COMP5328 Assignment 2

Coordinator:
Dr Tongliang Liu
Tutor:
Nicholas James
Group members:
Pierre Segonne, pseg5445, SID: 490547607
Thorben Pieper, tpie7343, SID: 490559851

Abstract

Label noise has become a largely discussed topic due to the apparition of fast and cheap non-expert labelling platforms. This paper aims to explore how it can be modelled, how it might affect the performance of different classification methods and how robust techniques can reduce its deleterous effects. Three different techniques are detailed, namely, the logistic regression under inverse flip rates, the label noise robust logistic regression and the label noise robust neural network. Their implementations' performance is compared on two different datasets, with varying levels of label noise contamination, to evaluate their robustness.

1 Introduction

This last decade's advancements in the field of machine learning can be partly attributed to the gathering of large scale datasets, such as the ImageNet dataset [1] detailed in [Den+09] which today contains more than 14 million labeled images. These advancements include large scale image classification [KSH12], semantic segmentation [Zha+17], object recognition [Ser+13], language processing [Yan+19], medical analysis [Kam+17] etc. In turn, the interest for machine learning raised by such advancements led to an increased demand of large labeled datasets. Labeling datasets cannot in practice been conducted by those who create methods that learn from these datasets, notably because it is very time consuming. Consequently, they need to rely on either, if the classification task requires prior knowledge, experts (e.g doctors for medical analysis) or non-experts that can be financially rewarded by classification task[2] or that do it voluntarily[3]. Whoever conducts the labeling of the dataset is prone to some labelling errors. The causes are multiple and include, as detailed in [FV13]:

- Insufficient information provided for the labelling task. [BF99]
- Subjectivity of the labelling task. [MBN06]
- Errors from the labeller himself. [Hic96]

These labeling errors can be detrimental to the performance of all methods tested on the contaminated datasets and be considered as noise [Hic96]. [ZW04] even shows that this label noise is more harmful than feature noise.

[1] http://www.image-net.org/
[2] See for example Amazon's mechanical turk: https://www.mturk.com/
[3] See for example https://apps.sentinel-hub.com/classificationApp/

Several assumptions can be made to model label noise. First, it can be assumed that the label noise is completely random, with a fixed probability. This corresponds to a case where labelling errors are as likely to occur for any observation (e.g if an expert classifies entries with a uniform small rate of error). Let's note x an observation from the dataset, y its true associated label and $\tilde{y}$ its noisy label, then the random label noise can be expressed in a multiclass setting, with c classes, as:

$$\forall (i,j) \in \{0,...,c-1\}^2 \text{ s.t } i \neq j,\ P(\tilde{y}=i|y=j,x) = \rho \tag{1}$$

and it follows that

$$\forall i \in \{0,...,c-1\},\ P(\tilde{y}=i|y=i,x) = 1-(c-1)\rho \tag{2}$$

Secondly, the label noise can be assumed to be class dependant. This corresponds to a case where the likeliness of errors depends on the true class label (e.g if a non-expert has to label images of dog breeds, he might repeatedly mistake alaskan malamutes for huskies). It can be expressed as:

$$\forall (i,j) \in \{0,...,c-1\}^2 \text{ s.t } i \neq j,\ P(\tilde{y}=i|y=j,x) = P(\tilde{y}=i|y=j) \tag{3}$$

Finally, the label noise can be assumed to be both class and instance dependant. This corresponds to the most realistic case, where both some classes and instances are more difficult to classify and thus more prone to errors. In that case, the noise probability is not reducible.

The following study will explore several robust methods to handle class dependant label noise. At first, related work will be examined, before describing the theory behind the methods that have been implemented in the scope of this study. These include two noise robust adjustments to the common Multinomial Logistic Regression and a noise robust Neural Network. Afterwards, the results of these implementations, run on three different data sets corrupted with label noise, and the respective comparison of the noise robust classifiers and their original counterparts will be given.

2 Related Works

2.1 Ensemble Methods

Ensemble methods aim to enhance the overall performance of a model (e.g a classifier) by combining the outputs of several instances of that model trained on different subsets of the dataset. Such combination is likely to provide several benefits, such as a lesser variance for the final model - which allows the use of model instances with greater variance and thus greater ability to tell observations apart in a classification setting. - or the appearance of "specialists" instances of the model, which will be able to assign observations that are hard to tell apart.

The two main ensemble methods families are bagging and boosting.

Bagging as originated in [DS79] and later developed in [Bre96] consists in generating from the original dataset $\mathcal{D}$ of size N, t random new datasets $\mathcal{D}_1, ..., \mathcal{D}_t$ of size N, in which a certain number of observations N' from the original dataset are omitted and replaced with non-omitted observations. t Models will then be trained respectively on the datasets and in the case of multiclass classification, the final prediction for any observation will be decided by majority voting i.e for $f(x)_t$ being the output of the instance t of the model for observation $x \in \mathcal{D}$, $f(x) = \arg\max_{c \in \{1,...,C\}} \{\text{Number of classifiers with output} f(x)_t = c\}$

Boosting extends bagging; instead of randomly selecting the points which will be omitted and replaced, it will assign to every observation a probability to be selected in each subset. By assigning higher probability to the observations that are the hardest to classify -Which corresponds to the specific AdaBoost method, very popular since its introduction in [FS97]-, i.e that are closest to the respective decision boundaries, boosting is able to fit closely non linear complex decision boundaries.

In the presence of label noise, the incorrectly labeled observation will be located on the wrong side of the true decision boundaries and will therefore introduce a deformation of that boundary. With bagging, the final decision boundary will not overfit these deformations and will therefore hopefully have a greater accuracy than a single model trained on the whole dataset. Boosting on the other hand, if applied with the previous probability rule, might conversely affect the accuracy by trying to fit closely the decision boundary around observations with noisy labels. This has been shown notably in e.g [Die00].

Nevertheless, some alternatives to adaboost have been devised to increase the robustness of boosting techniques to noisy labels. Notably, in [Ser03], R. Servedio defines a *SmoothBoost* that generates smooth distributions of observations and therefore avoids skewing the respective data distributions towards noisy labels, as AdaBoost does; in [FHT+00], J. Friedman et al, introduce *LogitBoost*, which fits at each stage an individual logistic regression function separately for each class, and which has been proved to be more robust to label noise than AdaBoost.

2.2 Bayesian Approaches

A multitude of methods to deal with label noise in the Bayesian setting have been proposed. The basic method of many of these is given in $[LS01]$, which gives a modified EM algorithm as follows: Given any arbitrary model $p(y \mid x, \theta)$ and a noisy data set $(X, \tilde{y})$, the objective is to maximize

$$L(\theta) = \sum_n \log(p(x_n \mid \tilde{y_n}, \theta)) \tag{4}$$

in θ. This is done, iteratively, in two steps. Firstly, by computing the posterior probability over the latent variables y_n (the true label)

$$p(y_n \mid x_n, \tilde{y_n}, \theta) = \frac{p(x_n, y_n \mid \tilde{y_n}, \theta)}{p(x_n \mid \tilde{y_n}, \theta}). \tag{5}$$

Secondly, using the newly computing latent probabilities in y_n, maximize

$$L(\theta) = \sum_n p(y_n \mid x_n, \tilde{y_n}, \theta) \log(p(x_n, y_n \mid \tilde{y}_n, \theta)). \tag{6}$$

Note that setting $\partial_\theta L(\theta) = 0$ will implicitly give updating rules for any class dependent features θ and flipping rates $p(\tilde{y} \mid y)$ involved in the model. After initialization, a basic EM algorithm iterates over these two steps until convergence is reached.

While this (very general) approach is the base of many Bayesian methods of dealing with label noise, a common problem is that the updating rules are heavily dependent on the underlying probabilistic model. These models will often times either be too simple for the data at hand, or become computationally very expensive when more complex models are considered ([LS01], p. 308).

3 Methods

3.1 Logistic Regression under inverse flip rates

Under class dependent label noise, any common classification method can be extended to a label noise robust classifier in the following way: Given a training data set X with corrupted labels $\tilde{y}$ on classes $1, .., C$, a classification method can correctly learn the probabilities

$$[P(\tilde{y} = 1 \mid X), ..., P(\tilde{y} = C \mid X)]. \tag{7}$$

We assume conditional independence of the clean labels from the data:

$$P(y \mid \tilde{y}, X) = P(y \mid \tilde{y}).$$

Then, given the inverse flip matrix $Q = (Q_{ij]})_{i,j \in 1,..,C}$ with

$$Q_{ij} = P(y = i \mid \tilde{y} = j),$$

a direct mapping to the class probabilities of the clean labels Y is given by

$$[P(y = 1 \mid X), ..., P(y = C \mid X)]^T = Q \ [P(\tilde{y} = 1 \mid X), ..., P(\tilde{y} = C \mid X)]^T. \tag{8}$$

This simple computation shifts the attention of building a label noise robust classifier from the training of a classification model to the approximation of the correct class-dependent flipping rates.

In respect thereof, a simple multinomial logistic regression has been implemented as a first method. Given the model

$$P(\tilde{y_n} = c \mid x_n, W) = \frac{\exp(w_c^T x_n)}{\sum_{l=1}^{C} \exp(w_l^T x_n)}, \tag{9}$$

the training weights can be learned by maximizing the log-likelihood on the training data

$$\mathcal{L}(W \mid X, \tilde{y}) = \sum_{n=1}^{N} \sum_{c=1}^{C} \tilde{y}_{n,c} \log \left(\frac{\exp(w_c^T x_n)}{\sum_{l=1}^{C} \exp(w_l^T x_n)} \right)^{\tilde{y}_{n,c}}, \tag{10}$$

where $\tilde{y}_{n,c} = 1$ if the sample $\tilde{y}_n$ is of class c and otherwise equals zero. The optimization is done via any gradient based method, where the gradient in W becomes

$$\nabla_W \mathcal{L}(W \mid X, \tilde{y}) = \sum_{n=1}^{N} (\mu_n - \tilde{Y_n}) \otimes x_n, \tag{11}$$

where $\mu_n = [P(\tilde{y_n} = 1 \mid x_n, W), ..., P(\tilde{y_n} = C \mid x_n, W)]$ and $\tilde{Y_n} = [\tilde{y}_{n,1}, ..., \tilde{y}_{n,C}]$ ([Mur12], p. 253).

Once the weights have been learned on the noisy training set $(X, \tilde{y})$, a new clean label prediction for a sample $\hat{x}$ can be computed via (8) as follows:

$$\hat{y} = argmax \ Q \cdot [P(\tilde{y_1} = 1, \mid \hat{x}, W), ..., P(\tilde{y_C} = 1, \mid \hat{x}, W)]^T. \tag{12}$$

3.2 Label Noise Robust Logistic Regression

A second logistic regression approach, based on [BK12], makes direct use of flipping rates between the labels in the model:

Given the flipping rates

$$\gamma_{jk} = P(\tilde{y} = k \mid y = j), \tag{13}$$

we model

$$P(\tilde{y}_n = c \mid x_n, W) = \sum_{l=1}^{C} \gamma_{lc} P(y_n = l \mid x_n, W), \tag{14}$$

where $P(y_n = l \mid x_n, W)$ is again computed via the softmax function

$$P(y_n = l \mid x_n, W) = \frac{\exp(w_l^T x_n)}{\sum_{j=1}^{C} \exp(w_j^T x_n)}. \tag{15}$$

Again, the model is trained by maximizing the likelihood on the training data

$$\mathcal{L}(W \mid X, \tilde{y}) = \sum_{n=1}^{N} \sum_{c=1}^{C} \tilde{y}_{n,c} \log(P(\tilde{y}_n = c \mid x_n, W)). \tag{16}$$

The optimization can be done via any gradient based method, where the gradient in $w_k, k \in 1...C$ becomes ([BK12], p. 146):

$$\nabla_W \mathcal{L}(W \mid X, \tilde{y}) = \sum_{n=1}^{N} \sum_{c=1}^{C} \frac{\tilde{y}_{n,c}}{P(\tilde{y}_n = c \mid x_n, W)} \cdot \frac{\exp(w_k^T x_n) \cdot x_n \cdot (\sum_{j=1}^{C} (\gamma_{kc} - \gamma_{jc}) \cdot \exp(w_j^T x_n)}{(\sum_{l=1}^{C} \exp(w_l^T x_n))^2}. \tag{17}$$

Note that the weights W are optimized to predict clean labels. Thus, for a new sample $\hat{x}$, a clean label prediction can be computed via (15) as follows:

$$\hat{y} = argmax[P(y = 1 \mid \hat{x}, W), ..., P(y = C \mid \hat{x}, W)]. \tag{18}$$

This rMRL method generalizes in another way. In the case that the flipping rates are unknown, the following multiplicative update of $\gamma_{j,k}$ will converge to the true value ([BK12], p. 6):

$$\gamma_{j,k} = \frac{1}{C} \gamma_{j,k} \sum_{n=1}^{N} \frac{\tilde{y}_{n,k}}{P(\tilde{y}_n = c \mid x_n, W)} \cdot \frac{\exp(w_j^T x_n)}{\sum_{l=1}^{C} \exp(W_l^T x_n)}, \tag{19}$$

where

$$C = \sum_{k=1}^{C} \gamma_{j,k} \sum_{n=1}^{N} \frac{\tilde{y}_{n,k}}{P(\tilde{y}_n = c \mid x_n, W)} \cdot \frac{\exp(w_j^T x_n)}{\sum_{l=1}^{C} \exp(W_l^T x_n)}.$$

Thus, rMRL optimizes predictive ability of clean labels for new samples and the original flipping rates at the same time. This will be made use of later on for the CIFAR dataset, where the flipping rates are unknown.

3.3 Label Noise Robust Neural Network

For a classification setting with c different classes, the observed samples $(x, y) \in \mathcal{X} \times \mathcal{Y}$, with $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1, ..., c-1\}$, are drawn from an unknown distribution $P(x, y)$ over $\mathcal{X}, \mathcal{Y}$. The distribution can be expressed as:

$$P(x, y) = P(y|x)P(x) \tag{20}$$

Training a classifier, in this case a neural network, to estimate under $\hat{P}(y|x)$ the class conditional probability $P(y|x)$ is therefore a valid method to predict class labels for observations.

A neural network with n layers for a classification problem with c classes is a transformation $f : \mathcal{X} -> \mathbb{R}^c$ with $f = (f^{(n)} \circ f^{(n-1)} \circ ... \circ f^{(1)})$ where $f^{(i)}$ is an intermediate transformation or layer defined by:

$$\forall x^{(i)} \in \mathbb{R}^{d^{(i)}}, f^{(i)}(x^{(i)}) = \begin{cases} \sigma(W^{(i)}x^{(i)} + b^{(i)}), \forall i \in [1, n-1] \\ \text{softmax}(W^{(i)}x^{(i)} + b^{(i)}), i = n \end{cases} \tag{21}$$

In the previous definition, $W^{(i)}$ represents the weights or parameters of the intermediate transformation, $x^{(i)}$ the intermediate feature vector obtained as $x^{(i)} = f^{(i-1)}(x^{(i-1)})$, σ a non linear convex activation function such as ReLU (rectified linear unit, [NH10]) $\sigma(z) = max(0, z)$, and the softmax operator is defined as

$$\forall f(x) = (f(x)_1, ..., f(x)_c) \in \mathbb{R}^c, \text{softmax}(f(x)) = (\frac{e^{f(x)_1}}{\sum_{i=1}^{c} e^{f(x)_i}}, ..., \frac{e^{f(x)_c}}{\sum_{i=1}^{c} e^{f(x)_i}}) \tag{22}$$

This definition ensures that the output of the network is indeed an estimation of the class-conditional probabilities $\hat{P}(y|x)$, and prediction of the instance label can be achieved by taking the most likely class given the instance: $\hat{y} = \arg\max_{i \in \{0,1,...,c-1\}} f(x)_i = \arg\max_{i \in \{0,1,...,c-1\}} \hat{P}(y = i|x)$

To train the network, the discrepancy between the estimated and true class-conditional probabilities must be minimized. This is achieved through gradient descent (called backpropagation in the case of neural networks) on the output of a loss function. Generally, the cross-entropy loss is used for classification. It is defined for the estimated class conditional probability $\hat{P}(y|x)$ against a class label y_i as:

$$\forall y_i \in \{0, 1, ..., c-1\}, L(y_i, \hat{P}(y|x)) = -\log(\hat{P}(y = y_i|x)) \tag{23}$$

Unfortunately, this loss is not robust to label noise. In a case where the training set contains label noise, then the samples are drawn from $P(x, \tilde{y})$ where $\tilde{y} \in \mathcal{Y}$ are noisy labels, and the learned class conditional probability will be $\hat{P}(\tilde{y}|x) \neq \hat{P}(y|x)$.

Nevertheless, it is possible to correct that loss to make it robust. In [Nat+13], N. Natarajan et al state that the following corrected loss, in a binary classification setting, where $y \in \{-1, 1\}$ the "true" class label, $p \in \{-1, 1\}$ the predicted class label and $\rho_i = P(\tilde{y} = -i|y = i)$, is robust.

$$\tilde{L}(y, p) = \frac{(1 - \rho_{-y})L(y, p) - \rho_y L(-y, p)}{1 - \rho_{+1} - \rho_{-1}} \tag{24}$$

indeed, $\tilde{L}$ is unbiased under the noisy labels

$$\mathbb{E}_{\tilde{y}}(\tilde{L}(p, \tilde{y})) = L(p, y) \tag{25}$$

which means that the empirical risk of the corrected loss function on the noisy training set of size N, defined as:

$$\hat{R}_{\tilde{L}}(\hat{P}(\tilde{y}|x)) = \frac{1}{N} \sum_{i=1}^{N} \tilde{L}(\tilde{y}_i, \hat{P}(\tilde{y}|x_i)) \tag{26}$$

converges towards the empirical risk of the original loss function on the training set non affected by label noise:

$$\mathbb{E}_{\tilde{y}}(\hat{R}_{\tilde{L},(x,\tilde{y})}(\hat{P}(\tilde{y}|x))) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{P}(\tilde{y}|x_i)) = \hat{R}_{L,(x,y)}(\hat{P}(\tilde{y}|x)) \tag{27}$$

and therefore, the estimated $\hat{P}(\tilde{y}|x)$ during training will actually converge towards $\hat{P}(y|x)$, which corresponds to the class dependant conditional probability, free of label noise.

In [Pat+17] G. Patrini et al extend that method to a multiclass setting. With some abuse of notation let's define the loss vector computed on every possible class label:

$$\mathbf{L}(\hat{P}(\tilde{y}|x)) = [L(0, \hat{P}(\tilde{y}|x)), ..., L(c-1, \hat{P}(\tilde{y}|x))]) \tag{28}$$

for the transition matrix,

$$T = \begin{bmatrix} P(\tilde{y}=1|y=1) & P(\tilde{y}=1|y=2) & ... & P(\tilde{y}=1|y=c) \\ P(\tilde{y}=2|y=1) & P(\tilde{y}=2|y=2) & ... & P(\tilde{y}=2|y=c) \\ ... & ... & ... & ... \\ P(\tilde{y}=c|y=1) & P(\tilde{y}=c|y=2) & ... & P(\tilde{y}=c|y=c) \end{bmatrix} \tag{29}$$

the corrected loss is then defined as:

$$\tilde{\mathbf{L}}(\hat{P}(\tilde{y}|x)) = (T^T)^{-1}\mathbf{L}(\hat{P}(\tilde{y}|x)) \tag{30}$$

It is unbiased, and will yield the same minimizer than the original loss function on the clean training set

$$\arg\min_{\hat{P}(y|x)} \mathbb{E}_{x,\tilde{y}}[\tilde{L}(y, \hat{P}(y|x))] = \arg\min_{\hat{P}(y|x)} \mathbb{E}_{x,y}[\tilde{L}(y, \hat{P}(y|x))] \tag{31}$$

indeed, the following holds:

$$\begin{aligned} \mathbb{E}_{\tilde{y}|x}[\tilde{\mathbf{L}}(\hat{P}(y|x))] &= \frac{1}{c}\sum_{i=0}^{c-1} P(\tilde{y}=i|x)\tilde{\mathbf{L}}(\hat{P}(y|x)) \\ &= \frac{1}{c}\sum_{i=0}^{c-1}\sum_{j=0}^{c-1} P(\tilde{y}=i|y=j,x)P(y=j|x)\tilde{\mathbf{L}}(\hat{P}(y|x)) \\ &= \frac{1}{c}\sum_{i=0}^{c-1}\sum_{j=0}^{c-1} P(\tilde{y}=i|y=j)P(y=j|x)\tilde{\mathbf{L}}(\hat{P}(y|x)) \\ &= \mathbb{E}_{y|x}[T^T\tilde{\mathbf{L}}(\hat{P}(y|x))] \\ &= \mathbb{E}_{y|x}[\mathbf{L}(\hat{P}(y|x))] \end{aligned} \tag{32}$$

To successfully implement in pytorch the corrected loss, a convolutional neural network was designed. Figure 1 shows the full architecture implemented; it features three convolutional layers, which have been shown to successfully extract features from input images [LBH15] [KSH12], with several fully connected layers that can combine these features together to establish meaningful class-conditional probabilities. For a complete description of the network, see 5 in Annex. Instead of using the package's standard cross-entropy loss function, the corrected robust loss function was implemented.

In practice, the output of the network, for an input $x \in \mathbb{R}^d$ is the vector $\hat{P}(\tilde{y}|x) = [\hat{P}(\tilde{y} = 0|x), ..., \hat{P}(\tilde{y} = c-1|x)]$. All possible losses are then computed under $\mathbf{L}(\hat{P}(\tilde{y}|x)) =$

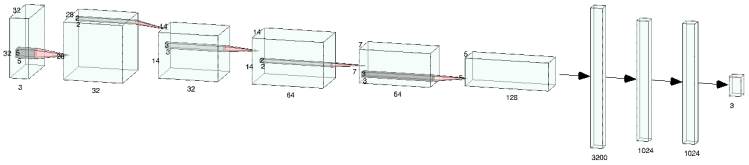


Figure 1: Network Architecture

$[L(0, \hat{P}(\tilde{y}|x)), ..., L(c-1, \hat{P}(\tilde{y}|x))]$ and combined, as according to equation 30 to $\tilde{\mathbf{L}}(\hat{P}(\tilde{y}|x)) = [\tilde{L}(0, \hat{P}(\tilde{y}|x)), ..., \tilde{L}(c-1, \hat{P}(\tilde{y}|x))]$ from which we only keep the loss for the label associated to the input $\tilde{y}_i \in \mathcal{Y}$, $\tilde{L}(\tilde{y}_i, \hat{P}(\tilde{y}|x))$. The back propagation finally starts on the selected corrected loss and will trigger an update of the weights of the network that will, as proven earlier, make the estimated $\hat{P}(\tilde{y}|x)$ converge towards $P(y|x)$

This corrected loss provides an efficient way to create a neural network robust to label noise, completely independent of the network architecture. It is worth noting that this method is actually applicable to any classifier that uses a similar loss function. Nevertheless, this method requires a prior knowledge of the transition matrix T to be applicable.

Luckily, in [Pat+17], G. Patrini et al, provide a simple way to estimate T. The following assumptions are necessary for such estimation:

- There exist anchor points, or perfect samples of each class $i \in \{0, ..., c-1\}$

$$\exists x^i \in \mathcal{X} \text{ s.t } P(x^i) > 0 \text{ and } P(y = i|x^i) = 1 \tag{33}$$

- Considering enough samples corrupted by label noise, the neural network function f models $P(\tilde{y}|x)$ accurately.

It follows that the components of the transition matrix are:

$$T_{ij} = P(\tilde{y} = i|x^j) \tag{34}$$

indeed, under the second assumption

$$\hat{P}(\tilde{y}|x) = P(\tilde{y}|x) \tag{35}$$

which then using the following equality

$$\begin{aligned} \forall i \in \{0, ..., c-1\}, P(\tilde{y} = i|x) &= \sum_{k=0}^{c-1} P(\tilde{y} = i|y = k, x) P(y = k|x) \\ &= \sum_{k=0}^{c-1} P(\tilde{y} = i|y = k) P(y = k|x) \\ &= \sum_{k=0}^{c-1} T_{ik} P(y = k|x) \end{aligned} \tag{36}$$

with the consequence of the first assumption $\forall k \neq j$, $p(y = k|x^j) = 0$, proves equation (34)

This provides an effective way to estimate the transition matrix:

1. Train the network on the corrupted data, so that $f(x) = \hat{P}(\tilde{y}|x) \approx P(\tilde{y}|x)$

2. Obtain a large sample of observations $\mathcal{X}^{'}$ (for example concatenate the training data with the test data) and find anchor points for each class by selecting $x^j = \arg\max_{x \in \mathcal{X}^{'}} P(\tilde{y} = j|x)$
3. Estimate the elements of the transition matrix $\hat{T}_{ij} = P(\tilde{y} = i|x^j)$

4 Experiments

4.1 Datasets

Two different datasets were used to evaluate the performance of methods robust to label noise. Both datasets contain images and are originally supposed to be label noise free. They were split in two sub-datasets, namely a training dataset, where labels where purposefully mixed up to create label noise, and a test dataset, kept clean from label noise.

4.1.1 The MNIST Fashion Dataset

The MNIST Fashion dataset[4], assembled by *Zalando Research* in [XRV17] contains 60 000 train and 10 000 test images, of size 28x28 pixels, in greyscale, depicting 10 different clothing items. This dataset was assembled with the goal to replace the original MNIST[5] dataset developed in [LeC+98] (It has the same number of images and the same image format), which is too easy, overused and does not represent modern CV tasks.

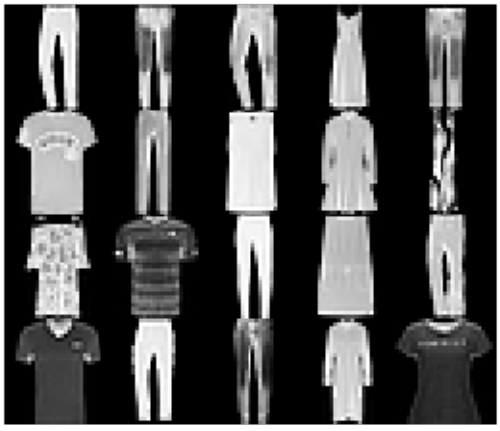


Figure 2: 20 samples from the reduced MNIST Fashion dataset (3 classes)

The Fashion MNIST dataset was reduced to only three classes for this study, trousers, dresses and t-shirts. This therefore decreased the size of the whole dataset to 18000 training images and 3000 testing images. Figure 2 shows a sample from the dataset. One can immediately see that the samples from the different classes are easily separable and have distinctive features (e.g the gap between the legs for trousers, or larger waist for dresses). It can therefore be expected that classifiers perform very well on such dataset. Furthermore, the samples are in fact so separable that common dimensionality reduction techniques could provide an efficient way to conduct classification without the need of labels.

Figure 3 represents respectively the t-SNE[6] and UMAP[7] reductions of the clean[8] Fasion MNIST dataset. For both projection, the three classes are clearly separated. Running a clustering algorithm, such as K-NN, first developed in [CH67], with three clusters and then manually selecting a few samples from each of the three clusters to assign each cluster a respective class would constitute a simple yet efficient method to handle label noise. Indeed, the clusters are formed without any labels, and the whole method is thus insensitive to label noise.

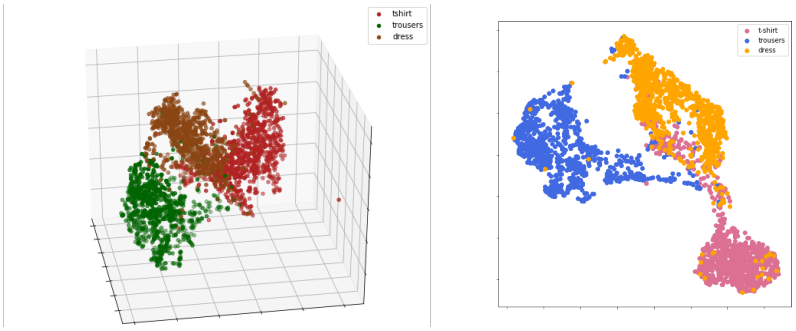
[4] https://github.com/zalandoresearch/fashion-mnist

[5] http://yann.lecun.com/exdb/mnist/

[6] See [MH08]

[7] see [McI+18]

[8] The clean dataset was used to accurately display the legend of the scatter plots.



(a) (b)

Figure 3: Dimensionality reduction techniques applied on the reduced MNIST Fashion dataset. On the left, figure 3a shows the result of the t-SNE projection, while the right, figure 3b, shows the result of UMAP projection

4.1.2 The CIFAR Dataset

The CIFAR dataset[9], assembled by Alex Krizhevsky and Geoffrey Hinton for [KH+09] contains 50 000 train and 10 000 test images, of size 32x32 pixels, in RGB, depicting 10 entities (namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).

The CIFAR dataset was reduced to only three classes for this study, airplane (or later referred as plane), automobile (or later referred as car) and cat. This therefore decreased the size of the whole dataset to 15000 training images and 3000 testing images. Figure 4 shows a sample from the dataset.

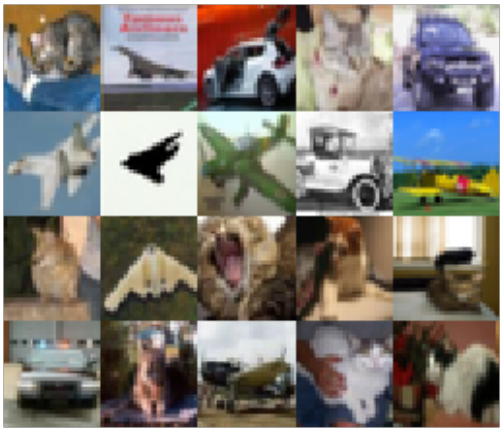
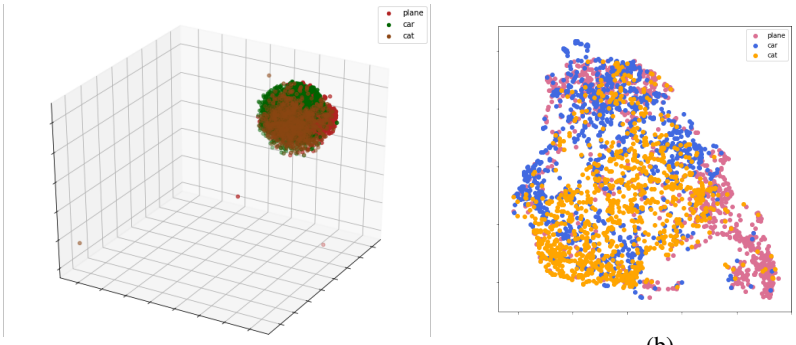


Figure 4: 20 samples from the reduced CIFAR dataset (3 classes)

Here, as opposed to the Fashion MNIST dataset, the samples lie in a higher dimensional space and are drawn from distributions less clearly separated. No obvious and simple features allow for a separation of the samples. This is made obvious in figure 5 as it shows that both t-SNE and UMAP fail to separate the three classes.

This therefore invalidate the clustering method described earlier, in 4.1.1, as a general method to handle with datasets with noisy labels. It only works because of the simplicity of the Fashion MNIST dataset.

[9]https://www.cs.toronto.edu/ kriz/cifar.html



(a) (b)

Figure 5: Dimensionality reduction techniques applied on the reduced CIFAR dataset. On the left, figure 5a shows the result of the t-SNE projection, while the right, figure 5b, shows the result of UMAP projection

4.2 Label Noise

The datasets used are originally not supposed to be contaminated by label noise. They are both repeatedly used by the machine learning and computer vision communities, and if they originally contained any label noise, it will be assumed that today it has been eliminated.

For this study, artificial label noise must be introduced to the training datasets, while the test datasets, which are used to test the generalization performance of the classifiers, are kept clean from label noise. The artificial label noise introduced is class dependant which means that for data x and a class label y the probability of noisy label is $P(\tilde{y}|x, y) = P(\tilde{y}|y)$. The overall noise level in the dataset can therefore be characterized by its transition matrix, defined for a dataset with c different classes by:

$$T = \begin{bmatrix} P(\tilde{y}=1|y=1) & P(\tilde{y}=1|y=2) & ... & P(\tilde{y}=1|y=c) \\ P(\tilde{y}=2|y=1) & P(\tilde{y}=2|y=2) & ... & P(\tilde{y}=2|y=c) \\ ... & ... & ... & ... \\ P(\tilde{y}=c|y=1) & P(\tilde{y}=c|y=2) & ... & P(\tilde{y}=c|y=c) \end{bmatrix} \tag{37}$$

For the **Fashion MNIST** dataset, two label noise corruptions were used. In the first case which will be referred to as *MNIST 0.5*, labels have an equal probability of being or not corrupted. The transition matrix is:

$$T_{mnist0.5} = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix} \tag{38}$$

In the second case, referred to as *MNIST 0.6*, labels have a greater probability of being corrupted. The transition matrix is:

$$T_{mnist0.6} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.3 & 0.4 & 0.3 \\ 0.3 & 0.3 & 0.4 \end{bmatrix} \tag{39}$$

Figure 6 shows samples from the MNIST 0.5 training dataset, with their associated labels. Numerous mismatches between the images and their labels are present, which very clearly shows that the training dataset is corrupted with label noise.

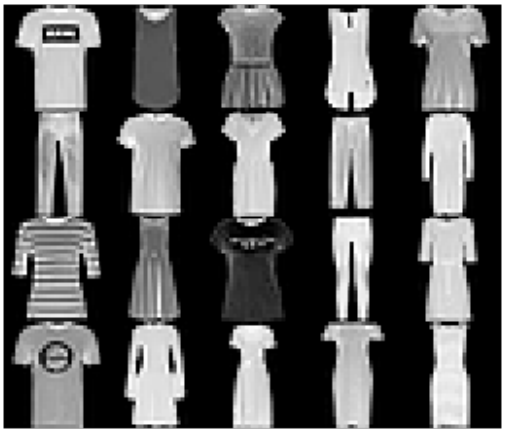


Figure 6: 20 samples from the reduced MNIST Fashion dataset that show the presence of label noise. Respective noisy labels —*dress, trousers, dress, trousers, dress, t-shirt, trousers, t-shirt, trousers, t-shirt, trousers, trousers, trousers, dress, dress, t-shirt, dress, dress, dress, dress*—

For the **CIFAR** dataset, an unknown label noise corruption was added to the training dataset. The transition matrix is therefore unknown at this stage in the study.

$$T_{cifar} = ? \tag{40}$$

Figure 7 shows samples from the CIFAR training dataset, with their associated labels. Numerous mismatches between the images and their labels are present, which very clearly shows that the training dataset is corrupted with label noise.

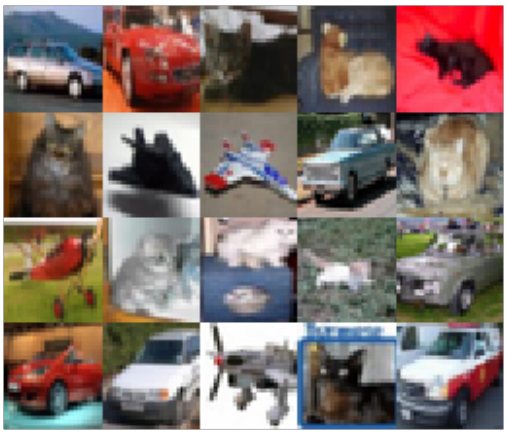


Figure 7: 20 samples from the reduced CIFAR dataset that show the presence of label noise. Respective noisy labels —*car, cat, plane, cat, car, plane, plane, plane, car, plane, cat, cat, plane, plane, car, car, cat, plane, car, car*—

It is worth noting that it is possible to estimate the prior probabilities of each class labels, both with and without label noise. Counting the number of instances of each class in the test sets provide an estimation of the priors of the clean class instances. The same procedure on the training sets provide an estimation of the priors of the noisy class instances.

Table 1 reveals that all classes, both with an without label noise corruption are balanced.

Dataset	$P(y=1)$	$P(y=2)$	$P(y=3)$	$P(\tilde{y}=1)$	$P(\tilde{y}=2)$	$P(\tilde{y}=3)$
MNIST 0.5	0.33	0.33	0.33	0.33	0.33	0.33
MNIST 0.6	0.33	0.33	0.33	0.33	0.33	0.33
CIFAR	0.33	0.33	0.33	0.33	0.33	0.33

Table 1: Prior probabilities for class labels

An interesting consequence of that balance is that the inverse transition matrix Q, defined as:

$$Q = \begin{bmatrix} P(y=1|\tilde{y}=1) & P(y=1|\tilde{y}=2) & ... & P(y=1|\tilde{y}=c) \\ P(y=2|\tilde{y}=1) & P(y=2|\tilde{y}=2) & ... & P(y=2|\tilde{y}=c) \\ ... & ... & ... & ... \\ P(y=c|\tilde{y}=1) & P(y=c|\tilde{y}=2) & ... & P(y=c|\tilde{y}=c) \end{bmatrix} \tag{41}$$

verifies, under completely balanced priors

$$Q = T^T \tag{42}$$

indeed, using Bayes' theorem, and noting $T_{i,j} = P(\tilde{y}=i|y=j)$ and $Q_{j,i} = P(y=j|\tilde{y}=i)$

$$T_{i,j} = Q_{j,i}\frac{P(\tilde{y}=i)}{P(y=j)} = Q_{j,i} \tag{43}$$

4.3 Measure of Performance of the Methods

Throughout the study, the classification accuracy was used as performance measure of the classifiers. It is expressed as:

$$A = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{Y[i]=Y_{pred}[i]\}} \tag{44}$$

4.4 Results

4.4.1 Benchmark Results

At first, to get a an accuracy performance benchmark, both methods (MLR and NN) have been cross-validated on the test set for each dataset. This is in order to test how well the methods perform on data that is not corrupted by label noise. This upper bound will serve to compare the robustness of the methods, which is the main subject of this study.

Table (2) shows the averaged accuracy (and standard deviation) over 5 folds on the respective test datasets. The MLR optimization was run via stochastic gradient descent with batch size 100, step size and $\eta = 0.1$ and 1000 maximum iterations. For the neural network, the training was stopped as soon as the validation loss plateaued, which corresponds to 7 epochs for MNIST and 30 for CIFAR, to prevent overfitting. The optimization was ran through stochastic gradient descent with batch size 252, learning rate 0.01 and momentum 0.9.

Method	MNIST	CIFAR
MLR	0.955 (0.0140)	0.746 (0.0468)
NN	0.953 (0.0092)	0.745 (0.0560)

Table 2: Upper bound for maximum possible accuracy

4.4.2 Test Results

In order to test how well the methods perform in terms of label noise robustness and generalization, all methods have been run 10 times on 90% of the respective training datasets (with noisy labels) and tested on the complete test dataset (with true labels) each time.

The MLR optimization was run via stochastic gradient descent with batch size 100, step size $\eta = 0.1$ and 1000 maximum iterations. The stochastic gradient descent optimization for the rMLR has been run with batch size 100, step size $\eta = 0.1$ and, to keep computing times in check, with 250 maximum iterations. All gradient descent runs have converged within a tolerance of $3e - 6$.

On the CIFAR dataset, rMLR has been run with a non-constant transition matrix in order to make use of the inherent estimation of the inverse flipping rates described in section (3.2). On another run, the transition matrix given by the rNN method has been used. This saw increased performance, and thus, this matrix T has also been used for the MLR runs on the CIFAR dataset.

For the implementation of the rNN, it is worth mentioning that for the sake of simplicity, the images coming from the Fashion MNIST datasets were resized to 32x32 pixels and transformed to RGB so that they could be fed in the same network as images coming from the CIFAR dataset. Figure 8 shows samples from Fashion MNIST after the transformation to RGB, some pixel values are altered but one can hope that these transformation will not alter the performance of the classification.

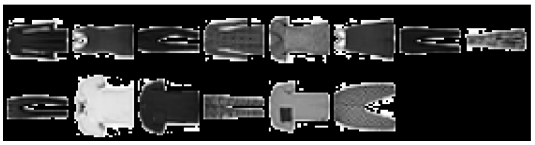


Figure 8: 14 samples from the RGB transformed Fashion MNIST dataset

The rNN optimization was also run via stochastic gradient descent with batch size 252 learning rate 0.01 and using momentum as advised in [Sut+13] of 0.9.

Lastly, for the neural network, the performance of the classifier, because it has many parameters to tune, depends on the number of epochs the training was ran on. An epoch being a complete run through the training dataset. Too few training epochs means that the model will underfit, too many many epochs on the other hand means that the model will overfit. The results are therefore presented for different epochs value, to study the influence of a longer training on the performance of the classifier.

The test results are displayed in Table (3). For the MLR and rNN results, the accuracy with and without adjusting the predictions via the transition matrix are given, where the label (flipped) corresponds to the results in which predictions based on noisy labels have been transformed to clean label predictions.

We make the following observations on the test results:

Method	**MNIST0.5**	**MNIST0.6**	**CIFAR**
MLR	0.764 (0.0183)	0.598 (0.044)	0.353 (0.0685)
MLR (Flipped)	0.834 (0.0166)	0.604 (0.048)	0.377 (0.0438)
rMLR	0.891 (0.0024)	0.812 (0.002)	0.477 (0.0074), 0.36 (0.0571)
rNN (2 Ep.)	0.863 (0.044)	0.842 (0.0315)	0.385 (0.0606)
rNN (2 Ep., Flipped)	0.726 (0.1993)	0.5733 (0.0976)	0.377 (0.0259)
rNN (7 Ep.)	0.931 (0.0049)	0.878 (0.0099)	0.470 (0.0751)
rNN (7 Ep., Flipped)	0.914 (0.0123)	0.775 (0.0295)	0.376 (0.0335)
rNN (30 Ep.)	0.6211 (0.0315)	0.482 (0.1024)	0.381 (0.0040)
rNN (30 Ep., Flipped)	0.9029 (0.0146)	0.808 (0.0153)	0.409 (0.0754)

Table 3: Test Results: Average Accuracy (Standard Deviation) on the Test Set, trained on 90% of Training Set over 10 folds

- Even though no method can match the benchmark performance on a clean training set (see table 2), it is clearly observable that the robust methods outperform their original counterparts:
  - Over all data sets, a standard MLR predicts worse than a MLR with transformed predictive probabilities [MLR (flipped)], which in turn performs worse than rMLR. This is in line with the theory, in the sense that rMLR is not dependent on the independence assumption made in section (3.2), and thus generalizes better in situations where this assumption does not necessarily apply.
  - In the case of the neural network, even though the maximum reached accuracy on all datasets is not reached using the corrected robust loss, a clear trend can be observed. The accuracy does not drop significantly when the number of epochs increases. Without this correction, it is intuitive to see the accuracy drop when the number of epochs increases; the network will be forced to overfit the noisy labels and consequently loose in generalization ability. With the correction, such overfitting does not occur anymore and is of great interest. If the datasets were more complex, and therefore required longer training to reach a high accuracy (to compare with 90% accuracy reached after only 2 epochs on Fashion MNIST), such noise correction would be paramount to reaching a final high classification accuracy.
- Over the three datasets, a generally worsening performance can be seen going from MNIST0.5 to MNIST0.6 and then again, going to CIFAR. The drop in performance between the first two is easily explained by the increase in label noise for MNIST0.6, which can be seen by comparing the respective transition matrices. For the case of the CIFAR dataset, the overall bad performance might be due to two issues. Firstly, this dataset is fundamentally more complex than the MNIST datasets, due to the RGB filters and more complex image shapes. Figure $(5b)$ in section (4.1.2) shows how complex the decision boundaries between the CIFAR classes are. Additionally, there are hints of another increase in label noise going from MNIST0.6 to CIFAR. The two transition matrices for the CIFAR classes, approximated by rMLR and rNN, can be seen in equations (45) and (46). Compared to $T_{mnist0.6}$, both approximations show an increase in flipping rates away from the original classes.

$$\hat{T}_{cifar}^{rLR} = \begin{bmatrix} 0.336(0.0039) & 0.353(0.0149) & 0.315(0.0146) \\ 0.341(0.0105) & 0.331(0.0080) & 0.327(0.0181) \\ 0.324(0.0137) & 0.316(0.0209) & 0.359(0.0318) \end{bmatrix} \tag{45}$$

$$\hat{T}_{cifar}^{rNN} = \begin{bmatrix} 0.366\ (0.0371) & 0.277\ (0.0781) & 0.283\ (0.0574) \\ 0.329\ (0.0345) & 0.414\ (0.0837) & 0.326\ (0.0472) \\ 0.304\ (0.0479) & 0.309\ (0.0637) & 0.391\ (0.0625) \end{bmatrix} \tag{46}$$

Indeed, the estimated transition matrices suggest that about only a third of all labels are correct in the dataset. Under such massive label noise, the implemented methods might run into a practical limit and their efficiency collapse. In the case where a real dataset would showcase such massive label noise, it would be wise to first preprocess the dataset to either filter out the noisy labels -if the dataset is large enough- or clean it by re-affecting noisy labels; robustness to label noise is in this case not sufficient.

5 Conclusion

This paper set out to examine the problem of class dependent label noise in common classification methods and possible ways to deal with its negative effects. In this regard, Multinomial Logistic Regression and two of its label noise adjustments have been implemented; one, by subsequent use of a transition matrix to transform corrupted label predictions into clean label predictions, and one by incorporating the flipping probabilities between classes in its underlying model. Secondly, a Neural Network classifier has been introduced, which uses a corrected loss function, leveraging the information provided by the transition matrix, to ensure robustness against label noise.

Given the experimental results on the three data sets given in section 4.4.2, we conclude that the adjustments have successfully made both methods, the MLR and the Neural Network, more robust to class dependent label noise. The exception in this regard were the experiments on the CIFAR data set. If this is due to the inherent complexity of the data or to the rate of noise applied to this set has to be explored in the future. Further points of exploration include:

- In the scope of the rMRL introduced in [BK12], a sparsity regulation term can be introduced. The sparsity constraint might help to deal with prediction problems when dealing high-dimensional data, as might be the case in the CIFAR set.
- In [Rol+17], D. Rolnick experimentally hints that deep enough neural networks are naturally robust to label noise, under certain conditions. These conditions include a network deep enough, batch size small enough and a dataset large enough to contain sufficiently numerous correctly labeled observations, no matter the amount of incorrectly labeled observations. It would therefore be of great interest to leverage these findings to increase the performance of our robust Neural Network by optimizing its depth as well as the batch size used during the optimization.done for me yes

Running our code

Running our code is as simple as it gets. Open *rNN.ipynb* and run all cells for everything related to the robust neural network or *rMLR.ipynb* and run all cells for everything related to the robust (or not) multinomial logistic regressions.

Contributions

Part	Main Contributor(s)
Abstract	Pierre
Introduction	Pierre
Related Works	
Ensemble Methods	Pierre
Bayesian Approaches	Thorben
Methods	
Logistic Regression	Thorben
Label Noise Robust Logistic Regression	Thorben
Label Noise Robust Neural Network	Pierre
Experiments	
Datasets	Pierre
Label Noise	Pierre
Measures	Pierre
Results	Thorben + Pierre
Conclusion	Thorben

Computer Specifications

The Algorithms were ran on our personal computers.

Pierre possesses a 13' Mac Book Pro Retina (2014) with a 2.8GHz dual-core Intel Core i5 processor [3MB shared L3 cache] and 8GB of 1600MHz DDR3L onboard memory.

To speed up the training and experiments with the neural network, google colab, running python3 with GPU accelerator was used. This corresponds to using a Intel(R) Xeon(R) CPU @ 2.30GHz with Nvidia Tesla K80

Thorben's possesses a 5th generation surface Pro with an Intel® Core™ i5 of 2.6GHz, and 4GB of RAM.

References

[CH67] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

[DS79] Belur V Dasarathy and Belur V Sheela. "A composite classifier system design: concepts and methodology". In: *Proceedings of the IEEE* 67.5 (1979), pp. 708–713.

[Bre96] Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.

[Hic96] Ray J Hickey. "Noise modelling and evaluating learning from examples". In: *Artificial Intelligence* 82.1-2 (1996), pp. 157–179.

[FS97] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[LeC+98] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[BF99] Carla E Brodley and Mark A Friedl. "Identifying mislabeled training data". In: *Journal of artificial intelligence research* 11 (1999), pp. 131–167.

[Die00] Thomas G Dietterich. "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization". In: *Machine learning* 40.2 (2000), pp. 139–157.

[FHT+00] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)". In: *The annals of statistics* 28.2 (2000), pp. 337–407.

[LS01] D Lawrence and B Schölkopf. "Estimating a Kernel Fisher Discriminant in the Presence of Label Noise". In: *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*. 2001, pp. 306–311.

[Ser03] Rocco A Servedio. "Smooth boosting and learning with malicious noise". In: *Journal of Machine Learning Research* 4.Sep (2003), pp. 633–648.

[ZW04] Xingquan Zhu and Xindong Wu. "Class noise vs. attribute noise: A quantitative study". In: *Artificial intelligence review* 22.3 (2004), pp. 177–210.

[MBN06] Andrea Malossini, Enrico Blanzieri, and Raymond T Ng. "Detecting potential labeling errors in microarrays by data perturbation". In: *Bioinformatics* 22.17 (2006), pp. 2114–2121.

[MH08] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[Den+09] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[NH10] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[BK12] J. Bootkrajan and Ata Kaban. "Label-Noise Robust Logistic Regression and Its Applications". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD*. 2012, pp. 143–158.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[Mur12] Kevin P. Murphy. *Machine Learning - A Probabilistic View*. MIT Press, 2012.

[FV13] Benoıt Frénay and Michel Verleysen. "Classification in the presence of label noise: a survey". In: *IEEE transactions on neural networks and learning systems* 25.5 (2013), pp. 845–869.

[Nat+13] Nagarajan Natarajan et al. "Learning with noisy labels". In: *Advances in neural information processing systems*. 2013, pp. 1196–1204.

[Ser+13] Pierre Sermanet et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint arXiv:1312.6229* (2013).

[Sut+13] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*. 2013, pp. 1139–1147.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[Kam+17] Konstantinos Kamnitsas et al. "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation". In: *Medical image analysis* 36 (2017), pp. 61–78.

[Pat+17] Giorgio Patrini et al. "Making deep neural networks robust to label noise: A loss correction approach". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1944–1952.

[Rol+17] David Rolnick et al. "Deep learning is robust to massive label noise". In: *arXiv preprint arXiv:1705.10694* (2017).

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[Zha+17] Hengshuang Zhao et al. “Pyramid scene parsing network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2881–2890.

[McI+18] Leland McInnes et al. “UMAP: Uniform Manifold Approximation and Projection”. In: *The Journal of Open Source Software* 3.29 (2018), p. 861.

[Yan+19] Zhilin Yang et al. “XLNet: Generalized Autoregressive Pretraining for Language Understanding”. In: *arXiv preprint arXiv:1906.08237* (2019).

Annex

Neural Network Architecture

The architecture of the neural network is defined as follows:

- INPUT: 32x32x3 tensor (corresponds to a 32x32 RGB image)
- ——-
- CONV 1: kernel size 5, stride 1, padding 0, output channels 32, activation function ReLU
- MAX POOL 1: kernel size, 2
- CONV 2: kernel size 3, stride 1, padding 1, output channels 64, activation function ReLU
- MAX POOL 2: kernel size, 2
- CONV 3: kernel size 3, stride 1, padding 0, output channels 128, activation function ReLU
- ——-
- FULLY CONNECTED 1: input channels: 3200, output channels 1024, with dropout (probability 0.5), activation function ReLU
- FULLY CONNECTED 2: input channels: 1024, output channels 1024, with dropout (probability 0.5), activation function ReLU
- FULLY CONNECTED 3: input channels: 1024, output channels 3, activation function Linear

Please note that the last fully connected layer uses a linear activation function, and not a softmax because pytorch automatically performs the softmax operation in the implementation of the loss function, see https://pytorch.org/docs/stable/nn.htmlcrossentropyloss for more details.