$ and $\hat{\mathcal{U}}$. Precisely, for a pair of samples (x_1, x_2) and their corresponding labels (p_1, p_2), the mixed (x', p') is given by:

$$\lambda \sim Beta(\alpha, \alpha), \tag{4.5}$$

$$\lambda' = max(\lambda, 1 - \lambda), \tag{4.6}$$

$$x' = \lambda' x_1 + (1 - \lambda') x_2, \tag{4.7}$$

$$p' = \lambda' p_1 + (1 - \lambda') p_2. \tag{4.8}$$

We transform $\hat{\mathcal{X}}$ and $\hat{\mathcal{U}}$ to $\mathcal{X}'$ $\mathcal{U}'$. Note that $\mathcal{X}$ is closer to $\hat{\mathcal{X}}$ than $\hat{\mathcal{U}}$, because of equation (4.6). The loss on $\mathcal{X}'$ is the cross-entropy loss (see equation (4.9)) and the loss on $\mathcal{U}'$ is the mean squared error (equation (4.10)):

$$\mathcal{L}_{\mathcal{X}} = -\frac{1}{|\mathcal{X}'|} \sum_{x,p \in \mathcal{X}'} \sum_{c} p_c log(p^c_{model}(x;\theta)), \tag{4.9}$$

$$\mathcal{L}_{\mathcal{U}} = \frac{1}{|\mathcal{U}'|} \sum_{x,p \in \mathcal{U}'} \| p - p_{model}(x;\theta) \|_2^2 . \tag{4.10}$$

Note that, under high levels of noise, the network would be encouraged to predict the same class to minimize the loss. Following [18] and [**?**], a regularization loss that uses a uniform prior distribution π (i.e. $\pi = 1/C$) to regularize the model's average output across all samples in the mini-batch:

$$\mathcal{L}_{reg} = \sum_{c} \pi_c log \left(\pi_c \middle/ \frac{1}{|\mathcal{X}'| + |\mathcal{U}'|} \sum_{x \in \mathcal{X}' + \mathcal{U}'} p^c_{model}(x;\theta) \right) . \tag{4.11}$$

Altogether, we get:

$$\mathcal{L} = \mathcal{L}_{\mathcal{X}} + \lambda_u \mathcal{L}_{\mathcal{U}} + \lambda_r \mathcal{L}_{reg}, \tag{4.12}$$

where λ_u and λ_r are hyper-parameters to control the influence of the unsupervised loss. Now the network learns performing backpropagation on $\mathcal{L}$ (equation (4.12)).

Algorithm 1 DivideMix with C2D. Line 4-8: co-divide; Line 17-18: label co-refinement; Line 20: label co-guessing.

Input: $\theta^{(1)}$ and $\theta^{(2)}$, training dataset $(\mathcal{X}, \mathcal{Y})$, clean probability threshold τ, number of augmentations M, sharpening temperature T, unsupervised loss weight λ_u, Beta distribution parameter α for MixMatch.

```
1:  $\theta^{(1)}, \theta^{(2)} = CoDivide(\mathcal{X}, \mathcal{Y}, \theta^{(1)}, \theta^{(2)})$   # self-supervised pre-training.
2:  $\theta^{(1)}, \theta^{(2)} = WarmUp(\mathcal{X}, \mathcal{Y}, \theta^{(1)}, \theta^{(2)})$   # standard training (with confidence penalty)
3:  while $e < MaxEpoch$ do
4:      $\mathcal{W}^{(2)} = GMM(\mathcal{X}, \mathcal{Y}, \theta^{(1)})$   # model per-sample loss with $\theta^{(1)}$ to obtain clean probability for $\theta^{(2)}$
5:      $\mathcal{W}^{(1)} = GMM(\mathcal{X}, \mathcal{Y}, \theta^{(2)})$   # model per-sample loss with $\theta^{(2)}$ to obtain clean probability for $\theta^{(1)}$
6:      for $k = 1, 2$ do   # train the two NN one by one
7:          $\mathcal{X}_e^{(k)} = \{(x_i, y_i, \omega_i) | \omega_i \geq \tau, \forall (x_i, y_i, \omega_i) \in (\mathcal{X}, \mathcal{Y}, \mathcal{W}^{(k)})\}$   # labeled training set for $\theta^{(k)}$
8:          $\mathcal{U}_e^{(k)} = \{x_i | \omega_i < \tau, \forall (x_i, \omega_i) \in (\mathcal{X}, \mathcal{W}^{(k)})\}$
9:          for $iter = 1$ $to$ $num\_iters$ do
10:             From $\mathcal{X}_e^{(k)}$, draw a mini-batch $(x_b, y_b, \omega_b) | b \in (1,\ \ldots,\ B)$
11:             From $\mathcal{U}_e^{(k)}$, draw a mini-batch $(u_b) | b \in (1,\ \ldots,\ B)$
12:             for $b = 1$ to $B$ do   # apply $m^{th}$ augmentation
13:                 for $m = 1$ to $M$ do
14:                     $\hat{x}_{b,m} = Augment(x_b)$
15:                     $\hat{u}_{b,m} = Augment(b_b)$
16:                 end for
17:                 $p_b = \frac{1}{M} \sum_m p_{model}(\hat{x}_{b,m})$   # average the predictions across augmentations of $x_b$
18:                 $\bar{y}_b = \omega_b y_b + (1 - \omega_b) p_b$   # refine ground-truth label guided by the clean probability produced by the other network
19:                 $\hat{y} = Sharpen(\bar{y}_b, T)$   # apply temperature sharpening to the refined label
20:                 $\bar{q}_b = \frac{1}{2M} \sum_m (p_{model}(\hat{u}_{b,m}; \theta(1)) + p_{model}(\hat{u}_{b,m}; \theta(2)))$   # co-guessing: average the predictions from both networks across augmentations of $u_b$
21:                 $q_b = Sharpen(\bar{q}_b, T)$# apply temperature sharpening to the guessed label
22:             end for
23:             $\hat{\mathcal{X}} = \{(\hat{x}_{b,m}, \hat{y}_b) | b \in (1, , B)\}$   # augmented labeled mini-batch
24:             $\mathcal{U} = \{(\hat{\ }_{\lfloor,\Updownarrow}, \amalg_{\lfloor}) | \hat{\lfloor} \in (\infty,\ \ldots,\ \mathcal{B}) u\}$   # augmented unlabeled mini-batch
25:             $\mathcal{L}_{\mathcal{X}}, \mathcal{L}_{\mathcal{U}} = MixMatch(\hat{\mathcal{X}}, \hat{\mathcal{X}})$   # apply MixMatch
26:             $\mathcal{L} = \mathcal{L}_{\mathcal{X}} + \lambda_u \mathcal{L}_{\mathcal{U}} + \lambda_r \mathcal{L}_{reg}$   # total loss
27:             $\theta^{(k) = SGD(\mathcal{L}, \theta^{(k)})}$   # update model parameters
28:         end for
29:     end for
30: end while
```

Chapter 5

Our proposals for improved LNL

Generally, in the problem of image classification, some classes are expected to perform better than others, it may be because they are comparatively more unique, they have limited backgrounds, more characteristic features... In LNL this phenomenon can be augmented by the noise, further separating the "unique" classes from the classes that are harder or that have features that are more common. However, DivideMix does not account for possible class imbalances and performs the same treatment to all the samples. This presents a window to improve DivideMix leveraging the underlying class-conditional behaviour. This chapter introduces our proposals for improved LNL, namely **Uncertainty Aware MixMatch** (ψ-MixMatch) and **Class-conditional Co-divide**.

5.1 Overview

The focus of this chapter is to detail two class-conditional approaches to improve DivideMix. Foremost, we propose an **Uncertainty Aware MixMatch** (ψ-MixMatch) as an extension of the MixMatch that is aware of uncertainty of every class. Secondly, we propose a **Class-conditional co-divide** (CCGMM) fitting a Gaussian mixture model for each different class to account for potential per-sample loss imbalance between the classes. Beforehand, it is important to lay out the challenges to overcome. Figure 5.1 outlines the proposals. The novelties are marked in red, and the rest is plain DivideMix. In red on can see the modifications proposed, the rest is DivideMix as seen in figure 4.5. The first figure is Class-conditional co-divide (CCGMM), a modification of the Gaussian mixture model over DivideMix. The second image is an Uncertainty MixMatch, the quantification of class uncertainty -ψ- and its application on MixMatch. A and B are the two networks involved in DivideMix, the colors of the arrows encode the influence of a network. Network A fits a GMM (or CCGMM) to its per-sample loss, then divides the training set in $\mathcal{X}$ and $\mathcal{U}$.

This division is then used by the network B to do training while network A is fixed and "helping" in the co-labeling and co-guessing steps.

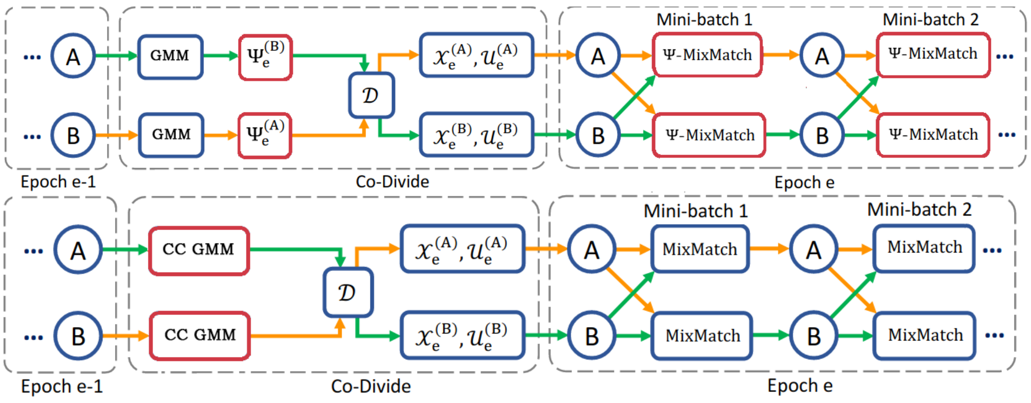

Figure 5.1: An overview of our proposals: on the top the ψ-MixMatch proposal and on the bottom Class-conditional co-divide proposal.

5.2 Uncertainty Aware MixMatch

5.2.1 Motivation

In deep learning, it is no surprise that some classes are learned easier than others. For example, imagine a ternary image classification problem with the classes *dog*, *cat* and *dolphin*, clearly the classes *dog* and *cat* have similar features and the class *dolphin* has unique features. Therefore, the model is expected to perform better identifying *dolphins* than *cats* and *dogs*. LNL can further magnify this phenomenon by blurring the semantic features of a class. Thus increasing the overall uncertainty of a class. Uncertainty Quantification techniques can be expected to measure the noisiness of every class, allowing for an uncertainty aware treatment.

Now, **entropy** is a measure of uncertainty. It is the average level of uncertainty inherent to the possible outcomes (equation 5.1). For a sample, the higher it is, the more the T observations of its loss differ:

$$P(x) = \frac{1}{T}\sum_{t=0}^{T}(Softmax(h_\theta(x)_t)),$$

$$\mathcal{H}(x) = -\sum_{c=0}^{C} P(x)^c \cdot log(P(x)^c), \tag{5.1}$$

where x is a sample, $h_\theta(x)_t$ is the output of the neural network for the sample x in the t-th forward pass, it is a vector with dimension C. Figure 5.2 shows the distribution of the per-sample loss and entropy, where the samples with higher entropy (equation 5.1) are the ones that are more susceptible to be miss-divided in the co-divide step -those that are on the fringe between the noisy set $\mathcal{N}$ - in red - and the clean set $\mathcal{K}$ - in blue. Therefore, the classes with higher uncertainty need more attention from the neural network. It is important for all the classes to be learned at a relatively similar pace to avoid bias.

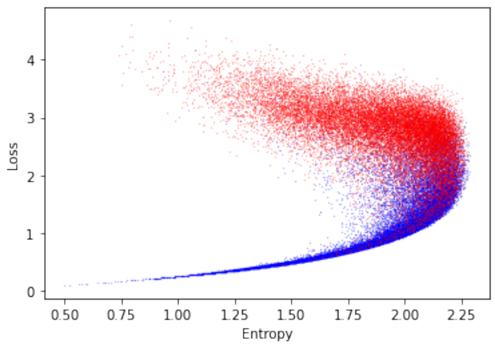

Figure 5.2: Analysis done in CIFAR-10 with 50% symmetric noise after epoch 9 with $T = 50$ stochastic forward passes. Y-axis is the per-sample loss, X axis is the per-sample entropy. In red $\mathcal{N}$, the noisy samples and in blue $\mathcal{K}$, the clean samples. Red dots are printed last, thus blue dots might be underneath. Check figure 2 for more views on this distribution.

5.2.2 Proposal for Uncertainty Aware MixMatch

Then, to encourage the neural network's focus on certain classes, their loss is increased. It can be interpreted as, the higher the loss of a sample (class) the higher the NN is rewarded to change its weights and biases in backpropagation, therefore "paying more attention" to this sample (class). There are three losses in DivideMix, $\mathcal{L}_{\mathcal{X}}$, $\mathcal{L}_{\mathcal{U}}$ and $\mathcal{L}_{reg}$, experimental results -see section 7.4- show a better performance when only weighting $\mathcal{L}_{\mathcal{X}}$.

The solution (ψ-MixMatch) proposes the mean of the variance of all the samples of a class to measure its class uncertainty. For each epoch -and for each neural network- it is computed at the co-divide step, and then used by the other network to weight the labeled loss $\mathcal{L}_{\mathcal{X}}$, leveraging the dual network architecture and avoiding possible confirmation bias. Precisely, (5.2) defines the vector class uncertainty. It is min-max normalized to control its range.

$$\psi' = \frac{1}{\#\mathcal{D}_y} \cdot \sum_{x \in \mathcal{D}_y} s_t^2(Softmax(h_\theta(x))^y), \tag{5.2}$$

$$\psi = \frac{\psi' - min_c\ \psi'}{max_c\ \psi' + min_c\ \psi'}. \tag{5.3}$$

Note that $h_\theta(x)$ is the output of the neural network given the input x, which is a vector of dimension (C,T) number of classes by number of stochastic forward passes. Then,

$s_t^2(Softmax(h_\theta(x))^y)$ is the variance of the Softmax of the class y across t.
All together,

$$\mathcal{L}_\mathcal{X} = -\frac{1}{\mathcal{X}'} \sum_{x,y,p \in \mathcal{X}'} (1 + \lambda_\mathcal{X} \cdot \psi^y) \cdot \sum_c p_c log(p^c_{model}(x; \theta)), \tag{5.4}$$

where $\lambda_\mathcal{X}$ is a hyper-parameter introduced to control the impact of ψ. Note that $\lambda_\mathcal{X} = 0$ is the DivideMix's case.

5.3 Class-conditional Co-divide

5.3.1 Motivation

Briefly recalling chapter 4, the step of co-divide is responsible for classifying the training data into: (i) the labeled set $\mathcal{X}$, hopefully made of the clean samples used for supervised training; and (ii) the unlabeled set $\mathcal{U}$, containing the samples deemed noisy for self-supervised training. This decision is based on the assumption that the per-sample loss of the clean samples follows a different distribution from that of the noisy samples, and that both distributions are normal. Hence, a GMM can be used to extract these two subpopulations from the data. However, the division might contain errors and these two sets might contain wrongly classified samples. Therefore, this binary classification has four hidden subsets:

a) True positives or $\mathcal{X}^{true}$, are the clean samples classified as the labeled set $\mathcal{X}$;

b) False Positives or $\mathcal{X}^{false}$, are noisy samples classified as the $\mathcal{X}$;

c) True Negatives or $\mathcal{U}^{true}$, are noisy samples classified as the unlabeled set $\mathcal{U}$;

d) False Negatives or $\mathcal{U}^{false}$, are clean samples classified as $\mathcal{U}$;

where $\mathcal{X}/\mathcal{U}$ is the co-divide prediction and $\mathcal{K}/\mathcal{N}$ is the ground-truth of the sample, obviously unknown unless the clean labels are available.

A sample in $\mathcal{X}^{false}$ has a loss low enough to be classified as labeled. This can seem counter-intuitive at first sight. However, as noted by 4.4.2, even noisy samples can have a low loss, if the neural network learns the noise present in the data. Conversely, a sample in $\mathcal{U}^{false}$ has a clean label, but a high enough loss to be classified as unlabeled. This can be the result of lack of generalization in that class. Henceforth, let $\mathcal{K}$ be the clean samples, let $\mathcal{N}$ be the noisy samples, let $D^{true} = \mathcal{X}^{true} \cup \mathcal{U}^{true}$ be the true split and let $D^{false} = \mathcal{X}^{false} \cup \mathcal{U}^{false}$ be the false split. Hence, solving the challenge of co-divide is not exclusively about improving the split $\mathcal{X}/\mathcal{U}$ accuracy (where accuracy is

$\#D^{true}/\#D$). The location of the errors is also relevant. Consider, for example, two co-divide approaches both with 90% accuracy; if one (A) has most of its errors in the labeled set $\mathcal{X}$ and the other one (B) has most of its errors in the unlabeled set $\mathcal{U}$, then $\mathcal{X}_A$ and $\mathcal{U}_A$ will be different than $\mathcal{X}_B$ and $\mathcal{U}_B$ leading to a different loss for A and B and consequently different learning. In addition, co-divide is also responsible for computing the probability of each sample being clean $P((x, y) \in \mathcal{K})$, to be used during the MixMatch phase of the algorithm, section 4.5. Therefore, any modification of co-divide must provide a splitting criterion and the probability of each sample being clean.

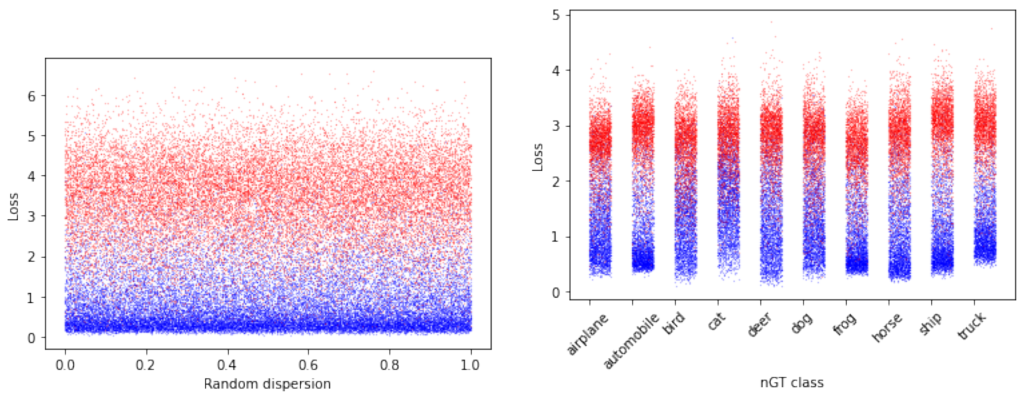

Figure 5.3: Per-sample loss distribution for *DivideMix* with 50% symmetric noise, after warm-up. In red $\mathcal{N}$, the noisy subset and in blue $\mathcal{K}$, the clean subset. a) y axis is per-sample loss, x axis is a random dispersion value intended to avoid points overlapping, intended for readability. b) y axis is per-sample loss, x axis is the class each sample belong to (note it could be noisy and therefore really belong to another class), similarly there is random dispersion value within each class band, intended to avoid points overlapping, intended for readability.

5.3.2 Our Proposal Class-conditional co-divide

This proposal's core assumption is that the per-sample loss distribution of the clean (noisy) samples depends on their class. Figure 5.3, shows that: a) the per-sample loss of the $\mathcal{K}$ and $\mathcal{N}$ follow a different distribution. Therefore, it is reasonable to expect a good split using a GMM on the per-sample loss. Moreover, b) the per-sample loss distribution varies depending on the label of the sample. Indeed, samples labeled as *cat* and samples labeled as *automobile* show significantly different plots. This evidence is leveraged by dividing the training data by its class labels and then fitting a GMM to the per-sample loss for every class independently. Therefore, the distribution of $\mathcal{K}/\mathcal{N}$ is tuned to each class.

Chapter 6

Implementation details

After a detailed explanation of DivideMix and the proposals, this chapter discusses the implementation details, the overall settings and the hardware available for the experimental analysis.

6.1 Baseline NN and self-supervised training

Since the victory of AlexNet at the LSVRC2012 classification contest, the ResNet [12] has become one of the most popular architectures in computer vision. Indeed, the state of the art in LNL uses a ResNet-50, 50 layers deep. However, to do experiments on CIFAR-10 and CIFAR-100, the ResNet-18, 18 layers deep, is enough to benchmark the algorithm's behaviour at a lower computational cost. Therefore, the architecture used in all the experiments in this work is the PreAct ResNet-18. As stated in 2.2.1 and 2.2.2, the architecture of the neural network - its stochastic regularisation technique - limits the uncertainty quantification method choice. Figure 6.1 displays a schema of the ResNet, where it is apparent that there are several Batch-Normalisation layers in each of the 18 residual layers. Furthermore, there is no Dropout in the ResNet. Therefore, the proposed uncertainty quantification technique is the MCBN.

Moreover, it is trained using stochastic gradient descent with a momentum of 0.9, a weight decay of 0.0005, a batch size of 128 and a learning rate of 0.02 reduced by a factor of 10 at epoch 150. However, for self-supervised pretraining, the PreAct Resnet-18 is trained using a SimCLR [37] implementation, trained for 1000 epochs on 4 NVIDIA 2080 Ti GPUs, downloaded from [38]. The experiments done on top of DivideMix follow its 10 epoch warm-up phase. And the experiments done on top of DivideMix C2D follow its 5 epoch warm-up phase.

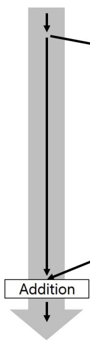

Figure 6.1: Schema of the architecture of the ResNet.

6.2 Uncertainty quantification

Again, the uncertainty quantification technique used is MCBN. Uncertainty is only measured in the co-divide step. First of all, a stochastic forward pass refers to the calculation process in which data (samples) is traversing through all the neurons from first to last layer. Now, the regular flow of DivideMix involves doing a single forward pass for each sample at co-divide. In comparison, MCBN requires T stochastic forward passes in the same stage in order to measure uncertainty. Then, T forward passes lead to T observations and the possibility to quantify the disagreements between them. Implementing MCBN has three keys:

- MCBN's variational inference comes from the fact that the Batch-Normalisation layer -at training time- depends on the variance and mean of the batch. In order to acquire T different observations, the batches must be different (contain different samples) at each forward pass. Otherwise, the mean and variance of the batch would be the same and so would the observations. Hence, the dataloader must have shuffling. Then each batch is composed of randomly selected samples from the dataset each time it is called, thus providing variational inference. The shuffling is implemented using the PyTorch's dataloader class.

- The hyperparameter T: applying MCBN introduces a hyper-parameter, T the number of forward passes to sample at every stage that involves uncertainty. Therefore, uncertainty information is T times as computationally expensive at each stage it is quantified. With the hardware available, it results in approximately $T * 17\ seconds * 2NN * n$, where n is the number of epochs after warm-up. It is clearly very expensive and non-viable for large T. In addition, uncertainty is averaged per class and min-max normalised consequently attenuating its result becoming more robust to imprecision. With this in mind, most the choice is $T = 5$.

- Most importantly, the first forward pass is done with the Batch-Normalisation layers in test mode and the rest $T - 1$ are done in train mode. This way there is a forward pass with the maximum capacity of the network to fit the GMM, and the rest to offer uncertainty quantification ability. Only one forward pass can be done in test mode because the Batch-Normalisation layers only depend on the batch in train mode.

6.3 Uncertainty Aware MixMatch implementation

This proposal has three key considerations. Firstly, note that it introduces a hyper-parameter, $\lambda_{\mathcal{X}}$ to weigh the assistance of the class variance to the labeled loss $\mathcal{L}_{\mathcal{X}}$. Secondly, the class uncertainty is defined as roughly the mean over the samples of the variance of the Softmax probability over the T stochastic forward passes (see equation 5.2). Consequently, for large datasets the result is attenuated, in addition to the min-max normalization, the value of T does not heavily impact this value. More importantly, the class uncertainty is a measure of uncertainty, which is computed following section 6.2. Particularly, in the co-divide step there are T forward passes, the first one -computed in test mode- is used to fit the GMM and to resume co-divide normally, and class uncertainty, ψ is computed with all the forward passes.

6.4 Class-conditional co-divide implementation

Again, class-conditional co-divide splits the training set according to its classes to fit a Gaussian mixture model. It is important to emphasize that the division of the samples by class is done using the ground truth label, potentially noisy, obviously, it is supposed that the clean label is not available. In addition, it introduces the hyper-parameter *p_threshold* used to threshold the probability of the samples belonging to the mixture $\mathcal{K}$. It is a way to interact with the balance of $\mathcal{X}^{false}$ and $\mathcal{U}^{false}$. DivideMix chooses $p\_threshold = 0.5$ and C2D $p\_threshold = 0.03$. These choices are followed unless explicitly said otherwise.

The implementation of the Gaussian mixture model is from SKlearn [39], fitted using the Expectation-Maximization algorithm.

6.5 Code

Fortunately, DivideMix and C2D open sourced their code. Therefore, all the code done in this thesis is built as an extension to the repository published by C2D[1], which is in itself an extension DivideMix[2], and can be found in `https://github.com/aldakata/`

[1]`https://github.com/ContrastToDivide/C2D`

[2]`https://github.com/LiJunnan1992/DivideMix`

`ClassConditionalDivideMix`.

6.5.1 Dependencies

The program is coded in *Python +3.6*. *PyTorch 1.5.1* is the main library used to handle the NNs and the GPU. *NumPy 1.19.2* is used for soft analysis to log interesting properties during training. The implementation of the Gaussian mixture model is imported from *Sklearn*.

6.5.2 Installation

Follow `https://github.com/aldakata/ClassConditionalDivideMix` and install the repository. From the parent root folder of the repo, run *pip install -r requirements.txt* to install all the libraries and dependencies, and it is all set up. To download the CIFAR datasets, a downloadable file can be found in `https://www.cs.toronto.edu/~kriz/cifar.html`

6.5.3 Example

Now, for example,

```
python3 main_cifar.py --r 0.8 --lambda_u 500 --dataset cifar100 --p_threshold 0.03
    --data_path ./cifar-100 --experiment-name simclr_resnet18 --method selfsup
    --net resnet50 --mcbn True --division ccgmm
```

will execute the code with these hyper-parameters. There is the possibility to tune all the mentioned hyper-parameters. Note that if the code is executed without *–mcbn True* then no uncertainty will be quantified at the co-divide step. Indeed, if *–division gmm* and *–mcbn False* the code will run equivalently to DivideMix. *–method selfsup* is to choose C2D pre-training and *–method reg* for vanilla DivideMix.

6.6 Hardware and drivers

All the experiments have been run either on the 8GB NVIDIA GeForce GTX 1080 Mobile, with driver version: 470.86, CUDA: 11.4, PyTorch: 1.5.1 or on the 16GB NVIDIA Tesla P100 available with Google Colab pro, driver version: 460.32.03, CUDA: 11.2, PyTorch: 1.5.1

Chapter 7

Validation.

This chapter provides an empirical analysis of DivideMix and of Uncertainty Based Class-conditional DivideMix. Section 7.1 lays out the common settings of the experiments. The datasets used are detailed in section 7.2. Section 7.3 provides different evaluation metrics for a fair comparison. Section 7.4 shows the results of Uncertainty MixMatch and discusses several experiments about it. Section 7.5 lays out the results of Class-conditional co-divide, and analyzes different experiments about its inner workings. Then, section 7.6 explains the experiments and results of leveraging uncertainty at the co-divide step - a novel proposal that does not improve the model, but offers meaningful insights. Finally, section 7.7 discusses the results.

7.1 General settings

The experiments are done either on top of DivideMix or on top of DivideMix with C2D, because it is interesting to study how the same model behaves with different initial conditions, namely, no pre-training and self-supervised pre-training. Note that DivideMix uses two NNs. Figure 7.1 shows that the behaviour of both NNs is relatively similar, especially as the number of epochs increases. Henceforth, all the experiments are conducted in only one network (always the same one), unless explicitly stated.

Moreover, the performance of DivideMix and DivideMix with C2D given in this chapter are extracted from their respective papers [7, 27]. Also note that the results on DivideMix with C2D are given as a mean and standard deviation. That is because training NNs is a stochastic process, and it is more reliable to average multiple runs than only to trust a single observation. However, the performance of Class-conditional co-divide and Uncertainty MixMatch is extracted from a single run.

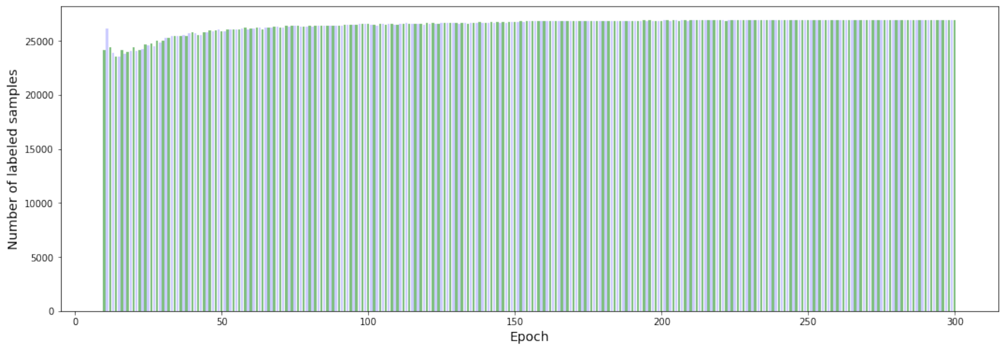

Figure 7.1: CIFAR-10, 50% Symmetric noise, *DivideMix*. Number of labeled samples for each NN. Each color green/blue is a different NN.

7.2 Datasets

All the experiments, regardless of their intent, are conducted on CIFAR-10, a selected few are also tested on CIFAR-100 [40]. Both CIFAR-10 and CIFAR-100 contain 50K training images and 10K testing images of size 32x32. However, CIFAR-10 has 10 classes, i.e. different labels, and CIFAR-100 has 100 classes. Figure 3 displays ten samples for each class from CIFAR-10.

CIFAR-10 and CIFAR-100 are human annotated datasets where there is no noise in their labels. The noise injection process follows previous works [7]. The experiments are done with two types of label noise: symmetric, asymmetric. Symmetric noise is generated by randomly replacing the labels in a percentage of the training data with a random label drawn from a uniform distribution over all labels. Therefore, the new label may be the real one. Working in the 20-90% range of noise, it is guaranteed that the clean label is the most frequent one for each class for any noise level. Indeed, the real number of mislabeled samples is lower than $\frac{1}{N_{classes}}$. Asymmetric noise is designed to mimic the structure of real-world label noise, where classes that are generally similar in appearance are more likely to switch labels, following a scheme proposed by Patrini et al. [41]. The injection of noise to the original labels raises notation needs. The class or label of a sample can be ambiguous, because there is the ground truth label, which is potentially noisy; and the original label, the semantic class of the sample -which is available, because the clean samples of CIFAR-10 and CIFAR-100 are known- it is used to study the model's behaviour such as its predictive accuracy or the accuracy of the co-divide division.

7.3 Evaluation metrics

In deep learning, there are a lot of attributes that dictate the viability of a method, such as model size, training time, number of hyper-parameters. Usually, all the methods are compared over their accuracy, the higher, the better. However, it is unrealistic to fine-tune all the proposals with limited resources, thus potentially discarding a good idea because it was tested with unfitting hyper-parameters. With this in mind, the results shown in this chapter are not to be judged solely on their best accuracy, but rather on the specific problem they are trying to solve.

Accuracy (equation (7.1)) measures the classification capacity of the model on the testing set. Generally, the higher, the better. However, in use cases in which there are more observations of one class than of another, it is not a useful metric. For example, imagine a binary classification task where 99% of the data points belong to class A and 1% of the data points belong to class B. A model that classified the 100% of the data points as class A, its accuracy would be 99%, a great score, but a poor model. In conclusion, it is a metric useful for understanding the quality of the classification on CIFAR-10 and CIFAR-100, because there is an even number of sample of each class, even after the noise injection. However, there is the need for more suitable metrics to representing the quality of L/U split in the co-divide, because the distribution of clean/noisy samples is not bound to be even. A **precise** (see equation (7.2)) model may not find all the positives, but the ones it finds are likely to be correct. A model with high **recall** (equation (7.3)) succeeds in finding all the positive samples, even though it may also wrongly identify some negative samples as positive. Ideally, it would have high precision and high recall. However, in many cases the model can be tweaked to increase precision at the cost of a lower recall or increase the recall at the cost of a lower precision. **F1 score** (equation (7.4)) is the harmonic mean of *precision* and *recall*, an alternative for the common arithmetic mean often used when averaging rates. Therefore, F1 score will be high if both Precision and Recall are, it will be low if both are, and it will be medium if one is high and the other one is low. In addition, the % of $\mathcal{X}^{false}$ and the % of $\mathcal{U}^{false}$ are being used to provide a clearer view of the performance of the analyzed strategies.

$$Accuracy = \frac{\#correct\ predictions}{\#total\ predictions} \tag{7.1}$$

$$Precision = \frac{\#\mathcal{X}^{true}}{\#\mathcal{X}^{true} + \#\mathcal{X}^{false}} \tag{7.2}$$

$$Recall = \frac{\#\mathcal{X}^{true}}{\#\mathcal{X}^{true} + \#\,\mathcal{U}^{false}} \tag{7.3}$$

$$F1\ score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{7.4}$$

7.4 Uncertainty Aware MixMatch

7.4.1 Results

Table 7.4.1 gives a comparison between the performance of the methods on their best epoch on different datasets. Bold marks the best result between each pair, *No modification* and *ψ-MixMatch*. It is important to note that classification on CIFAR-10 is considered easy -there are a lot of annotated samples and only 10 classes. And CIFAR-100 is considered a difficult classification with not enough training data. Then, a good result on CIFAR-100 can signal that the method is more robust to adversarial conditions, and a good result on CIFAR-10 can signal good performance on a sufficient dataset. Good performance overall is desired and easier to interpret. Additionally, know that the accuracy given for ψ-MixMatch is the result of training with the hyper-parameter $\lambda_{\mathcal{X}} = 1$ without any fine-tuning -i.e. no other choices for different hyper-parameters.

Then, Uncertainty MixMatch beats DivideMix in CIFAR-10 for 50% symmetric noise pretty significantly, by 0.38%. For DivideMix with C2D, the proposal does not improve the result. However, it falls into the performance range given. An explanation for its failure on DivideMix with C2D might be that in pre-training, the network learns the underlying patterns in the dataset, but that is not equivalent to learning the semantics of a class. This way in pre-training the network could be learning noise with high confidence that makes the class uncertainty meaningless. It also fails in the CIFAR-100 dataset, that can be because the uncertainty for all the classes is high regardless of the noise level as a consequence of the difficulty of classification on CIFAR-100 and not accounting for the noise injected in a class, or maybe it is consequence of a lack of fine-tuning for the hyper-parameter $\lambda_{\mathcal{X}}$.

Base method	**Modification**	**CIFAR-10**	**CIFAR-100**
		50%	**50%**
DivideMix	Original	94.6	**74.6**
	ψ-MixMatch	**94.98**	74.09
with C2D	Original	**95.32**±0.12	**76.43**±0.25
	ψ-MixMatch	95.26	76.15

Table 7.1: Comparison of the performance -best performing epoch- of the novelty Uncertainty MixMatch (ψ-MixMatch) over different methods and in different datasets. The results of DivideMix and DivideMix with C2D are extracted from their respective papers [7, 27]. Notice that the results on C2D are given as a mean and standard deviation over 5 runs. Note that all the experiments are done in 50% of symmetric noise rate.

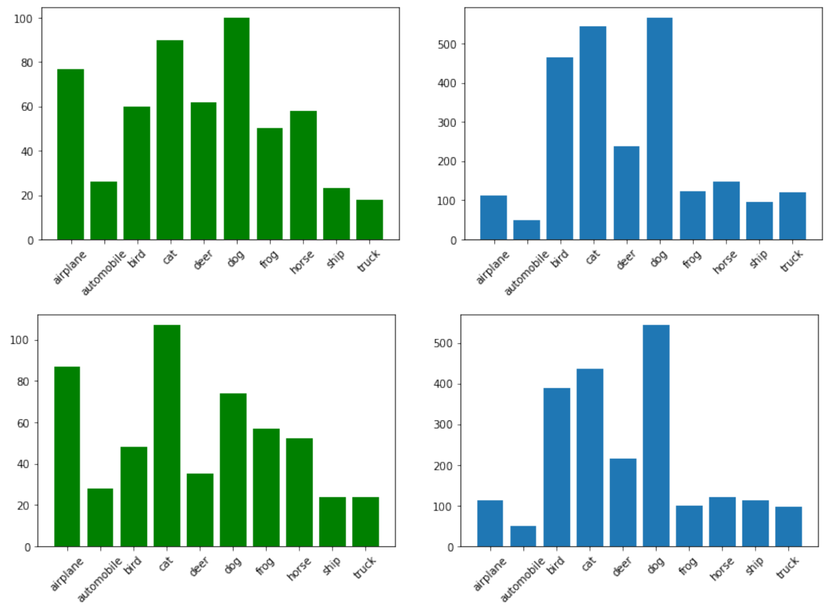

Figure 7.2: The clean class distribution of the classification errors, on CIFAR10 with 50% symmetric noise after the best epoch. In green, the distribution is extracted from the testing dataset and in blue from the training set. The first row is the result of DivideMix, with a standard deviation of 26.5 and 189.4 for the testing and training set respectively. And the second row is the result of DivideMix with ψ-MixMatch, with a standard deviation of 26.8 and 165.0 for the testing and training set respectively.

7.4.2 Analysis of Uncertainty Aware MixMatch

To study the Uncertainty MixMatch, it is important to empirically research its assumptions, namely, that there is class imbalance in the model. Therefore, the class distribution of the predictions of DivideMix at best epoch is studied. Usually, the studies are conducted using the testing data. However, in this case it is also interesting to see how does the neural network predict the training samples, to see if there is any class imbalance there.

Figure 7.2 shows the class frequency of the errors, that is, the number of samples in each class that are wrongly classified by the model, in green the predictions on the testing dataset and in blue the predictions on the training dataset for DivideMix (first row) and DivideMix with ψ-MixMatch. It displays the classes that are easier and harder for the model to learn, in the training and testing set. Training and testing set considered, it is seen that the neither DivideMix nor DivideMix with ψ-MixMatch learn all the classes evenly, dog, cat and airplane appear to be the worst ones. It shows that although both methods have a similar standard distribution -dispersion- on the testing set, on the training set ψ-MixMatch is able to learn the classes more evenly.

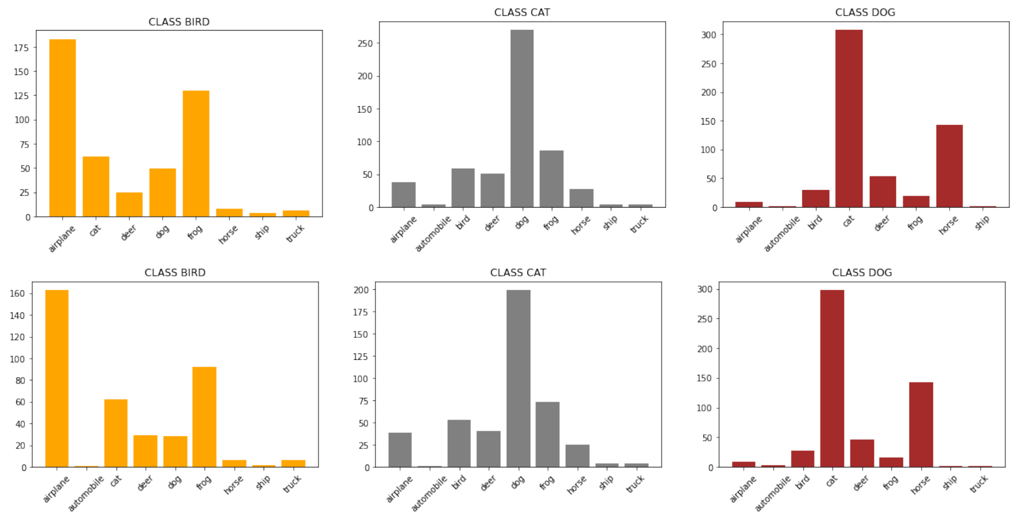

Figure 7.3: For all the samples of the true class *c*, the plot shows the frequency of their predicted class, only ignoring the correctly classified samples. First row is DivideMix and second row is DivideMix with ψ-MixMatch. Analysis done on CIFAR-10 with 50% symmetric noise after the best epoch -with 94.51 accuracy-. In yellow for the class, the *bird*. In gray for *cat*. In red for *dog*. All the figures ignore the correct class frequency.

In addition, figure 7.3 displays, for the samples in three different classes -c-, the frequency of their predicted labels, ignoring the frequency of the actual class. That is, what are the classes that are confused with bird, cat or dog. Again, for DivideMix -first row- and DivideMix with ψ-MixMatch -second row. It is shown that both models are confused between some classes -or semantic subsets. Specifically, it shows that the models confuse the class cat for dog and the other way around; and that the class bird is confused with airplane. However, it is also seen that for the class bird, ψ-MixMatch is able to reduce the "confusion" with airplane and that the error is more distributed where ψ-MixMatch's bird mistakes are with all the other 9 classes but DivideMix's are with 8 classes. Classes dog and cat are relatively similar.

Now, analysing the uncertainty, Figure 7.4 plots the uncertainty of every class as a function of the epoch -sampled every 5 epochs, where each color is a different class. It shows that the uncertainty is low at the beginning, then increases reaching its peak around epoch 100, and it decreases again. The figure to the right shows the same plot, but only for classes cat *red* and automobile *orange*. Recalling Figure 7.2 that shows how the predictive accuracy on the testing and training sets of these two classes is completely different, one would expect their class accuracy to be completely different. Generally, all the classes share the same trend. This is probably because there are a lot of good predictions in each class, attenuating the impact the uncertain classes samples make on the class uncertainty. Unfortunately, the quality of MCBN's uncertainty is not reliable and consequently the analysis of it is challenging.

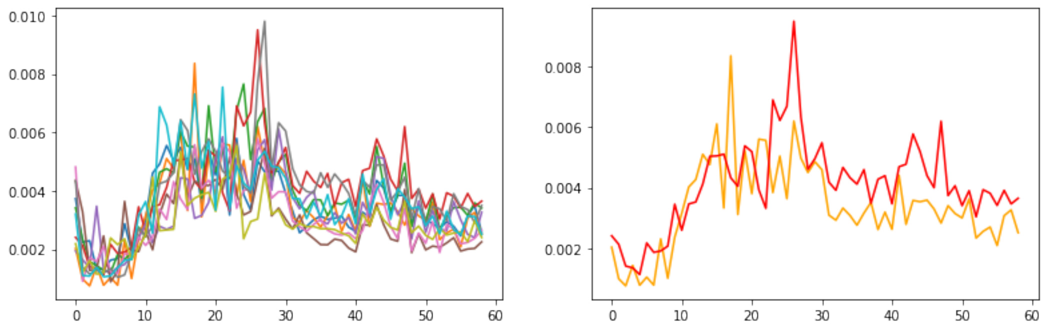

Figure 7.4: This figure shows the class uncertainty - sampled every 5 epochs - before min-max normalisation as a function of the epochs for a NN trained using DivideMix on CIFAR-10 with 50% symmetric noise. Each color is a different class. On the left for all the classes and on the right for the class automobile *orange* and cat *red*.

7.5 Class-conditional Co-divide

7.5.1 Results

Table 7.2 compares the methods' performance on different noise levels and different datasets on their best epoch. Bold marks the best result between each pair, *No modification* and *CC co-divide*. Changing the approach in the Gaussian mixture model could raise the need for a different thresholding hyper-parameter *p_threshold*. However, the accuracy given in this section is the result of running the method with $p\_threshold = 0.5$ for DivideMix and $p\_threshold = 0.03$ for C2D, no fine-tuning has been realized.

Base method	**Modification**	**CIFAR-10**	**CIFAR-100**	
		50%	**20%**	**50%**
DivideMix	Original	94.6	**77.3**	**74.6**
	CC co-divide	**94.79**	76.9	**-**
with C2D	Original	**95.32**±0.12	78.69±0.17	**76.43**±0.25
	CC co-divide	95.40	**79.07**	76.04

Table 7.2: Comparison of the performance -best performing epoch- of the novelty Class-conditional co-divide over different methods, in different datasets and noise rates. The results of DivideMix and DivideMix with C2D are extracted from their respective papers [7, 27]. Notice that the results on C2D are given as a mean and standard deviation over 5 runs. Also note that 20% and 50% stands for the symmetric noise rate present in the experiments on those columns.

For DivideMix with C2D. Class-conditional co-divide beats it in CIFAR-100 with 20% noise rate, where the maximum performance of C2D is 78.88% and the result accomplished is 79.07%, a 0.19% improvement. Such an improvement on the challenging CIFAR-100 is

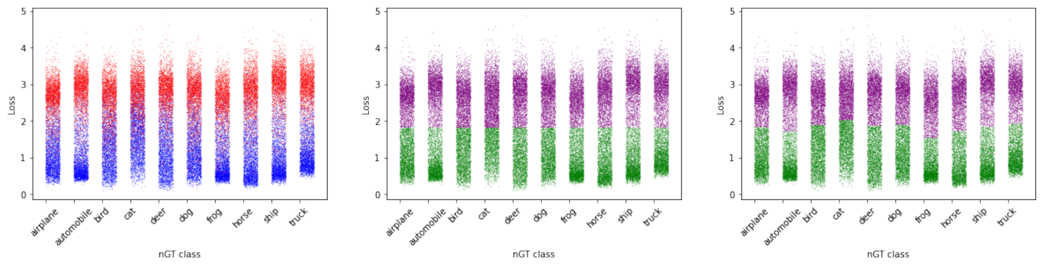

Figure 7.5: Settings: Epoch after warm-up, with *DivideMix*, 50% noise, p_threshold=0.5. Blue is $\mathcal{K}$, red is $\mathcal{N}$, green is predicted $\mathcal{X}$ purple is predicted $\mathcal{U}$. Left to right: Division $\mathcal{K}/\mathcal{N}$; division $\mathcal{L}/\mathcal{U}$ proposed by GMM; division $\mathcal{L}/\mathcal{U}$ proposed by CCGMM.

encouraging.

For CIFAR-10 with 50% noise rate, the performance falls inside the range of C2D and with only one run it cannot be known if the performance of Class-conditional co-divide beats DivideMix with C2D in these settings.

7.5.2 Analysis of Class-conditional Co-divide

Analysis of the per-sample loss class distribution on DivideMix is important in order to understand the viability of Class-conditional co-divide. Figure 7.5 displays the per-sample loss distribution of each class, blue (red) is for $\mathcal{K}$ ($\mathcal{N}$) - the clean class - green (purple) is for $\mathcal{X}$ ($\mathcal{U}$) - the predicted class using GMM and Class-conditional GMM (CCGMM). a) confirms that each class has a different loss distribution. b) shows the result of the DivideMix co-divide split prediction, where all the classes share the same horizontal line because the GMM was fit with the whole dataset, therefore, some classes are better fitted than others which could further increase the class imbalance. c) shows the result of class-conditional DivideMix with the same threshold. Each class has a different decision boundary. Figure 4 shows the per-sample loss distribution on epoch 280. Notice that the per-sample loss of the noisy samples is still different between the classes. However, the class-conditional division does not seem to be very different from the original GMM. In conclusion, it is clear that there exists class difference in the co-divide step and that DivideMix not aware of it.

Table 7.3 compares the accuracy, F1 score, precision, % of $\mathcal{X}^{false}$ and % of $\mathcal{U}^{true}$ of the DivideMix's co-divide (GMM) and of the proposed CCGMM on the method DivideMix trained on CIFAR-10 with 50% symmetric noise after warm-up. It shows that CCGMM has a slightly higher F1 score and accuracy for a slightly lower precision for the same threshold. However, it is also interesting to study the distribution of these errors across the classes. Figure 7.6 shows the distribution of the successful predictions of each class. On the left, for the method GMM and on the right for the method CCGMM. The total

threshold	Method	Accuracy	F1 score	Precision	Recall	$\%\mathcal{X}^{false}$	$\%\mathcal{U}^{false}$
0.5	GMM	0.867	0.728	6.19	0.825	0.037	0.096
	CCGMM	0.872	0.733	5.716	0.841	0.04	0.088
0.3	GMM	0.876	0.735	5.012	0.861	0.047	0.076
	CCGMM	0.879	0.736	4.663	0.874	0.052	0.069

Table 7.3: Co-divide comparison on the method DivideMix on CIFAR-10 with 50% symmetric noise trained after warm-up. Method is the approach used to compute the probability of the samples belonging to $\mathcal{K}$ (GMM or CCGMM).

Method	Accuracy	F1 score	Precision	Recall	$\%\mathcal{X}^{false}$	$\%\mathcal{U}^{false}$
GMM	**0.957**	**0.914**	**52.008**	**0.93**	0.005	0.039
CCGMM	0.956	0.913	51.536	0.929	0.005	0.039

Table 7.4: Co-divide comparison on the method DivideMix on CIFAR-10 with 50% symmetric noise trained to epoch 150 with *p_threshold=0.5*. Method is the approach used to compute the probability of the samples belonging to $\mathcal{K}$ (GMM or CCGMM).

number of successful predictions is higher in CCGMM and the standard deviation is lower, allowing for more class-balanced training. In addition, Table 7.4 compares the two division methods on epoch 150.

7.6 Uncertainty based co-divide

This section introduces an approach that leverages uncertainty at the co-divide step. Although this proposal does not improve DivideMix it is interesting and enlightening.

7.6.1 Proposal for Uncertainty Based Co-divide

This proposal is based on the idea of cooperation between the T stochastic forward passes of the per-sample loss. Recall figure 5.2 that displays the distribution of per-sample loss and per-sample entropy. For a sample, the higher the entropy is, the more the T observations of its loss differ.

At first glance, in Figure 5.2, the entropy does not add information to ease the split classification. However, the samples with low entropy are easily separable, and the samples with high entropy are those in the fringe of the classification decision, the hardest to split. Indeed, the uncertainty is adding information about the confidence of the loss of a sample rather than its cleanliness. With this in mind, different policies to leverage uncertainty rise the following proposals:

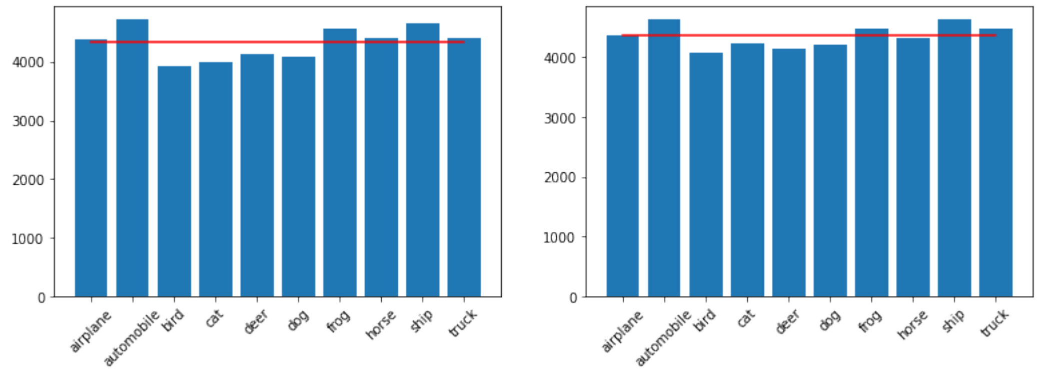

Figure 7.6: The number of successful divisions for each different class. The red line is the mean. On the left for the DivideMix's co-divide, with **mean=4330.0** and **standard deviation=265.33**. On the right for the Class-conditional co-divide, with **mean=4352.5, standard deviation=184.69**.

And-GMM

The And-GMM looks to exploit the different observations independently, aiming to minimize the $\mathcal{X}^{false}$ set at the expense of a bigger $\mathcal{U}^{false}$. Its base thought is the fact that the higher the entropy of a sample is, the more the different observations of its loss differ. Therefore, for the samples in the fringe, there might be an observation that classifies it as L and another as U. In detail, this proposal fits a GMM for each T, predicts the split $\mathcal{X}$ / $\mathcal{U}$ for each T probability, resulting in T predictions for every sample. Finally, combining the T predictions with a *logical and* operation, where a sample is predicted as clean only if all the observations have predicted it is clean, and noisy - otherwise. In addition, $P((x,y) \in \mathcal{K})$ is the mean of the T probabilities. Precisely,

$$\forall (x,y) \ \in \ D,$$

$$P((x,y) \in \mathcal{K}) = GMM(\frac{1}{T}\sum_{t=1}^{T} \ell(h_\theta(x)_t, y))$$

$$(x,y,\tilde{y}) \ \in \begin{cases} \mathcal{L} \quad if \ GMM(\ell(h_\theta(x)_t, y)) \ >= \ p\_threshold \quad \forall t = 1, \dots, T \\ \\ \mathcal{U} \quad if \ GMM(\ell(h_\theta(x)_t, y)) \ < \ p\_threshold \quad \exists t = 1, \dots, T \end{cases}$$

where $GMM(\cdot,\cdot)$ maps a vector of per-sample losses to the probability of belonging to the mixture with the lowest mean -the clean mixture.

Or-GMM

Similarly, Or-GMM aims to minimize the $\mathcal{U}^{false}$ set at the expense of a bigger $\mathcal{X}^{false}$. This proposal fits a GMM for each T, afterwards predicting the split $\mathcal{X}$ / $\mathcal{U}$ for each T. Finally, it combines the T predictions with a *logical or* operation, where a sample is predicted as clean if any of the observations has predicted it is clean, and noisy - otherwise. In addition, $P((x,y) \in \mathcal{K})$ is the mean of the T probabilities. Precisely,

$$\forall (x, y, \tilde{y}) \in D,$$

$$P((x,y) \in \mathcal{K}) = GMM(\frac{1}{T}\sum_{t=1}^{T} \ell(h_\theta(x)_t, y))$$

$$(x,y) \in \begin{cases} \mathcal{L} \quad if\ GMM(\ell(h_\theta(x)_t, y)) >= p\_threshold \quad \exists t = 1, \ldots, T \\ \\ \mathcal{U} \quad if\ GMM(\ell(h_\theta(x)_t, y)) < p\_threshold \quad \forall t = 1, \ldots, T \end{cases}$$

In conclusion, uncertainty at co-divide allows for the introduction of a new hyper-parameter, *division_method* with 2 different possibilities. Now, the restrictiveness of co-divide can be tuned with the hyper-parameter *p_threshold* and the policy method. Notice, that $P((x,y) \in \mathcal{K})$ for the different approaches, the three of them use the mean of the T probabilities, because it is a better estimator than any other studied. Finally, these policies are compatible with both co-divide's GMM and with the CCGMM.

7.6.2 Analysis of Uncertainty Based Co-divide

First of all, the resulting predictive accuracy of this proposal on DivideMix and DivideMix with C2D is around 1.5% worse for CIFAR-10. Figure 1 shows the result of the co-divide division with this different policies. In blue the unlabeled noisy set $\mathcal{U}^{true}$, in green the labeled clean set $\mathcal{X}^{true}$, in orange the unlabeled clean set $\mathcal{U}^{false}$ and in red the labeled noisy set $\mathcal{X}^{false}$. The first column uses DivideMix's Gaussian mixture model, and the second column uses the class-conditional variant. First row is without leveraging uncertainty, second row is with the AND policy and third row is with the OR policy. It is seen that the AND method reduces the red dots - the false clean - and that OR minimizes the orange dots - the false noisy. Table 7.5 compares the results. It is clear that OR provides a policy that maximizes the recall while AND maximizes the precision. These results look promising. However, recall that in training mode the Batch-Normalisation layers do not use their learned parameters and use the mini-batch statistics instead. During the first epochs the difference in performance is minimum, but as the epochs increase the difference in performance between the model with the BN layers in testing mode and the BN layers in training mode increases. Table 7.6 shows this phenomenon, comparing the results of the co-divide for the model with the Batch-Normalisation layers in testing mode and in training mode for the original co-divide -GMM- and the class-conditional co-divide -CCGMM. It is seen that the model in testing mode outperforms the training mode for all the metrics. Consequently, the $t = 2, \ldots, T$ have a worse "opinion" -comparatively- as the epoch increases and their influence impairs the $t = 1$ forward pas -the one done in testing mode- thus resulting in a lower performance.

Method	**Policy**	**Accuracy**	**F1 score**	**Precision**	**Recall**	$\%\mathcal{X}^{false}$	$\%\mathcal{U}^{false}$
GMM	Original	0.862	0.724	7.03	0.807	0.032	0.106
CCGMM	Original	0.881	**0.753**	7.183	0.842	0.032	0.087
GMM	AND	0.836	0.69	**10.83**	0.737	**0.019**	0.145
CCGMM	AND	0.852	0.714	9.72	0.77	0.022	0.127
GMM	OR	0.884	0.742	4.581	0.886	0.053	0.063
CCGMM	OR	**0.889**	0.748	4.459	**0.899**	0.055	**0.056**

Table 7.5: Comparison between the OR & AND policies with GMM and CCGMM.

BN Mode	**Method**	**Accuracy**	**F1 score**	**Precision**	**Recall**	$\%\mathcal{X}^{false}$	$\%\mathcal{U}^{false}$
TEST	GMM	**0.957**	**0.914**	**52.008**	**0.93**	0.005	0.039
	CCGMM	0.956	0.913	51.536	0.929	0.005	0.039
TRAIN	GMM	0.955	0.91	51.407	0.927	0.005	0.04
	CCGMM	0.954	0.91	51.593	0.926	0.005	0.041

Table 7.6: Comparison between the model with BN in training mode and in testing mode.

7.7 Discussions

Although the proposals do not improve the method all over the settings, the results of these modifications are encouraging and point that indeed a class-conditional treatment of some sort is sound. Table 8.1.3 shows all the results. In conclusion, ψ-MixMatch is able to distribute the final error more evenly, but the meaning of the per-class uncertainty is opaque and difficult to decipher. Class-conditional co-divide is successful in doing a more class-conscious co-divide step. Not only achieving better accuracy during the early epochs, but also distributing the errors of the division more evenly over the classes. In addition, there has been an experiment that used both class-conditional co-divide and ψ-MixMatch, and it performed worse on DivideMix with C2D than the baseline, and it achieved a similar result than original DivideMix.

Chapter 8

Conclusions

This thesis' hypothesis was that the current state of the art on LNL, namely DivideMix, is unaware of the underlying class-conditional behaviour of the samples. Consequently, the objective was to study this hypothesis and to research an approach that leverages an uncertainty estimation.

With our experiments, we have been able to showcase this class-conditional behaviour. We have also implemented an uncertainty quantification technique that requires no changes in the architecture. And we have presented that the per-sample uncertainty is correlated with the confidence of the co-divide split. That led us to the implementation of two improvements on DivideMix:

- **ψ-MixMatch** that leverages per-class uncertainty estimates to correct the per-sample loss. It results in an improvement of the Top-1 accuracy by 0.38% on CIFAR-10 with 50% symmetrical noise.

- **Class-conditional co-divide** that models the per-sample loss using a Gaussian mixture model for each class individually. It results in an improvement of the Top-1 accuracy by 0.38% on CIFAR-10 with 50% symmetrical noise resulting in slight improvements over different settings.

We have seen that the proposals are more aware of the class-conditional behaviour than the base method. In addition, we have researched other approaches and analysed their failure conditions, pointing to a limitation in the uncertainty quantification technique.

We view this work as a starting point to develop a class-conditional and uncertainty-aware approaches on LNL.

8.1 Future work and other ideas

The main challenge in this thesis has been to research DivideMix's behaviour and to come up with promising ideas. In the research process, it has been equally important to be open to distractions -in the form of new ideas- than to stick with the current plan. This chapter is a collection of both the next steps of this work and of new ideas that have sparked along the way.

8.1.1 Validation on new and bigger datasets

The expensive computational cost related to training a deep convolutional neural network has limited the accuracy presented in this work. The next steps are to fine-tune the proposals to know their accuracy with better precision. Moreover, CIFAR-10 AND CIFAR-100 are valuable for their simplicity and broad use. However, the most important benchmark datasets in LNL is Clothing1M, any promising idea has to show its worth in this dataset. In this regard, it follows to study the underlying class-conditional behaviour of Clothing1M.

8.1.2 New Proposal for Uncertainty-aware LNL

It is key to study the per-class uncertainty further to de-obfuscate its meaning. Regarding ψ-MixMatch, the proposed method uses the per-class uncertainty of one Neural Network to weight the loss of the other Neural Network, following the general flow of DivideMix. However, per-class uncertainty is not a feature of the dataset, but a feature of the neural network and maybe ψ-MixMatch would work better if the loss of a Neural Network was weighted with its own per-class uncertainty.

In addition, the Class-conditional co-divide could use an automatic threshold for each class individually.

8.1.3 Recursive models for LNL

What happens when a model achieves better accuracy than its % of noise? The state of the art in LNL seems to agree that for symmetric noise, the lower the noise in a dataset, the higher its accuracy will be.

Recursive learning's core idea is to exploit this opportunity. To do so, it could run DivideMix normally until it converges, then relabels the training set with the model at the best epoch (or last). At this point the training set has lower % of noise but, unfortunately, the noise is asymmetric, which is a way harder challenge in LNL. The next step is to: randomize the label of each sample with a probability according to its loss (a GMM) and retrain with the relabeled training set. If a sample has high uncertainty for every possible class, it should be classified as unlabeled directly. In addition, the training set could be expanded using the classification capability of this network, and then undergo retraining with the expanded training set.

Bibliography

[1] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. arXiv:1811.00982, 2018.

[2] Wen Li, Limin Wang, Wei Li, Eirikur Agustsson, and Luc Van Gool. Webvision database: Visual learning and understanding from web data. arXiv:1708.02862, 2017.

[3] Dhruv Mahajan, Ross B. Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In ECCV, pp. 185â201, 2018.

[4] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In ICLR, 2017.

[5] Alex Graves. Practical variational inference for neural networks. In Advances in neural information processing systems, pages 2348â2356, 2011.

[6] Yarin Gal, Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning, pages 1050â1059, 2016.

[7] Junnan Li and Richard Socher and Steven C.H. Hoi, DivideMix: Learning with Noisy Labels as Semi-supervised Learning. International Conference on Learning Representations, 2020.

[8] Kunihiko Fukishima. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. Biological Cybernetics 36, 1980.

[9] Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton.ImageNet Classification with Deep Convolutional Neural Networksâ. Neural Information Processing Systems 25, 2012.

[10] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556, 2014.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich. Going Deeper with Convolutions, 2014.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Identity mappings in deep residual networks. ECCV, pp. 630645, 2016.

[13] Ioffe, Sergey and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015.

[14] Arash Vahdat. Toward robustness against label noise in training deep discriminative neural networks. NIPS, 2017.

[15] Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo, and Li-Jia Li. Learning from noisy labels with distillation. In ICCV, pp. 1928â1936, 2017.

[16] Zhang, BeNgio, Hardt, Recht and Vinyals. Understanding deep learning requires rethinking generalization. 2016.

[17] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning datadriven curriculum for very deep neural networks on corrupted labels. ICML, pp. 2309â2318, 2018.

[18] Eric Arazo, Diego Ortego, Paul Albert, Noel E. OâConnor, and Kevin McGuinness, Unsupervised label noise modeling and loss correction. ICML, pp. 312321, 2019.

[19] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. NeurIPS, pp. 8536â8546, 2018.

[20] Wenkai Chen, Chuang Zhu and Yi Chen. Sample Prior Guided Robust Model Learning to Suppress Noisy Labels. arXiv:2112.01197v2, 2021.

[21] Ian Goodfellow, Yoshua Bengio, Aaron Courville Deep learning `https://www.deeplearningbook.org/` MIT Press, 2016.

[22] Nitish Srivastava, et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In Journal of Machine Learning Research 15.56, pp. 1929â1958, 2014.

[23] Sergios Theodoridis. Machine Learning || Learning in Parametric Modeling: Basic Concepts and Directions, 67120, 2020.

[24] Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning. Adaptative Computation and Machine Learning. In MIT Pess, Chapter 5.6: Bayesian Statistic, 2016.

[25] Loic Le Folgoc, Vasileios Baltatzis, Sujal Desai, Anand Devaraj, Sam Ellis, Octavio E. Martinez Manzanera, Arjun Nair, Huaqi Qiu1, Julia Schnabel, and Ben Glocker. Is MC Dropout Bayesian?

[26] Mattias Teye, Hossein Azizpour and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. In volume 80 of Proceedings of Machine Learning Research, pages 4907â4916, 2018.

[27] Evgenii Zheltonozhskii, Chaim Baskin, Avi Mendelson, Alex M. Bronstein, Or Litany, Contrast to Divide: Self-Supervised Pre-Training for Learning with Noisy Labels. ArXiv 2021.

[28] Devansh Arpit, Stanislaw Jastrzkebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron C. Courville, Yoshua Bengio, and Simon Lacoste-Julien, *A closer look at memorization in deep networks*. ICML, pp. 233241, 2017.

[29] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang, and Masashi Sugiyama, Co-teaching: Robust training of deep neural networks with extremely noisy labels. NeurIPS, pp. 85368546, 2018.

[30] Pengfei Chen, Benben Liao, Guangyong Chen, and Shengyu Zhang, Understanding and utilizing deep neural networks trained with noisy labels. ICML, pp. 10621070, 2019.

[31] Antti Tarvainen and Harri Valpola, Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. NIPS, pp. 11951204, 2017.

[32] Haim H. Permuter, Joseph M. Francos, and Ian Jermyn, A study of gaussian mixture models of color and texture features for image classification and segmentation. Pattern Recognition, 39(4): 695706, 2006.

[33] Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton, Regularizing neural networks by penalizing confident output distributions. ICLR Workshop, 2017.

[34] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton, Big self-supervised models are strong semi-supervised learners. arXiv preprint arXiv:2006.10029, 2020.

[35] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton, Big self-supervised models are strong semi-supervised learners. arXiv preprint arXiv:2006.10029, 2020.

[36] David Berthelot Nicholas Carlini, Ian Goodfellow, Avital Oliver, Nicolas Papernot, Colin Raffel, MixMatch: A Holistic Approach to Semi-Supervised Learning. NeurIPS 2019.

[37] Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton, A Simple Framework for Contrastive Learning of Visual Representations. arXiv preprint arXiv:2002.05709, 2020.

[38] HobbitLong, `https://github.com/HobbitLong/SupContrast` PyTorch implementation of "Supervised Contrastive Learning" (and SimCLR incidentally). Github, 2021.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12 28252830, 2011.

[40] Alex Krizhevsky, Learning Multiple Layers of Features from Tiny Images. 2009.

[41] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. CVPR, pp. 22332241, 2017.

Appendix

Base method	Modification	CIFAR-10	CIFAR-100	
		50%	**20%**	**50%**
DivideMix	Original	94.6	**77.3**	**74.6**
	ψ-MixMatch	**94.98**	-	74.09
	CC co-divide	94.79	76.9	-
with C2D	Original	**95.32**$\pm$0.12	78.69$\pm$0.17	**76.43**$\pm$0.25
	ψ-MixMatch	95.26	-	76.15
	CC co-divide	95.40	**79.07**	76.04

Table 1: Comparison of the performance -best performing epoch- of the all the proposals with DivideMix and DivideMix with C2D, in different datasets and noise rates. The results of DivideMix and DivideMix with C2D are extracted from their respective papers [7], [27]. Notice that the results on C2D are given as a mean and standard deviation over 5 runs. Also note that 20% and 50% stands for the symmetric noise rate present in the experiments on those columns.

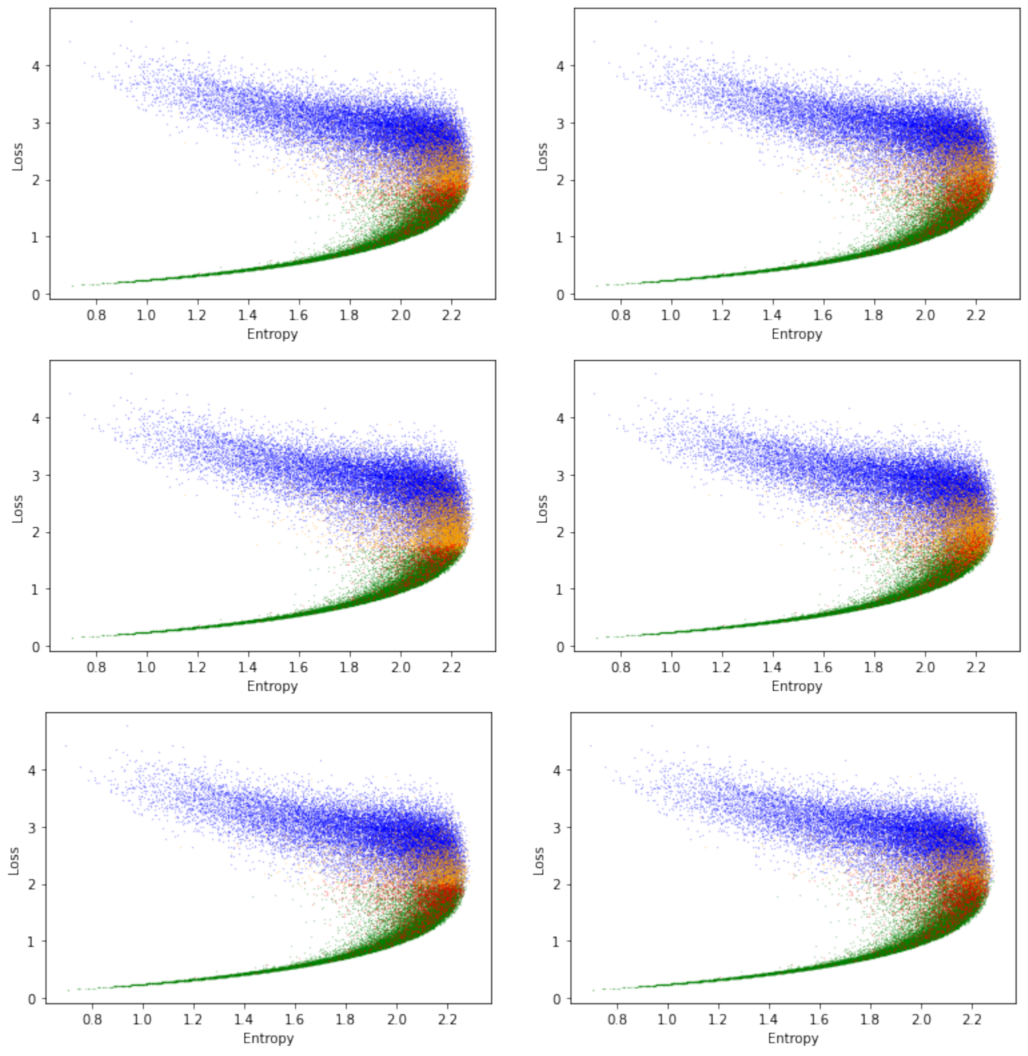

Figure 1: CIFAR-10, DivideMix, 50% symmetric noise, *p_threshold=0.5*. Plot of the Mean of the Loss over T x Entropy green for $\mathcal{X}^{true}$ blue $\mathcal{U}^{true}$ red for $\mathcal{X}^{false}$ and orange for $\mathcal{U}^{false}$. a) GMM, b) CCGMM, c)GMM AND, d) CCGMM AND, e)GMM OR, f)CCGMM OR

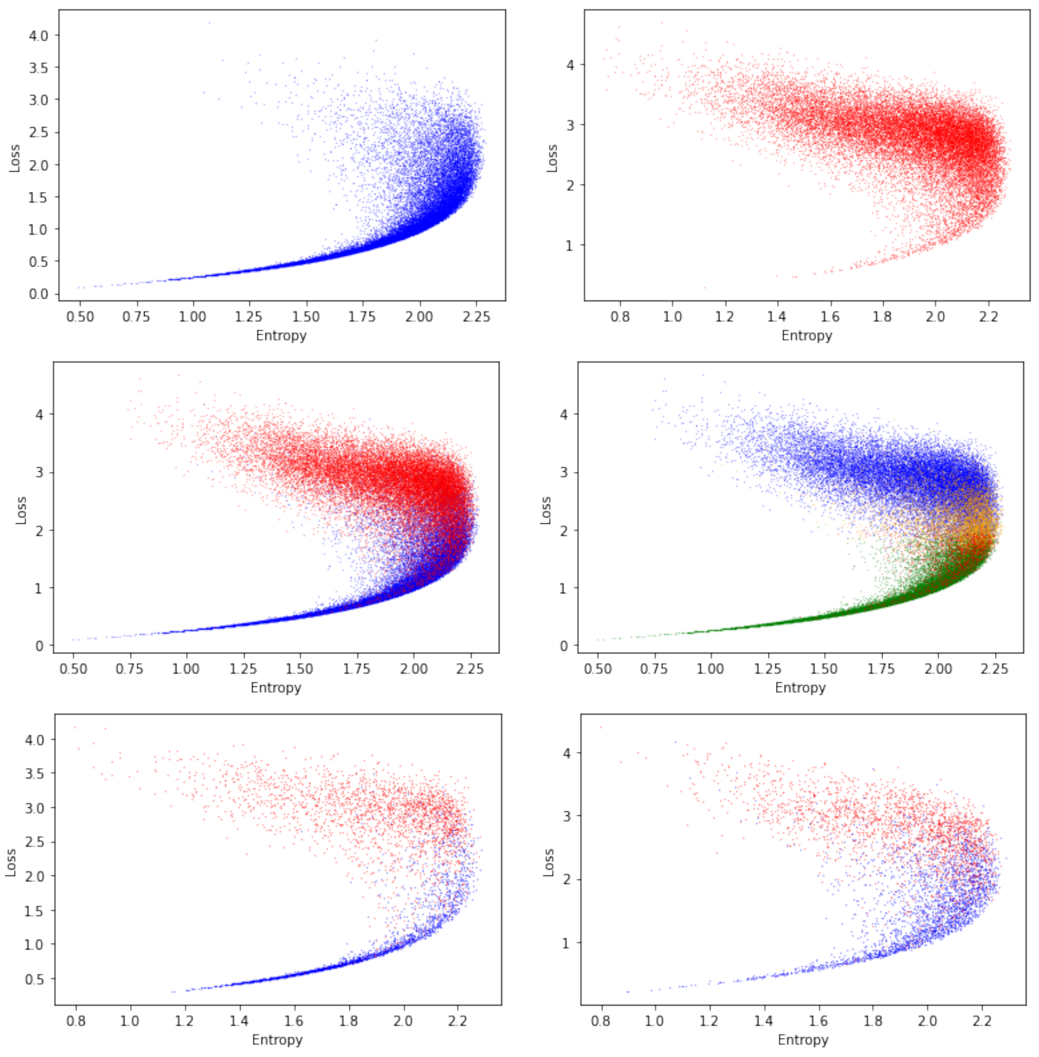

Figure 2: Analysis is done with the NN trained after warm-up and with $T = 50$ stochastic forward passes. From top to bottom, left to right: In blue $\mathcal{K}$ plotting the mean per-sample loss over T by the per-sample entropy of the clean samples. In red, plotting the mean per-sample loss over T by the per-sample entropy of the noisy samples. The mean per-sample loss over T by the per-sample entropy of all the samples, blue for $\mathcal{K}$ and red for $\mathcal{N}$. The mean per-sample loss over T by the per-sample entropy green for $\mathcal{X}^{true}$, blue for $\mathcal{U}^{true}$, red for $\mathcal{X}^{false}$ and orange for $\mathcal{U}^{false}$. The two last figures are the distribution of the mean per-sample loss over T by the per-sample entropy but only the class *automobile* (left) and *cat* (right). Note that the dots are opaque and that there may be blue dots under red ones.

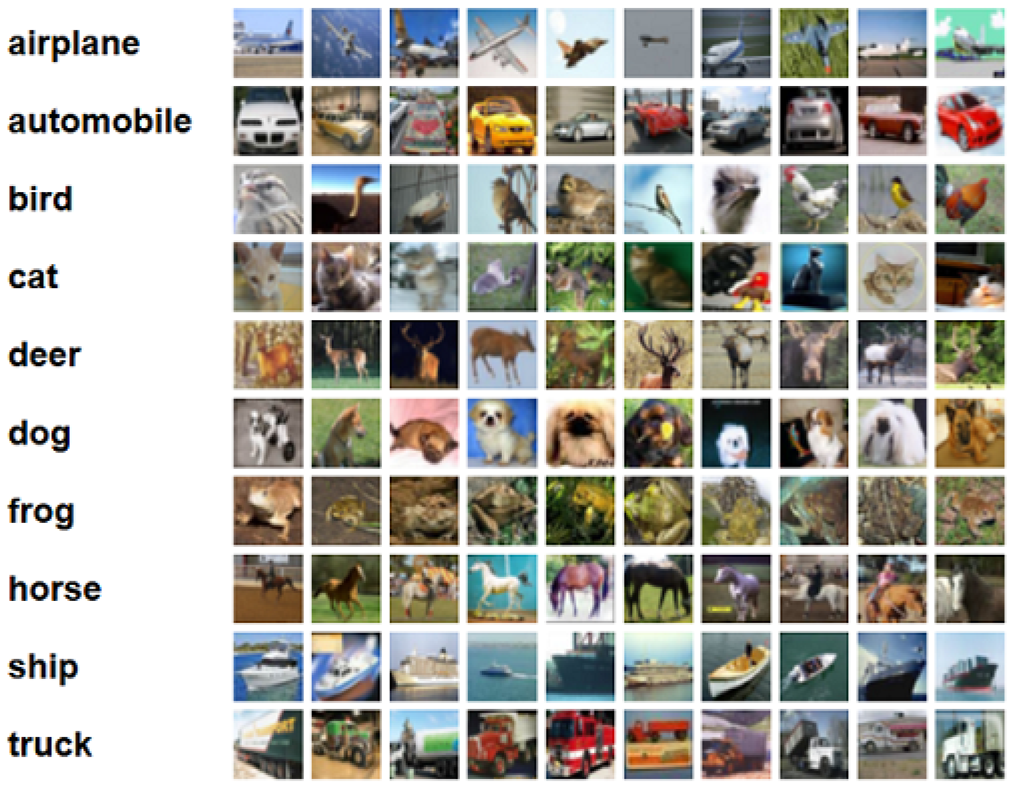

Figure 3: 10 sample images for each class from CIFAR-10.

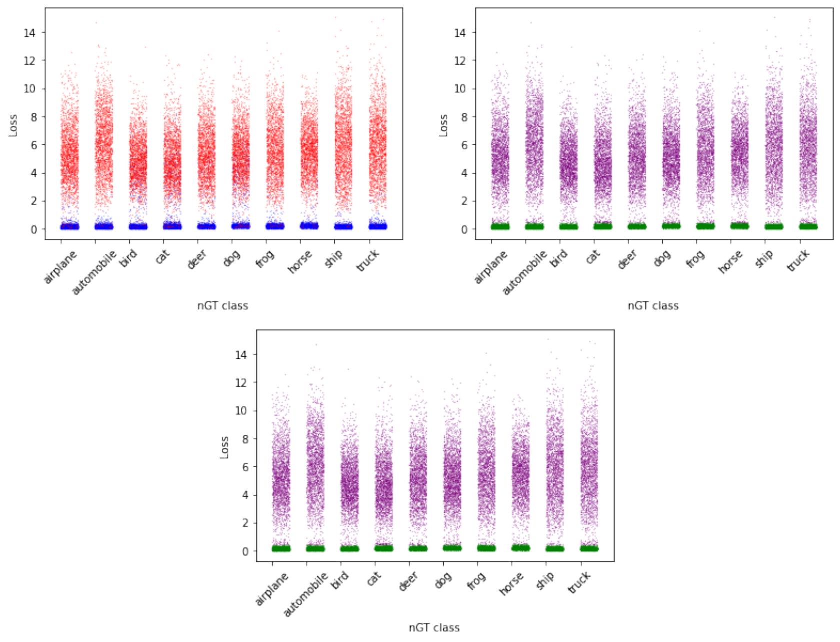

Figure 4: Settings: Epoch 280, with DivideMix on 50% symmetric noise, p_threshold=0.5. Blue is $\mathcal{K}$, red is $\mathcal{N}$, green is predicted $\mathcal{X}$ purple is predicted $\mathcal{U}$. a)Division $\mathcal{K}/\mathcal{N}$ b) Division $\mathcal{L}/\mathcal{U}$ proposed by GMM. c) Division $\mathcal{L}/\mathcal{U}$ proposed by CCGMM.