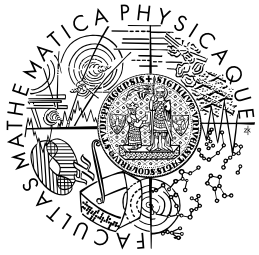


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Matěj Sochor

Semi-supervised Learning from Unfavorably Distributed Data

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: Mgr. Martin Pilát, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my thesis supervisor, Mgr. Martin Pilát, Ph.D., for being there whenever I needed an advice. I would also like to thank all those who helped me in my studies, be it by support, example, or great teaching.

Title: Semi-supervised Learning from Unfavorably Distributed Data

Author: Matěj Sochor

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Semi-supervised learning (SSL) is a branch of machine learning focusing on using not only labeled data samples, but also unlabeled ones, in an effort to decrease the need for labeled data and thus allow using machine learning even when labeling large amounts of data would be too costly. Despite its quick development in the recent years, there are still issues left to be solved before it can be broadly deployed in practice. One of those issues is class distribution mismatch. It arises when the unlabeled data contains samples not belonging to the classes present in the labeled data. This confuses the training and can even lead to getting a classifier performing worse than a classifier trained on the available data in purely supervised fashion.
We designed a filtration method called Unfavorable Data Filtering (UDF) which extracts important features from the data and then uses a similarity-based filter to filter the irrelevant data out according to those features. The filtering happens before any of the SSL training takes places, making UDF usable with any SSL algorithm. To judge its effectiveness, we performed many experiments, mainly on the CIFAR-10 dataset. We found out that UDF is capable of significantly improving the resulting accuracy when compared to not filtering the data, identified basic guidelines as to when to use it and discovered an important property of the class distribution mismatch issue.

Keywords: Semi-supervised Learning Deep Learning Unbalanced distribution

Contents

Introduction

Deep learning is a branch of machine learning using neural networks composed of many layers. Those layers identify more and more complex features of the input, until the last one, which is used to solve the task at hand, such as classifying the input into one of the predetermined classes. A decade ago, advances in hardware and neural network learning techniques finally made deep learning usable in practice, leading to a boom that revolutionized machine learning and allowed computers to solve issues that either had no automatic solutions before or the solutions available performed poorly. Image recognition, machine translation and many other areas suddenly acquired completely new state of the art models. Moreover, deep learning is developing extremely quickly, bringing new and exciting results every few months.

However, despite all the good it brings, it still has issues that often prevent its use in practice. Principal among them is the need for large amounts of labeled data. If there are insufficient quantities of such data, problems such as overfitting severely degrade performance of deep learning models, often up to the point of making deep learning useless. While sometimes it might be easy enough to procure more labeled data, in other cases it might be very expensive or downright impossible to do so. To solve this issue, various semi-supervised learning (SSL) algorithms have been developed. Unlike supervised learning, SSL uses not only the labeled data, but also unlabeled data, which is generally far easier to come by. It has been shown that SSL is capable of improving the results when we do not have enough labeled data for supervised learning. That might be taken as evidence that the problem is solved. However, SSL presents its own set of problems which severely limit its ability to perform well in practice.

One of those problems is its diminishing performance when the labeled and unlabeled data have different distributions. That can happen in a lot of different ways, from those well-known and studied in supervised learning, such as one of them having an unbalanced class distribution, up to those that are exclusive to SSL, for example the unlabeled data containing samples that do not belong to any of the classes from labeled data. The last issue mentioned is called class distribution mismatch. It has only been identified recently and has not been studied very much yet, even though it is a serious and very realistic issue. The goal of this thesis is to make SSL more robust to the issue of different distributions of labeled and unlabeled data. We decided to focus specifically on the class distribution mismatch issue, study what is known about it, add our own findings and design a technique mitigating the issue.

The thesis can be divided into two distinct parts - a theoretical part and a practical part. The purpose of the theoretical part is to provide information necessary to understand the rest of the thesis and to sum up information on the issue from available literature. This is done in the first two chapters of the thesis. The first chapter describes general machine-learning knowledge, while the second one describes the class distribution mismatch issue, what is known about it and how others have tried to solve it. In the practical part, on the other hand, we are concerned with identifying requirements for our solution of the issue, designing it and then evaluating its effectiveness in a series of experiments. It spans the other

two chapters. In the third chapter, we design our own solution, a filtering method trying to filter out data belonging to irrelevant classes before the actual training takes place. In the last chapter, we describe results of many different experiments and draw conclusions about the effectiveness of our solution and about the issue itself.

1. Machine Learning

Since this thesis investigates and tries to solve a problem in a specific branch of machine learning, we have to lay the theoretical foundations first, before delving into studying the problem itself and the proposed solution. Therefore, this chapter introduces basic concepts of machine learning and then focuses specifically on its areas needed to understand the rest of the thesis.

A large portion of information in this chapter comes from one of three books: Chapelle et al. [2006], Alpaydin [2014] or Goodfellow et al. [2016]. To not repeat ourselves constantly, we will not state these as sources in the text. Therefore, keep in mind that unless specified otherwise, these books or our own observations are the source of any of the stated information.

We should also note that the information present in this chapter is in no way exhaustive, since for the sake of brevity, we only focus on knowledge necessary to understand the rest of this thesis.

1.1 Introduction to Machine Learning

In the broadest meaning of the term, *machine learning* encompasses all algorithms that use knowledge automatically extracted from data or experience to improve their performance. That usually means creating a mathematical model and setting its parameters according to the available data. This allows us to solve many tasks for which we have no exact algorithm or the exact algorithm requires too much computation to be practical. The process of setting the parameters is often called *training* and the data used to set those parameters is referred to as *training data*. One update of the parameters is called a *step*, while a period when the entire training data is used is called an *epoch*. Lastly, we will refer to each instance in the training data as a *sample*.

1.1.1 Basic Terms

There are multiple types of machine learning tasks. Two of the most common are called *classification* and *regression*. A classification task consists of taking an input sample and assigning it to one of multiple predetermined classes. An example of a classification task could be deciding whether the animal in a picture is a cat or a dog. Output of the model in a classification task should be a decision to which class the sample belongs (or said in other words, what its *label* is), a regression task requires determining a number. This could, for example, mean determining a person's age based on a picture of their face. In this thesis, we concern ourselves with classification.

One important thing to note about classification is that the models we usually use do not output the decision directly, but rather a probability distribution over the possible classes. For example, for the aforementioned case an output of the model could be that with a probability of 80%, there is a dog in the picture, and with a probability of 20%, there is a cat. The actual decision takes place afterwards if needed, based on which class had the highest probability.

Another basic way to divide machine learning tasks is based on the data we use. Both classification and regression belong to the so-called *supervised learning*. In supervised learning, the training data consists of pairs, each pair containing the input and the desired output. In other cases, we might only have the inputs and want to find some useful knowledge about the structure of the data. This type of machine learning tasks is called *unsupervised learning* and some examples of when it is used include clustering, outlier detection or dimensionality reduction.

Between the supervised and unsupervised learning types lies a third type, called *semi-supervised learning* (SSL). In this case, for some samples, we only have the input and so we call them *unlabeled data*, while for others, we have the desired output as well as the input and so we call them *labeled data*. When used correctly, the inclusion of unlabeled data in the training process can increase quality of the model's outputs, which is especially useful if we have insufficient amount of labeled data for proper supervised training. SSL is the type of machine learning we will primarily be concerned with for the rest of the thesis. Additional information about it, as well as a description of some of its algorithms, can be found later in this chapter.

For supervised learning to work, the task at hand must meet certain assumptions. One of the most popular ones is called the *smoothness assumption*. For it to be met, it must hold for any two data samples that if their inputs are similar, their outputs should be similar as well. This assumption is absolutely necessary for both supervised learning and SSL, because without it, training data samples provide no indication to which class a new unseen sample should belong.

Once we have a model, we often need to test how well it performs. In classification tasks, this is commonly done by measuring the accuracy of its outputs. However, because the model set its parameters according to the training data, metrics measured on the training data are very often skewed. In severe cases, when the final model fits the training data too well, often to the detriment of its accuracy on unseen data, we talk about a problem called *overfitting*. To get more accurate metrics we usually keep a small portion of the available data apart. This data is called a *test set* or *test data* and is only used for this purpose at the end of the model-creating process. However, we might need to measure the model's performance even during the process in order to pick a model type or properly set its *hyperparameters* (parameters of the model not set during training). For this reason, another small portion of the data is customarily separated, called a *validation set* or *validation data*. Accuracies computed using training data, validation data and test data are called *training accuracy*, *validation accuracy* and *test accuracy* respectively. All the available data put together is referred to as a *dataset*.

In a research paper about a new machine learning technique reporting only its test accuracy is by itself usually not enough to properly illustrate its performance because there would be nothing to compare the figure to. For that reason, one or multiple baselines are often defined. A *baseline* is either a simple algorithm somehow solving the issue or, more often, one of the previously invented algorithms. In SSL, we can talk about a *supervised baseline*, which is often the same model, but trained by a supervised learning algorithm using only the available labeled data. Once we have measured performance metrics of the baseline, we can then compare metrics of our technique to it and prove that we outperformed it, which

Figure 1.1: Examples of images from the SVHN dataset.

.

Figure 1.2: Sample image from the ImageNet dataset.

is often what researchers are trying to achieve.

In this thesis, we will mainly be concerned with image recognition. *Image recognition* is the task of recognizing and identifying objects in an image. Its algorithms can be used for many purposes, from automatic extraction of text from a scanned page, through face recognition up to generating automatic descriptions of images.

1.1.2 Public Datasets

Some datasets have been made publicly available by various researchers and organizations. This allows researchers to easily compare their approaches by simply performing their experiments on and reporting results for a specific dataset or datasets often used by their colleagues. There are many such datasets, especially if we take into account all the different areas machine learning has been applied in. The ones mentioned in the rest of this thesis are:

MNIST (LeCun and Cortes [2010]) is a simple image recognition dataset, containing small grayscale images of handwritten digits. It has been very popular in the past but it is too simple to evaluate modern models.

SVHN (Netzer et al. [2011]) is similar to MNIST in that it contains images of digits, but it poses a far more difficult problem. Its images come from photos of real house numbers. Due to that they contain colors, the digits can be tilted and there could even be irrelevant additional digits in the image. Several examples of the images from the SVHN dataset can be found in figure 1.1.

80 million tiny images (Torralba et al. [2008]), as the name suggests, contains almost eighty million small (resolution of 32x32) images collected from the internet. Each image is loosely labeled with one of seventy-five thousand nouns.

CIFAR-10 and CIFAR-100 (Krizhevsky [2009]) are subsets of the previous dataset, which have been manually labeled. CIFAR-10 contains ten classes, six of them being animals (bird, cat, deer, dog, frog and horse) and the remaining four being means of transportation (airplane, automobile, ship and truck). It contains sixty thousand images, or six thousand for each class. CIFAR-100 has the same overall number of images, but they are divided into a hundred classes, containing not only animals and means of transportation, but also plants, man-made things small and large and natural sceneries. Several examples of CIFAR-10 images can be seen in the top row of figure 4.3.

Open Images Dataset (Kuznetsova et al. [2020]) is a huge labeled dataset containing nine million images and almost twenty thousand possible classes. Moreover, it also contains large amounts of object location annotations (a bounding box and object class), relationships between objects (e.g. "woman playing guitar"), object properties (e.g. material), object actions and much more. Thanks to this abundance of information about the images, many different tasks can be studied using this dataset and even more importantly, they can be studied jointly, on the same data. Its images have a far higher resolution than the CIFAR-10 and CIFAR-100 images, but when scaled down they are very similar, as illustrated by figure 4.5.

ImageNet (Russakovsky et al. [2015]) is another large dataset, containing millions of images belonging to more than a hundred thousand hierarchically organized classes. It is very popular as a benchmark in modern research papers. Just as images of the Open Images Dataset its images are available in higher resolutions. Larger images mean that larger models are needed to process the data, but they also provide more information to solve the classification task. To illustrate the detail available to the models, we provide figure 1.2.

UrbanSound8k (Salamon et al. [2014]) is a small audio dataset composed of less than nine thousand short sound recordings of urban sounds. Each recording is labeled with one of ten classes, for example "jackhammer".

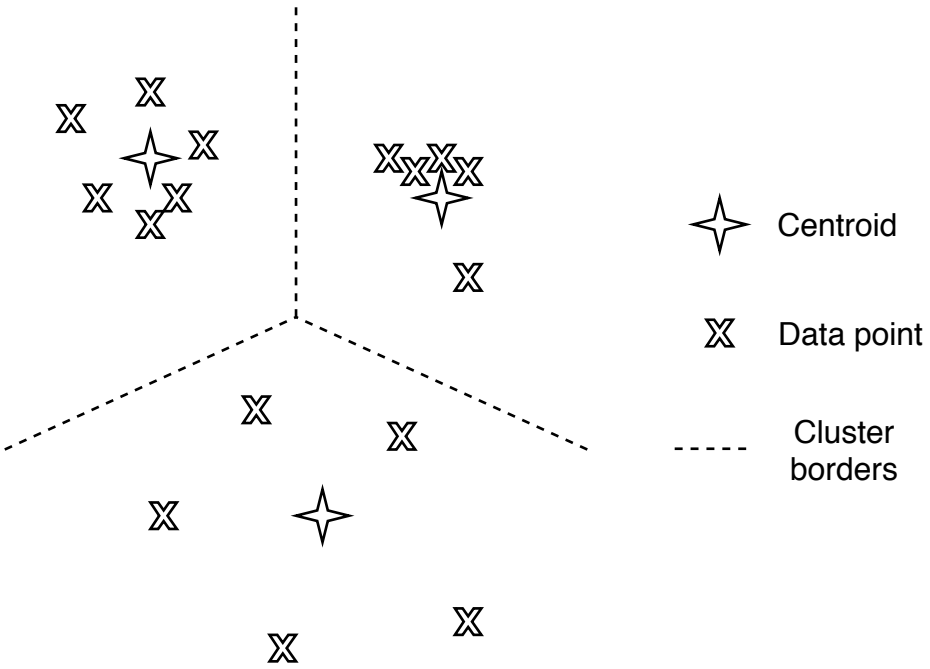


Figure 1.3: Example of the k-means end state. The positions of the centroids and the cluster borders are only approximate.

1.1.3 K-means

As mentioned earlier, *clustering* is one of the tasks for which unsupervised learning can be used. The aim of clustering is to find groups of related data inputs, for example, in order to better understand the structure of the data. There are various clustering algorithms, but for this thesis we only need one of them — the *k-means* algorithm.

K-means is a simple yet very useful algorithm. Its input are the data samples and a number k telling it how many clusters it should find. At the beginning of the algorithm, k points in the *feature space* (space whose dimensions are features of the data samples) are chosen, either randomly or using some heuristic. Those points are called *centroids*. Then the algorithm enters its iterative phase, repeating the same set of steps until a condition is met, for example for a predefined number of iterations. At each iteration, every data point (one of the data inputs projected into the feature space) is assigned to the closest centroid. Every centroid's coordinates are then recalculated as a mean of the coordinates of its data points. At the end, the centroids should roughly represent clusters of data in the feature space. An example of such a result can be seen in figure 1.3.

While k-means is often useful, it can also easily fail to capture the correct distribution of clusters in the data. One of the possible reasons for such a failure is the *curse of dimensionality*. The curse of dimensionality is a well known problem that arises when we are dealing with high-dimensional data. Volume increases exponentially with the number of dimensions, which means that we need exponentially more data to properly represent data density and, even worse, differences

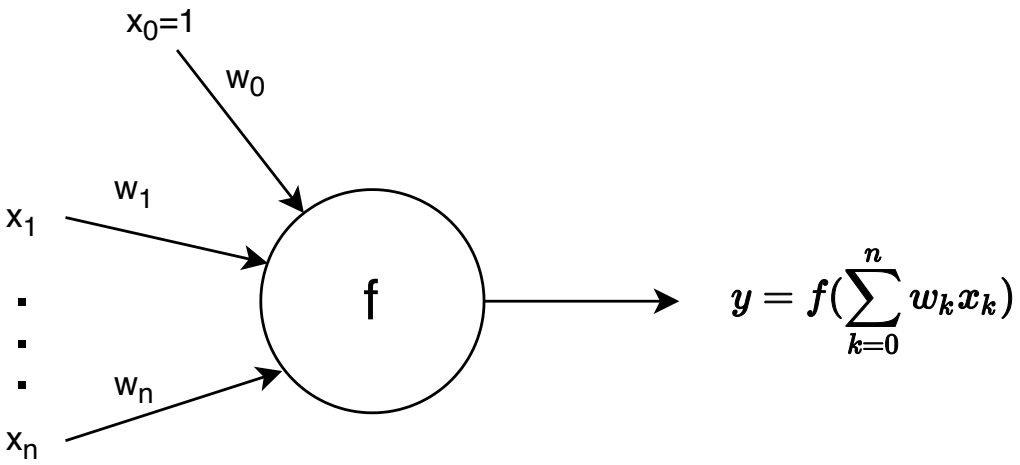


Figure 1.4: Illustration of artificial neuron, along with the formula to compute its output. Inputs are marked x_0 to x_n, input weights w_0 to w_n, activation function f and the output y.

of distances between pairs of data points become minuscule, making them far less useful in any similarity-based algorithm.

1.2 Deep Learning

Artificial neural network (ANN) is a type of machine learning model that takes inspiration from nervous systems of living beings. Just as our nervous system is composed of interconnected neurons, ANN is composed of small processing units, also called neurons. Each of these *artificial neurons* typically has a number of weighted inputs, an activation function and one output which can serve as an input for any number of other neurons. The value on the output of each neuron is computed by applying its *activation function* (for example a sigmoid) to the sum of its weighted inputs. The artificial neuron, along with the computation, is illustrated by figure 1.4. Together the input weights from all the neurons constitute the ANN's parameters. The neurons are usually composed in *layers*, with the first layer receiving inputs from the dataset and the last layer providing the ANN's outputs. When a new input enters the network through the first layer, it is processed by the layers one after the other, each getting its inputs from the outputs of the previous layer, until we reach the output layer, whose outputs can for example define the probability distribution over classes in a classification task.

While the initial dreams of being able to create a real intelligent machine did not pan out and the ANNs had several severe setbacks during the years, they managed to overcome these setbacks and currently represent the state of the art in many areas, such as image recognition or computer translation. They managed this thanks to the idea of *deep learning* and the increasing computational power that allowed this idea to manifest in reality. The basic idea of deep learning is fairly simple — make a network with many layers, where each layer extracts more and more abstract concepts from the outputs of the previous one, up to the last one, which will use the most abstract concepts to solve the task at hand, be it a classification task, a regression task or anything else. To give an example, when

the ANN's task is to recognize digits, one layer could recognize edges, the next one corners, the one after that basic shapes such as half circle, up to the last one, which would decide what digit it is actually seeing.

While the basic idea is very simple, training the deep ANNs is complicated. Modern training algorithms are usually variations of the *backpropagation* algorithm. We will not go into details of the algorithm because its internal workings are not important for understanding this thesis but its basic principle is that an input is passed through the network to get an output, the error on the output is calculated (usually the difference between the actual output and the desired output) and then, as the name suggests, propagated backwards through the network, calculating gradient of the error with respect to the weights in the process. This gradient is then used to change the weights and make the error lower. The function calculating the error is often called a *loss function* or an *error function* and it can contain other terms, aside from the difference between actual and desired output. Those other terms are often used to force some desirable property on the ANN, such as minimizing the amount of neurons it needs to be active for any one input sample.

While this may still seem fairly simple, training deep ANNs is rife with issues, such as the *vanishing gradient problem*. Vanishing gradients occur in ANNs with many layers because some activation functions tend to have gradients of a magnitude smaller than 1. During the calculations of gradients in layers further from the output, these small gradients from the later layers are multiplied, causing the computed gradients to exponentially quickly move towards zero and thus preventing the layers closer to the input layer from effectively learning. To combat this problem and many others, improvements have been devised, such as more sophisticated variations of the backpropagation algorithm, e.g. *Adam* (Kingma and Ba [2015]), or even unique architectures, such as convolutional neural networks (CNNs). However, the problem of reliably training ANN's is still very much open.

Another one of the ANN training issues relevant for this thesis is the aforementioned overfitting. It is very common in deep neural networks because they tend to have a lot of parameters and thus have high *capacity* (the higher the capacity, the more complex functions can be modelled using the model). That makes it easy for the network to memorize the training data without regard for how the network behaves in the areas between them, where the unseen examples we need the ANN to work well for lie. One approach to prevent overfitting is to choose appropriately complex model. However, this can be very difficult and imprecise. Another common approach is called *regularization*. When performing regularization, we usually penalize the model complexity during training, for example by adding a loss term penalizing weight magnitudes.

Another popular way to combat overfitting and generally improve training of ANNs is *batch normalization* (Ioffe and Szegedy [2015]). Batch normalization tries to solve a problem called *internal covariate shift*, which refers to the distribution of inputs of each layer changing whenever the parameters of the previous layer change. This makes it harder for the layers to learn because they have to constantly adapt to the changes of their inputs. To mitigate the issue, batch normalization does exactly what the name suggests — it normalizes outputs of each layer using statistics computed over its outputs on all the data currently being

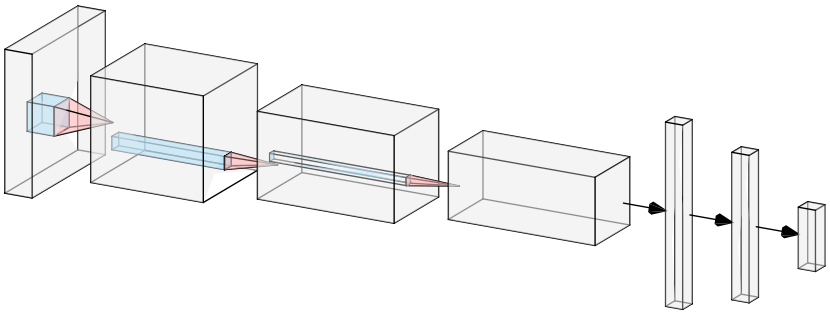


Figure 1.5: An example of basic CNN design, including examples of what parts of the previous layer are used as input. Generated using a tool made by LeNail [2019].

processed (a *mini-batch*) during the training. The actual normalization formula used, as summed up by Zajac et al. [2019], is

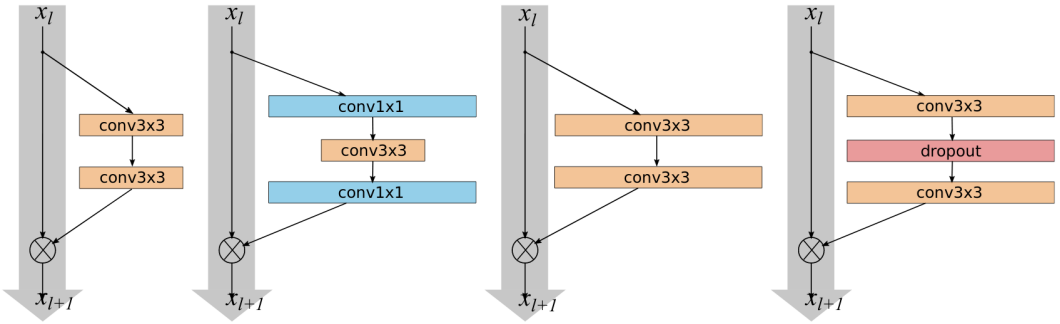
$$\hat{h} = \alpha \frac{h - \mu(h)}{\sigma(h)} + \beta,$$

where h are the outputs of the layer, $\hat{h}$ are the outputs after normalization, α and β are learnable parameters and $\mu(h)$ and $\sigma(h)$ are the statistics (mean and standard deviation) computed on the current mini-batch during training or on the entire training data after the training.

1.2.1 Convolutional Neural Networks

The previously described ANN architecture is called *fully connected neural network*. Fully connected, because every neuron gets its input from all the neurons in the previous layer. That makes sense when there are no spatial relationships between the features, for example when the input is a vector of figures about a person, like their height, weight, blood pressure and so on. However, sometimes we know there is some local structure in the network input. A typical example of this would be in image recognition, where we know that the pixels next to each other are more likely to be part of the same object and therefore are far more relevant to each other than the pixels from the other side of the image. Therefore, we might want to restrict the inputs of each neuron just to those neurons that represent locations close to the location this neuron represents. Moreover, when we are looking for some feature in the image, for example an edge, there is usually no reason to do it differently in different locations. An edge at the top of the picture will be the same as an edge at the bottom. Therefore, we can make all the neurons looking for the same feature share weights.

Together these two improvements constitute the difference between a fully connected neural network and a convolutional one. Switching a fully connected network for a *convolutional neural network* (CNN) drastically reduces the number of parameters along with explicitly forcing the neurons to use the local information, making the training much easier when there is a local structure to exploit.



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

Figure 1.6: Basic types of residual blocks used in residual networks.

Thanks to these advantages, the invention of CNNs revolutionized image recognition, heavily contributing to the current success of deep learning.

The typical basic architecture of a CNN consists of a series of convolutional layers of gradually decreasing dimensions, which look for more and more complex features (for example, first layer could be looking for edges, second layer for circles, third layer for faces and so on), until we reach a small number of fully connected layers that combine the most complex features to decide what the output should be. A *convolutional layer* is a combination of a number of feature maps. Each *feature map* is composed of neurons arranged in a grid, with all neurons within the same feature map sharing the same weights and getting input from the appropriate locations in all the feature maps of the previous layer. By using multiple feature maps, a layer can look for more than one type of a feature and thus, the next layer can combine multiple simpler features when searching for a more complex one. The basic CNN architecture is illustrated by figure 1.5, including the gradually decreasing height and width of the grid and increasing depth of the layers (i.e. number of feature maps). In reality, CNNs tend to be more complicated than that, but aside from the specific model described in the next paragraphs, we will not go into more details as they are irrelevant for this thesis.

Wide Residual Networks

One particular type of convolutional networks we are concerned with are the *wide residual networks* (Zagoruyko and Komodakis [2016]), or WRNs for short. These are a variant of an earlier type called *residual networks* (ResNets), proposed by He et al. [2016].

Theoretically, adding more layers to a network should always lead to the same or better performance, since even if the additional layers were not beneficial, they could always perform identity mapping and thus not change the outputs and achieve the same performance as the shallower network. He et al. [2016] observed that this is not what is happening. In practice, adding more depth to CNNs often led to diminishing performance. Initially, two obvious suspects were identified, the aforementioned overfitting and the vanishing gradient problem. As to overfitting, experiments showed that the added depth decreases not only the test accuracy, but also the training accuracy, which would be increasing if

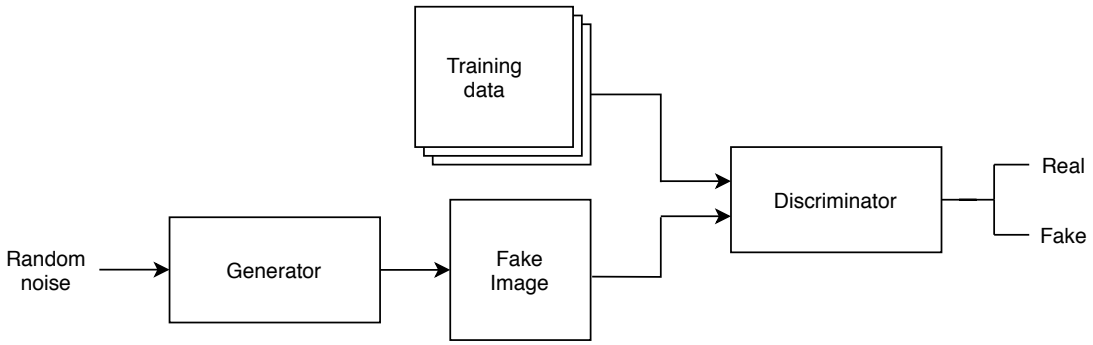


Figure 1.7: Basic training schema of a GAN.

overfitting had really been the cause. He et al. [2016] also performed experiments looking for symptoms of the vanishing gradient issue, but they concluded that as long as the network includes batch normalization, vanishing gradient should not be the culprit.

In order to solve the issue of performance diminishing with depth, He et al. [2016] decided to add residual connections. A *residual connection* is a connection within the neural network allowing it to easily skip some layers when processing inputs. He et al. [2016] added them in such a way that most of the network ended up being composed of *residual blocks*. Each residual block consists of several layers that can be bypassed using a residual connection. Learning an identity mapping in a residual block should be easy because while outputs of the last layer in the residual block are added to the values in the residual connection, the network can learn to always output zero from the layers. Several examples of different residual blocks can be seen in figure 1.6. According to the results of He et al. [2016], adding the residual connections helped with the issue, finally allowing convolutional networks to benefit from having a large number of layers.

However, Zagoruyko and Komodakis [2016] argue that the main part of the ResNet's success comes from the residual connections rather than the depth and that the same performance could be achieved using a smaller number of wider residual blocks (i.e. residual blocks whose convolutional layers have more feature maps). While not decreasing the performance, this would have the added benefit of faster training. Their results supported their assumptions, showing that wider residual networks can outperform even many times deeper narrower ones.

Later in this thesis we will be using a model, whose architecture is called WRN-28-2. This notation means that it is a residual network 28 layers deep and its convolutional layers have twice as many feature maps as the convolutional layers of the original ResNet would have.

1.2.2 Generative Adversarial Networks

Another important deep learning architecture we should mention is called *Generative Adversarial Network* (GAN) and it was proposed by Goodfellow et al. [2014]. A GAN consists of two models, a *generator* and a *discriminator*. A generator is a model that should take random noise on its input and produce an output that is similar to the training data. A discriminator on the other hand takes a data sample on its input and decides whether the sample is from the same

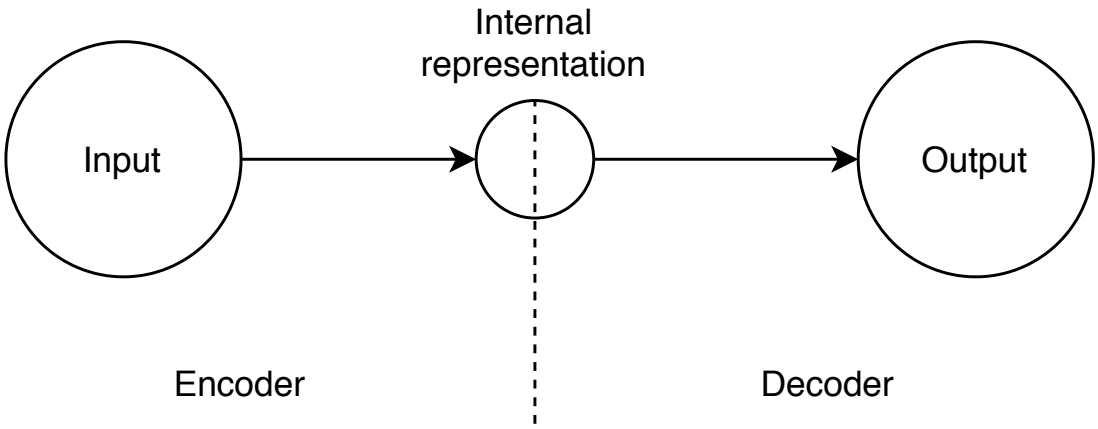


Figure 1.8: Diagram of the basic autoencoder structure.

distribution as the training data or not. During the training, these two models compete with each other, with generator learning to fool the discriminator and the discriminator learning to distinguish between real training data and data generated by the generator. This training schema is illustrated by diagram 1.7. Since this process does not require the labels, GANs can be trained in an unsupervised fashion. Once the training is done, the generator can be used to create artificial data samples.

GANs have been very successful in generating realistic images and many variants and improvements of the original architecture and training process have been developed, but their usability is still severely limited by several problems with their training, described for example by Metz et al. [2017]. The generator might learn to generate only a very little variety of outputs, for example only one out of all the classes in the data. This problem is called *mode collapse*. Even if it does not happen, the training might not converge and even when it does, it is no guarantee that we get high quality outputs out of the generator. Moreover, the primary way to evaluate the outputs at the end of training is a human judgement of the generated data, unlike for example in supervised learning, where we can calculate accuracy and other metrics. All this makes it very difficult and expensive to train GANs, needing not only computational resources, but also human supervision.

1.2.3 Autoencoders

One of the tasks we often find ourselves solving in machine learning is called *dimensionality reduction*. Unsurprisingly, that means decreasing the number of features of the data. One of the reasons we try to do that is that it often decreases both time and space complexity of the training and of running the trained model, along with making the training easier in general. Another reason is that it might allow us to avoid the curse of dimensionality, which makes it very important for the method we designed in this thesis.

There are two main approaches to dimensionality reduction — feature selection and feature extraction. *Feature selection* simply picks the dimensions of the inputs that give us the most useful information. *Feature extraction*, on the other hand, processes the data and creates a brand new set of features. There are many

ways feature extraction can be performed. One of those ways is by using a neural network architecture called an autoencoder.

An *autoencoder* is composed of two parts, an encoder and a decoder. An *encoder* takes the input and transforms it into an internal representation, while a *decoder* takes this internal representation and tries to reconstruct the input data on the output. This basic structure is illustrated by figure 1.8. Thanks to the fact that an autoencoder is simply learning to reconstruct its input on the output, it can once again be trained in an unsupervised fashion, without the need for any labels.

Simply training such a network to perfectly reconstruct its input would be pointless, of course. However, if we impose restrictions on the autoencoder, preventing it from learning a perfect identity function, it will be forced to extract the important information and throw away the rest. One such restriction could be limiting the internal representation to a smaller dimensionality than the input has. This type of an autoencoder is called *undercomplete autoencoder*. When training an undercomplete autoencoder, we have to make sure that its capacity is sufficiently low because otherwise it will simply learn the identity function even if it has fewer dimensions to work with. An example of a realistic undercomplete autoencoder architecture can be seen in figure 4.2. Once we are done training an undercomplete autoencoder, we can use the low-dimensional internal representation as the new data features and thus decrease the data dimensionality while keeping most of the information.

There are multiple other types of autoencoders. Aside from feature extraction, they can also be used for denoising the data, as a part of some SSL algorithms or even to generate new data samples similar to the ones from the training data.

1.2.4 Semi-supervised Learning in Deep Learning

There are many different SSL algorithms, even if we only restrict ourselves to the ones applicable to deep learning, and more are being developed right now. Some are mentioned in the next chapter, where we investigate methods specifically relevant to the problem we chose to tackle. Aside from those, there are several more SSL algorithms that we mention and show some results for (Virtual Adversarial Training, Pseudo-Labeling and Π-model) or use in our own experiments (Mean Teacher) and therefore we should succinctly describe them.

Pseudo-labeling (Lee [2013]) is the simplest of the four. It uses the model to predict classes for the unlabeled data, calling those predictions *pseudo-labels*. Both the labeled data and the unlabeled data with the pseudo-labels are then used for supervised learning weight updates, with the pseudo-labels being recomputed after each update.

The other three (described for example by Oliver et al. [2018]) are more sophisticated. They belong to the consistency regularization family of SSL algorithms. Algorithms of this family generally try to prevent the model from changing its output significantly when the data is subjected to a small realistic change. This is done by adding a new unsupervised (meaning it does not need the labels) term to the loss function. For the rest of the thesis, whenever we speak of data that is used for consistency regularization, we mean specifically for this unsupervised loss term.

Π-model is the simplest of the three. It requires the model to be *stochastic*, meaning that it can produce different outputs for the same input. This is often true about ANNs during training thanks to the use of stochastic regularization techniques. Π-model adds a loss term, which penalizes the differences in outputs of different passes of the same data through the network.

Mean Teacher (Tarvainen and Valpola [2017]) is more complicated. It does not rely on the stochasticity of the model. Instead, it keeps an exponential moving average of the model weights throughout the training and penalizes differences between output of the model with the current weights and the target defined as the output of the model with the exponentially averaged weights.

Virtual Adversarial Training (VAT) directly estimates what small change to the current input would change the output of the model the most and then penalizes the difference of the output for the original input and the output for the changed input.

2. Unfavorably Distributed Data in Semi-supervised Learning

Whereas the previous chapter contains fairly general information, this one focuses specifically on the titular problem of this thesis. Firstly, it specifies the problem we are trying to solve, and secondly, it investigates previously attempted solutions.

2.1 Problem Description

There are many ways the labeled and unlabeled data can have different distributions. Just to give a few examples, the unlabeled data might not contain some of the classes from the labeled data or one or both of their distributions might not be properly balanced or the unlabeled data might contain all the classes, but only contain a specific part of the feature space. Trying to solve multiple issues at once could lead to extremely difficult interpretation of the results or require a number of experiments that would take too much time with our limited computational resources. Because of that, we decided to only tackle one specific issue.

Among other things, Oliver et al. [2018] investigate the effect of differing distributions of labeled and unlabeled data on performance of several common semi-supervised learning algorithms. More precisely, they observe that when a significant portion of unlabeled data belongs to classes not present in the labeled data (and therefore irrelevant for the task being solved), accuracy of classifiers trained by these algorithms suffers, even to the point of being worse than a classifier trained purely by supervised learning using only the available labeled data. This is precisely the problem we will be trying to solve. The issue of differing distributions of labeled and unlabeled data with respect to classes is often referred to as *class distribution mismatch* or class mismatch in literature. Therefore, we will do so from now on too. For the sake of conciseness, we will from now on refer to the classes we are classifying into as *favorable classes*, to all the other classes as *unfavorable classes* and to the data belonging to those two types of classes as *favorable and unfavorable data* respectively. Another important term is *mismatch rate*, defined as:

$$\text{mismatch rate} = \frac{|\text{ unlabeled unfavorable data }|}{|\text{ unlabeled unfavorable data }| + |\text{ all favorable data }|}$$

It will be given either as a percentage (e.g. 75%) or as a decimal number (e.g. 0.75).

While this issue has to be artificially simulated when working with publicly available datasets commonly used to judge quality of machine learning algorithms, it is definitely a realistic one in practice. Imagine needing a classifier able to distinguish various forest terrestrial animals in images. To obtain the data necessary to train the desired classifier, you set up camera traps triggered by movement. After some time, you collect the images and pay someone to label a portion of them, wanting to use a semi-supervised learning algorithm to get your classifier without the expense of labelling all the images. However, soon you find out that the cameras were triggered not only by terrestrial animals, but also by humans

passing by, birds, bats, large insects, or even branches of the trees swaying in the wind. In the labeled data, all these cases can be manually filtered out, but doing so for the unlabeled data would be almost as labor intensive as actually labelling all of it and therefore too expensive. Even though this issue might sometimes be solvable by simply improving the data collection practices, at other times that might be impossible or we might even receive the data without being able to influence the data collection at all. Therefore, semi-supervised learning algorithms should be robust in respect to this problem.

Effects of class mismatch are further studied by Look and Riedelbauch [2019]. Whereas Oliver et al. [2018] worked with CIFAR-10 (for a description of their experimental setup, see section 4.2), Look and Riedelbauch [2019] perform their experiments on two different datasets — MNIST and UrbanSound8k. They observe the detrimental effects of class mismatch while performing experiments on MNIST, but observe little or no effects with UrbanSound8k. They conclude that adding unlabeled data that does not necessarily belong to the classes that are being classified into can be beneficial, but whether it will be beneficial or harmful in the end might depend on how similar the additional classes are to the classes we classify into. Therefore, if we could ensure that the additional data (even if irrelevant strictly speaking) is very similar to the data we want to classify, we might be able to negate the detrimental effects. Experiments done by Zajac et al. [2019] seem to further support this. They show that the detrimental effects of class distribution mismatch heavily depend on the particular dataset and task. At first, they study the effects on CIFAR-10, just as Oliver et al. [2018] did. The results there are very similar, again showing that applying SSL algorithms to a dataset with high mismatch rate can lead to results worse than the supervised baseline had. Then they perform experiments for two ImageNet subsets. In the first case, they randomly select twenty of the ImageNet classes to use for the labeled dataset. In this case, SSL significantly outperformed the supervised baseline even at mismatch rates of 75% and 100% (meaning there were very little or no favorable unlabeled data). This confirms that even completely irrelevant unlabeled data can theoretically be beneficial to the training process. In the second case, they picked eight animal classes and eight non-animal classes and only used the animal classes as favorable data. There, the results were similar to the CIFAR-10 results, again showing severe detrimental effects of class mismatch, with the supervised baseline outperforming SSL from a mismatch rate of 50% upwards. Since in this case, they ensured that the unfavorable classes were very different when compared to the favorable ones, while in the previous case, there could be many unfavorable data samples similar to the favorable data, this experiment seems to support the conclusion that the magnitude of the difference between favorable and unfavorable data matters very much to the final results.

Another research paper with important information for this thesis is Uesato et al. [2019]. While they explore *adversarial robustness* (robustness against examples specifically designed to confuse the model), their conclusions might still be valid even for non-adversarial setting, such as ours. They show that even though using data from the CIFAR-10 dataset as unlabeled data is better than using images from the 80 Million Tiny Images dataset, the second mentioned do still help when they are properly filtered. However, for this filtration, they use a classifier trained on the whole CIFAR-10 dataset, which is not applicable for

this thesis, because we do not want to count on having so much labeled data. Nevertheless, it suggests that accurately filtering the data should help with the issue.

While Oliver et al. [2018] show class mismatch causing problems in VAT, Mean Teacher, Pseudo-Labeling and Π-model, it is not exclusive to these algorithms. Kaizuka et al. [2019] show its detrimental effects on their proposed SSL method, ROI regularization. Chen et al. [2020] perform experiments similar to the ones done by Oliver et al. [2018], but add Temporal Ensembling (Laine and Aila [2017]), Stochastic Weight Averaging (Athiwaratkun et al. [2019]) and their own algorithm, called *Uncertainty-Aware Self-Distillation* (UASD), whose design and results will be described in the next section. The detrimental effects are present in some form and degree for all the mentioned algorithms and almost always follow the logical trend of worsening with increasing mismatch rate, but there are definite differences in how the algorithms are influenced by the issue, especially when looking at results across several datasets or tasks. For example, one algorithm might achieve better results than another on one task, even beating the supervised baseline, but get outperformed on another task. Even their behavior with regards to increasing mismatch rate varies. One algorithm might experience the highest performance degradation when going from 25% to 50%, while another going from 50% to 75%. This suggests that in order to achieve the best possible results, application of specific algorithms or other methods of solving the issue should depend on properties of the specific task.

Not much information on what precisely causes the issue can be found. Chen et al. [2020] claim that the detrimental effects of class mismatch are mainly caused by an *overconfidence issue* of deep neural networks which in this setting causes models to confidently assign unfavorable data to one of the favorable classes, leading to far more significant incorrect influence on model weights than if the model was unsure about the label. This claim is supported by their own results which show that an algorithm designed around keeping these predictions less confident suffers from the accuracy deterioration far less than algorithms suffering from the overconfidence issue. Nevertheless, the assertion needs to be tested on more tasks and more types of SSL algorithms, especially since it is currently only empirically backed for image data.

2.2 Previously Attempted Solutions

Even though it does not seem surprising that having a classifier learn from irrelevant data can worsen its performance, the problem appears to be understudied by researchers so far as mentioned for example by Oliver et al. [2018]. Nevertheless, some efforts to create semi-supervised learning algorithms robust in respect to this issue can be found in literature. In this section, we describe these attempts to arrive at a solution. Aside from those mentioned in the following paragraphs, we found several research papers claiming their model should be more robust in relation to class mismatch than others. However, those claims were not followed upon in those papers, which is why we decided not to include them in this section, since without results the claims cannot be verified and no information usable for the rest of the thesis can be extracted.

The most common attempts at solving the problems caused by class distri-

bution mismatch we identified during our investigation of the available literature resemble the *self-learning* SSL approach. As described by van Engelen and Hoos [2019], self-learning consists of iteratively training a classifier by supervised training and, in each iteration, adding some of the unlabeled data samples classified by the current classifier to the labeled data. When used as a solution for class mismatch, the data samples are not added to the labeled data. Instead, they are still used as unlabeled data, for example for consistency regularization, but they are only used when the current classifier is confident enough that they belong to one of the favorable classes. Effectiveness of this approach is examined by Nair et al. [2019]. They call their implementation *out-of-distribution* masking and they compute the required confidence level on each training step so that a predefined percentage of the unlabeled data is used. Results of their experiments with experimental setup similar to the ones performed by Oliver et al. [2018] show that when using this approach, their semi-supervised learning algorithm (*RealMix*) is always better than the supervised baseline and that specifically the out-of-distribution masking significantly improves the results. However, an important thing to note is that in their experimental setup, even Mean Teacher outperformed the supervised baseline in all cases except for 100% mismatch. Therefore, while encouraging as to the possibility of overcoming the issue, their results should be taken with a grain of salt. A common and much simpler variant of the same basic idea is described by Xie et al. [2019]. They only pick the unlabeled data that are to be used once, at the beginning of the training, using a classifier trained purely through supervised training.

Another SSL algorithm incorporating the same idea is the aforementioned UASD, developed by Chen et al. [2020]. They decide which samples of the unlabeled data should be used for training at the beginning of every epoch. First, they calculate average confidence of the predicted class when applying the current model to the validation data, and then they use it as a threshold. If the model is more confident than this threshold in its prediction about a particular unlabeled data sample, the sample is used for training in that epoch. However, according to their ablation study, this filtering is not the most significant part of UASD in regard to combating class mismatch. As mentioned in the previous section, they believe the detrimental effects are primarily caused by overconfident predictions of the other algorithms. Because of that, they use a process similar to the one previously described for Mean Teacher, but with one significant difference — instead of using exponential moving average of weights to get the target, they compute the target as arithmetic mean of predictions across all the epochs so far. This puts far less emphasis on the most recent predictions, which might be confident even when wrong, and allows them to be balanced out by different predictions made in the past, leading to far less pronounced influence of the unfavorable data on the weights. Their results are very positive, consistently outperforming the supervised baseline and all the other SSL algorithms they tested on any level of mismatch, in three different cases, using data from CIFAR-10, CIFAR-100 and ImageNet. While this shows that the problem can be beaten, it is still only shown to work for image recognition tasks and the method they use might not be applicable when a different kind (meaning not consistency regularization) of SSL algorithm would be preferable.

Zajac et al. [2019] propose their own attempt at mitigating the issue, which can

be applied whenever we use a neural network with batch normalization layers. They call their approach *Split Batch Normalization*, or Split-BN for short. It consists of computing the batch normalization statistics separately for the labeled and unlabeled data, with the justification that when the labeled and unlabeled distributions differ, these statistics should differ too or they will not be accurate for either. They perform several experiments with class mismatch, as described in the previous section. For these experiments, they use Mean Teacher and VAT with and without the Split-BN. Their conclusions are that Split-BN almost always improves the results when compared to using normal batch normalization and that in many of the cases where SSL under the effects of class mismatch performs worse than the supervised baseline, it improves the performance to the point of being roughly equal to the supervised baseline. These are very positive results, especially for such a simple change, because they almost make using SSL safe when compared to simply using supervised training. However, a real solution has to be capable of beating the supervised baseline, not just matching it.

An important observation we made while surveying the available literature is that currently, the different approaches often cannot be directly compared to each other due to differences in their experimental setup, ranging from training different model architectures to using completely different ways of simulating class mismatch, from only picking certain classes to be in the unlabeled data, up to using another dataset to provide different images. Just as Oliver et al. [2018] mention for SSL in general, a common experimental setup would be most useful, allowing direct comparison of the different approaches.

3. Unfavorable Data Filtering

In this chapter, we first sum up some of the findings from previous chapter relevant for the design of our solution of the class distribution mismatch issue, use them to formulate requirements for the solution and then describe the solution itself.

3.1 Findings and Requirements

As was described in the previous chapter, the most common approach to dealing with class distribution mismatch works by classifying each unlabeled example that is to be used and then deciding whether or not it should be used based on the results. While this approach might be a workable one, it might also create a positive feedback loop, gradually increasing bias of the classifier, because the outputs of the classifier are used to determine how the classifier will be trained. Moreover, in settings where only a minuscule amount of labeled data is available, this effect would be even more pronounced, possibly making the solution completely untenable.

However, the core idea (filtering out the irrelevant data) is a sound one. As shown by Uesato et al. [2019], if you correctly filter out the irrelevant data, semi-supervised learning becomes beneficial again. This is further reinforced by results of our experiments, described in section 4.3 and even more by the results presented by Look and Riedelbauch [2019], which suggest that you might not even need to filter all the unfavorable data out, because unfavorable data similar to the favorable data can even be beneficial to the SSL. Therefore, the solution we propose in this thesis should work on the basis of filtering out the irrelevant data.

That's why we try to create an approach that would filter out the irrelevant data, while minimizing the probability that the bias of improperly trained classifier will harm the learning process. Moreover, some of the other methods for solving the issue only work for a specific SSL algorithm. We, on the other hand, would like to create a technique that can be used with any semi-supervised learning algorithm.

These requirements impose some restrictions on the possible solutions. First of all, most of the machine learning involved should be unsupervised, to minimize the bias caused by not having enough labeled data. Second, it needs to be able to decide, whether a particular data sample is relevant to the task at hand or not. And third, it should work independently of the specific semi-supervised learning algorithm being used.

To satisfy all of these, we designed a technique that processes the dataset before the actual semi-supervised learning algorithm and removes examples that are deemed not relevant enough.

3.2 Basic Design

Our basic idea is to first extract features from the data samples and then use those features along with a similarity-based filter to determine which samples

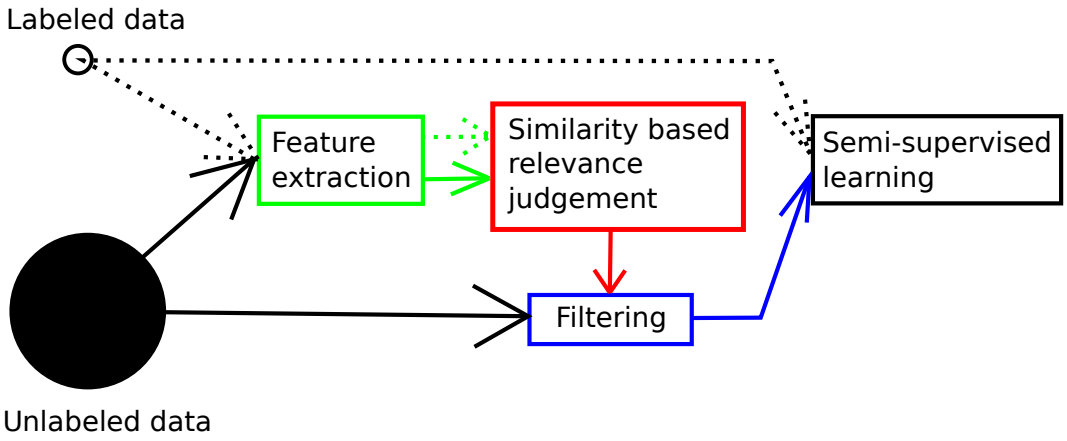


Figure 3.1: Basic design of UDF.

from the unlabeled data are probably relevant (come from the classes present in the labeled data) and which are irrelevant and should be filtered out. This process is illustrated by diagram 3.1. We call it *Unfavorable Data Filtering*, or UDF for short.

Since feature extraction can usually be done in an unsupervised way and the data is being filtered with no input from the subsequent SSL algorithm, all the previously identified requirements and restrictions are fulfilled by this solution. Moreover, as long as a way to extract features of good quality from a data sample exists, it does not matter what the data is. Therefore, basically the same solution should be applicable to various tasks, no matter whether they involve working with images, texts or other forms of data.

The previous section mentioned that removing unfavorable samples from the unlabeled data significantly improves the performance of the subsequent SSL. Of course, in our setting, we do not have access to all the labels and therefore we cannot do the filtering with perfect certainty. However, so long as the previously described smoothness assumption holds (which is also a necessary condition for SSL to work at all, according to Chapelle et al. [2006]), we should have a higher probability of estimating whether a sample belongs to one of the favorable classes using similarity to labeled data than if we did so by pure chance and therefore be able to decrease the class mismatch degree by the proposed filtering, thus improving the performance of the SSL. The extent of this improvement depends on the quality of the feature extraction and on the choice of the similarity-based filter and therefore, both of those should be chosen carefully. Moreover, while filtering based mostly on unsupervised learning might never be as precise as filtering while knowing the actual labels, it might also provide one advantage. Some images can provide useful information for solving the problem at hand, even if they do not belong to any of the classified labels. For example, an image of a plane could help the classifier learn more about the general shape of flying objects with wings and thus help the classifier better distinguish birds from terrestrial animals.

3.3 Feature Extraction

Images in the CIFAR-10 dataset have a resolution of 32x32. Moreover, each of those 1024 pixels has three color channels, making the overall dimensionality 3072. Due to the curse of dimensionality, similarity-based filters would not work well with that kind of data. Therefore we need to significantly decrease the number of dimensions using feature extraction techniques. There are many of those available for image data (Kumar and Bhatia [2014]), but we chose to mainly concentrate on the options from the field of deep learning. The three options we considered more closely are autoencoders, generative adversarial networks and the supervised baseline.

The simplest of those would be using the supervised baseline. Since convolutional neural networks gradually create more and more complex features, often with decreasing dimensionality, we could train a convolutional network, even one with the same architecture we will later use for SSL, and take the features it identifies (e.g. the last layer before the output) in order to make its decision. This way of obtaining features has the added advantage that, at least in research, we often create just such a network — the aforementioned supervised baseline. While that is certainly a reasonable option, it has two severe downsides. Firstly, it would limit the usability of our approach in cases with minimal amounts of labeled data. And secondly, it would make our solution very similar in the way it works to the previously described already attempted ones.

Earlier in this thesis, we mentioned that a GAN is a neural architecture consisting of two neural networks, generator and discriminator, competing with each other during training. When working with images, a discriminator can be a convolutional network. Once we are done training a GAN, we can extract the features from the discriminator in the same way as we described in the previous paragraph for supervised baseline. As GANs can be trained using unsupervised learning and they are significantly different from the previously attempted solutions, it would solve both the disadvantages of the previous option. However, as GANs are difficult to train and the result of their training is uncertain and very hard to objectively evaluate without interaction with humans, they are still not the option we chose.

The third and last option examined in detail are autoencoders. They are not overly complicated, their training can be objectively evaluated even without human input and they can be trained using just unlabeled data. Those are the reasons we chose to work with autoencoders in our experiments.

As mentioned before, there are multiple types of autoencoders. We decided to use undercomplete autoencoders since their intended function coincides with what we need — they decrease the dimensionality by extracting the most important features from the data. Since these autoencoders learn in an unsupervised fashion, we can use all the training data to train the autoencoder, trying to reconstruct each input image at the output, with a bottleneck in the middle from which we can extract the required features. Since autoencoders are neural networks, we can select from a huge number of different architectures. To select the best architecture, we performed many experiments. Information about their execution and conclusions can be found in section 4.4.

3.4 Similarity-Based Filter

The second, more complicated part of UDF is the similarity-based filter. Inputs of the filter are the unlabeled training images transformed into a low-dimensional feature space by the feature extractor from the previous section. Since similarity measures are often sensitive to variable scales, we standardize the features before using them. Outputs of the filter are decisions about whether each specific unlabeled data sample belongs to the favorable classes or to the unfavorable ones according to a decision process using a similarity measure between the transformed images. To provide the filter with information about what a favorable or unfavorable data sample should look like, we also allow it to access features extracted from the labeled training images and their labels. However, to improve the chances of the algorithm working in scenarios where we do not have much labeled data, we decided to only use the information about whether a sample belongs to the favorable or unfavorable classes, not to which specific class it belongs. This gives us more samples for the two possible values than we would have if we worked with class labels. Moreover, while it is quite realistic that we have samples where we know that they do not belong to the favorable classes because these would be identified during the labeling process (those doing the labeling would encounter them and discard them, thus telling us they are unfavorable), it is probably not realistic to expect the unfavorable data to be divisible into several neatly separated classes.

We tried several different ways of constructing the filter, ranging from searching for the closest labeled data sample, through a more complex one based on finding representatives of the labeled favorable data and comparing the unlabeled data to them, up to the final filter with the best performance, which we named *IterativeDoubleKMeansFilter*. The basic principle of IterativeDoubleKMeansFilter is still searching for representatives of the labeled data and then using them to make decisions about the unlabeled data samples. However, instead of simply checking whether the data sample is close to representatives of the favorable data, we now check whether it is closer to the favorable representatives than to the unfavorable ones and use an iterative process to fine-tune the representatives using the unlabeled data.

The final algorithm can be divided into three steps:

First step is processing the labeled data available to the filter. Representatives for the favorable and the unfavorable labeled data are defined as centroids found by clustering the respective data using the k-means algorithm. While these representatives are already good enough to change the mismatch rate by tens of percentage points, decisions made in the third step using them tend to produce false negatives on many favorable data samples, easily leading to the loss of half of the favorable unlabeled data in our preliminary experiments. To mitigate this loss, we added the next step.

Second step consists of using the previously determined representatives and the unlabeled data in an iterative fashion to get even better representatives. Each iteration consists of taking the representatives found so far and using very strict formulas to find unlabeled data samples that very probably belong to the favorable data and data samples that very probably belong to

the unfavorable data. These samples are then used to augment the labeled favorable and unfavorable data respectively for the purpose of finding better representatives using the same method as in the first step. This repeats for a predetermined number of iterations, slowly increasing the size of augmented labeled data and hopefully leading to better representatives. Our preliminary experiments showed that this step not only decreases the mismatch rate, but, even more importantly, increases the amount of favorable unlabeled data preserved for the SSL.

Third step contains the actual selection of the filtered training data using the found representatives of favorable and unfavorable data. By using a far more lenient formula than the second step, the third step is designed to find as many favorable unlabeled data samples as possible, while excluding the obviously unfavorable ones, thus trying to maximize the amount of favorable data we preserve while also significantly decreasing the mismatch rate.

What remains to be properly described are the exact formulas used to select samples in the second and the third step of the algorithm. To accurately describe them, we first need to define some terms. Each labeled data sample d from the unlabeled data belongs to one favorable centroid $c_f(d)$, and one unfavorable centroid $c_u(d)$, which are the closest centroid determined by k-means used on the favorable data and the closest centroid determined by k-means used on the unfavorable data respectively. Each centroid c has a *maximal distance* $md(c)$. This distance is calculated as

$$md(c) = dc_c(\lfloor r * a_c \rfloor),$$

where a_c is the amount of labeled data samples belonging to c according to the k-means algorithm, $dc_c(k)$ is the distance from c to the k-th least distant labeled data sample belonging to c, and $r \in [0, 1]$ is a predetermined coefficient. The maximal distance is defined this way to allow us to easily and intuitively modify how strict or permissive the later formulas will be. For example, when we set the r coefficient to be 0.5, it means that half of the labeled data that was previously belonging to the centroid would be judged in the second step of the filter algorithm as being represented by it and therefore probably having the same label. To be precise, *d is represented by c*, if $distance(c, d) \leq md(c)$, where $distance(c, d)$ is the distance between c and d according to the currently used distance measure.

With all the necessary values defined, we can get to the formula from the second step. All its conditions are based on the idea that the more similar a data sample is to either the favorable or the unfavorable data, the higher its chances of being favorable or unfavorable are. Since it only makes sense to compare the data sample to the closest favorable and unfavorable data samples and calculating similarity of the data sample to many others would take a long time, we use the centroids to represent the favorable and unfavorable data distributions. For each data sample d, we decide:

1. Whether it is represented by $c_f(d)$. This condition disqualifies data samples that are too far from the favorable centroids (and thus too dissimilar to the favorable data) and we therefore cannot justify considering them favorable.

2. Whether it is not represented by $c_u(d)$. This condition disqualifies data samples that are too close to an unfavorable centroid (and thus too similar to the unfavorable data) and therefore have a significant chance of being unfavorable.

3. Whether:

$$\frac{\ln(i(c_f(d)) + 1)}{\ln(i(c_u(d)) + 1)} * distance(c_u(d), d) < distance(c_f(d), d),$$

where $i(c)$ is the amount of labeled data samples belonging to centroid c according to the last k-means run. This condition is a direct implementation of the idea that if a data sample is more similar to the favorable data than to the unfavorable data, it has a higher chance of being favorable than of being unfavorable. The amounts of data samples belonging to the centroids ($i(c)$) are taken into account to make more accurate decisions in regions where the favorable and unfavorable data overlap. If a data sample is a similar distance from a favorable and an unfavorable centroid, it is more probable it belongs to the side which has a higher density in the area. The logarithms are present to scale down the effect, because one centroid can have orders of magnitude more data belonging to it than the other, yet we still do not want it to always dominate.

If all three conditions hold, the data sample is accepted to the augmented labeled data for the purpose of calculating centroids as a favorable data sample. To decide the same for the unfavorable side requires basically the same conditions, with $c_u(d)$ and $c_f(d)$ switched.

The formula from the third step is far less complicated. It is actually just the third condition of the formula from the second step, making it more permissive, to make sure we do not drop too much favorable unlabeled data during the filtering. Workings of this formula, along with the previous one, are illustrated by diagram 3.2.

Same as in the case of the feature extractor, we performed many preliminary experiments with the filter to set its parameters to reasonable values. These experiments and their conclusions are described in section 4.4.

3.5 Diagram

In order to allow easier understanding of the entire operation of UDF, we created diagram 3.3, depicting flow of the data during the process. The “2nd step conditions” and “3rd step conditions” of course refer to the conditions from the second and third step formulas, described in the previous section.

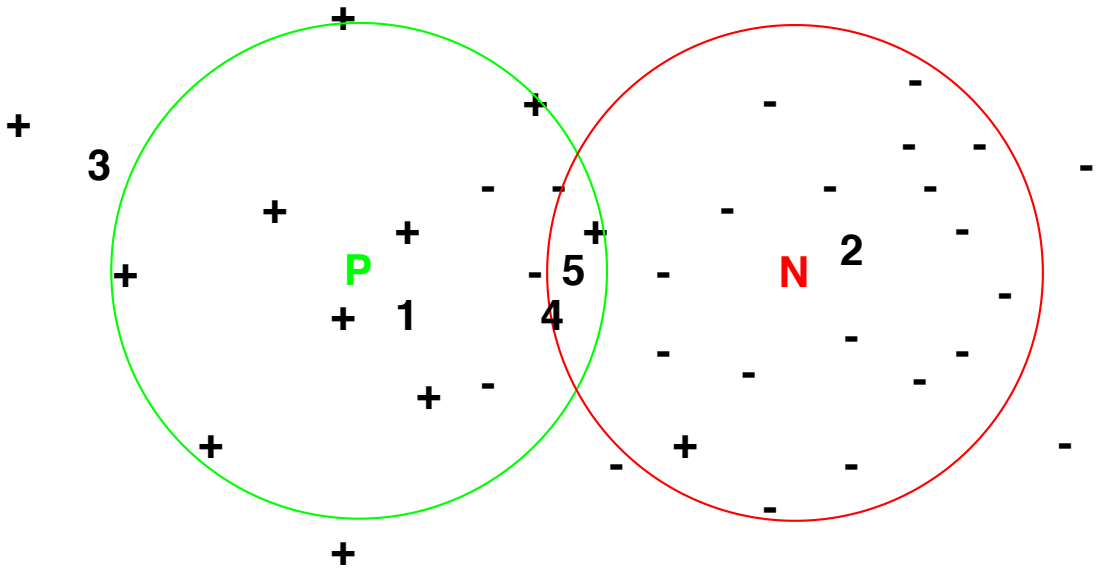


Figure 3.2: An example of how our filter works. The + signs mark the labeled favorable data samples, the - signs mark the labeled unfavorable data samples. The P and N letters mark the favorable and unfavorable centroid respectively, with the appropriately colored circles around them encompassing 75% of the labeled data belonging to each centroid. Thus, the circles illustrate the maximal distance of each centroid. The digits mark unlabeled samples that need to be filtered. In the second step of the algorithm, the rules are more strict. Thus, while sample 1 will be considered as favorable and sample 2 as unfavorable, sample 3 will not be considered favorable, because it is not within the maximal distance of the favorable centroid. Moreover, neither 4 nor 5 will be considered favorable or unfavorable, because they are represented by both centroids. The third step is more permissive, allowing 3 to be considered favorable. However, 4 and 5 will be filtered out, because the conditions will consider them more likely to be unfavorable. Sample 5 is equidistant from the two centroids and therefore cannot be considered favorable. The unfavorable centroid has more samples belonging to it, which is why it is more powerful, and even samples that are closer to the favorable centroid, like sample 4, might be filtered out (probably correctly, considering the shown distribution of the data).

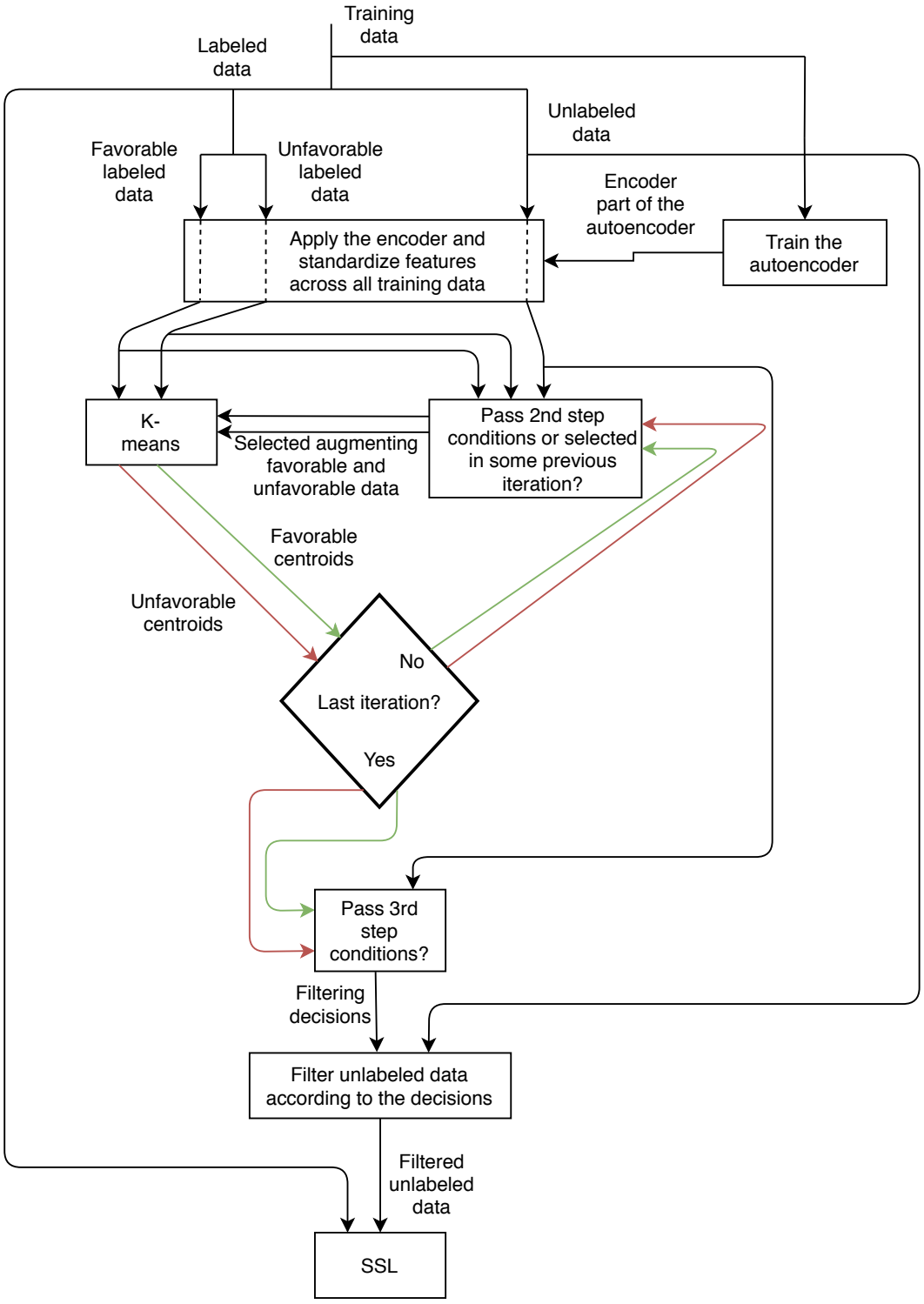


Figure 3.3: A diagram detailing the entire process of UDF.

4. Experiments and Their Evaluation

In the previous chapters, we described the problem of semi-supervised learning under class distribution mismatch and proposed a novel solution. The effectiveness of a solution has to be experimentally verified and measured. To that end, we performed many experiments. Their results and conclusions based on those results are described in this chapter.

Tables in this chapter often use the abbreviations "M.T.", "S.B." and "M.r.", meaning "Mean Teacher", "Supervised Baseline" and "Mismatch rate" respectively.

4.1 Source Code

Most of the source code used to perform the experiments is written in Python, using the Tensorflow library for implementation of neural networks. The experiments were performed using Python 3.7.6 and Tensorflow 1.13.1 (in a GPU version) on Ubuntu 19.10 operating system. They should also be runnable with different configurations, especially with regards to the Python version and the operating system. Many other common libraries were used, one example for all would be NumPy, a Python library focused on computations.

The source codes are included in the files attached to this thesis. Instructions to use the scripts to repeat the experiments are in the README file in the root directory of the source codes folder.

4.2 Experimental Setup

One of the problems with SSL algorithm evaluation identified by Oliver et al. [2018] was that different research papers often use different neural network architectures, different kinds of data preprocessing and so on. Due to this variance, results for different approaches often cannot be directly compared. To solve this issue, they perform all their experiments using a common experimental setup. They made source code for this setup public, allowing other researchers to easily compare results with theirs. That is why we decided to use this setup as well. Details about the setup can be found in the aforementioned paper. Most importantly, it uses a WRN-28-2 architecture with batch normalization and leaky ReLU (Maas et al. [2013]) activation functions.

Oliver et al. [2018] perform their experiments on two image datasets, CIFAR-10 and SVHN. However, the class mismatch is only investigated on CIFAR-10, which is why we chose to use that one as well (there are exceptions explicitly mentioned in the further text). The mismatch in the paper is investigated at four different mismatch rates — 25%, 50%, 75%, 100%, meaning that 25%, 50%, 75% or 100% of the data used for consistency regularization (not only unlabeled data are used, but also the labeled data, but without using their labels) does not belong to the favorable classes. Due to the limited computational resources

available we initially decided to perform the experiments on just one of those levels. We picked 75% because it is the level where the issue should be very noticeable according to the results presented by Oliver et al. [2018] (also included in this thesis as figure 4.6), but unlike with 100% we can be sure there still are valuable unlabeled data samples that can be used to train the classifier. As can be seen in the next sections, we ended up performing the experiments for other mismatch rates as well, for reasons described later. The same thought process led us to restrict the number of SSL algorithms we use in our experiments to just one from the original four Oliver et al. [2018] use to measure their results for class mismatch. We picked Mean Teacher, which has one of the better results out of the algorithms they tested, noticeably suffers from the class mismatch issue and is not too computationally expensive, allowing us to perform enough experiments.

While most of the experimental setup is the same as in Oliver et al. [2018], we made one significant change. As described before, the CIFAR-10 dataset has ten classes, six of them animals, the rest being means of transportation. The paper simulates class mismatch by only classifying into the six animal classes and adding data belonging to the other four into the unlabeled data to create the desired mismatch level. However, this is done by only allowing data from four classes into the unlabeled data with the mismatch level being controlled by how many of those classes belong to the animal classes and how many belong to the means of transportation. For example, for the 75% mismatch level, only one of the favorable classes is present in the unlabeled data. We think this artificially increases the impact of class mismatch and a more realistic scenario would be picking the data randomly from all the available classes while making sure the level of mismatch is the desired (for example) 75%. For this reason, we would pick 25% of the data used for consistency regularization from the six animal classes, making sure each class is represented by approximately the same number of samples, and the remaining 75% from the means of transportation classes in the same way. Aside from the data from favorable classes, we also kept some labeled data from the unfavorable classes (400 per class, the same as for the favorable classes) to be used by the filtering algorithm. We consider keeping that data realistic, because in the process of labeling the data, samples not from favorable classes would be encountered and marked as such. In this case, we know the precise labels of course, but we do not use them, so the results should be realistic. In the end, we are working with similar amounts of data as Oliver et al. [2018], but the data used for consistency regularization is distributed among the classes far more equally.

Same as Oliver et al. [2018], we train the model for 500000 steps (each step consisting of processing a batch of 50 labeled and 50 unlabeled samples). During the training, the model is regularly saved, to allow for metrics calculations at different points in the training. Unless specified otherwise, all the accuracies reported in this thesis are test accuracies, calculated based on the results of a saved model with the highest validation accuracy. This also copies the way evaluation is done by Oliver et al. [2018]. Our tables report mean accuracy and its difference to the mean accuracy of the supervised baseline, standard deviation, minimal achieved accuracy and maximal achieved accuracy, computed using accuracy values from multiple runs, using different datasets picked from the CIFAR-10 dataset according to the required configuration. All the hyperparameters and the opti-

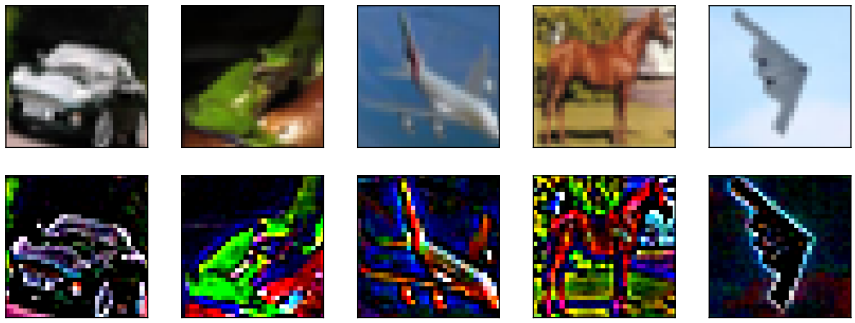


Figure 4.1: Images showing the effect of ZCA normalization. The top row contains original CIFAR-10 images, while the bottom row contains the same images after applying ZCA normalization.

mizer (Adam) are identical with the original as well. While tuning them for the specific case could produce better results, we decided not to do it for two reasons. One, it would be computationally expensive. And two, it would make the results unrealistic because, as mentioned by Oliver et al. [2018], the size of the validation set required to properly tune the hyperparameters far exceeds what would be realistic, as the validation set should usually be far smaller than the training set. On that note, we should disclose that, same as Oliver et al. [2018], the validation set we use is unrealistically large, up to the point of being slightly larger than the labeled training data. While it would be better to use a small validation set if we wanted to provide evidence that the technique can be used in practice, we are mostly trying to find basic principles about what would and what would not work in general, which is where more precise estimations using a larger validation set are advantageous. Also, just as them, we perform ZCA normalization (Pal and Sudeep [2016]) on the CIFAR-10 images before training the classifier. *ZCA normalization* is a whitening transformation that (among other effects) makes the edges in the images more prominent, as can be seen in figure 4.1, making it easier for the model to learn.

4.3 Baseline Experiments

Due to the possible differences in achieved results caused by different versions and implementations of the required software and also due to the changes in the way datasets for the experiments are constructed described in the previous section, accuracy values from Oliver et al. [2018] cannot be directly used as a baseline. Therefore, we performed our own baseline experiments. There are three types of them. The first one is a completely supervised baseline, only using the labeled portion of the available data. The second one is using Mean Teacher, training on all the available unlabeled data, even the data belonging to the four means of transportation classes. This type is called Complete Data in the tables. The last one is using Mean Teacher, training the model with the same unlabeled data but after filtering out samples belonging to the unfavorable classes. This one we refer to as Filtered Data. Results of these experiments are shown in table 4.1.

M.r., experiment type	Mean(-S.B. mean)	Std	Min	Max
N/A, Supervised Baseline	0.7688	0.0045	0.7632	0.7760
0.64, M.T., Complete Data	0.7611(-0.0076)	0.0037	0.7538	0.7655
0.64, M.T., Filtered Data	0.7883(0.0195)	0.0068	0.7765	0.7969
0.75, M.T., Complete Data	0.7507(-0.0181)	0.0056	0.7432	0.7567
0.75, M.T., Filtered Data	0.7783(0.0095)	0.0049	0.7717	0.7858

Table 4.1: Test accuracies achieved during the baseline experiments.

Aside from providing accuracies that can later be directly compared to accuracies achieved using our solution, several important things can be seen from these results. First of all, results of the supervised baseline are slightly more positive than the ones reported by Oliver et al. [2018], however not by a large margin. While the paper reports approximately 23.5% average error rate, our average error rate was approximately 23.1%. This difference could be caused purely by chance or be a result of the slightly different way of preparing datasets for the experiment or the aforementioned different versions of software and other minute differences. It is probably not significant in any way and does not warrant more investigation. It, however, reinforces the point that any comparison of our results with the results from the research paper needs to be taken with a grain of salt.

More significant conclusions can be drawn from the Mean Teacher results. The table contains results for two slightly different mismatch rates. We performed these baseline experiments for both, because we changed the way mismatch rate was calculated during the experiments. At first, data samples from unfavorable classes in the labeled data were used for consistency regularization as well. We, however, consider this unrealistic because if we know the labels, there is no reason to use those data samples. After the change, only labeled data belonging to the favorable classes was used for consistency regularization and the mismatch rate calculations reflect that, while before they only considered the unlabeled data. Because of this change, the data that previously had 75% mismatch rate now have a mismatch rate of 64% and therefore could not be used as a baseline, which is why we performed the other set of experiments.

As can be seen by comparing the Mean Teacher using all the data and the supervised baseline results, the problem is still present even in our more realistic data distribution scenario. Moreover, comparing the results using all the data with results using only data from favorable classes gives us the worst and the best-case accuracies achievable if we do not lose a significant portion of favorable unlabeled data. This comparison reveals that even when not using any additional data, the accuracy could be improved by up to several percentage points by correctly filtering the unfavorable data out. This demonstrates that even in our slightly more convenient scenario, solving the class mismatch issue is still worthwhile.

Comparing results of the experiments with the two different mismatch rates could provide an opportunity to examine the effects of changing the mismatch rate, but an important thing to keep in mind is that since the mismatch rate is quite high and there are six favorable classes as opposed to four unfavorable classes, the limiting factor of the dataset size used for training is the unfavorable data. Due to that, while a 15% increase in mismatch rate might not seem very

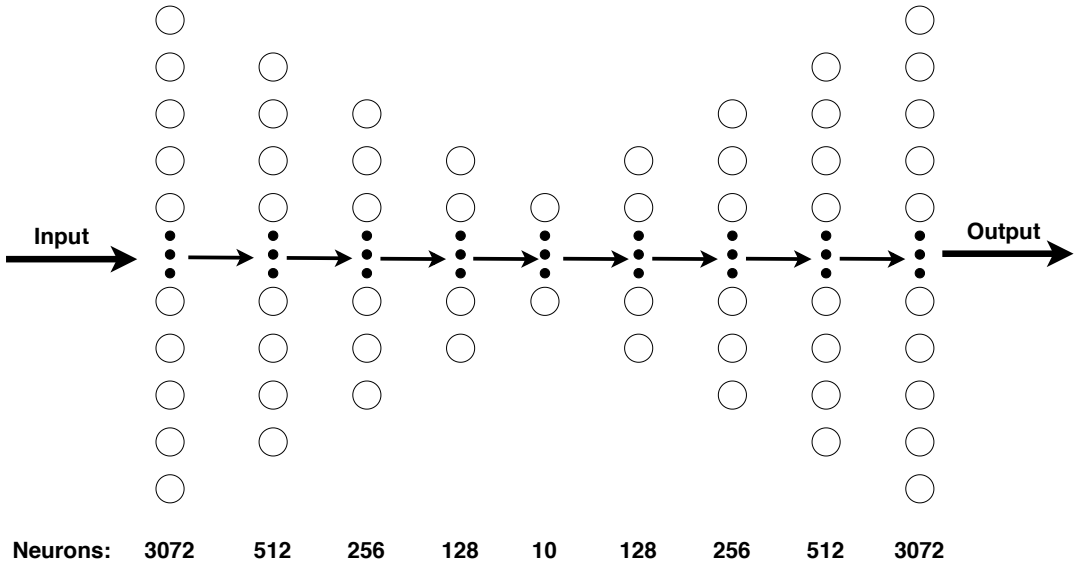


Figure 4.2: Diagram of the final autoencoder architecture we used. All neurons use leaky ReLU activation functions. All the layers are fully connected.

significant, it actually means losing approximately 2800 favorable unlabeled data samples out of the initial 5400. Because of this, the noticeable effects are probably caused more by the difference in the amount of favorable unlabeled data samples than by the difference in the mismatch rate itself. Nevertheless, comparing the results for the two mismatch rates shows that the effects of changing the mismatch rate by adding or removing favorable unlabeled data are as can be expected. Lower mismatch rate means milder negative effects of the class mismatch and higher positive effects (when compared to the supervised baseline) when the unfavorable data is filtered out, while higher mismatch rate leads to opposite effects.

4.4 Filtration Experiments

As mentioned in the previous chapter, we performed many experiments with our filtering before trying to run the SSL with the filtered data to make sure all the parameters are set to reasonable values and to check how well the different designs of the autoencoder and the filter are performing. While in the results presented in this chapter, we always use the actual mismatch rate and counts we were able to achieve, we would not be able to do so in practice, of course. For that reason, we also observed how well the filtration performed on the validation data and on the labeled training data. We found out that results on both of these seem to closely correlate with the results on the unlabeled training data, which would make setting the hyperparameters of UDF for the specific case viable in practice,

Figure 4.3: Images showing reconstruction quality achieved by the autoencoder architecture we used during our experiments. The top row contains CIFAR-10 images, while the bottom row contains the same images after passing through the trained autoencoder.

especially thanks to the results on the labeled training data. The validation data will be very small in practice, which might make it impossible to accurately determine how well the filtration is doing, but the labeled training data should be larger and we have not seen any evidence of the filter overfitting. Moreover, even if we only had very little labeled training data and thus were unable to tune the hyperparameters to the specific case, the change in performance would not be high. As it turns out, the two most important parameters are the filter type and the autoencoder architecture, with the precise settings of others responsible for only several out of the tens of percentage points of improvement of mismatch rate usually achieved, and these two parameters should not be hard to set — we have identified one filter type described in the previous chapter, which was clearly superior to all the others we tried to use, and one autoencoder architecture, which should be applicable in many different domains with only very small changes.

During the experiments with autoencoder architectures, we came to three surprising conclusions. The first one was that the best performing (with regard to the filtering performance) autoencoder architecture was not a convolutional autoencoder, as we assumed due to the advantages convolutional neural networks provide when dealing with images, but a fully connected autoencoder using leaky ReLU activations (the final architecture is illustrated by diagram 4.2). The second one was that the quality of reconstructed images at the output of the autoencoder does not seem to matter much. The best performing architecture has a bottleneck of just ten hidden units and as you can see by pictures in figure 4.3, the reconstructed images are mostly blurs, containing approximately the right colors at approximately the right places, but the subject of the original picture tends to be completely unrecognizable. Increasing the capacity of the autoencoder expectedly increases the quality of reconstructions, but severely decreases the filtering

performance. And last, but not least, ZCA normalization significantly decreased the filtering performance, which is why we decided to only use it for training the classifier and to use the original images for the filtering.

The less important parameters for the filter include the number of centroids, the number of iterations of the second step of the filter, the r coefficient and the distance metric used. As mentioned previously, we found out that as long as we set reasonable values for them, the details do not make much difference. Therefore, to avoid making the values too specific for the task at hand using knowledge we should not have (e.g. labels of the unlabeled training data), we manually set them to values we considered reasonable due to the results of the preliminary experiments. The final metric used is the Euclidean metric and the number of centroids was set to one-hundredth of the available labeled data count. Of course, just as the k-means runs are separate for the favorable and unfavorable data, the numbers of favorable and unfavorable centroids are also determined separately. The number of iterations in the second step was set to five. And last, the r coefficient was set to 0.75.

As for the final filtering results, they are very promising. For example, for a training data containing 431 unlabeled samples for each favorable class, 400 labeled samples for each class and having a mismatch rate of 75%, it is able to reduce the mismatch rate to just 44%, losing only a fifth (approximately 500) of the favorable unlabeled samples, while filtering out approximately 75% of the unfavorable unlabeled samples (ca 11500 out of 15000). If we use the same figures, just with a 50% initial mismatch rate, it is capable of decreasing the mismatch rate to just 21%, while again filtering out approximately 75% of the unfavorable data. The filtering seems to be balanced across classes, with the exception of the Ship class, which is consistently filtered out slightly more heavily than the other unfavorable classes, possibly thanks to the blue background color unusual in the favorable classes.

The time it takes to perform the filtering is very short, especially when compared to the SSL training afterwards. We trained the autoencoder for 300 epochs, which was more than enough to achieve the best results in all experiments. Even considering the autoencoder training, the total time spent running UDF was always under twenty minutes, while the subsequent SSL training on the same hardware took twelve hours. From this, we conclude that the time spent running UDF should not ever be a factor in deciding whether to use it or not, because it will probably always be eclipsed by the time needed to train the actual model. The memory requirements are not significant either, being linear in the dataset size. Even if the dataset does not fit into the memory, it can be processed in chunks, both during the training of the autoencoder and during the feature extraction and filtering after that training.

4.5 Experiments with Filtered Data

The first experiments with data filtered by our filter we performed were done with datasets with 75% mismatch rate. Their results are reported in table 4.2 as the "Our Filter" experiment type along with the previously presented baseline results for easier comparison. Applying our filter improved the average accuracy by more than half of a percentage point and its worst and best results are both better

M.r., experiment type	Mean(-S.B. mean)	Std	Min	Max
0.44, Our Filter	0.7570(-0.0118)	0.0037	0.7530	0.7620
0.75, M.T., Complete Data	0.7507(-0.0181)	0.0056	0.7432	0.7567
0.75, M.T., Filtered Data	0.7783(0.0095)	0.0049	0.7717	0.7858
N/A, Supervised Baseline	0.7688	0.0045	0.7632	0.7760

Table 4.2: Test accuracies achieved for training data with 400 labeled samples per favorable class, 431 unlabeled samples per favorable class and a mismatch rate of 75%.

than the worst and best results before its application, with the worst result being almost a full percentage point better. This shows that our filter really improves the results. Unfortunately, it is still not enough to make using SSL in this case worthwile, because the results are still not better than the supervised baseline.

M.r., experiment type	Mean(-S.B. mean)	Std	Min	Max
0.2, Our Filter	0.7814(0.0126)	0.0025	0.7780	0.7840
0.5, M.T., Complete Data	0.7944(0.0257)	0.0028	0.7907	0.7973
0.5, M.T., Filtered Data	0.8023			
N/A, Supervised Baseline	0.7688	0.0045	0.7632	0.7760

Table 4.3: Test accuracies achieved for training data with 400 labeled samples per favorable class, 2094 unlabeled samples per favorable class and a mismatch rate of 50%.

Our first assumption why that would be the case was that the mismatch rate was still too high, even after the filtration process, which only reduced it to 44%. To test whether this could be the case, we started experiments for the 50% mismatch rate. Their results can be observed in table 4.3. We only got one value for the "Filtered Data" experiment type, because we decided to abandon this set of experiments after getting enough results from the other types. As you can see, the results showed that SSL using both favorable and unfavorable data achieved better results than the supervised baseline. Since the mismatch rate was actually higher than the 44% mismatch rate we filtered data to previously, another factor had to play a significant role in the difference of results. As mentioned earlier, we already suspected that the absolute amount of the favorable data plays a significant role too, not just the mismatch rate. And the difference of absolute amounts between the cases was very large — while in previous experiments, we could only use 431 unlabeled samples per favorable class to make sure we had enough unfavorable data to fill the rest, the lower mismatch rate allowed us to increase this figure to 2094. The theory is further supported by the fact that while we were able to decrease the mismatch rate to a very low 20% by our filtering, the accuracies achieved using the filtered data were lower than in the case using all the data. This might have happened because the decrease in the amount of favorable unlabeled data due to mistakes made by the filter was more significant than the decrease in the mismatch rate.

To test this more, we run experiments with training data, whose mismatch rate was 50%, but who contained only the 431 unlabeled samples per favorable class. The resulting accuracies listed in table 4.4 support our theory. While

M.r., experiment type	Mean(-S.B. mean)	Std	Min	Max
0.21, Our Filter	0.7726(0.0038)	0.0055	0.7657	0.7787
0.5, M.T., Complete Data	0.7663(-0.0025)	0.0033	0.7618	0.7700
0.5, M.T., Filtered Data	0.7783(0.0095)	0.0049	0.7717	0.7858
N/A, Supervised Baseline	0.7688	0.0045	0.7632	0.7760

Table 4.4: Test accuracies achieved for training data with 400 labeled samples per favorable class, 431 unlabeled samples per favorable class and a mismatch rate of 50%.

using all the data expectably has smaller negative effects than it had when the mismatch rate was 75%, the accuracies are still slightly below the supervised baseline and are significantly worse than for the same mismatch rate but more favorable data samples.

Even more interestingly, SSL with our filtering actually outperforms the supervised baseline, making the SSL beneficial under these conditions. From the cases we have seen so far, we deduce that the decision of whether to use SSL with or without our filtering or whether to use pure supervised learning should be dependent on the mismatch rate and the amount of favorable data. Of course, we do not know the exact values of those in practice, so we would need to estimate them. For example, in cases where we perform the labeling by sampling from the data we have and manually assigning the labels, we know how many favorable and how many unfavorable data we encountered in the process and can use those figures to perform the estimation.

Once we have the two values, we would recommend using supervised learning when you only have a small amount of favorable unlabeled data (for example, as many per class as in the labeled data) and a mismatch rate significantly higher than 50% and SSL without filtering when you have a large amount of favorable unlabeled data (several times more than the labeled data or more). SSL with our filtering should be a good choice for the rest, meaning mainly smaller or medium amounts of favorable unlabeled data along with a mismatch rate of around 50% or less.

Naturally, all these guidelines could be highly specific to our particular case and to make them more general, performing extensive experiments on other datasets than CIFAR-10 and other SSL algorithms would be necessary.

4.6 Class Distribution Mismatch Grid

To better understand how the issue behaves in regard to mismatch rate and favorable unlabeled data counts, we run a number of additional experiments and generated heat map 4.4. While most of the fields contain values determined using only a single run and are therefore possibly inaccurate, the general trends can still be observed in it. Because of these possible inaccuracies, conclusions described in the next paragraphs are sometimes ignoring a case that might seem an exception, but is probably just caused by the variance of the results. This variance can be seen in previously described tables, where it is not uncommon for the best and worst results to be a full percentage point away from each other, even though they use exactly the same configuration.

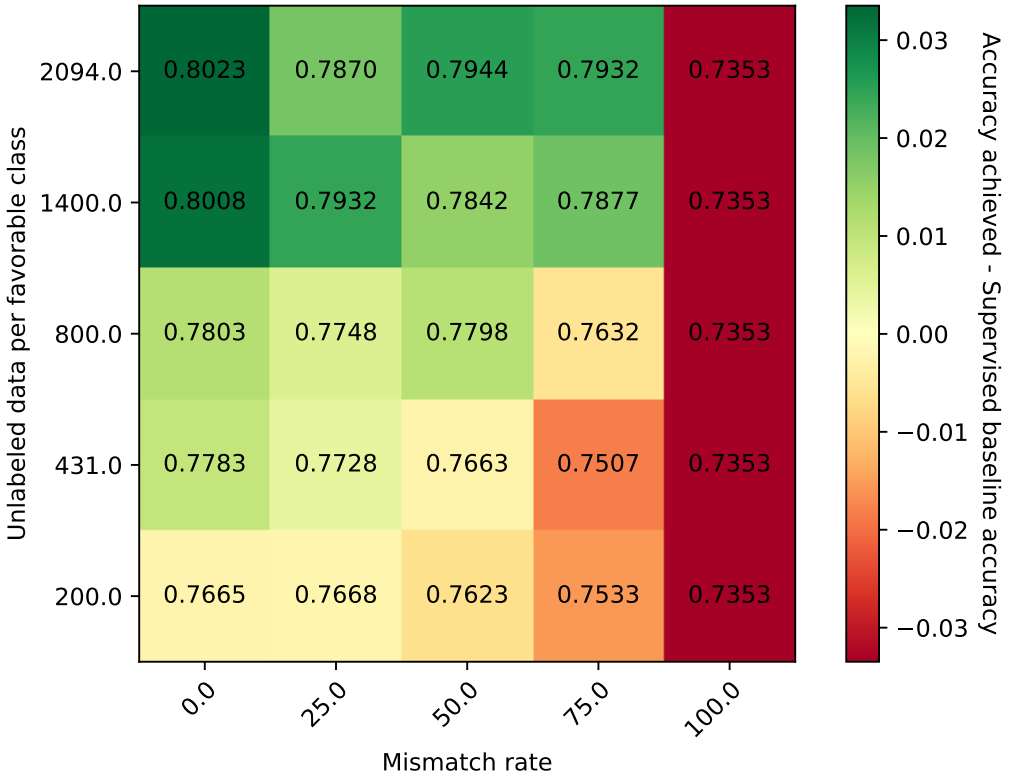


Figure 4.4: Accuracy difference to supervised baseline heatmap, containing mean accuracy for each field.

Figure 4.5: Examples of preprocessed images from the Open Images Dataset.

Another thing to keep in mind is that while most of the experiments used pure CIFAR-10, we had to augment the unfavorable classes using images from the Open Images Dataset during the experiments with mismatch rate of 75% and unlabeled favorable data count higher than 431 per class. While we tried to find alternatives as close as possible to the CIFAR-10 classes (Helicopter and Airplane for the Airplane CIFAR-10 class, Car for the Automobile class, Ship for the Ship class and Truck, Van and Bus for the Truck class), the images are still slightly different than the CIFAR-10 images, even after being scaled down (several examples of them can be seen in figure 4.5). That might have influenced the results, but we do not think the influence was significant, especially since the results are in accordance with the trends in the heat map.

First of all, the rightmost column represents a run using all the unfavorable data but no unlabeled favorable data. Thus, it is an extreme case, demonstrating what happens when you have completely irrelevant unlabeled data. It shows that while even irrelevant data might theoretically help the convolutional neural network identify important features, it is probably not worth using it in general.

Second, the leftmost column contains accuracies from runs using only favorable data in the specific quantities. It shows that, as expected, accuracy increases with the amount of favorable data, as the other columns do for non-zero mismatch rates.

Third, as expected, the accuracies are decreasing with increasing mismatch rates. It seems that they decrease similarly quickly in absolute terms, no matter the data size, which is why the issue is more significant when we have fewer data samples - losing a single percentage point when we are three percentage points above supervised baseline is not that bad, but losing that same point when we are only half of a percentage point better means using SSL suddenly becomes detrimental. This could also explain why using unfiltered data seems to be the best choice from some data amount upwards. At some point, the detrimental effects of losing 20% of the favorable unlabeled data (which is approximately what our filtering caused, as described before) overpower the positive effects of decreasing the mismatch rate because these remain fairly constant, while the effects of losing a percentage of the data increase the more data we have.

And last, but not least, one can also see that up until a mismatch rate between 50% and 75%, fields tend to be better than the fields with lower unlabeled favorable data counts regardless of mismatch rate, which further supports the conclusion that the favorable unlabeled data count is probably even more important than the mismatch rate and should always be considered when deciding to apply or not apply SSL and our filtering.

4.7 Comparison to Oliver et al.

While our results are not directly comparable to results achieved by Oliver et al. [2018] due to the change in the way training datasets are composed, it would be a missed opportunity to not compare our results with theirs because it might lead to some new findings about the issue. To that end, we include one of their charts as figure 4.6. We will be using our Complete Data figures from the cases with mismatch rates of 50% and 75% and 431 unlabeled samples per favorable class. That means that we are using approximately 5000 favorable samples for

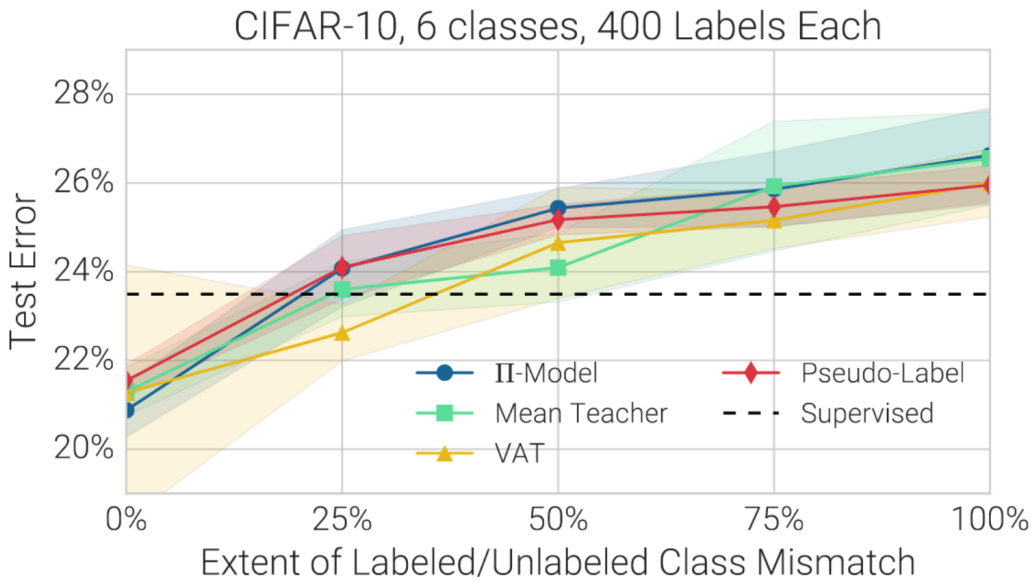


Figure 4.6: A chart from Oliver et al. [2018] showing results of their experiments with class mismatch.

consistency regularization, while they used approximately 4500 at 75% mismatch rate and 9000 at 50% mismatch rate. As we have seen earlier, favorable unlabeled data amounts affect the resulting accuracies very significantly and thus, these differences might cause some of the differences in the results.

As we also noted earlier, supervised baseline results are very similar, which is to be expected, since we made no significant changes in that regard. Our SSL test errors are lower than theirs at both mismatch rates, by almost 1 percentage point at the 75% mismatch rate and approximately 0.7 of a percentage point at the 50% mismatch rate. While the difference for 75% mismatch rate might be attributable to us using slightly more favorable unlabeled data, the difference at the 50% should be in the opposite direction if that was the only factor. Because of that, we would attribute the difference mainly to the previously described way they are creating training datasets, which leads to them having plenty of unlabeled training data for one or two favorable classes but none for the others, while we sample from the favorable classes equally. This would indicate that the class mismatch problem is even more significant if it involves unequal amounts of unlabeled data among the favorable classes. Further research would need to be done in that area, but if that is really the case, strategies to mitigate that additional issue should be considered whenever it is a factor. In the terms of our filtering that might mean for example creating class-specific centroids and trying to upsample or downsample data as needed according to the centroid they belong to. Of course, in the extreme case of not having any unlabeled data for a class, not much can be done.

Different SSL algorithms seem to behave slightly differently with regards to class mismatch. Therefore, it is possible that they will react in a different way to the changing amounts of favorable unlabeled data and our filtering. That reinforces the point made earlier that more experiments should be performed in this regard before applying the filtering to any other SSL method, to identify

guidelines for when to use it and when not to, just as we did for Mean Teacher.

4.8 SVHN Experiments

M.r., experiment type	Mean	Std	Min	Max
0.5, M.T., Complete Data	0.9390	0.0046	0.9349	0.9454
0.75, M.T., Complete Data	0.9414	0.0016	0.9395	0.9435
Either, M.T., Filtered Data	0.9413	0.0013	0.9403	0.9432

Table 4.5: Test accuracies achieved for training data picked from SVHN with 400 labeled samples per favorable class, 431 unlabeled samples per favorable class and a mismatch rate of either 50% or 75%. Mismatch rate for the Filtered Data experiment type is given as "Either", since we use the same amount of favorable data for both mismatch rates and that amount is the only thing that matters for that experiment type.

To show how much the behavior of the class distribution mismatch issue can differ depending on the specific task, several experiments with the SVHN dataset have been performed. We tried to make the conditions as equal as possible to allow for direct comparisons between the SVHN results and the CIFAR-10 results. We designated six classes of SVHN (the digits from 2 to 7) as favorable and the remaining four as unfavorable. We performed the experiments for the mismatch rates of 50% and 75%, 400 labeled samples per class and 431 unlabeled samples per favorable class. Everything else stayed the same too, including the architectures and their hyperparameters.

The results on both the mismatch rates revealed that class distribution mismatch is not an issue in this task on SVHN. Accuracies for Complete Data and Filtered Data experiment types, as seen in table 4.5, were practically identical. We attribute this to the fact that the task was far simpler than the one we performed experiments on using CIFAR-10 and therefore it would be far harder for the unfavorable data to confuse the classifier training. This conclusion is supported by the fact that while we usually saw accuracy values below 80% in the previous experiments, all the accuracy values on SVHN were well above 90%.

No matter the precise reason for the absence of the issue, these results show what we expected (and already noted from previously described literature), which is that the specific dataset and task are important factors in choosing how to handle the issue.

Conclusion

The goal of this thesis was to design an SSL method or an improvement of existing SSL methods that would improve robustness of SSL in regard to different distributions of labeled and unlabeled training data and evaluate its effectiveness. We decided to focus on the class distribution mismatch issue, which arises when unlabeled training data contain samples belonging to classes not present in the labeled data. This can confuse the learning algorithm and lead to diminished accuracy of the final classifier. To mitigate this issue, we developed a filtering method we called Unfavorable Data Filtering (UDF), which is trying to filter out the irrelevant data by using conditions based on similarity of the unlabeled data samples to the labeled favorable and unfavorable data representatives.

UDF turned out to be effective when tested on CIFAR-10 in some of the possible cases, especially when we have a small to medium amount of relevant labeled data and the mismatch rate (which is a proportion of irrelevant data within all the data used for the unsupervised portion of the SSL algorithm) is not much higher than 50%, while in other instances, it might be more beneficial to either not use SSL at all or to use all the data, rather than risk filtering some of the relevant data out by mistake. As shown by comparing the behavior of the issue on SVHN and on CIFAR-10, the properties of the class distribution mismatch issue can differ depending on the specific task we are trying to solve. Therefore, more experiments on different datasets are required to formulate more general guidelines on when to use UDF. The same thing is probably true for different SSL algorithms, as results from Oliver et al. [2018] show that different SSL algorithms behave slightly differently when subjected to class distribution mismatch. Nevertheless, proper application (including not applying it at all when appropriate) of UDF to filter the data before training should make SSL methods more robust to the class distribution mismatch issue and therefore, the overall goal of the thesis has been accomplished.

Results of the experiments also suggest an important property of the class distribution mismatch issue, namely that it becomes less of an issue if we have plenty of unlabeled data, compared to when we for example only have similar amounts of unlabeled data from the relevant classes as we have of labeled data. This could be used to potentially solve the problem in cases, where acquiring more unlabeled data is not too expensive.

As for the future, it is necessary to perform experiments similar to ours for different datasets and SSL algorithms, both to correctly estimate effectiveness of UDF in specific cases and to test whether the property described in the previous paragraph is general or specific to our case. There is also an enormous amount of possibilities for specific design decisions within UDF that could make it beneficial in a wider array of cases, for example using a GAN or a network pretrained on different dataset for the feature extraction part of the algorithm. Moreover, while we only studied one possible difference of the labeled and unlabeled distributions, there are more that UDF could be applied to, with some changes. For example, when the unlabeled distribution is severely unbalanced, a variant of UDF picking representatives separately for each class and deciding to which class the sample most probably belongs could be used to perform oversampling or undersampling

on the unlabeled data.

And last, but not least, while searching the available literature for ways others tried to solve the class distribution mismatch issue, we noticed that their results cannot be directly compared due to the differences in their experimental setup, such as training different model architectures or even using different ways of simulating class mismatch. For example, while some might simulate it by adding data from another dataset as the irrelevant data, others might only use data from a single dataset, choose relevant and irrelevant classes and pick only some of those classes to include in the unlabeled data, leaving others out. In the future, a common experimental setup should be adopted to allow direct comparisons between different approaches.

Bibliography

E. Alpaydin. *Introduction to Machine Learning*. Third Edition. The MIT Press, Cambridge, Massachusetts, 2014. ISBN 978-0-262-02818-9.

B. Athiwaratkun, M. Finzi, P. Izmailov, and A. G. Wilson. There are many consistent explanations of unlabeled data: Why you should average. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=rkgKBhA5Y7`.

O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. The MIT Press, Cambridge, Massachusetts, 2006. ISBN 978-0-262-03358-9.

Y. Chen, X. Zhu, W. Li, and S. Gong. Semi-supervised learning under class distribution mismatch. 2020. URL `https://xiatian-zhu.github.io/papers/ChenEtAl_AAAI2020.pdf`.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets`.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

H. Kaizuka, Y. Nagasaki, and R. Sako. ROI regularization for semi-supervised and supervised learning. *CoRR*, abs/1905.08615, 2019. URL `http://arxiv.org/abs/1905.08615`.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

G. Kumar and P. K. Bhatia. A detailed review of feature extraction in image processing systems. In *2014 Fourth International Conference on Advanced Computing Communication Technologies*, pages 5–12. IEEE, 2014.

A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *IJCV*, 2020.

S. Laine and T. Aila. Temporal ensembling for semi-supervised learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=BJ6oOfqge.

Y. LeCun and C. Cortes. MNIST handwritten digit database. *ATT Labs [Online]*, 2, 2010. URL http://yann.lecun.com/exdb/mnist/.

D.-H. Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 07 2013.

A. LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019. doi: 10.21105/joss.00747. URL https://doi.org/10.21105/joss.00747.

A. Look and S. Riedelbauch. Dealing with limited access to data: Comparison of deep learning approaches. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, July 2019. doi: 10.1109/IJCNN.2019.8852064.

A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. Citeseer, 2013.

L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=BydrOIcle.

V. Nair, J. Fuentes Alonso, and T. Beltramelli. Realmix: Towards realistic semi-supervised deep learning algorithms. *CoRR*, abs/1912.08766, 2019. URL http://arxiv.org/abs/1912.08766.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. Neural Information Processing Systems Foundation, Inc., 2011. URL http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

A. Oliver, A. Odena, C. Raffel, E. A. Cubuk, and I. Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In *Advances in Neural Information Processing Systems 31*, pages 3235–3246. Curran Associates, Inc., 2018.

K. K. Pal and K. S. Sudeep. Preprocessing for image classification by convolutional neural networks. In *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 1778–1781. IEEE, 2016.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*, pages 1041–1044, Orlando, FL, USA, Nov. 2014. Association for Computing Machinery. ISBN 978-1-4503-3063-3.

A. Tarvainen and H. Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1195–1204. Neural Information Processing Systems Foundation, Inc., 2017. ISBN 9781510860964.

A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

J. Uesato, J.-B. Alayrac, P.-S. Huang, R. Stanforth, A. Fawzi, and P. Kohli. Are labels required for improving adversarial robustness? *CoRR*, abs/1905.13725, 2019. URL `http://arxiv.org/abs/1905.13725`.

J. E. van Engelen and H. H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, nov 2019. doi: 10.1007/s10994-019-05855-6.

Q. Xie, Z. Dai, E. H. Hovy, M.-T. Luong, and Q. V. Le. Unsupervised data augmentation for consistency training. *CoRR*, abs/1904.12848, 2019. URL `http://arxiv.org/abs/1904.12848`.

S. Zagoruyko and N. Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press, 2016. ISBN 1-901725-59-6. URL `http://www.bmva.org/bmvc/2016/papers/paper087/index.html`.

M. Zajac, K. Zolna, and S. Jastrzebski. Split batch normalization: Improving semi-supervised learning under domain shift. *CoRR*, abs/1904.03515, 2019. URL `http://arxiv.org/abs/1904.03515`.

List of Figures

List of Tables

A. Attachments

A.1 Attached Code

All the source codes needed for reproducing our experiments are attached to this thesis. Information on what the various folders contain and how to run the experiments can be found in the README file in the root directory.