HELLENIC REPUBLIC
National and Kapodistrian
University of Athens
EST. 1837

Filippos Bellos, BSc
A.M.: 2018511

Iterative label cleaning for semi-supervised learning

Master's Thesis
to achieve the university degree of
Master of Science
Master's degree programme: Control and Computing

submitted to

National and Kapodistrian University of Athens
School of Sciences
Department of Physics

Supervisors
Yannis Avrithis
Inria Rennes-Bretagne Atlantique

Dionysios Reisis
National and Kapodistrian University of Athens

Examination committee: Yannis Avrithis, Dionysios Reisis, Anna Tzanakaki

Athens, July 2021

HELLENIC REPUBLIC

National and Kapodistrian University of Athens

EST. 1837

Φίλιππος Μπέλλος, Πτυχιούχος
Α.Μ.: 2018511

Επαναληπτικός καθαρισμός προβλέψεων για ημι-επιβλεπόμενη μάθηση

Διπλωματική Εργασία

για να λάβει
Μεταπτυχιακό Δίπλωμα Ειδίκευσης
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών: Ηλεκτρονικός Αυτοματισμός

κατατέθηκε στο

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Σχολή Θετικών Επιστημών
Τμήμα Φυσικής

Επιβλέποντες
Γιάννης Αβρίθης
Inria Rennes-Bretagne Atlantique

Διονύσιος Ρεΐσης
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εξεταστική επιτροπή: Γιάννης Αβρίθης, Διονύσιος Ρεΐσης, Άννα Τζανακάκη

Αθήνα, Ιούλιος 2021

Abstract

Deep neural networks have become the de facto model for computer vision applications. Their success is partially attributable to their scalability, i.e., the empirical observation that training them on larger datasets produces better performance. Deep networks often achieve their strong performance through supervised learning, which requires a labeled dataset. The performance benefit conferred by the use of a larger dataset can therefore come at a significant cost since labeling data often requires human labor. This cost can be particularly extreme when labeling must be done by an expert.

A powerful approach for training models on a large amount of data without requiring a large amount of labels is semi-supervised learning (SSL). SSL mitigates the requirement for labeled data by providing a means of leveraging unlabeled data. Since unlabeled data can often be obtained with minimal human labor, any performance boost conferred by SSL often comes with low cost. This has led to a plethora of SSL methods that are designed for deep networks.

In this thesis, we propose two methods that combine successful ideas in problems related to our task at hand. In particular, we propose CleanMatch and WeightMatch, two new semi-supervised learning methods that unify dominant approaches and address their limitations. CleanMatch consists of two stages: (1) iterative selection of the most confident pseudo-labels provided by a combination of consistency regularization and pseudo-labeling following FixMatch [96] and (2) augmentation of the labeled set with the selected examples of the first stage and semi-supervised training based on FixMatch on the augmented dataset. WeightMatch estimates a weight reflecting the confidence of each labeled example, forcing the model to rely more on the confident ones during training.

Our methods improve the state-of-the-art by a large margin on CIFAR-10, SVHN and CIFAR-100, especially on few label settings.

Subject area: Computer vision, Deep learning
Keywords: Semi-supervised learning, Noisy labels

Περίληψη

Τα βαθιά νευρωνικά δίκτυα έχουν γίνει το de facto μοντέλο για εφαρμογές όρασης υπολογιστών. Η επιτυχία τους οφείλεται εν μέρει στη δυνάτοτητα κλιμάκωσης τους, δηλαδή στην εμπειρική παρατήρηση ότι εκπαιδεύοντας τα σε μεγαλύτερα σύνολα δεδομένων παράγουν καλύτερη απόδοση. Τα βαθιά δίκτυα επιτυγχάνουν συχνά ισχυρή απόδοση μέσω επιβλεπόμενης μάθησης, η οποία απαιτεί ένα σύνολο δεδομένων με ετικέτες. Το όφελος στην απόδοση που αποδίδεται στη χρήση ενός μεγαλύτερου συνόλου δεδομένων μπορεί να έχει σημαντικό κόστος καθώς η εμβάπτιση δεδομένων συχνά απαιτεί ανθρώπινη εργασία. Αυτό το κόστος μπορεί να είναι ιδιαίτερα μεγάλο όταν πρέπει να γίνει εμβάπτιση από κάποιον ειδικό.

Μια ισχυρή προσέγγιση για την εκπαίδευση μοντέλων σε μεγάλο αριθμό δεδομένων χωρίς να απαιτείται μεγάλη ποσότητα ετικετών είναι η ημι-επιβλεπόμενη μάθηση. Η ημι-επιβλεπόμενη μάθηση μετριάζει την απαίτηση για δεδομένα με ετικέτες παρέχοντας ένα μέσο αξιοποίησης δεδομένων χωρίς εμβάπτιση. Δεδομένου ότι τα δεδομένα χωρίς εμβάπτιση μπορούν να ληφθούν με την ελάχιστη δυνατή ανθρώπινη εργασία, κάθε αύξηση της απόδοσης που παρέχεται από την ημι-επιβλεπόμενη μάθηση έρχεται συχνά με χαμηλό κόστος. Αυτό οδήγησε σε μια πληθώρα μεθόδων ημι-επιβλεπόμενης μάθησης που έχουν σχεδιαστεί για βαθιά δίκτυα.

Σε αυτή τη διατριβή, προτείνουμε δύο μεθόδους που συνδυάζουν επιτυχημένες ιδέες σε προβλήματα που σχετίζονται με το ζητούμενο που εξηγήθηκε προηγουμένως. Συγκεκριμένα, προτείνουμε το CleanMatch και το WeightMatch, δύο νέες ημι-επιβλεπόμενες μεθόδους μάθησης που ενοποιούν κυρίαρχες προσεγγίσεις και προτείνουν λύση στους περιορισμούς τους. Το CleanMatch αποτελείται από δύο στάδια: (1) επαναληπτική επιλογή των πιο σίγουρων ψευδο-ετικετών που παρέχονται από ένα συνδυασμό κανονικοποίησης συνέπειας και ψευδο-εμβάπτισης (consistency regularization and pseudo-labeling) ακολουθώντας το FixMatch [96] και (2) αύξηση του σετ ετικετών με τα επιλεγμένα παραδείγματα του πρώτου σταδίου και ημι-επιβλεπόμενη εκπαίδευση με βάση το FixMatch στο επαυξημένο σύνολο δεδομένων. Το WeightMatch υπολογίζει ένα βάρος που αντικατοπτρίζει την εμπιστοσύνη κάθε παραδείγματος, αναγκάζοντας το μοντέλο να βασίζεται περισσότερο στα πιο σίγουρα απάυτά κατά τη διάρκεια της εκπαίδευσης.

Οι μέθοδοι μας επιτυγχάνουν την καλύτερη απόδοση σε πολλά σύνολα δεδομένων. Επιτυγχάνουν σημαντικές βελτιώσεις στην ακρίβεια στα CIFAR-10, SVHN και CIFAR-100 σε σενάρια με λιγοστές ετικέτες.

Θεματική περιοχή: Όραση υπολογιστών, Βαθιά μάθηση

Λέξεις-κλειδιά: Ημι-επιβλεπόμενη μάθηση, Ετικέτες με θόρυβο

Acknowledgements

Part of this work was conducted in the Linkmedia Team of INRIA Rennes-Bretagne Atlantique. I would like to sincerely thank my supervisor Yannis Avrithis for his invaluable guidance and support throughout this whole project. His contribution goes beyond this thesis though, as through his lectures and the way he communicated his knowledge, I discovered my passion for the field, and I will always be grateful for that. I would also like to thank the Linkmedia Team as a whole for creating a pleasant and warm work environment, but especially Deniz Engin for her willingness to help me resolve whatever technical issue occured and Michalis Lazarou for our interesting discussions on this work and the field in general. Finally, I would like to thank my family for their endless support throughout this work.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

Today, the field of computer vision has become ubiquitous in our society, with applications in image understanding, image search, medicine, drones, and self driving cars. In particular, visual recognition is a central problem to computer vision research, and its goal is to automatically understand the contents of images. From robotics to information retrieval, many desired applications demand the ability to recognize objects, people, scenes, and activities. To train machines that are able to interpret the visual content of an image, the community has developed many algorithms and representations. The main tasks of visual recognition are image classification, detection and segmentation.

In this thesis, we mainly focus on the image classification problem. The goal is to predict if a semantic category is present in the image according to its visual content. This is one of the fundamental problems in computer vision that has a large variety of practical applications. The image classification problem becomes crucial because image and video data are one of the largest and fastest growing sources of information due to the popularization of digital photography (smartphones, digital cameras, etc.) coupled with the expansion of many social networks and mobile Internet access. For instance, there are 350 million photos uploads per day to Facebook and 80 million to Instagram. There are also 300 hours of video uploaded to YouTube every minute. In 2020, Cisco estimates that 82% of all the web traffic will be video, and that every second, a million minutes of video content will cross the network. To exploit that immense and increasing collection of visual data, we need to annotate each image with semantically rich terms, which is the purpose of image classification. Most of the major technology companies, including Google, Facebook, Microsoft, IBM, Yahoo!, Twitter and Adobe, as well as a quickly growing number of start-ups, initiate research and development projects to deploy image understanding products and services.

For a human, the image classification (or image annotation) problem is easy. Unfortunately for a machine it is challenging, because the machine only "sees" numbers, without semantic meaning. The goal is to map all these numbers (i.e. the digital image) into one or several labels. This implies understanding complex semantic meanings based on an image's visual content. The main challenge is that low-level image representations (i.e. the pixels) are not discriminative enough to directly predict semantic-level concepts. Smeulders *et al.*. [94] calls this

problem semantic gap. Bridging the semantic gap requires an image classification model which is able to extract high-level representations from raw image pixels. Moreover a good image classification model must be invariant to the intra-class variations (i.e. appearance variations, see Figure 1.1), while simultaneously retaining sensitivity to the inter-class variations.

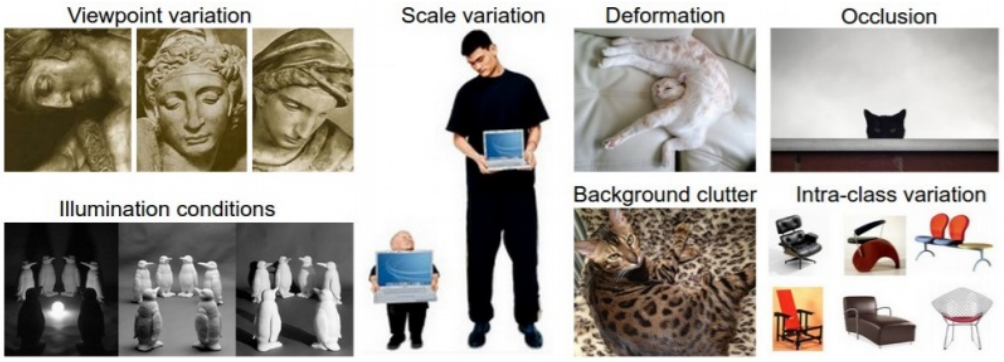


Figure 1.1: Appearance Variations. Image Source: http://cs231n.github.io/.

In the 2000s, most of the image representation models were based on handcrafted features. This approach requires careful engineering and considerable domain expertise to design a feature extractor that transforms the raw image pixels into a feature vector, with an eye for overcoming specific issues like occlusions and variations in scale and illumination. The design of handcrafted features often involves finding the right trade-off between accuracy and computational efficiency while the computation of handcrafted features, is normally a two-step process. First, a keypoint detector locates characteristic regions of an image (e.g. corners), which are then characterized by calculating a descriptor that is capable of distinguishing each particular keypoint from the others. The set of cues considered for building the descriptor depends on the specific feature being used. At a lower level, a descriptor is a vector of measurements that can be used to train a classifier.

However, for many tasks, it is difficult to know what features should be extracted. For example, suppose that we would like to write a program to detect cars in photographs. We know that cars have wheels, so we might like to use the presence of a wheel as a feature. Unfortunately, it is difficult to describe exactly what a wheel looks like in terms of pixel values. A wheel has a simple geometric shape but its image may be complicated by shadows falling on the wheel, the sun glaring off the metal parts of the wheel, the fender of the car or an object in the foreground obscuring part of the wheel, and so on.

One solution to this problem is to use machine learning to discover not only the mapping from representation to output but also the representation itself. This approach is known as representation learning. Learned representations often result in much better performance than can be obtained with hand-

designed representations. They also allow systems to rapidly adapt to new tasks, with minimal human intervention.

A representation learning algorithm can discover a good set of features for a simple task in minutes, or a complex task in hours to months. Manually designing features for a complex task requires a great deal of human time and effort; it can take decades for an entire community of researchers. The quintessential example of a representation learning algorithm is the autoencoder. An autoencoder is the combination of an encoder function that converts the input data into a different representation, and a decoder function that converts the new representation back into the original format. Autoencoders are trained to preserve as much information as possible when an input is run through the encoder and then the decoder, but are also trained to make the new representation have various nice properties. Different kinds of autoencoders aim to achieve different kinds of properties.

When designing features or algorithms for learning features, the goal is usually to separate the factors of variation that explain the observed data. Such factors are often not quantities that are directly observed. Instead, they may exist either as unobserved objects or unobserved forces in the physical world that affect observable quantities. They may also exist as constructs in the human mind that provide useful simplifying explanations or inferred causes of the observed data. They can be thought of as concepts or abstractions that help us make sense of the rich variability in the data. When analyzing an image of a car, the factors of variation include the position of the car, its color, and the angle and brightness of the sun.

A major source of difficulty in many real-world artificial intelligence applications is that many of the factors of variation influence every single piece of data that can be observed. The individual pixels in an image of a red car might be very close to black at night. The shape of the car's silhouette depends on the viewing angle. Most applications require us to disentangle the factors of variation and discard the ones that are not useful. Of course, it can be very difficult to extract such high-level, abstract features from raw data. Many of these factors of variation can be identified only using sophisticated, nearly human-level understanding of the data. When it is nearly as difficult to obtain a representation as to solve the original problem, representation learning does not, at first glance, seem to help us.

Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning allows the computer to build complex concepts out of simpler concepts. Figure 1.2 shows how a deep learning system can represent the concept of an image of a person by combining simpler concepts, such as corners and contours, which are in turn defined in terms of edges.

The quintessential example of a deep learning model is the feedforward deep network or multilayer perceptron (MLP). A multilayer perceptron is just a

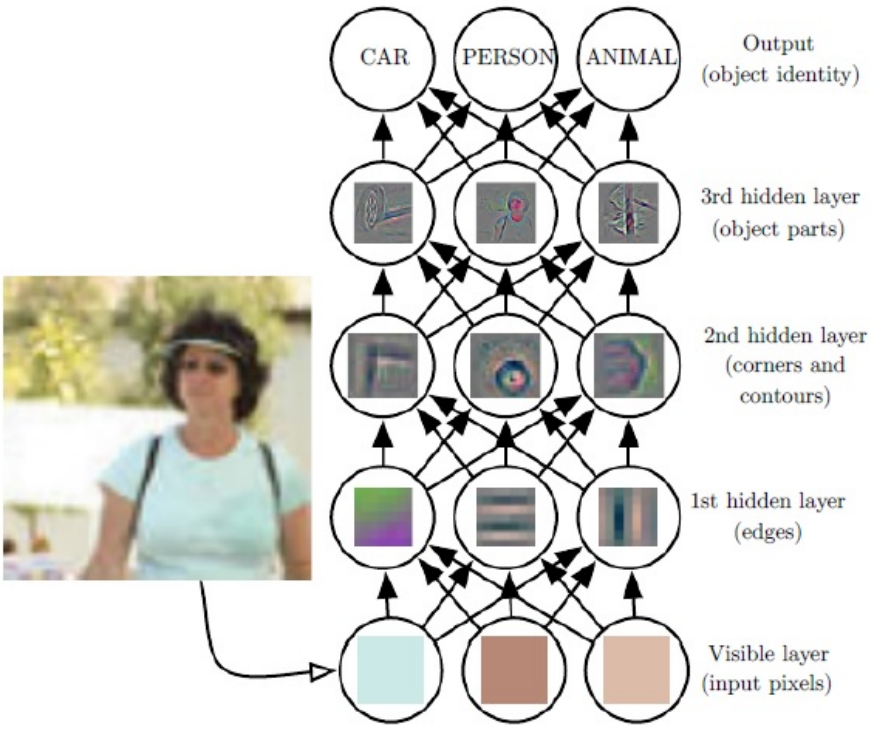


Figure 1.2: Illustration of a deep learning model. It is difficult for a computer to understand the meaning of raw sensory input data, such as this image represented as a collection of pixel values. The function mapping from a set of pixels to an object identity is very complicated. Learning or evaluating this mapping seems insurmountable if tackled directly. Deep learning resolves this difficulty by breaking the desired complicated mapping into a series of nested simple mappings, each described by a different layer of the model. The input is presented at the **visible layer**, so named because it contains the variables that we are able to observe. Then a series of **hidden layers** extracts increasingly abstract features from the image. These layers are called "hidden" because their values are not given in the data; instead the model must determine which concepts are useful for explaining the relationships in the observed data. The images here are visualizations of the kind of feature represented by each hidden unit. Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighboring pixels. Given the first hidden layer's description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. Given the second hidden layer's description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners. Finally, this description of the image in terms of the object parts it contains can be used to recognize the objects present in the image. Image Source: [109]

mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions. We can think of each application of a different mathematical function as providing a new representation of the input. The idea of learning the right representation for

the data provides one perspective on deep learning.

Another perspective on deep learning is that depth allows the computer to learn a multi-step computer program. Each layer of the representation can be thought of as the state of the computer's memory after executing another set of instructions in parallel. Networks with greater depth can execute more instructions in sequence. Sequential instructions offer great power because later instructions can refer back to the results of earlier instructions. According to this view of deep learning, not all of the information in a layer's activations necessarily encodes factors of variation that explain the input. The representation also stores state information that helps to execute a program that can make sense of the input. This state information could be analogous to a counter or pointer in a traditional computer program. It has nothing to do with the content of the input specifically, but it helps the model to organize its processing.

But the big deep learning success was only possible around the 2010s, and can be explained by two factors:

- a large amount of available labeled data to avoid over fitting. The ILSVRC classification (figure 1.3) dataset has 1.2M training images distributed in 1,000 classes.
- the use of Graphics Processing Units (GPUs), which enables training networks 10 or 20 times faster than with Central Processing Units (CPUs).

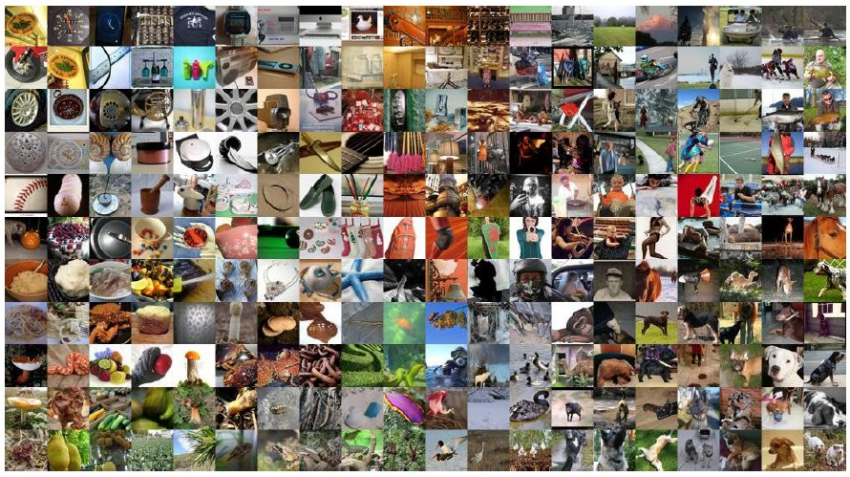


Figure 1.3: **Imagenet.** Contains in total 22k classes with 15M samples. Image Source: [86].

1.2 Motivation-limited supervision

As mentioned deep learning has achieved great successes in both theory and practice, especially in supervised learning scenarios, due to two reasons: by

leveraging a large amount of high-quality labeled data and by exploiting the computational power provided by GPUs.

However, labeled samples are often difficult, expensive, or time-consuming to obtain. The labeling process usually requires experts' efforts, which is one of the major limitations to train an excellent fully-supervised deep neural network. If only a few labeled samples are available, it is challenging to build a successful learning system. By contrast, the unlabeled data is usually abundant and can be easily or inexpensively obtained.

So, to overcome that need for large hand-labeled and expensive training sets and to leverage a large number of unlabeled data for improving the learning performance, there are deep learning systems use some form of weak supervision or some other related concept. These concepts are explained below:

Weakly-supervised learning In weakly-supervised learning, the objective is the same as in supervised learning, however, instead of a ground-truth labeled training set, we are provided with one or more weakly annotated examples, that could come from crowd workers, be the output of heuristic rules, the result of distant supervision, or the output of other classifiers. For example, in weakly-supervised semantic segmentation, pixel-level labels, which are harder and more expensive to acquire, are substituted for inexact annotations, e.g. image labels, points, scribbles and bounding boxes.

Active learning In active learning, the learning algorithm is provided with a large pool of unlabeled data points, with the ability to request the labeling of any given examples from the unlabeled set in an interactive manner. As opposed to classical passive learning, in which the examples to be labeled are chosen randomly from the unlabeled pool, active learning aims to carefully choose the examples to be labeled to achieve a higher accuracy while using as few requests as possible, thereby minimizing the cost of obtaining labeled data.

Transfer learning Transfer learning is used to improve a learner on one domain, called the target domain, by transferring the knowledge learned from a related domain, referred to as the source domain. For instance, we may wish to train the model on a synthetic, cheap to generate data, with the goal of using it on real data. In this case, the source domain used to train the model is related but different from the target domain used to test the model.

Meta-learning Meta-learning is also a useful paradigm. Meta-learning, also known as "learning to learn", aims to learn new skills or adapt to new tasks rapidly with previous knowledge and a few training examples. It is well known that a good machine learning model often requires a large number of samples for training. The meta-learning model is expected to adapt and generalize to new environments that have been encountered during the training process. The

adaptation process is essentially a mini learning session that occurs during the test but has limited exposure to new task configurations. Eventually, the adapted model can be trained on various learning tasks and optimized on the distribution of functions, including potentially unseen tasks.

Learning from noisy labels It can be challenging given the negative impact label noise can have on the performance of deep learning methods if the noise is significant. To overcome this, most existing methods for training deep neural networks with noisy labels seek to correct the loss function. One type of correction consists of treating all the examples as equal and relabeling the noisy examples, where pseudo label methods can be used for the relabeling procedure. Another type of correction applies a reweighing to the training examples to distinguish between the clean and noisy samples.

Semi-supervised learning It is the field of research in between supervised and unsupervised learning that studies the problem of learning from both labeled and unlabeled data. In Semi-supervised learning (SSL), we are provided with a dataset containing both labeled and unlabeled examples. The portion of labeled examples is usually quite small compared to the unlabeled example (e.g., 1 to 10% of the total number of examples). So with a full dataset containing a labeled subset and an unlabeled subset, the objective is to leverage the unlabeled examples to train a better performing model than what can be obtained using only the labeled portion. And hopefully, get closer to the desired optimal performance, in which all of the (full) dataset is labeled.

1.3 Contributions

SSL is a learning paradigm associated with constructing models that use both labeled and unlabeled data. SSL methods can improve learning performance by using additional unlabeled instances compared to supervised learning algorithms, which can use only labeled data. It is easy to obtain SSL algorithms by extending supervised learning algorithms or unsupervised learning algorithms.

A popular class of SSL methods can be viewed as producing an artificial label for unlabeled images and training the model to predict the artificial label when fed unlabeled images as input. For example, pseudo-labeling [64] (also called self-training [90, 105]) uses the model's class prediction as a label to train against. Similarly, consistency regularization [5, 87] obtains an artificial label using the model's predicted distribution after randomly modifying the input or model function. However, with the representation being fixed, the quality of these artificial labels (pseudo-labels) is critical in semi-supervised learning. At the same time, in learning with noisy labels [1, 46, 97], it is common to clean labels based on the loss value statistics of a small-capacity classifier.

In this thesis, we take advantage of these ideas to improve semi-supervised learning. We use the state-of-the-art SSL algorithm FixMatch, which produces artificial labels using both consistency regularization and pseudo-labeling. Crucially, the artificial label is produced based on a weakly-augmented unlabeled image (e.g., using only flip-and-shift data augmentation) which is used as a target to train the model on a strongly augmented version of the same input using standard cross-entropy loss. Inspired by UDA [106] and ReMixMatch [10], it leverages Cutout [25], CTAugment [10] , and RandAugment [22] for strong augmentation, which all produce heavily-distorted versions of a given image. Following the approach of pseudo-labeling [64], it only retains an artificial label if the model assigns a high probability to one of the possible classes. It is crucial though, that the retained artificial labels are of high quality in order for the performance of the network not to be tarnished. That is why, we clean those artificial labels by following O2U-net [46]. O2U-net only requires adjusting the hyper-parameters of the network to make it transfer from overfitting to underfitting cyclically. By calculating and ranking the normalized average loss of every sample, the mislabeled samples as well as the cleanest ones can be identified. In general, the lower the loss of a sample, the higher the probability of being a clean one.

In summary, in this work, we make the following contributions:

1. We combine the power of predicting pseudo-labels in semi-supervised learning [96] with label cleaning in learning from noisy labels [46].
2. We leverage the same label cleaning process but on the labeled set, to estimate a weight reflecting the importance of each labeled example on the training.
3. We achieve new state of the art in semi-supervised image classification.

Our methods are explained in detail in Section 3.1

2 Visual representations

In this section, we present two strategies to extract image representations: Bag of Words (BoW) and deep architectures. The BoW approach was the state-of-the-art model for image classification in the 2000s. This is a handcrafted representation, i.e. it is manually designed and relies on expert knowledge. Since 2012, the Convolutional Network (CNN) is becoming the state-of-the-art model for image classification. Contrary to the BoW representations, the CNN representations are learned from training data.

2.1 Handcrafted representation

Before diving into the Bag of Words (BoW) model it is important to analyze the process of extracting local features. Local feature representation aims to particularly describe the images based on regions of interest while remaining invariant to viewpoint and illumination changes. The images, therefore, are represented according to the local property by the local feature descriptors. The process of extracting local feature contains two primary stages that are feature detection and feature description as following.

2.1.1 Feature detection

Computing of Laplacian-of-Gaussian (LoG) that is a linear combination of second derivatives is a memory-dependent and time-consuming process. To speed up the process, Lowe[68] proposed the state-of-the-art approach based on local 3D extrema in the scale-space pyramid, along with Difference of Gaussian (DoG) filters. The DoG is an analogy to LoG. Hence, the type of features extracted by DoG can be treated as the same type of features as LoG. However, they have the typical limitation that is the local maxima can be detected by the area of straight edges, leading to the issues of sensitivity on outliers or light changes [71]. Harris Corner Detector, was proposed by C. Harris and Stephens[41] is a corner detection approach, which is commonly used in computer vision algorithms to extract corners and infer features of an image. It takes into account the difference between the corner point directly rather than using the displacement block at every 45-degree angle, and is proved to be able to distinguish the angle more accurately[26]. Furthermore, Harris-Laplace detector was proposed as the scale invariant corner detector, and it is consist of the Harris corner detector and the Gaussian scale space representation. In spite of the invariance

of rotation and illumination changes by Harris corner detector, the points are not invariant to the scale. The Harris-Laplace approach significantly reduces the number of redundant interest points compared to Multi-scale Harris. The points are invariant to scale changes, rotation, illumination, and the addition of noise. Moreover, the interest points are highly repeatable. However, the Harris-Laplace detector returns the much smaller number of points compared to the LoG or DoG detectors. The feature detectors, such as DoG and Harris-Laplace, present the invariance of rotation, orientation, and consistent scaling. However, the scale can be different in each direction rather than uniform scaling if the localization and scale are useless for the affine transformation so that it leads to the fail of the scale invariant detectors in affine transformations. With the development of image processing, some features detectors have been extended to extract features invariant to affine transformations. Schaffalitzky and Zisserman[89] modified the Harris-Laplace detector by affine normalization as the extension. And also, Mikolajczyk and Schmid[71] proposed the approach for scale and affine invariant interest point detection.

2.1.2 Feature description

Scale-Invariant Feature Transform (SIFT) is an algorithm in computer vision to detect and describe local features in images, proposed by Lowe. The SIFT descriptor is invariant to consistent scaling, orientation, illumination changes, and partially invariant to affine transformation. There are four main steps in SIFT algorithm. The first step is scale-space extrema detection. As known, it is impossible to use the same window to detect keypoints with different scale. Therefore, SIFT makes use of DoG, which is obtained as the difference of Gaussian blurring of an image with two different values. It is processed for various octaves of the image in Gaussian Pyramid, shown in fig.2.1 After obtaining DoG, the images can be found for local extrema through scale space.

After getting the potential locations for keypoints, SIFT is required to acquire more precise results as refinement because scale-space extrema detection generates few unstable keypoints. The aim of this step is to remove the low contrast keypoints. Besides, the DoG is sensitive to edges so that it is necessary to be removed according to the detector of Harris corner. After that, orientation is assigned to each keypoint to keep invariance to image rotation. A neighborhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region for all pixels around the keypoint using equation 2.3. The most important gradient orientations are identified using the histogram. Lastly, the keypoint descriptor is generated, and a 16*16 neighborhood around the keypoint is taken. It is divided into 16 sub-blocks of 4*4 size. For each sub-block, 8 bin orientation histogram is created. Therefore, a total of 128 bin values are generated. SIFT descriptor representation is designed to avoid the problems of boundary changes in location,

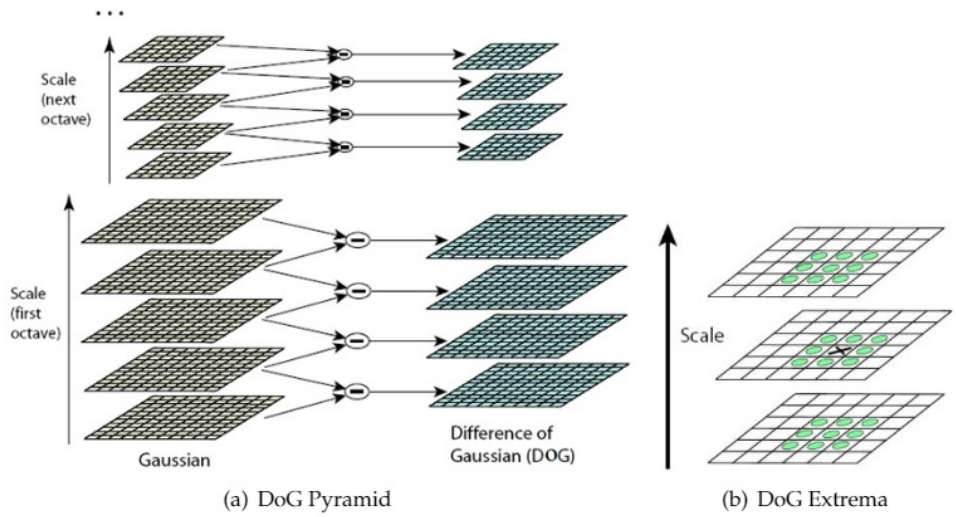


Figure 2.1: Difference-of-Gaussian (DoG) Pyramid and Extrema Selection. Image source:

orientation and scale do not cause radical changes in the feature vector.

Speeded up robust feature (SURF), was proposed by Bay, Tuytelaars, and Van Gool [6], is local feature descriptor inspired by SIFT descriptors. The SURF descriptor is based on the same principles and steps as SIFT. However, the details are different. The algorithm contains three critical steps, including interest point detection, local neighborhood description, and matching. The SURF was designed to the approximation to LoG with box filter, which is the better to calculate the convolution using box filter for integral images. Besides, the SURF depends on the determination of Hessian matrix for both scale and location. During the step of orientation assignment, the SURF makes use of wavelet responses in horizontal and vertical direction for a neighborhood, and also, enough Gaussian weights are applied to it. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60 degrees. Then, a square region is extracted in order to describe the region around the points. The point of interest is divided into 4x4 square sub-regions, and the Haar wavelet responses are extracted at 5x5 regularly sample points. Compared to SIFT, the SURF can accelerate the calculation process since it employs 64-dimensional feature vector to describe the local feature as advantages rather than 128 dimensions in SIFT. Furthermore, the Histogram of Oriented Gradient (HOG) was proposed to extract local features in images, which is the variant of SIFT[24]. In this research, it indicated that the HOG provides the excellent performance relative to other existing feature sets including wavelets. Also, Ojala, Pietikainen, and Maenpaa proposed the approach of Local Binary Patterns (LBP) [75] to extract the spatial information of the texture with the invariant to monotonic transformations of the gray levels. In a nutshell, the different approaches of feature extraction in image processing,

global feature and local feature, could deliver the different performance because of the existence of various situations for images, such as scalability, illumination, and rotation. Hence, the performance of each approach should be multiple evaluated by different image datasets for image classification.

The BoW model is inspired from textual information retrieval [88]. The concept is to represent a document as a histogram of occurrence rates of words from a dictionary. Ma et al.[69] was the first to adapt BoW for visual recognition in the NeTra toolbox, to represent an image as a bag of visual words. Then, Fournier et al. [31] extended this approach with Gabor filters. This method was popularized by [68], which employs Scale-Invariant Feature Transform (SIFT) local features. The process of creating BoVW model is shown in figure 2.3, which can be concluded to four key steps as follows. Firstly, it is to detect regions or points of interest. Then, computing local descriptors over those regions or points. After that, quantizing the descriptors into words to form the visual vocabulary. Lastly, finding the occurrences for each specific word in the vocabulary for constructing the BoVW model, namely the histogram of word frequencies. The first two steps are already described, so the two final steps are as follows:

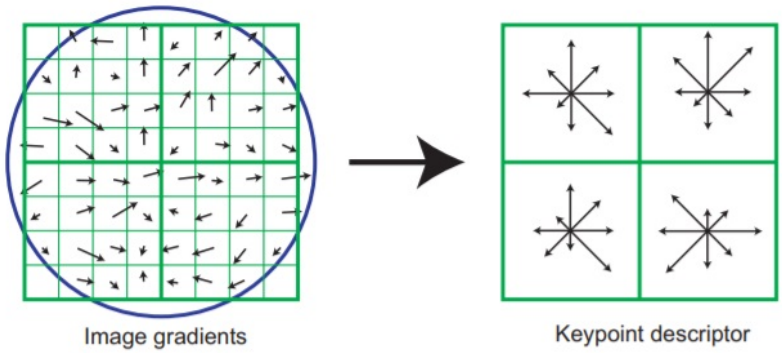


Figure 2.2: **SIFT.** Part of the SIFT pipeline for keypoint descriptor extraction. Image Source: [68].

- **Coding-vector quantization.** The coding step encodes the local descriptors as a function of the dictionary visual words, and outputs visual codes. To learn a visual dictionary, the most popular approach for image categorization is the k-means clustering algorithm [67]. The historical coding function is the hard assignment coding. To reduce the quantization errors and ambiguity resulting from the hard quantization, Gemert et al. [33] proposes the soft assignment. To keep more information, several methods propose to encode the distance in vectorial form: Fisher Vectors (FV) [78], Super-Vector Coding (SVC) [114], Vector of Locally Aggregated Descriptors (VLAD) method and its generalization Vector of Locally Aggregated Tensors (VLAT) [51].

The visual dictionary contains visual words which are used to project local descriptors into another feature space for the subsequent step in the BoW pipeline.

- **Pooling-BoW.** The pooling step constructs a single vectorial representation (or signature) from the set of local visual codes across the whole image. The standard pooling function is the average pooling (or sum pooling) [93]. Another popular method is the max pooling. [14] observes that the best pooling function may be an operation between average and max pooling. To incorporate spatial information, [61] introduces the Spatial Pyramid Matching (SPM). In another way, [4] introduces BossaNova (BN), where the standard scalar pooling is replace by a vectorial pooling to capture higher-order statistics. The image representation has the same dimensionality across all the images of possibly different sizes. Then, the final representation is normalized.

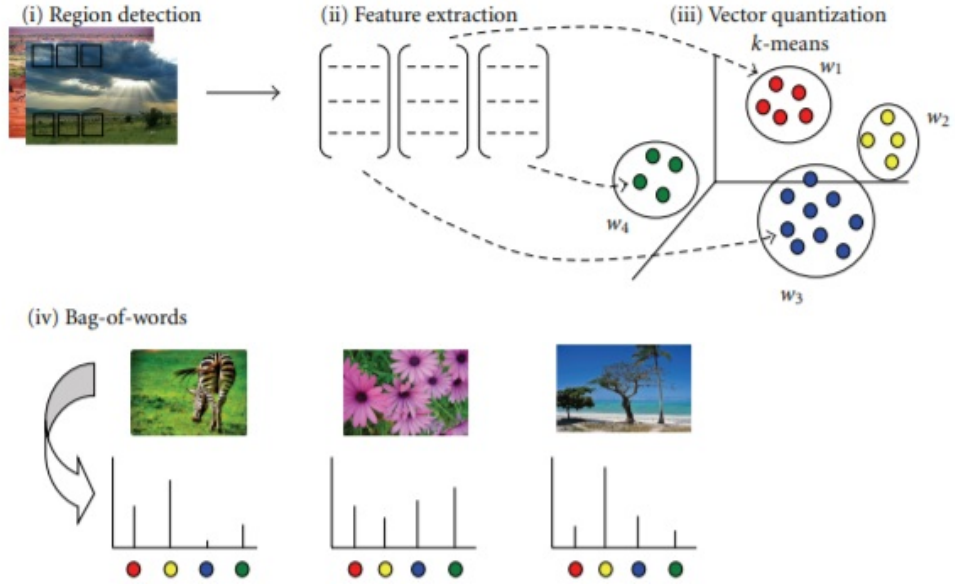


Figure 2.3: The BoVW 4-step process. Image Source: [103]

The global aim is gaining invariance to nuisance factors (locations of the objects, changes in the background, small changes in appearance, etc.), while preserving the discriminating power of the local descriptors. To predict labels, the common approach is to use these representations to train a classifier with supervised learning algorithms. A classifier is a function that maps the representations to one of the possible categories. The most popular classifiers used in computer vision are k-Nearest Neighbors (k-NN) [99], Decision Trees [16], Support Vector Machine (SVM) [13] and neural networks, which are presented in the next section.

2.2 Learned representation

2.2.1 Multi-Layer perceptrons

Multi-Layer Perceptrons (MLPs) are among the most fundamental building blocks in Artificial Neural Networks (ANNs). It refers to a set of computational models that are loosely inspired by the human brain. In general, they consist of two important elements, namely, artificial neurons (nodes) and synapses (weights) that connect them (Figure) . Inspired by the natural world, MLPs are designed to learn feature representations in an hierarchical manner, making them so powerful that in theory they can fit an arbitrary function to arbitrary accuracy. One way of thinking about feedforward neural networks is as a set of nonlinear transformations, where hidden layers are stacked on top of previous outputs. The hidden layer output h can be computed as:

$$y = f_\theta(x) \tag{2.1}$$

where x refers to the inputs to the hidden layer, and $f_\theta()$ is the nonlinear activation function. Traditionally, a sigmoid (fig.2.4) is widely used for f:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

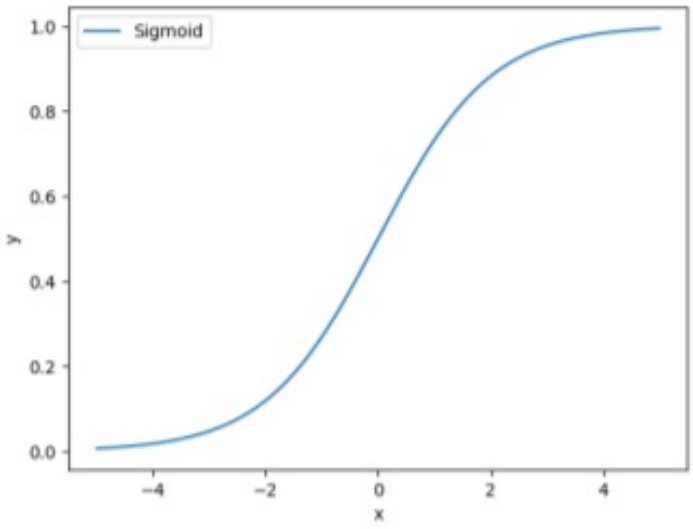


Figure 2.4: Sigmoid function.

More recent researches start using the Rectified Linear Units (fig.2.5):

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & otherwise \end{cases}$$

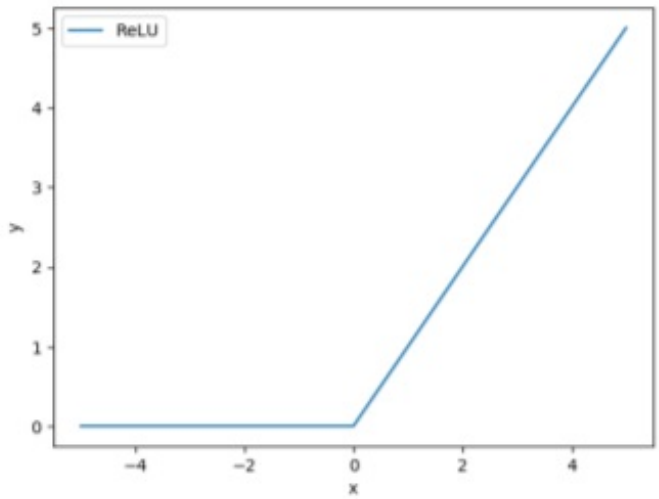


Figure 2.5: Rectified Linear Unit (ReLU).

2.2.2 Convolutional networks

In computer vision, one of the earliest types of convolutional networks dates back to "Neocognition" (fig. 2.6) in the 1980s [32] as already briefly stated in Section 1. Neocognition is a hierarchical network consisting of many layers, and variable connections between nodes in adjacent layers. Local features of the input are initially extracted by the lower level nodes, and gradually integrated into more global features. With the pooling layers, the networks can tolerate slight distortion at each stage, and eventually get robust to deformations, scales, and translations in the position of the inputs. After training, the networks have equipped the ability to perform simple pattern recognition.

Later on, inspired by the "Neocognition", Convolutional Neural Networks

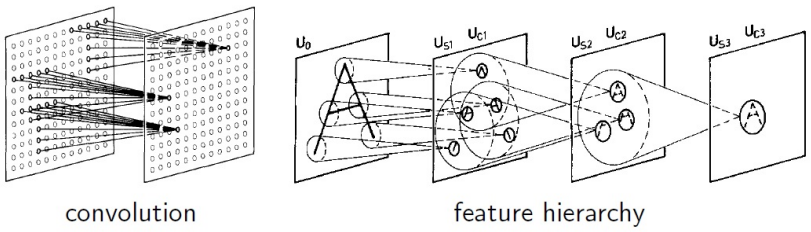


Figure 2.6: **Neocognitron.** A biologically-inspired convolutional network. Image Source: [32].

(CNNs) were introduced and successfully trained with back-propagationn(explained in next subsection). With the help of large-scale datasets and high-performance Graphic Processing Units (GPUs), CNNs started taking off in 2012.

The CNN is a type of feedforward network that uses convolution in at least

one of its layers. This model is called feedforward because information flows strictly in the forward direction, from the input units, through hidden units, if any, and finally to output units. When feedforward neural networks are extended to include feedback connections, they are called Recurrent Neural Networks (RNNs).

The convolution layer is the core layer of deep CNNs. The convolution exploits spatially local correlation by enforcing a local connectivity pattern between units of adjacent layers. Standard CNN architectures are built by stacking convolutional layers followed by non-linearities, and possibly introducing pooling layers to control the computational complexity of the architecture.

CNNs are typically represented by composing together many different functions or layers. For example, a feedforward network with n layers can be written

$$f_\theta(x) = f_{\theta_n}(f - \theta_{n1}(...f_{\theta_2}(f_{\theta_1}(x)))) \tag{2.3}$$

where x is the input, $\theta = [\theta 1, ...,]$ is the vector of CNN parameters, f_{θ_k} is the k_{th} layer and θ_k is the vector of parameters of the k-th layer. The model is associated with a directed acyclic graph describing how the layers are composed together. We note y the output of the network $y = f_\theta(x)$, and $h_k = f_{\theta_k}(h_{k1})$ the output of the k_{th} layer, and $h_0 = x$.

The goal of a CNN is to approximate some function $\hat{f}$. For example, for a classification problem, the CNN maps an input to a category. In general, for an input-output pair (x, y), a CNN defines a mapping $y = f_\theta(x)$, and learns the value of the parameters θ that results in the best function approximation of $\hat{f}$ with respect to a loss function L such that $L(\hat{y}, y) > 0$ measures the disagreement between a ground-truth label $\hat{y}$ and an output y.

We now present the most important **CNN architectures**.

LeNets

LeNets represent the first-generation CNNs, which was initially developed to recognize handwritten digits. Through the architecture, convolutional layers with sigmoid units alter with average pooling layers, where tiny invariance is added and computational burden reduced. At the end of architecture, classifier (fully connected layer) was appended and trained with the classification losses. Although LeNets have achieved the state-of-the-art result on the MNIST dataset, it was found difficult to generalize to real-world vision problems in the 1990s, mainly for two reasons. First, to train the large-scale CNNs, it usually took thousands of iterations to converge with stochastic gradient descent (SGD), however, there was not enough computation power at that time. Second, even by sharing convolutional kernels spatially, a typical CNNs still contains millions of paramters that leads to overfiting the dataset (MNIST) easily.

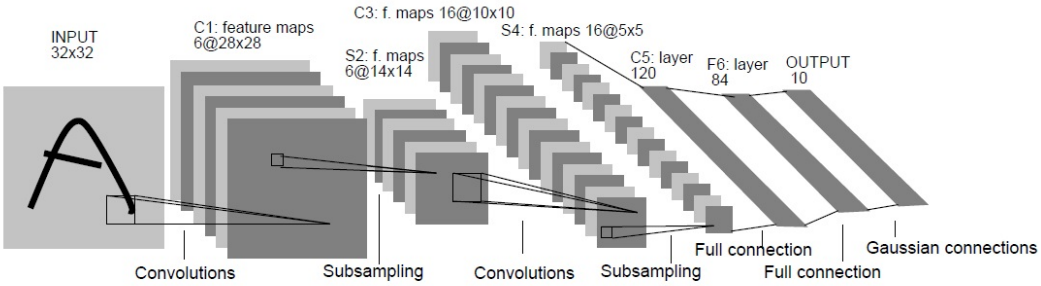


Figure 2.7: **LeNet-5.** It represents the first-generation CNNs, and was initially developed to recognize handwritten digits. Image Source: [62].

AlexNets

The first breakthrough of CNNs in computer vision happened in the year of 2012, AlexNets 2.8 was successfully trained to win the ImageNet Challenge [86]. With the help of a large amount of labelled data and high-performance GPUs, the large-scale CNNs were brought back to the stage. More specifically, AlexNets includes 8 convolutional layers with rectified linear units (ReLUs), max-poolings are used to downsample the feature maps and gradually add tiny invariance. To avoid over-fitting, dropout [98] and data augmentation are applied during training, e.g. color jitterings, random croppings, rotations.

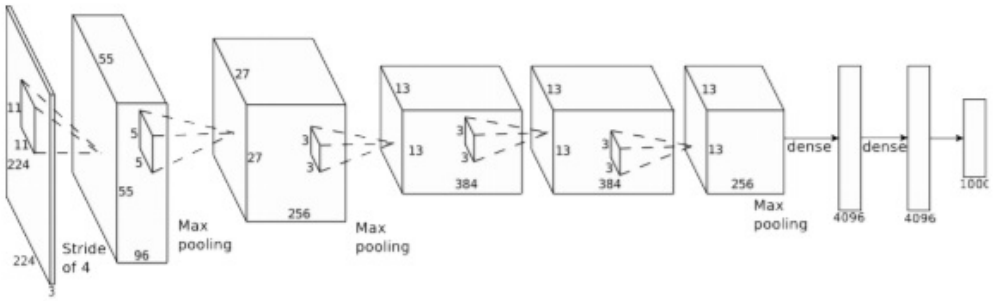


Figure 2.8: **AlexNet.**Outperformed all previous models on ILSVRC by 10% while it included in its implementation ReLU, data augmentation, local response normalization and dropout. Image Source: [59].

VGGNets & GoogLeNets

Since then, a number of works started to further advance image classification performance with CNNs. In 2014, both GoogleNets [100] and VGGNets [92] achieved significant performance boost by using deeper architectures. Specifically, the VGGNets (Figure 2.9) is constructed with 16 convolutional layers of 3 $\times$ 3 kernels, maxpooling layers are used after every two or three convolutional

layers, at the end, two fully connected layers (4096 nodes) are attached to summarize the global information. A downside of the VGG for example is that it is more expensive to evaluate and uses a lot more memory and parameters (138M). To reduce the number of parameters, Szegedy, Liu, et al. [100] intro-

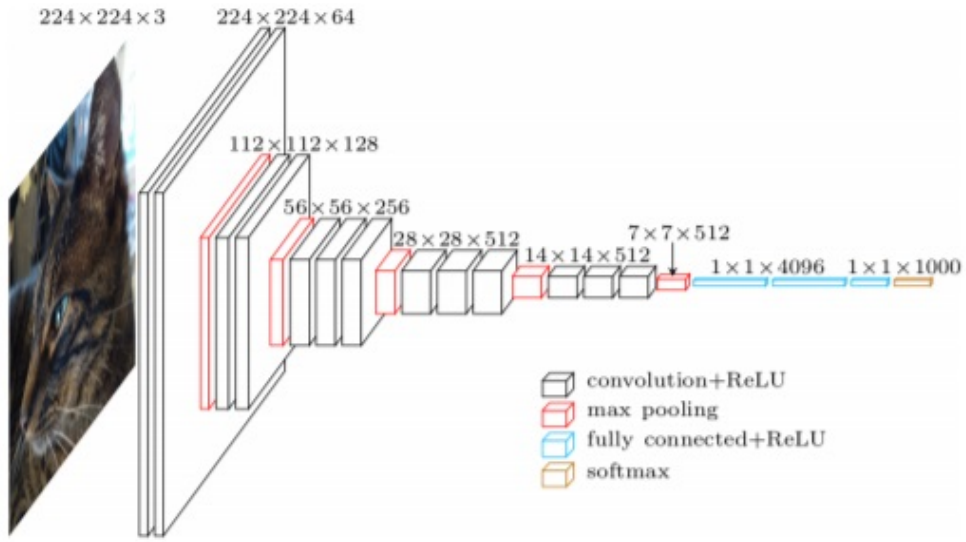


Figure 2.9: **VGGNets.** VGG16, 16 layers. Image Source: [92].

duces the GoogLeNet, that won the ILSVRC 2014 classification challenge. The main contribution is the development of the Inception module, where several 3 × 3 and 1 × 1 kernels are used within one module, that dramatically reduces the number of parameters in the network (7M, compared to AlexNet with 60M). Similar to [63], auxiliary supervisions are applied on several intermediate layers to overcome the gradient vanishing issue. GoogLeNet (22 layers) is deeper than VGGs and uses a Global Average Pooling (GAP) instead of fully-connected layers at the top of the CNN, eliminating a large amount of parameters. Except the fully-connected of the last layer used for classification, all the learned layers are convolution layers. The next step is to learn deeper models than VGGs and GoogLeNet by adding more layers. But this is not possible in practice because of the vanishing/exploding gradients problem ([8, 35]. To overcome this problem and to learn CNNs with several hundreds of layers, [43] builds residual networks (ResNet) by using residual blocks. The residual block uses shortcut connection that skips one or more layers (Figure 2.11). Each residual block can be expresesed in a general form:

$$y_l = h(x_l) + F_{\theta_l}(x_l)x_{l+1} = f(y_l) \tag{2.4}$$

where x_l and x_{l+1} are the input and output of the l_{th} unit respectively, and F is a residual function. It is generally conjectured that gradients are more easily backpropagated with the help of residual units. The authors also show that

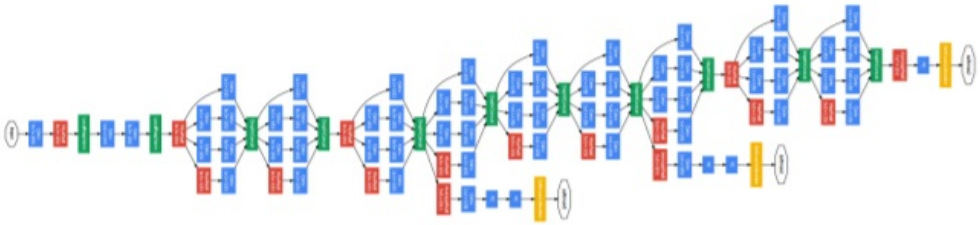


Figure 2.10: **GoogLeNet.** The network uses combinations of inception modules, each including some pooling, convolutions at different scales and concatenation operations. It also uses 1x1 feature convolutions that work like feature selectors. Increased depth to 22 layers . It achieved 6.7% top-5 error on ILSVRC'14. Image Source: [100].

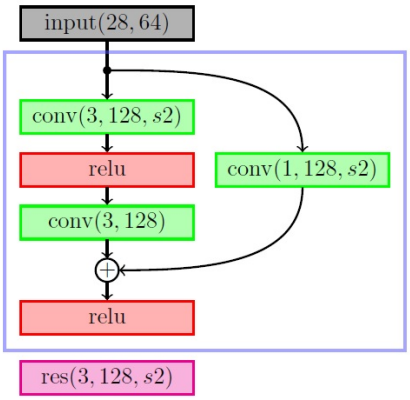


Figure 2.11: Example of Residual Unit. Image Source: [43].

the residual blocks combined with batch normalization [48] make the training easier. Like GoogLeNet, the only learned layers are convolution layers. The ResNets won both ILSVRC MS COCO2 2015 competitions.

Wide ResNets

Deep residual networks were shown to be able to scale up to thousands of layers and still have improving performance. However, these networks present a couple of significant problems.

1. Circuit Complexity Theory: The circuit complexity theory literature showing that: shallow circuits can require exponentially more components than deeper circuits. The authors of residual networks tried to make them as thin as possible in favor of increasing their depth and having less parameters, and even introduced a ≪bottleneck≫ block which makes ResNet blocks even thinner (see **??**(b)).
2. Diminishing Feature Reuse: As gradient flows through the network there is nothing to force it to go through residual block weights and it can avoid learning anything during training, so it is possible that there is either only

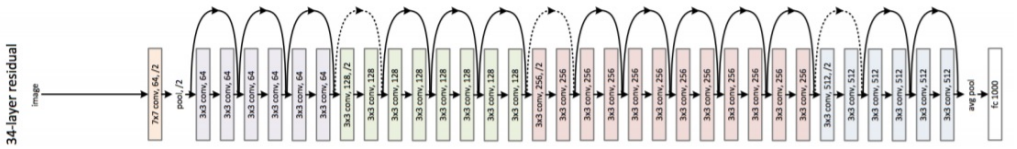


Figure 2.12: **ResNet.** Residual units are used extensively in the networks while at the end average pooling is used to aggregate global feature representations. 3.57% top-5 error on ILSVRC'15. Won first place on several ILSVRC and COCO 2015 tasks. Depth increased to 152 layers. Image Source: [43].

a few blocks that learn useful representations, or many blocks share very little information with small contribution to the final goal. This problem was formulated as diminishing feature reuse.

To tackle these problems, in 2017 Sergey Zagoruyko and Nikos Komodakis [108] conducted a detailed experimental study on the architecture of ResNet blocks, based on which they proposed a novel architecture where they decreased the depth and increased the width of residual networks (fig.**??**). They call the resulting network structures wide residual networks (WRNs) and show that these are far superior over their commonly used thin and very deep counterparts.

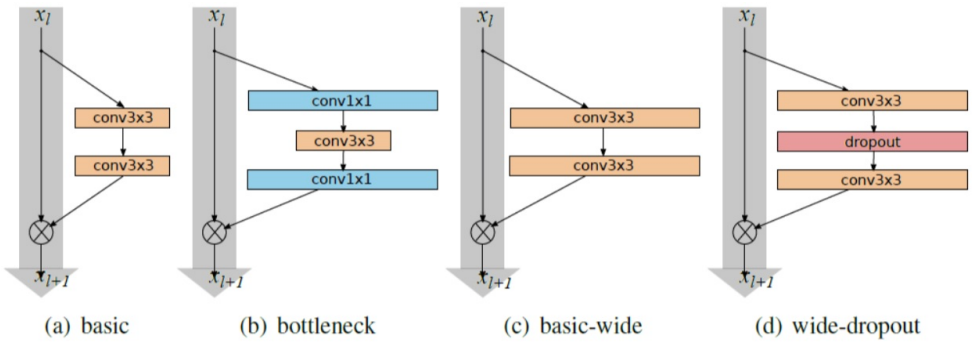


Figure 2.13: Various ResNets blocks. Image Source: [108].

2.2.3 Gradient based optimization

The problem of learning CNNs reduces to an optimization problem, that can be solved by gradient descent. Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \in R^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.

Gradient descent variants

There are three variants of gradient descent, which differ in how much data we use to compute the gradient of the objective function. Depending on the amount of data, we make a trade-off between the accuracy of the parameter update and the time it takes to perform an update.

Batch gradient descent Vanilla gradient descent, aka batch gradient descent, computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.5}$$

As we need to calculate the gradients for the whole dataset to perform just *one* update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory. Batch gradient descent also does not allow us to update our model *online*, i.e. with new examples on-the-fly.

For a pre-defined number of epochs, we first compute the gradient vector of the loss function for the whole dataset.

We then update our parameters in the direction of the gradients with the learning rate determining how big of an update we perform. Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

Stochastic gradient descent Stochastic gradient descent (SGD) in contrast performs a parameter update for *each* training example $x^{(i)}$ and label $y^{(i)}$:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.6}$$

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily as in Figure 2.14.

While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting. However, it has been shown that when we slowly decrease the learning rate, SGD shows the same convergence behaviour as batch gradient descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

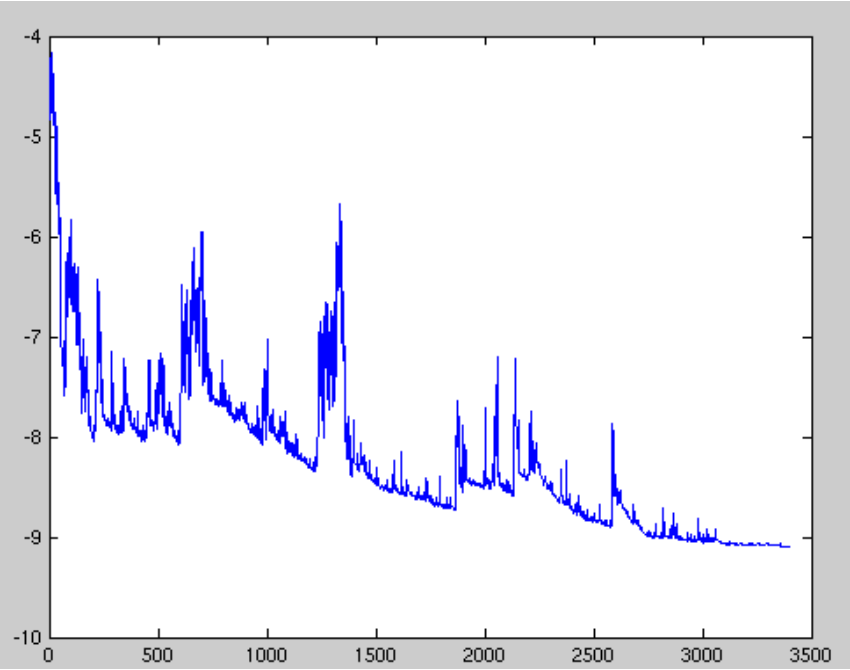


Figure 2.14: SGD fluctuation (Source: Wikipedia)

Mini-batch gradient descent Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples:

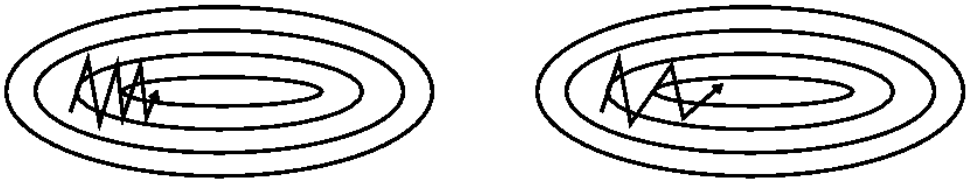
$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.7}$$

This way, it a) reduces the variance of the parameter updates, which can lead to more stable convergence; and b) can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient. Common mini-batch sizes range between 50 and 256, but can vary for different applications. Mini-batch gradient descent is typically the algorithm of choice when training a neural network and the term SGD usually is employed also when mini-batches are used.

Vanilla mini-batch gradient descent, however, does not guarantee good convergence. There are although some optimization algorithms that are widely used by the Deep Learning community to deal with that challenge. We will only discuss algorithms that were used in the implementation of our methods.

Momentum SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local

optimum as in Figure 2.15a.



a) SGD without momentum

b) SGD with momentum

Figure 2.15: Source: Genevieve B. Orr

Momentum [80] is a method that helps accelerate SGD in the relevant direction and dampens oscillations as can be seen in Figure 2.15b. It does this by adding a fraction γ of the update vector of the past time step to the current update vector.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{2.8}$$

The momentum term γ is usually set to 0.9 or a similar value.

Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity, if there is air resistance, i.e. $\gamma < 1$). The same thing happens to our parameter updates: The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

Nesterov accelerated gradient However, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory. We would like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.

Nesterov accelerated gradient (NAG) is a way to give our momentum term this kind of prescience. We know that we will use our momentum term $\gamma\, v_{t-1}$ to move the parameters θ. Computing $\theta - \gamma\, v_{t-1}$ thus gives us an approximation of the next position of the parameters (the gradient is missing for the full update), a rough idea where our parameters are going to be. We can now effectively look ahead by calculating the gradient not w.r.t. to our current parameters θ but w.r.t. the approximate future position of our parameters:

$$\begin{aligned} v_t &= \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{2.9}$$

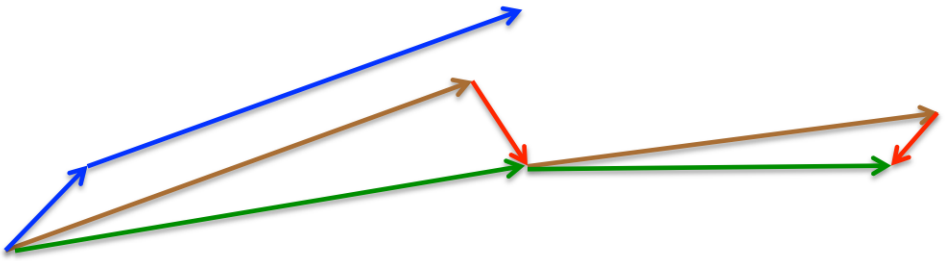


Figure 2.16: Nesterov update (Source: G. Hinton's lecture 6c)

Again, we set the momentum term γ to a value of around 0.9. While Momentum first computes the current gradient (small blue vector in Figure 2.16) and then takes a big jump in the direction of the updated accumulated gradient (big blue vector), NAG first makes a big jump in the direction of the previous accumulated gradient (brown vector), measures the gradient and then makes a correction (green vector). This anticipatory update prevents us from going too fast and results in increased responsiveness, which has significantly increased the performance of RNNs on a number of tasks.

Now that we are able to adapt our updates to the slope of our error function and speed up SGD in turn, we would also like to adapt our updates to each individual parameter to perform larger or smaller updates depending on their importance.

3 Semi-supervised learning

Depending on the key objective function of the systems, one may have a semi-supervised classification, a semi-supervised clustering, or a semi-supervised regression. In this work, we focus on classification and its' definition is as follows:

Semi-supervised classification. Given a training dataset that consists of both labeled instances and unlabeled instances, semi-supervised classification aims to train a classifier from both the labeled and unlabeled data, such that it is better than the supervised classifier trained only on the labeled data.

There have been a wide variety of SSL methods, including generative models , semi-supervised support vector machines, graph-based methods, and co-training, which provide a comprehensive overview of traditional SSL methods. Nowadays, deep neural networks have played a dominating role in many research areas. It is important to adopt the classic SSL framework and develop novel SSL methods for deep learning settings, which leads to deep semi-supervised learning (DSSL). DSSL studies how to effectively utilize both labeled and unlabeled data by deep neural networks. A considerable amount of DSSL methods have been proposed. According to the most distinctive features in semi-supervised loss functions and model designs, we classify DSSL into five categories, consistency regularization methods, proxy-label methods, generative methods, graph-based methods, and hybrid methods.

3.1 Background

In the following, we will present an overview of the techniques of SSL. Let $X = \{X_L, X_U\}$ denote the entire data set, including a small labeled data set $X_L = \{(x_i, y_i)\}_{i=1}^{L}$ with labels $Y_L = (y_1, y_2, \ldots, y_L)$ and a large scale unlabeled data set $X_U = \{(x_i)\}_{i=1}^{U}$, and $L \ll U$. We assume that the data have K classes and the first L examples within X are labeled by $\{y_i\}_{i=1}^{L} \in (y^1, y^2, \ldots, y^K)$. Formally, SSL aims to solve the following optimization problem,

$$\min_{\theta} \underbrace{\sum_{(x,y)\in X_L} \mathcal{L}_s(x, y, \theta)}_{\text{supervised loss}} + \alpha \underbrace{\sum_{(x)\in X_U} \mathcal{L}_u(x, \theta)}_{\text{unsupervised loss}} + \beta \underbrace{\sum_{(x)\in X} \mathcal{R}(x, \theta)}_{\text{regularization loss}}, \tag{3.1}$$

where $\mathcal{L}_s$ denote the per-example supervised loss, , cross-entropy for classification, $\mathcal{L}_u$ denotes the per-example unsupervised loss, and $\mathcal{R}$ denotes the

per-example regularization loss, , consistency loss or a designed regularization term. Note that unsupervised loss terms are often not strictly distinguished from regularization terms, as normally regularization terms are also not guided by label information. Lastly, θ denotes the model parameters and $\alpha, \beta \in R_{>0}$ denotes the scalar weight, which balances the loss terms. Different choices of the unsupervised loss and regularization term lead to different semi-supervised algorithms. Note that we do not make a clear distinction between unsupervised loss and regularization terms in many cases.

SSL first appeared in the form of self-training, which is also known as self-labeling or self-teaching. A model is first trained on labeled data. Then, iteratively, a portion of the unlabeled data is annotated using the trained model and added to the training set for the next training iteration. SSL took off in the 1970s after its success with iterative algorithms such as the expectation-maximization algorithm in which the labeled and unlabeled data are jointly used to maximize the likelihood of the model.

3.2 Assumptions

SSL aims to predict more accurately with unlabeled points than supervised learning that uses only labeled data. However, an essential prerequisite is that the example distribution should be under some assumptions. If this is not the case, SSL may not improve supervised learning and even degrade the prediction accuracy by misleading inferences.. The main assumptions in SSL are:

- **The Smoothness Assumption.** *If two points x_1, x_2 reside in a high-density region are close, then so should be their corresponding outputs y_1 y_2* [17]. Meaning that if two inputs are of the same class and belong to the same cluster, which is a high-density region of the input space, then their corresponding outputs need to be close. The inverse also holds true; if the two points are separated by a low-density region, the outputs must be distant from each other. This assumption can be quite helpful in a classification task, but not so much for regression.
- **The Cluster Assumption.** *If points are in the same cluster, they are likely to be of the same class* [17]. In this particular case of the smoothness assumption, we suppose that input data points form clusters, and each cluster corresponds to one of the output classes. The cluster assumption can also be seen as the low-density separation assumption: *The decision boundary should lie in the low-density regions.* The relation between the two assumptions is easy to see, if a given decision boundary lies in a high-density region, it will likely cut a cluster into two different classes, resulting in samples from different classes belonging to the same cluster, which is a violation of the cluster assumption. In this case, we can restrict

our model to have consistent predictions on the unlabeled data over some small perturbations pushing its decision boundary to low-density regions.
- **The Manifold Assumption.** *The (high-dimensional) data lie (roughly) on a low-dimensional manifold* [17]. In high dimensional spaces, where the volume grows exponentially with the number of dimensions, it can be quite hard to estimate the true data distribution for generative tasks. For discriminative tasks, the distances are similar regardless of the class type, making classification quite challenging. However, if our input data lies on some lower-dimensional manifold, we can try to find a low dimensional representation using the unlabeled data and then use the labeled data to solve the simplified task.

3.3 Approaches

There have been many SSL methods and approaches that have been introduced over the years. These algorithms can be broadly divided into the following categories:

- **Consistency Regularization (a.k.a Consistency Training).** Based on the assumption that if a realistic perturbation was applied to the unlabeled data points, the prediction should not change significantly. The model can then be trained to have a consistent prediction on a given unlabeled example and its perturbed version.
- **Proxy-label Methods.** Such methods leverage a trained model on the labeled set to produce additional training examples by labeling instances of the unlabeled set based on some heuristics. These approaches can also be referred to as *bootstrapping* [11] algorithms. We follow Ruder [85] and refer to them as proxy-label methods. Some examples of such methods are *Self-training*, *Co-training* and *Multi-View Learning*.
- **Generative Models.** Similar to the supervised setting, where the learned features on one task can be transferred to other downstream tasks. Generative models that are able to generate images from the data distribution $p(x)$ must learn transferable features to a supervised task $p(y|x)$ for a given task with targets y.
- **Graph-Based Methods.** The labeled and unlabeled data points can be considered as nodes of a graph, and the objective is to propagate the labels from the labeled nodes to the unlabeled ones by utilizing the similarity of two nodes x_i and x_j, which is reflected by how strong the edge e_{ij} between the two nodes.

In addition to these main categories, there is also some SSL work on *entropy minimization*, where we force the model to make confident predictions by minimizing the entropy of the predictions. Consistency training can also be

considered a proxy-label method, with a subtle difference, instead of considering the predictions as ground-truths and compute the cross-entropy loss, we enforce consistency of predictions by minimizing a given distance between the outputs. In this section, we will focus on consistency regularization, proxy-label and hybrid methods of those two.

SSL methods can also be categorized based on two dominant learning paradigms, **transductive learning** and **inductive learning**. Transductive learning aims to apply the trained classifier on the unlabeled instances observed at training time; in this case, it does not generalize to unobserved instances. This type of algorithm is mainly used on graphs, such as random walks for node embedding, where the objective is to label the unlabeled nodes of the graph that are present at training time. The more popular paradigm, inductive learning, aims to learn a classifier capable of generalizing to unobserved instances at test time.

3.3.1 Consistency regularization

In this section, we introduce the consistency regularization methods for semi-supervised deep learning. In these methods, a consistency regularization term is applied to the final loss function to specify the prior constraints assumed by researchers. Consistency regularization is based on the manifold assumption or the smoothness assumption, and describes a category of methods that the realistic perturbations of the data points should not change the output of the model [76]. Consequently, consistency regularization can be regarded to find a smooth manifold on which the dataset lies by leveraging the unlabeled data [7]. An overview of the methods can be seen in the figure below.

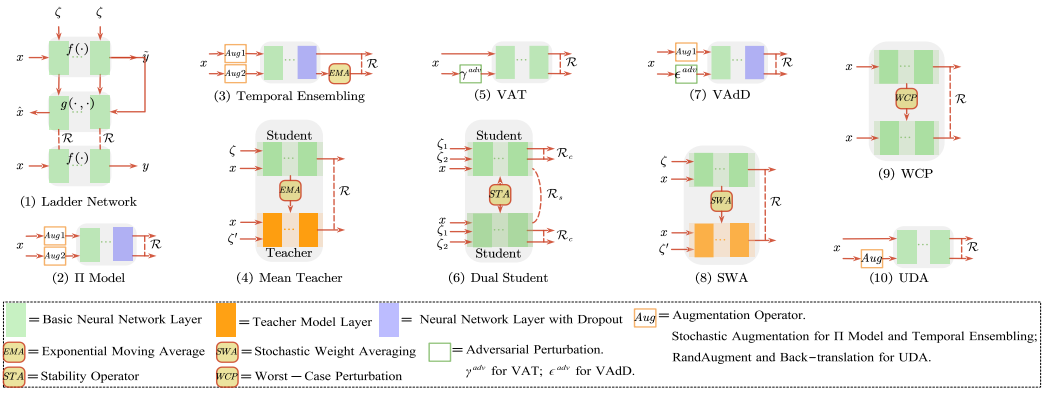


Figure 3.1: The diverse range of architectures used for consistency regularization semi-supervised methods. In addition to the identifiers in the figure, denotes the perturbation noise, and R is the consistency constraint. Image Source: [107].

The most common structure of consistency regularization SSL methods is the Teacher-Student structure. As a student, the model learns as before, and

as a teacher, the model generates targets simultaneously. Since the model itself generates targets, they may be incorrect and then used by themselves as students for learning. In essence, the consistency regularization methods suffer from confirmation bias [102], a risk that can be mitigated by improving the target's quality. Formally, following [53], we assume that dataset X consists of a labeled subset X_l and an unlabeled subset X_u. Let θ' denote the weight of the target, and θ denote the weights of the basic student. Concretely, given an unlabeled data point $x \in \mathcal{X}_U$ and its perturbed version $\hat{x}$, the objective is to minimize the distance between the two outputs $d(f_\theta(x), f_\theta(\hat{x}))$, where $f_\theta(x)$ is the prediction from model f_θ for input x. The popular distance measures d are mean squared error (MSE), Kullback-Leiber divergence (KL) and Jensen-Shannon divergence (JS). For two outputs $f_\theta(x)$ and $f_\theta(\hat{x})$ in the form of a probability distribution over the C classes, and $m = \frac{1}{2}(f_\theta(x) + f_\theta(\hat{x}))$, we can compute these measures as follows:

$$d_{\mathrm{MSE}}(f_\theta(x), f_\theta(\hat{x})) = \frac{1}{C}\sum_{k=1}^{C}(f_\theta(x)_k - f_\theta(\hat{x})_k)^2 \tag{3.2}$$

$$d_{\mathrm{KL}}(f_\theta(x), f_\theta(\hat{x})) = \frac{1}{C}\sum_{k=1}^{C} f_\theta(x)_k \log \frac{f_\theta(x)_k}{f_\theta(\hat{x})_k} \tag{3.3}$$

$$d_{\mathrm{JS}}(f_\theta(x), f_\theta(\hat{x})) = \frac{1}{2}d_{\mathrm{KL}}(f_\theta(x), m) + \frac{1}{2}d_{\mathrm{KL}}(f_\theta(\hat{x}), m) \tag{3.4}$$

Note that we can also enforce a consistency over two perturbed versions of x, $\hat{x}_1$ and $\hat{x}_2$.

Ladder Network. Ladder Network [83] is the first successful attempt towards using a Teacher-Student model that is inspired by a deep denoising AutoEncoder. The structure of the Ladder Network is shown in Fig. 3.1(1). In Encoder, noise ζ is injected into all hidden layers as the corrupted feedforward path $x + \zeta \rightarrow \frac{\text{Encoder}}{f(\cdot)} \rightarrow \tilde{z}_1 \rightarrow \tilde{z}_2$ and shares the mappings $f(\cdot)$ with the clean encoder feedforward path $x \rightarrow \frac{\text{Encoder}}{f(\cdot)} \rightarrow z_1 \rightarrow z_2 \rightarrow y$. The decoder path $\tilde{z}_1 \rightarrow \tilde{z}_2 \rightarrow \frac{\text{Decoder}}{g(\cdot,\cdot)} \rightarrow \hat{z}_2 \rightarrow \hat{z}_1$ consists of the denoising functions $g(\cdot,\cdot)$ and the unsupervised denoising square error on each layer consider as consistency loss between $\hat{z}_i$ and z_i. Through latent skip connections, the ladder network is differentiated from regular denoising AutoEncoder. This feature allows the higher layer features to focus on more abstract invariant features for the task. The unsupervised training loss $\mathcal{L}_u$ is then computed as the MSE between the activations of the clean encoder z and the reconstructed activations (after batch normalization), computed over all layers, from the input to the last layer, with a weighting λ_l for each layer's contribution to the total loss:

$$\mathcal{L}_u = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\sum_{l=0}^{L}\lambda_l d_{\mathrm{MSE}}(f_\theta(x), g(f_\theta(x+\zeta)) \tag{3.5}$$

Π Model. Unlike the perturbation used in Ladder Network, Π Model [82] is to create two random augmentations of a sample for both labeled and unlabeled data. Some techniques with non-deterministic behavior, such as randomized data augmentation, dropout, and random max-pooling, make an input sample pass through the network several times, leading to different predictions. The structure of the Π Model is shown in Fig. 3.1(2). In each epoch of the training process for Π Model, the same unlabeled sample propagates forward twice, while random perturbations are introduced by data augmentations and dropout. The forward propagation of the same sample may result in different predictions, and the Π Model expects the two predictions to be as consistent as possible. Therefore, it provides an unsupervised consistency loss function,

$$\mathcal{L}_u = w\frac{1}{|\mathcal{X}_U|}\sum_{x\in\mathcal{X}_U} d_{\mathrm{MSE}}(f_\theta(x+\zeta_1), f_\theta(x+\zeta_2)) \tag{3.6}$$

which minimizes the difference between the two predictions, with w as a weighting function, starting from 0 up to a fixed weight λ (30) after a given number of epochs.

Temporal Ensembling. Temporal Ensembling [60] is similar to the Π Model, which forms a consensus prediction under different regularization and input augmentation conditions. It modifies the Π Model by leveraging the Exponential Moving Average (EMA) of past epochs predictions. In other words, while Π Model needs to forward a sample twice in each iteration, Temporal Ensembling reduces this computational overhead by using EMA to accumulate the predictions over epochs as $\mathcal{T}_x$. Specifically, the ensemble outputs Z_i is updated with the network outputs z_i after each training epoch, $Z_i \leftarrow \alpha Z_i + (1-\alpha)\, z_i$, where α is a momentum term. During the training process, the Z can be considered to contain an average ensemble of $f(\cdot)$ outputs due to Dropout and stochastic augmentations. Thus, consistency loss is:

$$\mathcal{L}_u = w\frac{1}{|\mathcal{X}_U|}\sum_{x\in\mathcal{X}_U} d_{\mathrm{MSE}}(f_\theta(x+\zeta_1), \mathrm{EMA}(f_\theta(x+\zeta_2))) \tag{3.7}$$

Mean Teacher. Mean Teacher [102] averages model weights using EMA over training steps and tends to produce a more accurate model instead of directly using output predictions. The structure of Mean Teacher is shown in Fig. 3.1.

Mean Teacher consists of two models called Student and Teacher. The student model is a regular model similar to the Π Model, and the teacher model has the same architecture as the student model with exponential moving averaging the student weights. A training iteration of Mean Teacher is very similar to previous methods; the main difference is that Π-Model uses the same model as a student and a teacher $\theta' = \theta$, and Temporal Ensembling approximate a stable teacher $f_{\theta'}$ as an ensemble function with a weighted average of successive

predictions. While Mean Teacher defines the weights θ'_t of the teacher model $f_{\theta'}$ at a training step t as an EMA of successive student's weights θ:

$$\theta'_t = \alpha\theta'_{t-1} + (1-\alpha)\theta_t \tag{3.8}$$

The loss computation in this case is the sum of the supervised and unsupervised loss, where the teacher model is used to obtain the targets for the unsupervised loss for a given input x:

$$\mathcal{L}_u = w\frac{1}{|\mathcal{X}_U|}\sum_{x\in\mathcal{X}_U} d_{\text{MSE}}(f_\theta(x+\zeta), f_{\theta'}(x+\zeta')) \tag{3.9}$$

VAT. Virtual Adversarial Training [73] proposes the concept of adversarial attack for consistency regularization. The structure of VAT is shown in Fig. 3.1(5). This technique aims to generate an adversarial transformation of a sample, which can change the model prediction. Specifically, the adversarial training technique is used to find the optimal adversarial perturbation γ of a real input instance x such that $\gamma \leq \delta$. Afterward, the consistency constraint is applied between the model's output of the original input sample and the perturbed one, ,

$$\mathcal{L}_u = w\frac{1}{|\mathcal{X}_U|}\sum_{x\in\mathcal{X}_U} d_{\text{MSE}}(f_\theta(x), g_\theta(x+\gamma^{adv})), \tag{3.10}$$

where $\gamma^{adv} = argmax_{\gamma;\|\gamma\|\leq\delta} d_{\text{MSE}}(f_\theta(x), g_\theta(x+\gamma))$.

Dual Student. Dual Student [53] extends the Mean Teacher model by replacing the teacher with another student. The structure of Dual Student is shown in Fig. 3.1(6). One of the main drawbacks of using a Mean Teacher is that given a large number of training iterations, the teacher model weights will converge to that of the student model, and any biased and unstable predictions will be carried over to the student.

To solve this, Ke *et al.*propose a dual students step-up. Two student models with different initialization are simultaneously trained, and at a given iteration, one of them provides the targets for the other. To choose which one, we test for the most stable predictions that satisfy the following stability conditions:

- The predictions using two input versions, a clean x and a perturbed version $\hat{x}$ give the same results: $f(x) = f(\hat{x})$.
- Both predictions are confident, are far from the decision boundary. This can be tested by seeing if $f(x)$ (resp. $f(\hat{x})$) is greater than a confidence threshold ϵ, $\epsilon = 0.1$.

Given two student models, $f_{\hat{1}}$ and $f_{\hat{2}}$, an unlabeled input $x \in \mathcal{X}_U$ and its perturbed version $\hat{x}$. We compute four predictions: $f_{\theta_1}(x), f_{\theta_1}(\hat{x}), f_{\theta_2}(x)$, and

$f_{\theta_2}(\hat{x})$. In addition to training each model to minimize both the supervised and unsupervised losses, where the unsupervised loss is defined as follows:

$$\mathcal{L}_u = \lambda_1 \frac{1}{|\mathcal{X}_U|} \sum_{x \in \mathcal{X}_U} d_{\text{MSE}}(f_{\theta_i}(x), f_{\theta_i}(\hat{x})) \tag{3.11}$$

we also force one of the students to have similar predictions to its counterpart. To chose which one to update its weights, we check for both models' stability constraint; if the predictions one of the models is unstable, we update its weights. If both are stable, we update the model with the largest variation $\mathcal{E}^i = \|f_i(x) - f_i(\hat{x})\|^2$, the least stable. In this case, the least stable model is trained with an additional loss:

$$\lambda_2 \sum_{x \in \mathcal{X}_U} d_{\text{MSE}}(f_{\theta_i}(x), f_{\theta_j}(x)) \tag{3.12}$$

where λ_1 and λ_2 are hyperparameters specifying the contribution of each loss term.

SWA. Stochastic Weight Averaging (SWA) [50] improves generalization than conventional training. The aim is to average multiple points along the trajectory of stochastic gradient descent (SGD) with a cyclical learning rate and seek much flatter solutions than SGD. The consistency-based SWA [3] observes that SGD fails to converge on the consistency loss but continues to explore many solutions with high distances apart in predictions on the test data. The structure of SWA is shown in Fig. 3.1(7). The SWA procedure also approximates the Teacher-Student approach, such as Π Model and Mean Teacher with a single model. The authors propose fast-SWA, which adapts the SWA to increase the distance between the averaged weights by using a longer cyclical learning rate schedule and diversity of the corresponding predictions by averaging multiple network weights within each cycle. Generally, the consistency loss can be rewritten as follows:

$$\mathcal{L}_u = w \frac{1}{|\mathcal{X}_U|} \sum_{x \in \mathcal{X}_U} d_{\text{MSE}}(f_\theta(x), f_{\theta'}(x + \zeta)) \tag{3.13}$$

where,

$$\theta' = \text{SWA}(\theta) \tag{3.14}$$

VAdD. In VAT, the adversarial perturbation is defined as an additive noise unit vector applied to the input or embedding spaces, which has improved the generalization performance of SSL. Similarly, Virtual Adversarial Dropout (VAdD) [77] also employs adversarial training in addition to the Π Model. The structure of VAdD is shown in Fig. 3.1(8). Following the design of Π Model, the consistency constraint of VAdD is computed from two different dropped networks: one dropped network uses a random dropout mask, and the other applies adversarial training to the optimized dropout network. But first, we have to find the dropout conditions that are most sensitive to the model's

predictions. In a SSL setting, where we do not have access to the true labels, we use the model's predictions on the unlabeled data points to approximate the adversarial dropout mask ϵ^{adv}, which is subject to the boundary condition: $\|\epsilon^{adv} - \epsilon\|_2 \leq \delta H$ with H as the dropout layer dimension and a hyperparameter δ, which restricts adversarial dropout masks to be infinitesimally different from the random dropout mask ϵ. Without this constraint, the adversarial dropout might induce a layer without any connections. By restricting the adversarial dropout to be similar to the random dropout, we prevent finding such an irrational layer, which does not support backpropagation.

Similar to VAT, we start from a random dropout mask, we compute a KL-divergence loss between the outputs, with and without dropout, and given the gradients of the loss with respect to the activations before the dropout layer, we update the random dropout mask in an adversarial manner. The prediction function f_θ is divided into two parts, f_{θ_1} and f_{θ_2}, where $f_\theta(x, \epsilon) = f_{\theta_2}(f_{\theta_1}(x) \odot \epsilon)$, we start by computing an approximation of the Jacobian matrix as follows:

$$J(x, \epsilon) \approx f_{\theta_1}(x) \odot \nabla_{f_{\theta_1}(x)} d_{\mathrm{KL}}(f_\theta(x), f_\theta(x, \epsilon)) \tag{3.15}$$

Using $J(x, \epsilon)$, we can then update the random dropout mask ϵ to obtain ϵ^{adv}, so that if $\epsilon(i) = 0$ and $J(x, \epsilon)(i) > 0$ or $\epsilon(i) = 1$ and $J(x, \epsilon)(i) < 0$ at a given position i, we inverse the value of ϵ at that location. Resulting in ϵ^{adv}, which can then be used to compute the unsupervised loss:

$$\mathcal{L}_u = w \frac{1}{|\mathcal{X}_U|} \sum_{x \in \mathcal{X}_U} d_{\mathrm{MSE}}(f_\theta(x), f_\theta(x, \epsilon^{adv})) \tag{3.16}$$

WCP. A novel regularization mechanism for training deep SSL by minimizing the Worse-case Perturbation (WCP) is presented by Zhang *et al.*[111]. The structure of WCP is shown in Fig. 3.1(9). WCP considers two forms of WCP regularizations – additive and DropConnect perturbations, which impose additive perturbation on network weights and make structural changes by dropping the network connections, respectively. Instead of generating an ensemble of randomly corrupted networks, the WCP suggests enhancing the most vulnerable part of a network by making the most resilient weights and connections against the worst-case perturbations. It enforces an additive noise on the model parameters ζ, along with a constraint on the norm of the noise. In this case, the WCP regularization becomes,

$$\mathcal{L}_u = w \frac{1}{|\mathcal{X}_U|} \sum_{x \in \mathcal{X}_U} d_{\mathrm{MSE}}(f_\theta(x), g_{\theta'}(x)) \tag{3.17}$$

where $\theta' = \theta + \zeta$.

The second perturbation is at the network structure level by DropConnect, which drops some network connections. Specifically, for parameters θ, the

perturbed version is $\theta' = (1-\alpha)\theta$, and the $\alpha = 1$ denotes a dropped connection while $\alpha = 0$ indicates an intact one. By applying the consistency constraint, we have

$$\mathcal{L}_u = w\frac{1}{|\mathcal{X}_U|}\sum_{x\in\mathcal{X}_U} d_{\mathrm{MSE}}(f_\theta(x), f_{\theta'}(x)) \tag{3.18}$$

UDA. UDA stands for Unsupervised Data Augmentation [106] for image classification and text classification. The structure of UDA is shown in Fig. 3.1(10). This method investigates the role of noise injection in consistency training and substitutes simple noise operations with high-quality data augmentation methods, such as AutoAugment [23], RandAugment [22] for images, and Back-Translation [30, 91] for text. Following the consistency regularization framework, the UDA [106] extends the advancement in supervised data augmentation to SSL. As discussed above, let $f_\theta(x+\zeta)$ be the augmentation transformation from which one can draw an augmented example $(x+\zeta)$ based on an original example x. The consistency loss is:

$$\mathcal{L}_u = w\frac{1}{|\mathcal{X}_U|}\sum_{x\in\mathcal{X}_U} d_{\mathrm{MSE}}(f_\theta(x), f_\theta(x+\zeta)) \tag{3.19}$$

, where ζ represents the data augmentation operator to create an augmented version of an input x.

Summary. The core idea of consistency regularization methods is that the output of the model remains unchanged under realistic perturbation. Consistency constraints can be considered at three levels: input dataset, neural networks and training process. From the input dataset perspective, perturbations are usually added to the input examples: additive noise, random augmentation, or even adversarial training. We can drop some layers or connections for the networks, as WCP [111]. From the training process, we can use SWA to make the SGD fit the consistency training or EMA parameters of the model for some training epochs as new parameters.

3.3.2 Pseudo-label methods

Pseudo-label methods are the class of SSL algorithms that produce pseudo-labels on unlabeled data, using the prediction function itself or some variant of it without any supervision. These pseudo-labels are then used as targets together with the labeled data, providing some additional training information even if the produced labels are often noisy or weak and do not reflect the ground truth. These methods can be divided mainly into two groups: self-training, where the model itself produces the pseudo-labels, and disagreement-based learning or multi-view learning, where the pseudo-labels are produced by models trained on different views of the data. An overview of some of these methods can be seen below.

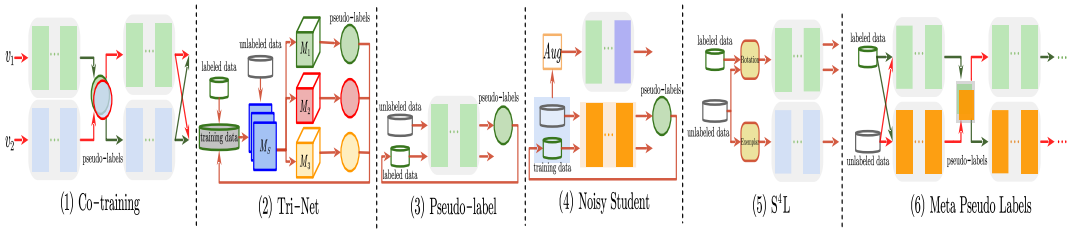


Figure 3.2: The diverse range of architectures used for pseudo-label semi-supervised methods. Image Source: [107].

Self-training

Self-training algorithm leverages the model's own confident predictions to produce the pseudo labels for unlabeled data. In other words, it can add more training data by using existing labeled data to predict the labels of unlabeled data. Because of its simplicity and generality, self-training is successfully applied in various tasks such as Named Entity Recognition [42], contour detection [113], machine translation [47], and object detection [72]. Specifically, [42] chooses the most confident named entity recognition predictions of the unlabeled data as the additional targets to boost the performance. [47] first trains the neural translation model on the parallel corpus and then uses the learned model to translate a monolingual corpus, wherein the monolingual corpus and its translations constitute an additional pseudo-parallel corpus.

EntMin Entropy Minimization (EntMin) [37] is a method of entropy regularization, which can be used to realize SSL by encouraging the model to make low-entropy predictions for unlabeled data and then using the unlabeled data in a standard supervised learning setting. In theory, entropy minimization can prevent the decision boundary from passing through a high-density data points region, otherwise it would be forced to produce low-confidence predictions for unlabeled data. The unsupervised is defined as follows:

$$\mathcal{L}_u = \frac{1}{|\mathcal{X}_U|} \sum_{x \in \mathcal{X}_U} \sum_{y \in Y} -f_\theta(y|x) \log f_\theta(y|x) \tag{3.20}$$

Pseudo-label Pseudo-label [64] proposes a simple and efficient formulation of training neural networks in a semi-supervised fashion, in which the network is trained in a supervised way with labeled and unlabeled data simultaneously. As illustrated in Fig. 3.2(3), the model is trained on labeled data in a usual supervised manner with a cross-entropy loss. For unlabeled data, the same model is used to get predictions for a batch of unlabeled samples. The maximum confidence prediction is called a pseudo-label, which has the maximum predicted probability.

That is, the pseudo-label model trains a neural network with the unsupervised loss function L_u, where:

$$\mathcal{L}_u = \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{K}\mathcal{H}(y_i^{'m}, f_i^{'m}), \tag{3.21}$$

where n is the number of mini-batch in labeled data for SGD, n' for unlabeled data, f_i^m is the output units of m's sample in labeled data, y_i^m is the label of that, $y_i^{'m}$ for unlabeled data, $y_i^{'m}$ is the pseudo-label of that for unlabeled data, $\alpha(t)$ is a coefficient balancing the supervised and unsupervised loss terms.

Noisy student Noisy Student [105] proposes a semi-supervised method inspired by knowledge distillation [45] with equal-or-larger student models. The framework is shown in Fig. 3.2(4). The teacher EfficientNet [101] model is first trained on labeled images to generate pseudo labels for unlabeled examples. Then a larger EfficientNet model as a student is trained on the combination of labeled and pseudo-labeled examples. These combined instances are augmented using data augmentation techniques such as RandAugment [22], and model noise such as Dropout and stochastic depth are also incorporated in the student model during training. After a few iterations of this algorithm, the student model becomes the new teacher to relabel the unlabeled data and this process is repeated. So, the unsupervised training objective is defined as follows:

$$\mathcal{L}_u = w\frac{1}{|\mathcal{X}_U|}\sum_{x\in\mathcal{X}_U}\mathcal{H}(\hat{y}, f_{\theta^s}(x+\zeta)) \tag{3.22}$$

where θ^s denotes the student model, $\hat{y}$ the pseudo-labels produced by a normal (i.e. not noised) teacher model and ζ the added noise.

S4L Self-supervised Semi-supervised Learning (S^4L) [110] tackles the problem of SSL by employing self-supervised learning [57] techniques to learn useful representations from the image databases. The architecture of S^4L is shown in Fig. 3.2(5). The conspicuous self-supervised techniques are predicting image rotation [34] and exemplar [28, 29]. Predicting image rotation is a pretext task that anticipates an angle of the rotation transformation applied to an input example. In S^4L, there are four rotation degrees $\{0°, 90°, 180°, 270°\}$ to rotate input images. The S^4L-Rotation loss is the cross-entropy loss on outputs predicted by those rotated images.

$$\mathcal{L}_{rot} = \frac{1}{|\mathcal{R}|}\sum_{r\in\mathcal{R}}\sum_{x\in X_u}\mathcal{L}(f_\theta(x^r), r) \tag{3.23}$$

, where $\mathcal{R}$ is the set of the 4 rotation degrees $\{0, 90, 180, 270\}$, x^r is the image x rotated by r, $f_\theta(\cdot)$ is the model with parameters θ,$\mathcal{L}$ is the cross-entropy loss.

This results in a 4-class classification problem. S^4L-Exemplar introduces an exemplar loss which encourages the model to learn a representation that is invariant to heavy image augmentations. Specifically, eight different instances of each image are produced by inception cropping [100], random horizontal mirroring, and HSV-space color randomization as in [28]. Following [57], the loss term on unsupervised images uses the batch hard triplet loss [44] with a soft margin.

MPL In the SSL, the target distributions are often generated on unlabeled data by a shaped teacher model trained on labeled data. The constructions for target distributions are heuristics that are designed prior to training, and cannot adapt to the learning state of the network training. Meta Pseudo Labels (MPL) [79] designs a teacher model that assigns distributions to input examples to train the student model. Throughout the course of the student's training, the teacher observes the student's performance on a held-out validation set, and learns to generate target distributions so that if the student learns from such distributions, the student will achieve good validation performance. The training procedure of MPL involves two alternating processes. As shown in Fig. 3.2(6), the teacher $g_\phi(\cdot)$ produces the conditional class distribution $g_\phi(x)$ to train the student. The pair $(x, g_\phi(x))$ is then fed into the student network $f_\theta(\cdot)$ to update its parameters θ from the cross-entropy loss. After the student network updates its parameters, the model evaluates the new parameters θ' based on the samples from the held-out validation dataset. Since the new parameters of the student depend on the teacher, the dependency allows us to compute the gradient of the loss to update the teacher's parameters.

Summary In general, self-training is a way to get more training data by a series of operations to get pseudo-labels of unlabeled data. Both EntMin [37] and Pseudo-label [64] use entropy minimization to get the pseudo-label with the highest confidence as the ground truth for unlabeled data. Noisy Student [105] utilizes a variety of techniques when training the student network, such as data augmentation, dropout and stochastic depth. S^4L [110] not only uses data augmentation techniques, but also adds another 4-category task to improve model performance. MPL [79] modifies Pseudo-label [64] by deriving the teacher network's update rule from the feedback of the student network. Emerging techniques (eg, rich data augmentation strategies, meta-learning, self-supervised learning) and network architecture (eg. EfficientNet [101]) provide powerful support for the development of self-training methods.

Disagreement-based learning

The idea of disagreement-based SSL is to train multiple learners for the task and exploit the disagreement during the learning process [116]. In such model

designs, two or three different networks are trained simultaneously and label unlabeled samples for each other. Disagreement-based methods differ in whether the data has multiple views , Co-training [12, 81] for multiview data or Tri-Net [18] for single-view data.

Co-training Co-training [12] requires that each data point x can be represented using two conditionally independent views $v_1(x)$ and $v_2(x)$, and that each view is sufficient to train a good model. After training two prediction functions f_{θ_1} and f_{θ_2} on a specific view on the labeled set $\mathcal{D}_l$. We start the proxy labeling procedure. At each iteration, an unlabeled data point is added to the training set of the model f_{θ_i} if the other model f_{θ_j} outputs a confident prediction with a probability higher than a threshold τ. This way, one of the models provides newly labeled examples where the other model is uncertain. Co-training has been combined with deep learning for some applications, such as object recognition [20] by utilizing RGB-D data, with RGB and depth as the two views used to train the two models, or for combining multi-modal data [2] (image and text) by training each model on a given modality and use it to provide pseudo-labels for other models. However, in many cases, the data have only one view rather than two, in this instance, different learning algorithms or different parameter configurations to learn two different classifiers can be employed. The two views $v_1(x)$ and $v_2(x)$ can also be generated by injecting noise or by applying different augmentations, for example, Qiao *et al.*[81] used adversarial perturbations to produce new views for deep co-training for image classification, where the models are encouraged to have the same predictions on $\mathcal{X}_l$ but make different errors when they are exposed to adversarial attacks. Here, $v_1(x)$ and $v_2(x)$ are convolutional representations of x before the final fully-connected layer $fi()$ that classifies $vi(x)$ to one of the categories.

$$\mathcal{L}_u = w\frac{1}{|\mathcal{X}_U|}\sum_{x\in\mathcal{X}_U} d_{\text{JS}}(f_1(v_1(x)), f_2(v_2(x))) \tag{3.24}$$

Democratic co-training [115]. An extension of Co-training, consists of replacing the different views of the input data with a number of models with different architectures and learning algorithms, which are first trained on the labeled examples. The trained models are then used to label a given example x if a majority of models confidently agree on its label.

Tri-training Tri-training [117] tries to overcome the lack of data with multiple views and reduce the bias of the predictions on unlabeled data produced with self-training by utilizing the agreement of three independently trained models instead of a single model. First, the labeled data $\mathcal{X}_l$ is used to train three prediction functions: f_{θ_1}, f_{θ_2} and f_{θ_3}. An unlabeled data point $x \in \mathcal{D}_u$ is then added to the supervised training set of the function f_{θ_i} if the other two models

agree on its predicted label. The training stops if no data points are being added to any of the models' training sets. Tri-training requires neither the existence of multiple views nor unique learning algorithms, making it more generally applicable. Using Tri-training with neural networks can be very expensive, requiring predictions for each one of the three models on all the unlabeled data. Ruder *et al.*[85] propose to sample a limited number of unlabeled data points at each training epoch, the candidate pool size is increased as the training progresses and the models become more accurate.

Multi-task tri-training [85] can also be used to reduce the time and sample complexity, where all three models share the same feature-extractor with model-specific classification layers. This way, the models are trained jointly with an additional orthogonality constraint on two of the three classification layers to be added to loss term, to avoid learning similar models and falling back to the standard case of self-training. Tri-Net [27] also falls in this category, with a shared module for joint learning and three output modules for tri-training, in addition to utilizing output smearing [15] to initialize these modules. After the pseudo-labeling iteration, a fine-tuning stage is conducted on the labeled data to augment diversity and eliminate unstable and suspicious pseudo-labeled data.

Cross-view training Clark *et al.*[21] propose Cross-View Training, where the model is trained to produce consistent predictions across different views of the inputs. Instead of using a single model as a teacher and a student, they propose to use a shared encoder, and then add auxiliary prediction modules that transform the encoder representations into predictions, these modules are then divided into auxiliary student modules and a primary teacher module. The input to each student prediction module is a subset of the model's intermediate representations corresponding to a restricted view of the input, such as feeding one of the student only the forward LSTM from a given Bi-LSTM layer, so it makes predictions without seeing any tokens to the right of the current one (fig:cvt). The primary teacher module in trained only on labeled examples, and is responsible of generating the pseudo-labels taking as input the full view of the unlabeled inputs, the students are trained to have consistent predictions with the teacher module. Given an encoder f, a teacher module t and K student modules s_i with $i \in [0, K]$, where each student receives a limited view of the input, the training unsupervised objective is written as follows:

$$\mathcal{L}_u = \frac{1}{|\mathcal{X}_U|} \sum_{x \in \mathcal{X}_U} \sum_{i=1}^{K} d_{\text{MSE}}(t(e(x)), s_i(f_\theta(x))) \tag{3.25}$$

Cross-view training takes advantage of unlabeled data by improving the encoder's representation learning. The student prediction modules can learn

from the teacher module predictions because this primary module has a better, unrestricted view of the inputs. As the student modules learn to make accurate predictions despite their restricted views of the input, they improve the quality of the representations produced by the encoder. Which, in turn, improves the full model, which uses the same shared representations.

Summary The disagreement-based SSL methods exploit the unlabeled data by training multiple learners, and the "disagreement" among these learners is crucial. When the data has two sufficient redundancy and conditional independence views, Deep Co-training [81] improves the disagreement by designing a View Difference Constraint. Tri-Net [18] obtains three labeled datasets by bootstrap sampling and trains three different learners. These methods in this category are less affected by model assumptions, non-convexity of the loss function and the scalability of the learning algorithms.

3.3.3 Hybrid methods

An emerging line of work in SSL is a set of hybrid approaches that try to unify the current dominant methods in SSL in a single framework, achieving better performances.

MixMatch Berthelot *et al.*[9] propose a *hybrid* approach which gracefully unifies ideas and components from the dominant paradigms for SSL, resulting in an algorithm that is greater than the sum of its parts and surpasses the performance of the traditional approaches.

MixMatch takes as input a batch from the labeled set $\mathcal{X}_l$ containing pairs of inputs and their corresponding one-hot targets, a batch from the unlabeled set $\mathcal{D}_u$ containing only unlabeled data, and a set of hyperparameters: the sharpening softmax temperature T, the number of augmentations K, and the Beta distribution parameter α for MixUp. Producing a batch of augmented labeled examples and a batch of augmented unlabeled examples with their proxy labels. These augmented examples can then be used to compute the losses and train the model. Precisely, MixMatch consists of the following steps:

- **Step 1: Data Augmentation.** Using a given transformation, a labeled example $x \in \mathcal{X}_l$ from the labeled batch is transformed, producing its augmented versions $\tilde{x}$. For an unlabeled example $x \in \mathcal{X}_U$, the augmentation function is applied K times, resulting in K augmented versions of the unlabeled examples $\tilde{x}_1$, ..., $\tilde{x}_K$.
- **Step 2: Label Guessing.** The second step consists of producing proxy labels for the unlabeled examples. First, we generate the predictions for the K augmented versions of each unlabeled example using the predictions function f_θ. The K predictions are then averaged together, obtaining a

pseudo-label $\hat{y} = 1/K \sum_{k=1}^{K}(\hat{y}_k)$ for each one of the augmentations of the unlabeled example x: $(\tilde{x}_1, \hat{y})$, ..., $(\tilde{x}_K, \hat{y})$.

- **Step 3: Sharpening.** To push the model to produce confident predictions and minimize the entropy of the output distribution, the generated proxy labels $\hat{y}$ in step 2 in the form of a probability distribution over C classes are sharpened by adjusting the temperature of the categorical distribution, computed as follows where $(\hat{y}^u)_k$ refers to the probability of class k out of C classes:

$$(\hat{y})_k = (\hat{y})_k^{\frac{1}{T}} / \sum_{k=1}^{C} (\hat{y})_k^{\frac{1}{T}} \tag{3.26}$$

- **Step 4 MixUp.** The previous steps resulted in two new augmented batches, a batch $\mathcal{L}$ of augmented labeled examples and their target, and a batch $\mathcal{U}$ of augmented unlabeled examples and their sharpened pseudo-labels. Note that the size of $\mathcal{U}$ is K times larger than the original batch given that each example $x \in \mathcal{X}_U$ is replaced by its K augmented versions. In the last step, we mix these two batches. First, a new batch merging both batches is created $\mathcal{W} = \text{Shuffle}(\text{Concat}(\mathcal{L}, \mathcal{U}))$. $\mathcal{W}$ is then divided into two batches: $\mathcal{W}_1$ of the same size as $\mathcal{L}$ and $\mathcal{W}_2$ of the same size as $\mathcal{L}$. Using the Mixup operation that is slightly adjusted so that the mixed example is closer the labeled examples, the final step is to create new labeled and unlabeled batches by mixing the produced batches together using Mixup as follows:

$$\mathcal{L}' = \text{MixUp}(\mathcal{L}, \mathcal{W}_1) \tag{3.27}$$

$$\mathcal{U}' = \text{MixUp}(\mathcal{U}, \mathcal{W}_2) \tag{3.28}$$

After creating two augmented batches $\mathcal{L}'$ and $\mathcal{U}'$ using MixMatch, we can then train the model using the standard SSL losses by computing the CE loss for the supervised loss:

$$\mathcal{L}_s = \frac{1}{|\mathcal{L}'|} \sum_{x,y \in \mathcal{L}'} \text{H}(y, f_\theta(x))) \tag{3.29}$$

and the consistency loss for the unsupervised loss:

$$\mathcal{L}_u = w \frac{1}{|\mathcal{U}'|} \sum_{x,\hat{y} \in \mathcal{U}'} d_{\text{MSE}}(\hat{y}, f_\theta(x)) \tag{3.30}$$

ReMixMatch Berthelot *et al.*[10] propose to improve MixMatch by introducing two new techniques: **distribution alignment** and **augmentation anchoring**. Distribution alignment encourages the marginal distribution of predictions on unlabeled data to be close to the marginal distribution of ground-truth

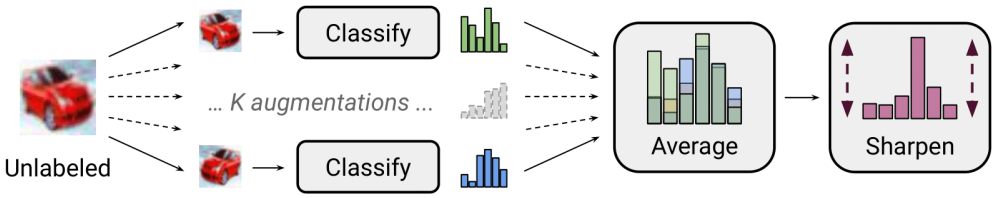


Figure 3.3: **MixMatch.** The procedure of label guessing process used in MixMatch, taking as input a batch of unlabeled examples, and outputting a batch of K augmented version of each input, with a corresponding sharpened proxy labels. Image Source: [9].

labels. Augmentation anchoring feeds multiple strongly augmented versions of the input into the model, encouraging each output to be close to the prediction for a weakly-augmented version of the same input.

Distribution alignment. In order to force that the aggregate of predictions on unlabeled data matches the distribution of the provided labeled data. Over the course of training, a running average $\tilde{y}$ of the model's predictions on unlabeled data is maintained over the last 128 batches. For the marginal class distribution $p(y)$, it is estimated based on the labeled examples seen during training. Given a prediction $f_\theta(x)$ on the unlabeled example x, the output probability distribution is aligned as follows: $f_\theta(x) =$ Normalize $(f_\theta(x) \times p(y)/\tilde{y})$.

Augmentation Anchoring. MixMatch uses a simple flip-and-crop augmentation strategy, ReMixMatch replaces the weak augmentations with strong augmentations learned using a control theory based augmentation strategy following AutoAugment. With such augmentations, the model's prediction for a weakly augmented unlabeled image is used as a proxy label for many strongly augmented versions of the same image in a standard cross-entropy loss.

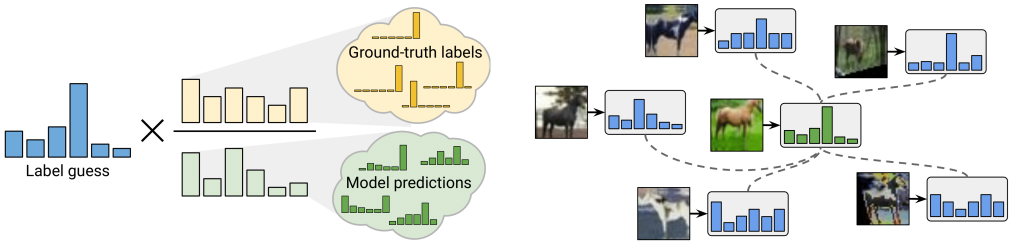
For training, MixMatch is applied to the unlabeled and labeled batches, with the application of distribution alignment and replacing the K weakly augmented example with a strongly augmented example, in addition to using the weakly augmented examples for predicting proxy labels for the unlabeled strongly augmented examples. With two augmented batches $\mathcal{L}'$ and $\mathcal{U}'$, the supervised and unsupervised losses are computed both using the cross-entropy loss as follows:

$$\mathcal{L} = \mathcal{L}_s + w\mathcal{L}_u = \frac{1}{|\mathcal{L}'|} \sum_{x,y \in \mathcal{L}'} \mathrm{H}(y, f_\theta(x))) + w \frac{1}{|\mathcal{U}'|} \sum_{x,\hat{y} \in \mathcal{U}'} \mathrm{H}(\hat{y}, f_\theta(x))) \tag{3.31}$$

In addition to these losses, the authors add a self-supervised loss. First, a new unlabeled batch $\hat{\mathcal{U}}'$ of examples is created by rotating all of the examples with an angle $r \sim \{0, 90, 180, 270\}$. The rotated examples are then used to compute a self-supervised loss, where the classification layer on top of the model predicts the correct applied rotation, in addition to the cross-entropy loss over

the rotated examples:

$$\mathcal{L}_{SL} = w' \frac{1}{|\hat{\mathcal{U}}'|} \sum_{x,\hat{y} \in \hat{\mathcal{U}}'} \mathrm{H}(\hat{y}, f_\theta(x))) + \lambda \frac{1}{|\hat{\mathcal{U}}'|} \sum_{x \in \hat{\mathcal{U}}'} \mathrm{H}(r, f_\theta(x))) \tag{3.32}$$



Distribution alignment. Augmentation anchoring.

Figure 3.4: **ReMixMatch.** *Left.* Distribution alignment adjusts the guessed labels distributions to match the ground-truth class distribution divided by the average model predictions on $\mathcal{D}_u$. *Right.* Augmentation anchoring uses the prediction obtained using a weakly augmented image as targets for a strongly augmented version of the same image. Image Source: [10].

FixMatch FixMatch [96] combines consistency regularization and pseudo-labeling while vastly simplifying the overall method. In FixMatch (fig. 3.5), both the supervised and unsupervised losses are computed using a cross-entropy loss. For labeled examples, the provided targets are used. For unlabeled examples $x \in \mathcal{X}_U$, a weakly augmented version is first computed using weak augmentation function A_w. As in self-training, the predicted label is then considered as a pseudo-label if the highest class probability is greater than a threshold τ. With a pseudo-label, K strongly augmented examples are generated using a strong augmentation function A_s. We then assign to these augmented versions the pseudo-label obtained with the weakly labeled version. The unsupervised loss can be written as follows:

$$\mathcal{L}_u = w \frac{1}{K|\mathcal{X}_U|} \sum_{x \in \mathcal{X}_U} \sum_{i=1}^{K} \mathbb{1}(\max(f_\theta(A_w(x))) \geq \tau) \mathrm{H}(f_\theta(A_w(x)), f_\theta(A_s(x))) \tag{3.33}$$

In FixMatch, weak augmentation is a standard flip-and-shift augmentation strategy, randomly flipping images horizontally with a probability. For strong augmentation, there are two approaches which are based on [23], , RandAugment [22] and CTAugment [10]. Moreover, Cutout [25] is followed by either of these strategies. Since our method uses FixMatch as a base for our cleaning process, it is better explained in the next section.

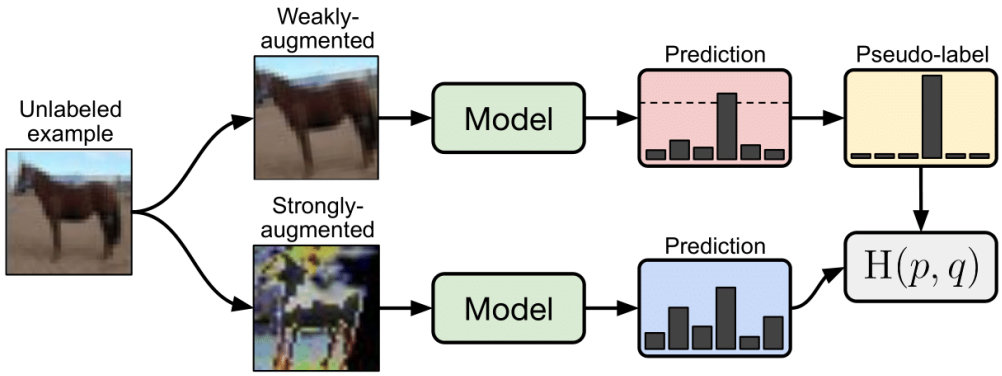


Figure 3.5: **FixMatch.** The model prediction on a weakly augmented input is considered as target if the maximum output class probability is above threshold, this target can then be used to train the model on a strongly augmented version of the same input using standard cross-entropy loss. Image Source: [10].

SelfMatch SelfMatch [55] combines the power of contrastive self-supervised learning and consistency regularization. It consists of two stages: (1) self-supervised pre-training based on contrastive learning and (2) semi-supervised finetuning based on augmentation consistency regularization. It adopts SimCLR [19] for the first stage, and FixMatch [96] for the second. The second stage along with its unsupervised loss L_u of the second stage is identical to Fixmatch's. For the second stage: SimCLR consists of four components: data augmentation $\mathcal{T}(\cdot)$, base encoder $f(\cdot)$, projection head $g(\cdot)$, and contrastive loss $\mathcal{L}_{u2}$. For data augmentation, SelfMatch uses random crop and color distortion. The base encoder is a ResNet-34. For the projection head, a MLP consisting of two layers with Dropout and ReLU activation is used. Finally, the contrastive loss is formulated as follows:

$$\mathcal{L}_{u2} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2K} \mathbb{1}(k \neq i)\exp(\text{sim}(z_i, z_k)/\tau)} \tag{3.34}$$

, where $\text{sim}(\cdot)$ is a similarity measure function, τ is a temperature parameter scaling the similarity, $\mathbb{1}(k \neq i)$ is an indicator function evaluating to 1 iff $k \neq i$.

By combining two closely related but complementary stages, SelfMatch improves from other methods such as S4L [110] that also uses self-supervised pre-training.

CoMatch In CoMatch [66], each image has two compact representations: a class probability produced by the classification head and a low-dimensional embedding produced by the projection head. The two representations interact with each other and jointly evolve in a co-training framework. Specifically, the classification head is trained using memory-smoothed pseudo-labels, where pseudo-labels are refined by aggregating information from nearby samples in the

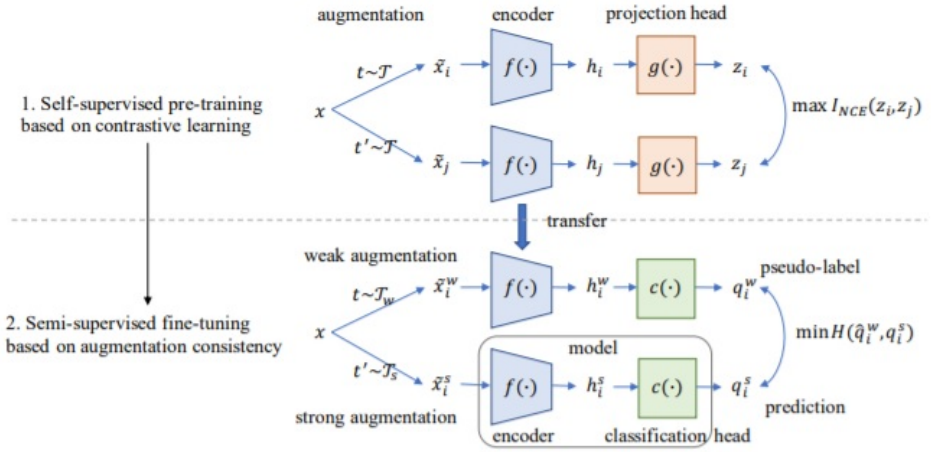


Figure 3.6: **SelfMatch.** Overview of the method depicting the 2 stages followed. Image Source: [55].

embedding space. The projection head is trained using contrastive learning on a pseudo-label graph, where samples with similar pseudo-labels are trained to have similar embeddings. CoMatch unifies dominant ideas including consistency regularization, entropy minimization, contrastive learning, and graph-based SSL.

Different from most existing semi-supervised and self-supervised learning methods, CoMatch jointly learns the encoder $f(\cdot)$, the classification head $h(\cdot)$, and the projection head $g(\cdot)$. $g(\cdot)$ is actually a non-linear projection head (a MLP), which transforms a feature $f(x)$ into a normalized low-dimensional embedding $z(x) = g(f(x))$. CoMatch jointly optimizes three losses: (1) a supervised classification loss on labeled data $\mathcal{L}_s$, (2) an unsupervised classification loss on unlabeled data $\mathcal{L}_{u1}$, and (3) a graph-based contrastive loss on unlabeled data $\mathcal{L}_{u2}$. $\mathcal{L}_s$ is defined as the cross-entropy between the ground-truth labels and the model's predictions, while the unsupervised classification loss $\mathcal{L}_{u1}$ is defined as the cross-entropy between the pseudo-labels q_b and the model's predictions, and is the same as FixMatch's.

Following FixMatch [96], CoMatch retains pseudo-labels whose largest class probability are above a threshold τ. Different from FixMatch, its soft pseudo-labels q_b are not converted to hard labels for entropy minimization. Instead, it achieves entropy minimization by optimizing the contrastive loss $\mathcal{L}_{u2}$.

The overall unsupervised training objective is:

$$\mathcal{L}_u = w_1 \mathcal{L}_{u1} + w_2 \mathcal{L}_{u2}, \tag{3.35}$$

where w_1 and w_2 are scalar hyperparameters to control the weight of the unsupervised losses.

The contrastive unsupervised loss is defined as follows:

$$L_{u2} = -\log \frac{\exp(z(\text{Aug}(x_i)) \cdot z(\text{Aug}(x_i))/t)}{\sum_{j=1}^{N} \exp(z(\text{Aug}(x_i)) \cdot z(\text{Aug}(x_j))/t)} \tag{3.36}$$

where $\text{Aug}(\cdot)$ is a stochastic transformation, t is a scalar temperature parameter, and x_j include x_i and $N-1$ other images (negative samples). Self-supervised contrastive learning can be interpreted as a form of class-agnostic consistency regularization, which enforces the same image with different augmentations to have similar embeddings, while different images have different embeddings.

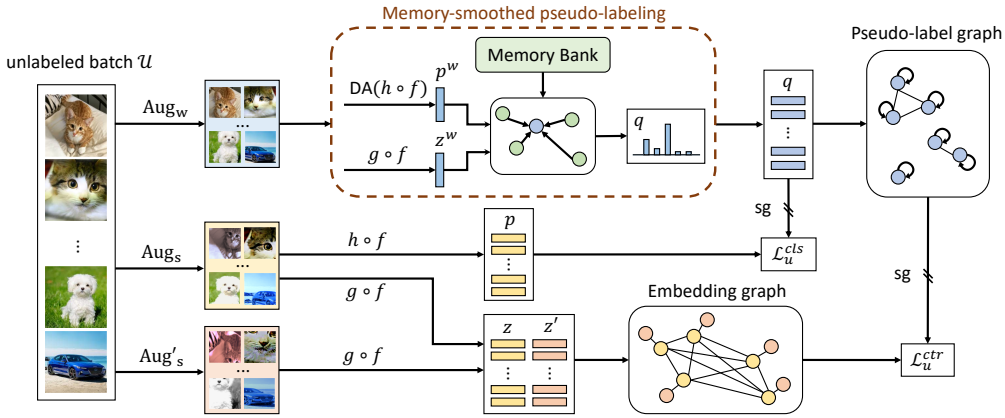


Figure 3.7: **CoMatch.** Given a batch of unlabeled images, their weakly-augmented images are used to produce memory-smoothed pseudo-labels, which are used as targets to train the class prediction on strongly-augmented images. A pseudo-label graph with self-loop is constructed to measure the similarity between samples, which is used to train an embedding graph such that images with similar pseudo-labels have similar embeddings. Image Source: [66]

Summary As discussed above, the hybrid methods unifies the most successful approaches in SSL, such as pseudo-labeling, entropy minimization and consistency regularization, and adapt them to achieve state-of-the-art performance.

3.4 Learning with noisy labels

In the literature, the solutions of learning with noisy labels can be classified into two types: 1) detecting noisy labels and then cleansing potential noisy labels or reduce their impacts in the following training; 2) directly training noise-robust models with noisy labels.

3.4.1 Noise-cleansing

Koh and Liang [56] propose an influence functions to measure which samples are "harmful" to model training. As the proposed approach requires intensive

computation on the impact of every training sample on all the validation samples, it is hardly implemented in industry. In [112], Zhang et al. propose an approach to detect both outlier samples and hard training set bugs using a small group of trusted data. As this approach requires a strong convex assumption on the objective function, it cannot be applied to most of the deep models because such an assumption can hardly hold. In [65], Lee et al. propose a joint neural embedding network named Clean-Net. This approach summarizes the knowledge of label noise from a fraction of manually verified classes. Transfer learning is then conducted to transfer the knowledge to other classes to handle label noise. The human verification lowers the applicability of this work. In [40], Han et al. propose a noisy label detection approach, named Co- teaching, in which two deep networks are trained simultaneously. Each network selects which samples the other network uses for training. Either of the networks teaches each other to identify noisy labels. Another similar work is proposed in [70]. In recent studies, curriculum learning [39] is applied to learning with noisy labels. In [38], Guo et al. propose CurriculumNet, in which training data are divided into several subsets by ranking their complexity via distribution density. The subsets are formed as a curriculum to teach the model in understanding label noise gradually. A similar idea is proposed in [52]. In this work, a MentorNet is trained to identify potential noisy labels. It then provides a data-driven curriculum for a StudentNet which is trained on the relatively clean data samples.

3.4.2 Noise-robust models

In [36], label noise is modeled by additional softmax layers to estimate the transition between correct labels and noisy labels. In [104], Xiao et al. propose a probabilistic model to describe the relations among images, truth labels, noisy labels and noise types. The probabilistic model requires a small set of verified clean labels. In [84], Reed and Lee propose the notion consistent to model noisy labels. Sample reconstruction errors are applied as the consistency objective to estimate the noise distribution. All the above noise-transition-estimation-based approaches aim at discovering the pattern of noise in data. Note that all the prior work of learning with noisy labels requires either particular assumptions (e.g., noise distribution estimation) or extra specifically designed loss functions or networks (e.g., Co-teaching and MentorNet). Those limit their applicability in practice. Our methods are based on O2U-net which is a more straightforward but effective approach. Different from the prior work, it only requires only appropriately adjusting the hyper-parameters of deep networks.

4 Our Methods

4.1 Introduction

Modern approaches to many computer vision problems exploit deep neural networks. These are popular for being very efficient and providing great performance at test time. The downside is a requirement of large amounts of training examples, which are labeled either by humans or automatically on proxy tasks.

Visual data are available in large quantities, however, data reliably annotated by humans are still very scarce. Obtaining large amounts of annotated training data for every single task is not only impractical, potentially costly, but it also turns out to be error prone. The low quality of crowd-sourced annotation is a common motivation to minimize the need of annotation.

Since labeling a large number of examples requires considerable time and cost, label efficient learning algorithms are in high demand. Semi-supervised learning is one of the attractive approaches that addresses the label inefficiency problem. Semi-supervised learning enables a deep neural network to learn with small labeled data by making use of large unlabeled data.

In this work, we introduce the two semi-superviesd methods we designed, implemented and experimented on. CleanMatch and WeightMatch are a combination of three components: consistency regularization, pseudo-labeling and learning from noisy data.

Their main novelty comes from the combination of the two first as used in FixMatch as well as the use of label cleaning to make the best use of them.

CleanMatch combines the power of predicting pseudo-labels in semi-supervised learning with label cleaning in order to expand the labeled dataset and ultimately train on the new split of labeled-unlabeled examples.

WeightMatch leverages the same label cleaning process but solely on the labeled set, to estimate a weight reflecting the importance of each labeled example on the training.

4.2 Preliminaries

In this section we formulate the *semi-supervised learning* problem and then we discuss the classifier, different loss functions that are commonly used in prior work, and finally a transductive learning approach that our method is based on. In our experiments we use a *convolutional neural network* (CNN) to perform

image classification, but this formulation applies to any network architecture in any domain.

Problem formulation We assume a collection of n examples comprising the dataset $\mathcal{X} = (x_1, \ldots, x_l, x_{l+1}, \ldots, x_n)$ with $x_i \in \mathcal{X}$. The first l examples x_i for $i \in L = \{1, \ldots, l\}$, denoted by $\mathcal{X}_\mathcal{L} = \{(x_i, y_i)\}_{i=1}^{L}$, where $\mathcal{Y}_\mathcal{L} = (y_1, y_2, \ldots, y_L)$ are the corresponding labels. where $mathcalC = \{1, \ldots, c\}$ is a discrete label set for c classes. The remaining $u = n - l$ examples x_i for $i \in \mathcal{U} = \{l+1, \ldots, n\}$, denoted by X_U, are unlabeled. The goal in SSL is to use all examples $\mathcal{X}$ and labels $\mathcal{Y}_\mathcal{L}$ to train a classifier that maps previously unseen samples to class labels.

Classifier The network takes an input example from $\mathcal{X}$ and produces a vector of class confidence scores. We denote it by $f_\theta : \mathcal{X} \to R^c$, where θ are the network parameters. It is conceptually divided in two parts. The first is a feature extraction network $\phi_\theta : \mathcal{X} \to R^d$ mapping the input to a feature vector, or descriptor. The second typically consists of a *fully connected* (FC) layer applied on top of ϕ_θ and followed by softmax, producing a vector of *confidence scores*. Function f_θ is the mapping from input space directly to confidence scores. The output of the network for the i-th example is $f_\theta(x_i)$ and the *prediction* is the one of maximum confidence score

$$\hat{y}_i = \arg\max_j f_\theta(x_i)_j, \tag{4.1}$$

where subscript j denotes the j-th dimension of the vector.

Consistency regularization It is an important component of recent state-of-the-art SSL algorithms. Consistency regularization utilizes unlabeled data by relying on the assumption that the model should output similar predictions when fed perturbed versions of the same image. The model is trained both via a standard supervised classification loss and on unlabeled data via the loss function

$$\|p(y|\mathrm{a}_1(x)) - p(y|\mathrm{a}_2(x))\|_2^2 \tag{4.2}$$

Note that in this work p implies the function f_θ. Extensions to this idea include using an adversarial transformation in place of $a1$ and a_2 using a running average or past model predictions for one invocation of p, using a cross-entropy loss in place of the squared l^2 loss, using stronger forms of augmentation, and using consistency regularization as a component in a larger SSL pipeline.

Pseudo-labeling It is a common method in many SSL algorithms, which encourages the classifier's decision boundary to pass through low-density regions

of the data distribution. It is either achieved explicitly by minimizing the entropy of $p(y|x)$ on unlabeled samples, or implicitly by constructing low-entropy pseudo-labels on unlabeled samples and using them as training targets in a cross-entropy loss.

In our case, it is the process of assigning a pseudo-label $\hat{y}_i$ to each example x_i for $i \in \mathcal{U}$. Denoting by $\mathcal{Y}_\mathcal{U} = (\hat{y}_{l+1}, \dots, \hat{y}_n)$ the collection of pseudo-labels for $\mathcal{X}_\mathcal{U}$, the following additional *pseudo-label* loss term applies

$$L_p(\mathcal{X}_\mathcal{U}, \mathcal{Y}_\mathcal{U}; \theta) = \sum_{i=l+1}^{n} \mathrm{H}\left(\hat{y}_i, f_\theta(x_i)\right), \tag{4.3}$$

where again H is the cross-entropy. An example is the approach proposed by Lee [64], who first train network f_θ with losuper and then assign pseudo-labels according to (4.1) for $i \in \mathcal{U}$.

Label cleaning It leverages the idea of identifying true-labeled examples from noisy training data via multi-network or multi-round learning. Let $\mathcal{C}_t \subseteq \mathcal{X}_t$ be the selected *clean* examples at time t. Therefore, the network is updated only for the selected clean examples $\mathcal{C}_t$,

$$\theta_{t+1} = \theta_t - \eta_t \nabla \Big(\frac{1}{|\mathcal{C}_t|} \sum_{(x,\hat{y}) \in \mathcal{C}_t} \ell\big(\hat{y}, f_{\theta_t}(x)\big) \Big). \tag{4.4}$$

This training scheme is well motivated and works well in general, but it still suffers from accumulated error caused by incorrect selection. Hence, recent approaches often take advantage multiple networks to cooperate with one another or run multiple training rounds. In order to avoid the need of maintaining additional networks, multi-round learning iteratively refines the selected set of clean examples by repeating the training round. Thus, the selected set keeps improved as the number of rounds increases. In particular, O2U-Net [46] repeats the whole training process with the cyclical learning rate until enough loss statistics of every examples are gathered. Next, the network is re-trained from scratch only for the clean data where false-labeled examples have been detected and removed based on statistics.

4.3 CleanMatch

An overview of CleanMatch is shown in Figure 4.1. CleanMatch consists of two stages:

1. Selection of the most confident unlabeled examples via a cleaning process based on O2U-Net applied iteratively on the pseudolabels provided by FixMatch
2. Expansion of the labeled set $\mathcal{X}_\mathcal{L}$ with the extracted clean examples, and semi-supervised training based on FixMatch on the expanded labeled set.

4.3.1 Stage 1

Semi-supervised training based on augmentation consistency For labeled examples, the provided targets are used. So, the supervised loss can be written as follows:

$$\mathcal{L}_s = \frac{1}{|\mathcal{X}_\mathcal{L}|} \sum_{x,y \in \mathcal{X}_\mathcal{L}} \mathrm{H}(y, f_\theta(\mathrm{A_w}(x))), \tag{4.5}$$

For unlabeled examples $x \in \mathcal{X}_U$, a weakly augmented version is first computed using weak augmentation function A_w. Weak augmentations consist of a standard flip-and-shift augmentation strategy. Specifically, the images are flipped horizontally with a probability of 50% on all datasets except SVHN, in addition to randomly translating images by up to 12.5% vertically and horizontally.

Given an instance, only when the model predicts a high-confidence label can the predicted pseudo-label be identified as ground-truth. In particular, the predicted label is considered as a pseudo-label if the highest class probability is greater than a threshold τ_1. So, in order to obtain an artificial label, we first compute the model's predicted class distribution given a weakly-augmented version of a given unlabeled image: $q = f_\theta(\mathrm{A_w}(x))$. Then, with a pseudo-label $\hat{y} = \arg\max q$, K strongly augmented examples are generated using a strong augmentation function A_s. For the strong augmentations, we use only RandAugment (the authors also use CTAugment) [10] where a given transformation (color inversion, translation, contrast adjustment, etc.) is randomly selected for each sample in a batch of training examples, and the amplitude of the transformation is a hyperparameter that is optimized during training. We then assign to these augmented versions the proxy label obtained with the weakly labeled version. The unsupervised loss can be written as follows:

$$\mathcal{L}_u = w \frac{1}{K|\hat{\mathcal{X}}_\mathcal{U}|} \sum_{x \in \hat{\mathcal{X}}_\mathcal{U}} \sum_{i=1}^{K} \mathrm{H}(\hat{y}, f_\theta(A_s(x))), \tag{4.6}$$

where

$$\hat{\mathcal{X}}_\mathcal{U} = \{x \in \mathcal{X}_\mathcal{U} : \max_j(q)_j \geq \tau_1)\} \tag{4.7}$$

We continue by setting a second threshold τ_2, where $\tau_2 > \tau_1$, to select an even more confident subset C_U containing examples with their corresponding pseudolabels,

$$\mathcal{C}_\mathcal{U} = \{(x \in \mathcal{X}_\mathcal{U}, \hat{y} \in \mathcal{Y}_\mathcal{U}) : \max_j(q)_j \geq \tau_2)\} \tag{4.8}$$

to pass in the cleaning process. For an unlabeled example with a pseudo-label to be selected for the subset, its' highest class probability has to be greater than threshold τ_2. We wait for the model to train till it has 50% of augmented

images' predictions over the threshold τ_2, before we start the iterative cleaning process in order to extract the most confident unlabeled examples.

Iterative cleaning

To overcome this issue, we follow O2U-Net which introduces multiple rounds of status transfer in training. We adjust the hyper-parameters of a classifier to make its status transferring from overfitting to underfitting cyclically. A straightforward way is to apply the cyclical learning rate. At the beginning of training, a large learning rate is set. The learning rate linearly decreases to some extent during training and is then reset to the original learning rate. This whole process repeats for multiple rounds until enough loss statistics are gathered. The idea behind is that, when the network almost converges to some minimum (nearly overfitting), a large learning rate makes the network jump out of the minimum. As a result, the network would abruptly become underfitting.

We repeat this process and track the loss of every sample. In learning with noisy labels, it is common to detect noisy labels based on statistics of this loss value for clean and noisy labels [3, 56]. However, this does not work well with predicted pseudo-labels [1], hence we select the unlabeled examples from C_U having the least average loss [20, 1]. In particular, given the labeled set $\mathcal{X}_\mathcal{L}$ and the confident pseudo-labeled set $\mathcal{C}_\mathcal{U}$, which contains the unlabeled examples that have a highest class probability greater than threshold τ_2, we train an N-way classifier g using the following losses for labeled and pseudo-labeled examples:

$$\ell_L = \frac{1}{|\mathcal{X}_\mathcal{L}|} \sum_{x_i, y_i \in \mathcal{X}_\mathcal{L}} \mathrm{H}(y_i, g(x_i))), \tag{4.9}$$

$$\ell_C = \frac{1}{|\mathcal{C}_\mathcal{U}|} \sum_{x_i, \hat{y}_i \in \mathcal{C}_\mathcal{U}} \mathrm{H}(\hat{y}_i, g(x_i))) \tag{4.10}$$

The loss term $\ell_{C_i} = \mathrm{H}(\hat{y}_i, g(x_i))$, corresponding to the pseudo-labeled example $x_i \in \mathcal{C}_\mathcal{U}$, is used for selection. Following O2U-Net, we use a *cyclical schedule* of learning rate and collect the average loss $\bar{\ell}_{C_i}$ over all epochs.

A smaller batch size than training is chosen to make the network more easily transfer from overfitting to underfitting. The network is then trained for multiple rounds based on the cyclical learning rate. The loss of every sample is recorded during the cyclical training. For a training epoch, we subtract the average loss of all the samples in this epoch from the loss of every sample to normalize the losses in different epochs. In the cyclical train, suppose the maximum cyclical learning rate is η_{max}, and the minimum learning rate is η_{min}, where $\eta_{max} > \eta_{min}$. The equation for learning rate adjustment during the cyclical training is as follows:

$$s_t = (1 + ((t-1))c \tag{4.11}$$

$$\eta_t = (1 - s_t)\eta_{max} + s_t\eta_{min} \tag{4.12}$$

where t refers to the t_{th} epoch in the cyclical training, c is the total number of epochs in each cyclical round and η_t is the learning rate applied at t. After the whole cyclical training, the average of the normalized losses of every sample is computed. All the average losses are then ranked in descending order. Since we shall iterate the process and to be more confident for our selection, we take the extreme approach of selecting *one unlabeled example*:

$$\mathcal{C} = \left\{ (x_i, \hat{y}_i) \in C_U : \arg\min_j (\bar{\ell}_{C_i})_j \right\}. \tag{4.13}$$

So, $\mathcal{C}$ will contain only one example x_c, that with the least average loss, along with its pseudo-label $\hat{y}_c$. Finally, we augment, the initially empty, clean set $\mathcal{X}_\mathcal{C}$ with the selected example and its' pseudo-label, while at the same time removing the selected example from $\mathcal{X}_\mathcal{U}$:

$$\mathcal{X}_\mathcal{C} \leftarrow \mathcal{X}_\mathcal{C} \cup \{(x_c, \hat{y}_c)\} \tag{4.14}$$
$$\mathcal{X}_\mathcal{U} \leftarrow \mathcal{X}_\mathcal{U} \setminus \{x_c\} \tag{4.15}$$

Discussion The predicted pseudo-labels are not necessarily correct, yet a classifier can be robust to such noise. This is the case when enough data is available to adapt the representation, such that the quality of pseudo-labels improves with training. Since data is limited here, we would like to select pseudo-labeled examples in $\mathcal{X}_\mathcal{U}$ that are most likely to be correct, treat them as truly labeled and add them to the labeled set $\mathcal{X}_\mathcal{L}$ to be used on the next stage. Iterating this process is an alternative way of improving the quality of pseudo-labels.

We interpret this problem as *learning with noisy labels*, leveraging recent advances in label cleaning [1, 46, 97]. Assuming that the classifier does not overfit the data, with small capacity, high learning rate or few iterations, the principle is that examples with clean labels exhibit less loss than examples with noisy labels.

Particularly, by observing the whole training procedure on a given dataset including label noise, it is found that noisy labels are usually memorized at the late stage of training as the "hard" samples. At the beginning of the training, the losses of noisy labels are larger than those of clean samples because clean samples quickly get fit at that beginning. At the late stage of training, the losses generated from noisy labels and clean labels are indistinguishable because both of them are memorized by the network. Therefore, by tracking the variation of loss of every sample at the different stages of training, it is possible to detect noisy labels to some extent. However, in an ordinary training process, the status of the network would change from underfitting to overfitting only once. Once

the noisy labels are memorized, their losses would fast decrease. Moreover, when the noisy labels are overfitted is unknown. As a result, the loss tracking for every sample may not be reliable because of the lack of sufficient statistics. That is the reason we follow a multi-round learning approach as explained above.

4.3.2 Stage 2

Stage 2 expands the labeled set with the selected examples, and uses it for the semi-supervised training. First, we augment $\mathcal{X}_\mathcal{L}$ with $\mathcal{X}_\mathcal{C}$. However, because we chose to be selecting only one unlabeled example, $\mathcal{X}_\mathcal{C}$ will be unbalanced. In order to avoid transferring that imbalance to $\mathcal{X}_\mathcal{L}$ we compute the number of new clean examples per class in $\mathcal{X}_\mathcal{C}$. We are interested in the least amount (min) of new examples among the classes. Finally, we randomly only keep min examples per class to preserve balance. Now that $\mathcal{X}_\mathcal{C}$ is a balanced set we proceed with the augmentation of $\mathcal{X}_\mathcal{L}$:

$$\mathcal{X}_\mathcal{L} \leftarrow \mathcal{X}_\mathcal{L} \cup \ \mathcal{X}_\mathcal{C} \tag{4.16}$$

Then we train FixMatch with the augmented $\mathcal{X}_\mathcal{L}$. The benefits from this augmentation are obvious. Assuming that $\mathcal{X}_\mathcal{C}$ contains only clean examples, meaning that their corresponding predicted labels are true, FixMatch setup of k labeled examples will be converted to k+l labeled examples, with l being the size of $\mathcal{X}_\mathcal{C}$. Therefore, it will be trained with a bigger ratio of labeled to unlabeled examples, yielding better final results.

4.4 WeightMatch

With WeightMatch we aim to derive an importance weight for each training labeled example to adapt its effect on model training.In particular,via a variant of the iterative cleaning process we force the minibatches FixMatch is trained on to contain the most confident labelled samples. In particular, In particular, given the labeled set $\mathcal{X}_\mathcal{L} = \{(x_i, y_i)\}_{i=1}^{L}$ we train an N-way classifier g using a *cross-entropy* loss

$$\ell = \frac{1}{|\mathcal{X}_\mathcal{L}|} \sum_{x,y \in \mathcal{X}_\mathcal{L}} \mathrm{H}(y, g(x))), \tag{4.17}$$

The classifier follows the previous cyclical training while the selection of the most confident labeled examples is based on the importance weight ranking, which is inversely proportional to the computed average loss ranking. For this method, we take the approach of selecting *one example per class* to preserve class balance.

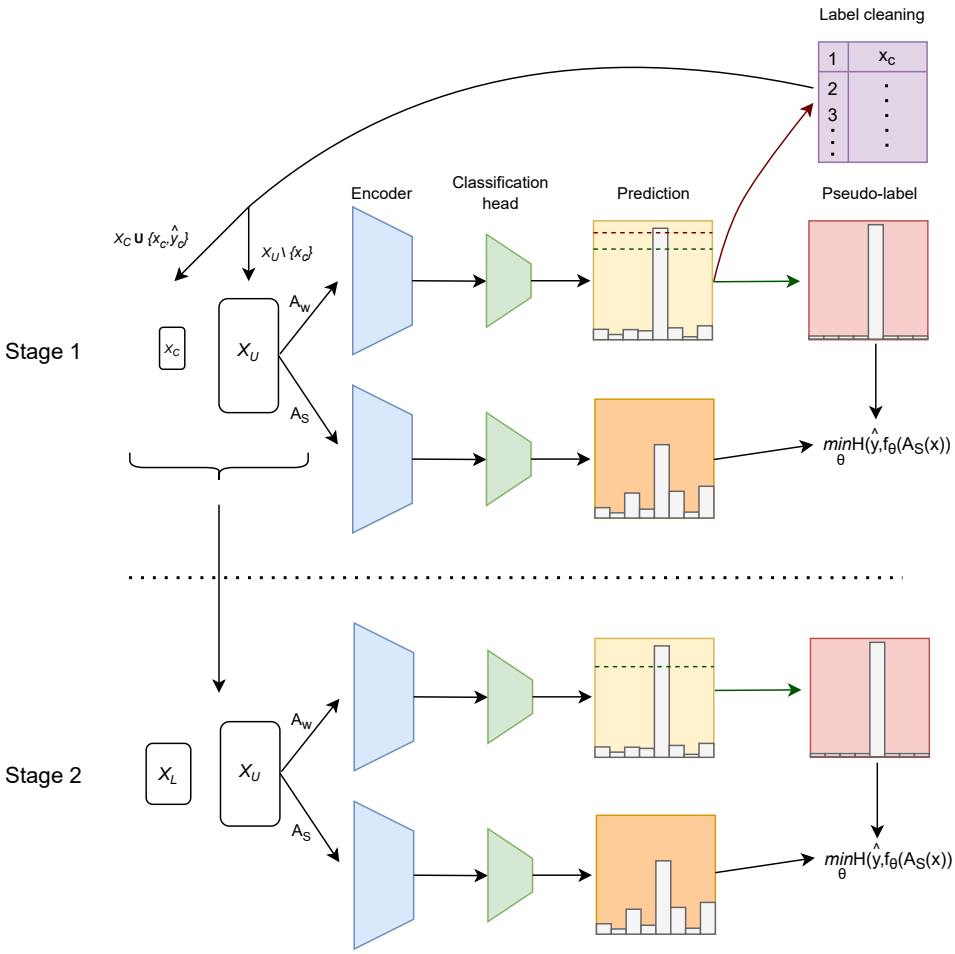


Figure 4.1: **CleanMatch.** *Stage 1*: An example goes through weak(A_W) and strong augmentation(A_S). The weakly-augmented version of the example is fed into the encoder to obtain predictions produced by a classifier head. If its maximum output class probability assigned by the network is above a threshold τ_1 (green dotted line), then the example along with its pseudo-label $\hat{y}$ contributes to the training (the model is trained to make its prediction on the strongly-augmented version match the pseudo-label via a cross-entropy loss). The model prediction on a weakly augmented input is considered as candidate for cleaning If the maximum output class probability is above threshold τ_2 (red dotted line), then it is a candidate for selection through label cleaning. If it is selected as clean(x_c) then it will be added to a clean set $\mathcal{X}_\mathcal{C}$, and subtracted from the unlabeled set $\mathcal{X}_\mathcal{U}$. *Stage 2*: We augment $\mathcal{X}_\mathcal{L}$ with $\mathcal{X}_\mathcal{C}$ and train as in Stage 1 without the iterative label cleaning using the new $\mathcal{X}_\mathcal{L}$.

5 Experiments

We evaluate the efficacy of CleanMatch and WeightMatch on several SSL image classification benchmarks. Specifically, we perform experiments with varying amounts of labeled data on CIFAR10, CIFAR100 [58] and SVHN [74] following standard SSL evaluation protocols [76]. In many cases, we perform experiments with fewer labels than usually considered since our methods are the most promising in extremely label-scarce settings.

5.1 CIFAR-10 and SVHN

First, we conduct experiments on CIFAR-10, CIFAR-100 and SVHN datasets. CIFAR-10 consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. SVHN is a digit classification benchmark dataset that contains 600000 32×32 RGB images of printed digits (from 0 to 9) cropped from pictures of house number plates. We vary the amount of labeled data and focus on the label-scarce scenario where few labels are available. We evaluate on 5 runs with different random seeds.

Baseline methods For CIFAR-10 and SVHN we compare our methods with the original FixMatch, ReMixMatch, SelfMatch and CoMatch. We omit previous methods such as MixMatch, -model, PseudoLabeling, and Mean Teacher due to their poorer performance as reported in [96]. Following [76], we performed all experiments using the same model architecture and the same codebase. However, we did not reimplement all the baselines due to computational power limitations, but they are reported as presented in the competitors' works.

Implementation details Following [76], in order to train with FixMatch for stage 1 and 2 of CleanMatch but also WeightMatch we used a Wide ResNet-28-2 [108] with 1.5M parameters for CIFAR-10 and SVHN. The model is trained using SGD with a momentum of 0.9 and a weight decay of 0.0005. We follow the original papers and train CleanMatch and WeightMatch for 1024 epochs, using a learning rate of 0.03 with cosine learning rate decay with a decay of $\eta \cos(\frac{7\pi t}{16T})$, where η is the initial learning rate, t is the current training step, and T is the total number of training iterations. For the hyperparameters in

our methods that also exist in [34], we follow [34] and set = 1, $\tau_1 = 0.95$, $\mu = 7$, B = 64 while the threshold τ_2 is set at 0.999

For the cleaning process based on O2U-net we test a 2 linear layer network and a 9-layer network. The batch size is set to 16. The cyclical learning rate is linearly adjusted from 0.01 to 0 in a cycle round, but the variation that starts from 0.1 seems to work better. In a cycle round, we adopt three different cycle lengths, 10, 30 and 50. The number of epochs is set from 10 to 300. We apply the SGD optimizer with the momentum factor set to 0.9 and weight decay to 0.0005. Further explanation on the label cleaning's parameters is provided in Section 4.3.

Results Tables 5.1 and 5.2 show the results on CIFAR-10 and SVHN. CleanMatch and WeightMatch outperform the best baseline across all settings. The improvement is more substantial when fewer labeled samples are available. For example, CleanMatch achieves an average accuracy of 93.41% on CIFAR-10 with only 1 label per class, whereas FixMatch (w. DA) has a lower accuracy of 73.64%.It has to be noted that WeightMatch does not offer any improvements for the setting of 1 label per class since the weighting applies to 1 sample per class, meaning in this setting all the labeled samples. With 2 labels per class, CleanMatch achieves 93.98% average accuracy while WeightMatch 91.40%, whereas CoMatch reaches 87.67%. On SVHN, we can see similar improvements (e.g. CleanMatch achieves 95.98% with 2 labels per class while our implementation of FixMatch achieves 88.27%).

Dataset	**CIFAR-10**					
Nb. labeled images	**10**	**20**	**40**	**250**	**1000**	**4000**
Supervised	96.30					
ReMixMatch[10]	-	-	80.90$\pm$9.64	94.56$\pm$0.05	94.27$\pm$0.10	95.28$\pm$0.13
SelfMatch[55]	-	-	93.19$\pm$1.08	95.13$\pm$0.26	-	95.94$\pm$0.08
CoMatch[66]	-	87.67$\pm$8.47	93.09$\pm$1.39	95.09$\pm$0.33	-	-
FixMatch(Pytorch)[54]	73.64$\pm$7.51	82.11$\pm$9.05	93.38$\pm$2.15	95.31$\pm$1.08	95.39$\pm$0.64	95.77$\pm$0.13
FixMatch(official)[96]	-	82.32$\pm$9.77	88.61$\pm$3.35	94.93$\pm$0.33	-	95.74$\pm$0.15
WeightMatch(ours)	-	**91.40**$\pm$4.12	**93.88**$\pm$3.91	**95.45**$\pm$1.07	**95.47**$\pm$0.82	**95.82**$\pm$0.18
CleanMatch(ours)	**93.41**$\pm$1.54	**93.98**$\pm$0.79	**95.17**$\pm$0.55	**95.62**$\pm$0.42	-	**95.85**$\pm$0.12

Table 5.1: Comparison of accuracy for CIFAR-10. All baseline methods use Wide ResNet 28-2.

5.2 CIFAR-100

We want to evaluate CleanMatch and WeightMatch on CIFAR100 to verify their efficacy on a more challenging dataset. CIFAR-100 is just like the CIFAR-10,

Dataset	SVHN			
Nb. labeled images	**20**	**40**	**250**	**1000**
Supervised	97.98			
ReMixMatch[10]	-	96.66±2.17	97.08±0.38	97.35±0.11
SelfMatch[55]	-	96.58±1.02	97.37±0.43	97.49±0.07
FixMatch(Pytorch)[54]	88.27±5.74	96.09±2.43	97.49±0.29	97.78±0.09
FixMatch(official)[96]	-	96.04±2.17	97.52±0.38	97.72±0.11
WeightMatch(ours)	**92.49**±2.59	**96.14**±2.10	**97.52**±0.26	**97.78**±0.07
CleanMatch(ours)	**95.98**±2.38	**97.36**±2.03	**97.60**±0.68	**97.81**±0.15

Table 5.2: Comparison of accuracy for SVHN. All baseline methods use Wide ResNet 28-2.

except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. We again vary the amount of labeled data and we evaluate on 5 runs with different random seeds.

Baseline methods For CIFAR-100 we compare our methods with ReMixMatch and FixMatch since they remain the state-of-the-art. Again, we followed [76] and performed all experiments using the same model architecture and the same codebase, but did not reimplement all the baseline experiments ourselves.

Implementation details For the training of our two methods we again followed [76] and used Wide ResNet-28-8 [108] for CIFAR-100. The model is again trained using SGD with a momentum of 0.9 but now a weight decay of 0.001 is selected. All the other hyperparameters remain the same with the exception of the batch size B which is set at 16. For the label cleaning the same implementation as before is followed, just with a batch size of 8.

Results In CIFAR-100 CleanMatch does not work. This is the result of failure in the iterative label cleaning process, since a small amount of the selected confident unlabeled samples that will augment the labeled set turn out to have false predicted pseudo-labels. However, WeightMatch works and as it can be seen in Table 5.3 that it achieves an average accuracy of 63.21% with 4 labels per class, whereas our reimplementation of FixMatch (w. DA) has a lower accuracy of 57.50%.

Dataset	CIFAR-100		
Nb. labeled images	**400**	**2500**	**10000**
Fully supervised		80.96	
ReMixMatch[10]	$55.72_{\pm 2.06}$	$72.57_{\pm 0.31}$	$76.97_{\pm 0.56}$
FixMatch(Pytorch)[54]	$57.50_{\pm 2.76}$	$72.93_{\pm 0.50}$	$78.12_{\pm 0.23}$
FixMatch(official)[96]	$51.15_{\pm 1.75}$	$71.71_{\pm 0.24}$	$77.40_{\pm 0.11}$
WeightMatch(ours)	$\mathbf{63.21}_{\pm 1.01}$	$\mathbf{75.80}_{\pm 0.17}$	$\mathbf{79.57}_{\pm 0.09}$

Table 5.3: Comparison of accuracy for CIFAR-100. All baseline methods use Wide ResNet 28-8.

5.3 Ablation study

We perform extensive ablation study to examine the effect of different components in our two methods. We use CIFAR-10 with 4 labels per class as the main experiment.

5.3.1 Iterative cleaning

Architectures We tested two classifier variants on which our label cleaning is based, a 2 linear layer network and a 9-layer CNN used in [40](we slightly modify its structure to fit it to different image sizes). The 9-layer CNN was proven to be the most efficient in terms of percentage of clean pseudo-labels selected. In particular, the 9-layer CNN was 100% accurate in its selection, while the 2 linear layer classifier was 94% accurate.

Hyperparameters We experimented with two different batch samplers, training with a mini-batch of size $B = B_L$ and a mini batch with mini-batch size $B = B_U + B_L$, where B_L images are labeled and B_U images are unlabeled/pseudo-labeled. The most effective when it comes to cleaning was the second sampler. We also experimented with the cyclical learning rate, setting η_{max} between 0.1 and 0.01. The accuracy of cleaning was best with $\eta_{max} = 0.1$. Finally for a cycle round, we adopted three different cycle lengths, 10, 30 and 50. The number of epochs is set from 10 to 300. The best combination was proven to be the one with cycle length of 30 and 300 epochs.

5.3.2 Training

Method We tested a version of CleanMatch where the semi-supervised training based on augmentation consistency regularization (FixMatch) and the iterative cleaning occurs in one stage. In particular, we let the model train for

a number of epochs and then start the cleaning process. By selecting only one sample with its pseudo-label and adding it to the labeled set, we observe that the training and testing accuracy stops improving. This is probably the result of the model focusing on certain classes, the majority classes. This is a problem because typically, the minority classes are more important since they are the most challenging and therefore the problem is more sensitive to classification errors for those classes than the majority ones.

Then, we tried selecting one label per class to preserve the balance of our dataset. However, this approach led to a few selected samples to be falsely pseudo-labeled, hindering once again the accuracy improvement of our model.

For the threshold τ_2 setting it at 0.999 works better in terms of retaining the most confident samples with correct pseudo-labels. When it was set lower, a few false pseudo-labels occured. Also, we experimented with the starting point of the cleaning process. Starting from the first epochs didn't work. What finally worked was letting the network train till 50% of all the predictions on the augmented images are equal or above the threshold τ_2. It happens around the 200th epoch.

6 Conclusion and Future work

6.1 Conclusion

Our solutions are conceptually simple and combine in a unique way ideas that have been successful in problems related to our task at hand. *Semi-supervised learning* based on a combination of augmentation consistency regularization and pseudo-labeling uses only standard cross-entropy losses on both labeled and unlabeled data, yielding remarkable results. *Label cleaning*, originally introduced for learning with noisy labels, is also very successful in cleaning predicted pseudo-labels. *Iterative selection* of just a few pseudo-labels as true labels bypasses the difficulty of single-shot detection of clean examples. *Weighting based on confidence* steers the semi-supervised training towards relying on the most discriminative examples per class.

Importantly, we empirically demonstrate that our methods outperform the state-of-the-art methods and can contribute to closing the gap between supervised learning and semi-supervised learning using only a few labels. In particular, our methods yield the most significant improvements in extremely label-scarce settings: below 1% of all the labeled examples for all the datasets. CleanMatch offers the greatest improvements on CIFAR-10 and SVHN but fails on CIFAR-100, while WeightMatch works on all datasets.

Finally, reasonable baselines, like predicting pseudo-labels by a classifier or iteratively re-using pseudo-labels without cleaning, fail completely.

6.2 Future work

This research only concentrates on inductive semi-supervised learning, where the goal is generalization to new unseen data, while the original training data are discarded. This is achieved e.g. by combining classification loss on labeled data with unsupervised objectives on all data, where the latter act as regularization. Or, as seen in this work, an existing classifier can be used to assign pseudo-labels, which is another form of algorithmic supervision.

However, there are efficient transductive learning algorithms such as label propagation [49] that infer pseudo-labels for unlabeled data, which are used to train the classifier. Label propagation is a graph-based method, and in [49] the graph is constructed exploiting the embeddings obtained by the classification network itself. This method has been proven to provide high-quality pseudo-

labels. Therefore, it would be useful to experiment and draw comparisons between our methods and that of label propagation. In particular, we would like to investigate the quality of pseudo-labels provided by label propagation and how they compare to the ones provided directly from a network (as used in this work). If they are proven to be of higher quality, label propagation can be adjusted to work complementary to our methods.

The aforementioned transductive approach could address the limitation of CleanMatch when tested on CIFAR-100. The potential higher quality pseudo-labels may enable the label cleaning process to work as intended. One other alternative that is worth investigating is applying the label cleaning more sparsely.

Regardless of the improvements on our existing methods we would like to experiment with other related tasks. In general, SSL methods have been mostly applied to image classification, whose labeling cost is relatively cheaper compared to other important problems in computer vision, such as object detection. Due to its expensive labeling cost, object detection demands a higher level of label efficiency, necessitating the development of strong SSL methods. That is why we want to apply our iterative label cleaning idea to improve existing SSL methods for object detection, like STAC [95]. STAC is an extension of FixMatch and it deploys highly confident pseudo labels of localized objects from an unlabeled image and updates the model by enforcing consistency via strong augmentations.

One other interesting direction, since SSL is such a realistic scenario, would be to combine it with other ones that are based on full supervision. In particular, it can be coupled with incremental learning, metric learning, long tail recognition and learning from noisy labels.

Finally, in representation learning, and particularly in self-supervised learning, it is a common practice to include a small percentage of a large-scale dataset(e.g. Imagenet) during training. Due to the fact that the datasets used for this purpose are curated, our label cleaning idea can turn out to be useful.

Bibliography

[1] Eric Arazo et al. "Unsupervised Label Noise Modeling and Loss Correction." In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 312–321. URL: http://proceedings.mlr.press/v97/arazo19a.html (cit. on pp. 7, 53).

[2] Ehsan Mohammady Ardehaly and Aron Culotta. "Co-training for demographic classification using deep learning from label proportions." In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2017, pp. 1017–1024 (cit. on p. 38).

[3] Ben Athiwaratkun et al. *There Are Many Consistent Explanations of Unlabeled Data: Why You Should Average*. 2019. arXiv: 1806.05594 [cs.LG] (cit. on p. 32).

[4] Sandra Avila et al. "Pooling in Image Representation: The Visual Codeword Point of View." In: *Comput. Vis. Image Underst.* 117.5 (May 2013), pp. 453–465. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2012.09.007. URL: https://doi.org/10.1016/j.cviu.2012.09.007 (cit. on p. 13).

[5] Philip Bachman, Ouais Alsharif, and Doina Precup. "Learning with Pseudo-Ensembles." In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/66be31e4c40d676991f2405aaecc6934-Paper.pdf (cit. on p. 7).

[6] Herbert Bay et al. "Speeded-Up Robust Features (SURF)." In: *Computer Vision and Image Understanding* 110.3 (2008). Similarity Matching in Computer Vision and Multimedia, pp. 346–359. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2007.09.014. URL: http://www.sciencedirect.com/science/article/pii/S1077314207001555 (cit. on p. 11).

[7] Mikhail Belkin and Partha Niyogi. "Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering." In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 585–591 (cit. on p. 28).

[8] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult." In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181 (cit. on p. 18).

[9] David Berthelot et al. "Mixmatch: A holistic approach to semi-supervised learning." In: *Advances in Neural Information Processing Systems*. 2019, pp. 5050–5060 (cit. on pp. 40, 42).

[10] David Berthelot et al. "ReMixMatch: Semi-Supervised Learning with Distribution Alignment and Augmentation Anchoring." In: *arXiv preprint arXiv:1911.09785* (2019) (cit. on pp. 8, 41, 43, 44, 51, 57–59).

[11] Christian Biemann. "Unsupervised and knowledge-free natural language processing in the structure discovery paradigm." Doctoral dissertation. Leipzig University, Germany, 2007 (cit. on p. 27).

[12] Avrim Blum and Tom Mitchell. "Combining labeled and unlabeled data with co-training." In: *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*. 1998, pp. 92–100 (cit. on p. 38).

[13] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers." In: *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*. ACM Press, 1992, pp. 144–152 (cit. on p. 13).

[14] Y-Lan Boureau et al. "Learning mid-level features for recognition." In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2559–2566. DOI: 10.1109/CVPR.2010.5539963 (cit. on p. 13).

[15] Leo Breiman. "Randomizing outputs to increase prediction accuracy." In: *Machine Learning* 40.3 (2000), pp. 229–242 (cit. on p. 39).

[16] Leo Breiman et al. *Classification and regression trees*. CRC press, 1984 (cit. on p. 13).

[17] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]." In: *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 542–542 (cit. on pp. 26, 27).

[18] Dong-Dong Chen et al. "Tri-Net for Semi-Supervised Deep Learning." In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI'18. Stockholm, Sweden: AAAI Press, 2018, pp. 2014–2020. ISBN: 9780999241127 (cit. on pp. 38, 40).

[19] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations." In: *CoRR* abs/2002.05709 (2020). arXiv: 2002.05709. URL: https://arxiv.org/abs/2002.05709 (cit. on p. 44).

[20] Yanhua Cheng et al. “Semi-supervised multimodal deep learning for RGB-D object recognition.” In: (2016) (cit. on p. 38).

[21] Kevin Clark et al. “Semi-supervised sequence modeling with cross-view training.” In: *arXiv preprint arXiv:1809.08370* (2018) (cit. on p. 39).

[22] Ekin D Cubuk et al. “Randaugment: Practical automated data augmentation with a reduced search space.” In: *CVPR Workshops*. 2020, pp. 702–703 (cit. on pp. 8, 34, 36, 43).

[23] Ekin Dogus Cubuk et al. “AutoAugment: Learning Augmentation Policies from Data.” In: *CoRR* abs/1805.09501 (2018). arXiv: 1805.09501. URL: http://arxiv.org/abs/1805.09501 (cit. on pp. 34, 43).

[24] N. Dalal and B. Triggs. “Histograms of Oriented Gradients for Human Detection.” In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* 1 (2005), pp. 886–893. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1467360 (cit. on p. 11).

[25] Terrance Devries and Graham W. Taylor. “Improved Regularization of Convolutional Neural Networks with Cutout.” In: *CoRR* abs/1708.04552 (2017). arXiv: 1708.04552. URL: http://arxiv.org/abs/1708.04552 (cit. on pp. 8, 43).

[26] N. Dey et al. “A Comparative Study between Moravec and Harris Corner Detection of Noisy Images Using Adaptive Wavelet Thresholding Technique.” In: *ArXiv* abs/1209.1558 (2012) (cit. on p. 9).

[27] WeiWang Dong-DongChen and Zhi-HuaZhou WeiGao. “Tri-net for semi-supervised deep learning.” In: *Proceedings of Twenty-Seventh International Joint Conference on Artificial Intelligence*. 2018, pp. 2014–2020 (cit. on p. 39).

[28] Alexey Dosovitskiy et al. “Discriminative Unsupervised Feature Learning with Convolutional Neural Networks.” In: *CoRR* abs/1406.6909 (2014). arXiv: 1406.6909. URL: http://arxiv.org/abs/1406.6909 (cit. on pp. 36, 37).

[29] Alexey Dosovitskiy et al. “Discriminative Unsupervised Feature Learning with Exemplar Convolutional Neural Networks.” In: *IEEE Trans. Pattern Anal. Mach. Intell.* 38.9 (2016), pp. 1734–1747. URL: http://dblp.uni-trier.de/db/journals/pami/pami38.html#DosovitskiyFSRB16 (cit. on p. 36).

[30] Sergey Edunov et al. “Understanding Back-Translation at Scale.” In: *CoRR* abs/1808.09381 (2018). arXiv: 1808.09381. URL: http://arxiv.org/abs/1808.09381 (cit. on p. 34).

[31] J. Fournier, M. Cord, and S. Philipp-Foliguet. "RETIN: A Content-Based Image Indexing and Retrieval System." In: *Pattern Analysis & Applications* 4 (2001), pp. 153–173 (cit. on p. 12).

[32] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." In: *Biological Cybernetics* 36.4 (Apr. 1980), pp. 193–202. DOI: 10.1007/bf00344251. URL: https://doi.org/10.10072Fbf00344251 (cit. on p. 15).

[33] J. V. Gemert et al. "Visual Word Ambiguity." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32 (2010), pp. 1271–1283 (cit. on p. 12).

[34] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. "Unsupervised Representation Learning by Predicting Image Rotations." In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=S1v4N2l0- (cit. on p. 36).

[35] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html (cit. on p. 18).

[36] J. Goldberger and E. Ben-Reuven. "Training deep neural-networks using a noise adaptation layer." In: *ICLR*. 2017 (cit. on p. 47).

[37] Yves Grandvalet and Yoshua Bengio. "Semi-supervised Learning by Entropy Minimization." In: *NIPS*. 2004, pp. 529–536 (cit. on pp. 35, 37).

[38] Sheng Guo et al. *CurriculumNet: Weakly Supervised Learning from Large-Scale Web Images*. 2018. arXiv: 1808.01097 [cs.CV] (cit. on p. 47).

[39] Guy Hacohen and Daphna Weinshall. *On The Power of Curriculum Learning in Training Deep Networks*. 2019. arXiv: 1904.03626 [cs.LG] (cit. on p. 47).

[40] Bo Han et al. "Co-sampling: Training Robust Networks for Extremely Noisy Supervision." In: *CoRR* abs/1804.06872 (2018). arXiv: 1804.06872. URL: http://arxiv.org/abs/1804.06872 (cit. on pp. 47, 59).

[41] C. G. Harris and M. Stephens. "A Combined Corner and Edge Detector." In: *Alvey Vision Conference*. 1988 (cit. on p. 9).

[42] Hangfeng He and Xu Sun. "A Unified Model for Cross-Domain and Semi-Supervised Named Entity Recognition in Chinese Social Media." In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, 2017, pp. 3216–3222 (cit. on p. 35).

[43] Kaiming He et al. "Deep Residual Learning for Image Recognition." In: *CVPR*. 2016, pp. 770–778 (cit. on pp. 18–20).

[44] Alexander Hermans, Lucas Beyer, and Bastian Leibe. "In Defense of the Triplet Loss for Person Re-Identification." In: *CoRR* abs/1703.07737 (2017). arXiv: 1703.07737. URL: http://arxiv.org/abs/1703.07737 (cit. on p. 37).

[45] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. "Distilling the Knowledge in a Neural Network." In: *CoRR* abs/1503.02531 (2015). arXiv: 1503.02531. URL: http://arxiv.org/abs/1503.02531 (cit. on p. 36).

[46] Jinchi Huang et al. "O2U-Net: A Simple Noisy Label Detection Approach for Deep Neural Networks." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019 (cit. on pp. 7, 8, 50, 53).

[47] Alexander G. Ororbia II, C. Lee Giles, and David Reitter. "Learning a Deep Hybrid Model for Semi-Supervised Text Classification." In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. Ed. by Llus Màrquez et al. The Association for Computational Linguistics, 2015, pp. 471–481. DOI: 10.18653/v1/d15-1053. URL: https://doi.org/10.18653/v1/d15-1053 (cit. on p. 35).

[48] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html (cit. on p. 19).

[49] Ahmet Iscen et al. "Label Propagation for Deep Semi-supervised Learning." In: *CoRR* abs/1904.04717 (2019). arXiv: 1904.04717. URL: http://arxiv.org/abs/1904.04717 (cit. on p. 61).

[50] Pavel Izmailov et al. *Averaging Weights Leads to Wider Optima and Better Generalization*. 2019. arXiv: 1803.05407 [cs.LG] (cit. on p. 32).

[51] Hervé Jégou et al. "Aggregating local descriptors into a compact image representation." In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 3304–3311. DOI: 10.1109/CVPR.2010.5540039 (cit. on p. 12).

[52] Lu Jiang et al. "MentorNet: Regularizing Very Deep Neural Networks on Corrupted Labels." In: *CoRR* abs/1712.05055 (2017). arXiv: 1712.05055. URL: http://arxiv.org/abs/1712.05055 (cit. on p. 47).

[53] Zhanghan Ke et al. "Dual Student: Breaking the Limits of the Teacher in Semi-supervised Learning." In: *CoRR* abs/1909.01804 (2019). arXiv: 1909.01804. URL: http://arxiv.org/abs/1909.01804 (cit. on pp. 29, 31).

[54] kekmodel. URL: https://github.com/kekmodel/FixMatch-pytorch (cit. on pp. 57–59).

[55] Byoungjip Kim et al. "SelfMatch: Combining Contrastive Self-Supervision and Consistency for Semi-Supervised Learning." In: *CoRR* abs/2101.06480 (2021). arXiv: 2101.06480. URL: https://arxiv.org/abs/2101.06480 (cit. on pp. 44, 45, 57, 58).

[56] Pang Wei Koh and Percy Liang. *Understanding Black-box Predictions via Influence Functions*. 2020. arXiv: 1703.04730 [stat.ML] (cit. on p. 46).

[57] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. "Revisiting Self-Supervised Visual Representation Learning." In: *CoRR* abs/1901.09005 (2019). arXiv: 1901.09005. URL: http://arxiv.org/abs/1901.09005 (cit. on pp. 36, 37).

[58] Alex Krizhevsky and Geoffrey Hinton. *Learning Multiple Layers of Features from Tiny Images*. Mater's thesis. University of Toronto, 2009 (cit. on p. 56).

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386 (cit. on p. 17).

[60] Samuli Laine and Timo Aila. "Temporal Ensembling for Semi-Supervised Learning." In: *ICLR (Poster)*. OpenReview.net, 2017. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2017.html#LaineA17 (cit. on p. 30).

[61] S. Lazebnik, C. Schmid, and J. Ponce. "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories." In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. 2006, pp. 2169–2178. DOI: 10.1109/CVPR.2006.68 (cit. on p. 13).

[62] Y. Lecun et al. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791 (cit. on p. 17).

[63] Chen-Yu Lee et al. *Deeply-Supervised Nets*. 2014. arXiv: 1409.5185 [stat.ML] (cit. on p. 18).

[64] Dong-Hyun Lee. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks." In: *Workshop on challenges in representation learning, ICML*. Vol. 3. 2013, p. 2 (cit. on pp. 7, 8, 35, 37, 50).

[65] Kuang-Huei Lee et al. "CleanNet: Transfer Learning for Scalable Image Classifier Training with Label Noise." In: *CoRR* abs/1711.07131 (2017). arXiv: 1711.07131. URL: http://arxiv.org/abs/1711.07131 (cit. on p. 47).

[66] Junnan Li, Caiming Xiong, and Steven C. H. Hoi. "CoMatch: Semi-supervised Learning with Contrastive Graph Regularization." In: *CoRR* abs/2011.11183 (2020). arXiv: 2011.11183. URL: https://arxiv.org/abs/2011.11183 (cit. on pp. 44, 46, 57).

[67] S. P. Lloyd. "Least squares quantization in PCM." In: *IEEE Trans. Inf. Theory* 28 (1982), pp. 129–136 (cit. on p. 12).

[68] G. LoweDavid. "Distinctive Image Features from Scale-Invariant Keypoints." In: *International Journal of Computer Vision* (2004) (cit. on pp. 9, 12).

[69] Wei-Ying Ma and B. S. Manjunath. "NeTra: A Toolbox for Navigating Large Image Databases." In: *Multimedia Syst.* 7.3 (May 1999), pp. 184–198. ISSN: 0942-4962. DOI: 10.1007/s005300050121. URL: https://doi.org/10.1007/s005300050121 (cit. on p. 12).

[70] Eran Malach and Shai Shalev-Shwartz. "Decoupling "when to update" from "how to update"." In: *CoRR* abs/1706.02613 (2017). arXiv: 1706.02613. URL: http://arxiv.org/abs/1706.02613 (cit. on p. 47).

[71] Krystian Mikolajczyk and Cordelia Schmid. "Scale Affine Invariant Interest Point Detectors." In: *Int. J. Comput. Vision* 60.1 (Oct. 2004), pp. 63–86. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000027790.02288.f2. URL: https://doi.org/10.1023/B:VISI.0000027790.02288.f2 (cit. on pp. 9, 10).

[72] Ishan Misra, Abhinav Shrivastava, and Martial Hebert. "Watch and Learn: Semi-Supervised Learning of Object Detectors from Videos." In: *CoRR* abs/1505.05769 (2015). arXiv: 1505.05769. URL: http://arxiv.org/abs/1505.05769 (cit. on p. 35).

[73] Takeru Miyato et al. *Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning*. 2018. arXiv: 1704.03976 [stat.ML] (cit. on p. 31).

[74] Yuval Netzer et al. "Reading digits in natural images with unsupervised feature learning." In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. 2011 (cit. on p. 56).

[75] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. "Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 24.7 (July 2002), pp. 971–987. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2002.1017623. URL: https://doi.org/10.1109/TPAMI.2002.1017623 (cit. on p. 11).

[76] Avital Oliver et al. "Realistic Evaluation of Deep Semi-Supervised Learning Algorithms." In: *NeurIPS*. Ed. by Samy Bengio et al. 2018, pp. 3239–3250 (cit. on pp. 28, 56, 58).

[77] Sungrae Park et al. "Adversarial Dropout for Supervised and Semi-supervised Learning." In: *CoRR* abs/1707.03631 (2017). arXiv: 1707.03631. URL: http://arxiv.org/abs/1707.03631 (cit. on p. 32).

[78] Florent Perronnin and Christopher Dance. "Fisher Kernels on Visual Vocabularies for Image Categorization." In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383266 (cit. on p. 12).

[79] Hieu Pham et al. "Meta Pseudo Labels." In: *CoRR* abs/2003.10580 (2020). arXiv: 2003.10580. URL: https://arxiv.org/abs/2003.10580 (cit. on p. 37).

[80] N. Qian. "On the momentum term in gradient descent learning algorithms." In: *Neural networks : the official journal of the International Neural Network Society* 12 1 (1999), pp. 145–151 (cit. on p. 23).

[81] Siyuan Qiao et al. "Deep Co-Training for Semi-Supervised Image Recognition." In: *CoRR* abs/1803.05984 (2018). arXiv: 1803.05984. URL: http://arxiv.org/abs/1803.05984 (cit. on pp. 38, 40).

[82] Antti Rasmus et al. "Semi-supervised Learning with Ladder Networks." In: *NIPS*. Ed. by Corinna Cortes et al. 2015, pp. 3546–3554 (cit. on p. 30).

[83] Antti Rasmus et al. "Semi-supervised learning with Ladder networks." English. In: *Advances in Neural Information Processing Systems*. Vol. 2015-January. VK: Raiko, T.; COIN; IEEE Conference on Neural Information Processing Systems, NIPS ; Conference date: 07-12-2015 Through 12-12-2015. Neural Information Processing Systems Foundation, 2015, pp. 3546–3554 (cit. on p. 29).

[84] Scott E. Reed et al. "Training Deep Neural Networks on Noisy Labels with Bootstrapping." In: *CoRR* abs/1412.6596 (2015) (cit. on p. 47).

[85] Sebastian Ruder and Barbara Plank. "Strong baselines for neural semi-supervised learning under domain shift." In: *arXiv preprint arXiv:1804.09530* (2018) (cit. on pp. 27, 39).

[86] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV] (cit. on pp. 5, 17).

[87] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. "Regularization With Stochastic Transformations and Perturbations for Deep Semi-Supervised Learning." In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/30ef30b64204a3088a26bc2e6ecf7602-Paper.pdf (cit. on p. 7).

[88] Gerard Salton. "Another Look at Automatic Text-Retrieval Systems." In: *Commun. ACM* 29.7 (July 1986), pp. 648–656. ISSN: 0001-0782. DOI: 10.1145/6138.6149. URL: https://doi.org/10.1145/6138.6149 (cit. on p. 12).

[89] Frederik Schaffalitzky and Andrew Zisserman. "Viewpoint invariant texture matching and wide baseline stereo." In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001* 2 (2001), 636–643 vol.2 (cit. on p. 10).

[90] H Scudder. "Probability of error of some adaptive pattern-recognition machines." In: *IEEE Transactions on Information Theory* 11.3 (1965), pp. 363–371 (cit. on p. 7).

[91] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Improving Neural Machine Translation Models with Monolingual Data." In: *CoRR* abs/1511.06709 (2015). arXiv: 1511.06709. URL: http://arxiv.org/abs/1511.06709 (cit. on p. 34).

[92] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1409.1556 (cit. on pp. 17, 18).

[93] Sivic and Zisserman. "Video Google: a text retrieval approach to object matching in videos." In: *Proceedings Ninth IEEE International Conference on Computer Vision*. 2003, 1470–1477 vol.2. DOI: 10.1109/ICCV.2003.1238663 (cit. on p. 13).

[94] A. Smeulders et al. "Content-Based Image Retrieval at the End of the Early Years." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 22 (2000), pp. 1349–1380 (cit. on p. 1).

[95] Kihyuk Sohn et al. "A Simple Semi-Supervised Learning Framework for Object Detection." In: *CoRR* abs/2005.04757 (2020). arXiv: 2005.04757. URL: https://arxiv.org/abs/2005.04757 (cit. on p. 62).

[96] Kihyuk Sohn et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." In: *arXiv preprint arXiv:2001.07685* (2020) (cit. on pp. v, vii, 8, 43–45, 56–59).

[97] Jiaming Song et al. "Robust and On-the-fly Dataset Denoising for Image Classification." In: *CoRR* abs/2003.10647 (2020). arXiv: 2003.10647. URL: https://arxiv.org/abs/2003.10647 (cit. on pp. 7, 53).

[98] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435 (cit. on p. 17).

[99] K. N. Stevens et al. *Nearest Neighbor Pattern Classification*. 1953 (cit. on p. 13).

[100] Christian Szegedy et al. "Going deeper with convolutions." In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594. URL: https://doi.org/10.1109/CVPR.2015.7298594 (cit. on pp. 17–19, 37).

[101] Mingxing Tan and Quoc V Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." In: *arXiv preprint arXiv:1905.11946* (2019) (cit. on pp. 36, 37).

[102] Antti Tarvainen and Harri Valpola. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results." In: *NIPS*. 2017, pp. 1195–1204 (cit. on pp. 29, 30).

[103] Chih-Fong Tsai. "Bag-of-Words Representation in Image Annotation: A Review." In: *International Scholarly Research Notices* 2012 (2012), pp. 1–19 (cit. on p. 13).

[104] Tong Xiao et al. "Learning from massive noisy labeled data for image classification." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 2691–2699. DOI: 10.1109/CVPR.2015.7298885 (cit. on p. 47).

[105] Qizhe Xie et al. "Self-training with Noisy Student improves ImageNet classification." In: *arXiv preprint arXiv:1911.04252* (2019) (cit. on pp. 7, 36, 37).

[106] Qizhe Xie et al. "Unsupervised Data Augmentation." In: *CoRR* abs/1904.12848 (2019). arXiv: 1904.12848. URL: http://arxiv.org/abs/1904.12848 (cit. on pp. 8, 34).

[107] Xiangli Yang et al. "A Survey on Deep Semi-supervised Learning." In: *CoRR* abs/2103.00550 (2021). arXiv: 2103.00550. URL: https://arxiv.org/abs/2103.00550 (cit. on pp. 28, 35).

[108] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks." In: *BMVC*. Ed. by Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith. 2016 (cit. on pp. 20, 56, 58).

[109] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks." In: *CoRR* abs/1311.2901 (2013). arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901 (cit. on p. 4).

[110] Xiaohua Zhai et al. "S^4L: Self-Supervised Semi-Supervised Learning." In: *CoRR* abs/1905.03670 (2019). arXiv: 1905.03670. URL: http://arxiv.org/abs/1905.03670 (cit. on pp. 36, 37, 44).

[111] Liheng Zhang and Guo-Jun Qi. "WCP: Worst-Case Perturbations for Semi-Supervised Deep Learning." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020 (cit. on pp. 33, 34).

[112] Xuezhou Zhang, Xiaojin Zhu, and Stephen J. Wright. "Training Set Debugging Using Trusted Items." In: *CoRR* abs/1801.08019 (2018). arXiv: 1801.08019. URL: http://arxiv.org/abs/1801.08019 (cit. on p. 47).

[113] Zizhao Zhang et al. "SemiContour: A Semi-Supervised Learning Approach for Contour Detection." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016 (cit. on p. 35).

[114] Xi Zhou et al. "Image Classification Using Super-Vector Coding of Local Image Descriptors." In: *ECCV*. 2010 (cit. on p. 12).

[115] Yan Zhou and Sally Goldman. "Democratic co-learning." In: *16th IEEE International Conference on Tools with Artificial Intelligence*. IEEE. 2004, pp. 594–602 (cit. on p. 38).

[116] Zhi-Hua Zhou and Ming Li. "Semi-supervised learning by disagreement." In: *Knowl. Inf. Syst.* 24.3 (2010), pp. 415–439. DOI: 10.1007/s10115-009-0209-z. URL: https://doi.org/10.1007/s10115-009-0209-z (cit. on p. 37).

[117] Zhi-Hua Zhou and Ming Li. "Tri-training: Exploiting unlabeled data using three classifiers." In: *IEEE Transactions on knowledge and Data Engineering* 17.11 (2005), pp. 1529–1541 (cit. on p. 38).